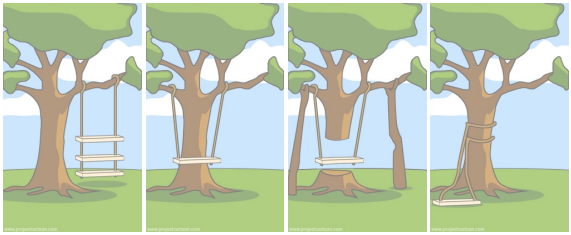
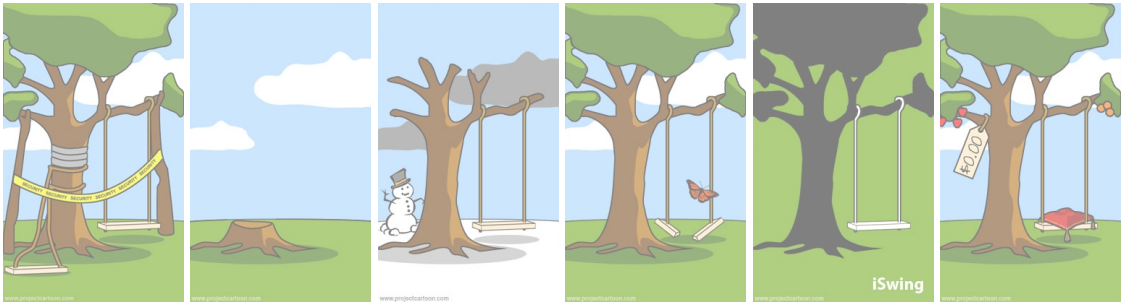
Software Engineering

11. Software Evolution | Thomas Thüm | May 6, 2021

Software Engineering I



Software Engineering II



how patches were applied	how it was supported	when it was delivered	how it performed under load	what marketing advertised	the open source version
software evolution	**software maintenance**	**configuration management**	**compilation and static analyses**	**software product lines**	**open-source software**

Lecture Overview

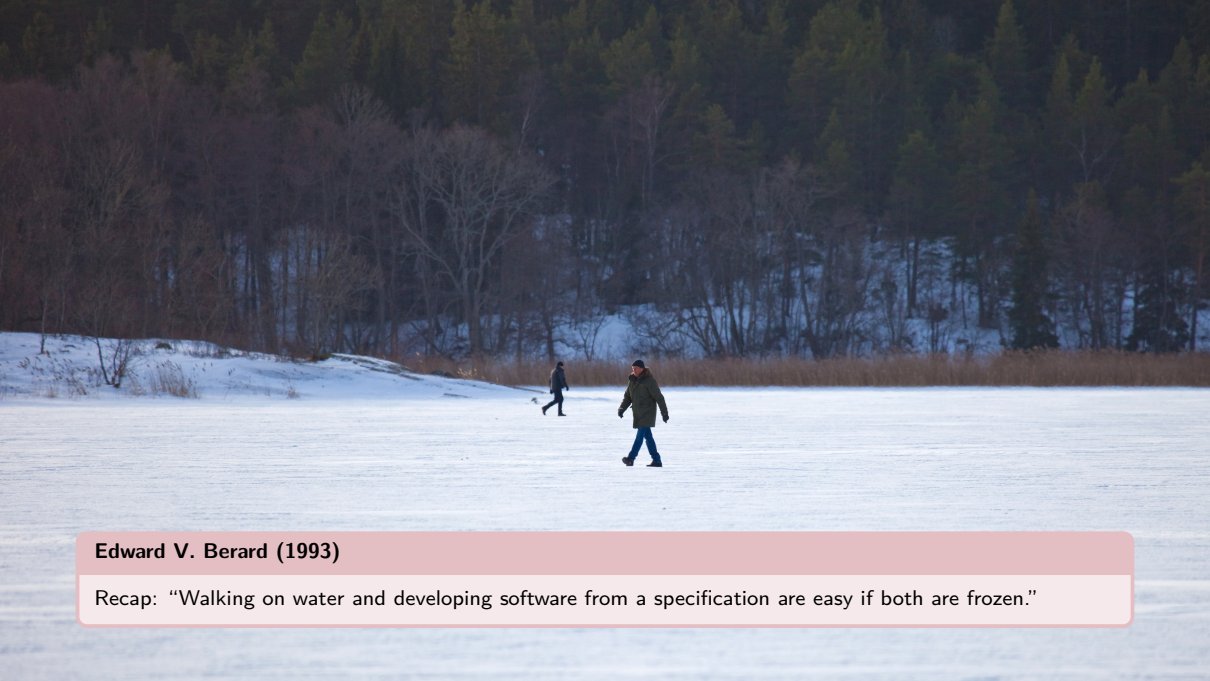
1. Software Evolution vs Software Engineering I
2. Programming Wisdom on Software Evolution
3. Smells and Refactorings

