English++

English for Computer Science Students

Complementary Course Book

open book

Jagiellonian Language Center
Jagiellonian University
Cracow 2008

Książka *English++ English for Computer Science Students* powstała w ramach niekomercyjnego projektu o nazwie English++, który został zrealizowany przez Monikę Stawicką wraz grupą studentów III roku informatyki uczących się języka angielskiego w Jagiellońskim Centrum Językowym Uniwersytetu Jagiellońskiego w Krakowie w roku akademickim 2007/2008. Autorka i realizatorzy projektu dziękują Dyrekcji Centrum za umożliwienie jego realizacji, a użytkowników skryptu w wersji papierowej lub elektronicznej zachęcają do korzystania z materiałów tam zawartych w sposób twórczy rozszerzając je, adoptując do własnych potrzeb, korygując lub tworząc nowe.

Autorka projektu English++	Monika Stawicka
Realizatorzy projektu English++	Monika Stawicka
	Aleksandra Bieńkowska
	Paweł Fidelus
	Bartłomiej Filipek
	Krystian Kichewko
	Szymon Kaczorowski
	Michał Kubak
	Ewa Matczyńska
	Tomasz Paczkowski
	Michał Pal
	Krzysztof Roksela
	Aleksandra Sendecka
	Artur Staszczyk
	Krzysztof Szromba
	Piotr Śmigielski

Contents

English++

Acknowledgements

Many people have assisted in the preparation of this first version of the book. But, of course, as the leader of the English++ project I alone feel responsible for any shortcomings. I would like to give special thanks to a group of enthusiastic 3rd year computer science students of the Jagiellonian University without whom this book would have never been prepared. They have worked as experts in the IT field, text selectors, authors of complementary exercises, and finally shaped the book. Above all I am grateful for the support and coordination of the project provided by Artur Staszczyk and Paweł Fidelus and for creativeness and engagement of all the English++ members: Aleksandra Sendecka, Aleksandra Bieńkowska, Ewa Matczyńska, Artur Staszczyk , Paweł Fidelus,Tomasz Paczkowski, Piotr Śmigielski, Bartłomiej Filipek, Krzysztof Szromba, Michał Kubak, Michał Pal, Krzysztof Roksela, Krystian Kichewko and Szymon Kaczorowski. They all have contributed to the accomplishment of the book not only by practicing their English language skills but also by actively using their knowledge as experts in the field. Their work goes much beyond standard requirements of an English university course.

Special thanks also go to: Dr Anna Ochal from the Institute of Computer Science, who revised the mathematical part of the manuscript and to Dr Jerzy Freundlich, a colleague of mine, who painstakingly revised reading and listening texts and the exercises; Małgorzata Świątek, Director of the Jagiellonian Language Center and Professor Marek Skomorowski, Director of the Institute of Computer Science for the support of my initiative; Professor Władysław Miodunka for helping me to maintain belief in the value of the Project, Dr Rafał Maciąg and Jerzy Zając for their help in the recording studio, Dr Monika Coghen for her supportive comments and Maciek Kwiatkowski for giving the book its final shape. Particular thanks go to Jolanta Krzyształowska, Financial Director of the Jagiellonian Language Center and the administrative staff of the Center.

The project members are grateful to Professor Stanisław Migórski, Dr Igor Podolak and Dr Piotr Kalita from the Institute of Computer Science of the Jagiellonian University for giving us permission to reproduce extracts of their work in our book: "An Introduction to the Modelling of Real–World Problem by the Simplest Ordinary Differential Equations" by Stanisław Migórski and "Mechanical Model of Artery Walls" by Piotr Kalita and Robert Schaefer. We are also grateful to the following authors for permission to reproduce extracts of their work in English++ book: Joel Spolsky for "Lord Palmerston on Programming", http://www.joelonsoftware.

com/articles/LordPalmerston.html; Randy Nash for "Cyber Warfare: Reality or Box Office Hit?", http://www.informit.com/articles/article.aspx?p=1016106; Roderick Hames for "History of Computers"), http://www.crews.org/curriculum/ex/compsci/articles/history.htm; Damien Stolarz for "How to Stream Video Over a Network or the Internet", http://www.informit.com/articles/article.aspx?p=331397&seqNum=1; John McCarthy for „What Is Artificial Intelligence? - Basic Questions", http://www-formal.stanford.edu/jmc/whatisai/node1.html; Martin Fowler and Pramod Sadalage for "Evolutionary Database Design", http://www.martinfowler.com/articles/evodb.html; Martin Fowler for "Writing Software Patterns", http://www.martinfowler.com/articles/writingPatterns.html; Tim Jones for "Anatomy of the Linux Kernel", http://www.ibm.com/developerworks/linux/library/l-linux-kernel; Robert Ahlfinger, Brenton Cheeseman, Patrick Doody for „The Pitch Correction Algorithm: an Overview", http://cnx.org/content/m12539/latest/. A special word of thanks go to those authors who additionally supported us with their enthusiastic mails.

The following texts come from open sources: "Computer Facial Animation", http://en.wikipedia.org/wiki/Facial_animation; "IloveYou Worm", http://en.wikipedia.org/wiki/ILOVEYOU; „Software Development Process", http://en.wikipedia.org/wiki/Software_development_process; „Quake - Game Engine" http://en.wikipedia.org/w/index.php?title=Quake&diff=172131494&oldid=172045276#Quake_engine; Open News episodes 25, 29 and 31 are under the licence of Creative Commons - http://opennewsshow.org/; Agile Software Development - http://itc.conversationsnetwork.org/shows/detail175.html. The pictures come from: History of Computers: http://www.flickr.com/photos/dmealiffe/171720479/sizes/o/; http://www.flickr.com/photos/indigoprime/2240131208/sizes/o/; http://www.flickr.com/photos/broughturner/2331185712/sizes/o/. ILoveyouWorm: Photoowned by LuiDuar (cc) http://www.flickr.com/photos/22258204@N03/2482225688/; http://www.flickr.com/photos/vidiot/35382084/sizes/l/; http://www.flickr.com/photos/joffley/135052908/sizes/l/. How to Stream Video Over a Network or the Internet: http://www.flickr.com/photos/inju/2493101180/sizes/l/; http:/www.flickr.com/photos/msjacoby/208642529/sizes/l/; http://www.flickr.com/photos/pbo31/2162247328/sizes/o/. Computer Simulation: http://www.flickr.com/photos/ensight/2125461254/; http://www.flickr.com/photos/oubliette/17538292/sizes/o/; http://www.flickr.com/photos/jurvetson/447302275/sizes/l/. Computer Facial Animation: http://www.flickr.com/photos/ko_an/53060221/sizes/o/; http://www.flickr.com/photos/giopuo/483784528/sizes/o/; http://www.flickr.com/photos/guccibear2005/173969294/sizes/l/; http://www.flickr.com/photos/coreburn/166237292/. Cyber Warfare: Reality or Box Office Hit? http://www.flickr.com/photos/devachan77/338153377/sizes/l/; http://www.flickr.com/photos/krazykory/2437404581/sizes/l/; http://www.flickr.com/photos/juan23/82888194/sizes/l/. Lord Palmerston on Programming: http://www.flickr.com/photos/vancouversun/446503999/sizes/l/; http://www.flickr.com/photos/icco/2246383366/sizes/l/; http://www.

flickr.com/photos/jonnowitts/2399505874/sizes/l/. Quake-GameEngine: http://www.flickr.com/photos/nothingpersonal/251603538/sizes/o/; http://www.flickr.com/photos/psycho_al/66541940/sizes/o/; http://www.flickr.com/photos/prh/412670043/sizes/l/. Mechanical Models of Artery Walls: http://www.flickr.com/photos/patrlynch/450142019/sizes/o/; http://www.flickr.com/photos/patrlynch/450129220/sizes/o/; http://www.flickr.com/photos/andycarvin/2220691059/sizes/o/. The Pitch Correctiom Algorythm. An Overwiev: Photo owned by woodleywonderworks (cc), http://www.flickr.com/photos/73645804@N00/2267564159/; http://www.flickr.com/photos/zen/40081589/sizes/o/; http://www.flickr.com/photos/timcaynes/102981762/sizes/l/. Software Development Process http://www.flickr.com/photos/kubernan/401923870/sizes/l/; http://www.flickr.com/photos/reinhold-behringer/1072000705/sizes/l/; http://www.flickr.com/photos/bootload/146803248/sizes/o/. What is Artificial Intelligence: http://www.flickr.com/photos/sashko/362105716/sizes/o/; http://www.flickr.com/photos/gla/1790915676/sizes/o/; http://www.flickr.com/photos/easysleazycheesy/1423829545/sizes/l/.

Monika Stawicka

July 2008

Introduction

English ++ to bardzo ciekawy projekt, zrealizowany w Jagiellońskim Centrum Językowym przez mgr Monikę Stawicką i studentów informatyki uczących się angielskiego. Ciekawy, bo pokazujący, jak naukę języka angielskiego można zintegrować z rozwojem zawodowym studentów informatyki. Ciekawy także dlatego, bo dowodzący, że bierne uczenie się języka obcego można zamienić w zajęcie kreatywne, w którym obie strony procesu nauczania – lektor i studenci – wiedzą, że robią coś nowego, co przynosi najwięcej korzyści im samym, ale także coś, co może się okazać pomocą dla innych uczących się.

Mam nadzieję, że wszyscy, którzy zetkną się z tym projektem, odczują radość tworzenia tak, jak ja ją odczułem w czasie spotkania z jego realizatorami w czerwcu 2008 roku.

Władysław T. Miodunka

Kraków, 18 lipca 2008

Introduction

English++ is an interesting project realized in the Jagiellonian Language Center by Monika Stawicka and computer science students learning English with her. The project is interesting first of all, because it shows how to integrate learning English with professional development. Moreover, it is interesting because it proves that passive acquisition of a foreign language can be transformed into creative activities, where both parties, a language teacher and students know that they do something really new and beneficial not only for themselves, but perhaps also for others who study English in a professional context.

I really believe that all of you who will come across Project English++ will share their creative enthusiasm as I did during the meeting with the Project team in June 2008.

Władysław T. Miodunka

Cracow, 18 July 2008

Introduction

The aim of the book

Language for Specific Purposes (LSP) is an important issue in studying foreign languages at *lektoraty* within the Polish higher education context. In the English++ project a foreign language is naturally English and a specific purpose is the field of computer science.

LSP
Language
for Specific Purposes

The idea of a complementary English course book for computer science students evolved when I was asked to run an English course for such students. A lack of appropriate and coherent materials for teaching and learning the ESP element was a serious drawback of the course. The reading or listening texts I brought to the classroom were not always at the satisfactionary level as far as the subject matter was concerned.

The main aim of English++ book, therefore, is an attempt to bridge the gap between the students' needs and teachers' competences in the area of English for computer science by offering, among other things, a selection of texts suitable for students at their level of professional development. The unique trait of this book is the fact that the texts have been selected and exercises have been prepared by future experts in the field – a group of 3rd year computer science students working under the supervision of their English teacher. All this to ease studying a foreign language in a subject specific context.

The book's audience

This book has been aimed at two kinds of users. One is a computer science student whose general competence in English is at least at an upper intermediate level (B2 level according to Common European Framework of Reference). He can use the book for self-study or in the classroom with his teacher's assistance. A teacher is the second kind of a user. He can use a given text as a starting point for creating his own activities in the classroom or he can simply follow the suggestions of exercises the book provides. Both groups of users can benefit additionally by becoming more familiar with various aspects of a broad area of computer science.

CEFR
Common European
Framework of Reference

The contents of the book

The book is divided into three chapters. They are the main chapter containing authentic reading material, the chapter with listening material and the chapter dealing with a difficult task of delivering an oral presentation. At the end of the book there are appendixes and the glossary. The second and third chapters and the appendixes are accompanied by audio and video material.

Reading Chapter

The following are parts of the reading chapter with a brief summary of their contents:

- Information on the reading text

This section contains 'technical' information on a reading text, such as IT sub-areas the article covers, the length of a text expressed in a number of words, levels of the English language complexity, computer science or math content, summaries in English and Polish, keywords with their definitions and learning objectives. This part has been designed for both teachers and students to make their preliminary choices for reading easier. The evaluation of the English level difficulty and subject matter complexity has been provided by the students.

ESP
English
for Specific Purposes

- Pre-reading questions

This section has been designed to encourage a reader to think about the topic which will be then discussed in a given text. The pre–reading questions are supposed to provoke an exchange of opinions or a short discussion. Some of them are accompanied by suggested answers provided in the section: Exercises.

IT
information technology

- Text

This section contains the whole text of an original article, or as it happens in the case of several articles, just excerpts. The remaining parts are then located in the e-version of the book. The texts have been selected for their intrinsic interest. They vary in length; therefore they may be suitable for either intensive or extensive reading and for practicing reading skills.

- Exercises

The first part of this section contains suggestions of pre-reading questions this time accompanied by suggestions of the answers, followed by comprehension questions also complemented by the answers. This might be practical when the book is used both by a learner for self - study and by a teacher in the classroom. We believe that providing suggested answers just after the text and not at the end of the book or in another book will make a teacher's life in the classroom a bit easier.

Listening Chapter

The organization of the listening chapter is similar to the organization of the reading chapter. Consequently, the listening chapter contains the following sections:

- Information on the listening text

Instead of a number of words a running time of a particular listening piece has been provided. The level of listening difficulty has been also evaluated by the students.

- Pre-listening questions

The questions should help students focus their attention and predict the content of a listening material. They may also provoke a short exchange of opinions.

- Transcript

It can be particularly useful when the listening material has been evaluated as difficult, which is expressed by medium or high English complexity or when computer science content is high, too.

- Exercises

They have been designed by the students and can be used as a starting point for a teacher to prepare his/her own exercises.

The listening chapter is accompanied by a CD with the recordings selected by the students from English++ team.

Presentation Chapter

The third chapter is devoted to the development of students' presentation skills. It contains the following sections:

- Practical tips

This part contains practical information on how to get ready for an oral presentation and make it effective. Therefore a reader will find there information about a preparatory phase, a dress rehearsal or visual aids.

- Repertoire of presentation phrases

This section contains a selection of ready to use presentation phrases to be implemented into different parts of an oral presentation to make it a coherent entity.

- Slide show

English++ DVD with a slide show: "Successful Presentations. A Few Tips from English++" shows examples of both well done and less successful presentations. It simply presents practical application of selected presentation phrases taken from the previous section of this chapter. The main actors are the students from English++ team, who additionally have decided to show what a presentation should not like.

Appendixes

In the appendixes of the book the material for pronunciation practice is included. Three appendixes A, B and C contain mathematical terminology, mathematical formulas and

the Greek alphabet, respectively. The material can be used for self-study or in the classroom to practice field related language.

Glossary

The glossary contains entries for the keywords which have been selected from the reading and listening texts of English++ book. The definitions have been provided by the students.

How to use English++ book

English++ book is an open book, which means that both teachers and students can use and/or modify the material it contains to adapt it for their own teaching/learning needs. However, any commercial usage of the book is prohibited. The e-version of English ++ open book can be found on the English++ webpage:
www.englishplusplus.jcj.uj.edu.pl

Pilot version

This is a pilot version of the English++ book. This means that over next academic year those teachers who run English courses for computer science or math students at B2 level or above can test it in their classroom to complement general English materials. It would be extremely valuable to have my colleagues' opinions, information on spotted inaccuracies or suggestions for improvements. It would be equally valuable to obtain the feedback from those for whom this book has been created - from computer science students. Feel free to be constructively critical and comment!

But first of all enjoy!

Monika Stawicka
author and leader of the project

Cracow, June 2008

Addresses for correspondence:
monika.stawicka@uj.edu.pl

Monika Stawicka
Jagiellońskie Centrum Językowe
Uniwersytet Jagielloński
ul. Krupnicza 2
31-123 Kraków

English++

English for Computer Science Students

Reading Chapter

History of Computers

Roderick Hames

Number of words	1050
Computer science content	Low
Math content	Low
English language complexity	Low

Learning objectives

- to acquire basic knowledge about computer history

Sub-areas covered

- Computer history

Keywords

- **punched card** - a card on which data can be recorded in the form of punched holes
- **binary code** - code using a string of 8 binary digits to represent characters

Summary

A short article which describes the history of computers and their precursors. It briefly mentions important events from 1600 up to the times when the first computer was built. A nice text written in a simple language. It could be used as a lead-in to interesting discussions about the future of computers or the pace of their evolution.

Krótki tekst opisujący historię komputerów wraz z tym, co można nazwać ich protoplastami. Pokrótce opisane są ważniejsze wydarzenia od 1600 roku aż do powstania pierwszego komputera. Przyjemny tekst, napisany nieskomplikowanym językiem. Może być wstępem do ciekawych dyskusji na przykład o przyszłości komputerów lub tempie ich dalszego rozwoju.

Pre-reading questions

1. Why do so many people not know how the modern computer began?
2. Why do you think the computer has changed more rapidly than anything else?
3. How do you think W.W.II might have been different if the ENIAC, the first all electrical computer, whose first job was to calculate the feasibility of a design for the hydrogen bomb, had not been invented then?

History of Computers

Early Start

Computers have been around for quite a few years. Some of your parents were probably around in 1951 when the first computer was bought by a business firm. Computers have changed so rapidly that many people cannot keep up with the changes. One newspaper tried to describe what the auto industry would look like if it had developed at a similar pace to changes in computer technology:

"Had the automobile developed at a pace equal to that of the computer during the past twenty years, today a Rolls Royce would cost less than $3.00, get 3 million miles to the gallon, deliver enough power to drive (the ship) the Queen Elizabeth II, and six of them would fit on the head of a pin!" These changes have occurred so rapidly that many people do not know how our modern computer got started.

punched card
a card on which data can be recorded in the form of punched holes

The First Computing Machines "Computers"

Since ancient times, people have had ways of dealing with data and numbers. Early people tied knots in rope and carved marks on clay tablets to keep track of livestock and trade. Some people consider the 5000-year-old ABACUS - a frame with beads strung on wires - to be the first true computing aid.

As the trade and tax system grew in complexity, people saw that faster, more reliable and accurate tools were needed for doing math and keeping records.

In the mid-600's, Blaise Pascal and his father, who was a tax officer himself, were working on taxes for the French government in Paris. The two spent hours figuring and refiguring taxes that each citizen owed. Young Blaise decided in 1642 to build an adding and subtraction machine that could assist in such a tedious and time-consuming process. The machine Blaise made had a set of eight gears that worked together in much the same way as an odometer keeps track of a car's mileage. His machine encountered many problems. For one thing, it was always breaking down. Second, the machine was

slow and extremely costly. And third, people were afraid to use the machine, thinking it might replace their jobs. Pascal later became famous for math and philosophy, but he is still remembered for his role in computer technology. In his honor, there is a computer language named Pascal.

binary code
code using a string of 8 binary digits to represent characters

The next big step for computers arrived in the 1830s, when Charles Babbage decided to build a machine to help him complete and print mathematical tables. Babbage was a mathematician who taught at Cambridge University in England. He began planning his calculating machine, calling it the Analytical Engine. The idea for this machine was amazingly like the computer we know today. It was to read a program from punched cards, figure and store the answers to different problems, and print the answer on paper. Babbage died before he could complete the machine. However, because of his remarkable ideas and work, Babbage is known as the Father of Computers.

The next huge step for computers came when Herman Hollerith entered a contest organised by the U.S. Census Bureau. The contest was to see who could build a machine that would count and record information the fastest. Hollerith, a young man working for the Bureau, built a machine called the Tabulating Machine that read and sorted data from punched cards. The holes punched in the cards matched each person's answers to questions. For example, married, single, and divorced were answers on the cards. The Tabulator read the punched cards as they passed over tiny brushes. Each time a brush found a hole, it completed an electrical circuit. This caused special counting dials to increase the data for that answer.

Thanks to Hollerith's machine, instead of taking seven and a half years to count the census information it only took three years, even with 13 million more people since the last census. Happy with his success, Hollerith formed the Tabulating Machine Company in 1896. The company was later sold in 1911 and in 1912 his company became the International Business Machines Corporation, better known today as IBM.

The First Electric Powered Computer

ENIAC
Electronic Numerical Integrator and Computer

What is considered to be the first computer was made in 1944 by Harvard Professor Howard Aiken. The Mark I computer was very much like the design of Charles Babbage's Analytical Engine, having mainly mechanical parts but with some electronic parts. His machine was designed to be programmed to do many computing jobs. This all-purpose machine is what we now know as the PC or personal computer. The Mark I was the first computer financed by IBM and was about 50 feet long and 8 feet tall. It used mechanical switches to open and close its electric circuits. It contained over 500 miles of wire and 750,000 parts.

The First All Electronic Computer

The first all electronic computer was the ENIAC (Electronic Numerical Integrator and Computer). ENIAC was a general purpose digital computer built in 1946 by J. Presper Eckert and John Mauchly. The ENIAC contained over 18,000 vacuum tubes (used instead of the mechanical switches of the Mark I) and was 1000 times faster than the Mark I. In twenty seconds, ENIAC could do a math problem that would have taken 40 hours for one person to finish. The ENIAC was built at the time of World War II and as its first job had to calculate the feasibility of a design for the hydrogen bomb. The ENIAC was 100 feet long and 10 feet tall.

More Modern Computers

A more modern type of computer began with John von Neumann's development of software written in binary code. It was von Neumann who began the practice of storing data and instructions in binary code and initiated the use of memory to store data, as well as programs. A computer called the EDVAC (Electronic Discrete Variable Computer) was built using binary code in 1950. Before the EDVAC, computers like the ENIAC could do only one task; then they had to be rewired to perform a different task or program. The EDVAC's concept of storing different programs on punched cards instead of rewiring computers led to the computers that we know today.

While the modern computer is far better and faster than the EDVAC of its time, computers of today would not have been possible without the knowledge and work of many great inventors and pioneers.

EDVAC
Electronic
Discrete Variable Computer

Exercises

Comprehension questions

1. **Why was Pascal honored with a computer language named for him?**

- This programming language was named as a tribute to Blaise Pascal, because of his contribution to computer development. He was the first to build a precursor of the modern computer-an adding an subtraction machine that could assist in tedious and time-consuming computational process.

2. **Who was the first to invent a machine whose operating principle is very similar to present-day computers? Describe these similarities.**

- Charles Babbage, whose idea was remarkably similar to the way modern computers work: read program from punched cards (input), figure and store the answers to different problems, and print the answer on paper (output)

3. **In which process was Hollerith's machine involved and what was its role?**

- Hollerith's machine helped with the counting of census information. It took three years, instead of seven and half, even with 13 million more people since the previous census. "The machine read and sorted data from punched cards. The holes punched in the cards matched each person's answers to questions. For example, married, single, and divorced were answers on the cards. The Tabulator read the punched cards as they passed over tiny brushes. Each time a brush found a hole, it completed an electrical circuit. This caused special counting dials to increase the data for that answer.

4. **Describe all the technical parameters of the first electric powered computer.**

- 50 feet long, 8 feet tall, electrical circuits are opened and closed by mechanical switches, contained 500 miles of wire and 750 000 parts

5. **What were the differences between the Mark I and the ENIAC?**

- Mark I - electric powered, 50 feet long and 8 tall, used mechanical switches to open/close electrical circuits
- ENIAC - all electronic computer, used 18,000 vacuum tubes instead of mechanical switches, 1000 times faster than Mark I, 100 feet long and 10 feet tall

6. **What is the main advantage of using binary code in storing data and instructions?**

- First computers like the ENIAC could do only one task, then they had to be rewired to perform a different task or program. The binary code concept of storing different programs on punched cards instead of rewiring computers led to computers that we know today.

Possible topics for discussion

1. Future of computers.

Possible difficulties

This is a fairly easy to text to encourage a reader to study a bit more advanced articles. There should not be any problems with understanding this article.

ILOVEYOU Worm

Wikipedia

Number of words	1150
Computer science content	Medium
Math content	Low
English language complexity	Medium

Learning objectives

- to understand why the ILOVEYOU worm was so successful
- to understand the architecture of the ILOVEYOU worm as a basic script virus
- to recognize how big an effect a single virus can have on global IT

Sub-areas covered

- Computer virus
- Computer worm
- Popular worms/viruses

Keywords

- **malware** - software designed to infiltrate or damage a computer system without the owner's informed consent
- **VBScript** - Visual Basic Scripting Edition - an Active Scripting; technology used in Windows to implement component-based scripting support; a language developed by Microsoft
- **social engineering** - practice of obtaining confidential information by manipulating users
- **Barok trojan** - this trojan horse gathers information such as user name, IP address and passwords, and attempts to send the information to the creator of the virus

Summary

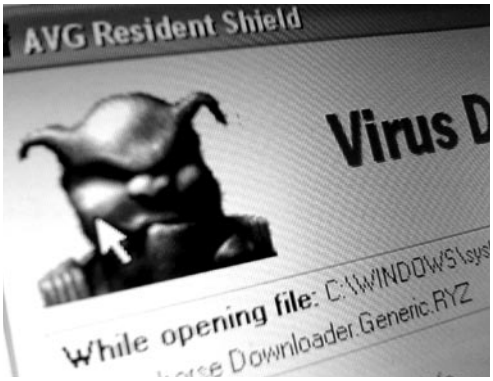
The article provides information about the creation and history of a well-known computer worm called 'ILOVEYOU'. The first section contains its basic description and explains the features of the worm that made it effective. The following sections describe how the worm spread and the effect it had worldwide. The article gives information about the author of the worm and the man who wrote the software that repaired the damage it caused. This is followed by a section on how the 'ILOVEYOU' worm affects computers. The text ends with brief information about the legal measures against the author of the worm.

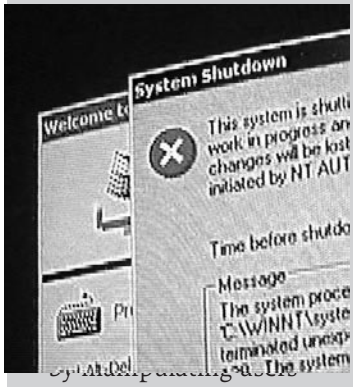
Tekst opisuje jeden z bardziej znanych robaków komputerowych - 'ILOVEYOU'. Pierwsza część zawiera podstawowe informacje o programie i tym, co uczyniło go efektywnym. Następna opowiada o rozprzestrzenianiu się wirusa. Kolejne przedstawiają globalne działanie robaka, informują o jego autorze oraz o twórcy programu naprawiającego szkody. Kolejna część tekstu dostarcza wiadomości o działaniach 'ILOVEYOU' na komputerze - co robi i jakie wywołuje szkody. Tekst kończy się informacją na temat konsekwencji prawnych wyciągniętych wobec autora 'ILOVEYOU'.

Pre-reading questions

1. What attacks have you heard about?
2. Have you ever had a virus or worm in your mail?
3. Name a few of the most famous viruses.

ILOVEYOU Worm





The ILOVEYOU worm, also known as VBS/Loveletter and Love Bug worm, is a computer worm written in VBScript.

Description

The worm, first discovered in Hong Kong, arrived in e-mail boxes on May 4, 2000, with the simple subject of "ILOVEYOU" and an attachment "LOVE-LETTER-FOR-YOU. TXT.vbs".

Two aspects of the worm made it effective:

- It relied on social engineering to entice users to open the e-mail and ensure its continued propagation.
- It employed a mechanism — VBScripts — that, while not entirely novel, had not been exploited to such a degree previously to direct attention to their potential, reducing the layers of protection that would have to be navigated for success.

Spread

Its massive spread moved westward as workers arrived at their offices and encountered messages generated by people from the East. Because the virus used mailing lists as its source of targets, the messages often appeared to come from an acquaintance and so might be considered "safe", providing further incentive to open them. All it took was a few users at each site to access the VBS attachment to generate the thousands and thousands of e-mails that would cripple e-mail systems under their weight, not to mention overwrite thousands of files on workstations and accessible servers.

Effects

It began in the Philippines on May 4, 2000, and spread across the world in one day (travelling from Hong-Kong to Europe to the United States), infecting 10 percent of all computers connected to the Internet and causing about $5.5 billion in damage. Most

VBScript
Visual Basic Scripting Edition - an Active Scripting (technology used in Windows to implement component-based scripting support) language developed by Microsoft

of the "damage" was the labor of getting rid of the virus. The Pentagon, CIA, and the British Parliament had to shut down their e-mail systems to get rid of the worm, as did most large corporations.

This particular malware caused widespread outrage, making it the most damaging worm ever. The worm overwrote important files, as well as music, multimedia and more, with a copy of itself. It also sent the worm to everyone on a user's contact list. This particular worm only affected computers running the Microsoft Windows operating system. While any computer accessing e-mail could receive an "ILOVEYOU" e-mail, only Microsoft Windows systems would be infected.

malware
software designed to infiltrate or damage a computer system without the owner's informed consent

Authorship

The ILOVEYOU worm is believed to have been written by Michael Buen. The Barok trojan horse used by the worm is believed to have been written by Onel de Guzman, a Filipino student of AMA Computer University in Makati, Philippines.

An international manhunt for the perpetrator finally led to a young programming student. On May 11 (one week after the virus spread), he held a news conference and said that he did not mean to cause so much harm. He was unable to graduate because the university rejected his thesis on the basis of its illegality. Helped by a group of friends called the Grammersoft Group, he distributed his virus the day before the school held their graduation ceremony.

Detection

Narinnat Suksawat, a 25-year-old Thai software engineer, was the first person to write software that repaired the damage caused by the worm, releasing it to the public on May 5, 2000, 24 hours after the worm had spread. "Rational Killer", the program he created, removed virus files and restored the previously removed system files so they again functioned normally. Two months later, Narinnat was offered a senior consultant job at Sun Microsystems and worked there for two years. He resigned to start his own business. Today, Narinnat owns a software company named Moscii Systems, a system management software company in Thailand.

Architecture of the worm

The worm is written using Microsoft Visual Basic Scripting (VBS), and requires that the end-user run the script in order to deliver its payload. It will add a set of registry keys to the Windows registry that will allow the malware to start up at every boot.

The worm will then search all drives which are connected to the infected computer and replace files with the extensions *.JPG, *.JPEG, *.VBS, *.VBE, *.JS, *.JSE, *.CSS, *.WSH, *.SCT, *.DOC *.HTA with copies of itself, while appending to the file name a .VBS. extension. The malware will also locate *.MP3 and *.MP2 files, and when found, makes the files hidden, copies itself with the same file name and appends a .VBS.

The worm propagates by sending out copies of itself to all entries in the Microsoft Outlook address book. It also has an additional component, in which it will download and execute an infected program called variously "WIN-BUGSFIX.EXE" or "Microsoftv25.exe". This is a password-stealing program which will e-mail cached passwords.

Barok trojan
this trojan horse gathers information such as user name, IP address and passwords, and attempts to send the information to the creator of the virus

Variants

1. Attachment: LOVE-LETTER-FOR-YOU.TXT.vbs
   Subject Line: ILOVEYOU
   Message Body: kindly check the attached LOVELETTER coming from me.
2. Attachment: Very Funny.vbs
   Subject Line: fwd: Joke
   Message Body: empty
3. Attachment: mothersday.vbs
   Subject Line: Mothers Day Order Confirmation
   Message Body: We have proceeded to charge your credit card for the amount of $326.92 for the mothers day diamond special. We have attached a detailed invoice to this email. Please print out the attachment and keep it in a safe place. Thanks Again and Have a Happy Mothers Day! mothersday@subdimension.com
4. Attachment: virus_warning.jpg.vbs
   Subject Line: Dangerous Virus Warning
   Message Body: There is a dangerous virus circulating. Please click attached picture to view it and learn to avoid it.
5. Attachment: protect.vbs
   Subject Line: Virus ALERT!!!
   Message Body: a long message regarding VBS.LoveLetter.A
6. Attachment: Important.TXT.vbs
   Subject Line: Important! Read carefully!!
   Message Body: Check the attached IMPORTANT coming from me!
7. Attachment: Virus-Protection-Instructions.vbs
   Subject Line: How to protect yourself from the IL0VEYOU bug!
   Message Body: Here's the easy way to fix the love virus.
8. Attachment: KillEmAll.TXT.VBS
   Subject Line: I Cant Believe This!!!
   Message Body: I Cant Believe I have Just received This Hate Email .. Take A Look!

9. Attachment: ArabAir.TXT.vbs
   Subject Line: Thank You For Flying With Arab Airlines
   Message Body: Please check if the bill is correct, by opening the attached file
10. Attachment: IMPORTANT.TXT.vbs
   Subject Line: Variant Test
   Message Body: This is a variant to the vbs virus.
11. Attachment: Vir-Killer.vbs
   Subject Line: Yeah, Yeah another time to DEATH...
   Message Body: This is the Killer for VBS.LOVE-LETTER.WORM.
12. Attachment: LOOK.vbs
   Subject Line: LOOK!
   Message Body: hehe...check this out.
13. Attachment: BEWERBUNG.TXT.vbs
   Subject Line: Bewerbung Kreolina
   Message Body: Sehr geehrte Damen und Herren!
14. Subject Line: Is this you in this picture?
   Message Body: Is this you in this picture?

Legislative aftermath

As there were no laws against virus-writing at the time, on August 21, 2000, the prosecutors dropped all charges against Onel A. de Guzman in a resolution signed by Jovencito Zuno. The original charges brought up against de Guzman dealt with the illegal use of passwords for credit card and bank transactions. The Philippines E-Commerce Law (Republic Act No. 8792), passed on June 14, 2000, laid out penalties for cybercrime. Under the law, those who spread computer viruses or otherwise engage in cybercrime (including copyright infringement and software cracking) can be fined a minimum of 100,000 pesos (about USD$2,000), and a maximum commensurate with the damage caused, and imprisoned for six months to three years.

Exercises

Pre-reading exercises

1. What attacks have you heard about?
2. Have you ever had a virus or worm in mail?
3. Name a few of the most famous viruses.

- Melissa
- Love Bug
- Code Red
- Bugbear
- Blaster

Comprehension questions

1. What kind of attachment was in the ILOVEYOU worm?

- It was a visual basic script.

2. In what language was the ILOVEYOU worm written?

- VBScript programming language

3. When was the ILOVEYOU worm detected?

- 4th May, 2000

4. Who created the Barok trojan?

- Onel de Guzman, a Filipino student

5. What action did the Pentagon take in order to protect itself from the 'I Love You' virus?

- Pentagon had to shut down own e-mail system.

Further exercises

1. Match headings to paragraphs.
2. There are several different variants of emails with this virus in the article. Write your own variant of email that will encourage people to open the attachment.

Possible topics for discussion

1. Why was this virus so dangerous and harmful?

- The worm overwrote important files, as well as music, multimedia and more, with a copy of itself. It also sent the worm to everyone on a user's contact list.

2. Why did it attack only Windows Operating Systems?

- Bad security policy, holes in the security mechanism system and a lot of users oblivious of danger - these factors have added to virus success.

3. What legal consequences should be faced by the authors of computer viruses?

- high fines
- ban on access to computers

4. What action each computer user can take to protect their computers against computer worms?

- install anti-virus software
- be careful what e-mail attachment you open, what websites you visit
- check source of software which you install
- check data mediums, before you open them, – especially pen-drives

Possible difficulties

To help you understand certain parts of the text, make sure you know the meaning of the keywords before you start reading.

Anatomy of the Linux Kernel

Tim Jones

Number of words	2730
Computer science content	High
Math content	Low
English language complexity	Low

Learning objectives

- to acquire basic vocabulary related to operating systems
- to understand the basics of Linux kernel architecture

Sub-areas covered

- Linux kernel and its subsystems

Keywords

- **kernel** - the central component of most computer operating systems (OS). Its functions include managing the system's resources (the communication between hardware and software components)
- **Linux kernel** - Unix-like operating system kernel
- **VFS(Virtual file system)** - an abstraction layer on top of a more concrete file system
- **GNU** - a computer operating system composed entirely of free software, initiated in 1984 by Richard Stallman
- **GPL** - a widely used free software license, originally written by Richard Stallman for the GNU project
- **Minix** - free/open source, Unix-like operating system (OS) based on a microkernel architecture
- **Unix** - a computer operating system originally developed in 1969 by a group of AT&T employees at Bell Labs including Ken Thompson, Dennis Ritchie and Douglas Ilroy
- **operating system** - the software that manages the sharing of the resources of a computer and provides programmers with an interface used to access those resources
- **buffer** - a region of memory used to temporarily hold data while it is being moved from one place to another
- **buffer cache** - a collection of data duplicating original values stored elsewhere or computed earlier, where the original data is expensive to fetch (owing to longer access time) or to compute, compared to the cost of reading the cache

Summary

As the title suggests, this article is about the linux kernel. It starts with a historical introduction, which includes information on unix and minix, the predecessors of linux. The next section is about the linux kernel in general and how it is constructed. The third section is divided into sub-sections which describe the subsystems in the linux kernel. Subsequent sections deal with memory management, process management, drivers layer or network stack. The article is not very complex, being only an introduction to the linux kernel. It concludes with a list of links to longer articles about the linux kernel and its subsystems.

Jak wskazuje tytuł, artykuł ten jest o jądrze linuxa. Na początku mały wstęp historyczny. W tej części zawarte są informacje o unixie i minixie, poprzednikach linuxa. Następna część traktuje o jądrze linuxa jako całości, czyli jak ogólnie jest ono zbudowane. Trzecia część jest złożona z mniejszych podczęści. Każda opisuje jeden z podsystemów jądra. Kolejne części mówią o: zarządzaniu pamięcią, zarządzaniu procesami, warstwą sterowników, systemie plików, interfejsie wywołań systemowych oraz stosie sieciowym. Artykuł nie jest zbyt rozbudowany, jest on tylko wstępem do jądra linuxa. Na końcu tekstu znajduje się lista odnośników do innych artykułów bardziej szczegółowo omawiających różne części jądra.

Pre-reading exercises

1. What are the most popular operating systems?
2. What are the advantages of Linux?
3. What are the disadvantages of Linux?

Anatomy of the Linux Kernel

The Linux® kernel is the core of a large and complex operating system, and while it is huge, it is well organized in terms of subsystems and layers. In this article, you can explore the general structure of the Linux kernel and get to know its major subsystems and core interfaces. Where possible, you get links to other IBM articles to help you dig deeper.

Given that the goal of this article is to introduce you to the Linux kernel and explore its architecture and major components, let's start with a short tour of Linux kernel history, then look at the Linux kernel architecture from 30,000 feet, and, finally, examine its major subsystems. The Linux kernel is over six million lines of code, so this introduction is not exhaustive. Use the pointers to more content to dig in further.

A short tour of Linux history

While Linux is arguably the most popular open source operating system, its history is actually quite short considering the timeline of operating systems. In the early days of computing, programmers developed on the bare hardware in the hardware's language. The lack of an operating system meant that only one application (and one user) could use the large and expensive device at a time. Early operating systems were developed in the 1950s to provide a simpler development experience. Examples include the General Motors Operating System (GMOS) developed for the IBM 701 and the FORTRAN Monitor System (FMS) developed by North American Aviation for the IBM 709.

In the 1960s, the Massachusetts Institute of Technology (MIT) and a host of companies developed an experimental operating system called Multics (or Multiplexed Information and Computing Service) for the GE-645. One of the developers of this operating system, AT&T, dropped out of Multics and developed their own operating system in 1970 called Unics. Along with this operating system was the C language, for which C was developed and then rewritten to make operating system development portable.

Twenty years later, Andrew Tanenbaum created a microkernel version of UNIX®, called MINIX (for minimal UNIX), that ran on small personal computers. This open source operating system inspired Linus Torvalds' initial development of Linux in the early 1990s

Linux quickly evolved from a single-person project to a world-wide development project involving thousands of developers. One of the most important decisions for Linux was its adoption of the GNU General Public License (GPL). Under the GPL, the Linux kernel was protected from commercial exploitation, and it also benefited from

operating system
the software that manages the sharing of the resources of a computer and provides programmers with an interface used to access those resources

the user-space development of the GNU project (of Richard Stallman, whose source dwarfs that of the Linux kernel). This allowed useful applications such as the GNU Compiler Collection (GCC) and various shell support.

Introduction to the Linux kernel

Now on to a high-altitude look at the GNU/Linux operating system architecture. You can think about an operating system from two levels.

buffer
a region of memory used to temporarily hold data while it is being moved from one place to another

At the top is the user, or application, space. This is where the user applications are executed. Below the user space is the kernel space. Here, the Linux kernel exists.

There is also the GNU C Library (glibc). This provides the system call interface that connects to the kernel and provides the mechanism to transition between the user-space application and the kernel. This is important because the kernel and user application occupy different protected address spaces. And while each user-space process occupies its own virtual address space, the kernel occupies a single address space. For more information, see the links in the resources section.

The Linux kernel can be further divided into three gross levels. At the top is the system call interface, which implements the basic functions such as read and write. Below the system call interface is the kernel code, which can be more accurately defined as the architecture-independent kernel code. This code is common to all of the processor architectures supported by Linux. Below this is the architecture-dependent code, which forms what is more commonly called a BSP (Board Support Package). This code serves as the processor and platform-specific code for the given architecture.

Properties of the Linux kernel

VFS(Virtual File System)
an abstraction layer on top of a more concrete file system

When discussing the architecture of a large and complex system, you can view the system from many perspectives. One goal of an architectural decomposition is to provide a way to understand the source better and that's what we'll do here.

The Linux kernel implements a number of important architectural attributes. At a high level, and at lower levels, the kernel is layered into a number of distinct subsystems. Linux can also be considered monolithic because it lumps all of the basic services into the kernel. This differs from a microkernel architecture, where the kernel provides basic services such as communication, I/O, and memory and process management, and more specific services are plugged in to the microkernel layer. Each has its own advantages, but I'll steer clear of that debate.

Over time, the Linux kernel has become efficient in terms of both memory and CPU usage, as well as extremely stable. But the most interesting aspect of Linux, given its

size and complexity, is its portability. Linux can be compiled to run on a huge number of processors and platforms with different architectural constraints and needs. One example is the ability of Linux to run on a process with a memory management unit (MMU), as well as those that provide no MMU. The uClinux port of the Linux kernel provides for non-MMU support. See the resources section for more details.

Major subsystems of the Linux kernel

Now let's look at some of the major components of the Linux kernel using the breakdown.

System call interface

The SCI is a thin layer that provides the means to perform function calls from user space into the kernel. As discussed previously, this interface can be architecture dependent, even within the same processor family. The SCI is actually an interesting function-call multiplexing and demultiplexing service. You can find the SCI implementation in ./linux/kernel, as well as architecture-dependent portions in ./linux/arch. More details for this component are available in the resources section.

