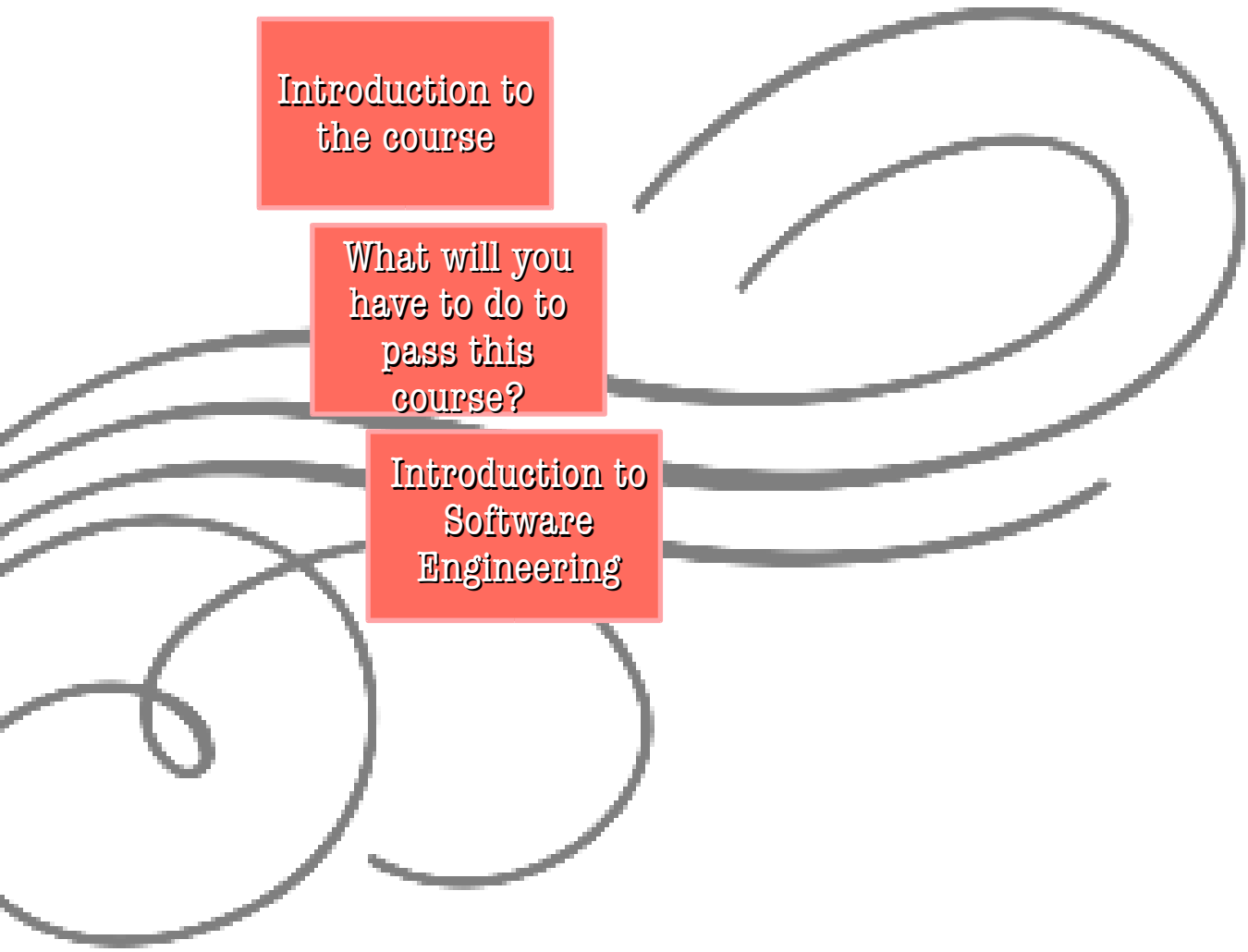
AGILE Software Construction

Beatrice Åkerblom
beatrice@dsv.su.se

Outline for the lecture



Introduction to the course

The goal of today is to present the course as a whole and to present an overwiew of classical Software Engineering. We start out by taking a look at the goals and contents of the course.

