UNIVERSITÄT BERN

Software Evolution

Tudor Gîrba
www.tudorgirba.com

Lecturers:	Dr. Tudor Gîrba (girba@iam.unibe.ch) Prof. Oscar Nierstrasz (oscar@iam.unibe.ch)
Assistant:	Jorge Ressia
Lecture:	Thursdays, 15:15 - 17:00
Lab:	Thursdays, 17:15 - 18:00
Web:	http://scg.unibe.ch/Teaching/EVO http://scglectures.unibe.ch/evo
Mailing list:	evo-vorlesung@iam.unibe.ch

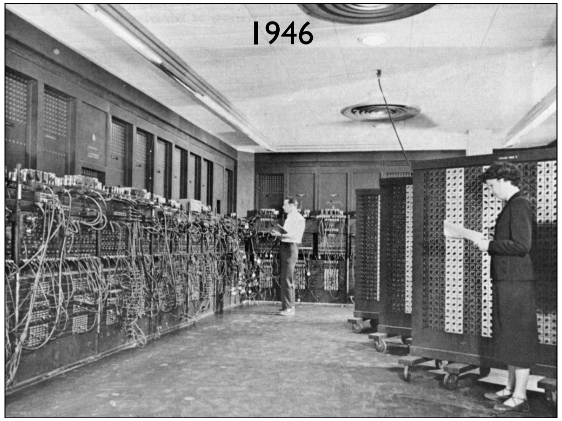
http://scglectures.unibe.ch/evo will be online in a few days.

But, software does not rot

Software Evolution

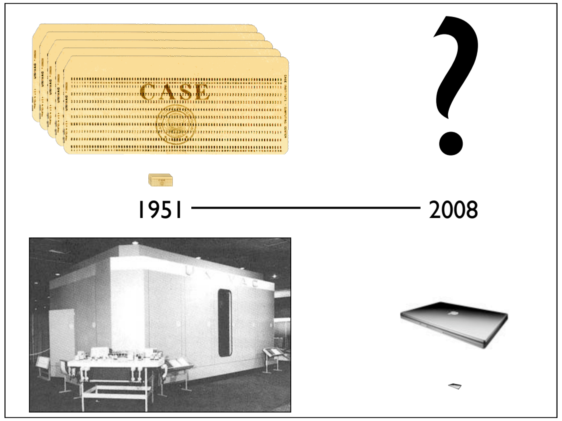
So, why is this a problem?

The course is called Software Evolution. Why is this a problem?



This is a picture of ENIAC I (1946).
(http://en.wikipedia.org/wiki/ENIAC)

The interesting thing about it is that you can see the complexity of the program in how intricate the cables are. Another interesting thing is that you see who is working on what.



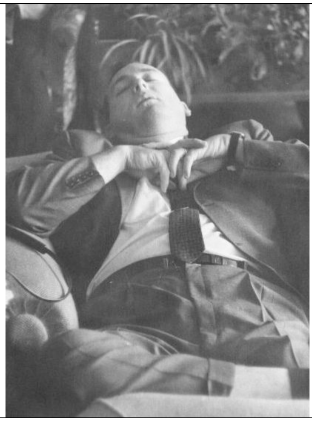
UNIVAC is the first “mass produced” computer. They built 40 pieces, each costing 1 million dollars.
http://en.wikipedia.org/wiki/UNIVAC
http://www.city-net.com/~ched/help/general/tech_history.html

From UNIVAC on, the program became hidden. When people needed to name the building, they said it’s the hardware because it was a heavy thing, hard to build and manipulate. As opposite to that, the program was “softer”, just a bunch of cards.

The machine was 25 feet by 50 feet in length, contained 5,600 tubes, 18,000 crystal diodes, and 300 relays. It utilized serial circuitry, 2.25 MHz bit rate, and had an internal storage capacity 1,000 words or 12,000 characters.

Because it was hard and heavy (13 tons), we wanted to make it smaller and more manageable. So, after 50 years we can carry the computer with us. But what happened with the “soft” thing?