Linux kernel
Unix-like
operating system kernel

Process management

Process management is focused on the execution of processes. In the kernel, these are called threads and represent an individual virtualization of the processor (thread code, data, stack, and CPU registers). In user space, the term process is typically used, though the Linux implementation does not separate the two concepts (processes and threads). The kernel provides an application program interface (API) through the SCI to create a new process (fork, exec, or Portable Operating System Interface [POSIX] functions), stop a process (kill, exit), and communicate and synchronize between them (signal, or POSIX mechanisms).

kernel
the central component
of most computer
operating systems (OS).
Its functions include
managing the system's
resources
(the communication
between hardware and
software components

Also in process management there is a need to share the CPU between the active threads. The kernel implements a novel scheduling algorithm that operates in constant time, regardless of the number of threads vying for the CPU. This is called the O(1) scheduler, denoting that the same amount of time is taken to schedule one thread as it is to schedule many. The O(1) scheduler also supports multiple processors (called Symmetric MultiProcessing, or SMP). You can find the process management sources in ./linux/kernel and architecture-dependent sources in ./linux/arch). You can learn more about this algorithm in the resources section.

Memory management

Another important resource that's managed by the kernel is memory. For efficiency, given the way that the hardware manages virtual memory, memory is managed in

Minix
free/open source, Unix-like operating system (OS) based on a microkernel architecture

what are called pages (4KB in size for most architectures). Linux includes the means to manage the available memory, as well as the hardware mechanisms for physical and virtual mappings.

But memory management is much more than managing 4KB buffers. Linux provides abstractions over 4KB buffers, such as the slab allocator. This memory management scheme uses 4KB buffers as its base, but then allocates structures from within, keeping track of which pages are full, partially used, and empty. This allows the scheme to dynamically grow and shrink based on the needs of the greater system.

Supporting multiple users of memory, there are times when the available memory can be exhausted. For this reason, pages can be moved out of memory and onto the disk. This process is called swapping because the pages are swapped from memory onto the hard disk. You can find the memory management sources in ./linux/mm.

Virtual file system

The virtual file system (VFS) is an interesting aspect of the Linux kernel because it provides a common interface abstraction for file systems. The VFS provides a switching layer between the SCI and the file systems supported by the kernel.

At the top of the VFS is a common API abstraction of functions such as open, close, read, and write. At the bottom of the VFS are the file system abstractions that define how the upper-layer functions are implemented. These are plug-ins for the given file system (of which over 50 exist). You can find the file system sources in ./linux/fs.

GPL
a widely used free software license, originally written by Richard Stallman for the GNU project

Below the file system layer is the buffer cache, which provides a common set of functions to the file system layer (independent of any particular file system). This caching layer optimizes access to the physical devices by keeping data around for a short time (or speculatively read ahead so that the data is available when needed). Below the buffer cache are the device drivers, which implement the interface for the particular physical device.

Network stack

The network stack, by design, follows a layered architecture modeled after the protocols themselves. Recall that the Internet Protocol (IP) is the core network layer protocol that sits below the transport protocol (most commonly the Transmission Control Protocol, or TCP). Above TCP is the sockets layer, which is invoked through the SCI.

The sockets layer is the standard API to the networking subsystem and provides a user interface to a variety of networking protocols. From raw frame access to IP protocol data units (PDUs) and up to TCP and the User Datagram Protocol (UDP), the sockets

layer provides a standardized way to manage connections and move data between endpoints. You can find the networking sources in the kernel at ./linux/net.

Device drivers

The vast majority of the source code in the Linux kernel exists in device drivers that make a particular hardware device usable. The Linux source tree provides a drivers subdirectory that is further divided by the various devices that are supported, such as Bluetooth, I2C, serial, and so on. You can find the device driver sources in ./linux/drivers.

Architecture-dependent code

While much of Linux is independent of the architecture on which it runs, there are elements that must consider the architecture for normal operation and for efficiency. The /linux/arch subdirectory defines the architecture-dependent portion of the kernel source contained in a number of subdirectories that are specific to the architecture (collectively forming the BSP). For a typical desktop, the i386 directory is used. Each architecture subdirectory contains a number of other subdirectories that focus on a particular aspect of the kernel, such as boot, kernel, memory management, and others. You can find the architecture-dependent code in ./linux/arch.

GNU
a computer operating system composed entirely of free software, initiated in 1984 by Richard Stallman

Interesting features of the Linux kernel

If the portability and efficiency of the Linux kernel weren't enough, it provides some other features that could not be classified in the previous decomposition.

Linux, being a production operating system and open source, is a great test bed for new protocols and advancements of those protocols. Linux supports a large number of networking protocols, including the typical TCP/IP, and also extension for high-speed networking (greater than 1 Gigabit Ethernet [GbE] and 10 GbE). Linux also supports protocols such as the Stream Control Transmission Protocol (SCTP), which provides many advanced features above TCP (as a replacement transport level protocol).

Linux is also a dynamic kernel, supporting the addition and removal of software components on the fly. These are called dynamically loadable kernel modules, and they can be inserted at boot when they're needed (when a particular device is found requiring the module) or at any time by the user.

A recent advancement of Linux is its use as an operating system for other operating systems (called a hypervisor). Recently, a modification to the kernel was made called the Kernel-based Virtual Machine (KVM). This modification enabled a new interface to user space that allows other operating systems to run above the KVM-enabled kernel. In addition to running another instance of Linux, Microsoft® Windows® can

also be virtualized. The only constraint is that the underlying processor must support the new virtualization instructions. See the resource section for more information.

Going further

This article just scratched the surface of the Linux kernel architecture and its features and capabilities. You can check out the Documentation directory that is provided in every Linux distribution for detailed information about the contents of the kernel.

Resources

buffer cache
a collection of data duplicating original values stored elsewhere or computed earlier, where the original data is expensive to fetch (owing to longer access time) or to compute, compared to the cost of reading the cache

- The GNU site (http://www.gnu.org/licenses) describes the GNU GPL that covers the Linux kernel and most useful applications provided with it. Also described is a less restrictive form of the GPL called the Lesser GPL (LGPL).
- UNIX (http://en.wikipedia.org/wiki/Unics), MINIX (http://en.wikipedia.org/wiki/Minix) and Linux (http://en.wikipedia.org/wiki/Linux) are covered in Wikipedia, along with a detailed family tree of the operating systems.
- The GNU C Library (http://www.gnu.org/software/libc/), or glibc, is the implementation of the standard C library. It's used in the GNU/Linux operating system, as well as the GNU/Hurd (http://directory.fsf.org/hurd.html) microkernel operating system.
- uClinux (http://www.uclinux.org/) is a port of the Linux kernel that can execute on systems that lack an MMU. This allows the Linux kernel to run on very small embedded platforms, such as the Motorola DragonBall processor used in the PalmPilot Personal Digital Assistants (PDAs).
- "Kernel command using Linux system calls" (http://www.ibm.com/developerworks/linux/library/l-system-calls/) (developerWorks, March 2007) covers the SCI, which is an important layer in the Linux kernel, with user-space support from glibc that enables function calls between user space and the kernel.
- "Inside the Linux scheduler" (http://www.ibm.com/developerworks/linux/library/l-scheduler/) (developerWorks, June 2006) explores the new O(1) scheduler introduced in Linux 2.6 that is efficient, scales with a large number of processes (threads), and takes advantage of SMP systems.
- "Access the Linux kernel using the /proc filesystem" (http://www.ibm.com/developerworks/linux/library/l-proc.html) (developerWorks, March 2006) looks at the /proc file system, which is a virtual file system that provides a novel way for user-space applications to communicate with the kernel. This article demonstrates /proc, as well as loadable kernel modules.
- "Server clinic: Put virtual filesystems to work" (http://www.ibm.com/developerworks/linux/library/l-sc12.html) (developerWorks, April 2003) delves into the VFS layer that allows Linux to support a variety of different file systems through a common interface. This same interface is also used for other types of devices, such as sockets.

- "Inside the Linux boot process" (http://www.ibm.com/developerworks/linux/library/l-linuxboot/index.html) (developerWorks, May 2006) examines the Linux boot process, which takes care of bringing up a Linux system and is the same basic process whether you're booting from a hard disk, floppy, USB memory stick, or over the network.
- "Linux initial RAM disk (initrd) overview" (http://www.ibm.com/developerworks/linux/library/l-initrd.html) (developerWorks, July 2006) inspects the initial RAM disk, which isolates the boot process from the physical medium from which it's booting.
- "Better networking with SCTP" (http://www.ibm.com/developerworks/linux/library/l-sctp/) (developerWorks, February 2006) covers one of the most interesting networking protocols, Stream Control Transmission Protocol, which operates like TCP but adds a number of useful features such as messaging, multi-homing, and multi-streaming. Linux, like BSD, is a great operating system if you're interested in networking protocols.
- "Anatomy of the Linux slab allocator" (http://www.ibm.com/developerworks/linux/library/l-linux-slab-allocator/) (developerWorks, May 2007) covers one of the most interesting aspects of memory management in Linux, the slab allocator. This mechanism originated in SunOS, but it's found a friendly home inside the Linux kernel.
- "Virtual Linux" (http://www.ibm.com/developerworks/linux/library/l-linuxvirt/) (developerWorks, December 2006) shows how Linux can take advantage of processors with virtualization capabilities.
- "Linux and symmetric multiprocessing" (http://www.ibm.com/developerworks/library/l-linux-smp/) (developerWorks, March 2007) discusses how Linux can also take advantage of processors that offer chip-level multiprocessing.
- "Discover the Linux Kernel Virtual Machine" (http://www.ibm.com/developerworks/linux/library/l-linux-kvm/) (developerWorks, April 2007) covers the recent introduction of virtualization into the kernel, which turns the Linux kernel into a hypervisor for other virtualized operating systems.
- Check out Tim's book GNU/Linux Application Programming (http://www.charlesriver.com/Books/BookDetail.aspx?productID=91525) for more information on programming Linux in user space.
- In the developerWorks Linux zone (http://www.ibm.com/developerworks/linux/), find more resources for Linux developers, including Linux tutorials (http://www.ibm.com/developerworks/views/linux/libraryview.jsp?type_by=Tutorials), as well as our readers' favorite Linux articles and tutorials (http://www.ibm.com/developerworks/linux/library/l-top-10.html) over the last month.
- Stay current with developerWorks technical events and Webcasts (http://www.ibm.com/developerworks/offers/techbriefings/?S_TACT=105AGX03&S_CMP=art).

Unix
a computer operating system originally developed in 1969 by a group of AT&T employees at Bell Labs including Ken Thompson, Dennis Ritchie and Douglas Ilroy

Exercises

Pre-reading exercises

1. What are the most popular operating systems?
- Windows, Linux, Mac OS
2. What are the advantages of Linux?
- GPL license (free to use)
- stability
- security
3. What are the disadvantages of Linux?
- difficult in use
- some software isn't available in a Linux version

Comprehension questions

1. **Name a few of the earliest operating systems.**
- GMOS, FMS.
2. **Who created MINIX?**
- Andrew Tanenbaum
3. **How many lines of code does the Linux kernel have now?**
- over 6 million
4. **What are the advantages of Linux adoption of the GPL?**
- protection from commercial exploitation
- benefits from the user-space development
- various shell support
5. **Name the three gross levels of the Linux kernel.**
- system call interface
- kernel code
- architecture-dependant code (BSP)
6. **What are the major components of the Linux kernel?**
- System Call Interface
- Process Management
- Memory Management
- Virtual File System
- Network Stack
- Arch
- Device Drivers
7. **Describe in a few words what process management does?**
- Process management is focused on the execution of processes. The kernel provides an application program interface through the SCI to create a new process, stop a process or communicate and synchronize between them. There is also a need to share the CPU between the active threads in the process management.

8. What could you do if you have Linux and you want to run a Windows program?

- find its Linux equivalent
- virtualize Microsoft Windows
- install Windows as a second system

Possible topics for discussion

1. Which operating system, Windows or Linux, is better and why? What is your opinion?
2. Which operating system architecture is better: one with a number of distinct subsystems or one with a single microkernel?
3. Is it better to build small kernels which are easy to understand or large ones with all the necessary functions?

Possible difficulties

The article can be hard to understand because of a big amount of words and phrases from the area of IT.

How to Stream Video over a Network or the Internet

Damien Stolarz

Number of words	1880
Computer science content	High
Math content	Low
English language complexity	Medium

Learning objectives

- to understand different kinds of transfer: radio, television, telephone, internet
- to recognize general differences in the data transfer of radio, television, telephone and the Internet
- to understand why internet video streaming is vulnerable to delays, and why delays do not affect radio, television and telephone

Sub-areas covered

- Computer networks
- Transferring data through the Internet

Keywords

- **streaming protocol** - a set of rules to ensure that data will be supported in real time
- **network protocol** - a set of rules that set out how to establish communication between two or more computers over the network
- **routing** - the process of selecting paths in a network along which data can be sent between computers (through the router)
- **router** - a network device that groups computers in the network and establishes an area inside it
- **Internet** - a global network connecting millions of computers
- **peering** - the arrangement of traffic exchange between the Internet service providers (ISPs).
- **Internet provider** - a company that sells bandwidth and access to the Internet
- **bandwidth** - the amount of data that can be transferred through a specific path in the network usually expressed in kb/s [kilo bits per second]

- **circuit switching network** - a network in which computers establish a constant bandwidth connection before they start to share any data
- **broadcast** – the action of sending data by one computer in the network to all the others that are available inside the network

Summary

The text, divided into several parts, familiarises the reader with the problem of receiving video data transferred through a network (especially the Internet). The author outlines the methods of transferring data by means of phone lines, radio waves and television wires. He explains why the transfer of data is not problematic in those cases, while in computer networks it poses a real problem. After giving a brief account of the basis of the global network, the author describes the transfer of video data through the Internet. The article gives detailed descriptions of methods of transferring that type of data and lists reasons why receiving them can be a problem.

Tekst przybliża czytelnikowi problem odbioru danych wideo transportowanych przez sieć, zwłaszcza przez Internet. By naświetlić problem przesyłania tego typu danych autor artykułu opisuje sposób transferu danych przez linię telefoniczną, fale radiowe oraz sieć telewizyjną. Zaznacza on powody, dla których wymienione rozwiązania nie nastręczają problemów typowych dla sieci komputerowych. Artykuł przedstawia także zarys funkcjonowania intersieci oraz opisuje główne powody, dla których przesyłanie i właściwe odbieranie tego typu danych jest problematyczne.

Pre-reading exercises

1. What do you know about the history of TV and radio?
2. Do you know the origins of the Internet?
3. Which kind of media do you find important and valuable?
4. Can you explain how connections are made between 'users' of radio, TV and the Internet?

How to Stream Video over a Network or the Internet

How Video Travels Across the Internet



As noted in earlier chapters and as any end user would be quick to point out, viewing streaming video over the Internet is hardly a seamless experience. Streaming video suffers from hiccups, delays, drop-outs, skips, and connection loss. In this section, we explain how the Internet moves data and how this affects video playback.

It's sometimes hard to understand why the Internet has trouble moving audio and video when radio, television, and telephones do it fairly well and have existed for almost 100 years. So first let's look at the mechanisms of these traditional media.

Radio

router
a network device that groups computers in the network and establishes an area inside it

Radio works simply because a single tower broadcasts the same signal to many receivers. Everyone listens to the same thing at the same time. All the stations are available at any time; you simply have to tune into a different frequency signal. The main barriers to radio transmission are distance; physical barriers such as hills, buildings, and tunnels that block the signal; and interference between two strong signals near each other on the dial. In terms of communication, radio is a one-way broadcast transmission.

Television

Television works much like radio, except that television broadcasting is organized into national networks. The same program is delivered to television receivers around the country by broadcasting the originating signal to branch offices, which broadcasts it out from towers, out through cable companies, or to people with satellite dishes. In any case, the same signal is sent to everyone at the same time - a one-way broadcast. All the channels are available at any time; there is no noticeable delay caused by changing channels. The main barriers to television reception are bent or frayed cables, badly aimed antennas or dishes, physical barriers as in radio, and interference of stations with each other.

Telephone

Telephone calls use many of the same wires used by the Internet. The telephone central office maintains devices called switches (automated versions of the classic telephone switchboard) that are used to connect the call to the next location. Telephone calls create a two-way circuit all the way from caller to receiver. The message "All circuits are busy" - usually heard only during disasters or radio call-in concert ticket giveaways - means the switch does not have any more slots in which to carry this call. The main barriers to telephone transmission are found at the beginning of the call— if there are not enough circuits to place the call. While a call is in progress, the entire route between the caller and recipient is reserved for their use only, even if there is silence and no one is talking. Telephones use what is called a circuit-switched connection.

routing
the process of selecting paths in a network along which data can be sent between computers (through the router)

Internet Basics

The path from a website to a web browser is different than these other systems. Conceptually, it is similar to the telephone conversation: It's a two-way conversation in which the browser asks for a document and the server sends it. Unlike the telephone call, however, there is no reserved circuit. Data, in the form of requests and responses, are organized into chunks called packets and sent between the requesting web browser and the web server.

In between the requester and the server is a series of routers. These machines route traffic between different smaller networks. Each time a packet crosses the boundary from one ISP to another, or from one kind of network to another, it goes through a router. The packets "hop" from router to router like a bucket brigade. This type of data transmission is called packet switching, instead of circuit switching. Internet packet switching has some attributes that make it reliable and unreliable at the same time.

bandwidth
the amount of data that can be transferred through a specific path in the network (usually expressed in kb/s [kilo bits per second])

The Internet is an extremely heterogeneous network, consisting of several different kinds of networks and ways of connecting networks to the Internet, as described in the next few sections.

The Internet Backbone

The Internet backbone (as much as a large, shapeless and ever-shifting cloud of networks can have a backbone!) consists of long-haul connections that carry large volumes of Internet traffic (packets) across and between continents.

streaming protocol
a set of rules to ensure that data will be supported in real time

network protocol
- a set of rules that set out how to establish communication between two or more computers over the network

Public Exchange Points

Public exchange points exist at various points on continents and are the major nerve centers where many regional private networks, Internet providers, corporations, schools, and government divisions—large and small—converge to exchange traffic destined for other points on the Internet. You can compare these centers to major public airports, where international and domestic flights arrive 24 hours a day and trade passengers from different airlines.

Peering

The process of connecting a network to the Internet at one of these exchange points is called peering, and connecting to the backbone this way makes one a Tier-1 Internet provider. ISPs that rent their connection from a Tier-1 provider are called Tier-2 providers, and so on. The policies, prices, and agreements that cover how data is treated on these connections are as numerous as there are companies involved. This is the first source of variability for our packet switching.

Private Peering

Peering is simply two networks connecting to each other with routers. Public peering occurs at large exchange points, but any two networks that find a lot of traffic flowing between them can choose to create a direct private link between the networks (called private peering). This reduces the cost of access through a public exchange point or other provider for all the bandwidth that travels between these two networks. It also decreases the number of intermediate connections between the networks. For instance, when several schools in the same organization link together, their inter-campus network traffic does not have to go out to the Internet at large, and is often more reliable as a result. In this scenario, though, each school has its own connection to the Internet. What if one of the school's Internet connections went down? Would it be fair to send its traffic through the private peering connection and use another school's Internet connection? The way these kinds of questions are answered and the internal policies in this regard are another contributing factor to the variability of Internet packet switching.

circuit switching network
a network in which computers establish a constant bandwidth connection before they start to share any data

Internet Complexity

As everything "goes digital," the distinction between cable TV wires, telephone wires, radio waves, and satellite transmission blurs. However, there are many ways to send data over these media. Internet data transmission can be complicated, leading to a variety of undesirable transmission characteristics.

Packet Loss

Circuit switching on the Internet is described as "best-effort," meaning that one of the routers along the way can lose a packet before it reaches its destination. In this case, the sender or receiver must somehow note that the packet was lost (perhaps by receiving the next packet and noting that it is out of context) and re-requesting the lost packet. This mechanism is fairly reliable in that two machines will usually (and eventually) figure out what went wrong and resend the missing packets. Packet loss causes audio and video to pause if the packets are eventually resent, and it causes video to pause, drop out, and skip if the packets are not resent at all. In our analogy of a public exchange point being a major airport, if it's a "foggy day" at that exchange, the part of the Internet that goes through that exchange can be slowed down (called a brownout) by the data that can't "take off."

Different Routes

Not all packets in a file follow the same route to the destination computer. This is not unlike the airline's hub and spoke system: One packet might go "direct" from San Francisco to Washington; others might "transfer" in Atlanta or Chicago to get to Washington. Contributing to this issue is private peering and the variable rules and costs associated with all the choices to be made. Alternate routes can be excellent when one path between two machines goes down and a packet can use another path. It can also cause strange effects, such as when a packet is sent down a slow route, is assumed lost, is resent - and then later reappears as a duplicate packet! Audio can stutter and skip if duplicate packets are not detected and discarded. Also, some paths travel far out of the way, hopping through many more routers than necessary and causing large delays. The more "hops" or routers between two machines, the higher the chance of unexpected delays.

Delay (Latency)

Because of the many different routers a packet has to go through to get from sender to receiver and because there are no reserved circuits on the Internet like there are for telephones, the delay of any given packet can be high or low, or change unexpectedly. This can be caused by a variety of factors such as: A router is too busy and can't keep up with traffic. A particular link between sender and receiver becomes saturated. A link goes down, causing traffic to be rerouted to a different link. One or more routers in between can't think fast enough. A firewall looks at all the packets for viruses. Delay is added due to the use of older technology, such as modems. Other downloads on a pipe cause it to delay. Packets are lost, resulting in resends, and other packets get bunched up behind them. These factors make predicting how long it will take to get packets back from a server difficult. Because of varying latency, video can take a

English++

peering
the arrangement
of traffic exchange
between the Internet
service providers (ISPs).

broadcast
the action of sending data
by one computer
in the network to
all the others
that are available
inside the network

long time to start playing; fast-forward and rewind features can be slow and clunky; and video can pause, stutter, skip, and stop altogether. Bandwidth Variation Another factor on the Internet is the variability of bandwidth. With broadcast media, such as radio or television, as well as telephones, the bandwidth is always the same - just enough to carry the channel or the conversation. There is no wasted bandwidth; the size of the channel is just enough to carry the data. It was designed to be that way. Because the Internet is designed to allow different computers of different speeds and different channel sizes communicate, it is possible to have bottlenecks, not just due to traffic that the size of the channel varies from sender to receiver. The Internet link for a major website's hosting provider might be excellent. The links between the host's ISP and its branch in a particular city might be high-capacity. However, the Internet link provided by a small ISP to the end user might be very small due to oversubscription. If that Internet provider has incurred a good deal of customer growth without upgrading its own connection to the Internet backbone, the potentially high-bandwidth connection from the website host is lowered to the slowest intermediate link in the chain. In other words, the bandwidth between a website and a client is no faster than its slowest link.

Internet provider
a company that sells bandwidth and access to the Internet

NOTE

Fundamentally, the Internet is far better suited for sending web pages than real-time media because web pages are far smaller and far less sensitive to delays. There is not much difference between a one- and two-second delay in getting a web page, but a one-second pause in real-time video is unacceptable. The brute-force approach of keeping the bit rate of the video far below the maximum bandwidth of the Internet connection can be effective in getting Internet video to perform predictably.

Internet
a global network connecting millions of computers

Exercises

Pre-reading questions

1. What do you know about the history of TV and radio?

- The first use of radio took place Franklin Institute in Philadelphia in February 1893. It was the demonstration of wireless telegraph. Transmission of voice was the invention of the beginning of XX century. Development of TV was divided into two paths: the mechanical and electrical and purely electrical. The first idea of construction the TV (electromechanical) was abandoned about 1925. Demonstration of first color television display was given on August 16, 1944.

2. Do you know the origins of the Internet?

- The first idea of global network of computers was created by J.C.R. Licklider. He moved to the Defense Advanced Research Projects Agency (DARPA) to develop it. The first connection of two computers (by the phone line) took place in 1965. The first form of the Internet, called ARPANET, was brought online in 1969. In those time, the first ideas of reliability of connections (nowadays included in many protocols) was implemented into networks.

3. Which kind of media do you find important and valuable?

- I find the Internet the most important and valuable kind of media. Any information that can be found in other media (tv or radio) can also be found in the Internet.

4. Can you explain how connections are made between 'users' of radio, TV and the Internet?

- The radio waves are divided into frequency which enables the receivers to distinguish the channels. The same way of transferring data in the wireless TV is used to receive from broadcasting stations. When using cable connected TV receiver no division of data stream is needed as one channel is received through the cable in time. In the Internet every media is shared so protocols (describing the way the data is transferred) need to provide effective way to avoid collisions of the packets.

Comprehension questions

1. **Can you point out the differences between TV, radio and the Internet in terms of communication?**

- only broadcast communication (TV, Radio)
- each of nodes can be both sender and recipient (the Internet)
- information can be reviewed any time (the Internet)

2. **Can you explain the process of establishing connections by telephone and on the Internet?**

- Telephone calls use many of the same wires used by the Internet. The telephone central office maintains devices called switches (automated versions of the classic telephone switchboard) that are used to connect the call to the next location.

Telephone calls create a two-way circuit all the way from caller to receiver. The message "All circuits are busy"—usually heard only during disasters or radio call-in concert ticket giveaways—means the switch does not have any more slots in which to carry this call. The main barriers to telephone transmission are found at the beginning of the call— if there are not enough circuits to place the call. While a call is in progress, the entire route between the caller and recipient is reserved for their use only, even if there is silence and no one is talking. Telephones use what is called a circuit-switched connection.

3. **What is the idea of "packet communication"?**

- The path from a website to a web browser is different than these other systems. Conceptually, it is similar to the telephone conversation: It's a two way conversation in which the browser asks for a document and the server sends it. Unlike the telephone call, however, there is no reserved circuit. Data, in the form of requests and responses, are organized into chunks called packets and sent between the requesting web browser and the web server. In between the requester and the server are a series of routers. These machines route traffic between different smaller networks. Each time a packet crosses the boundary from one ISP (Internet Service Provider) to another or from one kind of network to another, it goes through a router. The packets "hop" from router to router like a bucket brigade. This type of data transmission is called packet switching, instead of circuit switching. Internet packet switching has some attributes that make it reliable and unreliable at the same time.

4. **Explain, in general terms, how routing works.**

- Routing is the process of selecting paths in a network along which to send data or physical traffic, usually directs forwarding on the basis of routing tables which maintain a record of the routes to various network destinations. Thus constructing routing tables, which are held in the routers' memory, becomes very important for efficient routing.

5. **Give 5 reasons for packet delays.**

- a router is too busy and can't keep up with traffic
- a particular link between sender and receiver becomes saturated
- a link goes down, causing traffic to be rerouted to a different link
- one or more routers in between can't think fast enough
- a firewall looks at all the packets for viruses
- delay is added due to the use of older technology, such as modems
- other downloads on a pipe cause it to delay
- packets are lost, resulting in resends, and other packets get bunched up behind them

Further exercises

1. Pre-prepare a list of features. Classify them into four groups: Internet, telephone, radio, television. The features could be taken from the text and/or worked out by students working in groups.

TV	Radio	Phone	Internet
broadcast	broadcast	direct	broadcast/direct
real-time	real-time	real-time	real-time/ prepared documents
		circuit-switched	packet-based
			routing
			streaming transmission

Possible topics for discussion

1. What do you think is the main cause of interruptions in internet transmission?
2. Why is audio data not transferred in the same way as telephone calls although both the Internet and telephone calls use the same wires?

- Fundamentally, the Internet is far better suited for sending web pages than real-time media because web pages are far smaller and far less sensitive to delays. There is not much difference between a one- and two-second delay in getting a web page, but a one-second pause in real-time video is unacceptable.

3. How do you use the Internet? (Which Internet services do you use most often?)
4. Do you use the Internet for making phone calls? Why/why not? What are advantages and disadvantages?

- low price
- video transmission is possible
- there is no difference (in costs) between long and short distance communication
- the phone call quality depends on bandwidth of the connection
- requires computer connected to the Internet

Computer Simulation

Wikipedia

Number of words	2220
Computer science content	Low
Math content	Medium
English language complexity	Medium

Learning objectives

- to acquire basic knowledge about simulation kinds and history
- to understand difference between simulation and modelling
- to understand the power of simulation and its influence on modern science

Sub-areas covered

- Computer simulation
- Computer graphics
- Maths

Keywords

- **mathematical model** - an abstract model that uses mathematical language to describe a system
- **stochastic process** - a process with an indeterminate or random element as opposed to a deterministic process that has no random element
- **discrete** - not supporting or requiring the notion of continuity; discrete objects are countable sets such as integers
- **Computer Generated Imagery (CGI)** - an application of the field of computer graphics (or, more specifically, 3D computer graphics) to special effects in films, television programs, commercials and simulation
- **differential equation** - a mathematical equation for an unknown function of one or several variables that relates the values of the function itself and of its derivatives
- **gamut** - a complete range or extent
- **Monte Carlo method** - a computational algorithm which relies on repeated random sampling to compute its results

Summary

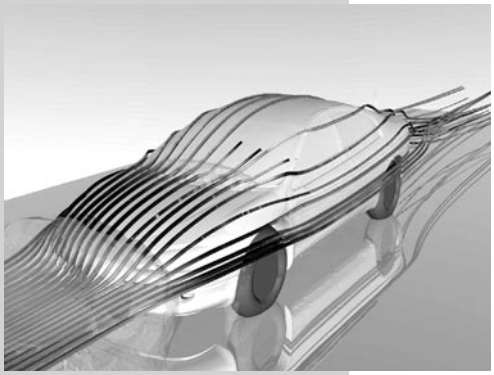
The article presents a general view of computer simulations. It explains what a computer simulation is and the differences between simulation and modelling. It provides a brief overview of the history of computer simulation and divides computer simulation into specific types: stochastic and deterministic and continuous and discrete, giving examples of each kind. It sets out the advantages of CGI simulations and describes the impact of simulations on present-day science. The final part of the article lists practical contexts for computer simulation which are important in everyday life.

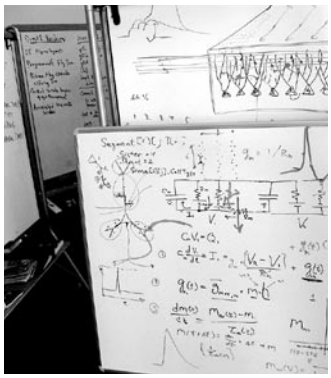
Artykuł prezentuje ogólną ideę symulacji komputerowych. Wyjaśnia, co nazywamy symulacją komputerową oraz jaka jest różnica między symulacją a modelowaniem. Krótko opowiada o historii symulacji komputerowych, oraz prezentuje różne typy symulacji: stochastyczne i deterministyczne, ciągłe i dyskretne wraz z przykładami. Przedstawia zalety symulacji typu CGI oraz wpływ symulacji na współczesną naukę. Na końcu możemy się dowiedzieć, gdzie spotykamy się z symulacją w naszym codziennym życiu.

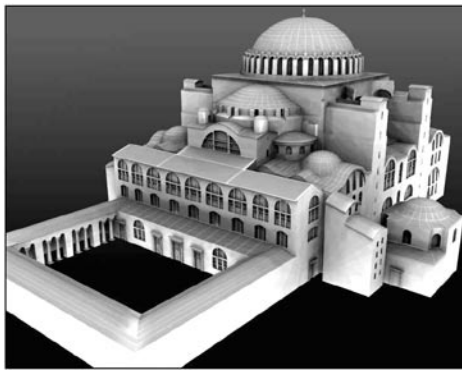
Pre-reading exercises

1. How can we solve problems in cases where physical models are too complex or expensive to build?
2. Name some problems which can be solved using computer simulation?

Computer simulation







A computer simulation (also referred to as a computer model or a computational model) is a computer program, or network of computers, that attempts to simulate an abstract model of a particular system. Computer simulations have become a useful part of the mathematical modelling of many natural systems in physics (computational physics), chemistry and biology; human systems in economics, psychology, and social science, and in the process of engineering new technology, so as to gain insight into the operation of those systems or to observe their behaviour.

mathematical model
an abstract model that uses mathematical language to describe a system

Computer simulations vary from computer programs that run a few minutes, to network--based groups of computers running for hours or ongoing simulations that run for days. The scale of events being simulated by computer simulations has far exceeded anything possible (or perhaps even imaginable) using the traditional paper-and-pencil mathematical modelling: over 10 years ago. A desert-battle simulation, of one force invading another, involved the modelling of 66,239 tanks, trucks and other vehicles on simulated terrain around Kuwait, using multiple supercomputers in the DoD High Performance Computer Modernization Program. Another simulation ran a 1-billion--atom model, where previously, a 2.64-million-atom model of a ribosome, in 2005, had been considered a massive computer simulation. And the Blue Brain project at EPFL (Switzerland) began in May 2005 to create the first computer simulation of the entire human brain, right down to the molecular level.

Simulation versus modelling

Traditionally, the formal modelling, or modelling, of systems has been via a mathematical model, which attempts to find analytical solutions to problems, which enables the prediction of the behaviour of the system from a set of parameters and initial conditions.

While computer simulations might use some algorithms from purely mathematical models, computers can combine simulations with the reality of actual events, such as generating input responses to simulate test subjects who are no longer present. Although the missing test subjects (i.e. the users of equipment or systems) are being

modelled/simulated, the whole process can be conducted with the actual equipment or system they use, revealing performance limits or defects in long-term use by the simulated users.

Note that the term computer simulation is broader than computer modelling, which implies that all aspects are being modelled in the computer representation. However, computer simulation also includes generating inputs from simulated users to run actual computer software or equipment, with only part of the system being modelled: an example would be flight simulators which can run machines as well as actual flight software.

Computer simulations are used in many fields, including science, technology, entertainment, and business planning and scheduling.

stochastic process
a process
with an indeterminate
or random element
as opposed to
a deterministic process
that has no random
element

History

Computer simulation was developed hand-in-hand with the rapid growth of the computer, following its first large-scale deployment during the Manhattan Project in World War II to model the process of nuclear detonation. It was a simulation of 12 hard spheres using a Monte Carlo algorithm. Computer simulation is often used as an adjunct to, or substitution for, modelling systems for which simple closed form analytic solutions are not possible. There are many different types of computer simulation; the common feature they all share is the attempt to generate a sample of representative scenarios for a model in which a complete enumeration of all possible states of the model would be prohibitive or impossible. Computer models were initially used as a supplement for other arguments, but their use later became rather widespread.

Data preparation

The data input/output for the simulation can be either through formatted text files or a pre- and post processor.

Types of computer simulation

Computer models can be classified according to several criteria including:

- stochastic or deterministic (and as a special case of deterministic, chaotic)
- steady-state or dynamic
- continuous or discrete (and as an important special case of discrete, discrete event or DE models)
- local or distributed.

For example, steady-state models use equations defining the relationships between elements of the modelled system and attempt to find a state in which the system is in equilibrium. Such models are often used in simulating physical systems as a simpler

gamut
a complete range or extent

modelling case before dynamic simulation is attempted. Dynamic simulations model changes in a system in response to (usually changing) input signals. Stochastic models use random number generators to model chance or random events; they are also called Monte Carlo simulations. A discrete event simulation (DES) manages events in time. Most computer, logic-test and fault-tree simulations are of this type. In this type of simulation, the simulator maintains a queue of events sorted by the simulated time in which they should occur. The simulator reads the queue and triggers new events as each event is processed. It is not important to execute the simulation in real time. It's often more important to be able to access the data produced by the simulation, to discover logic defects in the design or the sequence of events. A continuous dynamic simulation performs numerical solutions of differential-algebraic equations or differential equations (either partial or ordinary). Periodically, the simulation program solves all the equations, and uses the numbers to change the state and output of the simulation. Applications include flight simulators, simulation games, chemical process modelling, and simulations of electrical circuits. Originally, these kinds of simulations were actually implemented on analogue computers, where the differential equations could be represented directly by various electrical components such as op-amps. By the late 1980s, however, most "analogue" simulations were run on conventional digital computers that emulate the behaviour of an analogue computer. A special type of discrete simulation which does not rely on a model with an underlying equation, but can nonetheless be represented formally, is agent-based simulation. In agent-based simulation, the individual entities (such as molecules, cells, trees or consumers) in the model are represented directly (rather than by their density or concentration) and possess an internal state and set of behaviours or rules which determine how the agent's state is updated from one time-step to the next. Distributed models run on a network of interconnected computers, possibly through the Internet. Simulations dispersed across multiple host computers like this are often referred to as "distributed simulations". There are several standards for distributed simulation, including Aggregate Level Simulation Protocol (ALSP), Distributed Interactive Simulation (DIS), the High Level Architecture (HLA) and the Test and Training Enabling Architecture (TENA).

discrete
not supporting or requiring the notion of continuity; discrete objects are countable sets such as integers

CGI computer simulation

Formerly, the output data from a computer simulation was sometimes presented in a table, or a matrix, showing how data was affected by numerous changes in the simulation parameters. The use of the matrix format was related to the traditional use of the matrix concept in mathematical models; however, psychologists and others noted that humans could quickly perceive trends by looking at graphs or even moving-images or motion-pictures generated from the data, as displayed by computer -generated-imagery (CGI) animation. Although observers couldn't necessarily read

out numbers, or spout maths formulas, from observing a moving weather chart, they might be able to predict events (and "see that rain was headed their way"), much faster than scanning tables of rain-cloud coordinates. Such intense graphical displays, which transcended the world of numbers and formulae, sometimes also led to output that lacked a coordinate grid or omitted timestamps, as if straying too far from numeric data displays. Today, weather forecasting models tend to balance the view of moving rain/snow clouds against a map that uses numeric coordinates and numeric timestamps of events.

Similarly, CGI computer simulations of CAT scans can simulate how a tumour might shrink or change, during an extended period of medical treatment, presenting the passage of time as a spinning view of the visible human head, as the tumour changes.

Other applications of CGI computer simulations are being developed to graphically display large amounts of data in motion, as changes occur during a simulation run.

Computer simulation in science

The following are generic examples of types of computer simulations in science, which are derived from an underlying mathematical description:

A numerical simulation of differential equations which cannot be solved analytically. Falling into this category are:

- theories which involve continuous systems such as phenomena in physical cosmology
- fluid dynamics (e.g. climate models, roadway noise models, roadway air dispersion models)
- continuum mechanics and chemical kinetics

A stochastic simulation, typically used for discrete systems where events occur probabilistically, and which cannot be described directly with differential equations (this is a discrete simulation in the above sense). Phenomena in this category include:

- genetic drift
- biochemical or gene regulatory networks with small numbers of molecules (See also: Monte Carlo method).

Specific examples of computer simulations follow:

- statistical simulations based upon an agglomeration of a large number of input profiles, such as the forecasting of equilibrium temperature of receiving waters, allowing the gamut of meteorological data to be input for a specific locale. This technique was developed for thermal pollution forecasting.
- agent based simulation has been used effectively in ecology, where it is often called individual based modelling and has been used in situations for which individual

Computer Generated Imagery (CGI)
an application of the field of computer graphics (or, more specifically, 3D computer graphics) to special effects in films, television programs, commercials and simulation

variability in the agents cannot be neglected, such as population dynamics of salmon and trout (most purely mathematical models assume all trout behave identically)

- time stepped dynamic model; in hydrology there are several such hydrology transport models such as the SWMM and DSSAM Models developed by the U.S. Environmental Protection Agency for river water quality forecasting
- computer simulations have also been used to formally model theories of human cognition and performance, e.g. ACT-R
- computer simulation using molecular modelling for drug discovery
- computational fluid dynamics simulations are used to simulate the behaviour of flowing air, water and other fluids. There are one-, two- and three- dimensional models used. A one dimensional model might simulate the effects of water hammer in a pipe. A two-dimensional model might be used to simulate the drag forces on the cross-section of an aeroplane wing. A three-dimensional simulation might estimate the heating and cooling requirements of a large building.

Monte Carlo method
a computational algorithm which relies on repeated random sampling to compute its results

Understanding of statistical thermodynamic molecular theory is fundamental to the appreciation of molecular solutions. Development of the Potential Distribution Theorem (PDT) allows one to simplify this complex subject to down-to-earth presentations of molecular theory.

Notable, and sometimes controversial, computer simulations used in science include:

- Donella Meadows' World3 used in the Limits to Growth
- James Lovelock's Daisyworld
- Thomas Ray's Tierra.

Simulation environments for physics and engineering

Graphical environments to design simulations have been developed. Special care was taken to handle "events" (situations in which the simulation equations are not valid and have to be changed). The open project Open Source Physics was started in order to develop reusable libraries for simulations in Java, together with Easy Java Simulations, a complete graphical environment that generates code based on these libraries.

Pitfalls in computer simulation

Although sometimes ignored in computer simulations, it is very important to perform sensitivity analysis to ensure that the accuracy of the results is properly understood. For example, the probabilistic risk analysis of factors determining the success of an oilfield exploration program involves combining samples from a variety of statistical distributions using the Monte Carlo method. If, for instance, one of the key parameters (i.e. the net ratio of oil-bearing strata) is known to only one significant figure, then the result of the simulation might not be more precise than one significant figure, although it might (misleadingly) be presented as having four significant figures.

Computer simulation in practical contexts

Computer simulations are used in a wide variety of practical contexts, such as:

- analysis of air pollutant dispersion using atmospheric dispersion modelling
- design of complex systems such as aircraft and logistics systems
- design of noise barriers to effect roadway noise mitigation
- flight simulators to train pilots
- weather forecasting
- behaviour of structures (such as buildings and industrial parts) under stress and other conditions
- design of industrial processes, such as chemical processing plants
- strategic management and organizational studies
- reservoir simulation for the petroleum engineering to model the subsurface reservoir
- Process Engineering Simulation tools
- robot simulators for the design of robots and robot control algorithms

The reliability and the trust people put in computer simulations depends on the validity of the simulation model, therefore verification and validation are of crucial importance in the development of computer simulations. Another important aspect of computer simulations is that of reproducibility of the results, meaning that a simulation model should not provide a different answer for each execution. Although this might seem obvious, this is a special point of attention in stochastic simulations, where random numbers should actually be semi-random numbers. An exception to reproducibility are "human-in-the-loop" simulations, such as flight simulations and computer games. Here a human is part of the simulation and thus influences the outcome in a way that is hard if not impossible to reproduce exactly.