Course goals

Having successfully completed this course, a student knows:

- how to find suitable software engineering methods for his or her work in future projects
- several techniques for, and how to reason about techniques for
  - requirements elicitation and requirement management
  - resource planning
  - risk analysis and risk management
  - test planning
  - version management
  - time and cost estimations
  - project automation

Course goals, contd

- how to reason about the applicability of a certain technique in some known project
- the principles and theories behind Extreme Programming (XP)

Course syllabus

After the course, the student will have:

- written an essay on a topic relevant for software developing projects
- given a lecture on the essay topic to other students on the course

Course syllabus

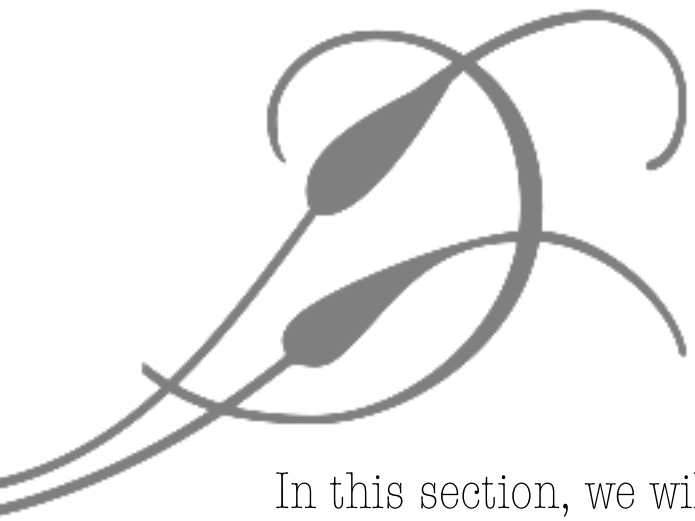
The student will also have participated in a software development project with close interaction with the customer using agile methods where the following has been practised:

- Agile development methodology with Extreme Programming
- customer-driven development
- test-driven development with unit testing
- pair programming
- refactoring
- requirement management
- resource management

Course syllabus, contd

- planning, documentation of the plans and continuous updating of them
- basic time and cost estimation
- version management of both documentation and code
- analysis of the project and its succes factors and failures to be used in future projects

What will I have to do to pass this course?



In this section, we will cover the different activities that are part of this course and take a look at what you will have to do to pass the course.

Activities during the course

- Reading exercises
- Essay writing
- Mini lecture
- Project work
- Take-home-exam

Reading Exercises

- The literature for the reading exercises will be avilable in the course library.
- The literature for the reading exercises consists of several different books on classical Software Engineering and possibly additional papers
- The readning exercises will be done before every lecture read either a section in one of the Software Engineering books or read the paper selected by the lecturer.
- Before the lecture you should send a short summary of what you have read to **beatrice@dsv.su.se**
- LaTeX template is available on the course's website.

Essay Writing

- All students taking the course will write an individual essay.
- The subject could be chosen from the list available on the course's website or proposed by the student. To propose your own essay subject you should first contact the course coordinator.
- By friday, **March 17:th (24:00)** , everyone should have sent an email with their choice or proposal to the course coordinator.
- On **March 24:th (24:00)** , the "Version 0.5" of all essays should be handed in by email to the **beatrice@dsv.su.se** for commenting.

Essay Writing (contd)

- In the seminars on **April 4:th and 6:th** all students will give lectures presenting their subject and essay results to the other students.
- The "Version 1.0" of the essays will be handed in in connection to the these "mini lectures".
- The "Version 1.0" of the essay will be commented on if necessary and will (when finished) be published on the course's website.

Essay Writing (contd)

- The essays must be written in an “accepted academic style”.
- The essay must written in “good” English (or Swedish).

Mini Lecture

- The mini lectures given by students will be approximately 20 minutes per student.
- The presentation should contain a introduction to your subject, a summary of your work and your results from the essay.
- The contents is of course important, but a good work deserves a good presentation.

Project Work

- Project work, individually:
  - Everyone will keep a project diary. This diary will later be used for project analysis. In addition to this, everyone is supposed to at least once a week during the project write 1/2-1 page describing what principles from “The Pragmatic Programmer” you have used, how and in what situations.

Project Work (contd)

- Project work, in project group:
  - The project will be done in groups of 5-6 students and the groups will be created by the course coordinator.
  - To make it possible to create better groups, all students must fill in the questionnaire that will be available on the course's web page on friday
  - In the project you will build a storage-, order- and logistics system which will be used by foodstores in the "Vo"-chain.

“Take-home-exam”

- A “take-home-exam” is an exam written by you on your own accompanied by the course literature, other literature and notes from the course.

- Writing a “take-home-exam” you have plenty of time to structure your answers and checking that your argumentation is clear and easy to follow - the demands on structure and language are considerably higher than when writing an ordinary exam.