Software Evolution vs Software Engineering I

Change is Inevitable

Sommerville:

"Change is inevitable in all large software projects. The **system requirements change** as businesses respond to external pressures, competition, and changed management priorities. As **new technologies become available**, new approaches to design and implementation become possible. Therefore whatever software process model is used, it is essential that it can **accommodate changes** to the software being developed. [...] It may then be necessary to **redesign** the system to deliver the new requirements, **change any programs** that have been developed, and **retest** the system."



Edward V. Berard (1993)

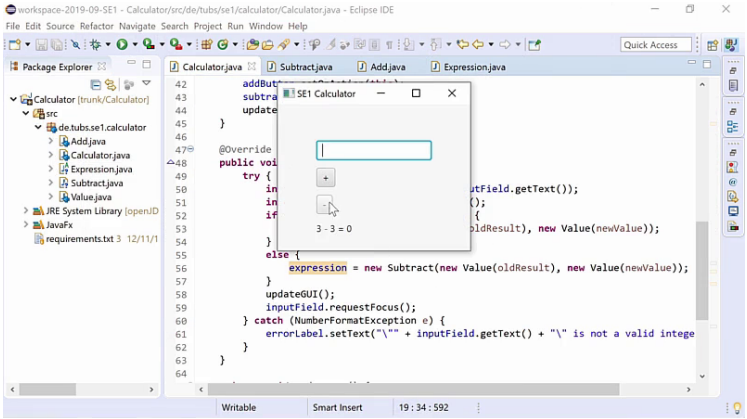
Recap: “Walking on water and developing software from a specification are easy if both are frozen.”

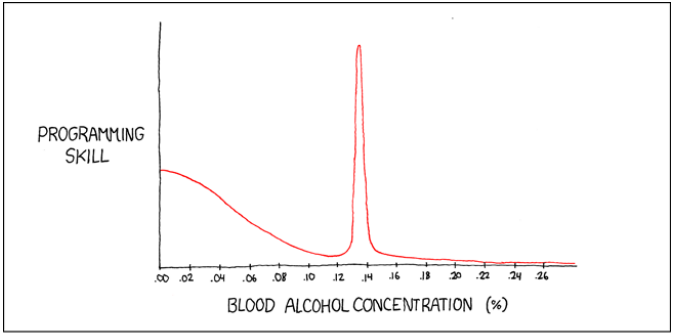
Changes to Requirements

Motivation [Sommerville]