NATO Software Engineering Conferences (1968, 1969)
http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF
http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1969.PDF
http://en.wikipedia.org/wiki/History_of_software_engineering
http://en.wikipedia.org/wiki/Software_crisis
http://www.princeton.edu/~hos/mike/articles/sweroots.pdf



Some of the sessions were very intense.

[On software crisis] There is a widening gap between ambitions and achievements in software engineering.

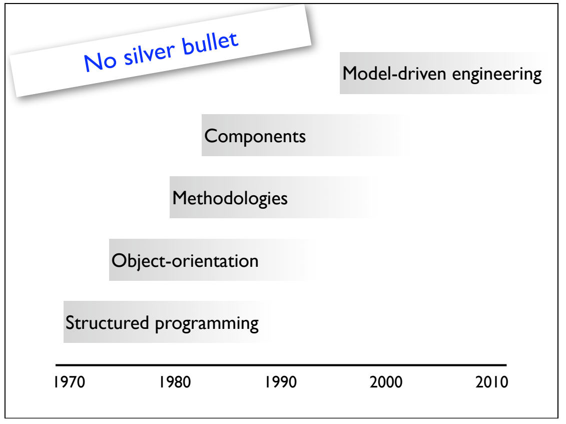
NATO, 1968

The consensus of the conference was that there is a software crisis, even if not everyone liked the term crisis:
- Kolence: “There are many areas where there is no such thing as a crisis — sort routines, payroll applications, for example. It is large systems that are encountering great difficulties.”
- Kolence: “I do not like the use of the word ‘crisis’. It’s a very emotional word. The basic problem is that certain classes of systems are placing demands on us which are beyond our capabilities and our theories and methods of design and production at this time.”

As long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now when we have gigantic computers, programming has become an equally gigantic problem.

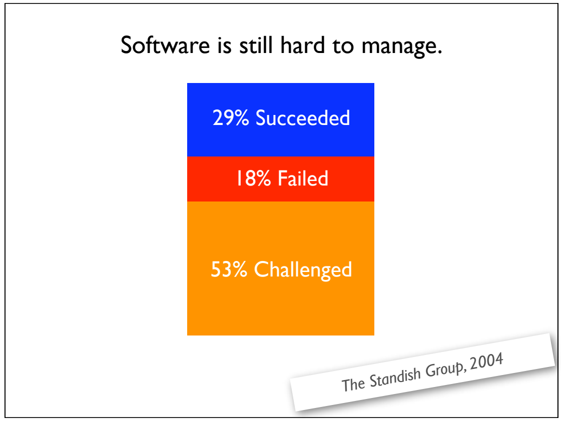
Edsger Dijkstra, 1972

http://en.wikipedia.org/wiki/Software_crisis
http://www.cs.utexas.edu/~EWD/transcriptions/EWD03xx/EWD340.html



Software crisis was tackled in several ways over the past 40 years. Yet, no solution provided a silver bullet.

Related article:
http://www.virtualschool.edu/mon/SoftwareEngineering/BrooksNoSilverBullet.html



Yet, after 50 years, software is not "soft" anymore. It is heavy and difficult to manage.

The Standish Group defined a project to be successful if it is both on time and in budget. The challenged projects were at least significantly over time or over budget. The failed projects were cancelled altogether.

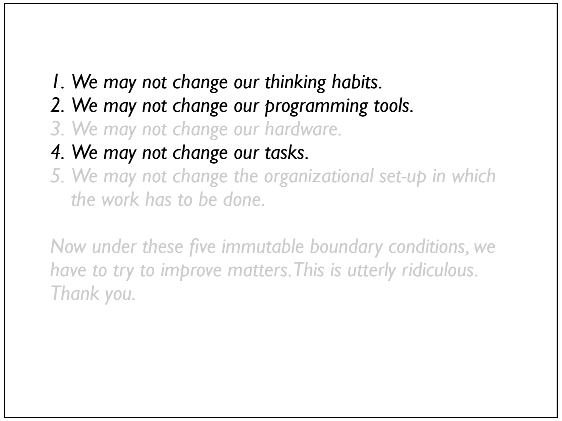
1. *We may not change our thinking habits.*
2. *We may not change our programming tools.*
3. *We may not change our hardware.*
4. *We may not change our tasks.*
5. *We may not change the organizational set-up in which the work has to be done.*

Now under these five immutable boundary conditions, we have to try to improve matters. This is utterly ridiculous. Thank you.