- Since you have all course literature available, you are expected to use it when writing your answers and you are supposed to show that you use it by referring to it correctly.

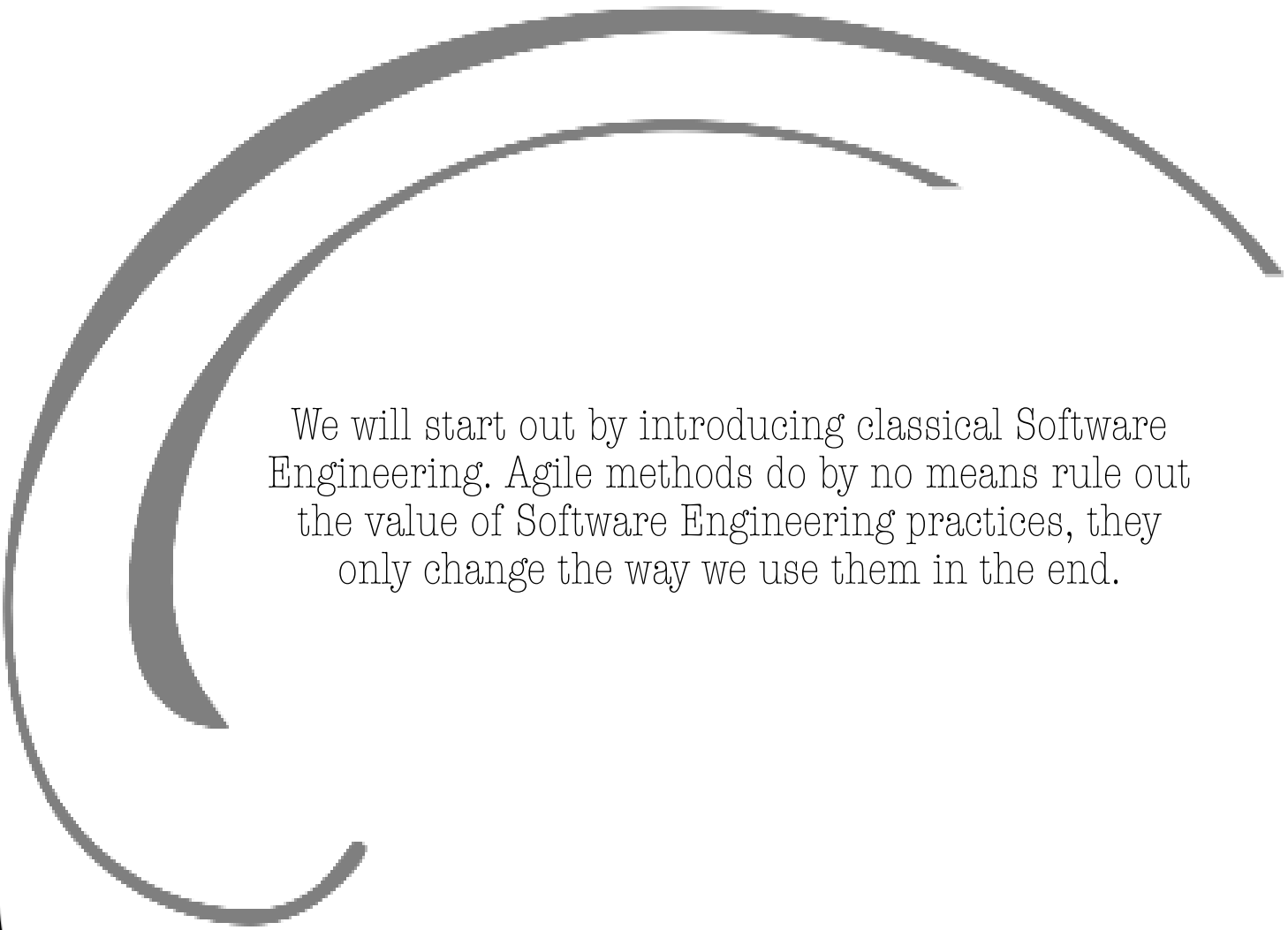
Literature

- XP Explained by Kent Beck
- The Pragmatic Programmer by Andrew Hunt and David Thomas

“Course board”

- The course board will meet with the course co-ordinator regularly during the course.
- The role of the course board is to bring forward to the course co-ordinator any opinions about the course, contents or how things are done, from their fellow students.
- If you as a srtudent want to let the course co-ordinator know his or her opinion about anything on the course and don't want to contact her on your own, you can contact one of the members of the course board.