Computer graphics can be used to display the results of a computer simulation. Animations can be used to experience a simulation in real-time e.g. in training simulations. In some cases animations may also be useful in faster than real-time or even slower than real-time modes. For example, faster than real-time animations can be useful in visualizing the build-up of queues in the simulation of humans evacuating a building. Furthermore, simulation results are often aggregated into static images using various ways of scientific visualization.

In debugging, simulating a program execution under test (rather than executing natively) can detect far more errors than the hardware itself can detect and, at the same time, log useful debugging information such as instruction trace, memory alterations and instruction counts. This technique can also detect buffer overflow and similar "hard to detect" errors as well as produce performance information and tuning data.

differential equation
a mathematical equation for an unknown function of one or several variables that relates the values of the function itself and of its derivatives

Exercises

Pre-reading exercises

1. How can we solve problems in cases where physical models are too complex or expensive to build?

- computer simulation

2. Name some problems which can be solved using computer simulation?

- spread of infectious diseases
- panic in crowds
- crash testing
- explosions modelling
- aerodynamic solutions testing
- building demolition

Comprehension question

1. **Explain the differences between discrete/continuous and stochastic/deterministic simulation, giving examples.**

- Stochastic models use random number generators to model chance or random events; they are also called Monte Carlo simulations.
- Deterministic models have no random elements.
- A discrete event simulation (DES) manages events in time. Most computer, logic-test and fault-tree simulations are of this type. In this type of simulation, the simulator maintains a queue of events sorted by the simulated time they should occur. The simulator reads the queue and triggers new events as each event is processed. It is not important to execute the simulation in real time.
- A continuous dynamic simulation performs numerical solution of differential -algebraic equations or differential equations (either partial or ordinary). Periodically, the simulation program solves all the equations, and uses the numbers to change the state and output of the simulation. Applications include flight simulators, simulation games, chemical process modelling, and simulations of electrical circuits.

Possible topics for discussion

1. Is it possible to simulate any problem?

Possible difficulties

A big number of mathematical terms.

Rather than focusing on the words you don't understand, try to get the main idea.

.

Computer Facial Animation

Wikipedia

Number of words	1970
Computer science content	Medium
Math content	Low
English language complexity	Medium

Learning objectives

- to acquire basic vocabulary related to facial
- to become familiar with different 3D-animation systems
- to gain understanding of the major problems connected with speech animation

Sub-areas covered

- Computer graphics

Keywords

- **morphing** - a special effect in motion pictures and animations that changes (or morphs) one image into another through a seamless transition
- **rendering** - the process of generating an image from a model by means of computer programs
- **keyframe** - or a key frame in animation and film making is a drawing which defines the starting and ending points of any smooth transition texture - a bitmap image applied to a surface in computer graphics
- **texture** - a bitmap image applied to a surface in computer graphics
- **computervision** - a branch of artificial intelligence that deals with computer processing of images from the real world
- **alignment** - the adjustment of an object in relation to other objects, or a static orientation of some object or set of objects in relation to others
- **motion capture** - a technique of recording the actions of human actors and using that information to animate digital character models in 3D animation

Summary

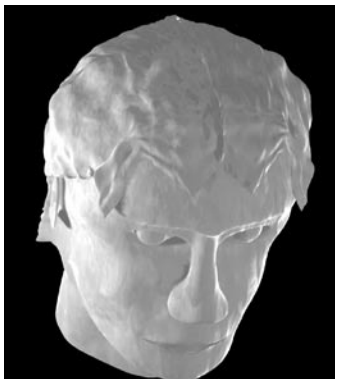
A fairly general text about facial animation in computer graphics. It contains a short history of facial animation from the Facial Action Coding System, which defines 64 basic facial Action Units, to films like: The Matrix or The Lord of the Rings. The second part describes animation techniques: 2D animation, 3D animation and speech animation.

Tekst traktuje w sposób ogólny problem komputerowej animacji twarzy. Pierwsza część zapoznaje nas z historią tego zagadnienia poczynając od roku 1978, gdy pierwszy raz pojawił się FACS (Facial Action Coding System) zawierający 64 podstawowe jednostki ruchu twarzy, na animacji twarzy w filmach takich jak Matrix, czy Władca Pierścieni kończąc. Dalsza część tekstu zapoznaje nas z technikami używanymi w komputerowej animacji twarzy. Po kolei omawiane są techniki 2D, 3D i animacja mowy.

Pre-reading questions

1. Students are given pictures of human faces and try to guess what emotions they show.
2. What is the number of facial muscles?
3. Why are facial expressions so important in our lives?
4. Where can facial animations be used?
5. What is the rendering process in computer graphics? What is the motion capture technique?

Computer facial animation




Computer facial animation is primarily an area of computer graphics that encapsulates models and techniques for generating and animating images of the human head and face. Due to its subject and output type, it is also related to many other scientific and artistic fields from psychology to traditional animation. The importance of human faces in verbal and non-verbal communication and advances in computer graphics hardware and software have caused considerable scientific, technological, and artistic interest in computer facial animation.

Although the development of computer graphics methods for facial animation started in the early 1970s, major achievements in this field are more recent and have taken place since the late 1980s.

Computer facial animation includes a variety of techniques from morphing to three -dimensional modelling and rendering. It has become well-known and popular through animated feature films and computer games but its applications include many more areas such as communication, education, scientific simulation, and agent-based systems (for example, online customer service representatives).

rendering
the process of generating an image from a model by means of computer programs

History

Human facial expressions have been the subject of scientific investigation for more than one hundred years. The study of facial movements and expressions started from a biological point of view. After some older investigations, i.e. by John Bulwer in late 1640s, Charles Darwin's book The Expression of the Emotions in Men and Animals can be considered a major departure for modern research in behavioural biology.

More recently, one of the most important attempts to describe facial activities (movements) was the Facial Action Coding System (FACS). Introduced by Ekman and Friesen in 1978, FACS defines 64 basic facial Action Units (AUs). A major group of these Action Units represent primitive movements of facial muscles in actions such as raising brows, winking, and talking. Eight AUs are for rigid three-dimensional head movements, i.e. turning and tilting left and right and going up, down, forward and backward. FACS

morphing
a special effect in motion pictures and animations that changes (or morphs) one image into another through a seamless transition

keyframe
or a key frame in animation and film making is a drawing which defines the starting and ending points of any smooth transition texture - a bitmap image applied to a surface in computer graphics

has been successfully used for describing desired movements of synthetic faces and also in tracking facial activities.

Computer based facial expression modelling and animation is not a new endeavour. The earliest work with computer based facial representation was done in the early 1970s. The first three-dimensional facial animation was created by Parke in 1972. In 1973, Gillenson developed an interactive system to assemble and edit line drawn facial images. And in 1974, Parke developed a parameterized three-dimensional facial model.

The early 1980s saw the development of the first physically-based muscle-controlled face model by Platt and the development of techniques for facial caricatures by Brennan. In 1985, the short animated film Tony de Peltrie was a landmark for facial animation; for the first time computer facial expression and speech animation were a fundamental part of telling the story.

The late 1980s saw the development of a new muscle-based model by Waters, the development of an abstract muscle action model by Magnenat-Thalmann and colleagues, and approaches to automatic speech synchronization by Lewis and by Hill. The 1990s saw increasing activity in the development of facial animation techniques and the use of computer facial animation as a key storytelling component as illustrated in animated films such as Toy Story, Antz, Shrek, and Monsters, Inc, and computer games such as Sims. Casper (1995) is a milestone in this period, being the first movie with a lead actor produced exclusively using digital facial animation (Toy Story was released later the same year).

computer vision
the science and technology of machines that see

The sophistication of the films increased after 2000. In The Matrix Reloaded and Matrix Revolutions dense optical flow from several high-definition cameras was used to capture realistic facial movement at every point on the face. Polar Express used a large Vicon system to capture upward of 150 points. Although these systems are automated, a large amount of manual clean-up effort is still needed to make the data usable. Another milestone in facial animation was reached by The Lord of the Rings where a character specific shape base system was developed. Mark Sagar pioneered the use of FACS in entertainment facial animation, and FACS based systems developed by Sagar were used on Monster House, King Kong, and other films.

Techniques

2D Animation

Two-dimensional facial animation is commonly based upon the transformation of images, including both images from still photography and sequences of video. Image morphing is a technique which allows in-between transitional images to be generated

between a pair of target still images or between frames from sequences of video. These morphing techniques usually consist of a combination of a geometric deformation technique, which aligns the target images, and a cross-fade, which creates the smooth transition in the image texture. An early example of image morphing can be seen in Michael Jackson's video for Black And White. In 1997 Ezzat and Poggio working at the MIT Center for Biological and Computational Learning created a system called Miketalk, which morphs between image keyframes, representing visemes, to create speech animation.

Another form of animation from images consists of concatenating together sequences captured from video. In 1997 Bregler et al. described a technique called video-rewrite, where existing footage of an actor is cut into segments corresponding to phonetic units which are blended together to create new animations of a speaker. Video -rewrite uses computer vision techniques to automatically track lip movements in video and these features are used in the alignment and blending of the extracted phonetic units. This animation technique only generates animations of the lower part of the face, these are then composited with video of the original actor to produce the final animation.

computer vision
a branch of artificial intelligence that deals with computer processing of images from the real world

3D Animation

Three-dimensional head models provide the most powerful means of generating computer facial animation. One of the earliest works on computerized head models for graphics and animation was done by Parke. The model was a mesh of 3D points controlled by a set of conformation and expression parameters. The former group controls the relative location of facial feature points such as eye and lip corners. Changing these parameters can re-shape a base model to create new heads. The latter group of parameters (expression) are facial actions that can be performed on a face, such as stretching lips or closing eyes. This model was extended by other researchers to include more facial features and add more flexibility. Different methods for initializing such "generic" models based on individual (3D or 2D) data have been proposed and successfully implemented. The parameterized models are effective due to the use of limited parameters, associated with the main facial feature points. The MPEG-4 standard defines a minimum set of parameters for facial animation.

Animation is done by changing parameters over time. Facial animation is approached in different ways. Traditional techniques include:

1. shapes/morph targets,
2. bones/cages,
3. skeleton-muscle systems,
4. motion capture on points on the face and

5. knowledge based solver deformations.

1. Shape based systems offer a fast playback as well as a high degree of fidelity of expressions. The technique involves modelling portions of the face mesh to approximate expressions and visemes and then blending the different sub meshes, known as morph targets or shapes. Perhaps the most accomplished character using this technique was Gollum, from The Lord of the Rings. Drawbacks of this technique are that they involve intensive manual labor, are specific to each character and must be animated by slider parameter tables.

2. 'Envelope Bones' or 'Cages' are commonly used in games. They produce simple and fast models, but are not prone to portray subtlety.

3. Skeletal Muscle systems, physically-based head models form another approach in modelling the head and face. Here the physical and anatomical characteristics of bones, tissues, and skin are simulated to provide a realistic appearance (e.g. spring-like elasticity). Such methods can be very powerful for creating realism but the complexity of facial structures make them computationally expensive and difficult to create. Considering the effectiveness of parameterized models for communicative purposes (as explained in the next section), it may be argued that physically-based models are not a very efficient choice in many applications. This does not deny the advantages of physically-based models or the fact that they can even be used within the context of parameterized models to provide local details when needed. Waters, Terzopoulos, Kahler, and Seidel (among others) have developed physically-based facial animation systems.

4. Motion capture uses cameras placed around a subject. The subject is generally fitted either with reflectors (passive motion capture) or sources (active motion capture) that precisely determine the subject's position in space. The data recorded by the cameras is then digitized and converted into a three-dimensional computer model of the subject. Until recently, the size of the detectors/sources used by motion capture systems made the technology inappropriate for facial capture. However, miniaturization and other advancements have made motion capture a viable tool for computer facial animation. Facial motion capture was used extensively in Polar Express, where hundreds of motion points were captured. This film was very accomplished and while it attempted to recreate realism, it was criticised for having fallen in the 'uncanny valley', the realm where animation realism is sufficient for human recognition but fails to convey the emotional message. The main difficulties of motion capture are the quality of the data which may include vibration as well as the retargeting of the geometry of the points.

5. Deformation Solver Face Robot.

alignment
the adjustment of an object in relation to other objects, or a static orientation of some object or set of objects in relation to others

Speech Animation

Speech is usually treated in a different way to the animation of facial expressions; this is because simple keyframe-based approaches to animation typically provide a poor approximation to real speech dynamics. Often visemes are used to represent the key poses in observed speech (i.e. the position of the lips, jaw and tongue when producing a particular phoneme); however, there is a great deal of variation in the realisation of visemes during the production of natural speech. The source of this variation is termed coarticulation, which is the influence of surrounding visemes upon the current viseme (i.e. the effect of context.) To account for coarticulation, current systems either explicitly take into account context when blending viseme keyframes or use longer units such as diphone, triphone, syllable or even word and sentence-length units.

One of the most common approaches to speech animation is the use of dominance functions introduced by Cohen and Massaro. Each dominance function represents the influence over time that a viseme has on a speech utterance. Typically the influence will be greatest at the center of the viseme and will degrade with distance from the viseme center. Dominance functions are blended together to generate a speech trajectory in much the same way that spline basis functions are blended together to generate a curve. The shape of each dominance function will be different according to both which viseme it represents and which aspect of the face is being controlled (e.g. lip width, jaw rotation etc.) This approach to computer-generated speech animation can be seen in the Baldi talking head.

Other models of speech use basis units which include context (e.g. diphones, triphones etc.) instead of visemes. As the basis units already incorporate the variation of each viseme according to context and to some degree the dynamics of each viseme, no model of coarticulation is required. Speech is simply generated by selecting appropriate units from a database and blending the units together. This is similar to concatenative techniques in audio speech synthesis. The disadvantage to these models is that a large amount of captured data is required to produce natural results, and whilst longer units produce more natural results, the size of database required expands with the average length of each unit.

Finally, some models directly generate speech animations from audio. These systems typically use hidden Markov models or neural nets to transform audio parameters into a stream of control parameters for a facial model.

motion capture
a technique of recording the actions of human actors and using that information to animate digital character models in 3D animation

texture
a bitmap image applied to a surface in computer graphics

Exercises

Pre-reading questions

1. Students are given pictures of human faces and try to guess what emotions they show.
2. What is the number of facial muscles?

- 48

3. Why are facial expressions so important in our lives?

- Gestures and facial expressions in general mean more than words or sentences.

4. Where can facial animations be used?

- In cartoons, 3D rendered films (like Toy Story, Shrek) and in computer games. Facial animation is constantly improving and in a couple of years it will be hard to distinguish between a real actor and a virtual one.

5. What is the rendering process in computer graphics? What is the motion capture technique?

- Rendering: all of the methods used to make a picture using a computer. It involves maths calculations, data loading and so on.
- Motion capture is a technique used to obtain realistic animated movement. An actor wears a special outfit with a number of sensors. When he moves, a computer captures all the necessary movement data.

Comprehension questions

1. How would you define facial animation?

- Area of computer graphics that encapsulates models and techniques for generating and animating images of the human head and face.

2. What is FACS?

- Facial Action Coding System - defines 64 basic facial Action Units which represent primitive movements of muscles in actions, e.g. raising brows, talking.

3. What was Michael Jackson's contribution to computer animation?

- His video of the song "Black Or White" was the first to use image morphing animation. The video showed a lot of human faces which are morphed from one to another.

4. What can 2D animation be based on: images from still photography or sequences of video?

- Both. There are methods for animating still images and others for animating sequences of video.

5. What is the most realistic method of 3D animation? What makes it so good?

- The skeleton-muscle system. It is so effective because the method tries to imitate human head muscles, skin and bones. It is very complicated to build a model consistent with this method but the effects are breathtaking. It cannot be used

in computer games or on-line animation because of its complexity but is great for high-budget films.

6. **How does motion capture work?**

- The subject is fitted with reflectors or sources that precisely determine its position in space. Every movement of the sensor is recorded and then retraced on the model. It's very realistic but there are also some disadvantages like vibrations of sensors causing disturbances.

7. **How are visemes used in speech animation?**

- Visemes are movements of lips and mouth muscles performed during speaking. Visemes are used to represent key poses in observed speech. (position of lips, jaw, tongue).

Possible topics for discussion

1. How do you see the future of facial animation?
2. Think of your favourite computer animated character. Try to name the techniques used to create it.
3. Is it probable that facial animation will achieve such a level of sophistication that it will be impossible to distinguish real faces from virtual ones?

Possible difficulties

The text contains professional vocabulary items. Some of them are explained.

An Introduction to the Modelling of Real-World Problems by the Simplest Ordinary Differential Equations*

Stanisław Migórski

Number of words	9100
Computer science content	Low
Math content	High
English language complexity	Medium

Sub-areas covered

- Calculus
- Ordinary differential equations
- Modelling

Learning objectives

- to understand the relations between math and real-world phenomena
- to show how mathematical descriptions of real-world phenomena are created

Keywords

ODE - ordinary differential equation - a relation that contains functions of only one independent variable, and one or more of its derivatives with respect to that variable

interval - a set of real numbers with the property that any number that lies between two numbers in the set is also included in the set. For example, the set of all numbers x satisfying $0 \leq x \leq 1$ is an interval which contains 0 and 1, as well as all numbers between them. The set of positive numbers is also an interval.

differentiable function - a real function is said to be differentiable at a point if its derivative exists at that point

---

* This work was prepared as part of the English++ project carried out by students of the Institute of Computer Science, Jagiellonian University, Kraków in the academic year 2007–2008.

* The author is grateful for the encouragement he received from the students and Faculty at the Jagiellonian University. His special thanks go to Monika Stawicka, who through her untiring efforts and devotion to duty convinced him to join the project.

separable differential equation - a class of equations that can be separated into a pair of integrals

autonomous differential equation - a system of ordinary differential equations which does not depend on the independent variable

homogeneous linear differential equation - a differential equation is said to be homogeneous if there is no isolated constant term in the equation, e.g., each term in a differential equation for y has y or some derivative of y in each term

IVP (initial value problem) or Cauchy problem - an ordinary differential equation together with specified value, called the initial condition, of the unknown function at a given point in the domain of the solution

general solution - a general solution of a differential equation is the set of all of its particular solutions, often expressed using constants, which could have any fixed value

model for the system - a set of differential equations describing the behaviour of a system

rate of change - an indicator showing the difference between parameters in a specific unit of time

Summary

In this note we present several simple ordinary differential equations modeling selected real-world phenomena met in physics, archeology, art forgeries, population dynamics, heat radiation, epidemiology and economics.

Niniejszy tekst jest wstępem do problematyki równań różniczkowych zwyczajnych. Zawiera definicje podstawowych pojęć potrzebnych do zrozumienia tematu. Pokazuje jak sprowadzić dany problem z życia codziennego do modelu matematycznego, co jest zilustrowane prostym, ciekawym przykładem. Tekst składa się z wybranych fragmentów, szerszego opracowania profesora Stanislawa Migórskiego.

Pre-reading questions

1. Do you know how to build a mathematical model?

2. Do you think that theoretical mathematical models can be applied to solve practical problems?

autonomous differential equation - a system of ordinary differential equations which does not depend on the independent variable

interval a set of real numbers with the property that any number that lies between two numbers in the set is also included in the set. For example, the set of all numbers x satisfying $0 \leq x \leq 1$ is an interval which contains 0 and 1, as well as all numbers between them. The set of positive numbers is also an interval

differentiable function a real function is said to be differentiable at a point if its derivative exists at that point

1 Introduction and preliminary terminology

This note is a preliminary version of a self-contained introduction to the modeling of selected real-life phenomena by simple ordinary differential equations.We restrict ourselves to a discussion on first order ordinary differential equations, leaving the details on models described by higher order equations and systems of first-order equations to the sequel of the present work.

The text was designed for computer science students who have a calculus back ground[1] and have taken prior physics courses (cf. [4]). It is our belief that computer science students should know how to model a selected problem, particularly in light of rapidly changing technologies. We begin with preliminary notations and defnitions.

By a **differential equation** we mean a mathematical equation involving an unknown function of one or several variables and its derivatives of various orders. The order of the differential equation is the order of the highest derivative of the unknown function involved in the equation. For instance, the equation

$$x' = f(t, x) \tag{1}$$

is called an **ordinary differential equation** (often abbreviated to ODE) **of first order**[2] in the normal form. It relates an **independent variable** t to an unknown function x and its first order derivative. The function f is given $f: I \times E \longrightarrow E$, where I is an **interval**[3] and E is a prescribed space and we are looking for a function $x: I \longrightarrow E$. Of course the equation (1) obliges the unknown function x to have some restrictions: it should be **differentiable** in a suitable sense and the compose function f(t, x(t)) should have a meaning. Classically, we admit the following

Definition 1 A function $x: I \longrightarrow E$ is called a solution of the equation (1) on the interval I, if $x \in C^1(I,E)$[4] and $x'(t) = f(t, x(t))$ for all $t \in I$.

Remark 2 We remark that if $f: I \times E \longrightarrow E$ is a continuous function, we may require that a solution $x: I \longrightarrow E$ is only differentiable. This follows from the observation that in this case the composition of continuous functions $f(t, \cdot)$ and $x(\cdot)$ is continuous and it is equal to $x'(\cdot)$. Hence x' is continuous on I and thus x is a solution.

If the right hand side of the equation (1) can be expressed as a product g(t)h(x) where g depends only on t and h depends only on x, then the differential equation (1) is called

[1] Some universities, the Jagiellonian among them, make linear algebra a prerequisite for differential equations. Many schools, especially engineering, only require calculus.

[2] The most general (implicit) form of an ordinary differential equation of the first-order is as follows: $F(t, x, x') = 0$ with a prescribed function F.

[3] The interval I may have a form [a, b], [a, b), (a, b], (a, b), $(-\infty, b]$ and $[a, +\infty)$.

[4] A function $x: I \to E$ is C^1, if x is differentiable on I and its derivative x' is a continuous function.

separable. If the right hand side f is independent of t, then the resulting differential equation is called **autonomous**. Every autonomous differential equation is separable.

When the function f in (1) is (affine) linear with respect to the second variable, the first order equation of the form $x' = \alpha(t)x + \beta(t)$ with prescribed functions α and β is called a nonhomogeneous linear differential equation. If $\beta = 0$, then the equation $x = \alpha(t)x$ with a prescribed function α is called a **homogeneous linear** differential equation.

Definition 3 Let $t_0 \in I$ and $x_0 \in E$. A problem in which we are looking for the unknown function $x: I \longrightarrow E$ of a differential equation (1) where the value of the unknown function at some point is known $x(t_0) = x_0$ [5] is called an initial value problem (in short IVP) or a **Cauchy**[6] **problem** for (1).

Analogously as above, we have

Definition 4 Let $t_0 \in I$ and $x_0 \in E$. We say that a function $x: I \longrightarrow E$ is a solution of the IVP

$$\begin{cases} x' = f(t, x) \\ x(t_0) = x_0 \end{cases}$$

if x is a solution to equation (1) on the interval I and it satisfies $x(t_0) = x_0$.

Sometimes if no initial condition is given, we call the family of all solutions to the differential equation (1) the **general** solution.

2 Modelling

"How do we translate a physical phenomenon into a set of equations which describes it?" – this is certainly one of the most difficult problems that scientists deal with in their everyday research. This problem is a difficult one since it is usually impossible to describe a phenomenon totally, so one often tries to reformulate a real-world problem as a mathematical one making certain simplyfying assumptions. As a result one usually describes the system approximately and adequately.

The problem of generating "good" equations is not an easy task. The set of equations one deals with is called a **model** for the system. In general, once we have built a set of equations, we compare the data generated by the equations with real data collected from the system (by measurement). When the two sets of data "agree" (or are "sufficiently" close), we gain confidence that the set of equations will lead to a good description of the real-world system. If a prediction from the equations leads to some conclusions which are by no means close to real-world future behavior, we should modify and "correct" the underlying equations. When creating a model, it is necessary to formulate the problem under consideration into questions that can be answered mathematically.

initial value problem (IVP)
an ordinary differential equation together with specified value, called the initial condition, of the unknown function at a given point in the domain of the solution

Cauchy problem
see IVP

general solution
a general solution of a differential equation is the set of all of its particular solutions, often expressed using constants which could have any fixed value. (Source: http://math.uww.edu/~mcfarlat/250dej.htm)

[5] This condition is called an initial condition.

[6] Augustin Louis Cauchy (1789–1857), a French mathematician.

separable differential equation
a class of equations that can be separated into a pair of integrals

homogeneous linear differential equation
a differential equation is said to be homogeneous if there is no isolated constant term in the equation, e.g., each term in a differential equation for y has y or some derivative of y in each term

ODE ordinary differential equation
- a relation that contains functions of only one independent variable, and one or more of its derivatives with respect to that variable

The following are **basic steps** in building a model.

(1) Clearly state the assumptions on which the model will be based. These assumptions should describe the relationships between the quantities to be studied.
(2) Completely describe the parameters and variables to be used in the model.
(3) Use the assumptions (from step (1)) to derive mathematical equations relating the parameters and variables (from step (2)).

A large number of laws of physics, chemistry, economics, medicine, etc. can be formulated as differential equations. They serve as models that describe the behavior of complex systems. The mathematical theory of differential equations first developed together with the sciences where the equations had originated and where the results found application. Whenever a mathematical model involves the **rate of change** of one variable with respect to another, a differential equation is apt to appear. As we will see in the following sections, diverse problems, sometimes originating in quite distinct scientific fields, may give rise to identical differential equations. Whenever this happens, the mathematical theory behind the equations can be viewed as a unifying principle behind diverse phenomena.

In the following sections we provide examples of mathematical models of several relatively simple phenomena which are described by ordinary differential equations of the form (1). We mainly concentrate on modeling issues. The modern theory of ordinary differential equations together with known techniques, methods and applications can be found in references [1] and [2].

3 Radioactive decay

It is known that for many radioactive materials, for small amounts of the substance and in a short time interval, the amount of disintegrated mass is proportional to the length of the time interval and the amount of radioactive mass. Assuming that the amount of the material changes in time continuously, find the dependence of radioactive mass on time.

Radioactive decay is the process in which an unstable atomic nucleus loses energy by emitting radiation in the form of particles or electromagnetic waves. We denote the time by t and the amount of radioactive substance by m(t). We will show that the mass m(t) is described by the following ordinary differential equation of the first order

$$m'(t) = -km(t). \qquad (2)$$

Let $\Delta t \neq 0$ be the length of a time interval. Then we can write

$$\underbrace{m(t+\Delta t) - m(t)}_{\text{amount of disintegrated mass}} \sim m(t)\,\Delta t.$$

We denote by k a positive constant being the proportionality coefficient which characterizes the radioactive material. Since the rate of change of m(t) with respect to time t is negative (the amount of mass decreases in time), we obtain $m(t+\Delta t) - m(t) = -km(t)\Delta t$ and

$$\frac{m(t+\Delta t) - m(t)}{\Delta t} = -km(t).$$

Since the amount of the material changes in time continuously[7], this leads to the equation

$$m'(t) = \lim_{\Delta t \to 0} \frac{m(t+\Delta t) - m(t)}{\Delta t} = -km(t).$$

which means that the rate of decay is proportional to the amount of the substance present. The mass m is the unknown function to be determinated. The equation

$$m'(t) = -km(t) \qquad (3)$$

is called the equation of **radioactive decay**. It is a linear homogeneous differential equation.

model for the system
A set of differential equations describing the behaviour of a system

rate of change
an indicator showing the difference between parameters in a specific unit of time

Assuming that the mass is positive, we can solve (3) as follows. We observe that the equality $\frac{m'(t)}{m(t)} = -k$ is equivalent to $\frac{d}{dt}(\ln m(t)) = -k$. Integrating the latter, we have $\ln m(t) = -kt + C_1$, where C_1 is a constant of integration. Hence,

$$m(t) = e^{-kt+C_1} = e^{-kt}e^{C_1} = Ce^{-kt}$$

with a positive constant C.

Furthermore, we remark[8] that every function $m(t) = Ce^{-kt}$ with $C \in R$ is a solution to the differential equation $m'(t) = -km(t)$[9]. The value of C is determined if the initial amount of radioactive substance is given. It can be found without difficulty[10] that if $m(t_0) = m_0$, m_0 denotes the initial amount of substance at time instant t_0, then the value of C can be found and the Cauchy problem

$$\begin{cases} m'(t) = km(t) \\ m(t_0) = m_0 \end{cases}$$

admits a unique solution of the form

$$m(t) = m_0 e^{-k(t-t_0)}. \qquad (4)$$

After taking into account the "real" constraints on the problem under consideration, this solution gives a formula for the amount of radioactive substance at any future time t.

---

[7] In fact, we assume that the derivative m'(t) exists at every time instant t.

[8] We cannot expect to get a unique solution, since there will be an arbitrary "constant of integration".

[9] So there are infinitely many solutions to this ODE.

[10] This is left to the reader.

Summing up, we have the implication

if a function m(·) describes the radioactive decay, then
the function m(·) satisfies the differential equation (3).

The converse does not hold because there are solutions which do not have a physical meaning. For example, $m(t) = -e^{-kt}$ is a solution which is not acceptable (mass can not be negative) or $m(t) = 10^{10^{10}} e^{-kt}$ solves the equation but the mass is too large to exist on Earth. In consequence, there are physical reasons which eliminate some solutions to the equation, i.e. for the solution to make sense we suppose that

$$0 \leq m_0 \leq m_{crit}$$

where m_{crit} denotes the critical mass of radioactive material. Then, we clearly get that $0 \leq m(t) \leq m_{crit}$ for all $t \geq t_0$.

Despite the physical aspects, the following question arises. Do any other solutions to the equation of radioactive decay exist? The answer is negative.

Remark 5 *All solutions to the equation $m'(t) = -km(t)$ have the form $m(t) = Ce^{-kt}$ with $C \in R$. There are no other solutions. Indeed, let ϕ be a solution to the equation. Consider the function $\psi(t) := \phi(t)e^{kt}$. Since ϕ solves the equation, we have*

$$\psi'(t) = \phi'(t)e^{kt} + k\phi(t)e^{kt} = e^{kt}(\phi'(t) + k\phi(t)) = 0.$$

Hence the function ψ is constant, say C_1, which means that $\phi(t)e^{kt} = C_1$. Finally, we get $\phi(t) = C_1e^{-kt}$.

We have seen that function (4) describes radioactive decay. Clearly, in order to determine m(t) we need to find the constant k. This can be done using what is called the half-life of the radioactive material.

Definition 6 *The half-life is the time span needed to disintegrate half of the radioactive substance, i.e. a time T is the half-life of the material if $m(T) = \frac{1}{2}m_0$.*

If T is the half-life, then from (4), we have

$$m(T) = m_0e^{-kT} = \frac{m_0}{2}$$

which implies $kT = \ln 2$. Hence, if we know T, we can get k and vice-versa. Inserting the value of k into (4), we have

$$m(t) = m_0e^{-\frac{t}{T}\ln 2}. \qquad (5)$$

Many chemistry textbooks contain the half-life of some important radioactive materials. As Table 1 shows, some radioactive materials disintegrate almost immediately, others need milions of years.

There are several applications of the decay of radioactive materials.

isotope	half-life
^{14}C carbon	5730 y
^{131}I iodine	8.02070 d
^{214}Po polonium	$1.64 \cdot 10^{-4}$ sec
^{226}Ra radium	1602 y
^{238}U uranium	$4.468 \cdot 10^{9}$ y

Table 1: Half-lives of isotopes

Example 7 *(Chernobyl disaster) One of the most dangerous radioisotopes released in the Chernobyl catastrophe in 1986 was Strontium ^{90}Sr whose half-life is 28 years. (a) How much time does it take to reduce the isotope fallout to 10% of its initial value? (b) Assuming that during the catastrophe the number of released atoms was 10^{27}, how much time does it take for all the atoms of the substance to disappear?*

Solution. Let $t_0 = 0$ and half-life $T = 28$. (a) We use formula (5). Since the half-life is given in years, we will measure time in years. The time t needed to reduce the isotope fallout to p% of its initial value is calculated from the equality

$$\frac{m(t)}{m_0}\,100\% = p\%.$$

We obtain

$$e^{-\frac{t}{T}\ln 2} = \frac{p}{100}\,;\; e^{\frac{t}{T}\ln 2} = \frac{100}{p}\,;$$

and finally

$$t = T\,\frac{\ln\frac{100}{p}}{\ln 2}. \qquad (6)$$

Inserting the value $p = 10$, we have $t \approx 93.04$ years[11].

(b) Let $N = 10^{-27}$. Again we use formula (5) and calculate, as in (6), the time for all the atoms of the radioisotope to disappear

$$\frac{m(t)}{m_0}\,100 = N \text{ and hence } t = T\,\frac{\ln\frac{100}{N}}{\ln 2}.$$

An easy calculation yields $t = 28\,\frac{\ln 10^{29}}{\ln 2} = 28\cdot 29\,\frac{\ln 10}{\ln 2} \approx 2698$ years.

Example 8 *(Radiocarbon dating in archeological research) Carbon dating is often used to determine the age of a fossil. For instance, a humanoid skull was found in a cave*

[11] In these and other computations in this note, we round our answers to two decimal places.

in South Africa along with the remains of a campfire. Archeologists believe the age of the skull to be the same as the campfire. It is determined that only 2% of the original amount of Carbon ^{14}C remains in the burnt wood of the campfire. Estimate the age of the skull if the half-life of Carbon ^{14}C is about 5730 years.

Solution. Denote by p the percent of mass remaining. We use formula (6) to estimate the age of the skull. If the half-life $T = 5730$ and $p = 2$, then

$$t = T\,\frac{\ln\frac{100}{p}}{\ln 2} = 5730\,\frac{\ln 50}{\ln 2} = 5730\,\frac{3.9120}{0.6931} \approx 31325 \text{ years.}$$

To see how sensitive the technique of carbon dating is, we can change the values of the two parameters of half-life T and percent p of mass remaining. For example, fix $p = 2\%$ and calculate the estimated age of the skull for $T = 5700$, 5650, 5600 and 5550. We obtain from (6) the following values of the age $t = 32171$, 31890, 31607 and 31325 respectively. A similar sensitivity analysis can be carried out with respect to the changes in the parameter p. Precision is very dependent on the care with which the radiometric dating procedure is performed.

Remark 9 *(K-Ar dating in archeology) Potassium-argon (or K-Ar dating) is based on measuring the products of the radioactive decay of potassium (K). The radioactive isotope ^{40}K decays to ^{40}Ar with a half-life of $1.26 \cdot 10^9$ years. Due to the long half-life, the technique is most applicable for dating minerals and rocks more than 100 000 years old. It plays an important role in archeology. In 1959 in the Olduvai Gorge in the Serengeti plains of northern Tanzania, archeologists Mary and Louis Leakey uncovered the tools and fossils of an ancient humanoid called Zinjanthropus. According to the K-Ar dating method, there was a 1.75 million-year-old skull. In 1965 their team uncovered a Homo erectus skull, dated at one million years old. It was in 1978 at the Laetoli site, located 45 km south of Olduvai gorge, that Mary discovered the footprint-bearing layers, dated by the K-Ar method to 3.75 million years ago. The stratigraphy of the Olduvai Gorge is extremely deep and layers of volcanic ashes and stones allow radiometric dating of the embedded artifacts, mostly through K-Ar dating. The archeologists have been bracketing the age of archeological deposits at Olduvai Gorge by dating lava flows above and below the deposits. The Olduvai Gorge is now commonly referred to as "The Cradle of Mankind".*

By the way, modern geologists consider the age of the Earth to be around 4.54 billion years ($4.54 \cdot 10^9$ years). This age has been determined by radiometric age dating of meteorite material and is consistent with the ages of the oldest-known terrestrial and lunar samples. The uranium-lead radiometric dating scheme is one of the oldest available, as well as one of the most highly respected. It has been refined to the point where the error in dates of rocks about three billion years old is no more than two million years.

Example 10 *Using the data in Table 1, find the percentage of the initial amount of Carbon ^{14}C which remains after 10 000 years.*

Solution. By Table 1 we know that the half-life of the isotope T = 5730 years. Using formula (5), we have

$$\frac{m(10000)}{m_0}\,100\% = \exp\left(-\frac{10000}{5730}\ln 2\right) \approx 29.83\ \%$$

Example 11 *A radioactive isotope has a half-life of 14 days. You wish to have 50 g at the end of 30 days. How much radioisotope should you start with?*

Solution. Since the half-life T = 14 days, we will measure time in days. Let m(t) be the amount present at time t and m_0 the initial amount we are looking for. From (5) we obtain

$$m_0 = e^{\frac{t}{T}\ln 2}.$$

Putting t = 30, m(t) = 50, we find $m_0 = 50e^{\frac{3}{14}\ln 2} \approx 220.8$ g.

Remark 12 *(Turin shroud and carbon dating) The Shroud of Turin is a linen cloth bearing the hidden image of a man who appears to have been physically traumatized in a manner consistent with crucifixion. It is kept in the royal chapel of the Cathedral of Saint John the Baptist in Turin, Italy.*

Various tests have been performed on the shroud and the debates about its origin continue. Radiocarbon dating in 1988 by three independent teams of scientists produced consistent results indicating that the analysed portion of the shroud dated from the 13th to 14th centuries (1260–1390). This research has raised doubts about the carbon dating method. One argument against the results of the radiocarbon tests was a bacterial contamination, which was unknown during the 1988 testing. There are examples of ancient textiles that have been grossly misdated, especially in the earliest days of radiocarbon testing (the father of modern ^{14}C testing was Harry Gove). Most notable of these is mummy 1770 of the British Museum, whose bones were dated some 800 to 1000 years earlier than its cloth wrappings. The skewed results were thought to be caused by organic contaminants on the wrappings similar to those proposed for the shroud. Also, the result of the ^{14}C dating of the Turin shroud was probably caused by the effects of high temperature in the 1532 fire (and another series of small burns).

A follow-up analysis published in 2005, however, indicated that the sample dated by the teams was taken from an area of the shroud that was not a part of the original cloth.

Remark 13 *(Art forgeries) During World War II, many paintings of famous artists were lost. Several works of art were found in strange locations just after the war. Some of*

the paintings discovered were actually art forgeries and scientists often had difficulty proving that a painting was authentic. Using modeling based on a differential equation, a method was discovered to distinguish between a painting dating back 300 years and one painted only 20 years ago. It turns out that a small amount of lead-210 is present in all paintings. This element is found in lead oxide (commonly known as white lead), which is a pigment artists have used for more than 2000 years. White lead is made from lead metal which in turn is extracted from lead ore in a process called smelting. An analysis based on a first-order differential equation (where the unknown function is the amount of lead-210 per gram of white lead) makes it possible to estimate the age of a work of art and check whether a painting is a modern forgery. It was also observed that lead-210 is a difficult element to measure and that polonium-210, a much easier element to measure, has the same half-life as lead-210. Therefore a substitution of the two can be made in the analysis.

Remark 14 *(Phosphorus) Six radioactive isotopes of phosphorus are known. One radioactive isotope, ^{32}P, has applications in medicine, industry, and tracer studies. An isotopic tracer (called also an isotopic marker or an isotopic label) is a radioactive isotope whose presence in a system can easily be detected. The isotope is injected into the system, where it gives off radiation. The radiation is followed by means of detectors placed around the system. Phosphorus-32 is especially useful in medical studies, because phosphorus occurs in many parts of the body. Radioactive phosphorus can be used as a tracer to study parts of the body as well as chemical changes inside the body. Radioactive phosphorus can also determine how much blood is in a person's body. It can also help locate the presence of tumors in the brain, eyes, breasts, and skin. Finally, it is sometimes used to treat certain forms of cancer. Radiation given off by the ^{32}P may kill cancer cells and help slow or stop the disease. This isotope is important in a variety of scientific studies. For example, it is added to tires when they are made. Then, the radiation it gives off can be studied as the tires are used. This information indicates where the tire wears out and how long it takes to wear out.*

4 Population dynamics

Find the dependence of the population on time assuming that the growth rate is proportional to the population present.

Let p(t) be the population at time t. We may think about a single population of a species contained in a compartment (a petri dish, an island, a country). The population is always a natural number, it is usually large enough so that very little error is introduced in assuming that p(t) is a continuous function.

Consider a population of bacteria that reproduce by simple cell division, i.e. the parent cell does not die, but becomes two new cells (the death rate is zero). In this model

assumption that the growth rate is proportional to the existing population is consistent with observations of bacteria growth. The initial value problem is

$$\begin{cases} \frac{dp}{dt} = kp \\ p(0) = p_0, \end{cases} \tag{7}$$

where k is the proportionality constant (called population growth constant) and $p_0 > 0$ is the population at time 0[12]. This model is called the **Malthusian**[13] or **exponential law** of population growth. The equation in (7) is linear and separable, and the solution of (7) is given by

$$p(t) = p_0 e^{kt}. \tag{8}$$

When we consider the initial condition of the form $p(t_0) = p_0$, the solution to the above model is of the form

$$p(t) = p_0 e^{k(t-t_0)}. \tag{9}$$

Example 15 *(Population of the world) According to the 2007 UNFPA[14] Report, the number of people of the world was: 1800 - $1.0 \cdot 10^9$, 1960 - $3.0 \cdot 10^9$ and 2007 - $6.5 \cdot 10^9$. Use the exponential law of population growth to estimate the population in 2050.*

Solution. We will use formula (9). First, we estimate the population constant k and subsequently we estimate the size of the human population. From (9) we easily deduce

$$k = \frac{\ln \frac{p(t)}{p_0}}{t - t_0}$$

The estimation of k can be done in the following two ways.

1^0) Let $t_0 = 0$, $p_0 = 1.0 \cdot 10^9$, $t = 1960 - 1800 = 160$ and $p(160) = 3.0 \cdot 10^9$. Then

$$k = \frac{\ln \frac{p(t)}{p_0}}{t - t_0} = \frac{\ln 3}{160} \approx 0.006866$$

which means that in 1800–1960 the population increased by 0.6866% per year.

2^0) Let $t_0 = 0$, $p_0 = 3.0 \cdot 10^9$, $t = 2007 - 1960 = 47$ and $p(47) = 6.5 \cdot 10^9$. Then

$$k = \frac{\ln \frac{p(t)}{p_0}}{t - t_0} = \frac{\ln 6{,}53 - \ln 3.0}{47} \approx 0.01645$$

---

[12] Analogously, we can treat the problem with $p(t_0) = p_0$.