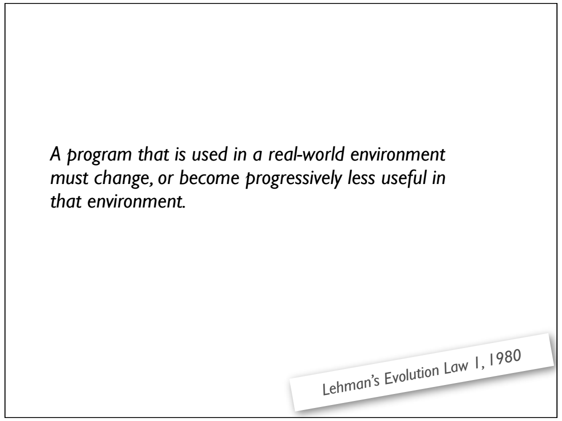
Dijkstra, 1969

http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1969.PDF

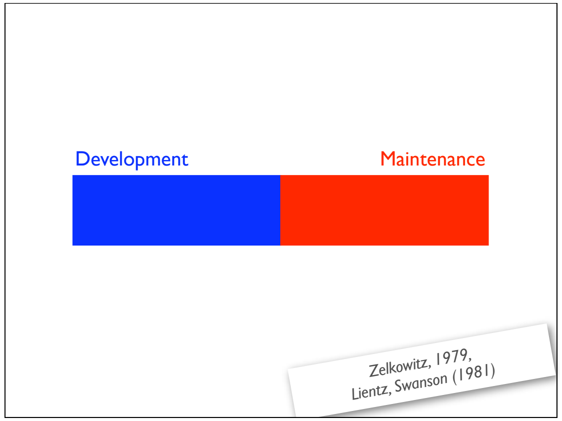
Let's start from what Dijkstra said in 1969 during the NATO 1969 conference, in a discussion related to software crisis.



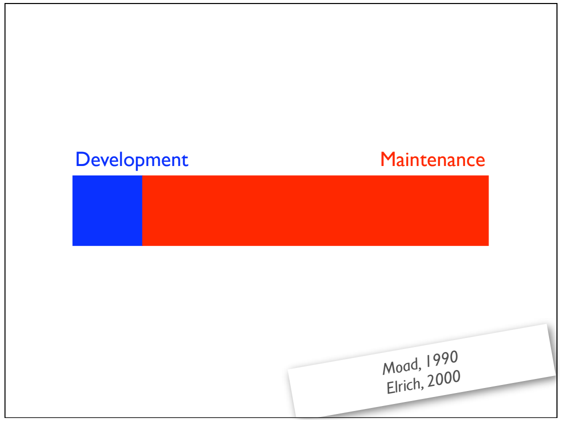
In this lecture, we will focus on just three of the five points: a new way of looking at development, a new set of tasks and a new set of tools that help us accomplish these tasks.



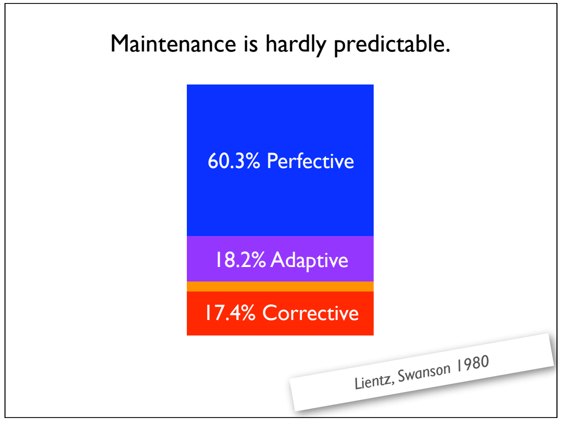
Manny Lehman and Les Belady, Program Evolution: Processes of Software Change, London Academic Press, London, 1985. (ftp://ftp.umh.ac.be/pub/ftp_infofs/1985/ProgramEvolution.pdf)
M.M. Lehman. "Programs, life cycles, and laws of software evolution", Proceedings of IEEE, pages 1060–1076, September 1980
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.4491



The following website contains a list of useful pointers related to software maintenance costs:
http://users.jyu.fi/~koskinen/smcosts.htm



The following website contains a list of useful pointers related to software maintenance costs:
http://users.jyu.fi/~koskinen/smcosts.htm



Bennett Lientz and Burton Swanson, Software Maintenance Management, Addison Wesley, Boston, MA, 1980.