Introduction to Software Engineering



We will start out by introducing classical Software Engineering. Agile methods do by no means rule out the value of Software Engineering practices, they only change the way we use them in the end.

AGILE Software Construction?

- Agile methods is an addition to the tool-box of software constructors.
- Agile methods give us new strategies for software construction
- Using agile methods, you still need to practice classical software engineering tasks like planning, requirements analysis, design, coding, testing, and documentation.
- Agile methods aim at futher decreasing the risks involved in software development.
- Using agile methods does **not** mean that the work will be undisciplined.

The story of the $0.00 bill

A well-known story tells of a man who once received a $0.00 bill. He laughed at it and threw it away. When he received the reminder bill, he reacted the same way. When he received the next bill claiming that legal actions would be taken if he did not immediately pay $0.00 he got worried and decided to mail a check for $0.00. This had the desired effect and a few days later he received the receipt for $0.00.

A few days later the man got a phone-call from his bank asking if he had paid someone with a $0.00 check. The man agreed and told the whole story. When he had finished, the bankwoman didn't laugh, but quietly asked “Have you any idea what your check for $0.00 did to **our** computer system?”.

The story of the Soviet missiles

On November 9, 1979, the US Military control system reports that the Soviet Union has launched missiles aimed towards the USA.

What had really happened was that a simulated attack had been interpreted as being a real one...

What is Software Engineering?

- Software development rather than software engineering?
- Craftsmanship, a better metaphor than engineering?
- Is the field mature enough to warrant the title "engineering"?
- What are the criteria for distinguishing someone who is a software engineer from someone who is not a software engineer?

Debating Software Engineering

- The activities that was formerly called programming and systems analysis
- A broad term for all aspects of the practice of computer programming, as opposed to the theory of computer programming, which is called computer science
- A specific approach to computer programming, claiming that it should be treated as an engineering profession rather than an art or a craft
- Software engineering is according to IEEE Standard 610.12:
  - the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, that is, the application of engineering to software
  - studying the approaches above

What is Software Engineering?

- Software engineering is an engineering discipline which is concerned with all aspects of software production

- Software engineers should adopt a systematic and organised approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available

Difference between Software Engineering and computer science?

- Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software
- Computer science theories are currently insufficient to act as a complete underpinning for software engineering

The nature of Software Engineering

Software Engineering resembles many different fields in many different ways:

- Mathematics
- Science
- Engineering
- Manufacturing
- Project management
- Art
- Performance

60 years of Software Engineering

- 1940s: First computer users wrote machine code by hand.
- 1950s: Early tools, such as macro assemblers and interpreters were created and widely used to improve productivity and quality. First generation optimizing compilers.
- 1960s: Second generation tools like optimizing compilers and inspections were being used to improve productivity and quality. The concept of software engineering was widely discussed. First really big (1000 programmer) projects. Commercial mainframes and custom software for big business.

60 years of Software Engineering (contd)

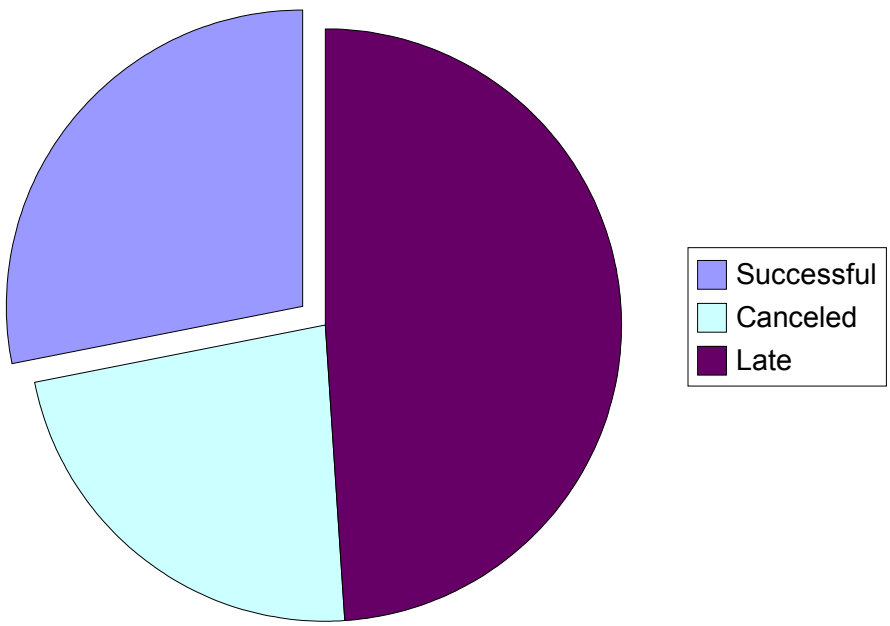
- 1970s: Collaborative software tools, such as Unix , code repositories, make, and so on. Minicomputers and the rise of small business software.
- 1980s: Personal computers and workstations and an emphasis on process like the CMM. The rise of consumer software.
- 1990s: Object-oriented programming and agile processes like eXtreme Programming gain mainstream acceptance. The WWW and hand-held computers make software even more widely available.