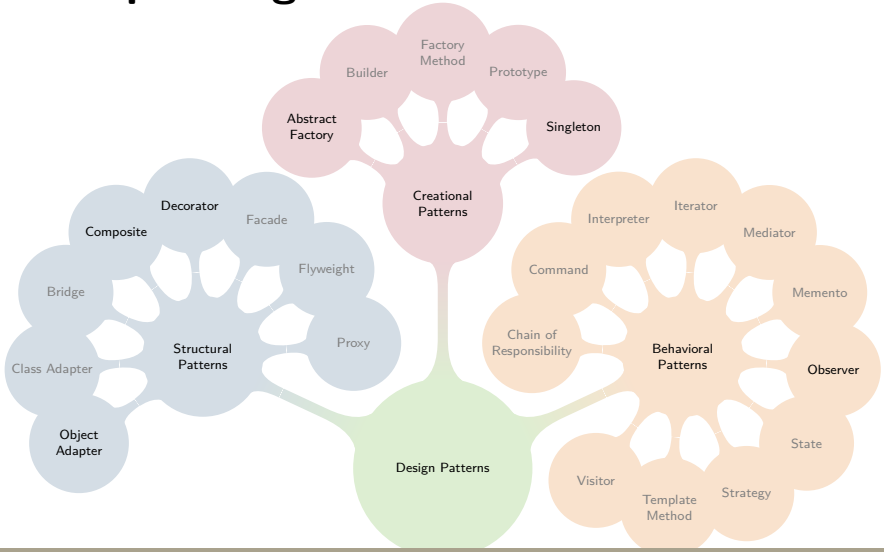
- changed problem understanding for all stakeholders
- new or updated hardware
- interfacing to other systems
- new legislation and regulations
- new compromise on conflicting requirements of different stakeholders

Recap: Extending Thomas' Calculator



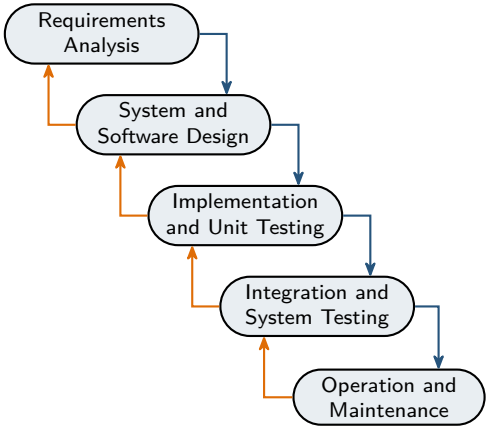

PROGRAMMING SKILL
.00 .02 .04 .06 .08 .10 .12 .14 .16 .18 .20 .22 .24 .26
BLOOD ALCOHOL CONCENTRATION (%)
CALLED THE BALLMER PEAK, IT WAS DISCOVERED BY MICROSOFT IN THE LATE 80's. THE CAUSE IS UNKNOWN, BUT SOMEHOW A B.A.C. BETWEEN 0.129% AND 0.138% CONFERS SUPERHUMAN PROGRAMMING ABILITY.
HOWEVER, IT'S A DELICATE EFFECT REQUIRING CAREFUL CALIBRATION— YOU CAN'T JUST GIVE A TEAM OF CODERS A YEAR'S SUPPLY OF WHISKEY AND TELL THEM TO GET CRACKING.
...HAS THAT EVER HAPPENED?
REMEMBER WINDOWS ME?
I KNEW IT!

Recap: Design Patterns [Gang of Four (GoF)]

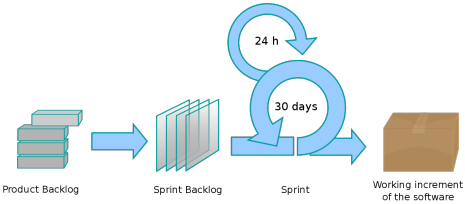


Recap: Process Models

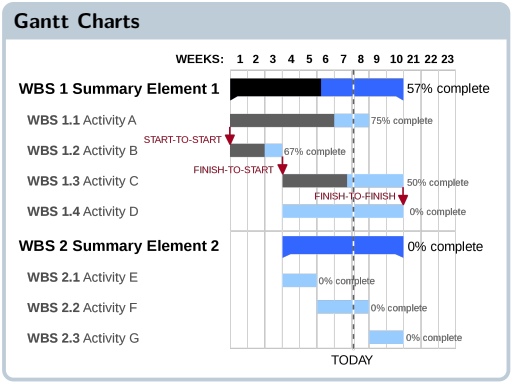
Waterfall Model



Scrum



Recap: Project Management



Regression Testing [Ludewig and Lichter]

Regression Testing (Regressionstesten)

"Selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements."