[13] Thomas Robert Malthus (1766–1834), the English demographer and political economist.

[14] United Nations Fund for Population Activities.

which means that in 1960–2007 the population increased by 1.645% per year.

From the form of solution (9), we can estimate the world population in 2007 and 2050 using different population constants. For $k = 0.006866$, we have

$$p(2007) = 3.0 \cdot 10^9 \cdot e^{0.006866(2007-1960)} \approx 4.16 \cdot 10^9$$

while for $k = 0.01645$, we get

$$p(2050) = 6.5 \cdot 10^9 \cdot e^{0.01645(2005-2007)} \approx 13.18 \cdot 10^9.$$

These results are estimated by the Malthusian model only and do not take into account many important factors for the human population[15].

Remark 16 *There are populations (like the human one) for which the assumption that the death rate is zero is certainly wrong. However, we may revise the Malthusian model to incorporate only death by natural causes. It might be expected that the death rate is also proportional to the size of the population. This leads to the equation*

$$\frac{dp}{dt} = kp - k_d p = (k - k_d)p = \widetilde{k}\, p \qquad (10)$$

where $\widetilde{k} = k - k_d$ and k_d is the proportionality constant for the death rate. Assuming $k > k_d$, so that $\widetilde{k} > 0$, we again obtain the Malthusian model.

Bearing in mind solution (8) of the exponential model, we conclude the following.

If $k > 0$, then the population grows and continues to expand to infinity, that is,

$$\lim_{t \to +\infty} p(t) = +\infty$$

If $k < 0$, then the population will shrink and tend to 0. In other words we would be facing extinction. The first case, $k > 0$, is not adequate and the model can be dropped. The main argument for this has to do with environmental limitations. The complication is that population growth is eventually limited by some factor, usually one from among many essential resources. When a population is far from its limits of growth it can grow exponentially. However, when nearing its limits, population size can fluctuate, even chaotically.

Another model was proposed to remedy this flaw in the exponential model. It is called the logistic model (also called Verhulst-Pearl model[16]) and it takes into account factors (like premature deaths due to malnutrition, inadequate medical supplies, wars, communicable diseases, violent crimes, etc.) involving competition within the population. In order to justify the logistic model, we assume there is another component

---

[15] According to the UNFPA the population estimate for 2050 is 8.9 bilion.

[16] P.F. Verhulst (1804–1849), a mathematician and a doctor in number theory from the University of Ghent.

of the death rate that is proportional to the number of two-party interactions. For population of size p, there are p(p - 1)/2 such interactions. We may suppose the death rate due to intraspecies competition is

$$k_c \frac{p(p-1)}{2}$$

where k_c is another proportionality constant (the competition constant). Together with equation (10), this leads to

$$\frac{dp}{dt} = kp - k_d p + k_c \frac{p(p-1)}{2} = kp - k_d p + k_c \frac{p}{2} - k_c \frac{p^2}{2}$$

The logistic model is the following initial value problem

$$\begin{cases} \frac{dp}{dt} = ap - bp^2 \\ p(0) = p_0 \end{cases}$$

where $a = k\text{-}k_d + \frac{k_c}{2}$ and $b = \frac{k_c}{2}$. The equation is autonomous (hence separable) and nonlinear because of the presence of p^2.

We observe that the equation $dp/dt = ap - bp^2$ has two constant solutions $p \equiv 0$ and $p \equiv a/b$. The non-constant solutions may obtained by separating the variables and using a partial fractions expansion:

$$\int \frac{1}{p(a - bp)}\, dp = \int dt \qquad (11)$$

$$\frac{1}{a}\int \frac{dp}{p} - \frac{1}{a}\int \frac{-b}{a-bp}\, dp = \int dt$$

$$\frac{1}{a}\ln|p| - \frac{1}{a}\ln|a - bp| = t + C_1$$

$$\ln\left|\frac{p}{a - bp}\right| = at + C_2$$

$$\frac{p}{a - bp} = C_3 e^{at}.$$

Solving for p, we have

$$p(t) = \frac{aC_3 e^{at}}{bC_3 e^{at} + 1} = \frac{aC_3}{bC_3 + e^{-at}}$$

Using the initial condition in (11) (assuming that p_0 is not equal to both 0 or a/b), we get

$$C_3 = \frac{p_0}{a - bp_0}$$

which, once substituted into the expression for p(t) and simplified, we find

$$p(t) = \frac{ap_0}{bp_0 + (a - bp_0)e^{-at}} \qquad (12)$$

This function is called the **logistic function** and its graph is called the **logistic curve**. An important property of the logistic function is that

$$\lim_{t \to +\infty} p(t) = \frac{a}{b} \quad \text{if } p_0 > 0.$$

The quantity a/b is a limiting size for the population (also called the **carrying capacity**). Consequently, any population which follows the logistic model will remain bounded. Namely, when $p_0 > a/b$, the population will decrease toward a/b and when the initial population $p_0 < a/b$, it will increase toward a/b. However, this is still not satisfactory because this model does not tell us when a population is facing extinction since it never implies that. Even starting with a small population it will always tend to the carrying capacity a/b.

In deriving the solution given by (12), we excluded the cases $p_0 = 0$ or a/b. These two populations are referred to as **equilibrium populations**. Notice that the graphs of equilibrium populations are the horizontal asymptotes of the logistic curves[17].

In population dynamics there are natural questions to be answered.

(1) What will the population of a certain country be in several years?

(2) How are we protecting the resources from extinction?

More can be said about the population dynamics problem but in this little review we will not discuss them in detail.

Newton's law of cooling

It is known from experimental observations that the surface temperature of an object changes at a rate proportional to its relative temperature. The latter is the difference between its temperature and the temperature of the surrounding environment. This is what is known as Newton's law of cooling. Find the dependence of the temperature of the object on time.

We denote by T(t) the temperature of the object at time instant t and by M the constant temperature[18] of the environment. The Newton law of cooling gives a differential equation of the first order

$$\frac{dT}{dt} = -k(T - M) \qquad (13)$$

where $k > 0$. The proportionality constant in (13) is negative since the temperatures

[17] The interested reader can sketch the graphs of the logistic curve for $0 < p_0 < a/b$ and $p_0 > a/b$.

[18] The constant M is called the medium temperature since it is the temperature of the medium an object is immersed in.

decrease in time and the derivative $\frac{dT}{dt}$ must be negative for $T > M$. The equation (13) is a separable ODE. We assume that the initial temperature of the object is prescribed

$$T(t_0) = T_0 \tag{14}$$

where T_0 stands for the initial temperature of the object. Thus, the dependence of the temperature of the object on time is described by the initial value problem (13) and (14). Its solution has a form[19]

$$T(t) = M + (T_0 - M)e^{-kt}. \tag{15}$$

Example 17 *(Hot coffee) A cup of coffee initially at 95^0C cools to 80^0C in 5 minutes while sitting in a room of temperature 21^0C [20]. Using the Newton law of cooling, determine when the temperature of the coffee will be a nice 50^0C.*

Solution Let $T_0 = 95$, $M = 21$, $t = 5$ and $T(5) = 80$. We proceed in two steps and make use of formula (15). First, we find the value of the coefficient k which can be calculated

$$k = \frac{1}{t} \ln \frac{T_0 - M}{T(t) - M} = \frac{\ln 74 - \ln 59}{5} \approx 0.045305.$$

Then, knowing the value of k, we are able to find a time instant t_1 at which the temperature of the coffee will be $T(t_1) = 50$.

$$t_1 = \frac{1}{k} \ln \frac{T_0 - M}{T(t_1) - M} = \frac{\ln 74 - \ln 29}{k} \approx 20.676.$$

Hence, the temperature of the coffee will be 50^0C after 20.7 minutes from the initial time instant.

Example 18 *(Time of death) It was noon on a cold May day in Krakow: 16^0C. Detective John Kowalski arrived at the crime scene in a hotel room to find the sergeant leaning over the body. The sergeant said there were several suspects. If they knew the exact time of death, then they could narrow the list. Kowalski took out a thermometer and measured the temperature of the corpse: 34.5^0C. He then left for lunch. Upon returning at 1:00pm, he found the body temperature to be 33.7^0C. Find the time the murder occured.*

Solution. As in Example 17, we proceed in two steps. Let $T_0 = 34.5$, $M = 16$, $t = 60$ and $T(60) = 33.7$. First, we find the value of the coefficient k as follows

$$k = \frac{1}{t} \ln \frac{T_0 - M}{T(t) - M} = \frac{\ln 8.5 - \ln 17.7}{60} \approx 7.3677 \cdot 10^{-4}.$$

[19] This can easily be checked by the reader.

[20] *This is not always attained in the Jagiellonian University buildings.*

In other words, we have determined k from the information on two temperatures measured by Kowalski, at the point $T_0 = 34.5$ and the point $T(60) = 33.7$.

Knowing the value of k, we are able to find a time instant t_d at which the crime happened. To this end we take into account other two points $T(0) = 37$[21] and $T(t_d) = 34.5$.

$$t_d = \frac{1}{k} \ln \frac{T(0)-M}{T(t_d)-M} = \frac{\ln (37-16)-\ln (34{,}5-16)}{k} = \frac{\ln 21-\ln 18{,}5}{k} \approx 172.036.$$

Therefore the murder occured about 172 minutes before noon, that is, around 9:08am. We conclude that in order to find the time of death it is necessary to measure the body temperature twice[22].

Remark 19 *(Heating and cooling of buildings) Newton's law can be used to formulate a model that describes the temperature profile inside a building as a function of the ouside temperature, the heat generated inside the building, and the furnace heating or air conditioning cooling. The resulting Cauchy problem can be formulated as in Chapter 3.3 of [5] and reads as follows*

$$\begin{cases} \frac{dT}{dt} = -k(T(t) - M(t)) + H(t) + U(t) \\ T(t_0) = T_0. \end{cases} \qquad (16)$$

The unknown function T(t) represents the temperature inside the building at time t and, viewing the building as a single compartment, M(t) is the outside temperature, H(t) is the additional heating rate (in terms of energy per unit time), U(t) is the furnace heating or air conditioner cooling rate, k is a positive constant and T_0 is the initial temperature at time t_0.

This linear model (16) is a generalization of the problem (13) and (14). The constant $\frac{1}{k}$ is called the time constant for the building and it is one of the characteristics of buildings[23]. More details on this model can be found in [5].

6 Stefan's law of thermal radiation

The Stefan law of thermal radiation states that the rate of change in the temperature of a body at T(t) degrees in a medium at temperature M degrees is proportional to the quantity $M^4 - T^4$.

[21] Assuming the dead person was not sick and had a temperature of 37^0C. The interested reader may estimate the time of death, if instead of 37^0C we suppose 36.6^0C.

[22] Remember this method unless you are sergeant Colombo.

[23] The building owners should know this constant for economic reasons.

As before, we denote by T(t) the temperature of the body at time instant t and by M the constant temperature of the surrounding medium. We obtain the following nonlinear differential equation of the first order

$$\frac{dT}{dt} = -k(T^4 - M^4), \qquad (17)$$

where $k > 0$. Equation (17) is a separable ODE. Stefan's and Newton's laws are nearly the same when T is close to M, cf. Chapter 3 of [5].

7 An epidemiology model

The spread of diseases can be assessed by using differential equations. A simple model for the number of people infected with a particular disease, say chicken pox, in a closed environment, like a university, is as follows.

$$I'(t) = rI(t) - aI(t) \qquad (18)$$

where:

- t is time measured in days;
- I(t) is the number of infected people with the disease at time t;
- $r > 0$ is a parameter measuring the rate at which people are infected by an infectious person (how many people an infected person infects per day);
- $a > 0$ is a parameter measuring the rate of removal, that is, the rate at which people cease to be infectious; they either recover or die (1/the number of days a person is infectious)

Of course this equation can be written as $I'(t) = (r - a)I(t)$ and therefore $r - a$ can be considered as a parameter of this equation. The general solution to (18) is an exponential function

$$I(t) = Ce^{(r-a)t} \text{ with } C \in R.$$

However, the negative constant C is not acceptable in the solution since the function I, the infected population, can never be negative. As can be seen from the previous examples, the character of this solution depends on the parameter r-a. The infection rate r will vary based on several factors, including the nature of the disease and the nature of contact between people. The removal rate a also depends on several factors, including how fast people are diagnosed (so they can be told to stay at home), and how quickly they get better (so they are no longer infectious). Interestingly, another way people can be "removed" from the infectious population is to die.

Consider now an initial condition. At time $t = 0$, we start with a certain number of sick people, I_0. We may think of $t = 0$ as the start of time that we get involved, t is negative in the past, and t is positive in the future. The solution now looks as follows:

$$I(t) = I_0 e^{(r-a)t}.$$

If $I_0 = 0$, then I(t) is always zero for all time t. It is easy to see that if $r > a$, we have growth, and if $r < a$, we have a reduction. It does not matter to the equation whether people are dying or getting better, if they are not spreading the disease for long, the removal rate a is high.

The disease is called an epidemic if $\frac{dI}{dt} > 0$. This happens if and only if $r > a$. The specific changes in r and a could prevent an epidemic. This simple model helps to predict the number of infected people I(t).

8 Free fall

An object falls through the air toward Earth. Assuming that the only forces acting on the object are gravity and air resistance, find the velocity of the object as a function of time.

Newton's second law states that force is equal to mass times acceleration, i.e.

$$F = m\frac{dv}{dt},$$

where F represents the total force on the object, m is the mass of the object and the acceleration is expressed as the derivative of the velocity v with respect to time t. Near the Earth's surface, the force due to gravity (being just the weight of the object) is given by mg, where g is the acceleration due to gravity[24]. On the other hand, there is no general law that precisely models the air resistance acting on the object. The air resistence is a force that can be represented by -kv, which means that it is proportional to the velocity of the object and it opposes the motion. The positive constant k depends on the density of air and shape of the object. Therefore, the total force acting on the object is mg - kv. This leads to the following ordinary differential equation:

$$m\frac{dv}{dt} = mg - kv. \qquad (19)$$

This equation is linear, homogeneous and autonomous, so it is separable. Exploiting the separation of variables[25] we solve this equation and find

$$v(t) = \frac{mg}{k} - \frac{C}{k}e^{-kt/m} \text{ with } C \in \mathbb{R}. \qquad (20)$$

This is a general solution to equation (19) and every solution to (19) is of the form (20). Assuming that the initial velocity of the object is v_0, we obtain the following initial value problem

[24] Recall that on Earth $g = 9.81 m/s^2$.

[25] This was already used in the radioactive model.

$$\begin{cases} m\dfrac{dv}{dt} = mg - kv \\ v(0) = v_0. \end{cases} \tag{21}$$

Substituting $t = 0$ in the general solution and using the initial datum, easy algebraic manipulations give $C = mg - kv_0$. The solution of (21) is the following:

$$v(t) = \frac{mg}{k} + \left(v_0 - \frac{mg}{k}\right) e^{-kt/m} \tag{22}$$

We make certain observations. Since

$$\lim_{t \to +\infty} v(t) = \frac{mg}{k},$$

it appears that, regardless of the initial velocity v_0, the velocity of the object approaches mg/k. The value mg/k is the horizontal asymptote for v(t). The constant mg/k is referred to as the **limiting** or **terminal velocity** of the object and it is approximately the weight, mg, divided by the coefficient of air resistance, k. Hence, the heavier the object, the faster it will fall, assuming shapes and sizes are the same. Also, when the air resistance is reduced (k is made smaller), the object will fall faster. These observations agree with our experience.

Example 20 *(Bungee jumper) The jumper jumps off a very high bridge with a long elastic rope tied to his legs, so that the rope pulls him back before he hits the ground. Assuming only air resistance and gravity acting on the jumper, find his terminal velocity just before the rope pulls him back, provided m = 100kg, k = 5kg/s and v(0) = 10m/s.*

Solution. The terminal velocity of the jumper is independent of his initial velocity and is equal to

$$\frac{mg}{k} = \frac{100\ \text{kg} \cdot 9.81 \frac{\text{m}}{\text{s}^2}}{5 \frac{\text{kg}}{\text{s}}} = 196.2\ \frac{\text{m}}{\text{s}}.$$

The model discussed above can be extended to determine the equation of motion of falling object.

This can be done as follows. Let us denote by s(t) the distance the object has fallen in time t. The position s(t) is found by integrating the equation

$$\frac{ds(t)}{dt} = v(t).$$

From (22) we obtain

$$s(t) = \int v(t)\, dt = \frac{mg}{k} t - \frac{m}{k}\left(v_0 - \frac{mg}{k}\right) e^{-kt/m} + C$$

where C is a constant. Setting $s(0) = 0$, we find that

$$C = \frac{m}{k}\left(v_0 - \frac{mg}{k}\right).$$

Hence the equation of motion is

$$s(t) = \frac{mg}{k}t + \frac{m}{k}\left(v_0 - \frac{mg}{k}\right)\left(1 - e^{-kt/m}\right).$$

Example 21 *(Bungee jumper, continued) Consider the bungee jumper from Example 20. Suppose additionally that the bridge is 500m above the ground. Determine when the jumper will strike the ground if the rope is broken.*

Solution. We can use (23). Because the jumper is released 500 m above the ground,we determine when he strikes the ground by setting $s(t) = 500$ and solving the following equation for t

$$s(t) = \frac{mg}{k}t + \frac{m}{k}\left(v_0 - \frac{mg}{k}\right)\left(1 - e^{-kt/m}\right).$$

Unfortunately, this equation can not be solved explicitely for t. In general we might try to find t by using, for instance, the Newton approximation method. However, in this case it is not necessary since the term e^{-t} will be very small for t near the time we are looking for. We simply ignore the term e^{-t} and obtain a linear equation

$$s(t) = \frac{mg}{k}t + \frac{m}{k}\left(v_0 - \frac{mg}{k}\right)$$

which we easily solve for t:

$$t = \frac{ks(t)}{mg} - \frac{v_0}{g} + \frac{m}{k} = 2.54s - 1.01s + 20s = 21.52s.$$

This means that the jumper will reach the ground within 21.52 s. Ignoring the term e^{-t} we made a reasonable approximation since e^{-t} near $t = 21.52$ is very small ($e^{-21.52} \approx 0.45 \cdot 10^{-9}$).

Example 22 *(Parachutist) A parachutist whose mass is 75 kg drops from a helicopter hovering 4000 m above the earth and falls toward the ground under the influence of gravity. Assume the gravitational force is constant and the force due to air resistance is proportional to the velocity of the parachutist, with the proportionality constant $k_1 = 15\frac{kg}{s}$ when the chute is closed and with constant $k_2 = 105\frac{kg}{s}$ when the chute is open. If the chute does not open until 1 minute after the parachutist leaves the helicopter, after how many seconds will he hit the ground?*

Solution. We use equations (22) and (23). We first consider the motion before the chute opens and then when the chute is open. We denote $m = 75$ kg, $h = 4000$ m, $g = 9.81\frac{m}{s^2}$, $t = 60$ s and $v = 0$.

Phase 1: the chute is closed. From equations (22) and (23), we find the velocity

$$v_1(t) = \frac{mg}{k} + \left(v_0 - \frac{mg}{k}\right)e^{-k_1 t/m} = \frac{75 \cdot 9.81}{15} - \frac{75 \cdot 9.81}{15}\exp\left(\frac{15 \cdot 60}{75}\right) = 49.5\frac{m}{s}$$

and the distance the parachutist has fallen in t seconds

$$s_1(t)=\frac{mg}{k_1}t+\frac{m}{k_1}\left(v_0-\frac{mg}{k_1}\right)(1-e^{-k_1t/m})=$$

$$=\frac{75\cdot 9.81\cdot 60}{15}-\frac{75}{15}\cdot\frac{75\cdot 9.81}{15}\cdot(1-e^{-12})=2697.75\text{m}.$$

Hence, after t = 60 s the parachutist is $s_2 = h - s_1 = 4000 - 2697.75 = 1302.25$ m above the ground traveling at a velocity of $49.05\frac{m}{s}$

Phase 2: the chute is open. In order to determine when the parachutist will hit the ground, we employ the equation

$$s_2(t)=\frac{mg}{k_2}t+\frac{m}{k_2}\left(\overline{v}_0-\frac{mg}{k_2}\right)(1-e^{-k_2t/m}),$$

where $s_2 = 1302.25$ m is the distance to the ground, and v_0 is the initial velocity at the time the chute opens. We would like to solve this equation for t. As in Example 21, this nonlinear equation cannot be solved explicitely for t. However, since the term $e^{-k_2t/m}$ will be very small for t near the time we are looking for, we ignore this term and obtain a linear equation

$$s_2(t)=\frac{mg}{k_2}t+\frac{m}{k_2}\left(\overline{v}_0-\frac{mg}{k_2}\right)$$

Solving the latter for t, we have

$$t=\frac{k_2s_2}{mg}-\frac{\overline{v}_0}{g}+\frac{m}{k_2}=185.84-5.04+0.71=181.51\text{s}.$$

This means that the parachutist will strike the earth 181.51 s after the parachute opens or 241.51 s after dropping from the helicopter. The ignored term $e^{-k_2t/m}$ near t = 181.51 is very small ($\approx e^{-254.2}$) hence negligible and thus the approximation we have made is reasonable.

We also remark that the velocity of the parachutist at impact is

$$\frac{mg}{k_2}=\frac{75\cdot 9.81}{105}=7.01\frac{m}{s}$$

which is the limiting velocity for his fall with the chute open.

9 Continuous compounding

A bank used to compound interest continuously. Show that the amount P(t) of dollars in a savings bank account at time instant t satisfies the ordinary differential equation

$$\frac{dP}{dt}=\frac{r}{100}P$$

where r denotes the nominal annual interest rate and t is the total time in years.

Compound interest is the concept of adding accumulated interest back to the principal, so that interest is earned on interest from that moment on. The effect of compounding depends on the frequency with which interest is compounded and the periodic interest rate which is applied.

When computers were scarce, banks used to compound interest periodically, often quarterly. That meant that four times a year they would have an "interest day", when everybody's balance got bumped up by one fourth of the going interest rate. If you held an account in those days, every year your balance would increase by a factor of $(1 + r/4)^4$, where r is the nominal annual interest rate. Today it is possible to compound interest monthly, daily, and in the limiting case, continuously, meaning that your balance grows by a small amount every time instant.

To find the formula we will start out with interest compounded n times per year. Let $P_0 = P(0)$ be an initial amount in a savings bank account. The future value after t years is

$$P_0\left(1+\frac{r}{100n}\right)^{nt}.$$

Note that the total number of compounding periods is nt. Continuous compounding can be thought as making the compounding period infinitely small, therefore achieved by taking the limit of n to infinity. Thus, continuous compounding is the limit as the compounding period approaches zero. We have

$$P(t) = \lim_{n\to+\infty} P_0\left(1+\frac{r}{100n}\right)^{nt} = P_0 \lim_{n\to+\infty}\left[\left(1+\frac{r}{100n}\right)^{n/r}\right]^{rt/100} = P_0 e^{0.01rt}$$

The number e is the base of the natural logarithm (the Euler[26] number) and the limit of the following sequence

$$e = \lim_{n\to+\infty}\left(1+\frac{1}{n}\right)^n \approx 2.7182818284...$$

We conclude that as n increases, the rate approaches an upper limit of e^r. This rate is called **continuous compounding**. Since $P'(t) = P_0 \frac{r}{100} e^{0.01rt} = \frac{r}{100} P(t)$, the function P(t) is the solution of the desired differential equation.

Example 23 *(The Rule of 72 for compound interest) The continuous investment of of money can be described by the ordinary differential equation $P'(t) = 0.01\ r\ P(t)$, where P(t) is the amount of savings at time t and r is the nominal annual interest rate. Investors often use a method called "the rule of 72" which says that a number of years needed to retain an investment before it doubles in value with the annual interest rate r, is equal to 72/r. Show that the above rule of 72 merely gives an approximation of the time needed and find the precise time.*

[26] L. Euler (1707–1783), a Swiss mathematician and physicist.

Solution. The solution of the differential equation is of the form $P(t) = P(0)e^{0.01rt}$.The condition when the investment doubles in value at time t is written as $P(t) = 2P(0)$. Hence,

$$2P(0) = P(0)e^{0.01rt}$$

which implies $\ln 2 = 0.01rt$ and

$$t = \frac{100 \ln 2}{r} = \frac{69.315}{r}.$$

Therefore, according to the mathematical model, the rule of 72 should be replaced by the rule of 69.315.

Example 24 *(Compound interest) A bank pays a yearly interest rate of 5% compounded continuously, the initial amount is P(0) = 1000 PLN* [27] *and no monies are withdrawn. How much will be in the account after 2 years? When will the account reach 4000 PLN?*

Solution. The solution of the differential equation is of the form $P(t) = P(0)e^{0.01rt}$. For $r = 5$, $P(0) = 1000$ and $t = 2$, we have

$$P(2) = 1000e^{0.01 \cdot 5 \cdot 2} = 1000e^{0.1} = 1105.2.$$

So, after 2 years there will be 1105.2 PLN in the account. To answer the second question, from the equality $P(t) = P(0)e^{0.01rt}$ we calculate time t and get

$$t = \frac{100 \ln \frac{P(t)}{P(0)}}{r} = \frac{100 \ln \frac{4000}{1000}}{5} = 20 \ln 4 \approx 27.73.$$

The account will reach 4000 PLN after 27.73 years.

10 Escape velocity

Consider a projectile of constant mass m being fired vertically from Earth. Find a differential equation that governs the motion of the projectile under the Earth gravitational force.

We use Newton's law of gravitation which says that the attractive force between two objects varies inversely as the square of the distance between them, i.e.

$$F_g = G \frac{m_1 m_2}{r^2},$$

where m_1 and m_2 are the masses of the objects, r is the distance between them (center to center), G is the constant of proportionality and F_g is the attractive force.

Denote by r the distance between the projectile and the center of Earth, R is the radius of Earth and M is the mass of Earth. Let t represent time and m, v the mass and the velocity of the projectile respectively. From the equation $F = -F_g$, where

[27] Polish zloty, PLN, is still the currency of Poland.

$$F = m\frac{dv}{dt} \text{ and } F_g = G\frac{m\,M}{r^2}$$

we have

$$\frac{dv}{dt} = -\frac{G\,M}{r^2}$$

Introducing the acceleration due to gravity for the Earth $g = \frac{G\,M}{R^2}$, we obtain that the motion of projectile is governed by the equation

$$\frac{dv}{dt} = -\frac{g\,R^2}{r^2}.$$

It is useful to find the velocity v as a function of the distance r of the projectile to the center of the Earth. Using the fact that $v = \frac{dr}{dt}$ from the above equation, we obtain

$$v\frac{dv}{dr} = -\frac{g\,R^2}{r^2}$$

Integrating the equation, we find

$$\int v\,dv = -gR^2\int\frac{dr}{r^2}.$$

so its general solution

$$v^2(r) = \frac{2g\,R^2}{r} + C \text{ with } C \in R.$$

When the projectile leaves the Earth's surface with velocity v_0, from the initial condition $v(R) = v_0$, we find $C = v_0^2 - 2gR$ and the solution

$$v^2(r) = \frac{2g\,R^2}{r} + v_0^2 - 2gR.$$

It is clear that the velocity of the projectile remains positive if and only if $C = v_0^2 - 2gR > 0$. The velocity

$$v_e = \sqrt{2gR}$$

is called the **escape velocity** of Earth. Assuming $g = 9.81\frac{m}{s^2}$ and R = 6370 km, we obtain the Earth escape velocity

$$v_e = \sqrt{2 \cdot 9.81 \cdot 637 \cdot 100^2} = 11179.31\frac{m}{s} = 11.18\frac{km}{s}.$$

Analogous calculations for the moon (the acceleration due to gravity for the moon is $g_m = g/6$ and the radius of the moon is $R_m = 1738$ km) give

$$v_e = \sqrt{2g_mR_m} = \sqrt{2 \cdot \frac{g}{6} \cdot R_m} = \sqrt{3.27} \cdot \sqrt{173.8} \cdot 100 = 2383.93\,\frac{m}{s} = 2.38\,\frac{km}{s}.$$

11 Atmospheric pressure

The atmospheric pressure on Earth can be modeled assuming the rate of its change with respect to the height above sea level is proportional to the pressure. Determine a differential equation that governs this phenomenon.

We denote the height above sea level by h and the atmospheric pressure by p(h). A reasoning analogous to that presented in Section 3 leads to the following Cauchy problem for an ordinary differential equation of the first order

$$\begin{cases} \frac{dp}{dh} = -kp \\ p(0) = p_0 \end{cases} \tag{24}$$

where k is a positive constant and p_0 is the pressure on sea level (where h = 0). The unique solution to (24) is the following function[28]

$$p(h) = p_0 e^{-kh}.$$

Example 25 *Suppose that at sea level the atmospheric pressure is 1013 mbar*[29] *and at an altitude of 20 km it is equal to 50 mbar. (a) Find the atmospheric pressure at an altitude of 50 km. (b) Find the altitude at which the pressure is 900 mbar.*

Solution. Let $p_0 = 1013$, $h = 20$ and $p(20) = 50$. We will be able to answer the questions after calculating the coeffcient k. From the equation $p(h) = p_0 e^{-kh}$, we find

$$k = \frac{\ln \frac{p_0}{p(h)}}{h} = \frac{\ln \frac{1013}{50}}{20} = 0.1504 \text{km}^{-1}.$$

(a) Having the value of the constant k, we have

$$p(50) = p_0 e^{-50 \cdot k} = 1013 \cdot e^{-0.1504 \cdot 50} = 0.5484$$

which means that the atmospheric pressure at an altitude of 50 km is equal to 0.55 mbar.

(b) The altitude can be easily found from the equation

$$h = \frac{\ln \frac{p_0}{p(h)}}{k} = \frac{\ln \frac{1013}{900}}{0.1504} = 0.4668.$$

The height above sea level at which the pressure is 900 mbar is about 467 m.

12 Other models

We list below a few other phenomena which can be modeled by differential equations of the first order. For more details, see [5].

Mixing problem. The problem of mixing the fluid in a tank is an example of a one-compartment system. Let x(t) denote the amount of a substance in a tank at time t. Since the derivative of x with respect to t can be interpreted as the rate of change in the amount of the substance in the tank, the problem is described by the following model

---

[28] Cf. also the solution to the equation of radioactive decay given by (4).

[29] Recall that 1 mbar (millibar) equals to 1 hPa (hectopascal).

$$\frac{dx}{dt} = \text{input rate - rate of exit.}$$

In mixing problems, one is often given the rate at which a fluid containing the substance flows into the tank, along with the concentration of the substance in that fluid. The input rate is the product of the flow rate by the concentration. The output rate of the substance is usually more difficult to determine. It is possible in many practical problems, cf. Section 3.2 of [5].

Aquaculture. Aquaculture is the art of cultivating plants and animals indigenous to water. For example, we can consider a batch of catfish raised in a pond. We are interested in finding the best time for harvesting the fish so that the cost per kilogram of raising the fish is minimized. The growth of fish can be modeled by a differential equation of the form

$$\frac{dW}{dt} = kW^{\alpha}$$

where W(t) is the weight of the fish at time t, k and α are empirically determined growth constants.

Curve of pursuit. An interesting geometric model arises when one tries to find the path of a pursuer chasing its prey. This path is called the **curve of pursuit**. These problems were analyzed using methods of the calculus of variations[30]. The simplest problem is to determine the curve along which a vessel moves in pursuing another vessel that flees along a stright line, supposing the speeds of two vessels are constant.

Secretion of hormones. The secretion of hormones into blood is often a periodic activity. If a hormone is secreted on a 24-hr cycle, then the rate of change of the level of the hormone in the blood may be represented by the following Cauchy problem

$$\begin{cases} \frac{dx}{dt} = \alpha - \beta \cos\frac{\pi t}{12} - kx \\ x(0) = x_0 \end{cases},$$

where x(t) is the amount of the hormone in the blood at time t, α is the average secretion rate, β is the amount of daily variation in the secretion, and k is a positive constant reflecting the rate at which the body removes the hormone from the blood.

[30] It was Leonardo da Vinci (1452-1519) who considered this problem.

References

[1] Z. Denkowski, S. Migórski and N.S. Papageorgiou, *An Introduction to Nonlinear Analysis: Theory.* Kluwer Academic/Plenum Publishers, Boston, Dordrecht, London, New York (2003).

[2] Z. Denkowski, S.Migórski and N.S. Papageorgiou, *An Introduction to Nonlinear Analysis: Applications.* Kluwer Academic/Plenum Publishers, Boston, Dordrecht, London, New York (2003).

[3] G. Duvaut and J. L. Lions, *Inequalities in Mechanics and Physics*, Springer Verlag, Berlin (1976).

[4] N. Garcia and A. C. Damask, *Physics for Computer Science Students*, John Wiley & Sons, New York, NY (1986).

[5] R. K. Nagle and E. B. Saff, *Fundamentals of Differential Equations and Boundary Value Problems*, Addison-Wesley, Reading, MA (1999).

Exercises

Pre-reading questions

1. Do you know how to build a mathematical model?

- Clearly state the assumptions on which the model will be based. These assumptions should describe the relationships between the quantities to be studied.
- Completely describe the parameters and variables to be used in the model.
- Use the assumptions from step 1 to derive mathematical equations relating the parameters and variables from step 2.

2. Do you think that theoretical mathematical model can be applied to solve practical problems?

Comprehension questions

1. How can we check a model for system correctness?

- In general, once we have built a set of equations, we compare the data generated by the equations with real data collected from the system (by measurement). When the two sets of data "agree" (or are "sufficiently" close), we gain confidence that the set of equations will lead to a good description of the real-world system.

2. In what situation is the use of differential equations obvious?

- Whenever a mathematical model involves the rate of change of one variable with respect to another.

3. What does Newton's law of cooling state?

- It is known from experimental observations that the surface temperature of an object changes at a rate proportional to its relative temperature. The latter is the difference between its temperature and the temperature of the surrounding environment.

Further reading

The whole article is located on English++ webpage.

Writing Software Patterns

Martin Fowler

Number of words	980 (selected part)
Computer science content	Medium
Business content	Medium
English language complexity	Medium

Sub-areas covered

Software engineering

- Object-oriented programming

Learning objectives

- to understand what design patterns are
- to see why patterns are so important in software development
- to be aware of the basic characteristics of patterns

Keywords

- **design pattern** - a general repeatable solution to a commonly occurring problem in software design. A design pattern is not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.
- **cookbook** - a book of recipes and solutions for different problems on writing software, usually connected with a particular programming language and platform
- **patterns book** - a book on design patterns
- **identity map** - a database access design pattern used to improve performance by providing a context-specific in-memory cache to prevent duplicate retrieval of the same object data from the database
- **decorator pattern** - a design pattern that allows new/additional behaviour to be added to an existing class dynamically

Summary

Martin Fowler is a famous author and international speaker on software architecture, specializing in object-oriented analysis and design, UML, patterns, and agile software development methodologies, including extreme programming. His article is a kind of introduction to the software design process. He is trying to show, with a handful of examples, why design patterns are important in software engineering.

Martin Fowler, autor licznych prac związanych z inżynierią oprogramowania, specjalizujący się między innymi w projektowaniu i programowaniu obiektowo-orientowanym, UML, wzorcach projektowych i "zwinnych" technikach wytwarzania oprogramowania (agile software development methodologies), w tym także w technice programowania "ekstremalnego". Jego artykuł stanowi rodzaj wstępu do opisu procesu projektowania oprogramowania. Autor wyjaśnia znaczenie wzorców projektowych w tym procesie, popierając swoją wypowiedź licznymi przykładami.

Pre-reading questions

1. What do you think might help in building software? What could make this process faster and more effective?
2. Do you think that using ordinary solutions might help or would that be useless in a specific task?
3. What types of software patterns do you know? Which one is your favourite and why?

Writing Software Patterns

What is a Pattern?

A common definition of a pattern is that it is a solution to a problem in a context. That's a definition that's always struck me as being rather unhelpful.

For me, a pattern is primarily a way to chunk up advice about a topic. Chunking is important because there's such a huge amount of knowledge you need to write software. As a result there needs be ways to divide knowledge up so you don't need to remember it all - what you need is to be able to get at a particular chunk of knowledge when you need it. Only then do you need details.

The solution provides a useful focus for the chunking. With some young eager programmer asking some grizzly veteran (i.e. anyone over thirty) how to deal with a particular situation and hear the veteran say "oh - you'll need an identity map there". The colleague can then look up identity map in some suitable patterns book.

So to make this chunking work each pattern should name a solution. This solution should be concrete, at least at the level of discussion we are talking about. You should be able to go away and use the pattern once you're given the reference. If you're successful the name should enter the vocabulary of the profession. It can take a while to do this, but when you say 'decorator' any reasonable professional should know what you mean.

Patterns should have recurrence, which means the solution must be applicable in lots of different situations. If you are talking about something that's a one-off, then it's not adding the name to the profession's vocabulary.

One of the interesting things here is that a singular solution can often lead to a recurrent pattern. This usually crops up when you see two different singular solutions which look completely different on the surface, yet have a deeper similarity - what Christopher Alexander* refers to as the "core of the solution". Let me give an example for this. I was looking at one of our early Java web projects. On this project the team

identity map
a database access design pattern used to improve performance by providing a context-specific in-memory cache to prevent duplicate retrieval of the same object data from the database

patterns book
a book on design patterns

* Christopher Alexander (born October 4, 1936 in Vienna, Austria) is an architect noted for his theories about design. He concluded that many design problems are so complex and so ancient that they are best resolved by learning from solutions which have proved successful over an endless period of time. He describes these solutions as 'archetypes' or 'patterns'. They are particularly useful in outdoor design, where patterns are comparatively easy to identify and difficult to explain. He characterizes this approach in the following words: "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem in such a way that you can use this solution a million times over, without ever doing it the same way twice" : http://www.lih.gre.ac.uk/resource/archetype.htm

wasn't allowed to use JSPs. So they wrote a set of Java classes which walked through a structure of domain objects, and produced the appropriate HTML for a particular domain object. They noticed they were getting duplication in the code for spitting out common HTML structures for fields, tables, etc. So they pulled all of the HTML spitting code out into a second utility class that had methods like renderField (String label). When they did this they noticed that they could make drastic changes to the entire web application's appearance just by altering code in the utility class.

design pattern
a general repeatable solution to a commonly occurring problem in software design. A design pattern is not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

Later on I saw a different project. They were using XSLT to turn XML into HTML pages, much as I do on this site. But they needed to support multiple organizations who wanted the same data displayed in their own format. So they split the transformation into two steps, first producing an intermediate XML with elements like field and table, with the second stage actually producing the HTML. They would have a different second stage for each organization.

Although it seems obvious as I write it now, when I first saw these two projects I sensed there was something similar in their approaches. However, it took me several months to understand the key point - splitting a transformation into two steps: logical page and physical (HTML) page. This is the "core of the solution" which I wrote up as Two Step View. One of the great intellectual challenges of patterns is finding and isolating this core amongst all the surrounding stuff that's needed on real projects.

Patterns versus Recipes

decorator pattern
a design pattern that allows new/additional behaviour to be added to an existing class dynamically

A popular, and very effective, form of technical writing is the cookbook style (eg The Perl Cookbook, Rails Recipes). There is a lot of similarity between cookbooks and patterns books. Both emphasize a problem-solution style.

I see the big difference between the two in the notion of building a vocabulary. Recipes tend to be more particular, usually tied to a particular programming language and platform. Even when patterns are tied to a platform, they try to describe more general concepts.

As a consequence of this recipes have a stronger problem focus than the solution focus in patterns.

Although my writing interest is in patterns, this reflects my interest in general design principles rather than a judgment on the relative usefulness of the two styles. Both are effective for same basic reason - they chunk based on a concrete thing somebody wants to get done today. As a result I find both very effective. You can also learn great principles from them, but it's the answers to particular questions that bring you to the table.

Why are Patterns important?

One of the quotes that I find particularly appealing when I think about the need for patterns that that part of interest in patterns came from "...observations that projects fail despite the latest technology for lack of ordinary solutions"[PLoPD 1]. Patterns provide a way to organize and name those ordinary solutions to make it easier for people to use them.

Since these solutions are ordinary, it's common that experts in a field won't find anything new in a patterns book. For such people the biggest value of a patterns book is to help them to pass on the solutions to their colleagues.

Despite my liking for patterns, I don't think that patterns are the right approach for all situations. Even in my own latest patterns book, I used a mixture of patterns and narrative text. I think the patterns helped focus the narrative and provided a good way for me to separate the details of the solutions from the overview discussion of them. Patterns are a communication medium, and like any communication technique there are situations where they work well and those where they work badly. Practice and familiarity help you tell the difference.

cookbook
a book of recipes
and solutions
for different problems
on writing software,
usually connected with
a particular programming
language and platform.

Exercises

Pre-reading questions

1. What do you think might help in building software? What could make this process faster and more effective?
2. Do you think that using ordinary solutions might help or would that be useless in a specific task?
3. What types of software patterns do you know? Which one is your favourite and why?

Comprehension questions

1. What are the differences and similarities between cookbooks and pattern books?

2. What are the main characteristics of patterns? Name at least three key features of them.

- chunking information, naming a solution, recurrence/core of the solution

3. What do you understand by:

a. recurrence of the pattern?

- solution must be applicable in different situations

b. core of the solution?

- a deeper similarity of problems

4. Why are ordinary solutions so important? How can they help in writing software?

- e.g. by providing well-known and tested solutions for new problems

5. Explain how patterns can be used as a communication technique.

Further exercises

- Are the following statements TRUE or FALSE?

1. The author defines a pattern as the solution to a problem in a context.
   (FALSE)
2. According to the author of the text, dividing knowledge up simplifies writing software.
   (TRUE)
3. If you find a solution to a singular problem it's completely useless as a pattern.
   (FALSE, you can evolve it into a wide-context solution)
4. In the author's opinion, recipes are more general than patterns and you can basically learn principles from them.
   (FALSE)
5. The main similarity between recipes and patterns is the idea of chunking information to make finding answers for particular questions easier.
   (TRUE)
6. The author recommends using patterns in all possible situations.
   (FALSE)

• Find words or phrases in the text that match the folowing meanings:

1. break up, divide up something (e.g. information, text); **[chunk up]**
2. to try to find a piece of information in a book or on a computer; **[look up]**
3. affecting or relating to a person or thing; **[applicable]**
4. make a useful contribution to a discussion; **[bring to the table]**
5. release (e.g. results); **[spit out]**
6. belief or idea, approach to some problem **[notion]**

Possible topics for discussion

- What are design patterns in architecture?
- What could this term refer to in software engineering?
- Which design patterns have you met so far? (e.g. MVC - Model-View-Controller)

Possible difficulties

No difficulties should be found in that part of the article.