Scope of Software Engineering

- Historical Aspects
  - 1968 NATO Conference, Garmisch
  - Aim: to solve the “Software Crisis”
  - Software is delivered
    - Late
    - Over budget
    - With residual faults

Background

- Software Engineering emerged as an effort to solve the “software crisis”.
- In 2000, the Standish Group analysed software development projects
  - 28% successfully completed
  - 23% canceled before completion
  - 49 % over budget, late or with fewer functions than initially specified.



Scope of Software Engineering (contd)

- Why cannot bridge-building techniques be used to build operating systems?
  - Attitude to collapse
  - Imperfect engineering
  - Complexity
  - Maintenance

Conclusion

- Software Engineering is not “Engineering”

Economic Aspects

- Economically viable techniques
- Coding method CM-new is 10% faster than currently used method CM-old.
  - Should it be used?
- Common sense answer
  - “Of course!”
- Software Engineering answer
  - Consider the effect of CM-new on maintenance.

Life-cycle phases

- Requirements phase
- Specification phase
- Design phase
- Implementation phase
- Integration phase (in parallel with implementation)
- Maintenance phase
- Retirement

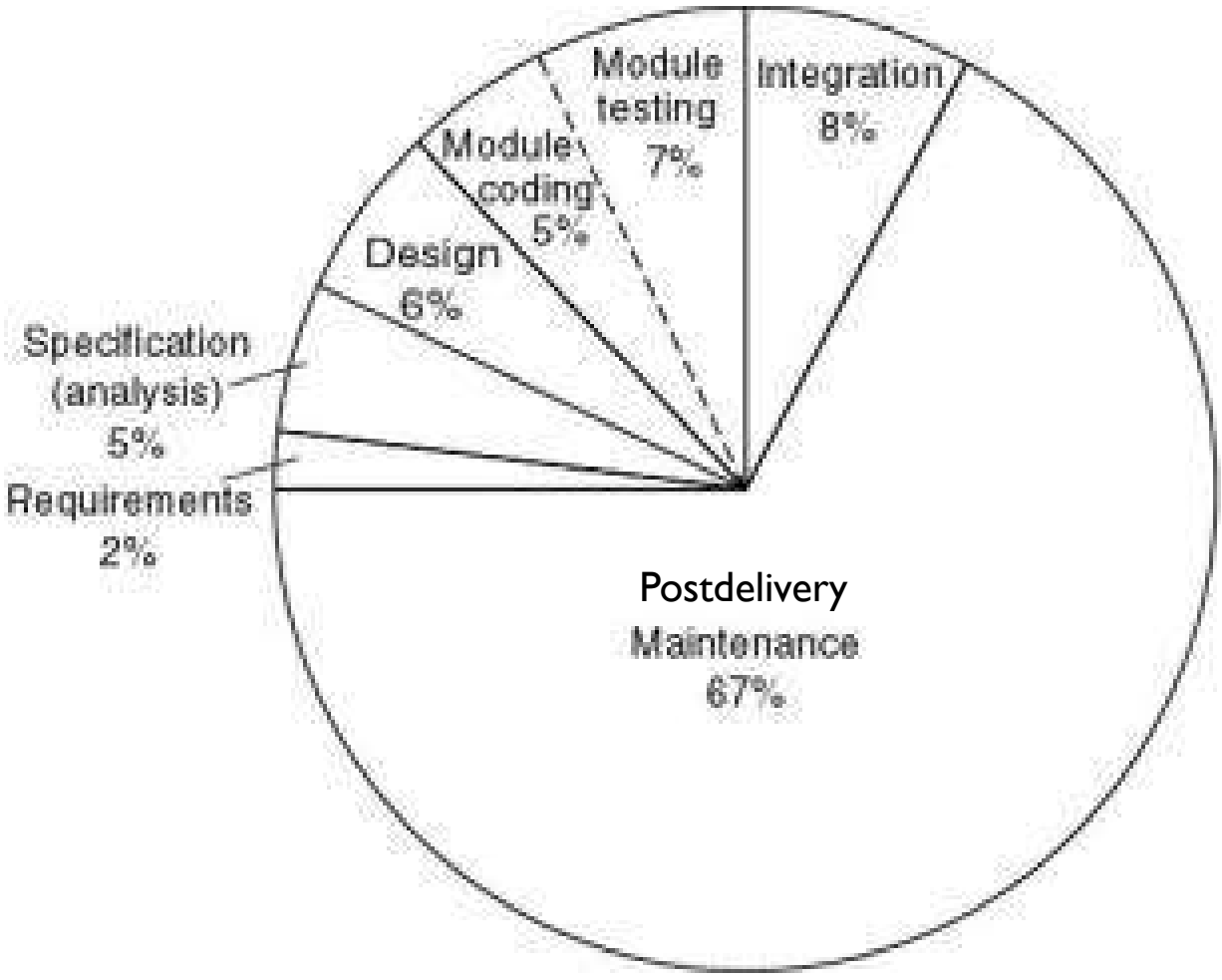
Software Life-Cycle

- The Software Life-Cycle is the practical development work
  - The way we produce software, including
    - The life-cycle model
    - The individuals
    - Tools

Classical vs. Modern views on Manintenance

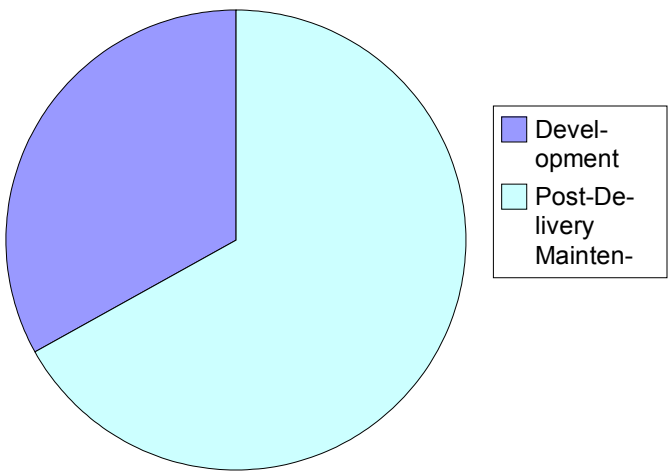
- Development-then-maintenance model
  - Temporal definition
- Maintenance is the process that occurs when “software undergoes modification to code and associated documentation due to a problem or the need for adaptation”. (ISO/IEC 12207, 1995)
  - Operational definition

Approximate Relative Cost of Each Phase

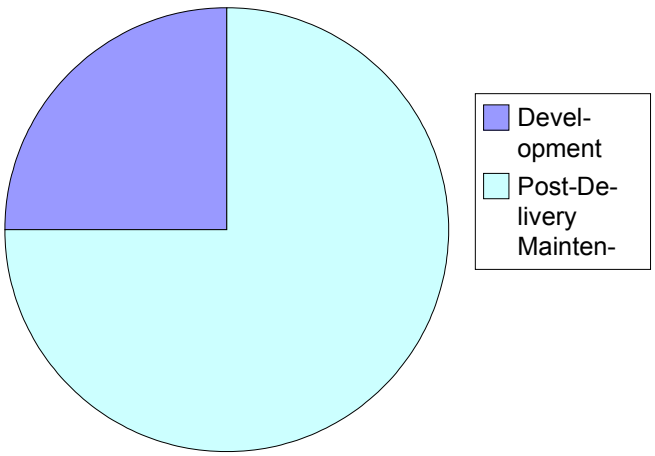


Relative Cost Development and Maintenance

1976 - 1981



1992 - 1998



Comparative Relative Cost of Each Phase

	Various Projects between 1976 and 1981	132 More Recent Hewlett-Packard Projects
Requirements and specification (analysis) phases	21%	18%
Design phase	18	19
Implementation phase	36	34
Integration phase	24	29