Goal

- every change has desired effects
- regression testing aims to detect undesired side-effects

Assumptions

- minor change to the program behavior
- program was largely correct before
- results only differ in old and new version where desired

Characteristics

- requires to document test results before changes
- actual results are not only compared to specification, but also to previous results for the same inputs

Software Evolution vs Software Engineering I

Lessons Learned

- Incremental development and changed requirements cause evolution
- Design patterns ease evolution
- Regression testing incorporates results of new **and old** version
- Further Reading: Sommerville, 2.3 Coping with Change and 4.6 Requirements Change
- Further Reading: Ludewig and Lichter, 19.1.3 (Der Regressionstest)

Practice

- Give an example of a change that is amendable to regression testing and an example that is not
- Upload your examples to Moodle: `https://moodle.uni-ulm.de/mod/moodleoverflow/discussion.php?d=2735`

Lecture Contents

Programming Wisdom on Software Evolution

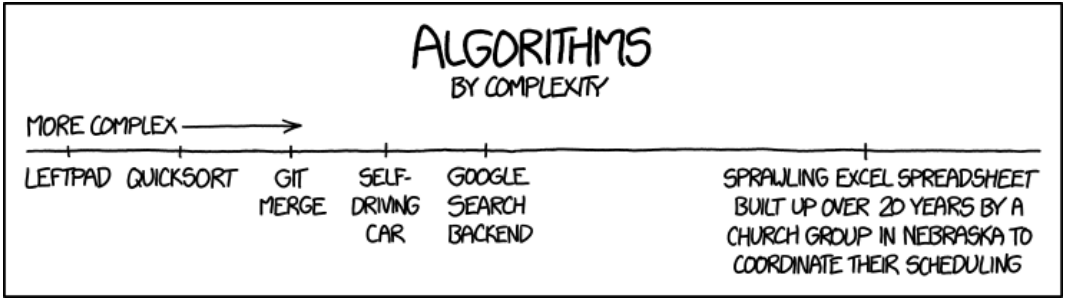
Avoiding Complexity

Richard E. Pattis

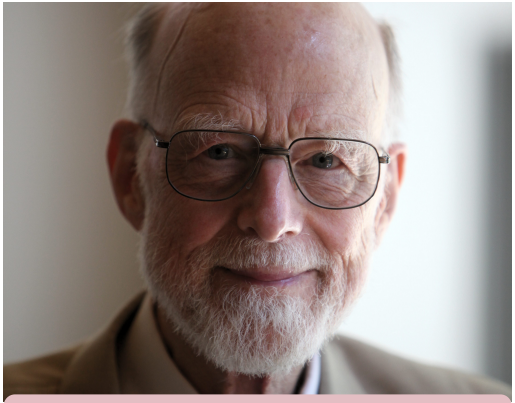
"When debugging, novices insert corrective code; experts remove defective code."

Ken Thompson (born 1943)

"One of my most productive days was throwing away 1,000 lines of code."


ALGORITHMS
BY COMPLEXITY
MORE COMPLEX
LEFTPAD
QUICKSORT
GIT MERGE
SELF-DRIVING CAR
GOOGLE SEARCH BACKEND
SPRAWLING EXCEL SPREADSHEET BUILT UP OVER 20 YEARS BY A CHURCH GROUP IN NEBRASKA TO COORDINATE THEIR SCHEDULING

Increasing Complexity



Tony Hoare (born 1934)

"Inside every large program, there is a small program trying to get out."

Meir "Manny" Lehman (1925–2010)

"An evolving system increases its complexity unless work is done to reduce it."

Lehman's Laws of Software Evolution [Lehman et al. 1997]

Lehman's Laws (excerpt)

- Continuing Change: systems must be continually adapted to stay satisfactory
- Increasing Complexity: complexity increases during evolution unless work is done to maintain or reduce it
- Conservation of Familiarity: satisfactory evolution excludes excessive growth
- Continuing Growth: functionality must be continually increased to maintain user satisfaction
- Declining Quality: quality will decline unless rigorously maintained and adapted to operational environment changes

Essence of the Laws

- software that is used will be modified
- when modified, its complexity will increase (unless one does actively work against it)

Consequences

- functional changes are inevitable
- changes are not necessarily a consequence of errors (e.g., in requirements engineering or programming)
- there are limits to what a development team can achieve (cf. Continuing Growth)

Simplicity over Performance

Wes Dyer

"Make it correct, make it clear, make it concise, make it fast. In that order."