Keith H. Bennett and Vaclav T. Rajlich, “Software maintenance and evolution: a roadmap,” ICSE '00: Proceedings of the Conference on The Future of Software Engineering, ACM Press, New York, NY, USA, 2000, pp. 73—87. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.8225

legacy |ˈlegəsē|

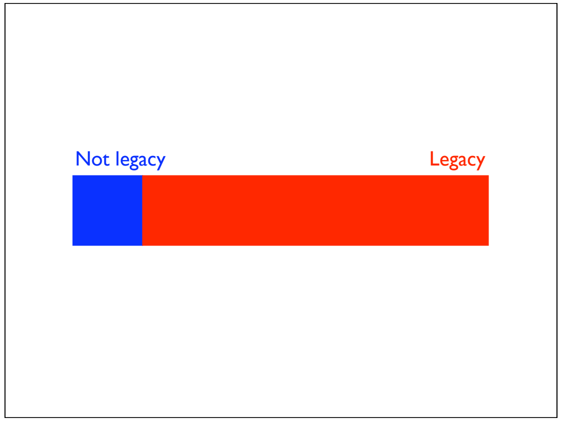
an amount of money or property left to someone in a will.
a thing handed down by a predecessor.

Maintenance is about legacy.

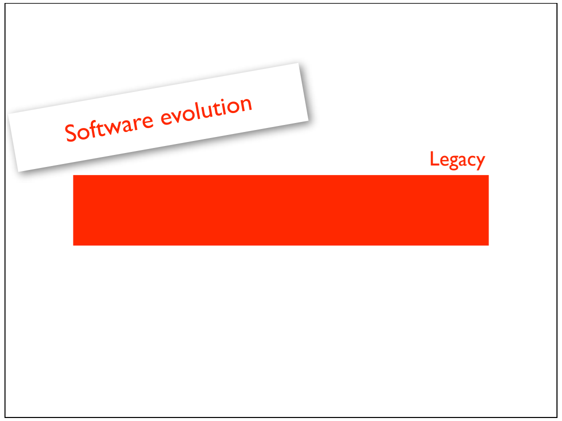
legacy system |ˈlegəsē ˈsistəm|

an inherited software system that is valuable.

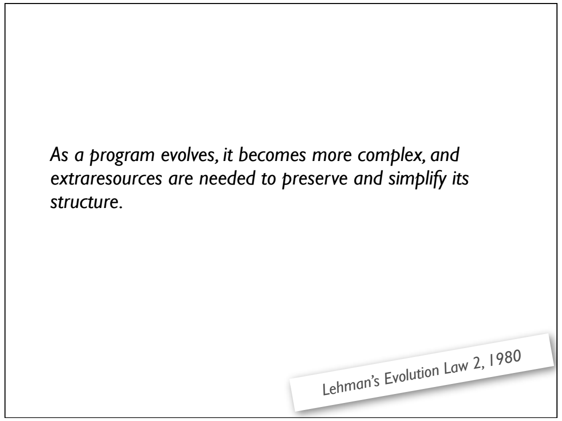
In particular it is about legacy systems.



In the “classical” view on software, development is about not legacy code, and maintenance is about legacy code.
http://users.jyu.fi/~koskinen/smcosts.htm

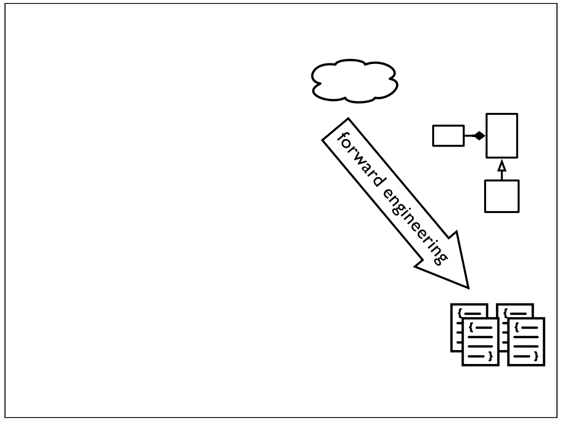


Most of our effort should be concentrated on dealing with legacy code. Thus, instead of making a distinction between development and maintenance, we better just consider the entire effort as a continuous evolution.