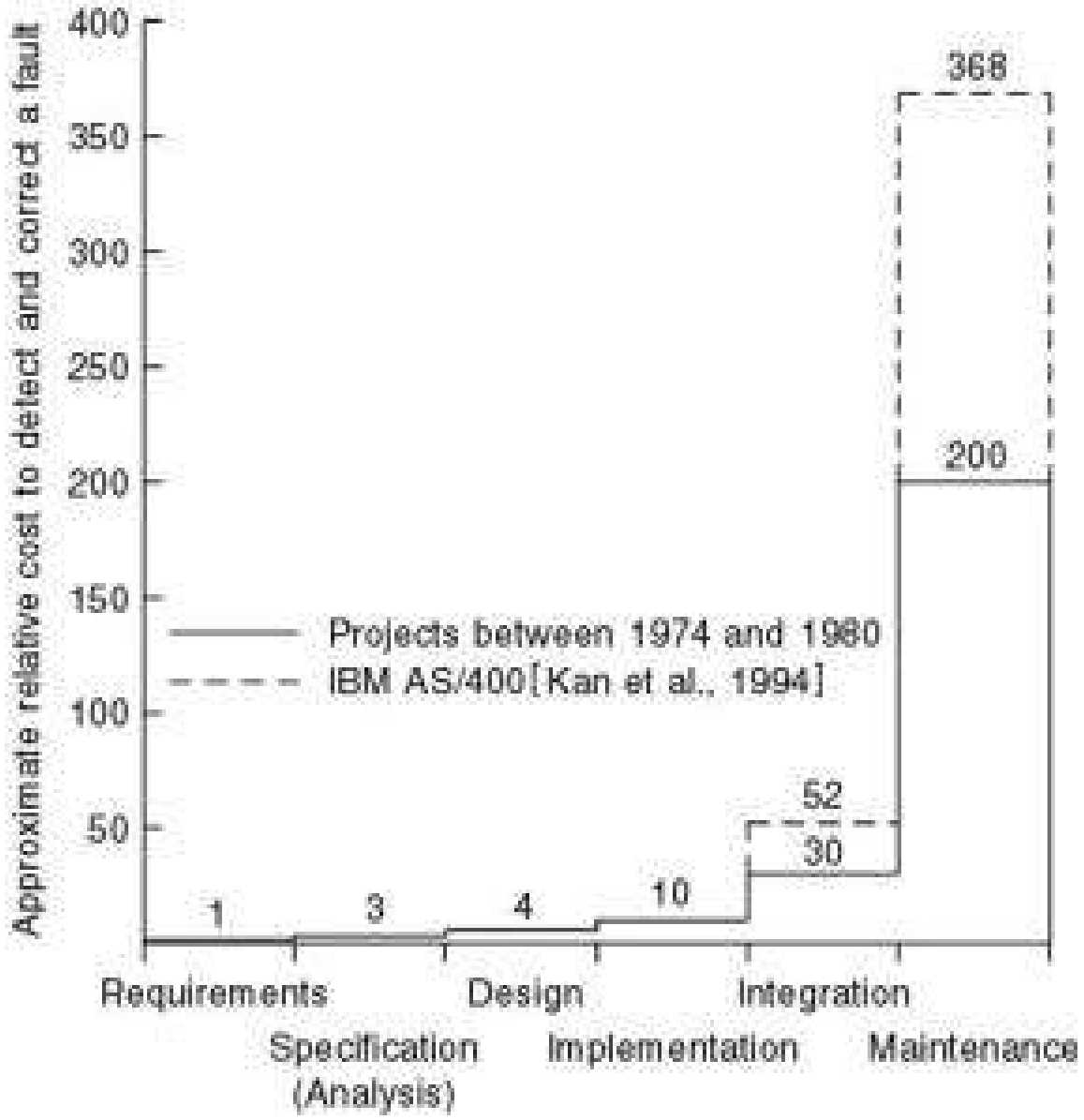
Good and Bad Software

- Good software is maintained—bad software is discarded
- Different types of maintenance
  - Corrective maintenance [about 20%]
  - Enhancement
    - Perfective maintenance [about 60%]
    - Adaptive maintenance [about 20%]
- Effect of CM-new on maintenance

Where do we make and find faults?

- 60 to 70 percent of faults are specification and design faults
- Data of Kelly, Sherif, and Hops [1992]
  - 1.9 faults per page of specification
  - 0.9 faults per page of design
  - 0.3 faults per page of code

Cost to Detect and Correct a Fault



Team Programming Aspects

- Hardware is cheap
  - We can build products that are too large to be written by one person in the available time
- Teams
  - Interface problems
  - Meetings

