Slide 1.1

Object-Oriented Software Engineering

WCB/McGraw-Hill, 2008

Stephen R. Schach
srs@vuse.vanderbilt.edu

CHAPTER 1

Slide 1.2

THE SCOPE OF OBJECT-ORIENTED SOFTWARE ENGINEERING

Outline

Slide 1.3

- Historical aspects
- Economic aspects
- Maintenance aspects
- Requirements, analysis, and design aspects
- Team development aspects
- Why there is no planning phase

Outline (contd)

Slide 1.4

- Why there is no testing phase
- Why there is no documentation phase
- The object-oriented paradigm
- Terminology
- Ethical issues

1.1 Historical Aspects

Slide 1.5

- 1968 NATO Conference, Garmisch, Germany
- Aim: To solve the *software crisis*
- Software is delivered
  - Late
  - Over budget
  - With residual faults

Standish Group Data

Slide 1.6

- Data on 9236 projects completed in 2004

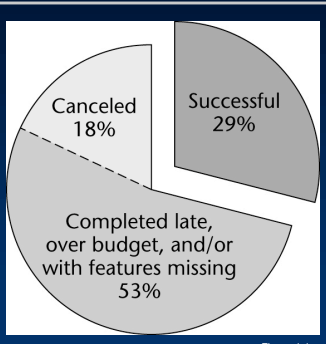

Figure 1.1

Cutter Consortium Data

Slide 1.7

- 2002 survey of information technology organizations
  - 78% have been involved in disputes ending in litigation
- For the organizations that entered into litigation:
  - In 67% of the disputes, the functionality of the information system as delivered did not meet up to the claims of the developers
  - In 56% of the disputes, the promised delivery date slipped several times
  - In 45% of the disputes, the defects were so severe that the information system was unusable

Conclusion

Slide 1.8

- The software crisis has not been solved
- Perhaps it should be called the *software depression*
  - Long duration
  - Poor prognosis

1.2 Economic Aspects

Slide 1.9

- 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 cost of training
  - Consider the impact of introducing a new technology
  - Consider the effect of CM_{new} on maintenance

1.3 Maintenance Aspects

Slide 1.10

- Life-cycle model
  - The steps (*phases*) to follow when building software
  - A theoretical description of what should be done
- Life cycle
  - The actual steps performed on a specific product

Waterfall Life-Cycle Model

Slide 1.11

- Classical model (1970)

1. Requirements phase
2. Analysis (specification) phase
3. Design phase
4. Implementation phase
5. Postdelivery maintenance
6. Retirement

Figure 1.2

Typical Classical Phases

Slide 1.12

- Requirements phase
  - Explore the concept
  - Elicit the client's requirements
- Analysis (specification) phase
  - Analyze the client's requirements
  - Draw up the specification document
  - Draw up the software project management plan
  - "What the product is supposed to do"

Typical Classical Phases (contd)

Slide 1.13

- Design phase
  - Architectural design, followed by
  - Detailed design
  - "How the product does it"
- Implementation phase
  - Coding
  - Unit testing
  - Integration
  - Acceptance testing

Typical Classical Phases (contd)

Slide 1.14

- Postdelivery maintenance
  - Corrective maintenance
  - Perfective maintenance
  - Adaptive maintenance
- Retirement

1.3.1 The Modern View of Maintenance

Slide 1.15

- Classical maintenance
  - Development-then-maintenance model
- This is a temporal definition
  - Classification as development or maintenance depends on when an activity is performed

Classical Maintenance Defn — Consequence 1

Slide 1.16

- A fault is detected and corrected one day after the software product was installed
  - Classical maintenance
- The identical fault is detected and corrected one day before installation
  - Classical development

Classical Maintenance Defn — Consequence 2

Slide 1.17

- A software product has been installed
- The client wants its functionality to be increased
  - Classical (perfective) maintenance
- The client wants the identical change to be made just before installation ("moving target problem")
  - Classical development

Classical Maintenance Definition

Slide 1.18

- The reason for these and similar unexpected consequences
  - Classically, maintenance is defined in terms of the time at which the activity is performed
- Another problem:
  - Development (building software from scratch) is rare today
  - Reuse is widespread

Modern Maintenance Definition

Slide 1.19

- In 1995, the International Standards Organization and International Electrotechnical Commission defined maintenance *operationally*
- Maintenance is nowadays defined as
  - The process that occurs when a software artifact is modified because of a problem or because of a need for improvement or adaptation