Reading suggestions

- http://en.wikipedia.org/wiki/Design_pattern_(computer_science)
- Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software

Possible links to other topics

- SoftwareEntropy
- SoftwareDevelopmentProcess
- ExtremeProgramming

Cyber Warfare: Reality or Box Office Hit?

Randy Nash

Number of words	1600
Computer science content	Medium
English language complexity	Medium

Learning objectives

- to acquire basic knowledge about cyber threats
- to acquire knowledge about types of cyber attacks

Sub-areas covered

- Network safety
- cyber attacks

Keywords

- **cyber war** - the usage of computers and the Internet in conducting warfare in the cyberspace
- **cyber vandalism** - cyber attacks that deface web pages
- **hacktivism** - combination of words: "hack(er)" and "activism"
- **Denial of Service** - a kind of cyber attack which results in unavailability of service. It uses two methods: forcing the targeted computer(s) to reset, thus consuming its resources so that it can no longer provide its intended service; or obstructing the communication media between the intended users and the victim so that they can no longer communicate adequately.
- **mainframe** - computer used by large companies for critical operations like financial transaction processing
- **bot** - a program used on the Internet that performs a repetitive function such as posting a message to multiple newsgroups or searching for information or news
- **botnet** - a collection of software robots, or bots linked and cooperating with each other

Summary

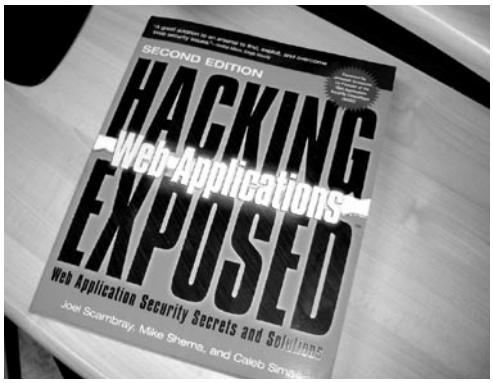
A short text, which is a good introduction to the topic of network security. There is a presentation of cyberterrorism as a new threat. The majority of people are not yet aware of how dangerous this phenomenon is. The author gives examples of the largest

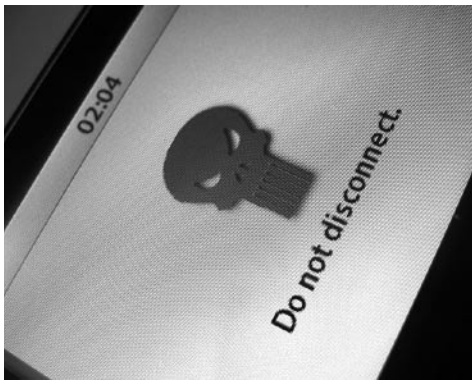
attacks in the past and describes the most popular techniques. The article is also a good introduction to a long discussion about prevention, awareness of the threat and the future.

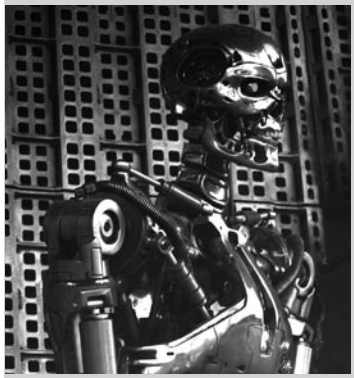
Krótki tekst wprowadzający w tematykę bezpieczeństwa w sieci. Przedstawia wizję zupełnie nowego rodzaju zagrożenia jakim jest cyberterroryzm. Autorem jest Randy Nash, który oprócz kilku artykułów na portalu InformIT.com, prowadzi swój własny blog nt związane z bezpieczeństwem w sieci. Autor podaje przykłady największych ataków w historii i opisuje pokrótce kilka najpopularniejszych technik. Artykuł ponadto prowokuje do dyskusji o świadomości zagrożenia, prewencji i przyszłości.

Pre-reading questions

1. What does the threat of cyber terrorism mean to you?
2. Where, in your opinion, does the main threat of cyber terrorism come from?
3. Have you thought about the threat of cyber terrorism?







Cyber Warfare: Reality or Box Office Hit?

The specter of cyber warfare has reared its ugly head in the media again. Some experts seem to think that the future is here, and battles will be fought and won via the Internet. But how much of this can be considered actual warfare, and how much is hype? Randy Nash searches for a working definition of cyber warfare, looks at the historical profile of attacks, and discusses the potential of a devastating electronic Pearl Harbor.

cyber war
the usage of computers and the Internet in conducting warfare in the cyberspace

cyber vandalism
cyber attacks that deface web pages

The threat of cyber warfare has been brought forth again by the media; not only in news outlets but also in the entertainment industry. But is this threat real or imagined?

What Is Cyber Warfare?

To properly assess the risk, we first need a working definition of what cyber war is and how it might manifest. There is much disagreement about the term and its actual definition, and varying degrees in the types of cyber attacks that might be perpetrated. I think cyber warfare has some critical characteristics. First, warfare is considered the process of military struggle between two nations or groups of nations. Warfare generally includes attacks against critical communications channels, and against the military and civilian population resulting in loss of life. Next, the cyber aspect can be considered to refer to the realm of electronic communication. So I think we can summarize the definition of cyber war as follows: cyber war (Cyber warfare): the process of military struggle between two nations or groups of nations conducted via various forms of electronic communications, or the Internet, resulting in the disruption of communications and/or loss of life.

What Is It Not?

Okay, we have a definition of cyber war, but what are other forms of cyber threats that would not fit into the definition of cyber warfare? Here's a list of some online threats that I don't feel fall into the warfare category:

Cyber vandalism: This is primarily the "script kiddies" that consider defacing a website to be "hacking." I think of this as electronic graffiti. This is low-level harassment.

Civil disobedience or hacktivism: Hacktivism or Electronic Civil Disobedience (ECD) generally takes some form of Denial of Service (DoS) attack against the website of some target, usually political in nature. Those who consider themselves hactivists claim that hacktivism is the fusion of hacking and activism; politics and technology. More specifically, hacktivism is described as hacking for a political cause. Furthermore, hacktivists claim that Electronic Civil Disobedience (ECD) is a legitimate form of nonviolent, direct action utilized to bring pressure on institutions engaged in unethical or criminal actions. Within the electronic environment, ECD aims to disrupt the

operation of information and capital flows of carefully selected target sites without causing serious damage.

Cyber crime: Cyber crime can take many different forms; theft of intellectual property, extortion based on the threat of DDOS attacks, fraud based on identity theft, espionage, and so on.

While each of these types of activities is criminal in nature and might constitute an aspect of cyber warfare actions, they do not constitute cyber war by themselves. These actions, while criminal in nature, are generally not going to cost lives or affect military operations.

What is the Real Threat?

The threat comes primarily from nation states with the will, motive, and technology to launch attacks against the United States. In fact, nations that do not possess a powerful military are probably more likely to choose this form of attack because of its much lower cost of implementation (think "asymmetric warfare"). The most obvious avenues of attack would be against our nation's critical infrastructure, so the government formed a body to analyze any threats against our infrastructure. This body, originally known as the President's Critical Infrastructure Protection Board (PCCIB), is now known as the National Infrastructure Advisory Council (NIAC) and operates within the U.S. Department of Homeland Security. This council provides the President with advice on the security of the critical infrastructure sectors and their information systems. Over time, the sectors identified as belonging to the critical infrastructure have changed.

Denial of Service
a kind of cyber attack which results in unavailability of service. It uses two methods: forcing the targeted computer(s) to reset, thus consuming its resources so that it can no longer provide its intended service; or obstructing the communication media between the intended users and the victim so that they can no longer communicate adequately.

Historical Precedents

Just as our understanding of this critical infrastructure has changed and matured over time, so has the threat. These threats have existed for some time. Back in May 1998, all seven members of the L0pht (Brian Oblivion, Kingpin, Mudge, Space Rogue, Stefan Von Neumann, John Tan, and Weld Pond) famously testified before the Congress of the United States that they could shut down the entire Internet in 30 minutes.

NOTE Selected transcripts are in the section entitled "WEAK COMPUTER SECURITY IN GOVERNMENT: IS THE PUBLIC AT RISK?" MAY 19–20, 1998—COMMITTEE ON GOVERNMENTAL AFFAIRS.

Shortly after this testimony, the first distributed denial-of-service (DDoS) attacks appeared. The first well-documented DDoS attack appears to have occurred in August 1999, when a DDoS tool called Trinoo (described below) was deployed in at least 227 systems, of which at least 114 were on Internet2, to flood a single University of

Minnesota computer. This system was knocked off the air for more than two days. In the following months, Yahoo!, Amazon, Buy.com, CNN, and eBay were all hit with similar attacks. These commerce sites suffered large financial losses during the downtime because of these attacks. Today, incredibly large and complex botnets exist that can be used to launch a variety of attacks against multiple targets.

mainframe
computer used by large companies f or critical operations like financial transaction processing

The current focus of these botnets appears to be primarily SPAM and DDoS attacks, but they could easily be used in cyber warfare activities. Other attacks of historic significance include the following:

Solar Sunrise: Solar Sunrise is the name given to a series of attacks against the Pentagon and MIT in February 1998. The Department of Defense called these attacks "the most organized and systematic attack to date." The DoD actually declared the U.S. to be in a state of "cyber war." These attacks appeared to be originating from Russian-owned IP address space, so the attack was considered to be "state-sponsored."

Moonlight Maze: Moonlight Maze refers to an incident in which U.S. officials accidentally discovered a pattern of probing of computer systems at the Pentagon, NASA, Energy Department, private universities, and research labs that began in March 1998 and went on for nearly two years. It seems that these hackers had been able to access tens of thousands of files (including maps of military installations, troop configurations, and military hardware designs). The Defense Department traced the attacks back to a mainframe computer in the former Soviet Union.

bot
a program used on the Internet that performs a repetitive function such as posting a message to multiple newsgroups or searching for information or news

Titan Rain: Titan Rain is the name given to a well-organized Chinese military hacking effort against the U.S. military. The hackers, believed to have been based in the Chinese province of Guangdong, are thought to have stolen U.S. military secrets, including aviation specifications and flight-planning software. These attacks apparently started in 2003 and lasted until 2005.

Current Situations

Much of the news related to cyber warfare tends to be a bit "sensational". For example:

Russia accused of unleashing cyber war to disable Estonia, Estonia hit by "Moscow cyber war", Cyber war: Russia vs. Estonia.

All this sounds very dramatic and serious, but let's look at the details. On April 27, officials in Estonia relocated the "Bronze Soldier," a Soviet-era war memorial commemorating an unknown Russian who died fighting the Nazis. This led to political furor among ethnic Russians and to the blockading of the Estonian Embassy in Moscow.

The event also marked the beginning of a large and sustained distributed denial-of -service attack on several Estonian national Web sites, including those of government

ministries and the prime minister's Reform Party. There were no attacks against any critical infrastructure or services. This was simply a political statement. At the most, I'd refer to this event as hacktivism, and most likely launched by a group of hackers that have no affiliation with any government agency. It seems there was no cyber war after all. But wait, that's not all: America prepares for "cyber war" with China!

Is Cyber war really that imminent? Are we about to fall under an attack of bits and bytes? A report in the UK branch of ZDNet proclaimed Cyber warfare "a reality in 12 months."

Unfortunately, that report was back in January 2004. The article is a good source of information about the types of vulnerable systems that may be attacked, but their predicted timeline is way off. .

botnet
a collection of software robots, or bots linked and cooperating with each other

Cyber Warfare—An analysis of the means and motivations of selected nation states For a more measured appraisal, I recommend this report, which was written in response to a grant provided by the Department of Homeland Security. This report is an assessment of potential foreign computer threats to information technology networks in the United States. This is one of the most level-headed write-ups I've seen to-date.

Looking to the Future

There should be little doubt that future wars will inevitably include cyber warfare tactics. It is increasingly apparent that other nations are gearing up to take advantage of the ever-increasing complexity and inter-connected nature of various national infrastructures.

Current efforts at security computer systems and networks will likely prove to be insufficient to prevent such future attacks. The U.S. military has been developing cyber warfare strategies for some time.

hacktivism
combination of words: "hack(er)" and "activism"

It appears that even Al Qaeda has been developing cyber war capabilities. With our enemies working toward this goal, we obviously cannot overlook the possibilities. The scenario in the movie Live Free or Die Hard might seem farfetched, but the potential exists for some of the attacks portrayed in that movie (although it won't be nearly as easy as it seemed on the big screen). I'm hoping that somewhere in either our government or military cyberdefense forces we have our own John McClane?

Exercises

Pre-reading questions

1. What does the threat of cyber terrorism mean to you?
- dangerous financial transactions
- theft data from hard disk
- eavesdrop traffic in network
2. Where, in your opinion, does the main threat of cyber terrorism come from?
- nowadays - hackers acted individually, usually want to prove their skills
- in the future – very good organised teams of specialists, supported by government institutions
3. Have you thought about the threat of cyber terrorism?

Comprehension questions

1. Using the information in the text about the first DDoS attack and your own knowledge, try to explain how DDoS works?

- A distributed denial of service attack occurs when multiple compromised systems flood the bandwidth or resources of a targeted system, usually one or more web servers. It uses two methods: forcing the targeted computer(s) to reset, consume its resources such that it can no longer provide its intended service or obstructing the communication media between the intended users and the victim so that they can no longer communicate adequately.

2. Can you name the large-scale attacks and who they were directed against?

- Solar Sunrise: Solar Sunrise is the name given to a series of attacks against the Pentagon and MIT in February 1998.
- Moonlight Maze: Moonlight Maze refers to an incident in which U.S. officials accidentally discovered a pattern of probing of computer systems at the Pentagon, NASA, Energy Department, private universities, and research labs that began in March 1998 and went on for nearly two years.
- Titan Rain: Titan Rain is the name given to a well-organized Chinese military hacking effort against the U.S. military.

Further exercises

In groups, explain the meaning of these words:

- hacktivism
- mainframe
- cyberdefense
- bot
- botnet

Compare your answers with other groups.

Possible topics for discussion

1. Do you know of any other kinds of cyber attacks or threats? Which do you think, are the most dangerous for you?
2. Are you afraid of this kind of threat? Do you think you might lose something in a cyber attack?

Possible difficulties

The article contains a lot of factual information, but it should make it easier to read.

Evolutionary Database Design

Martin Fowler & Pramod Sadalage

Number of words	4345
Computer science content	Medium
English language complexity	Medium

Sub-areas covered

- database design
- agile methodologies

Learning objectives

- to understand the differences between up-front and agile methodologies
- to see why team communication is so important while working on a project
- to understand what is meant by refactoring

Keywords

database - software designed for holding large amounts of data

database schema - a description of the tables and views in a database together with the relationships between them

Agile Methodology - a particular method of producing software, emphasising self-organizing teams and broad customer-programmer contact

XP (extreme programming) - a programing methodology which is very closely connected with the Agile style

up-front designing - describes a method in which the design phase of a project is completed before the coding of the project begins

DBA - database administrator; the main role of a database administrator has to do with overseeing the installation and ongoing function of software on a system designed for use by a number of users

refactoring - making changes to a computer system (e.g. source code, database) that improve its readability and simplify its structure without adding any new functionality; for example, changing the names of variables to be more readable

Summary

Martin Fowler is a widely acknowledged expert in the areas of software design, methodologies and design patterns. In this article he introduces an alternative (to waterfall methodology) method of developing databases and programs. The main

idea is to regard database design as an evolutionary process.

Tekst, choć dość długi, ciekawie przedstawia podejście do projektowania baz danych w sposób podobny do zwinnego programowania. Artykuł wprowadza w tematykę tej nowej metodologii, jako alternatywy dla tzw. metodologii "waterfall". Dostajemy wiele informacji na temat tego jak traktować projektowanie baz danych w sposób jakby to był duży ewoluujący obiekt.

Pre-reading questions

1. Have you heard about Agile Methodology? What are they used for?
2. What does DBA stand for?
3. What are DBA's functions?

Evolutionary Database Design

Abstract

Over the last few years the authors have developed a number of techniques that allow a database design to evolve as an application develops. This is a very important capability for agile methodologies. The techniques rely on applying continuous integration and automated refactoring to database development, together with a close collaboration between DBAs and application developers. The techniques work in both pre-production and released systems.

In the last few years, we've seen the rise of a new breed of software methodologies, the agile methodologies. These make some new and significant demands on database design. One of the most central of these demands is the idea of evolutionary design. On an agile project you assume that you cannot fix the requirements of the system up-front. As a result having a detailed design phase at the beginning of a project becomes impractical. The design of the system has to evolve through the various iterations of the software. Agile methods, in particular extreme programming (XP), have a number of practices that make this evolutionary design practical.

Many people have questioned whether evolutionary design can be applied to a system with a large database component. Indeed many people told us that it was impossible - a troubling thought as ThoughtWorks embarked on a large database-oriented project using many agile and XP techniques.

This article describes the practices that we've used to allow us to do this impossible thing. We won't say that we've completely solved the database evolution problem, but we do think we've demonstrated a set of techniques that many people will find useful.

database
software
designed for holding
large amounts of data

Dealing with Change

One of the primary features of agile methods is their attitude towards change. Most of the thinking about software process is about understanding requirements early, signing off on these requirements, using the requirements as a basis for design, signing off on that, and then proceeding with construction. This is a plan-driven cycle, often referred to (usually with derision) as the waterfall approach.

Such approaches look to minimize changes by doing extensive up-front work. Once the early work is done, changes cause significant problems. As a result such approaches run into trouble if requirements are changing, and requirements churn is a big problem for such processes.

Agile processes approach change differently. They seek to embrace change, allowing changes to occur even late in a development project. Changes are controlled, but the attitude of the process is to enable change as much as possible. Partly this is in response to the inherent instability of requirements in many projects, partly it is to better support dynamic business environments by helping them change with the competitive pressures.

In order to make this work, you need a different attitude to design. Instead of thinking of design as a phase, which is mostly completed before you begin construction, you look at design as an on-going process that is interleaved with construction, testing, and even delivery. This is the contrast between planned and evolutionary design. One of the vital contributions of agile methods is that they have come up with practices that allow evolutionary design to work in a controlled manner. So instead of the common chaos that often happens when design isn't planned up-front, these methods provide techniques to control evolutionary design and make them practical.

database schema
a description of the tables and views in a database together with the relationships between them

An important part of this approach is iterative development, where you run the entire software life-cycle many times during the life of a project. Agile processes run complete life cycles in each iteration, completing the iteration with working, tested, integrated code for a small subset of the requirements of the final product. These iterations are short, usually running between a week and a couple of months, with a preference towards shorter iterations.

While these techniques have grown in use and interest, one of the biggest questions is how to make evolutionary design work for databases. Most people consider that database design is something that absolutely needs up-front planning. Changing the database schema late in the development tends to cause widespread breakages in application software. Furthermore changing a schema after deployment results in painful data migration problems.

Over the course of the last three years we've been involved in a large project (called Atlas) that has used evolutionary database design and made it work. The project involved almost 100 people in multiple sites world-wide (US, Australia, and India). It is around half a million lines of code and has over 200 tables. The database evolved during a year and a half of initial development and continues to evolve even though it's in production for multiple customers. During this project we started with iterations of a month, but after a few months changed to two week iterations which worked better. The techniques we describe here are the ones that we (or more accurately Pramod) used to make this work.

Since that project got going we've spread these techniques over more of our projects, gaining more experience from more cases. We've also found inspiration, ideas, and experience from other agile projects.

Limitations

Before we dive into the techniques, it's important to state that we haven't solved all the problems of evolutionary database design. In particular:

- We developed an application database for a single application rather than an integration database that tries to integrate multiple databases.
- We don't have to keep the production databases up 24/7

We don't consider these problems to be inherently unsolvable, after all many people believed we couldn't solve this one. But until we do, we won't claim we can solve them either.

The Practices

Our approach to evolutionary database design depends on a handful of important practices.

DBAs collaborate closely with developers

One of the tenets of agile methods is that people with different skills and backgrounds need to collaborate very closely together. They can't communicate mainly through formal meetings and documents. Instead they need to be out talking with each other and working with each other all the time.Everybody is affected by this: analysts, PMs, domain experts, developers... and DBAs.

Every task that a developer works on potentially needs a DBA's help. Both the developers and the DBA need to consider whether a development task is going to make a significant change to the database schema. If so the developer needs to consult with the DBA to decide how to make the change. The developer knows what new functionality is needed, and the DBA has a global view of the data in the application.

To make this happen the DBA has to make himself approachable and available. Make it easy for a developer to just pop over for a few minutes and ask some questions. Make sure the DBAs and developers sit close to each other so they can easily get together. Ensure that application design sessions are known about so the DBA can pop in easily. In many environments we see people erecting barriers between the DBA and application development functions. These barriers must come down for an evolutionary database design process to work.

Everybody gets their own database instance

Evolutionary design recognizes that people learn by trying things out. In programming terms developers experiment with how to implement a certain feature and may make a few attempts before settling down to a preferred alternative. Database design can be

like that too. As a result it's important for each developer to have their own sandbox where they can experiment, and not have their changes affect anyone else.

Many DBA experts see multiple databases as anathema, too difficult to work in practice, but we've found that you can easily manage a hundred or so database instances. The vital thing is to have to tools to allow you to manipulate databases much as you would manipulate files.

Developers frequently integrate into a shared master

Although developers can experiment frequently in their own area, it's important to bring the different approaches back together again frequently. An application needs a shared master database that all work flows from. When a developer begins a task they copy the master into their own workspace, manipulate, and then integrate their changes back into the master. As a rule of thumb each developer should integrate once a day.

Let's take an example where Mike starts a development task at 10am (assuming he actually comes in that early). As part of this task he needs to change the database schema. If the change is easy, like adding a column, he just decides how to make the change himself, Mike also makes sure the column he wants to add does not already exist in the database, with the help of the data dictionary (discussed later). If it's more complicated then he grabs the DBA and talks over the likely changes with him.

Once he's ready to begin he takes a copy of the database master and can modify both the database schema and code freely. As he's in a sandbox any changes he makes don't impact anyone else's. At some point, say around 3pm, he's pretty comfortable that he knows what the database change needs to be, even though he's not completely done with his programming task. At that point he grabs the DBA, and tells him about the change. At this point the DBA can raise any issues that Mike hasn't considered. Most of the time all is well and the DBA goes off and makes the change (by applying one or more database refactorings, which we'll come to below). The DBA makes the changes right away (unless they are destructive changes - again more on that below). Mike can continue to work on his task and commit his code any time he likes once the DBA has applied these changes to the master.

You may well recognize this principle as similar to the practice of Continuous Integration, which is applied to source code management. Indeed this is really about treating the database as another piece of source code. As such the master database is kept under configuration management in much the same way as the source code. Whenever we have a successful build, the database is checked into the configuration management system together with the code, so that we have a complete and synchronized version history of both.

DBA
database administrator; the main role of a database administrator has to do with overseeing the installation and ongoing function of software on a system designed for use by a number of users

With source code, much of the pain of integration is handled by source code control systems. For databases there's a bit more effort involved. Any changes to the database need to done properly, as automated database refactorings, which we'll discuss shortly. In addition the DBA needs to look at any database changes and ensure that they fit within the overall scheme of the database schema. For this to work smoothly, big changes shouldn't come as surprises at integration time - hence the need for the DBA to collaborate closely with the developers.

up-front designing describes a method in which the design phase of a project is completed before the coding of the project begins

We emphasize integrating frequently because we've found that it's much easier to do frequent small integrations rather than infrequent large integrations. It seems that the pain of integration increases exponentially with the size of the integration. As such doing many small changes is much easier in practice, even though it often seems counter-intuitive to many. This same effect's been noticed by people in the Software Configuration Management community for source code.

A database consists of schema and test data

When we talk about a database here, we mean not just the schema of the database, but also a fair amount of data. This data consists of common standing data for the application, such as the inevitable list of all the states in the US, and also sample test data such as a few sample customers.

The data is there for a number of reasons. The main reason is to enable testing. We are great believers in using a large body of automated tests to help stabilize the development of an application. Such a body of tests is a common approach in agile methods. For these tests to work efficiently, it makes sense to work on a database that is seeded with some sample test data, which all tests can assume is in place before they run.

As well as helping test the code, this sample test data also allows to test our migrations as we alter the schema of the database. By having sample data, we are forced to ensure that any schema changes also handle sample data.

In most projects we've seen this sample data be fictional. However in a few projects we've seen people use real data for the samples. In these cases this data's been extracted from prior legacy systems with automated data migration scripts. Obviously you can't migrate all the data right away, as in early iterations only a small part of the database is actually built. But the idea is to iteratively develop the migration scripts just as the application and the database are developed iteratively. Not only does this help flush out migration problems early, it makes it much easier for domain experts to work with the growing system as they are familiar with the data they are looking at and can often help to identify problem cases that may cause problems for the da-

tabase and application design. As a result we are now of the view that you should try to introduce real data from the very first iteration of your project.

All changes are database refactorings

The technique of refactoring is all about applying disciplined and controlled techniques to changing an existing code base. Similarly we've identified several database refactorings that provide similar control and discipline to changing a database.

One of the big differences about database refactorings is that they involve three different changes that have to be done together

- Changing the database schema
- Migrating the data in the database
- Changing the database access code

Thus whenever we describe a database refactoring, we have to describe all three aspects of the change and ensure that all three are applied before we apply any other refactorings.

We are still in the process of documenting the various database refactorings, so we aren't able to go into detail on them yet. However there are a few things we can point out. Like code refactorings, database refactorings are very small. The concept of chaining together a sequence of very small changes is much the same for databases as it is for code. The triple nature of the change makes it all the more important to keep to small changes.

Many database refactorings, such as adding a column, can be done without having to update all the code that accesses the system. If code uses the new schema without being aware of it, the column will just go unused. Many changes, however don't have this property. We call these destructive changes, an example of which is making an existing nullable column not null.

Destructive changes need a bit more care, the degree of which depends on the degree of destruction involved. An example of a minor destructive change is that of changing a column from nullable to not null. In this case you can probably just go ahead and do it. The refactoring will take care of any data in the database that's null. Usually the only developer who cares about this property is the one who requested the change, and that developer will update the database mapping code. As a result the update won't break anyone else's code and if by some strange chance it does, they find out as soon as they run a build and use their tests. (On our large project we gave ourselves some extra breathing space by waiting a week before making the database change.)

Splitting a heavily used table into two however is a rather more complicated case. In this case it's important to let everyone know that the change is coming up so they can

prepare themselves for it. In addition it's worth waiting for a safer moment to make the change. (These kinds of changes we would defer until the start of a new iteration - we like to use iterations of two weeks or less).

The important thing here is to choose a procedure that's appropriate for the kind of change that you're making. If in doubt try to err on the side of making changes easier. Our experience is that we got burned much less frequently than many people would think, and with a strong configuration control of the entire system it's not difficult to revert should the worst happen.

Automate the refactorings

Agile Methodology
a particular method of producing software, emphasising self-organizing teams and broad customer-programmer contact

In the world of code we are seeing tools for some languages to automate many of the identified refactorings. Such automation is essential for databases; at least in the areas of schema changes and data migration.

As a result every database refactoring is automated by writing it in the form of SQL DDL (for the schema change) and DML (for the data migration). These changes are never applied manually, instead they are applied to the master by running a small SQL script to perform the changes.

Once done, we keep hold of these script files to produce a complete change log of all the alterations done to the database as a result of database refactorings. We can then update any database instance to the latest master by running the change log of all the changes since we copied the master to produce the older database instance.

This ability to sequence automated changes is an essential tool both for the continuous integration process in development, and for migrating production databases to a new release.

For production databases we don't make changes during the usual iteration cycles. Once we do a release, which may occur at the end of any iteration, we apply the full change log of database refactorings since the previous release. This is a big change, and one that so far we've only done by taking the application offline. (We have some ideas for doing this in a 24/7 environment, but we haven't actually had to do it yet.) It's also wise to test this migration schema before applying it to the live database. So far, we've found that this technique has worked remarkably well. By breaking down all the database changes into a sequence of small, simple changes; we've been able to make quite large changes to production data without getting ourselves in trouble.

As well as automating the forward changes, you can consider automating reverse changes for each refactoring. If you do this you'll be able to back out changes to a database in the same automated way. We haven't done this yet, as we've not had a much demand for it, but it's the same basic principle.

(A similar thing that we have done is to support an old version of an application with an updated version of the database. This involved writing a compatibility layer that allowed the application to think it was talking to the older version of the database even though it was actually talking to the newer one.)

Automatically Update all Database Developers

It's all very well for people to make changes and update the master, but how do they find out the master has changed? In a traditional continuous integration environment with source code, developers update to the master before doing a commit. That way they can resolve any build issues on their own machine before committing their changes to the shared master. There's no reason you can't do that with the database, but we found a better way.

We automatically update everyone on the project whenever a change is made to the database master. The same refactoring script that updates the master automatically updates everyone's databases. When we've described this, people are usually concerned that automatically updating developers' databases underneath them will cause a problem, but we found it worked just fine.

This only worked when people were connected to the network. If they worked offline, such as on an airplane, then they had to resync with the master manually once they got back to the office.

Clearly separate all database access code

To understand the consequences of database refactorings, it's important to be able to see how the database is used by the application. If SQL is scattered willy-nilly around the code base, this is very hard to do. As a result it's important to have a clear database access layer to show where the database is being used and how. To do this we suggest following one of the data source architectural patterns from P of EAA.

Having a clear database layer has a number of valuable side benefits. It minimizes the areas of the system where developers need SQL knowledge to manipulate the database, which makes life easier to developers who often are not particularly skilled with SQL. For the DBA it provides a clear section of the code that he can look at to see how the database is being used. This helps in preparing indexes, database optimization, and also looking at the SQL to see how it could be reformulated to perform better. This allows the DBA to get a better understanding of how the database is used.

refactoring
making changes
to a computer system
(e.g. source code, database)
that improve
its readability and
simplify its structure with
out adding
any new functionality.
For example, changing the
names of variables
to be more readable

Variations

Like any set of practices, these should be varied depending on your specific circumstances. These practices are still pretty new, so we haven't come across that many variations, but here are some we have.

Keeping multiple database lineages

extreme programming (XP) a programing methodology which is very closely connected with the Agile style.

A simple project can survive with just a single database master in the repository. With more complex projects there's a need to support multiple varieties of the project database, which we refer to as database lineages. We may create a new lineage if we have to branch an application that's put into production. In essence creating a new database lineage is much the same as branching the source code on the application, with the added twist that you also make a lineage when you need a different set of sample data, such as if you need a lot of data for performance testing.

When a developer takes a copy of a master they need to register which lineage they are modifying. As the DBA applies updates to a master for a particular lineage the updates propagate to all the developers who are registered for that lineage.

You don't need a DBA

All of this sounds like it would be a lot of work, but in fact it doesn't require a huge amount of manpower. On the Atlas project we had thirty-odd developers and a team size (including, QA, analysts and management) of close to a hundred. On any given day we would have a hundred or so copies of various lineages out on people's workstations. Yet all this activity needed only one full time DBA (Pramod) with a couple of developers doing some part-time assistance and cover.

On smaller projects even that isn't needed. We've been using these techniques on a number of smaller projects (about a dozen people) and we find these projects don't need a full time DBA. Instead we rely on a couple of developers with an interest in DB issues who handle the DBA tasks part-time.

The reason for this is automation. If you are determined to automate every task, you can handle a lot of work with much less people.

Tools to Help

Doing this kind of thing requires a lot of repetitive tasks. The good news is that whenever you run into repetitive tasks in software development you are ideally placed to automate them. As a result we've developed a fair amount of often simple tools to help us.

One of the most valuable pieces of automation is a simple set of scripts for common database tasks.

- Bring a user up to date with the current master.
- Create a new user
- Copy a database schema, for example Sue finds a bug with her database, now Mike can copy Sue's database and try to debug the application
- Move a database, for example from a workstation to a different workstation, this is essentially Copy database and Delete database combined as one
- Drop a user
- Export a user so team members can make offline backups of the database that they are working with.
- Import a user, so if the team members have a backup copy of the database, they can import the backup and create a new schema.
- Export a baseline - make a backup copy of the master database. This is a specialized case of Export a User
- Create a difference report of any number of schemas, so that Mike can find out what is different structurally between his database and Sue's.
- Diff a schema against the master, so that developers can compare their local copy against the master.
- List all the users

Analysts and QA folks often need to look at the test data in the database and to be able to easily change it. For that we created an Excel application with VBA scripts to pull data down from the database into an excel file, allow people to edit the file, and send the data back up to the database. Although other tools exist for viewing and editing the contents of a database, excel works well because so many people are familiar with it.

Everybody on the project needs to be able to explore the database design easily, that way they can find out what tables are available and how they are used. We built an HTML based tool to do this that used servlets to query database metadata. So Mike before adding the column to a table, could go look up if the column he is adding already exists, by searching the metadata of the tables and columns. We did the data modeling using ERwin and pulled data from ERwin into our own metadata tables.

Exercises

Pre-reading questions

1. Have you heard about Agile Methodology? What are they used for?
2. What does DBA stand for?
3. What are DBA's functions?

Comprehension questions

Dealing with Change

1. What are the main advantages of Agile Methodology?

- higher flexibility to change of development plans - changes to a system are controlled at each stage of a project
- higher quality through earlier feedback from customers
- shortened development cycle time
- higher stability of workloads

2. What is the main difference between planned and evolutionary design?

- in planned methodology: treat design as a independent phase of project development before implementation and testing
- evolutionary design: run the entire software life-cycle many times during the life of a project

3. Why does agile methodology appear, at first glance, unsuitable for database design?

- "Most people consider that database design is something that absolutely needs up-front planning. Changing the database schema late in the development tends to cause widespread breakages in application software. Furthermore, changing a schema after deployment result in painful data migration problems."

4. What limitations do the authors admit to in using an evolutionary approach to database design?

- They developed an application database for a single application rather than an integration database that tries to integrate multiple databases.
- They don't have to keep the production databases up 24/7

The Practices

1. Why is close contact between developers and DBAs essential?

- " Every task that a developer works on potentially needs a DBA's help. Both the developers and the DBA need to consider whether a development task is going to make a significant change to the database schema. If so the developer needs to consult with the DBA to decide how to make the change. The developer knows what new functionality is needed, and the DBA has a global view of the data in the application. To make this happen the DBA has to make himself approachable

and available. Make it easy for a developer to just pop over for a few minutes and ask some questions. Make sure the DBAs and developers sit close to each other so they can easily get together. Ensure that application design sessions are known about so the DBA can pop in easily."

2. What allows individual programmers to independently test their own solutions on a database without negative consequences for the project?

- "It's important for each developer to have their own sandbox where they can experiment, and not have their changes affect anyone else. Many DBA experts see multiple databases as anathema, too difficult to work in practice, but we've found that you can easily manage a hundred or so database instances. The vital thing is to have to tools to allow you to manipulate databases much as you would manipulate files."

3. What is the difference between a source code and database integration?

- "With source code, much of the pain of integration is handled by source code control systems. For databases there's a bit more effort involved. Any changes to the database need to done properly, as automated database refactorings. In addition the DBA needs to look at any database changes and ensure that it fits within the overall scheme of the database schema. For this to work smoothly, big changes shouldn't come as surprises at integration time - hence the need for the DBA to collaborate closely with the developers."

4. What is the purpose of having test data?

- " The main reason is to enable testing. We are great believers in using a large body of automated tests to help stabilize the development of an application. Such a body of tests is a common approach in agile methods. For these tests to work efficiently, it makes sense to work on a database that is seeded with some sample test data, which all tests can assume is in place before they run."
- It allows the testing of migrations as the schema of the database are altered.

5. How are automated changes in databases accomplished?

"Every database refactoring is automated by writing it in the form of SQL DDL (for the schema change) and DML (for the data migration). These changes are never applied manually, instead they are applied to the master by running a small SQL script to perform the changes. Once done, we keep hold of these script files to produce a complete change log of all the alterations done to the database as a result of database refactorings. We can then update any database instance to the latest master by running the change log of all the changes since we copied the master to produce the older database instance."

6. What are the advantages of separating the database layer from the code base?

- It minimizes the areas of the system where developers need SQL knowledge to manipulate the database, which makes life easier to developers who often are not particularly skilled with SQL.

- It provides DBA with a clear section of the code that he can look at to see how the database is being used.
- It helps in preparing indexes, database optimization, and also looking at the SQL to see how it could be reformulated to perform better.

Possible topics for discussion

- What other methodologies do you know?
- What are the possible advantages of these methodologies? Do they have any drawbacks?
- How do you usually design your programs?

Possible difficulties

It is quite a long text but it can be read quite easily.

Reading suggestions

- Wikipedia has an interesting page about agile development.
- Martin Fowler has a very nice web page with a lot of articles worth reading. Be Make sure to visit it.
- If you want to find out more about Agile take a look at Agile Alliance.

Lord Palmerston on Programming

Joel Spolsky

Number of words	2230
Computer science content	Medium
Business content	Low
English language complexity	High

Sub-areas covered

- self-developing
- learning
- programmers' culture
- programming expertise

Learning objectives

- to understand what is important in a programmer's work
- to recognize practical attitudes towards programming and work
- to be aware of the kind of problems programmers have to face

Keywords

regexp - an abbreviation for regular expressions. In computing, regular expressions provide a concise and flexible means for identifying strings of text of interest, such as particular characters, words, or patterns of characters

Windows - an operating system

Api (Application programming interface) - a source code interface that an operating system, library or service provides to support requests made by computer programs

ASP (Action Server Pages) - Microsoft's first server-side script engine for dynamically -generated web pages

GUI (Graphical User Interface) - a type of user interface which allows people to interact with a computer and computer-controlled devices

VBScript - an Active Scripting language developed by Microsoft

Summary

This article is about different computer programming "worlds" and how much they differ from one another. The author suggests that the difference between programming "worlds" is so big that it is almost impossible for one person to understand and compare them. He also stresses the need for a programmer to never stop learning so as to stay

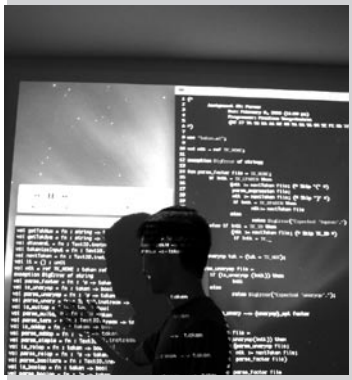
proficient in his/her computer programming "world".

Artykuł traktuje o konieczności ciągłego uczenia się dla zachowania statusu eksperta w programowaniu. Rozległość wiedzy, którą powinien posiąść programista rośnie coraz szybciej i tam, gdzie kiedyś wystarczyło przeczytać jedną książkę by wiedzieć wszystko, teraz nie wystarczają kilkuletnie studia. Autor podkreśla również wagę praktycznego doświadczenia przy pracach programistycznych.

Pre-reading questions

1. What does the phrase "a good programmer" mean to you?
2. What abilities/skills should a good programmer have?
3. Api or library? Which is more important?
4. Give some examples of the main programming areas.
5. Which programming languages are important now?

Lord Palmerston on Programming



There was a time when if you read one book by Peter Norton, you literally knew everything there was to know about programming the IBM-PC. Over the last 20 years, programmers around the world have been hard at work building abstraction upon abstraction on top of the IBM-PC to make it easier to program and more powerful.

But the law of leaky abstractions means that even as they built the abstractions that are supposed to make programming easier, the sheer amount of stuff you have to know to be a great programmer is expanding all the time.

Becoming proficient, really proficient, in just one programming world takes years. Sure, lots of bright teenagers learn Delphi one week and Python the next week and Perl the next week and think they are proficient. Yet they don't have the foggiest clue how much they're missing.

I've been working with ASP and VBScript since it first came out. VBScript is the dinkiest language on earth and ASP programming consists of learning about 5 classes, only two of which you use very often. And only now do I finally feel like I know the best way to architect an ASP/VBScript application. I finally think I know where the best place to put database access code is, the best way to use ADO to get recordsets, the best way to separate HTML and code, etc. And I finally use regexps instead of one-off string manipulation functions. Only last week, I learned how to get COM objects out of memory so you can recompile them (without restarting the whole web server).

Fog Creek is too small to have specialists, so when I needed to write a really good installer for FogBUGZ, our ASP/VBScript based product, I drew on several years of C++/MFC experience, and years of experience with Windows APIs, and good Corel Photopaint skills to create a neat picture in the corner of the wizard. Then to get FogBUGZ to work perfectly with Unicode, I had to write a little ActiveX control using C++ and ATL, which drew upon years of C++ and COM experience and a week or so learning about character encodings when I implemented that code in CityDesk.

Windows
an operating system

Api
application programming interface
- a source code interface that an operating system, library or service provides to support requests made by computer programs.

So when we had a weird NT 4.0-only bug, it took me 3 minutes to debug, because I knew how to use VMWare, and I had a clean NT 4.0 machine set up in VMWare, and I knew how to do remote debugging with Visual C++, and I knew to look in the EAX register to get the return value from a function. Someone who was new to this all might have taken an hour or more to debug the same problem, but I already knew a tremendous amount of "stuff" that I've been learning, basically, since 1982 when I got my first IBM-PC and that Norton book.

Leaky abstractions mean that we live with a hockey stick learning curve: you can learn 90% of what you use day by day with a week of learning. But the other 10% might take you a couple of years catching up. That's where the really experienced programmers will shine over the people who say "whatever you want me to do, I can just pick up the book and learn how to do it." If you're building a team, it's OK to have a lot of less experienced programmers cranking out big blocks of code using the abstract tools, but the team is not going to work if you don't have some really experienced members to do the really hard stuff.