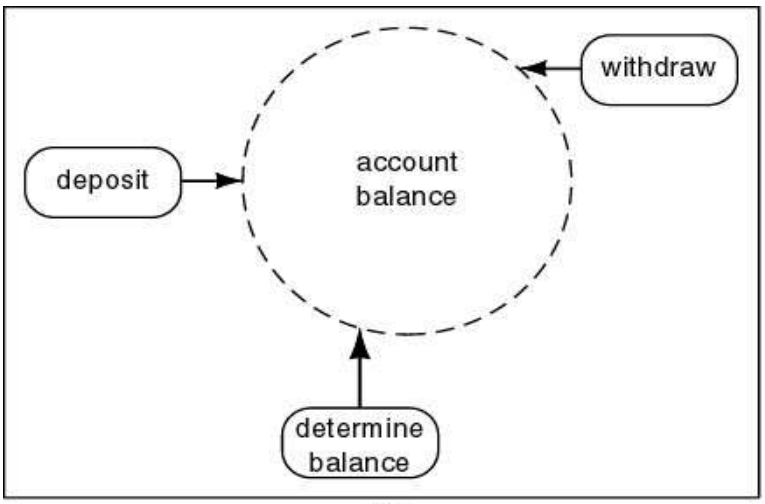
The Object-Oriented Paradigm

- The structured paradigm had great successes initially
  - It started to fail with larger products (> 50,000 LOC)
- Maintenance problems (today, up to 80% of effort)
- Reason: structured methods are
  - Action oriented (finite state machines, data flow diagrams); or
  - Data oriented (entity-relationship diagrams, Jackson's method);
  - But not both

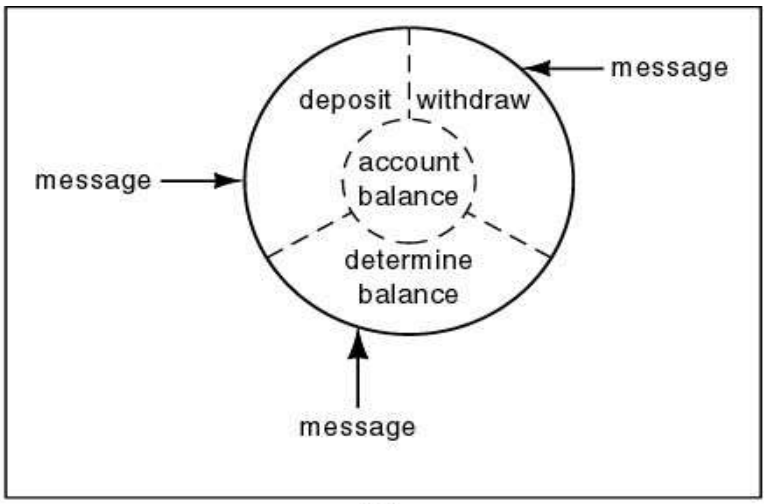
The Object-Oriented Paradigm (contd)

- Both data and actions are of equal importance
- Object:
  - Software component that incorporates both data and the actions that are performed on that data
- Example: Bank account
  - Data: account balance
  - Actions: deposit, withdraw, determine balancenance

Structured versus Object-Oriented Paradigm



(a)



(b)

- Information hiding
- Responsibility-driven design
- Impact on maintenance, development

Key Aspects of Object-Oriented Solution

- Conceptual independence
  - Encapsulation
- Physical independence
  - Information hiding
- Impact on development
  - Physical counterpart
- Impact on maintenance
  - Independence effects

Responsibility-Driven Design

- Also called “Design by Contract”
- Send flowers to your aunt in Iowa City
  - Call 1-800-FLOWERS
  - Where is 1-800-FLOWERS?
  - Which Iowa City florist does the delivery?
  - Information hiding
- Object-oriented paradigm
  - “Send a message to a method [action] of an object“

Transition From Analysis to Design

Structured Paradigm	Object-OrientedParadigm
1. Requirements phase	1. Requirements phase
2. Specification (analysis) phase	2′. Object-oriented analysis phase
3. Design phase	3′. Object-oriented design phase
4. Implementation phase	4′. Object-oriented programming phase
5. Integration phase	5. Integration phase
6. Maintenance phase	6. Maintenance phase
7. Retirement	7. Retirement

- Structured paradigm:
  - Jolt between analysis (what) and design (how)
- Object-oriented paradigm:
  - Objects enter from very beginning

Analysis/Design "Hump"

- Systems analysis
  - Determine what has to be done
- Design
  - Determine how to do it
  - Architectural design–determine modules
  - Detailed design–design each module

Removing the "Hump"

- Object-oriented analysis
  - Determine what has to be done
  - Determine the objects
- Object-oriented design
  - Determine how to do it
  - Design the objects

End of Today's Lecture

Thanks for your attention!