Joshua J. Bloch (born 1961)

"The cleaner and nicer the program, the faster it's going to run. And if it doesn't, it'll be easy to make it fast."

Programming Wisdom on Software Evolution

Lessons Learned

- Sources for complexity: workarounds, unnecessary code, performance tweaks
- Lehman's laws: software is modified and gets more complex
- Further Reading: Ludewig and Lichter, 23.1 (Software-Evolution)

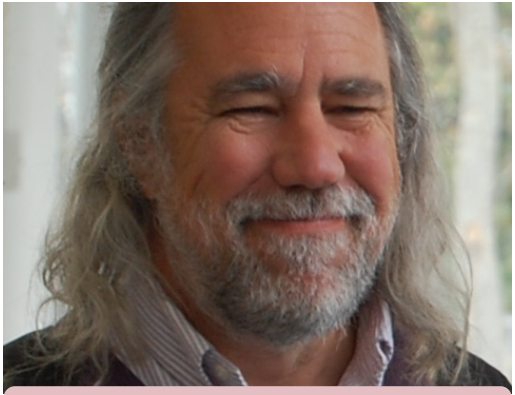
Practice

- Choose a successful piece of software and at least one of Lehman's laws
- Discuss whether or not the law applies to that example in Moodle: `https://moodle.uni-ulm.de/mod/moodleoverflow/discussion.php?d=2736`

Lecture Contents

Smells and Refactorings

Simple and Clean



Grady Booch (born 1955)

"The function of good software is to make the complex appear to be simple."

Robert C. Martin (Uncle Bob, born 1952)

Boy Scouts Rule: "Leave the campground cleaner than the way you found it."

Code Refactoring [Sommerville]

Motivation

- anticipating changes is typically infeasible
- anticipated changes may not materialize and unanticipated changes are required
- make changes easier by constantly refactoring the code

Refactoring

"Refactoring means that the programming team looks for possible improvements to the software and implements them immediately. When team members see code that can be improved, they make these improvements even in situations where there is no immediate need for them."

Structure of the Software

"A fundamental problem of incremental development is that local changes tend to **degrade the software structure**. Consequently, further changes to the software become harder and harder to implement. Essentially, the development proceeds by finding workarounds to problems, with the result that code is often duplicated, parts of the software are reused in inappropriate ways, and the overall structure degrades as code is added to the system. **Refactoring improves the software structure and readability and so avoids the structural deterioration that naturally occurs when software is changed.**"

Refactoring in Practice

Theory vs Practice [Sommerville]

"In principle, when refactoring is part of the development process, the software should always be easy to understand and change as new requirements are proposed. In practice, this is not always the case. Sometimes **development pressure means that refactoring is delayed** because the time is devoted to the implementation of new functionality. Some new features and changes cannot readily be accommodated by code-level refactoring and **require that the architecture of the system be modified**."

Grandma Beck's Child-Rearing Philosophy

"If it stinks, change it."

Smells [Fowler's Refactoring]

Mysterious Name

"Puzzling over some text to understand what's going on is a great thing if you're reading a detective novel, but not when you're reading code. [...] One of the most important parts of clear code is good names, so we put a lot of thought into naming functions, modules, variables, classes, so they clearly communicate what they do and how to use them."
— Rename Method/Field/Variable Refactoring

Duplicated Code

"If you see the same code structure in more than one place, you can be sure that your program will be better if you find a way to unify them. Duplication means that every time you read these copies, you need to read them carefully to see if there's any difference. If you need to change the duplicated code, you have to find and catch each duplication."
— Extract/Pull-Up Method Refactoring

Smells [Fowler's Refactoring]

Long Method

"Since the early days of programming, people have realized that the longer a function is, the more difficult it is to understand. Older languages carried an overhead in subroutine calls, which deterred people from small functions. [...] [T]he real key to making it easy to understand small functions is good naming. If you have a good name for a function, you mostly don't need to look at its body. [...] A heuristic we follow is that whenever we feel the need to comment something, we write a function instead."
— Extract Method Refactoring