There are a lot of programming worlds, each of which requires a tremendous amount of knowledge for real proficiency. Here are the three I personally know best:

- MFC/C++/Windows
- VBScript/ASP
- Visual Basic

Regexp
an abbreviation for regular expressions. In computing, regular expressions provide a concise and flexible means for identifying strings of text of interest, such as particular characters, words, or patterns of characters

All, basically, what you would call Windows programming. Yes, I've written Unix code and Java code, but not very much. My proficiency in Windows programming comes from knowing not just the basic technologies but also the whole supporting infrastructure. So, I claim, I'm really good at Windows programming because I also know COM, ATL, C++, 80x86 Assembler, Windows APIs, IDispatch (OLE Automation), HTML, the DOM, the Internet Explorer object model, Windows NT and Windows 95 internals, LAN Manager and NT networking, including security (ACEs, ACLs, and all that stuff), SQL and SQL Server, Jet and Access, JavaScript, XML, and a few other cheerful facts about the square of the hypotenuse. When I can't get the StrConv function in VB to do what I want, I bang out an COM control so I can drop into C++ with ATL and call the MLang functions without dropping a beat. It took me years to get to this point.

There are lots of other programming worlds. There's the world of people developing for BEA Weblogic who know J2EE, Oracle, and all kinds of Java things that I don't even know enough about to enumerate. There are hard core Macintosh developers who know CodeWarrior, MPW, Toolbox programming in System 6 through X, Cocoa, Carbon, and even nice obsolete things like OpenDoc that don't help any more.

Very few people, though, know more than one or two worlds, because there's just so much to learn that unless you have to work in one of these worlds for more than a couple of years, you don't really grok it all.

But learn you must

People get kind of miffed when they go on job interviews and get rejected because, for example, they don't have Win32 (or J2EE, or Mac programming, or whatever) experience. Or they get annoyed because idiot recruiters, who would not know an MSMQ if it bit them in the tailbone, call them up and ask if they "have 5 years MSMQ."

Until you've done Windows programming for a while, you may think that Win32 is just a library, like any other library, you'll read the book and learn it and call it when you need to. You might think that basic programming, say, your expert C++ skills, are the 90% and all the APIs are the 10% fluff you can catch up on in a few weeks. To these people I humbly suggest: times have changed. The ratio has reversed.

Very few people get to work on low level C algorithms that just move bytes around any more. Most of us spend all our time these days calling APIs, not moving bytes. Someone who is a fantastic C++ coder with no API experience only knows about 10% of what you use every day writing code that runs on an API. When the economy is doing well, this doesn't matter. You still get jobs, and employers pay the cost of your getting up to speed on the platform. But when the economy is a mess and 600 people apply for every job opening, employers have the luxury of choosing programmers who are already experts at the platform in question. Like programmers who can name four ways to FTP a file from Visual Basic code and the pros and cons of each.

The huge surface area of all these worlds of programming leads to pointless flame wars over whose world is better. Here's a smug comment somebody anonymously made on my discussion board:

"Just one more reason why I'm glad to be living in the 'free world.' Free as in speech (almost) and freedom from pandering to things like setup programs and the registry - just to name a few."

I think this person was trying to say that in the Linux world they don't write setup programs. Well, I hate to disappoint you, but you have something just as complicated: imake, make, config files, and all that stuff, and when you're done, you still distribute applications with a 20KB INSTALL file full of witty instructions like "You're going to need zlib" (what's that?) or "This may take a while. Go get some runts." (Runts are a kind of candy, I think.) And the registry -- instead of one big organized hive of name/value pairs, you have a thousand different file formats, one per application, with .whateverrc and foo.conf files living all over the place. And emacs wants you to learn

VBScript
an Active Scripting language developed by Microsoft

how to program lisp if you're going to change settings, and each shell wants you to learn its personal dialect of shell script programming if you want to change settings, and on and on.

GUI
(Graphical User Interface)
a type of user interface which allows people to interact with a computer and computer-controlled devices

People who only know one world get really smarmy, and every time they hear about the complications in the other world, it makes them think that their world doesn't have complications. But they do. You've just moved beyond them because you are proficient in them. These worlds are just too big and complicated to compare any more. Lord Palmerston: "The Schleswig-Holstein question is so complicated, only three men in Europe have ever understood it. One was Prince Albert, who is dead. The second was a German professor who became mad. I am the third and I have forgotten all about it." The software worlds are so huge and complicated and multifaceted that when I see otherwise intelligent people writing blog entries saying something vacuous like "Microsoft is bad at operating systems," frankly, they just look dumb. Imagine trying to summarize millions of lines of code with hundreds of major feature areas created by thousands of programmers over a decade or two, where no one person can begin to understand even a large portion of it. I'm not even defending Microsoft, I'm just saying that big handwavy generalizations made from a position of deep ignorance is one of the biggest wastes of time on the net today.

Frequent readers, by now, have noticed that I've been thinking of the problem of how one might deliver an application on Linux, Macintosh, and Windows without paying disproportionately for the Linux and Macintosh versions. For this you need some kind of cross-platform library.

Java attempted this but Sun didn't grok GUIs well enough to deliver really slick native-feeling applications. Like the space alien in Star Trek watching Earth through a telescope, they knew exactly what human food was supposed to look like but they didn't realize it was supposed to taste like something. Java apps have menus in the right places but there are all these keyboard things that don't work the same way as every other Windows app and their tabbed dialogs look a little scary. And there is no way, no matter how hard you try, to make their menubars look exactly like Excel's menubars. Why? Because Java doesn't give you a very good way to drop down to the native facilities whenever the abstraction fails. When you're programming in AWT, you can't figure out the HWND of a window, you can't call the Microsoft APIs, and you certainly can't intercept WM_PAINT and do it differently. And Sun made it plenty clear that if you tried to do that, you weren't Pure. You were Polluted, and to hell with you.

After a number of highly publicized failures to build GUIs with Java (e.g. Corel's Java Office suite and Netscape's Javagator), enough people know to stay away from this world. Eclipse built their own windowing library from the ground up using native widgets just so they could write Java code that had a reasonably native look and feel.

The Mozilla engineers decided to address the cross platform problem with their own invention called XUL. So far, I'm impressed. Mozilla finally got to the point where it tastes like real food. Even my favorite bugaboo, Alt+Space N to minimize a window, works in Mozilla; it took them long enough but they did it.

Mitch Kapor, who founded Lotus and created 123, decided for his next application to go with something called wxWindows and wxPython for cross platform support.

Which is better, XUL, Eclipse's SWT, or wxWindows? I don't know. They are all such huge worlds that I couldn't really evaluate them and tell. It's not enough to read the tutorials. You have to sweat and bleed with the thing for a year or two before you really know it's good enough or realize that no matter how hard you try you can't make your UI taste like real food. Unfortunately, for most projects, you have to decide on which world to use before you can write the first line of code, which is precisely the moment when you have the least information. At a previous job we had to live with some pretty bad architecture because the first programmers used the project to teach themselves C++ and Windows programming at the same time. Some of the oldest code was written without any comprehension of event-driven programming. The core string class (of course, we had our own string class) was a textbook example of all the mistakes you could make in designing a C++ class. Eventually we cleaned up and refactored a lot of that old code but it haunted us for a while.

So for now, my advice is this: don't start a new project without at least one architect with several years of solid experience in the language, classes, APIs, and platforms you're building on. If you have a choice of platforms, use the one your team has the most skills with, even if it's not the trendiest or nominally the most productive. And when you're designing abstractions or programming tools, go the extra mile to make them leak proof.

ASP (Action server pages)
Microsoft's first server-side script engine for dynamically-generated web pages

Exercises

Pre-reading questions

1. What does the phrase "a good programmer" mean to you?
2. What abilities/skills should a good programmer have?
3. Api or library? Which is more important?
4. Give some examples of the main programming areas.
5. Which programming languages are important now?

Comprehension questions

1. What is a "programming world"?

A programming area like windows api, networking, multimedia, games, etc.

2. Which IT areas is the author proficient in?

Windows and web programming

3. According to the article, what is the 90/10 rule?

- You can learn 90% of what you use day by day with a week of learning. But the other 10% might take you a couple of years of catching up. Or the second version: 90% of programming skills is the knowledge of different Api's; 10% is the languages

4. Why was the Java GUI system not very popular?

- Because the Sun company didn't put the proper "taste" in it, it isn't as native as Windows controls Because ultimately it was not compatible with native Windows facilities. The explanation is in the paragraph beginning: 'Java attempted this ...'.

5. What should every good programming team include?

- There should be at least one very experienced programmer.

Further exercises:

Match the languages to the technologies.

- C++ with Windows, JavaScript with Web, etc

Possible topics for discussion

1. How do you see the future of programming?
2. What kind of programming experience is most important in the job market?

Reading suggestions

Joel's blog is a well-known source of articles about programming and programming culture. It is certainly worth reading, as are his books.

Quake - Game Engine

Wikipedia

Number of words	700
Computer science content	Medium
Math content	Low
English language complexity	Medium

Learning objectives

- to understand pre-rendering process
- to acquire technical vocabulary associated with computer graphics
- to recognize differences between true 3D games and the earliest 2D or pseudo-3D games
- to learn why Quake became a milestone in game history

Sub-areas covered

- Computer graphics
- Computer games
- Computer animation
- Game engine

Keywords

- **rendering** - the process of generating an image from a three dimensional object by means of computer programs
- **to prune** - to cut parts of the map invisible for the player
- **skybox** - a method to create a background to make a computer and video game location look bigger than it really is
- **sector-based static lighting** – the texture with information about lighting which is not affected by the angle or position of the light source
- **CAD** - Computer Aided Design

Summary

The text describes new solutions to graphical problems in computer games delivered in "Quake". It introduces the idea of displaying only those elements which are visible to the player and calculating the light depending on its angle. The article explains the "mouselook option", which appears in the game.

Tekst opisuje nowe rozwiązania problemów związanych z grafiką w grach komputerowych, wprowadzone w kultowej grze "Quake". Przybliża ideę wyświetlania tylko tych elementów, które są bezpośrednio widoczne dla gracza, wprowadza w zagadnienie obliczania światła w zależności od jego kąta padania. W tekście zaznaczono, że "Quake" był jedną z pierwszych gier wykorzystujących sprzęt do wspierania grafiki trójwymiarowej, a także przedstawiono jej wpływ na proces projektowania gier komputerowych. Artykuł wyjaśnia także, na czym polega pojawiająca się w grze opcja "mouselook".

Pre-reading questions

What you know about the following:

1. the difference between true 3D and "2.5D" games
2. optimizing the rendering process
3. preprocessing the rendered world
4. hardware acceleration
5. various problems in games modelling and scientific design

Quake - game engine

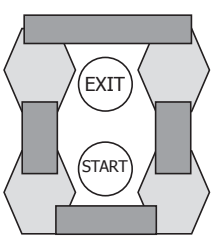
Quake popularized several major advances in the 3D game genre: it uses 3-dimensional models for players and monsters instead of 2-dimensional sprites; and the world in which play takes place is created as a true 3-dimensional space, rather than a 2-dimensional map. Some previous 3D games, such as Duke Nukem 3D, Doom and Wolfenstein 3D used some mathematical tricks to simulate the 3-dimensional world. This allowed a true 3D view, but was so restricted that only looking straight ahead would give a sense of 3 dimensions.

rendering
the process of generating an image from a three dimensional object by means of computer programs

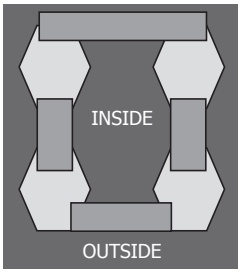
Reducing 3D complexity to increase speed

Quake was the first true 3D game to use a special map design system that preprocessed and pre-rendered the 3D environment to reduce the load when playing the game on the 50-75 MHz CPUs of the time. The 3D environment in which the game takes place is referred to as a map, even though it is three-dimensional in nature rather than a flat 2D space. The map editor program uses a number of simple convex 3D geometric objects known as brushes that are sized and rotated to build the environment. The brushes are overlapped in order to create an enclosed, empty, volumetric space, and when the design is complete the map is run through the rendering preprocessor. The preprocessor is used to locate two types of empty space in the map, the empty space enclosed by brushes where the game will be played, and the other empty space outside the brushes that the player will never see. The preprocessor then strips away the back-faces of the individual brushes which are outside the game-space, leaving only the few polygons that define the outer perimeter of the enclosed game space.

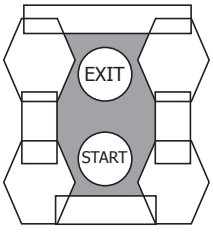
to prune
to cut parts of the map invisible for the player



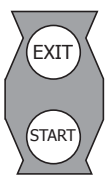
1. Use brushes to make an enclosed play area



2. Preprocessor finds the inside and outside spaces



3. Preprocessor cuts away faces not visible inside maps



4. Finished low-polygon, fast-rendering game map

skybox
a method to create a background to make a computer and video game location look bigger than it really is

It was possible to edit a processed map by opening it in a special vertex editor and editing the raw vertex data, or to add or remove individual triangle faces. Though difficult, this technique was occasionally used by cheaters to create windows in walls, to see normally hidden enemies approaching from behind doors and walls, and resulted in an anti-cheat mechanism used in recent 3D games that calculates a checksum for each file used in the game, to detect players using potentially hacked map files.

Once the map had been pruned of excess faces that the player inside will never see anyway, the polygon count can be reduced by 50% to 80% compared to an original unprocessed map. On the 50-75 MHz PCs of the time, it was common for this pruning step to take many hours to complete on a map, often running overnight if the map design was extremely complex.

sector-based static lighting
the texture with information about lighting which is not affected by the angle or position of the light source

Hardware 3D acceleration

Quake was also one of the first games to support 3D hardware acceleration. While initially released with only software rendering, John Carmack created a version of the Quake executable that took advantage of Rendition's Vérité 1000 graphics chip. OpenGL support was soon added in the form of the GLQuake executable for Windows 95 and higher. Many believe that this kick-started the independent 3D graphics card revolution, "GLQuake" being the first application to truly demonstrate the capabilities of the 3dfx "Voodoo" chipset at the time. The only two other cards capable of rendering GLQuake were a professional (very expensive) Integraph 3D OpenGL card, and, later, the PowerVR cards.

CAD
Computer Aided Design

Impact on modern game design

Nearly all the games created after Quake have used this 3D preprocessing optimization, which enhanced the speed of the game on a personal computer or gaming console. 3D games are therefore able to push the limits of visual styles and effects because so much excess modelling data was stripped out before the end-user ever saw the game.

In this way, most games are significantly different from professional 3D CAD and design problems, where there is no time available to do preprocessing between an engineer making a change and seeing it on the screen. Nothing can be thrown away to increase the rendering speed of a 3D engineering model, since any part of the design can change at any moment. For this reason, professional 3D graphics cards are significantly more expensive and powerful than the 3D cards used in home computers simply for playing games, because the professional 3D card needs far greater processing power to deal with the full complexity of an un-preprocessed 3D render space.

Exercises

Pre-reading questions

What you know about the following:

1. The difference between true 3D and "2.5D" games.
- using 2-dimensional sprites instead of 3D models
- using some mathematical tricks to simulate the 3-D world
2. Optimizing the rendering process.
- avoid rendering unnecessary polygons
- avoid rendering invariable polygons
- take care on number of details
3. Preprocessing the rendered world
- reducing number of polygons, which should be displayed
- increase quality of frame using some mathematical tricks, just before the frame is showed to user
4. Hardware acceleration
- using special graphic chips to increase speed of rendering scenes
- Quake was first game to support hardware acceleration
5. Various problems in games modelling and scientific design
- time available to render prepared scenes
- the oldest problem – graphics quality/application speed
- optimization

Comprehension questions

1. What did "shooter" games look like before the Quake revolution?

- There were 2-dimensional sprites instead of full 3D models and 2D maps. In Quake, a player can move to each direction; earlier games used mathematical tricks to simulate a 3D world.

2. What is graphic preprocessing in general?

- Preparing a picture before it is shown. The main aim of this process is to improve the efficiency and quality of the rendered picture.

3. How could players see their opponents through a wall?

- It was possible to edit a processed map by opening it in a special vertex editor and editing the raw vertex data, or to add or remove individual triangle faces.

4. What does the Quake pre-rendering process depend on?

- The map editor program uses a number of simple convex 3D geometric objects known as brushes that are sized and rotated to build the environment. The brushes are overlapped in order to create an enclosed, empty, volumetric space, and when the design is complete the map is run through the rendering preprocessor. The pre-

processor is used to locate two types of empty space in the map, the empty space enclosed by brushes where the game will be played, and the other empty space outside the brushes that the player will never see. The preprocessor then strips away the back-faces of the individual brushes which are outside the game-space, leaving only the few polygons that define the outer perimeter of the enclosed game space.

Possible topics for discussion

1. Which games, in your opinion, were a milestone in game development and why?

Mechanical Models of Artery Walls

Chapter 3: History of Artery Wall Modelling

Piotr Kalita and Robert Schaefer

Number of words	860 (selected part)
Computer science content	Low
Math content	Low
English language complexity	High

Learning objectives

- to understand the basic structure of the cardiovascular system
- to learn the terminology associated with the cardiovascular system
- to learn about mechanical models of artery walls
- to learn about possible application of computer science in medicine

Main sub-areas covered

- cardiovascular models

Keywords

- **artery** - one of the tubes that carries blood from the heart to the rest of the body
- **Windkessel** - (air kettle), one of the models presented in the article
- **ODE** - Ordinary Differential Equations
- **PDE** - Partial Differential Equations
- **heterogenous** - consisting of various parts that are very different from each other

Summary

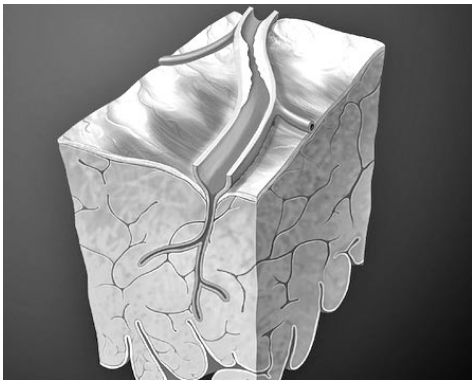
This is the third chapter of an article on mechanical models of artery walls, presenting a short review of artery wall modeling history. It starts with the derivation of the word artery. Next, the authors present models of blood circulation, describing their characteristics and their advantages and disadvantages.

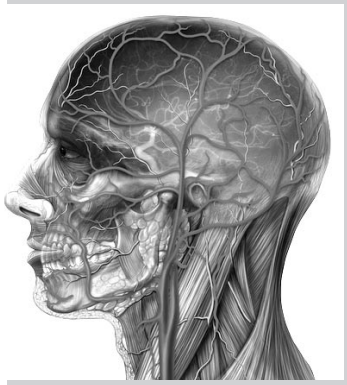
Jest to trzeci rozdział pracy poświęconej mechanicznemu modelowaniu ścianek tętnic, w którym autorzy skoncentrowali się na krótkim przedstawieniu historii tego zagadnienia. Ta część rozpoczyna się od wyprowadzenia znaczenia słowa artery (tętnica), a następnie zaprezentowane zostają kolejne modele układu krwionośnego wraz z ich opisami.

Pre-reading exercises

1. Draw a simple model of the cardiovascular system and name its main parts (arteries, veins, heart etc.)
2. Think about a model of the system as tree with the heart as its root

History of Artery Wall Modelling





The word artery comes from Greek, it means air pipe (aer is air and terein means to keep in Greek) as in ancient Greece it was believed that arteries carry air. This was corrected by Galen in the 2nd century AD but he still did not recognize that blood circulates and he also did not connect the pulse in arteries with the beating of the heart. The circulation of blood and the role of the heart as a pump was discovered in the 18th century by William Harvey. However, the first to consider the elasticity of artery walls and its significance in the pulsatile character of blood flow was Stephen Hales (1733), who originated the Windkessel (air kettle) theory later developed by Otto Frank near the end of, the 19th century (Otto Frank is actually the originator of the name Windkessel). In this theory, the artery walls serve as a reservoir of energy, which is stored in their deformation during the systole by the stretching of the walls and then, during the diastole, is restored to the blood, which is thereby pumped to the capillaries in a constant and non- pulsatory way (as it is from the heart to the arteries). Pure Windkessel theory leads to ODEs that do not take spatial relations into account. In particular, they do not reflect the fact that due to its wavelike character, the pulse propagates along the arterial tree and is able to reflect. Further development of artery wall models followed the development of mechanics and throughout the 19th century important contributions to the field were added by Young, Poiseuille, Hagenbach, Moens and Korteweg.

PDE models which take into account the dependence of variables on both space and time have been in existence since the 19th century; however, their use has been boosted by the use of computers and the increase in their computational capabilities as well as the development of medical measurement techniques from the 1940s to the present day.. (Timmons [131] claims that "we can expect highly detailed models emerging within the next few years as parallel computers become economical"). Two developments important for artery wall modeling took place in the 50s and 60s: modern nonlinear mechanics, the theoretical framework most commonly used for artery models today; and the Finite Element Method (FEM), which is used to formulate relatively simple algorithms for numerical solutions of the underlying problems. From the 60s to the 80s various models based on nonlinear mechanics were used to

artery
one of the tubes
that carries blood
from the heart
to the rest of the body

Windkessel (air kettle)
one of the models
presented in the article

reflect artery wall behavior. The leading researcher in this area was (and still is) Yuan Cheng Fung, who is known as the father of biomechanics.

Windkessel based models, being simple and fairly accurate, remain popular for global considerations, although one has to keep in mind that they are insufficient for quantitative physiological predictions. For local approaches, on the other hand, complex cardiovascular PDE models are used. PDE models range from the 1D models in which the unknown is the cross section area of the vessel (see Sect. 4 in this article) to the most detailed 3D models, by which one can find the distribution of displacement of the wall in time and space (see Sects. 7 and 8). Often PDE systems are considered with equations of hydrodynamics for flow coupled with equations of elasticity for the wall. Such approaches are the most accurate but the underlying problems are very complex: equations are highly nonlinear, difficult in theoretical considerations, and require very large computational power for numeric solution.

ODE
Ordinary
Differential Equations

PDE
Partial
Differential Equations

The complex 3D models are also sensitive to parameters like the geometry of the domain, initial and boundary conditions and physical parameters of the equations and all these quantities are still difficult to determine using present measurement techniques. Furthermore, the development of models is limited by the complexity of the underlying physiology. New physiological phenomena are constantly being discovered, including sophisticated control and self-regulation mechanisms, the presence of stresses and strains in unloaded arteries (residual stresses) and the structure and function of various components of the wall which determine its heterogeneous and anisotropic behavior. Models reflecting these phenomena include a large number of parameters and one must keep in mind that fitting the models to the measurement data in such cases is susceptible to mistakes and, furthermore, it is sometimes possible that a model which is actually inaccurate can be made to fit the tests in some limited measurement regime.

heterogenous
consisting of various parts that are very different from each other

Because of these limitations, the simulations which are global (i.e. involve relations between various organs of the body) use only simplified cardiovascular models (like the Windkessel or its extensions). On the other hand, local specialized models are available for specialized purposes only: for instance they are used in the design of stents and grafts as well as the prediction and treatment of atherosclerosis Among the complex phenomena which have been intensively investigated and incorporated into cardiovascular models in the last ten years are:

– including the residual stresses
– modelling the fluid structure interaction
– considering the tissue anisotropy and the presence of protein fibres
– considering the heterogeneity and layered structure of the wall
– investigating the multiscale models which involve coupling the simple 1d PDE or ODE models with complex 3d models (see [109]).

Exercises

Pre-reading questions

1. Draw a simple model of the cardiovascular system and name its main parts (arteries, veins, heart etc.)
2. Think about a model of the system a tree with the heart as its root

Comprehension questions

1. What is the meaning of the words systole and diastole?

- the contraction of heart chambers, driving blood out of the chambers
- the period of time when the heart relaxes after systole (contraction)

2. What do ODE and PDE stand for (as they're not explained in the article)?

- ODE - Ordinary Differential Equations
- PDE - Partial Differential Equations

3. What is the role of the artery walls in the Windkessel theory?

- a reservoir of energy

Possible topics for discussion

1. Do you think it is possible to create a mathematical model of the cardiovascular system that will perfectly match the real one?

Possible difficulty

A difficult text as far as English goes but with relatively low computer science content; requires minimum medical knowledge.

Further reading

The whole article is located on English++ webpage.

The Pitch Correction Algorithm: an Overview

Robert Ahlfinger, Brenton Cheeseman, Patrick Doody

Number of words	1100
Computer science content	High
Math content	High
English language complexity	High

Learning objectives

- to acquire advanced mathematical vocabulary
- to develop reading skills by identifying the most relevant information,
- to become familiar with scientific jargon

Sub-areas covered

- Sound modification
- Wave modification
- Sound clearing

Keywords

- **DFT** - Discrete Fourier Transform
- **FFT** - Fast Fourier transform
- **matrix** - a rectangular table of elements (or entries), which may be numbers or, more generally, any abstract quantities that can be added and multiplied
- **matricize** - changing sound into parts and writing into a matrix

Summary

This is an introduction to the specific techniques involved in the creation of a pitch corrector.

Artykuł jest wprowadzeniem do technik związanych z tworzeniem modyfikatora wysokości dźwięku.

Pre-reading questions

1. What kinds of digital music formats are you familiar with?
2. Do you know how digital audio is stored?
3. How do you understand the word "pitch"?

The Pitch Correction Algorithm: An Overview

1 Time-Domain vs. Frequency-Domain

Clearly, the goal of this algorithm is to take an input voice signal, change the pitch of the voice, and output the otherwise unaltered signal. In order to do so, the first step is to decide whether to analyze and manipulate these signals in the time domain or the frequency domain. Because our algorithm is primarily concerned with quickly identifying and shifting individual frequencies, we worked solely in the realm of the frequency domain. Of course, there are effective ways to deal with this problem without the frequency domain. However, as will later become obvious, there are some very useful techniques we developed that are not possible in the time domain. With pitch correction it seems that Parseval has made a mistake; there is simply more power in the spectrum.

matrix
a rectangular table of elements (or entries), which may be numbers or, more generally, any abstract quantities that can be added and multiplied.

2 Basic System Model

Now that we have decided how to look at our signals, we need to develop a general layout and strategy for how it will work.

2.1 General Process Summary

First, the signal is Matricized, a term we coined to describe our particular algorithm to break up the signal and convert it into the Fourier Domain. Basically, the signal comes in as a long string of sampled values that together represent the whole sound. We, in turn, convert this vector of samples into a matrix for which each column represents the spectrum of one slice, or chunk, of the signal. Although any chunk size could be used, we found the best performance with chunk sizes of 512 samples, which represents about .02 seconds of sound for the 22 kHz sampling rate used on our signals. Next, we take the Discrete Fourier Transform for each of the chunks, showing us the frequencies present at every given moment during the speech. These DFTs are then collected into a matrix with 512 rows and as many columns as there are .02 second long chunks in the voice. With a given chunk, our Harmonic Detection algorithm has the extremely difficult task of accurately and consistently identifying the first harmonic of the voice. With that information in hand, the program reconstructs a new DFT representation for the current chunk by first sliding the first harmonic down the spectrum by the desired shift in pitch, and then following up with all of the other harmonics, shifting each one by an incremental multiple of the first shift. After all of the chunks have been processed and put into a matrix, this new matrix is dematricized in order to convert the information back into the time domain as a new string of digital samples that represent the freshly manipulated voice.

DFT
Discrete
Fourier Transform

3 Detailed System Model: Step-by-Step

The pitch synthesizer relies on several algorithms to properly alter the pitch of a person's voice without mutilating its clarity.

3.1 Matricize

First, the signal is matricized, a term we coined to represent the task of transforming the string of speech samples into a matrix whose columns each represent the spectrum of an overlapping rectangular window, or chunk, of the signal. Each portion of the voice is contained twice in this information since exactly one half of each chunk is overlapped and contained within an adjacent chunk. Next, each column of the matrix is processed separately, meaning we attempt to change the characteristics of the voice one piece at a time and do so redundantly.

3.2 Harmonic Detection

FFT
Fast Fourier transform

Now that we have isolated the spectrum of a chunk of our signal, we use a harmonic detector to find the first harmonic of the voice at that particular point in time. This task is harder than it first appears and its level of accuracy makes the single biggest contribution to the functionality and accuracy of the pitch synthesizer as a whole. Voiced vowel noises are the only parts of speech that contain pitch, so they need to be processed differently than the rest of the signal. However, since there are many periods of noise as well as voiced (s and z sounds) and unvoiced (like f and t) fricatives alongside these important voiced vowel noises, the harmonic detector must wade through each chunk and first determine whether or not it is dealing with a voiced vowel noise. If so, it computes the index of the first harmonic of the sample by taking the DFT of the first half of the magnitude of the DFT of the original signal chunk. The resulting spectrum will have a very large DC component which represents the grab bag of frequencies present in the original signal, as well as repeating peaks corresponding to the only periodic aspect of the original DFT the signal's harmonics. Therefore, the harmonic detector compares the DC amplitude with the next biggest peak, determining simultaneously whether or not this chunk is likely to be a voiced vowel noise and if so the frequency of its first harmonic.

3.3 Frequency Shift

With this information in hand, our program determines how far each and every frequency must be shifted. Since you interpret the pitch of a voice as the frequency of its first harmonic, the first harmonic is shifted by exactly the desired result. In turn, the frequency of every harmonic is a multiple of the first, so the second harmonic must be shifted twice as far as the first; the third is shifted three times as far, and so

matricize
changing sound into parts and writing into a matrix

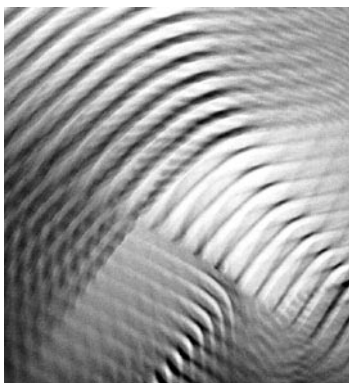
on. In fact, we use the index of the first harmonic to determine how much each and every frequency in the original chunk will shift to build up the first half of the DFT for our new, processed chunk. We are trying to alter the pitch without affecting the length of the sound, so this stretched out DFT must be cut off at half the length of the original DFT, at which point we have the completed version of the front half of the new DFT. To complete the second half, we rely on the DFT's symmetry properties, noting that our original and final sound signals are both purely real. Therefore, the real portion of the DFT is mirrored about the middle, and the imaginary portion is mirrored and flipped. Finally we have completely processed the given window of the original signal.

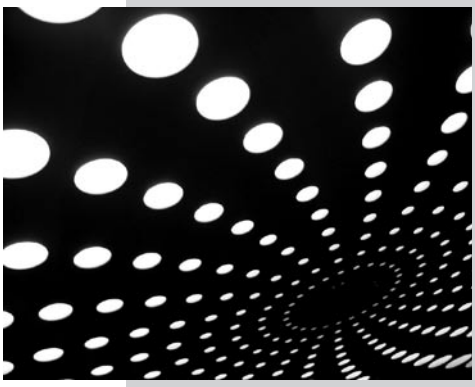
3.4 Reconstruction

To reconstruct the original signal, these processed DFT's each become a column of another matrix which is then dematricized by taking the inverse FFT of each spectrum and placing them side by side into a new signal that has the same length as the original. The only difference of course being that the voice in the signal has become as high or as low as the desired shift.

Pitch Shifted Speech Examples

Unaltered voice	Original
Pitch Shifted Voice	Up Down





Exercises

Pre-reading questions

1. What kinds of digital music formats are you familiar with?

- Kinds of digital music which I am familiar with are as follows: wav, ogg, mpc, flac, aiff, raw, au, gsm, dct, vox, acc, mp4/m4a, mp3, wma, wav, ra, ram, dss, msv, dvf, mp4.

2. Do you know how digital audio is stored?

- Though a WAV file can hold compressed audio, the most common WAV format contains uncompressed audio in the pulse-code modulation (PCM) format. PCM audio is the standard audio file format for CDs, containing two channels of 44,100 samples per second, 16 bits per sample. Since PCM uses an uncompressed, lossless storage method, which keeps all the samples of an audio track, professional users or audio experts may use the WAV format for maximum audio quality. WAV audio can also be edited and manipulated with relative ease using software.

3. How do you understand the word "pitch"?

- to set at a particular pitch, or determine the key or keynote of (a melody),
- the degree of height or depth of a tone or of sound, depending upon the relative rapidity of the vibrations by which it is produced,
- the particular tonal standard with which given tones may be compared in respect to their relative level,
- apparent predominant frequency sounded by an acoustic source.

Comprehension questions

1. Explain these words:

- to (de)matricize
  to change linear into matrix
- harmonic
  pertaining to harmony, as distinguished from melody and rhythm; an acoustical frequency that is higher in frequency than the fundamental.

2. What is the pitch correction algorithm used for?

- It is used for changing the pitch of voice signals.

3. What are the only parts of speech that contain pitch?

- Vowels.

4. Explain the abbreviations DFT, FFT, DC

- DFT - Discrete Fourier Transform
- FFT - Fast Fourier Transform
- DC - Direct Current

Possible topics for discussion

1. In what kinds of situations do you think this algorithm might be used?

- Pitch correction has numerous applications and is commonly used to add harmony to certain words or phrases without re-recording those words or phrases again and again at the necessary pitches. Depending on the specific model used, various vocal effects can be added

Possible difficulties

The article contains a lot of field specific information.

Software Development Process

Wikipedia

Number of words	2330
Computer science content	High
Math content	Low
Business content	Medium
English language complexity	High

Learning objectives

- to recognize differences between the Iterative and Waterfall Process of software development
- to be able to give examples of the Iterative Process
- to understand the whole process from the customer's order to implementation
- to acquire basic vocabulary linked with the software development process

Sub-areas covered

- Software engineering
- Software development methodologies

Keywords

- **software** - in general, computer programs
- **hardware** - all the physical parts of the computer
- **developer** - person who develops the software
- **agile development** - a way of building software
- **code - (coding)** refers to the process of transforming a concept into a program code
- **maintenance** - "looking after" the written software
- **bug** - an error in software, something that spoils the flow of the program
- **up-front** (design, analysis, requirements) - the oldest method of building software

Summary

The text deals with the problem of software development. It starts with a short description of some standards (methodologies) that were introduced to unify the process of creating software. In a section entitled: "The Main Steps of Software Development", the reader is introduced to the typical steps that a team follows while developing software. For each step in the process, there is a brief description of its basic characteristics. Then the text describes two different approaches to

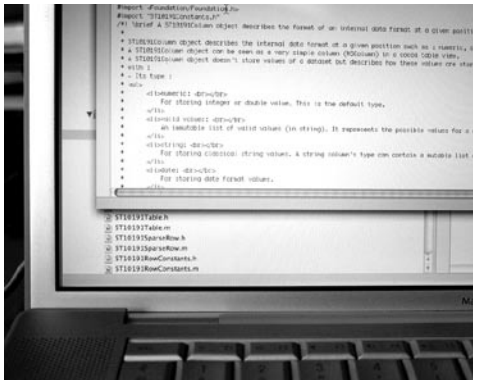
software development, in which the previously listed steps are applied. The final part discusses the iterative development process and its three main types. This section also presents some advantages and disadvantages of the iterative process compared with the "Waterfall Process" model of design, described earlier in the text.

Tekst dotyczy problematyki tworzenia oprogramowania. Na wstępie, opisane są standardy (metodologie) jakie zostały wprowadzone by ujednolicić ten proces. W następnej części, zatytułowanej "Main Steps Of The Software Development", przybliżone zostają główne kroki, którymi zwykle podąża zespół tworzący oprogramowanie. Kroki te opatrzone są krótkimi opisami ich cech lub problemów, jakie ze sobą niosą. Dwie ostatnie części tekstu opisują dwa różne podejścia do tworzenia oprogramowania, w których wykorzystane są opisane wcześniej kroki. W ostatniej części, opisującej iteracyjny proces projektowania przedstawione są trzy spośród jego najważniejszych odmian (Agile Software Development, Extreme Programming i Test Driven Development). Część ta przedstawia także niektóre zalety oraz wady iteracyjnego procesu projektowania w zestawieniu w modelem projektowania "Waterfall Processes" opisanym wcześniej.

Pre-reading questions

1. What kind of job do you see yourself holding five years from now?
2. Do you know any companies developing software? Would you like to join any of them?
3. How do you imagine work is organized in these companies? What software development methods do you know?
4. Have you used any of these methods so far?

Software Development Process



A software development process is a structure imposed on the development of a software product. Synonyms include software life cycle and software process. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process.

A growing body of software development organizations implement process methodologies. Many of them are in the defense industry, which in the U.S. requires a rating based on 'process models' to obtain contracts. The international standard for describing the method of selecting, implementing and monitoring the life cycle for software is ISO 12207.

software
in general
computer programs

The Capability Maturity Model (CMM) is one of the leading models. Independent assessments grade organizations on how well they follow their defined processes, not on the quality of those processes or the software produced. CMM is gradually being replaced by CMMI. ISO 9000 describes standards for formally organizing processes with documentation.

ISO 15504, also known as Software Process Improvement Capability Determination (SPICE), is a "framework for the assessment of software processes". This standard is aimed at setting out a clear model for process comparison. SPICE is used much like CMM and CMMI. It models processes to manage, control, guide and monitor software development. This model is then used to measure what a development organization or project team actually does during software development. This information is analyzed to identify weaknesses and drive improvement. It also identifies strengths that can be continued or integrated into common practice for that organization or team.

Six Sigma is a methodology to manage process variations that uses data and statistical analysis to measure and improve a company's operational performance. It works by identifying and eliminating defects in manufacturing and service-related processes. The maximum permissible defects is 3.4 per one million opportunities. However, Six Sigma is manufacturing-oriented and needs further research on its relevance to software development.

The Main Steps of Software Development

Domain Analysis

Often the first step in attempting to design a new piece of software, whether it be an addition to an existing software, a new application, a new subsystem or a whole new system, is, what is generally referred to as "Domain Analysis". Assuming that the developers (including the analysts) are not sufficiently knowledgeable in the subject area of the new software, the first task is to investigate the so-called "domain" of the software. The more knowledgeable they are about the domain already, the less the work required. Another objective of this work is to get the analysts (who will later try to elicit and gather the requirements from the area) experts or professionals to speak with them in the domain's own terminology and to better understand what is being said by these people, otherwise they will not be taken seriously. So, this phase is an important prelude to extracting and gathering the requirements. The following quote captures the kind of situation an analyst who hasn't done his homework well may face in speaking with a professional from the domain: "I know you believe you understood what you think I said, but I am not sure you realize what you heard is not what I meant."

hardware
all the physical parts of the computer

Software Elements Analysis

The most important task in creating a software product is extracting the requirements. Customers typically know what they want, but not what software should do, while incomplete, ambiguous or contradictory requirements are recognized by skilled and experienced software engineers. Frequently demonstrating live code may help reduce the risk that the requirements are incorrect.

Specification

Specification is the task of precisely describing the software to be written, possibly in a rigorous way. In practice, most successful specifications are written to understand and fine-tune applications that were already well-developed, although safety-critical software systems are often carefully specified prior to application development. Specifications are most important for external interfaces that must remain stable.

Software architecture

The architecture of a software system refers to an abstract representation of that system. Architecture is concerned with making sure the software system will meet the requirements of the product, as well as ensuring that future requirements can be addressed. The architecture step also addresses interfaces between the software

system and other software products, as well as the underlying hardware or the host operating system.

Implementation (or coding)

Reducing a design to code may be the most obvious part of the software engineering job, but it is not necessarily the largest portion.

Testing

Testing of parts of software, especially where code by two different engineers must work together, falls to the software engineer.

Documentation

An important (and often overlooked) task is documenting the internal design of software for the purpose of future maintenance and enhancement. Documentation is most important for external interfaces.

Software Training and Support

developer
person who develops the software

A large percentage of software projects fail because the developers fail to realize that it doesn't matter how much time and planning a development team puts into creating software if nobody in an organization ends up using it. People are occasionally resistant to change and avoid venturing into an unfamiliar area so, as a part of the deployment phase, it is very important to have training classes for the most enthusiastic software users, shifting the training towards the neutral users intermixed with the avid supporters, and finally incorporate the rest of the organization into adopting the new software. Users will have lots of questions and software problems which leads to the next phase of software.

Maintenance

Maintaining and enhancing software to cope with newly discovered problems or new requirements can take far more time than the initial development of the software. Not only may it be necessary to add code that does not fit the original design but just determining how software works at some point after it is completed may require significant effort by a software engineer. About 60% of all software engineering work is maintenance, but this statistic can be misleading. A small part of that is fixing bugs. Most maintenance is extending systems to do new things, which in many ways can be considered new work.

agile development
a way of building software

Waterfall processes

The best-known and oldest process is the waterfall model, where developers (roughly) follow these steps in order:

- state requirements
- analyze requirements
- design a solution approach
- architect a software framework for that solution
- develop code
- test (perhaps unit tests then system tests)
- deploy
- post-implementation.

After each step is finished, the process proceeds to the next step, just as builders don't revise the foundation of a house after the framing has been erected. There is a misconception that the process has no provision for correcting errors in early steps (for example, in the requirements). In fact this is where the domain of requirements management comes in which includes change control. This approach is used in high risk projects, particularly large defense contracts. The problems in waterfall do not arise from "immature engineering practices, particularly in requirements analysis and requirements management". Studies of the failure rate of the DOD-STD-2167 specification, which enforced waterfall, have shown that the more closely a project follows its process, specifically in up-front requirements gathering, the more likely the project is to release features that are not used in their current form.

Iterative processes

Iterative development prescribes the construction of initially small but ever larger portions of a software project to help all those involved to uncover important issues early before problems or faulty assumptions can lead to disaster. Iterative processes are preferred by commercial developers because it allows a potential of reaching the design goals of a customer who does not know how to define what they want.

Agile software development processes are built on the foundation of iterative development. To that foundation they add a lighter, more people-centric viewpoint than traditional approaches. Agile processes use feedback, rather than planning, as their primary control mechanism. The feedback is driven by regular tests and releases of the evolving software.