Modern Maintenance Definition (contd)

Slide 1.20

- In terms of the ISO/IEC definition
  - Maintenance occurs whenever software is modified
  - Regardless of whether this takes place before or after installation of the software product
- The ISO/IEC definition has also been adopted by IEEE and EIA

Maintenance Terminology in This Book

Slide 1.21

- *Postdelivery maintenance*
  - Changes after delivery and installation [IEEE 1990]
- *Modern maintenance* (or just *maintenance*)
  - Corrective, perfective, or adaptive maintenance performed at any time [ISO/IEC 1995, IEEE/EIA 1998]

1.3.2 The Importance of Postdelivery Maintenance

Slide 1.22

- Bad software is discarded
- Good software is maintained, for 10, 20 years or more
- Software is a model of reality, which is constantly changing

Time (= Cost) of Postdelivery Maintenance

Slide 1.23

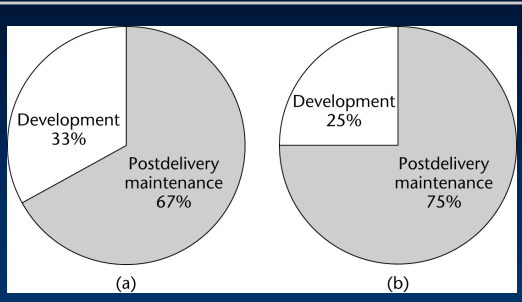

Figure 1.3

(a) Between 1976 and 1981
(b) Between 1992 and 1998

1.4 Requirements, Analysis, and Design Aspects

Slide 1.24

- The earlier we detect and correct a fault, the less it costs us

Requirements, Analysis, and Design Aspects (contd)

Slide 1.26

- The cost of detecting and correcting a fault at each phase

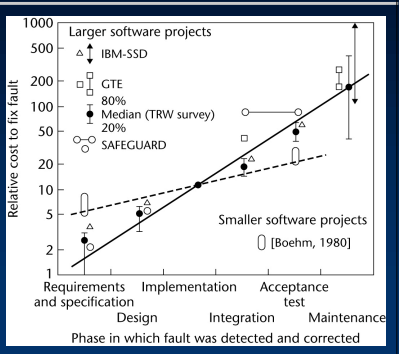

Figure 1.4

Requirements, Analysis, and Design Aspects (contd)

Slide 1.26

- The previous figure redrawn on a linear scale

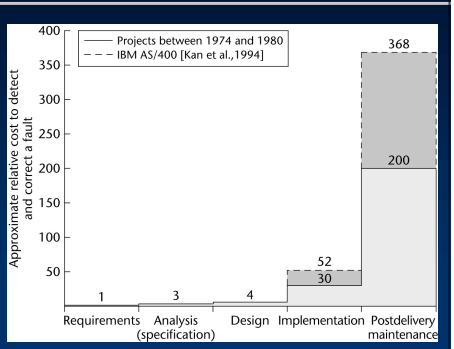

Figure 1.5

Requirements, Analysis, and Design Aspects (contd)

Slide 1.27

- To correct a fault early in the life cycle
  - Usually just a document needs to be changed
- To correct a fault late in the life cycle
  - Change the code and the documentation
  - Test the change itself
  - Perform regression testing
  - Reinstall the product on the client's computer(s)

Requirements, Analysis, and Design Aspects (contd)

Slide 1.28

- Between 60 and 70% of all faults in large-scale products are requirements, analysis, and design faults
- Example: Jet Propulsion Laboratory inspections
  - 1.9 faults per page of specifications
  - 0.9 per page of design
  - 0.3 per page of code

Conclusion

Slide 1.29

- It is vital to improve our requirements, analysis, and design techniques
  - To find faults as early as possible
  - To reduce the overall number of faults (and, hence, the overall cost)

1.5 Team Programming Aspects

Slide 1.30

- Hardware is cheap
  - We now can build products that are too large to be written by one person in the available time
- So Software is built by teams
  - Interfacing problems between modules
  - Communication problems among team members

1.6 Why There Is No Planning Phase

- We cannot plan at the beginning of the project —we do not yet know exactly what is to be built

Planning Activities of the Waterfall Model

- Preliminary planning of the requirements and analysis phases at the start of the project
- The software project management plan is drawn up when the specifications have been signed off by the client
- Management needs to monitor the SPMP throughout the rest of the project