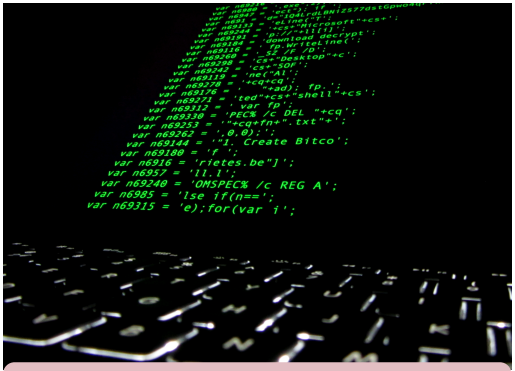
Robert C. Martin (Uncle Bob, born 1952)

"Functions should do one thing. They should do it well. They should do it only."

Writing Requires Reading

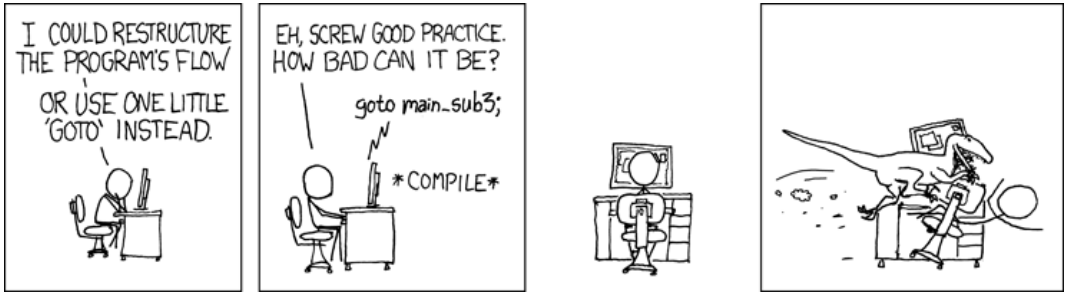
Robert C. Martin (Uncle Bob, born 1952)

"So if you want to go fast, if you want to get done quickly, if you want your code to be easy to write, make it easy to read."

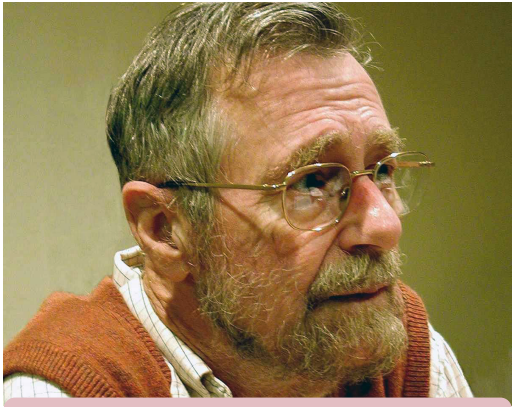


Eagleson's Law

"Any code of your own that you haven't looked at for six or more months might as well have been written by someone else."


I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
COMPILE

Spaghetti or Lasagne?



Edsger W. Dijkstra (1968)

"Go-To Statement Considered Harmful"
(today known as spaghetti code)

Roberto Waltman

"In the one and only true way. The object-oriented version of 'Spaghetti code' is, of course, 'Lasagna code'. (Too many layers)."

Smells and Refactorings

Lessons Learned

- Smells (anti patterns): structures to avoid
- Examples: mysterious name, duplicated code, long method
- Refactorings: clean-up of structures before and after every change
- Examples: rename X refactoring, extract/pull-up method refactoring
- Further Reading: Sommerville, 3.2.2 Refactoring
- Further Reading: Fowler's Refactoring
- Further Reading: Uncle Bob's Clean Code

Practice

- Research another smell and report it (incl. source) to Moodle: `https://moodle.uni-ulm.de/mod/moodleoverflow/discussion.php?d=2737`
- Find a suitable refactoring for an anti-pattern of your colleagues (answer to their post)

Lecture Contents