Agile processes seem to be more efficient than older methodologies, using less programmer time to produce more functional, higher quality software, but have the drawback from a business perspective that they do not provide long-term planning capability. However, polls show gains, sometimes significant. For example, a sur-

vey, published in August 2006 by Version One and Agile Alliance and based on polling more than 700 companies shows the following benefits of Agile approach (http://www.agilejournal.com/articles/from-the-editor/agile-survey-results%3a-solid-experience-and-real-results.html). The survey was repeated in August 2007 with about 1,700 respondents (http://www.agilejournal.com/articles/from-the-editor/agile-survey-results%3a-widespread-adoption,-emphasis-on-productivity-and-quality.html). Here are some results:

code (coding) refers to the process of transforming a concept into a program code

1. Accelerated time to market.

1. 2006: 10% or higher improvements reported by 86% of respondents
2. 2007: 10% or higher improvements reported by 90% of respondents
3. 2006: 25% or higher improvements reported by 60% of respondents
4. 2007: 25% or higher improvements reported by 54% of respondents

2. Increased productivity.

1. 2006: 10% or higher improvements reported by 87% of respondents
2. 2007: 10% or higher improvements reported by 83% of respondents
3. 2006: 25% or higher improvements reported by 55% of respondents
4. 2007: 25% or higher improvements reported by 55% of respondents

3. Reduced software defects.

1. 2006: 10% or higher improvements reported by 86% of respondents
2. 2007: 10% or higher improvements reported by 85% of respondents
3. 2006: 25% or higher improvements reported by 55% of respondents
4. 2007: 25% or higher improvements reported by 54% of respondents

4. Reduced cost.

1. 2006: 10% or higher improvements reported by 63% of respondents
2. 2007: 10% or higher improvements reported by 28% of respondents
3. 2006: 25% or higher improvements reported by 26% of respondents
4. 2007: 25% or higher improvements reported by 28% of respondents

The stability of results on a broader set of respondents suggests that we see the actual trend.

Among other interesting improvements reported were:

- Enhanced ability to manage changing priorities
- Alignment between IT and business goals
- Improved team morale
- Reduced project risk

There is also an interesting chart at http://versionone.com/Resources/ AgileBenefits. asp that shows Agile development value proposition in comparison to traditional development.

Extreme Programming, XP, is the best-known iterative process. In XP, the phases are carried out in extremely small (or "continuous") steps compared to the older, "batch" processes. The (intentionally incomplete) first pass through the steps might take a day or a week, rather than the months or years of each complete step in the Waterfall model. First, one writes automated tests, to provide concrete goals for development. Next is coding (by a pair of programmers), which is complete when all the tests pass, and the programmers can't think of any more tests that are needed. Design and architecture emerge out of refactoring, and come after coding. Design is done by the same people who do the coding. (Only the last feature - merging design and code - is common to all the other agile processes.) The incomplete but functional system is deployed or demonstrated for (some subset of) the users (at least one of which is on the development team). At this point, the practitioners start again on writing tests for the next most important part of the system.

maintenance
"looking after"
the written software

Test Driven Development (TDD) is a useful output of the Agile camp but raises a conundrum. TDD requires that a unit test be written for a class before the class is written. Therefore, the class firstly has to be "discovered" and secondly defined in sufficient detail to allow the write-test-once-and-code-until-class-passes model that TDD actually uses. This is actually counter to Agile approaches, particularly (so-called) Agile Modeling, where developers are still encouraged to code early, with light design. Obviously, to get the claimed benefits of TDD, a full design down to class and responsibilities (captured using, for example, Design By Contract) is necessary. This counts towards iterative development, with a design locked down, but not iterative design - as heavy refactoring and re-engineering negate the usefulness of TDD.

bug
an error in software,
something that spoils
the flow of the program

While iterative development approaches have their advantages, software architects are still faced with the challenge of creating a reliable foundation upon which to develop. Such a foundation often requires a fair amount of up-front analysis and prototyping to build a development model. The development model often relies upon specific design patterns and entity relationship diagrams (ERD). Without this upfront foundation, iterative development can create long term challenges that are significant in terms of cost and quality.

Critics of iterative development approaches point out that these processes place what may be an unreasonable expectation upon the recipient of the software: that they must possess the skills and experience of a seasoned software developer. The approach can also be very expensive if iterations are not small enough to mitigate risk; akin to... "If you don't know what kind of house you want, let me build you one

up-front (design, analysis, requirements)
– the oldest method of building software

and see if you like it. If you don't, we'll tear it all down and start over". By analogy the critic argues that up-front design is as necessary for software development as it is for architecture. The problem with this criticism is that the whole point of iterative programming is that you don't have to build the whole house before you get feedback from the recipient. Indeed, in a sense conventional programming places more of this burden on the recipient, as the requirements and planning phases take place entirely before the development begins, and testing only occurs after development is officially over.

In fact, a relatively quiet turn around in the Agile community has occurred on the notion of "evolving" the software without the requirements locked down. In the old world this was called requirements creep and never made commercial sense. The Agile community has similarly been "burnt" because, in the end, when the customer asks for something that breaks the architecture, and won't pay for the re-work, the project terminates in an Agile manner.

These approaches have been developed along with web based technologies. As such, they are actually more akin to maintenance life cycles given that most of the architecture and capability of the solutions is embodied within the technology selected as the back bone of the application.

Exercises

Pre-reading questions

1. What kind of job do you see yourself holding five years from now?
- computer programmer/software developer, graphic designer, scientist
2. Do you know any companies developing software? Would you like to join any of them?
- IBM, Google, Microsoft, Sabre, Motorola, Onet, Interia
3. How do you imagine work is organized in these companies? What software development methods do you know?
- methods: for example, iterative (extreme programming)
4. Have you used any of these methods so far?

Comprehension questions

1. **What is the idea behind Test Driven Development?**
- Writing tests for all the steps of software development, from planning to coding.
2. **What are the main benefits of using Agile Methods?**
- Better productivity, reduction of software defects, cost reduction, reduced project risk.
3. **What is the main aim of introducing the SPICE standard?**
- To simplify the comparison of different development models.
4. **What is the aim of utilizing the Six Sigma methodology, and how can it help the performance of some kinds of companies?**
- The aim of using the Six Sigma methodology is to improve the performance of a company. The use of this methodology can identify and eliminate defects in manufacturing and service-related processes.
5. **Which of the software development steps involve most contact with the customer? (possible topic for discussion)**
- Software elements analysis, Specification, Software training and support.
6. **What kind of projects often requires the use of the Waterfall Process of software development?**
- Usually high-risk projects.
7. **What is the function of requirements management, mentioned in reference to Waterfall Process?**
- To control changes and correct mistakes during requirements establishing.
8. **With reference to iterative processes, what is agile development mainly based on?**
- It is mainly based on the feedback given by the customer, who is repeatedly provided with constantly evolving software.
9. **What condition must the coding process fulfil before it can be completed?**
- The implemented part has to pass all the tests designed before coding.

Further exercises

1. Students work in two teams: A/B. Group A are the customers, group B the developers. Imagine how the iterative process of development works on any application (e.g. building a house) and simulate it. Prepare a very short report which sums up the negotiations between the teams.
2. Prepare a comparison between iterative and waterfall processes. Write it down in table form. Focusing on the differences, answer the following questions:

- Which features of these techniques do you think are better?
- Do you use any of them? Which ones and why?

Possible topics for discussion

1. Do you agree with the critics of the iterative processes?
2. What are the steps of software development that are indispensable for creating a good academic project?
3. Imagine that you are the manager of a team of several people. Write the outline of a project that you have heard about during your studies. How would you divide the work within your team to make the final product perfect?

Possible difficulties

In some parts of the text the language is complicated, so intensive reading is required.

English++

English for Computer Science Students

Listening Chapter

What is Artificial Intelligence? Basic Questions

John McCarthy

Running time	17'41
Computer science content	Medium
Math content	Low
English language complexity	Medium

Learning objectives

- to understand what exactly is meant by the term artificial intelligence (AI)
- to be aware of the current situation in AI systems and the possibilities for future development
- to understand general AI theories

Keywords

- **confine** - to define boundaries; to limit the extent (of an activity)
- **computational** - adj.1. involving computers, 2. that can be computed
- **correlate** - to have a close similarity, connection or causal relationship with
- **heuristic hypothesis** - a hypothesis that has a very high probability of being true on the basis of reasoning and past experience
- **cognitive science** - the scientific study of mind or intelligence based on relevant fields, including psychology, philosophy, neuroscience, linguistics, anthropology, computer science, and biology
- **incoherent** - confused and inconsistent; illogical
- **computational complexity** - an area of computer design dealing with the problems of algorithms and their ability to solve a given problem

Pre-listening questions

1. How would you define "intelligence"?
2. Have you ever thought of computer programs as being "intelligent"?
3. Do you think that computers or machines will ever be as intelligent as humans?
4. Are you afraid of the nightmare scenario in which machines have control over people, possibly leading to the annihilation of the human race?

Summary

This text, in the form of an interview, is written by Prof. John McCarthy of Stanford University, who is an authority in the field of AI and was the first to use this term. It covers the main ideas of AI and its differences from human intelligence as well as describing AI in the context of biology, psychology and even philosophy. In the interview Professor McCarthy answers basic questions about artificial intelligence.

Ciekawy tekst opisujący podstawowe i bardzo ogólne zagadnienia z problemu sztucznej inteligencji. Jego niewątpliwym walorem jest fakt, iż pomimo ogólnej treści dotyka on dziedzin takich jak: psychologia, nauki kognitywne czy złożoność obliczeniowa. Można w nim znaleźć pytania, które mógłby zadać całkowity laik i zadowolić się ich odpowiedzią, jednocześnie interesując i informatyków. Dodatkowo, ponieważ tekst jest tak ogólny, może zachęcić studentów do poszukiwania bardziej zaawansowanych tekstów na temat AI. Tekst jest napisany w formie FAQ, i może służyć jako przybliżenie tej formy pisanego tekstu.

Transcript

What is Artificial Intelligence? Basic Questions

Q. What is artificial intelligence?

A. It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable.

Q. Yes, but what is intelligence?

A. Intelligence is the computational part of the ability to achieve goals in the world. Varying kinds and degrees of intelligence occur in people, many animals and some machines.

Q. Isn't there a solid definition of intelligence that doesn't depend on relating it to human intelligence?

A. Not yet. The problem is that we cannot yet characterize in general what kinds of computational procedures we want to call intelligent. We understand some of the mechanisms of intelligence and not others.

Q. Is intelligence a single thing so that one can ask a yes or no question Is this machine intelligent or not?

A. No. Intelligence involves mechanisms, and AI research has discovered how to make computers carry out some of them and not others. If doing a task requires only mechanisms that are well understood today, computer programs can give very impressive performances on these tasks. Such programs should be considered somewhat intelligent.

Q. Isn't AI about simulating human intelligence?

A. Sometimes but not always or even usually. On the one hand, we can learn something about how to make machines solve problems by observing other people or just by observing our own methods. On the other hand, most work in AI involves studying the problems the world presents to intelligence rather than studying people or animals. AI researchers are free to use methods that are not observed in people or that involve much more computing than people can do.

Q. What about IQ? Do computer programs have IQs?

A. No. IQ is based on the rates at which intelligence develops in children. It is the ratio of the age at which a child normally makes a certain score to the child's age.

correlate
to have a close similarity, connection or causal relationship with

The scale is extended to adults in a suitable way. IQ correlates well with various measures of success or failure in life, but making computers that can score high on IQ tests would be weakly correlated with their usefulness. For example, the ability of a child to repeat back a long sequence of digits correlates well with other intellectual abilities, perhaps because it measures how much information the child can compute with at once. However, digit span is trivial for even extremely limited computers. However, some of the problems on IQ tests are useful challenges for AI.

Q. What about other comparisons between human and computer intelligence?

Arthur R. Jensen [Jen98], a leading researcher in human intelligence, suggests as a heuristic hypothesis that all normal humans have the same intellectual mechanisms and that differences in intelligence are related to quantitative biochemical and physiological conditions. I see them as speed, short term memory, and the ability to form accurate and retrievable long term memories. Whether or not Jensen is right about human intelligence, the situation in AI today is the reverse. Computer programs have plenty of speed and memory but their abilities correspond to the intellectual mechanisms that program designers understand well enough to put in programs. Some abilities that children normally don't develop till they are teenagers may be in, and some abilities possessed by two year olds are still out. The matter is further complicated by the fact that the cognitive sciences still have not succeeded in determining exactly what the human abilities are. Very likely the organization of the intellectual mechanisms for AI can usefully be different from that in people. Whenever people do better than computers on some task or computers use a lot of computation to do as well as people, this demonstrates that the program designers lack understanding of the intellectual mechanisms required to do the task efficiently.

Q. When did AI research start?

heuristic hypothesis
a hypothesis that has a very high probability of being true on the basis of reasoning and past experience

A. After WW II, a number of people independently started to work on intelligent machines. The English mathematician Alan Turing may have been the first. He gave a lecture on it in 1947. He also may have been the first to decide that AI was best researched by programming computers rather than by building machines. By the late 1950s, there were many researchers on AI, and most of them were basing their work on programming computers.

Q. Does AI aim to put the human mind into the computer?

A. Some researchers say they have that objective, but maybe they are using the phrase metaphorically. The human mind has a lot of peculiarities, and I'm not sure anyone is serious about imitating all of them.

Q. What is the Turing test?

A. Alan Turing's 1950 article Computing Machinery and Intelligence [Tur50] discussed conditions for considering a machine to be intelligent. He argued that if the machine could successfully pretend to be human to a knowledgeable observer then you certainly should consider it intelligent. This test would satisfy most people but not all philosophers. The observer could interact with the machine and a human by teletype (to avoid requiring that the machine imitate the appearance or voice of the person), and the human would try to persuade the observer that it was human and the machine would try to fool the observer.

The Turing test is a one-sided test. A machine that passes the test should certainly be considered intelligent, but a machine could still be considered intelligent without knowing enough about humans to imitate a human. Daniel Dennett's book Brainchildren [Den98] has an excellent discussion of the Turing test and the various partial Turing tests that have been implemented, i.e. with restrictions on the observer's knowledge of AI and the subject matter of questioning. It turns out that some people are easily led into believing that a rather dumb program is intelligent.

computational complexity
an area of computer design dealing with the problems of algorithms and their ability to solve a given problem

Q. Does AI aim at human-level intelligence?

A. Yes. The ultimate effort is to make computer programs that can solve problems and achieve goals in the world as well as humans. However, many people involved in particular research areas are much less ambitious.

Q. How far is AI from reaching human-level intelligence? When will it happen?

A. A few people think that human-level intelligence can be achieved by writing large numbers of programs of the kind people are now writing and assembling vast knowledge bases of facts in the languages now used for expressing knowledge. However, most AI researchers believe that new fundamental ideas are required, and therefore it cannot be predicted when human-level intelligence will be achieved.

Q. Are computers the right kind of machine to be made intelligent?

A. Computers can be programmed to simulate any kind of machine. Many researchers invented non-computer machines, hoping that they would be intelligent in different ways than the computer programs could be. However, they usually simulate their invented machines on a computer and come to doubt that the new machine is worth building. Because many billions of dollars have been spent in making computers faster and faster, another kind of machine would have to be very fast to perform better than a program on a computer simulating the machine.

Q. Are computers fast enough to be intelligent?

A. Some people think much faster computers are required as well as new ideas. My own opinion is that the computers of 30 years ago were fast enough if only we knew how to program them. Of course, quite apart from the ambitions of AI researchers, computers will keep getting faster.

Q. What about parallel machines?

A. Machines with many processors are much faster than single processors can be. Parallelism itself presents no advantages, and parallel machines are somewhat awkward to program. When extreme speed is required, it is necessary to face this awkwardness.

Q. What about making a child machine that could improve by reading and by learning from experience?

cognitive science
the scientific study of mind or intelligence based on relevant fields, including psychology, philosophy, neuroscience, linguistics, anthropology, computer science, and biology

A. This idea has been proposed many times, starting in the 1940s. Eventually, it will be made to work. However, AI programs haven't yet reached the level of being able to learn much of what a child learns from physical experience. Nor do present programs understand language well enough to learn much by reading.

Q. Might an AI system be able to bootstrap itself to higher and higher level intelligence by thinking about AI?

A. I think yes, but we aren't yet at a level of AI at which this process can begin.

Q. What about chess?

A. Alexander Kronrod, a Russian AI researcher, said Chess is the Drosophila of AI. He was making an analogy with geneticists' use of that fruit fly to study inheritance. Playing chess requires certain intellectual mechanisms and not others. Chess programs now play at grandmaster level, but they do it with limited intellectual mechanisms compared to those used by a human chess player, substituting large amounts of computation for understanding. Once we understand these mechanisms better, we can build human-level chess programs that do far less computation than do present programs. Unfortunately, the competitive and commercial aspects of making computers play chess have taken precedence over using chess as a scientific domain. It is as if the geneticists after 1910 had organized fruit fly races and concentrated their efforts on breeding fruit flies that could win these races.

Q. What about Go?

A. The Chinese and Japanese game of Go is also a board game in which the players take turns moving. Go exposes the weakness of our present understanding of the intellectual mechanisms involved in human game playing. Go programs are very bad players, in spite of considerable effort (not as much as for chess). The problem seems

to be that a position in Go has to be divided mentally into a collection of sub-positions which are first analyzed separately followed by an analysis of their interaction. Humans use this in chess also, but chess programs consider the position as a whole. Chess programs compensate for the lack of this intellectual mechanism by doing thousands or, in the case of Deep Blue, many millions of times as much computation. Sooner or later, AI research will overcome this scandalous weakness.

Q. Don't some people say that AI is a bad idea?

A. The philosopher John Searle says that the idea of a non-biological machine being intelligent is incoherent. He proposes the Chinese room argument www-formal.stanford.edu/jmc/chinese.html The philosopher Hubert Dreyfus says that AI is impossible. The computer scientist Joseph Weizenbaum says the idea is obscene, anti-human and immoral. Various people have said that since artificial intelligence hasn't reached human level by now, it must be impossible. Still other people are disappointed that companies they invested in went bankrupt.

confine
to define boundaries;
to limit the extent
(of an activity)

Q. Aren't computability theory and computational complexity the keys to AI? [Note to the layman and beginners in computer science: These are quite technical branches of mathematical logic and computer science, and the answer to the question has to be somewhat technical.]

computational
1. involving computers
2. that can be computed

A. No. These theories are relevant but don't address the fundamental problems of AI. In the 1930s mathematical logicians, especially Kurt Gödel and Alan Turing, established that there did not exist algorithms that were guaranteed to solve all problems in certain important mathematical domains. Whether a sentence of first order logic is a theorem is one example, and whether a polynomial equation in several variables has integer solutions is another. Humans solve problems in these domains all the time, and this has been offered as an argument (usually with some decorations) that computers are intrinsically incapable of doing what people do. Roger Penrose claims this. However, people can't guarantee to solve arbitrary problems in these domains either. See my Review of The Emperor's New Mind by Roger Penrose. More essays and reviews defending AI research are in my book Concepts of Logical AI [McC96a].

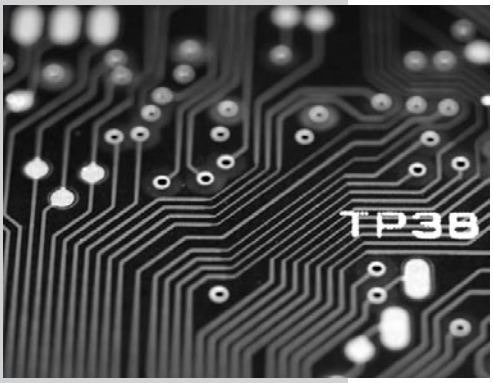
In the 1960s computer scientists, especially Steve Cook and Richard Karp developed the theory of NP-complete problem domains. Problems in these domains are solvable, but seem to take time exponential in the size of the problem. Which sentences of propositional calculus are satisfiable is a basic example of an NP-complete problem domain. Humans often solve problems in NP-complete domains in times much shorter than is guaranteed by the general algorithms, but can't solve them quickly in general.

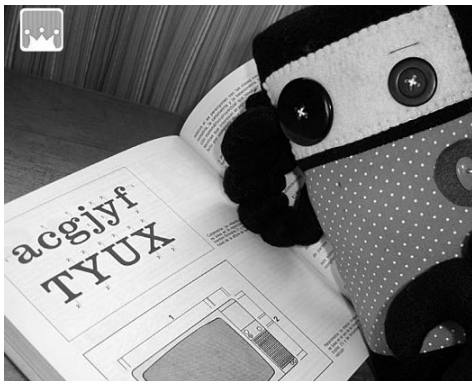
What is important for AI is to have algorithms as capable as people at solving problems. The identification of subdomains for which good algorithms exist is important, but a

incoherent
confused
and inconsistent; illogical

lot of AI problem solvers are not associated with readily identified subdomains.

The theory of the difficulty of general classes of problems is called computational complexity. So far this theory hasn't interacted with AI as much as might have been hoped. Success in problem solving by humans and by AI programs seems to rely on properties of problems and problem solving methods that the neither the complexity researchers nor the AI community have been able to identify precisely. Algorithmic complexity theory as developed by Solomonoff, Kolmogorov and Chaitin (independently of one another) is also relevant. It defines the complexity of a symbolic object as the length of the shortest program that will generate it. Proving that a candidate program is the shortest or close to the shortest is an unsolvable problem, but representing objects by short programs that generate them should sometimes be illuminating even when you can't prove that the program is the shortest.





Exercises

Pre-listening questions

1. How would you define "intelligence"?
2. Have you ever thought of computer programs as being "intelligent"?
3. Do you think that computers or machines will ever be as intelligent as humans?
4. Are you afraid of the nightmare scenario in which machines have control over people, possibly leading to the annihilation of the human race?

• Note taking exercises.

Listen to the recording and answer the following questions:

1. What does human intelligence depend on?

• "Arthur R. Jensen, a leading researcher in human intelligence, suggests as a heuristic hypothesis that all normal humans have the same intellectual mechanisms and that differences in intelligence are related to quantitative biochemical and physiological conditions. I see them as speed, short term memory, and the ability to form accurate and retrievable long term memories."

2. What is the main advantage of computers over people and vice versa?

• computation power (computer)

• abstract thinking (human)

3. What is the main aim of the Turing test?

• "Alan Turing's 1950 article Computing Machinery and Intelligence [Tur50] discussed conditions for considering a machine to be intelligent. He argued that if the machine could successfully pretend to be human to a knowledgeable observer then you certainly should consider it intelligent. This test would satisfy most people but not all philosophers. The observer could interact with the machine and a human by teletype (to avoid requiring that the machine imitate the appearance or voice of the person), and the human would try to persuade the observer that it was human and the machine would try to fool the observer. The Turing test is a one-sided test. A machine that passes the test should certainly be considered intelligent, but a machine could still be considered intelligent without knowing enough about humans to imitate a human."

4. In what ways have researchers tried to build an AI system similar to human intelligence?

• "A few people think that human-level intelligence can be achieved by writing large numbers of programs of the kind people are now writing and assembling vast knowledge bases of facts in the languages now used for expressing knowledge. However, most AI researchers believe that new fundamental ideas are required, and therefore it cannot be predicted when human-level intelligence will be achieved."

The same questions can be asked when the text is used for a reading exercise.

Possible topics for discussion

1. How has AI been depicted in books and films? (s-f)
2. Is AI an ethical or moral issue? (ethical)
3. What effect could AI have on religion?
4. Do you think machines with AI could ever be programmed to have religious consciousness or religious feelings?

Reading suggestions

Artificial Intelligence by Patrick Henry Winston, 2003, Addison Wesley Publishing Company.

Possible difficulties

Although in this interview the questions are basic, the answers are quite dense and formal.

Agile Software Development

from IT Conversation

Running time	5'36
Computer science content	Medium
Business content	Medium
English language complexity	Medium

Learning objectives

- to become familiar with certain characteristics of Agile methodology in comparison with others used in the software development process
- to understand how the introduction of Agile methodology can influence a company's work

Sub-areas covered

- Software development
- Agile Methodology

Keywords

- **software** - programs which run on a computer
- **Agile Software Development** - a conceptual framework for software development which promotes development iterations throughout the life-cycle of the project
- **software development process** - a structure imposed on the development of a software product. Synonyms include software life cycle and software process. There are several models for such processes, each describing approaches to a variety of tasks or activities which take place during the process.
- **pair programming** - a software development technique in which two programmers work together at one keyboard. One types in code while the other reviews each line of code as it is typed in. The person typing is called the driver. The person reviewing the code is called the observer or navigator. The two programmers switch roles frequently.
- **contract** - a legally binding exchange of promises or agreement between parties which is enforceable by law
- **software documentation (or source code documentation)** - written text which accompanies computer software. It explains either how it operates or how to use it, and may mean different things to people in different roles.

Pre-listening questions

1. How do you start your software projects? Do you have a plan or do you simply start coding?
2. Do you know any software development methodologies? Describe them.
3. How would you imagine collaborating with customers in software projects?

Summary

This recording is part of an interview with Alistair Cockburn conducted by Doug Kaye. Alistair Cockburn talks about the main characteristics of Agile methodology used in software development. He also points out the differences between this methodology and others utilized in software production. Being an expert in designing with Agile methodology, he suggests some practices he has developed himself.

Audio clip I

In this clip Alistair Cockburn emphasizes the importance of communication, cooperation and good interaction between people during the process of creating a project. He mentions tools supporting good communication and explains how to plan and design projects using Agile methodology.

Audio clip II

This concerns collaboration with customers and how to make contracts.

Audio clip III

Alistair Cockburn explains how to make plans and be flexible about adjusting them to changing situations during a project.

Audio clip IV

This part of the interview discusses the importance of documentation in capturing the intent of the developers. Alistair Cockburn mentions different forms of documentation and gives the example of making video recordings of developers' discussions.

Nagranie jest fragmentem wywiadu z Alistairem Cockburnem przeprowadzonym przez Douga Kayea. Alistair Cockburn mówi o głównych cechach metodologii Agile w zastosowaniach w rozwoju oprogramowania. Podkreśla tym samym różnice między tą metodologią a innymi, znajdującymi zastosowanie w wytwarzaniu oprogramowania. Jako osoba o dużym doświadczeniu w projektowaniu z wykorzystaniem tej metodologii, zaleca on również wypracowane przez siebie praktyki.

Fragment I

W tym fragmencie Alistair Cockburn podkreśla znaczenie komunikacji i kooperacji pomiędzy twórcami projektu i podaje przykłady narzędzi wspierających dobrą komunikację. Opowiada on także o sposobach planowania i projektowania w metodologii Agile.

Fragment II

Fragment traktuje o współpracy z klientami i o sposobach zawierania umów.

Fragment III

Alistair Cockburn mówi o sposobie planowania i elastycznego dostosowywania planów do zmieniającej się sytuacji w czasie trwania projektu.

Fragment IV

Ten fragment wywiadu traktuje o znaczeniu robienia dokumentacji w celu uchwycenia intencji twórców. Alistair Cockburn wymienia różne formy dokumentacji i podaje przykład nagrania video dyskutujących programistów.

software
programs which run on a computer

Agile Software Development
a conceptual framework for software development which promotes development iterations throughout the life-cycle of the project

Software development process
a structure imposed on the development of a software product. Synonyms include software life cycle and software process. There are several models for such processes, each describing approaches to a variety of tasks or activities which take place during the process.

Transcript

Audio clip I

We value individuals and interactions over processes and tools. The idea there is, uh, you know, it's funny, if you look at the arguments in the Agile world, they're all over process, the XP process, the Scrum process, the Crystal process, whatever. So we do talk about process, and we cherry-pick our tools very carefully. And I just wrote an article for Crosstalk Magazine on the Agile toolkit; cherry-pick them very carefully, want them very pointed and very efficient with clearly demonstrated value and so on. But, when we get down to it, the heart of what makes a good project work is not just the individual people that you have there, but the way in which they interact with each other. So we highlight individuals' interactions and so we talk a lot about collocation, collaboration. I talk a lot about morale, amicability, community. If you go to the toolset, now you want collaboration tools that allow multiple modes of communication, you want sound, you want video, you want archiving, you want online chat; you want all of these things at the same time. So you're watching it; there's a lot of workshop-based activities in Agile Development. Planning is not done by the project manager sitting alone in a room, but it's is done by the entire group of people who, whether it's coarse grain or fine grain, you're looking at the set of things that need to be done and jostling each other either to think of something new or to change their estimates or suggest solutions or something like that. That's done in design, also. There's pair programming, there's all sorts of things, but the emphasis is very, very strong on increasing, if you will, the brain-to-brain communication.

sound

Audio clip II

Customer collaboration over contract negotiation. So, yeah, there's a place for contracts, not to pretend like you can't ever do contracts, but even better than a contract is if everybody gets on the same side of the table and they work together. So this idea of collaboration not just within the team, but collaboration across organizational boundaries, in particular with the sponsors and end users, then you can actually optimize, for instance, the requirements. You could discover that the requirements as written aren't what the people really want. So now if you're out here in a contract situation, you have to talk about changing the contract. If you're in a customer-collaboration-system situation, you can work together and decide whether to make a change in the requirements or not. That's customer collaboration.

sound

Audio clip III

Responding to change over following a plan. So while we all agree that planning is a useful activity, the plan rarely survives, sort of, three days into the project. It's most of the time between three days and a week after the plan is built, it's out of date, something's happened. And so this responding-to-change idea is what I call paying attention to current reality. You come into the project one day – you know, whatever you're faced with, it could be anything - we don't get hung up on, say: well, but the plan said something; we go, hmmm! Okay, so what's the best thing to do now in the current situation? Do we, in fact, follow the plan or do we change the plan, or what do we do? So responding to change over following the plan actually has an implication that we do a lot more planning, but we do it collaborative, quick and either typically short-time horizon, fine-grained and sometimes long-time horizon coarse-grained, if that makes sense to you

sound

Audio clip IV

Alistair Cockburn: And 'adequate' now becomes the question. So what would be a barely sufficient amount of documentation? When would it be done? What would it look like to help the future people come on? That is a really tough question, it's a really ugly one and, typically, people don't do enough. So I'm looking for really inexpensive ways to do this, and I think white-board photos, videotapes of designers discussing at the white board, that kind of thing, may give us a handle on getting this.

Doug Kaye: That also has the advantage that it captures the intent of the developers, which sometimes can be more valuable than the actual code. After all, the code is there.

Alistair Cockburn: Classically, classically impossible to catch. I don't know how many research projects I've seen where they have as an intent to find a way to capture that. But if you have, for instance, somebody from the team who's got the software and somebody from a different team who doesn't know the software, and they take some aspect of the system and they go over the design for it, what'll happen is the person who knows the design will say, "Well, basically it works like this", and sketch some stuff out. And then the other person will interrupt and ask questions, "Well, why not this, and why not that, and what about this, and what about that?", just as the newcomer developer would want to ask, too. And then the conversation gets more convoluted and complex and deeper as they go. And if you have that all on video tape, then in a sense you've got this very rich conversation into the future that you can archive.

sound

English++

pair programming
a software development technique in which two programmers work together at one keyboard. One types in code while the other reviews each line of code as it is typed in. The person typing is called the driver. The person reviewing the code is called the observer or navigator. The two programmers switch roles frequently.

Contract
a legally binding exchange of promises or agreement between parties, which is enforceable by law

Software documentation (or source code documentation)
written text which accompanies computer software. It explains either how it operates or how to use it, and may mean different things to people in different roles.

Exercises

Pre-listening questions

1. How do you start your software projects? Do you have a plan or do you simply start coding?
2. Do you know any software development methodologies? Describe them.
3. How would you imagine collaborating with customers in software projects?

Comprehension questions

Audio clip 1

1. What is the basis of creating a good project?

- interaction, collaboration, morale, a sense of community

2. How is the idea of interaction realized during:

a. planning? (working as a group; brain storming, etc.)

b. design? (pair programming)

3. Which tools are used to help communication?

- sound, video, archiving, online chat, etc.

Audio clip 2

4. What does the idea of customer collaboration focus on?

- working out the contract in co-operation with customer

Audio clip 3

5. Is the plan for a project unchangeable? What happens if the situation during a project changes?

- no; there needs to be flexible adjustment to the situation and quick re-planning

Audio clip 4

6. What are the different forms of documentation during a project? Give examples. (white-board photos, videotapes of designers discussing at the white board, traditional written documentation)

7. What is the main purpose of doing documentation?

- capturing the intentions of a developer

Possibile topics for discussion

1. Is Agile software development always the best way to make software? (advantages/disadvantages)
2. One of the priorities in Agile is customer collaboration. How would you imagine this collaboration if you were developing software for billions of people around the globe?
3. How would you design the process of preparing documentation to make this less time-consuming?

Open News

Episode 25

from A Weekly Open Source News Podcast

Running time	4'05
Computer science content	High
Business content	Medium
English language complexity	Medium

Learning objectives

- to become familiar with the spectrum of Google's business activity
- to understand the importance of copyright law and what may be the consequences of breaking it

Sub-areas covered

- Operating systems
- Copyright law

Keywords

- **software** - programs run on a computer
- **Google** - an American public corporation whose domain is open-source software and whose earning revenue from advertising related to its Internet search, web-based e-mail, online mapping, office productivity, social networking and video sharing activities.
- **desktop (computer)** - a personal computer used at home or work (stationary)
- **bug** - an error in software, something that interferes with the flow of the program
- **downloading** - getting/transferring files (software, music, films, etc.) from a remote computer via the Internet
- **OS (Operating System)** - a piece of software that is responsible for the management of a computer and, above all, running programs (examples: MS Windows, Linux, Mac OS)
- **BitTorrent** - a protocol for sharing and distributing computer files via the Internet

Summary

Part 1 Goobuntu Confirmed

This audio clip is about the introduction of the operating system designed by Google to the market.

Part 2 Windows As Punishment

This audio clip tells the story of the man who was convicted of illegally downloading a film via the Internet and then forced to use the MS Windows operating system as part of his sentence.

Część 1 Goobuntu Confirmed

Treścią nagrania jest informacja o wprowadzanym na rynek przez firmę Google systemie operacyjnym.

Część 2 Windows As Punishment

Nagranie opowiada historię osoby, która będąc oskarżoną o nielegalne ściągnięcie filmu przez Internet, została zmuszona do używania SO Windows, co było rezultatem wyroku.

Pre-listening questions

1. What is the main area of Google's business activity?

2. Who are the main rivals of Microsoft's Windows operating system? What makes them ever more popular?

Transcript

Goobuntu Confirmed

There have been rumors floating around for years now that Google would release its own desktop operating system to compete with Microsoft Windows. Well now the existence of Google operating system has been confirmed. According to theregister.co.uk, Google is preparing its own distribution of Linux for the desktop in a possible bid to take on Microsoft in its core business - desktop software. A version of the increasingly popular Ubuntu desktop Linux distribution, based on Debian and the Gnome desktop, it is known internally as 'Goobuntu'. Google has confirmed it is working on a desktop Linux project called Goobuntu, but declined to supply further details, including what the project is for. It's possible that it's just one of the toys Googleplex engineers play with on Fridays, when they get time off from buffing the search engine code or filtering out entries about Tiananmen Square. It could be for wider deployments on the company's own desktops, as an alternative to Microsoft, but still for internal use only. But it's possible Google plans to distribute it to the general public, as a free alternative to Windows. Google has already demonstrated an interest in building a presence on the desktop. At CES Las Vegas last month, it announced the Google Pack, a collection of desktop software bundled together for easy downloading. Whatever Google's intentions, the input of Google engineers and developers, writing new features and fixing bugs, will be a huge boost to the Ubuntu project.

sound

Windows As Punishment

Many Linux users still use Microsoft Windows in one form or another. Some of us, including myself, use Windows strictly for gaming. Others are forced to use Windows at work. But what if you were legally required to use Windows and only Windows? Well that's what happened this week, according to vnunet.com. An article by Iain Thomson states:

A man convicted of illegally downloading an episode of Star Wars has been told that he can no longer use his computer with an Ubuntu Linux operating system. Scott McCausland pleaded guilty last year to 'conspiracy to commit copyright infringement' and 'criminal copyright infringement' by downloading Star Wars: Episode III illegally. He served five months in prison and is now on probation, but has been told that he cannot use his Linux computer.

"I had a meeting with my probation officer today and he told me that he had to install monitoring software on my PC. No big deal to me; that's part of my sentence," he wrote on his Lost and Alone blog. "However, the monitoring software doesn't support

software
programs run on a computer

Google
an American public corporation whose domain is open-source software and whose earning revenue from advertising related to its Internet search, web-based e-mail, online mapping, office productivity, social networking and video sharing activities.

desktop (computer)
a personal computer used at home or work (stationary computer)

bug
an error in software, something that interferes with the flow of the program

downloading
getting/transferring files
(software, music, films, etc.)
from a remote computer
via the Internet

OS (Operating System)
a piece of software
that is responsible
for the management
of a computer and,
above all,
running programs (examples:
MS Windows, Linux,
Mac OS)

BitTorrent
a protocol for sharing
and distributing
computer files
via the Internet

Linux. So he told me that if I want to use a computer, I would have to use an OS that the software can be installed on, which basically means Microsoft and monitoring software or no computer. I use Ubuntu 7.04 now, and they are trying to force me to switch."

McCausland, also known as 'sk0t', (or S-K-zero-T), was the first person to be jailed for BitTorrent use and was a member of the Elitetorrents site that was shut down by the FBI in 2005. He claimed that he's not trying to subvert restrictions placed on him, but does not see why he should have to buy a whole new operating system to do so.

"The terms of my plea state: 'I consent to periodic checks of my PC by my probation officer and/or the installation of software to monitor my internet activity,'" he wrote. "I am consenting to all of it, but it just so happens that the OS I use might not be supported by the software they use to monitor. So I do not feel (neither does my lawyer) that the government can force me to switch OS."

McCausland has now started an appeal on his website to cover the cost of buying Windows.

sound

Exercises

Goobontu confirmed

Pre-listening questions

1. What is the main area of Google's business activity?

- development of the Internet search engine, web services such as Gmail, Google News, iGoogle and stand-alone applications like Google Earth, Google Desktop, Hello, Picasa and so on

2. Who are the main rivals of Microsoft's Windows operating system? What makes them ever more popular?

- the main rivals are Unix-based systems and Mac OS - their growing popularity can be attributed to their security and ease of use

Comprehension questions

1. What system is the Google OS based on?

- It is based on Ubuntu Linux distribution.

2. What does the speaker imply when he mentions filtering out entries about Tiananmen Square?

- He implies that Google agreed to blue-pencil the information about the tragedy in Tiananmen Square when entering the Chinese market).

3. Does Goobuntu present an immediate threat to the current popularity of the Microsoft Windows operating system?

- No, because this system [Goobuntu] is designed only for internal use within Google and so far there is only a possibility that it will be distributed it to the general public).

4. Which product will enable Google to play a more important role in the desktop applications market?

- Google Pack, the collection of free, useful desktop applications

Possible topics for discussion

1. What is the role of competition in the domestic and global market?
2. Would Google's operating system have a chance of winning a competition with Windows? Discuss how Google could make this happen.

Windows as punishment

Pre-listening questions

1. What are the means by which a government can protect copyright law on the Internet?

- It can, for example, take away a person's computer and search it for software or files that the owner does not have the right to use. It can also install monitoring software on someone's PC or monitor Internet traffic in a specific network to watch out for people downloading illegally.

Comprehension questions

1.What was Scott McCausland accused of?

- he was accused of violating copyright law by downloading Star Wars Episode III illegally.

2. How long did McCausland spend in prison?

- five months.

3. Why was McCausland forced to use the Microsoft Windows operating system?

- he had to have monitoring software installed on his computer which was compatible only with Windows.

4. What had Scott McCausland been accused of?

- illegal BitTorrent use.

Possible topics for discussion

1. What is your opinion about the sentence that force people to buy something?

Open News

Episode 29

from A Weekly Open Source News Podcast

Running time	6'23
Computer science content	High
Business content	Medium
English language complexity	High

Learning objectives

- to become familiar with recent developments in open source software
- to understand the importance of patents for software
- to know how the OSI institute works in particular cases

Sub-areas covered

- News from the IT world
- Open source technologies.

Keywords

- **Suse, Red Hat, Ubuntu** - examples of linux distributions
- **SCO** - a software company formerly called Caldera Systems and Caldera International. After acquiring the Santa Cruz Operation's Server Software and Services divisions, as well as UnixWare and OpenServer technologies, the company changed its focus to UNIX. Later on, Caldera changed its name to The SCO Group to reflect that change in focus.
- **user interface** - the aggregate of means by which people, the users, interact with the system, a particular machine, device, computer program or other complex tools. The user interface provides means of input, allowing the users to manipulate a system and output, allowing the system to produce the effects of the users' manipulation.
- **software license (**UK spelling**: licence)** - there are several important IT licenses like GNU GPL(open source), Mozilla Public License or Microsoft Public License
- **Firefox** - one of the most popular web browsers (after Internet Explorer)
- **MPL** - Microsoft Public Licence
- **MRL** - Microsoft Reciprocal Licence

Summary

This is a podcast with interesting news from the open source community. In this recording we hear about:

- some recent developments in brief
- Patents (the legal battle between MS and RedHat)
- Mozilla's plans to develop a mobile browser
- OSI's approval of MS licenses

Jest to podcast z ciekawymi informacjami ze świata open source. W tym materiale możemy usłyszeć kilka krótkich wiadomości:

- patenty i wojna pomiędzy MS a RedHatem
- Mozilla chce wyprodukować wersję przeglądarki na telefony komórkowe
- OSI akceptuje licencje Microsoftu

Transcript

First, here are some news briefs.

If you've been wanting to learn more about Ubuntu, now's your chance. Ubuntu this week announced Ubuntu Open Week, "a week-long series of online workshops", where you can chat with developers, attend training sessions, and even talk to Mark Shuttleworth himself. You can find out more at wiki.ubuntu.com/UbuntuOpenWeek.

It appears that Firefox is gaining ground on Internet Explorer, at least if you're a BitTorrent user. Isohunt, a popular BitTorrent indexing site, reported on their blog this week that Firefox users on isohunt.com edged out IE users for the first time since they have been monitoring.

Apparently officials in Amsterdam have been testing Open Source applications to see if they are ready for prime time. This week the testing committee gave the green light and now over 10,000 PCs can start using Open Source software. Any incremental costs of running the open source software will be offset by savings in license fees that would have been paid to Microsoft.

And now on to the main stories of the week.