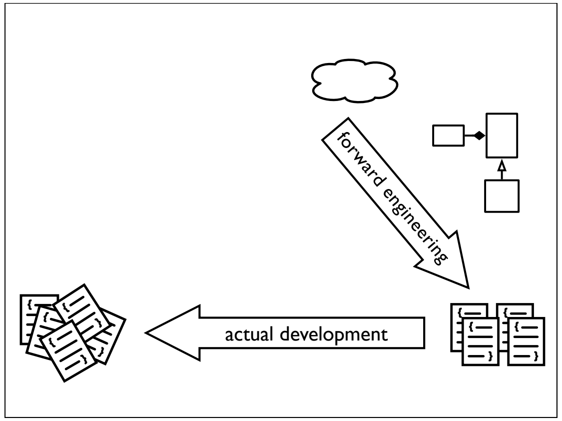
Manny Lehman and Les Belady, Program Evolution: Processes of Software Change, London Academic Press, London, 1985. (ftp://ftp.umh.ac.be/pub/ftp_infofs/1985/ProgramEvolution.pdf)

M.M. Lehman. "Programs, life cycles, and laws of software evolution", Proceedings of IEEE, pages 1060–1076, September 1980
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.4491

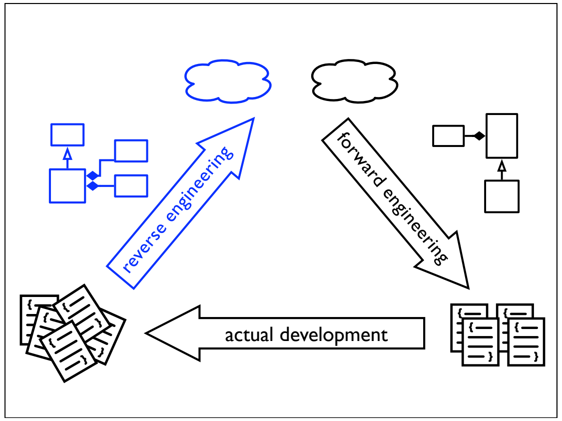


How do these legacy systems look like?

All projects start with great intensions. The idea is clear, the plans are neat and the implementation is tidy.



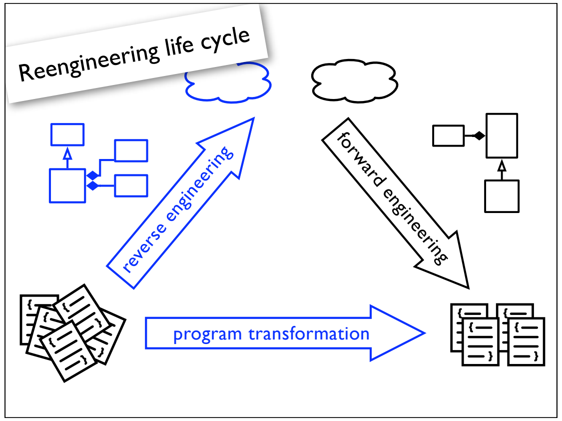
The problem is that in most projects, the actual development happens only at the code level, with only little documentation, and several years later the system is not tidy anymore.



Forward Engineering is the traditional process of moving from high- level abstractions and logical, implementation-independent designs to the physical implementation of a system.

Reverse Engineering is the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction.

Elliot Chikofsky and James Cross II, "Reverse Engineering and Design Recovery: A Taxonomy," IEEE Software, vol. 7, no. 1, January 1990, pp. 13—17.



Reengineering ... is the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form.