Conclusion

- Planning activities are carried out throughout the life cycle
- There is no separate planning phase

1.7 Why There Is No Testing Phase

- It is far too late to test after development and before delivery

Testing Activities of the Waterfall Model

- Verification
  - Testing at the end of each phase (too late)
- Validation
  - Testing at the end of the project (far too late)

Conclusion

- Continual testing activities must be carried out throughout the life cycle
- This testing is the responsibility of
  - Every software professional, and
  - The software quality assurance group
- There is no separate testing phase

1.8 Why There Is No Documentation Phase

- It is far too late to document after development and before delivery

Documentation Must Always be Current

- Key individuals may leave before the documentation is complete
- We cannot perform a phase without having the documentation of the previous phase
- We cannot test without documentation
- We cannot maintain without documentation

Conclusion

- Documentation activities must be performed in parallel with all other development and maintenance activities
- There is no separate documentation phase

1.9 The Object-Oriented Paradigm

- The structured paradigm was successful initially
  - It started to fail with larger products (> 50,000 LOC)
- Postdelivery maintenance problems (today, 70 to 80% of total effort)
- Reason: Classical methods are
  - Action oriented; or
  - Data oriented;
  - But not both

The Object-Oriented Paradigm (contd)

- Both data and actions are of equal importance
- Object:
  - A 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 balance

Strengths of the Object-Oriented Paradigm

- 1. With information hiding, postdelivery maintenance is safer
  - The chances of a regression fault are reduced
- 2. Development is easier
  - Objects generally have physical counterparts
  - This simplifies modeling (a key aspect of the object-oriented paradigm)

Strengths of the Object-Oriented Paradigm (contd)

- 3. Well-designed objects are independent units
  - Everything that relates to the real-world item being modeled is in the corresponding object — *encapsulation*
  - Communication is by sending *messages*
  - This independence is enhanced by *responsibility-driven design (the way the operation is done is the responsibility of the object)*

Weaknesses of the Object-Oriented Paradigm

- 1. The object-oriented paradigm has to be used correctly
  - All paradigms are easy to misuse
- 2. When used correctly, the object-oriented paradigm can solve some (but not all) of the problems of the classical paradigm

Weaknesses of the Object-Oriented Paradigm (contd)

- 3. The object-oriented paradigm has problems of its own
- 4. The object-oriented paradigm is the best alternative available today
  - However, it is certain to be superceded by something better in the future

1.10 Terminology

- Client, developer, user
- Internal software
- Contract software
- Commercial off-the-shelf (COTS) software (shrink-wrapped sw or clickware)
- Open-source software
  - Linus's Law ("given enough eyeballs, all bugs are shallow")

Terminology (contd)

- Software
- Program (sw ind. Unit), system (sw and hw), product (non-trivial piece of sw)
- Methodology, paradigm
  - Object-oriented paradigm
  - Classical (traditional) paradigm
- Technique (specific to phase or task)

Terminology (contd)

- Mistake, fault, failure, error
- Defect
- Bug
  - "A bug crept into the code"
    instead of
  - "I made a mistake"

Object-Oriented Terminology

Slide 1.49

- Data component of an object
  - State variable
  - Instance variable (Java)
  - Field (C++)
  - Attribute (generic)
- Action component of an object
  - Member function (C++)
  - Method (generic)

Object-Oriented Terminology (contd)

Slide 1.50

- C++: A member is either an
  - Attribute ("field"), or a
  - Method ("member function")
- Java: A field is either an
  - Attribute ("instance variable"), or a
  - Method

Definition of Object-Oriented Software Engineering

Slide 1.51

- Software engineering
  - A discipline whose aims are
    - » The production of fault-free software,
    - » Delivered on time and within budget,
    - » That satisfies the client's needs
    - » Furthermore, the software must be easy to modify when the client's needs change
- Object-oriented software engineering
  - A discipline that utilizes the object-oriented paradigm to achieve the aims of software engineering

1.11 Ethical Issues

Slide 1.52

- Developers and maintainers need to be
  - Hard working
  - Intelligent
  - Sensible
  - Up to date and, above all,
  - Ethical
- IEEE-CS ACM Software Engineering Code of Ethics and Professional Practice
  www.acm.org/serving/se/code.htm