It was a big week for patent litigation and it all started with more threatening words from Steve Ballmer of Microsoft. According to Tom Sanders on vnunet.com, Microsoft chief executive Steve Ballmer has warned users of Red Hat Linux that they will have to pay Microsoft for its intellectual property. "People who use Red Hat, at least with respect to our intellectual property, in a sense have an obligation to compensate us," Ballmer said last week at a company event in London discussing online services in the UK. Red Hat has repeatedly stated that it will not engage in a patent licensing deal similar to the Novell-Microsoft partnership, referring to it as an 'innovation tax'. Microsoft has been the second most aggressive party in pursuing alleged intellectual property claims against Linux and open source in general. The firm ranks behind SCO, which failed in its attempt to prove that it owns the intellectual property to Linux and now faces bankruptcy. Microsoft inked a partnership with Novell last year in which Novell agreed to license Microsoft's intellectual property in exchange for a patent pledge to users of Novell's SuSE Linux. Ballmer praised Novell at the UK event for valuing intellectual property, and suggested that open source vendors will be forced to strike similar deals with other patent holders.

sound

Mozilla, the company behind popular open source applications like Firefox and Thunderbird, announced this week that it will develop a mobile browser for cell phones

English++

software license
(UK spelling**: licence)**
there are several
important IT licenses
like GNU GPL
(open source),
Mozilla Public License
or Microsoft Public License

Firefox
one of the most popular
web browsers
(after Internet Explorer)

MPL
Microsoft Public Licence

Suse, Red Hat, Ubuntu
examples of linux
distributions

MRL
Microsoft
Reciprocal Licence

SCO
a software company formerly called Caldera Systems and Caldera International. After acquiring the Santa Cruz Operation's Server Software and Services divisions, as well as UnixWare and OpenServer technologies, the company changed its focus to UNIX. Later on, Caldera changed its name to The SCO Group to reflect that change in focus

user interface
the aggregate of means by which people, the users, interact with the system, a particular machine, device, computer program or other complex tools. The user interface provides means of input, allowing the users to manipulate a system and output, allowing the system to produce the effects of the users' manipulation.

and other devices. According to an article by Cade Metz on regdeveloper.co.uk, Mozilla is prepping a mobile version of Firefox, the world's most popular open-source web browser. "People ask us all the time what Mozilla's going to do about the mobile web," reads a blog post from Vice-President of engineering Mike Schroepfer, "and I'm very excited to announce that we plan to rock it." He says the company will introduce a mobile version of Firefox at some unspecified date after December 31, 2007. It will run Firefox extensions, and developers will have the power to build their own apps for the browser via Mozilla's user interface language, called XUL. As Schroepfer points out, a Mozilla-based browser is already available for Nokia's N800 wireless handheld. But the company has yet to decide which devices Mobile Firefox will run on.

sound

It was announced yesterday that the OSI has approved two Microsoft licenses as open source licenses. According to infoworld.com, the board of the Open Source Initiative, or OSI, has approved two Microsoft licenses that will allow proprietary source code to be shared, a move that is likely to inspire protest and spur controversy for die-hard open source proponents. The Microsoft Public License, or MPL, and the Microsoft Reciprocal License, or MRL, two of Microsoft's "shared source" licenses, are now viable OSI licenses for distributing open source code alongside more widely used community licenses such as the GNU General Public License and the Mozilla Public License. "Today's approval by the OSI concludes a tremendous learning experience for Microsoft, and I look forward to our continuing participation in the open source community," said Microsoft general manager of Windows Server Marketing and Platform Strategy Bill Hilf in a press statement. Red Hat executive Michael Tiemann, who also serves as president of the OSI, said Tuesday that while some in the community balked at the OSI accepting licenses from a company that historically has not been open source friendly, in the end, the licenses spoke for themselves. "They do have two licenses that went through the community process and did sustain the open source definition," he said. "I've received three e-mails in the last hour from people who say, 'To heck with the OSI, you guys are just now pawns in Microsoft's game. You have made a deal with the devil,'" he said. However, Tiemann believes the OSI had a responsibility to be fair and impartial in letting Microsoft submit its licenses. "What would you think of a club that would say, 'We won't take any members that come from the city of Chicago, even if two people in Chicago meet the criteria for entry?'" he posited. "We said from the outset that we would be fair and, to be honest, some people said, 'No, you don't'". Tiemann said it remains to be seen if any companies other than Microsoft will license code under MPL or MRL. "However," he said, "if Microsoft plans to embed patented technology in software licensed under an OSI-approved license and call it open source - as some open source proponents fear - they had better think twice."

Exercises

• Answer the following questions

1. Who will be using open source software in Amsterdam?

• officials

2. What is Steve Ballmer's position in Microsoft?

• Microsoft chief executive

3. When does Mozilla plan to introduce a new version of their web browser?

• some time after the 31st December 2007

4. Name some open source licenses.

• GNU GPL

• Mozilla Public License

• Microsoft Public License

5. What is the OSI?

• The Open Source Initiative

• Listen to the recording and fill the gaps

News in brief

1. Ubuntu this week announced Ubuntu Open Week, "a week-long series of **[online workshops]**"
2. It appears that Firefox is gaining ground on Internet Explorer, at least if you're a **[BitTorrent]** user.
3. Isohunt, a popular BitTorrent indexing site, reported on their blog this week that Firefox users on **[isohunt.com]** edged out IE users.
4. Any incremental costs of running the open source software will be offset by savings in **[license fees]** that would have been paid to Microsoft.

Story One

1. People who use Red Hat, at least with respect to our **[intellectual property]**,
2. Red Hat has repeatedly stated that it will not engage in a **[patent licensing deal]** similar to the Novell-Microsoft partnership, referring to it as an **[innovation tax]**.
3. The firm ranks behind SCO, which failed in its attempt to prove that it owns the intellectual property to Linux and now faces **[bankruptcy**].

Story Two

1. Mozilla is prepping a mobile version of **[Firefox]**, the world's most popular **[open-source]** web browser.
2. People ask us all the time about what Mozilla's going to do about the **[mobile web]**.
3. It will run Firefox **[extensions]**, and developers will have the power to build their own apps for the browser via Mozilla's **[user interface language]**, called XUL.
4. As Schroepfer points out, a Mozilla-based browser is already available for Nokia's N800 **[wireless handheld]**.

Story Three

1. The board of the **[Open Source Initiative]**, or OSI, has approved two Microsoft licenses…
2. The Microsoft Public License, or MPL, and the **[Microsoft Reciprocal License]**, r MRL, two of Microsoft's "shared source" licenses…
3. …said Microsoft general manager of Windows Server Marketing and **[Platform Strategy]** Bill Hilf in a press statement.
4. They do have two licenses that went through the **[community process]** and did sustain the open source definition
5. However, he said if Microsoft plans to embed **[patented technology]** in software licensed under an OSI-approved license and call it open source …

Possible topics for discussion

- What do you know about the GPL or MPL licenses?
- Do you like using free software? Yes, why. No, why.

Open News

Episode 31

from A Weekly Open Source News Podcast

Running time	6'04
Computer science content	Medium
Business content	Medium
English language complexity	Medium/High

Learning objectives

- to acquire knowledge on open source software usage
- to become familiar with licencing and what effect it may have on open source

Sub-areas covered

- Open source issues
- FOSS software

Keywords

- **Podcast** - a series of digital-media files which are distributed over the Internet using syndication feeds for playback on portable media players and computers
- **Gnome** - one of the biggest window managers for Linux
- **FOSS** - the acronym for Free and Open Source Software, software that can be used, studied, and modified without restriction
- **OHA** - Open Handset Alliance™, a group of more than 30 technology and mobile companies who have developed Android, the first complete, open, and free mobile platform
- **LiSP** - The Linux Phone Standards Forum, a consortium founded by a group of telephony operators, device manufacturers, silicon and software vendors who have a strategic focus on Linux telephony

Summary

This is an audio podcast with news from the FOSS world.

Clip 1. This clip is about film-makers using open-source software and the main reasons for them doing so.

Clip 2. This is about Google's purchase of the mobile phone platform company Android. It contains different opinions about this development and discusses the possible consequences and significance for the mobile Linux community.

Clip 3. This is a report on running the Linux system on school computers in Macedonia.

Jest to audio podcast z wiadomościami ze świata wolnego oprogramowania

Fragment 1. W tym fragmencie zawarto informację o używaniu oprogramowania open-source przez twórców filmów, a także przedstawiono główne przyczyny takiej tendencji.

Fragment 2. Ten fragment donosi o zakupieniu platformy telefonii komórkowej przez firmę Google. Przedstawiono w nim różne opinie na ten temat, a także znaczenie i możliwe konsekwencje tego faktu dla społeczności twórców wolnego oprogramowania.

Fragment 3. Ten fragment zawiera informację o wprowadzeniu systemu Linux w szkolnych komputerach w całej Macedonii.

Transcript

More and more film-makers are turning to open-source software to meet the demanding needs of their production environments. Lisa Hoover, in an article on linuxinsider.com, writes: 'Since the advent of the music (The speaker says 'music' but this is a mistake for 'movie'.) industry, the process has remained largely unchanged. A writer creates a script, finds someone willing to invest money in producing it, and a director is brought in to oversee the film's production.' While that process is still typical of most movie-making ventures, more and more independent filmmakers are turning toward open source tools and methods to produce their projects. For many filmmakers, the economics alone make open source film making an attractive option. When free or low-cost production tools like CinePaint and Blender are combined with free hosting services like Internet Archive, writers and filmmakers are free to focus on film production rather than financing. Some filmmakers are even going beyond simply sharing ideas, and are also sharing the footage they've shot. "It makes sense when resources can be shared and costs can be cut," writer and independent film director Valentin Spirik told LinuxInsider. "I also see a lot of potential for remix-related projects where the source footage is used in many different productions. Right now this is a niche genre online, but it might just be the future of interactive storytelling."

sound

You may have heard recently that Google purchased the mobile phone platform company Android, but what does that mean for comapnies like Open Moco and Tratec, who are trying to develop their own platforms based on open-source software. According to a story by Phil Manchester at regdeveloper.co.uk: While the mobile Linux community has reacted positively to Google's Android, the new platform has also given some cause for concern. The arrival of a giant player with very clear ideas of roles it wants mobile Linux to fill was bound to ruffle a few feathers and, despite public proclamations of welcome and support , the Linux establishment is showing a few cracks. Like it or not, Google has achieved something that none of the established knitting circles has managed so far: it has created a single target platform for developers to aim for. But a unified standard does not necessarily play well with the established mobile Linux players. The LiPS or L-i-P-S Forum, for example, says that it "regards OHA as complementary" and acknowledges that Android and the OHA have confirmed the popularity of Linux in mobile and embedded applications. LiPS also says that Android shares its mission "to reduce fragmentation among Linux-based mobile platforms" - only with a different approach. While LiPS aims to unify the development of mobile Linux through open standards, it sees the Android and OHA team as working to the same end with shared code. But elsewhere LiPS general manager Bill Weinberg has expressed concerns about the limitations of Google's use of the Apache license for Android and suggests that far from reducing fragmentation, Android might increase

Podcast
a series of digital-media files which are distributed over the Internet using syndication feeds for playback on portable media players and computers

Gnome
one of the biggest window managers for Linux

FOSS
the acronym for Free and Open Source Software, software that can be used, studied, and modified without restriction. For further info see wikipedia

OHA
Open Handset Alliance™, a group of more than 30 technology and mobile companies who have developed Android, the first complete, open, and free mobile platform

it. Morgan Gillis, executive director of the LiMo Foundation, said publicly that he welcomed Google's move and also sees the role of the OHA as complementary. "Google's angle is," he said, "focused on the end-user experience and bringing the Web to mobile devices while the LiMo Foundation wants to create a common middleware platform to underpin mobile applications." But in the same interview he pointed out that those in the mobile Linux area have a stark choice: "Work with Google, or think very seriously about how to achieve the next-generation mobile internet experience for their customers on their own." The next six months will certainly be critical in the growth of Android ecosystem. It could have the same galvanising effect on mobile application development that Windows had on PC applications in the late 1980s. Developers need to pick it up and create some compelling applications to match the appeal of, say, Apple's iPhone. But despite the concerns over licensing and the possibilities of a broader long-term agenda from Google, it would be hard (and, perhaps, foolish) to bet against it.

sound

Apparently, the Macedonian government is going for Linux in a big way. According to an article on the ubuntu.com website, the Macedonia Ministry of Education and Science will deploy more than 180,000 workstations running Canonical's ubuntu 7.04 as part of its "Computer for Every Child" project.

LiSP
The Linux Phone Standards Forum, a consortium founded by a group of telephony operators, device manufacturers, silicon and software vendors who have a strategic focus on Linux telephony

The Macedonia "Computer for Every Child" project is one of the largest known thin client and desktop Linux deployments ever undertaken. Half of elementary and secondary Macedonia students attend school in the morning, and half attend in the afternoon, so 180,000 workstations will allow for one classroom computing device per student for the entire Republic's public school population. The first 7000 computers pre-installed with Ubuntu were shipped on September 4th 2007. The Ubuntu operating system will run on 160,000 virtual PC terminals and 20,000 PCs (which also support a student on the attached monitor) supplied by NComputing and procured and installed by The Haier Company, a diversified manufacturer and PC maker based in China. The project will enable a range of innovative educational programs, including interactive web-based classes in which specialized experts teach lessons in such areas as mathematics, biology, chemistry and physics to multiple schools and classrooms around the country.

Exercises

• Answer the following questions

1. What are film-makers increasingly using to smooth the way for their projects?

• open source software

2. Which open source tools and services are mentioned as being used by filmmakers?. What functions do they perform? Do you know of any other examples?

Examples from listening:

• Cine Paint - open source painting and image retouching program designed to work best with 35mm film and other high resolution high dynamic range images.

• Blender - open source, cross platform suite of tools for 3D creation.

• Internet Archive - nonprofit organisation established to preserve websites by taking regular "snapshots". The Wayback Machine provides links to older versions of a webpage.

• Listen to the recording and fill the gaps

1. While the mobile Linux community has reacted positively to **[Google's Android]**.
2. Like it or not, Google has achieved something that none of the established **[knitting circles]** has managed so far. LiPS also says that Android shares its mission to reduce **[fragmentation among Linux-based]** mobile platforms
3. While LiPS aims to **[unify]** the development of mobile Linux through open standards
4. Morgan Gillis, executive director of the **[LiMo Foundation]** …
5. mobile devices while the LiMo Foundation wants to create a common **[middleware platform]** to underpin mobile applications.
6. The next six months will certainly be critical in the growth of **[Android ecosystem]**.
7. Developers need to pick it up and create some **[compelling applications]** to match the appeal of, say, Apple's iPhone.

• Listen to the recording and fill the gaps

1. The Macedonia Ministry of Education and Science will deploy more than 180,000 workstations running Canonical's Ubuntu 7.04 as part of its **[Computer for Every Child]** project.
2. The Macedonia "Computer for Every Child" project is one of the largest known thin client and **[desktop Linux]** deployments ever undertaken
3. The first 7000 computers **[pre-installed]** with Ubuntu were shipped on September 4th 2007
4. The Ubuntu operating system will run on **[160,000 virtual PC terminals]** and 20,000 PCs

5. The project will enable a range of innovative educational programs, including interactive **[web-based classes]**

Possible topics for discussion

1. Should film-makers use open source rather than commercial software?
2. Is open-source software the best choice for school computers? Give arguments for and against this proposal.

English++

English for Computer Science Students

Presentation Chapter

Presentation Chapter

This chapter deals briefly with a difficult task of speaking in public.

We hope that the tips which we worked out during our English classes will be helpful to those of you who are planning to make presentations in English. To make that task easier a repertoire of 'presentation phrases' is included. You can make a reasonable selection of useful phrases to start your presentation, develop its main points and finish it successfully.

This part of the book is accompanied by a DVD with a slide show produced by English++ team. "Successful presentations. A few tips from English++" presents examples of well-prepared and well-done presentations. Obviously, you can always improve one thing or another, but the point is not to criticize, but learn. As English++ project leader I am grateful to those who let us use their own material for the purpose of the this book.

The slide show also includes some clips which show a little bit less effective productions. Why? See for yourself. Naturally, you should take them with a pinch of salt, but perhaps there is something in it.

How to Give a Successful Presentation?

Practical information

Pre-preparation. Selecting the topic and materials

An essential task in a pre-preparatory phase is to ask yourself the following questions:

- What is the purpose of my presentation?
- What are the main points that I would like to get across?

Start getting ready for your presentation a few weeks before you are due to speak.

Collect the materials on the basis of which you would like to prepare the presentation.

Make a wise choice out of the collected materials.

Prepare reliable bibliography (the author's name, the title of the article/book, a year of its publication, the website address, when the material was retrieved, etc.).

From the material select the keywords for your presentation and do not forget to put them on the handout.

Preparation

- Make the first plan of the presentation (you can modify it later)
- Remember about a logical structure of the presentation:
  Introduction - say, what you're going to say
  Main Body - say it; develop the above mentioned issue(s).
  Conclusions - sum up what you've just said
- Make the first draft of your presentation. Read it carefully. If there is information not related to the topic, remove it.
- If there are issues which you cannot express in a precise or clear way, probably it is because you do not really understand them yourself. So it is better not to talk about them.
- Never read from your notes. You should know well the material you want to present. If you do not know it, maybe you should not go for giving a presentation.
- Prepare a set of numbered clue cards, on which you can write the main points or/ and keywords and which will help you during the presentation. Make sure that on your clue cards you've marked an appropriate visual aid (a transparency or slide), which you are going to refer to in your presentation.

- Find time to rehearse again and again!
  By doing that you are going to make yourself familiar with your own voice, to check and adjust the presenting time, to see whether visual aids (if you use them) actually illustrate your presentation and whether you coordinate well their usage with what you say. If you have problems with a foreign language in which you present the material, it could be better not to use visuals at all. Otherwise the chance that something goes wrong is bigger than when you concentrate solely on the oral aspect of the presentation.
- Keep to the time! Do not exceed the time limit.
  It is better to shorten the presentation by two minutes rather than extend it by two minutes. Remember that exceeding the time limit may mean taking the next speaker's time. And this is really unfair!
- Follow the plan of your presentation!
  Do not digress! Usually digressions take more time than we think. Successful presenters have "spontaneous digressions" well thought over and well planned.
- Leave time for questions from the audience.
  Questions may help you to get your message across better.
- Design good visuals to help you get your message across more efficiently.
  Remember that tatty visuals will leave a similar impression on the listeners.
  Visual aids should speak for themselves illustrating your point. Give listeners time to take them in. Reading out what you have written on a transparency or slide is counterproductive. Visuals are always welcomed – they may help to catch the audience's attention, but if you do not feel comfortable with them, give yourself more time for practice.

Visual aids

There are different types of visuals. The choice depends on the type of presentation and your needs.

- OHP transparencies - overhead projector transparencies
- Presentations in PowerPoint
- Flipcharts
- Whiteboard and marker
- Video
- Short episodes or scenes from films
- 35 mm slides
- Real objects, which you can provide listeners with to explore them

Remember that badly prepared and/or badly used visuals can ruin your presentation!

Make sure you know in advance how to connect the equipment and what to do to have a desired slide or diagram on the screen. Sometimes, during real presentations

a technician is designated to operate the equipment. In that case make sure that he understands your signals, which you are going to send when you wish, for example, to change a slide.

Slides and transparencies should contain minimum information which is needed to illustrate your point. Too much on a slide makes it unreadable and diverts audience's attention from what you say.

A slide is said to be readable if it contains no more than ten words in 18pt Times Roman font or bigger and which can be read without a projector from the distance of two meters. Never use as your visuals pages photocopied from books or other materials. They are not adapted for such usage, not to mention the fact that they don not prove your professionalism.

For transparencies, use multiple colours, but be careful with orange and yellow - they do not come out well on the screen. On slides, the text is often in yellow or white colour on the blue background. Avoid adding/drawing information on a presented transparency. While speaking avoid indicating with your finger or a marker. A pointer and the screen do that job much better.

Turn off a projector if you do not need it for the next few minutes. Two minutes is the maximum time that the same slide should be shown on the screen.

Presentation - dress rehearsal

Presentation should not take place without practicing it earlier a lot of times in front of a mirror or/and with a tape recorder. Observing yourself and listening to yourself "in action" is stressful, but very formative. It provides you with the material on which you can gradually develop your presentational skills and competencies.

A couple of final tips:

- Speak clearly.
- Avoid raised voice, whispering or mumbling "under your nose".
- Try to maintain your natural pace of speaking typical for an official, not everyday situation.
- Make pauses in places which you consider critical for your presentation; this treatment emphasizes the crucial information you wish to transmit to the audience.
- Try to control your body language; avoid excessive gesticulation.
- Keep an eye contact with your listeners but do not focus on one person only.
- Don't turn back to the audience if you want to show something on the screen and don't 'talk to the screen' either.
- Don't stand in the light of a projector covering the screen.
- Observe your audience's reactions – maybe you should shorten the presentation

by two minutes and move on to conclusions.

- Don't forget to thank the audience for their attention and encourage them to ask questions. If you are not sure about the answer of if you simply do not know it, don't be afraid to admit that, but suggest the source in which the answer can be found.
- Enjoy your presentation. Try to treat it as your new experience and show your enthusiasm!

Good luck!

Repertoire of presentation phrases

Part one – introduction

1. Signaling the start

OK, then, shall we start?

OK, then, I'd like to begin.

Let's start …

2. Greeting

Good morning/afternoon, ladies and gentlemen/everyone.

Thank you for coming.

I'm very happy that you've come here today.

3. Self-introduction

First of all, I'd like to introduce myself.

My name is …

Let me start with just a few words about myself/my own background.

I'm ……(name) from ……(country/city).

I'm from ……..(organisation).

I work as a ……..(job) for …….(organisation).

I study ……(subject) at …….(university). I'm in my third year.

I represent ……../ I'm a representative of …

4.Introducing the subject

Today, I'm going to talk about…

I'd like to talk to you today about …

I'm going to present the recent …/inform you about …/describe …

The subject/focus/topic of my talk/presentation/speech is …

5. Stating the purpose

We are here today to decide/learn about/discuss …

The aim/objective/purpose today is to update you on …/give you the background to …

In my presentation today I'll be discussing …/I'm going to explain …

What I'd like to do today is to present …

6. Outline (main points/sequencing/length)

I've divided my presentation into four parts/sections. They are …

My presentation will consist of …

Right, I'd like to begin with my first point.

Firstly/First of all ……..I will …

Secondly/then/next …….I would like to …

Thirdly / and then we come to …

After that/later……..I'm going to talk about / look at …

Finally/lastly/last of all I'd like to analyze/discuss/look at/consider/explain/tell you about/ show you how/speak to you about …

7. Inviting questions

I'd be glad to answer any questions at the end of my talk.

If you have any questions, please feel free to interrupt.

Part two – main body

1. Ordering

A reaching the end of one point

Right, I've talked about/ mentioned…

We've looked at …

That's all I have to say about …

That covers …

B moving on to the next point

Now we come to … That brings us to …

I'd like to move on/draw your attention to my next point.

Let's move on to …

Let's look now at …The next thing I'd like to talk about is …

2. Giving reasons

The main explanation for this is …

There are two reasons/explanations for this. First,… Second,…

This can be explained by two factors. Firstly, … Secondly, …

This is due to …

One reason for this is …

Another reason is …

3. Developing the point

Where does that take us?

Let's look at this in more detail.

What does that mean for us?

4. Giving examples

A good example of this is …

To illustrate this point, …

To support what I've said …

I'd like to give you some examples …

for instance … for example … such as … like…

5. Referring to visuals

Take/Have a…

Let's…

I'd like you to look at this transparency/graph/table/pie chart/flow chart/bar chart/diagram/chart

I'd like to draw your attention to …

This shows/represents …

As you can see …

If you look at …you can see from the transparency, etc. that …

Let me show you …

Could you look at your handouts …

The next slide/picture/graph shows …

6. Relating ideas

A showing consequence

therefore … so … consequently … because of this … as a result ….

B comparing

similarly … in the same way …

one similarity/difference between …… is that …

C contrasting

but … however … although … nevertheless…

even though … despite / in spite (of the fact that) … whereas/while…

in contrast to … by contrast with … on the other hand…

D highlighting

in particular … especially …

– changing the word order, e.g.: 'What was important was the final conclusion.' (instead of: 'The final conclusion was important.')

– repeating key words, e.g.: 'We need to compare past achievements and present achievements.' (instead of 'We need to compare past and present achievements.')

E showing an additional argument

moreover … in addition to this … not only … but also furthermore…

Part three – ending

1. Signaling the end

That brings me to the end of my presentation.

That completes my presentation.

That covers all I want/wanted to say today.

Before I stop/finish, let me just say …

2. Summarizing

Let me just run over the key points again.

I'll briefly summarize the main issues.

To sum up…

Let's recap, shall we?

If I can just sum up the main points …

Finally, I'd like to go over/review …

Firstly, I talked about ……. Secondly, I discussed……Thirdly, I looked at ……

3. Concluding

In conclusion …

I'd like to conclude by saying …

As a conclusion, I'd like to …

I'd like to leave you with the following thought/idea.

4. Closing

Thank you for your attention / being so attentive/listening.

It was pleasure talking to you …

I will be giving you handouts.

You will find handouts at the entrance.

There are copies on the table.

5. Inviting questions

Are there any questions?

Have you got any questions?

6. Asking questions

Could I go back to the point you made about…?

I was interested in your comment on …

You said that …

Could you say a little more about that?

Could you clarify what you said about …?

I'd like to ask about …

May I ask you a question?

I'm interested in your opinion about …

Do you mind if I ask you…

I'm interested to know …

7. Handling questions

A clarifying

If I understand you correctly, you are saying/asking …

I didn't quite catch that.

Could you go over that again?

I'm not sure what you're getting at.

Sorry, I'm not sure what you mean.

B playing for time

That's a good/interesting/difficult point/question/comment.

I'm glad you raised that point.

C saying you don't know

I'm sorry I don't have that information at this moment.

I'm afraid I don't know at the moment.

D avoiding giving answers

Perhaps we could deal with that later.

Can we talk about that on another occasion?

I'm afraid that's not my field.

I don't have the figures with me.

I'm sure Mr/Ms … could answer this question.

I'll get back to you if time permits.

E checking if the questioner is satisfied

Does that answer your question?

Is that clear?

Can we go on?

F concluding the questions

Right, if nobody wants to ask anything else, I think we can finish here.

Right, if there are no more questions …

Thank you.

English++

English for Computer Science Students

Appendixes

Appendix A
Mathematical terminology

Listen to the recording

algebra	algebra
algorithm	algorytm
analyze analytical	analizować analityczny obliczeniowy
approximation	przybliżenie aproksymacja
arc	łuk
area	pole obszar
argument	argument w sensie: dowód, argument (funkcji)
assert	dowieść
assumption	założenie
average mean	średnia
axis axes (pl)	oś
ball unit ball	kula kula jednostkowa
binomial	dwumian (Newtona)
brace	{}
square bracket	[ ]
cancel	anulować, skreślić
common fraction	ułamek zwykły
complex number	liczba zespolona
cube	sześcian [podnieść do sześcianu]
cube root	pierwiastek sześcienny
curve	krzywa
data	dane

decimal fraction	ułamek dziesiętny
deduce	wywnioskować wydedukować
derive (v) **derivation** (n)	wyprowadzać (np. wzór) obliczanie pochodnej
derivative (n, adj)	pochodna pochodny
diagonal (n, adj)	przekątna diagonalny przekątny
dimension	wymiar
divide	dzielić
differential (n, adj)	różniczka różniczkowy
domain (of a function)	dziedzina funkcji
edge	krawędź
equivalent	równoważny
estimate (v, n)	szacować szacunek liczba szacunkowa
even	parzysty (o liczbie)
exponent	wykładnik (potęgowy)
exponential function	funkcja wykładnicza
factor	czynnik współczynnik mnożnik
factorial	silnia
finite	skończony
fraction	ułamek
hypotenuse	przeciwprostokątna
image of a function	obraz funkcji
imaginary number	liczba urojona
infinite (set)	nieskończony (zbiór)
infinity	nieskończoność
integer	liczba całkowita
integral (n, adj)	całka całkowy
interval	interwał przedział
leg	bok trójkąta równoramiennego

logarithm	logarytm
logarithmic (function)	funkcja logarytmiczna
mapping (one-to-one mapping)	odwzorowanie (wzajemnie jednoznaczne)
matrix **matrices** (pl)	macierz
median	mediana
meeting point	punkt przecięcia
midpoint	środek
minus	minus znak odejmowania
multiplication	mnożenie
numerator	licznik (ułamka)
oblique	ukośny
obtuse angle	kąt rozwarty
obtuse- angled triangle	trójkąt rozwartokątny
odd	nieparzysty (o liczbie)
parallel	równoległy
per cent	procent
permutation	permutacja
plane (n, adj)	płaszczyzna płaski
plane figure	figura płaska
polygon	wielobok wielokąt
polynominal	wielomian
proposition	twierdzenie teza
probability	prawdopodobieństwo
pyramid	ostrosłup
sphere	sfera
quadrangle	czworokąt
quadrant	ćwiartka
quotient	iloraz

raise to the x-th power	podnieść do x-tej potęgi
real number	liczba rzeczywista
rhomb rhombus	romb
right angle	kąt prosty
root	pierwiastek
round brackets **parentheses**	( )
trapezium	trapez
vertex	wierzchołek
sine	sinus
cosine	cosinus
tangent (n, adj)	styczna styczny
tangent of an angle	tangens kąta
cotangent of an angle	cotangens kąta

Abbreviations:
n-noun-rzeczownik
v-verb-czasownik
adj-adjective-przymiotnik
pl-plural-liczba mnoga

Appendix B

Mathematical formulas

Many universities stress the importance of studying mathematics during computer science courses, mainly in the first year. The point is simply not to make students learn something difficult and perhaps, somewhat boring but to give them the ability to use the universal language of mathematics, without which computer science could not exist. So, why not to learn mathematical formulas in English to enable you to use them in the global world of science?

Listen to the recording

1.	**a + b**	a plus b	addition
2.	**a - b**	a minus b	subtraction
3.	**a * b**	a multiplied by b	multiplication
4.	**a / b**	a divided by b	division
5.	**¬ a**	not a	
6.	**a ∧ b**	a and b	
7.	**a ∨ b**	a or b	
8.	**a ⊥ b**	a is orthogonal/perpendicular to b	
9.	**a || b**	a is parallel to b	
10.	a ∈ A	a is contained in A	
11.	$\mathbf{a^2}$	a raised to the second power	a squared
12.	$\mathbf{a^x}$	a raised to the xth power	a to the power of x
13.	**a ≤ b**	a is less than or equal to b	
14.	**a > c**	a is greater than c	
15.	**2x + 4 = 8**	2 x plus 4 equals /is equal to 8	
16.	**2/8 = 1/4**	2 over 8 equals 1 over 4	
17.	**| -5| = 5**	the absolute value of -5 equals 5	

18.	$\log_a b = c$	the logarithm of b to the base a equals c
19.	$A \supset B$	A includes B B is a subset of A
20.	$\forall x \in A \quad x \in B$	for all x in A x is contained in B x is in B x lies in B
21.	$\sqrt[n]{a} \quad \sqrt{a}$	the n^{th} root of a square root
22.	a_n	a sub n

Appendix C

Greek alphabet

The Greek alphabet is the basis of what used to be a very important and, in some ways, universal language in ancient Europe. Nowadays, this alphabet constitutes a significant part of the universal language of mathematics. As mathematics is fundamental to most modern sciences, with computer sc ience foremost among them, it is vital to be familiar with it.

Listen to the recording

Greek letter	English name
Α α	Alpha
Β β	Beta
Γ γ	Gamma
Δ δ	Delta
Ε ε	Epsilon
Ζ ζ	Zeta
Η η	Eta
Θ θ	Theta
Ιι	Iota
Κ κ	Kappa
Λ λ	Lambda
Μ μ	Mu
Ν ν	Nu
Ξ ξ	Xi
Ο ο	Omicron
Π π	Pi
Ρ ρ	Rho

Σ σ ς	Sigma
Τ τ	Tau
Υ υ	Upsilon
Φ ϕ	Phi
Χ χ	Chi
Ψ ψ	Psi
Ω ω	Omega

English++

English for Computer Science Students

Glossary

English++

Glossary

This glossary contains entries for the keywords or key terms cited in bold in English++ book. They have been selected from the reading and listening texts and then their definitions have been provided by the members of English++ team.

agile development - a way of building software

Agile Methodology - a particular method of producing software, emphasizing self-organizing teams and broad customer-programmer contact

Agile Software Development - a conceptual framework for software development which promotes development iterations throughout the life-cycle of the project

alignment - the adjustment of an object in relation to other objects, or a static orientation of some object or set of objects in relation to others

Api - a source code interface that an operating system, library or service provides to support requests made by computer programs

artery - one of the tubes that carries blood from the heart to the rest of the body

ASP (Action Server Pages) - Microsoft's first server-side script engine for dynamically-generated web pages

autonomous differential equation - a system of ordinary differential equations which does not depend on the independent variable

bandwidth - the amount of data that can be transferred through a specific path in the network (usually expressed in kb/s [kilo bits per second])

Barok trojan - this trojan horse gathers information such as user name, IP address and passwords, and attempts to send the information to the creator of the virus

binary code - code using a string of 8 binary digits to represent characters

BitTorrent - a protocol for sharing and distributing computer files via the Internet

bot - a program used on the Internet that performs a repetitive function such as posting a message to multiple news-groups or searching for information or news

botnet - a collection of software robots, or bots linked and cooperating with each other

broadcast - the action of sending data by one computer in the network to all other computers that are available inside the network

buffer - a region of memory used to temporarily hold data while it is being moved from one place to another

buffer cache - a collection of data duplicating original values stored elsewhere or computed earlier, where the original data is expensive to fetch (owing to longer access time) or to compute, compared to the cost of reading the cache

bug - an error in software, something that spoils the flow of the program

CAD - Computer Aided Design

Cauchy problem - see IVP

circuit switching network - a network in which computers establish a constant bandwidth connection before they start to share any data

code, coding - refers to the process of transforming a concept into a program code

cognitive science - the scientific study of mind or intelligence based on relevant fields, including psychology, philosophy, neuroscience, linguistics, anthropology, computer science, and biology

computational - adj.1. involving computers, 2. that can be computed

computational complexity - an area of computer design dealing with the problems of algorithms and their ability to solve a given problem

Computer Generated Imagery (CGI) - an application of the field of computer graphics (or, more specifically, 3D computer graphics) to special effects in films, television programs, commercials and simulation

computer vision - a branch of artificial intelligence that deals with computer processing of images from the real world

confine - to define boundaries; to limit the extent (of an activity)

contract - a legally binding exchange of promises or agreement between parties which is enforceable by law

cookbook - a book of recipes and solutions for different problems on writing software, usually connected with a particular programming language and platform

correlate (with) - to have a close similarity, connection or causal relationship with

cyber vandalism - cyber attacks that deface web pages

cyber war - the usage of computers and the Internet in conducting warfare in the cyberspace

database - software designed for holding large amounts of data

database schema - a description of the tables and views in a database together with the relationships between them

DBA - database administrator; the main role of a database administrator has to do with overseeing the installation and ongoing function of software on a system designed for use by a number of users

decorator pattern - a design pattern that allows new/additional behaviour to be added to an existing class dynamically

Denial of Service - a kind of cyber attack which results in unavailability of service; it uses two methods: forcing the targeted computer(s) to reset, thus consuming its resources such that it can no longer provide its intended service; or obstructing the communication media between the intended users and the victim so that they can no longer communicate adequately

design pattern - a general repeatable solution to a commonly occurring problem in software design; a design pattern is not a finished design that can be transformed directly into code; it is a description or template for how to solve a problem that can be used in many different situations

desktop - a personal computer used at home or work (stationary)

developer - person who develops the software

DFT - Discrete Fourier Transform

differentiable - a real function is said to be differentiable at a point if its derivative exists at that point

differential equation - a mathematical equation for an unknown function of one or several variables that relates the values of the function itself and of its derivatives

discrete - not supporting or requiring the notion of continuity; discrete objects are countable sets such as integers

downloading - getting/transferring files (software, music, films, etc.) from a remote computer via the Internet

extreme programming (XP) - a programming methodology which is very closely connected with the Agile style

FFT - Fast Fourier transform

Firefox - one of the most popular web browsers (after Internet Explorer)

FOSS - the acronym for Free and Open Source Software; the software that can be used, studied, and modified without restriction

gamut - a complete range or extent

general solution - a general solution of a differential equation is the set of all of its particular solutions, often expressed using a constant C (or K) which could have any fixed value

Gnome - one of the biggest window managers for Linux

GNU - a computer operating system composed entirely of free software, initiated in 1984 by Richard Stallman

Google - an American public corporation whose domain is open-source software and whose earning revenue from advertising related to its Internet search, web-based e-mail, online mapping, office productivity, social networking and video sharing activities

GPL - a widely used free software license, originally written by Richard Stallman for the GNU project

GUI (Graphical User Interface) - a type of user interface which allows people to interact with a computer and computer-controlled devices

hacktivism - combination of words: "hack(er)" and "activism"

hardware - all the physical parts of the computer

heterogenous - consisting of various parts that are very different from each other

heuristic hypothesis - a hypothesis that has a very high probability of being true on the basis of reasoning and past experience

homogeneous linear differential equation - a differential equation is said to be homogeneous if there is no isolated constant term in the equation, e.g., each term in a differential equation for y has y or some derivative of y in each term

identity map - a database access design pattern used to improve performance by providing a context-specific in-memory cache to prevent duplicate retrieval of the same object data from the database

incoherent - confused and inconsistent, illogical

IVP (initial value problem) - an ordinary differential equation together with specified value, called the initial condition, of the unknown function at a given point in the domain of the solution

Internet - a global network connecting millions of computers

Internet provider - a company that sells bandwidth and access to the Internet

interval - a set of real numbers with the property that any number that lies between two numbers in the set is also included in the set; for example, the set of all numbers x satisfying 0 <= x <= 1 is an interval which contains 0 and 1, as well as all numbers between them; the set of positive numbers is also an interval

kernel - the central component of most computer operating systems (OS); its functions include managing the system's resources (the communication between hardware and software components)

keyframe - (or key frame) in animation and film making is a drawing which defines the starting and ending points of any smooth transition texture - a bitmap image a plied to a surface in computer graphics

Linux kernel - Unix-like operating system kernel

LiSP - The Linux Phone Standards Forum, a consortium founded by a group of telephony operators, device manufacturers, silicon and software vendors who have a strategic focus on Linux telephony

mainframe - computer used by large companies for critical operations like financial transaction processing

maintenance - "looking after" the written software

malware - software designed to infiltrate or damage a computer system without the owner's informed consent

mathematical model - an abstract model that uses mathematical language to describe a system

matricize - changing sound into parts and writing into a matrix

matrix - a rectangular table of elements (or entries), which may be numbers or, more generally, any abstract quantities that can be added and multiplied.

Minix - free/open source, Unix-like operating system (OS) based on a microkernel architecture

model for the system - a set of differential equations describing the behaviour of a system

Monte Carlo method - a computational algorithm which relies on repeated random sampling to compute its results

morphing - a special effect in motion pictures and animations that changes (or morphs) one image into another through a seamless transition

motion capture - a technique of recording the actions of human actors and using that information to animate digital character models in 3D animation

MPL - Microsoft Public Licence

MRL - Microsoft Reciprocal Licence

network protocol - a set of rules that set out how to establish communication between two or more computers over the network

ODE (ordinary differential equation) - a relation that contains functions of only one independent variable, and one or more of its derivatives with respect to that variable

OHA - Open Handset Alliance™, a group of more than 30 technology and mobile companies who have developed Android, the first complete, open, and free mobile platform

operating system - the software that manages the sharing of the resources of a computer and provides programmers with an interface used to access those resources

OS (Operating System) - a piece of software that is responsible for the management of a computer and, above all, running programs (examples: MS Windows, Linux, Mac OS)

pair programming - a software development technique in which two programmers work together at one keyboard: one types in code while the other reviews each line of code as it is typed in; the person typing is called the driver; the person reviewing the code is called the observer or navigator; the two programmers switch roles frequently

patterns book - a book on design patterns

PDE - Partial Differential Equations

peering - the arrangement of traffic exchange between the Internet service providers (ISPs)

Podcast - a series of digital-media files which are distributed over the Internet using syndication feeds for playback on portable media players and computers

punched card - a card on which data can be recorded in the form of punched holes

rate of change - an indicator showing the difference between parameters in a specific unit of time

refactoring - making changes to a computer system (e.g. source code, database) that improve its readability and simplify its structure without adding any new functionality; for example, changing the names of variables to be more readable

regexp - an abbreviation for regular expressions. In computing, regular expressions provide a concise and flexible means for identifying strings of text of interest, such as particular characters, words, or patterns of characters

rendering - the process of generating an image from a three dimensional object by means of computer programs

router - a network device that groups computers in the network and establishes an area inside it

routing - the process of selecting paths in a network along which data can be sent between computers (through the router)

SCO - a software company formerly called Caldera Systems and Caldera International; after acquiring the Santa Cruz Operation's Ser er Software and Services divisions, as well as UnixWare and OpenServer technologies, the company changed its focus to UNIX; later on, Caldera changed its name to The SCO Group to reflect that change in focus

sector-based static lighting - the texture with information about lighting, which is not affected by the angle or position of the light source

separable differential equation - a class of equations that can be separated into a pair of integrals

skybox - a method to create a background to make a computer and video game location look bigger than it really is

social engineering - practice of obtaining confidential information by manipulating users

software - in general, computer programs

software development process
- a structure imposed on the development of a software product. Synonyms include software life cycle and software process; there are several models for such processes, each describing approaches to a variety of tasks or activities which take place during the process

Software documentation (or **source code documentation)** - written text which accompanies computer software; it explains either how it operates or how to use it, and may mean different things to people in different roles

software license (UK spelling: **licence)** - there are several important IT licenses like GNU GPL(open source), Mozilla Public License or Microsoft Public License

stochastic process - a process with an indeterminate or random element as opposed to a deterministic process that has no random element

streaming protocol - a set of rules to ensure that data will be supported in real time

Suse, Red Hat, Ubuntu - examples of Linux distributions

texture - a bitmap image applied to a surface in computer graphics

to prune - to cut parts of the map invisible for the player

Unix - a computer operating system originally developed in 1969 by a group of AT&T employees at Bell Labs including Ken Thompson, Dennis Ritchie and Douglas Ilroy

up-front (design, analysis, requirements) - the oldest method of building software

up-front designing - describes a method in which the design phase of a project is completed before the coding of the project begins

user interface - the aggregate of means by which people, the users, interact with the system, a particular machine, device, computer program or other complex tools; the user interface provides means of input, allowing the users to manipulate a system and output, allowing the system to produce the effects of the users' manipulation

VBScript - Visual Basic Scripting Edition - an Active Scripting (technology used in Windows to implement component-based scripting support) language developed by Microsoft

VFS (Virtual File System) - an abstraction layer on top of a more concrete file system

Windows - an operating system