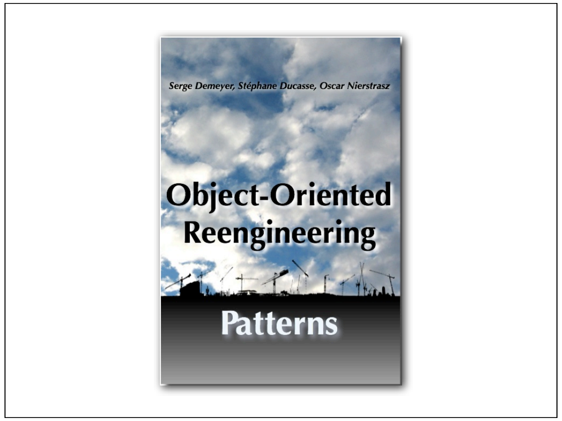
Elliot Chikofsky and James Cross II, "Reverse Engineering and Design Recovery: A Taxonomy," IEEE Software, vol. 7, no. 1, January 1990, pp. 13—17.

02	Reverse engineering
03	
04	
05	Metrics and problem detection
06	Software visualization
07	Building reverse engineering tools
08	Dynamic analysis
09	History analysis
10	Software understanding in the large
11	Restructuring
12	Testing and Migration
13	TBA
14	

What we will do.

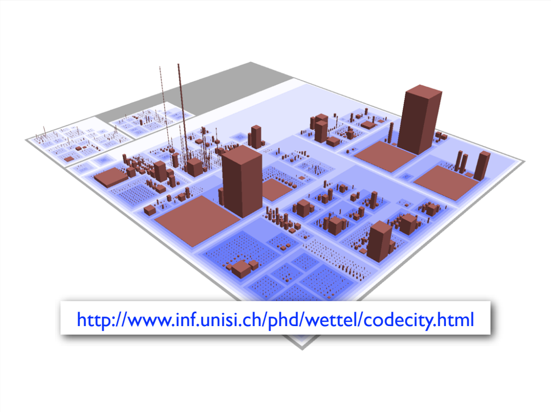
02	Reverse engineering	Understanding a legacy system
03	Understanding a legacy system	
04	Results presentations	
05	Metrics and problem detection	Using Moose
06	Software visualization	Using Moose
07	Building reverse engineering tools	Using Moose. New case studies
08	Dynamic analysis	Results presentations
09	History analysis	Building an analysis tool
10	Software understanding in the large	Building an analysis tool
11	Restructuring	Results presentations. Surprise project
12	Testing and Migration	Surprise project
13	TBA	Results presentations
14	Final exam	

What you will do.

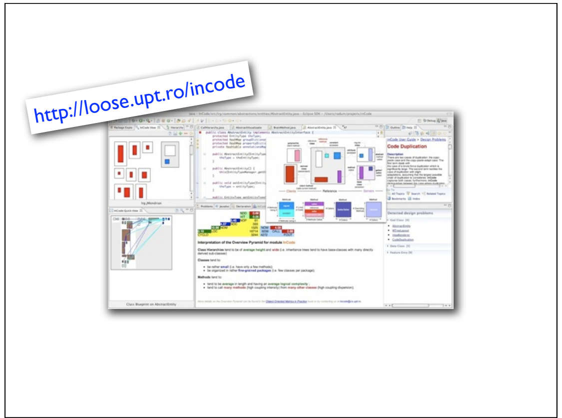


This book is used as inspiration for the course. The book is now open source and can be found at:
http://www.iam.unibe.ch/~scg/OORP/

Moose is a platform for software analysis. It was started at Software Composition Group, University of Bern, and it is now used in several universities. More details can be found at:
http://moose.unibe.ch



Code City shows software using a city metaphor. Code City is built by Richard Wettel and is based on Moose. More details at:
http://www.inf.unisi.ch/phd/wettel/codecity.html



inCode is an Eclipse plugin dedicated to quality assurance. It developed at the LOOSE Research Group, Politehnica University of Timisoara. More details can be found at:
http://loose.upt.ro/incode

The lab assignments are mandatory to get in the exam

Please subscribe to the mailing list:

http://www.iam.unibe.ch/mailman/listinfo/evo-vorlesung

Please form teams and send them by email to:

girba@iam.unibe.ch