Software Engineering

Adapted from slides of Stephen R. Schach

THE SCOPE OF SOFTWARE ENGINEERING

1.1 Historical Aspects

- 1968 NATO Conference, Garmisch, Germany

- Aim: To solve the *software crisis*

- Software is delivered
  - Late
  - Over budget
  - With residual faults

Standish Group Data

- Data on projects completed in 2006

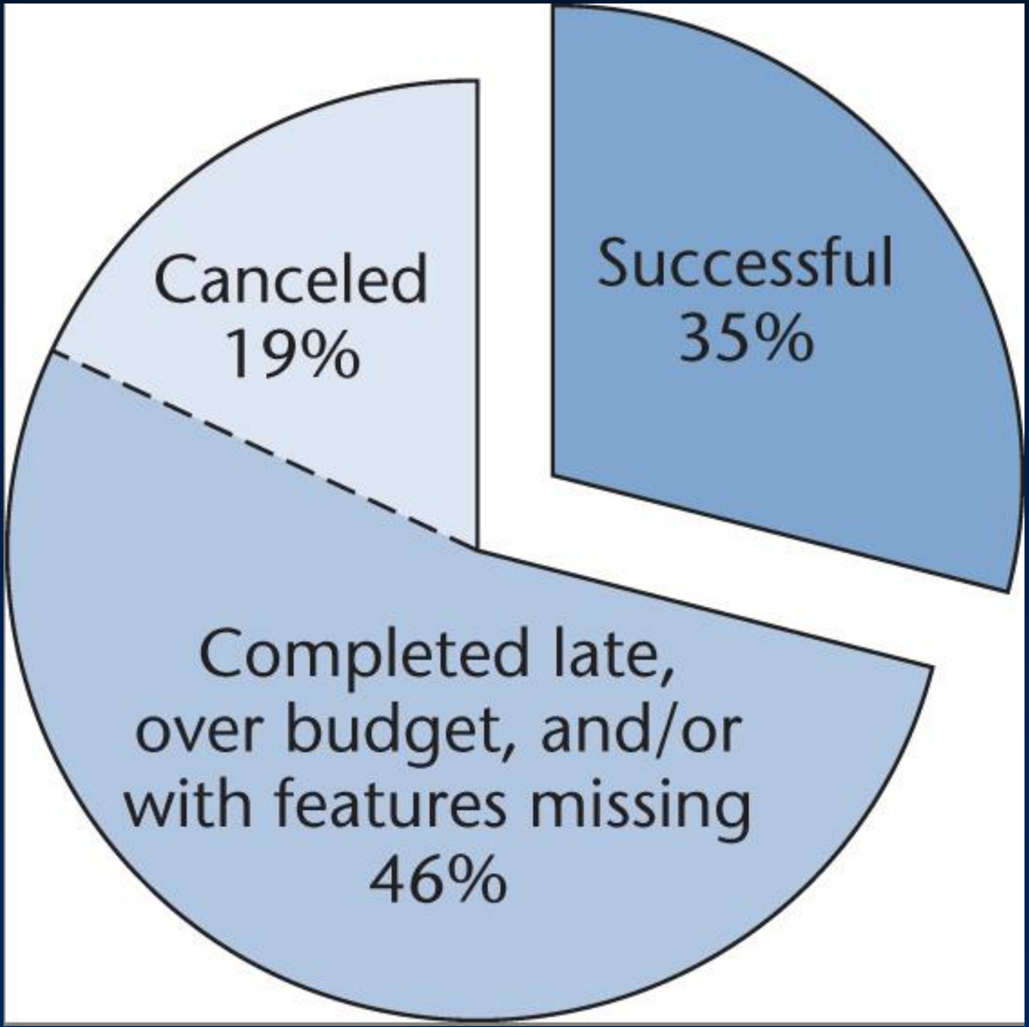

Figure 1.1

- Just over one in three projects was successful

Cutter Consortium Data

- 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

- The software crisis has not been solved

- Perhaps it should be called the *software depression*
  - Long duration
  - Poor prognosis

1.2 Economic Aspects

- 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

- 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

- 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

- 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)

- 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)

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

- Retirement

1.3.1 Classical and Modern Views of Maintenance

- 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

- 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

- 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

- 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

- 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)

- 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

- *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

- 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

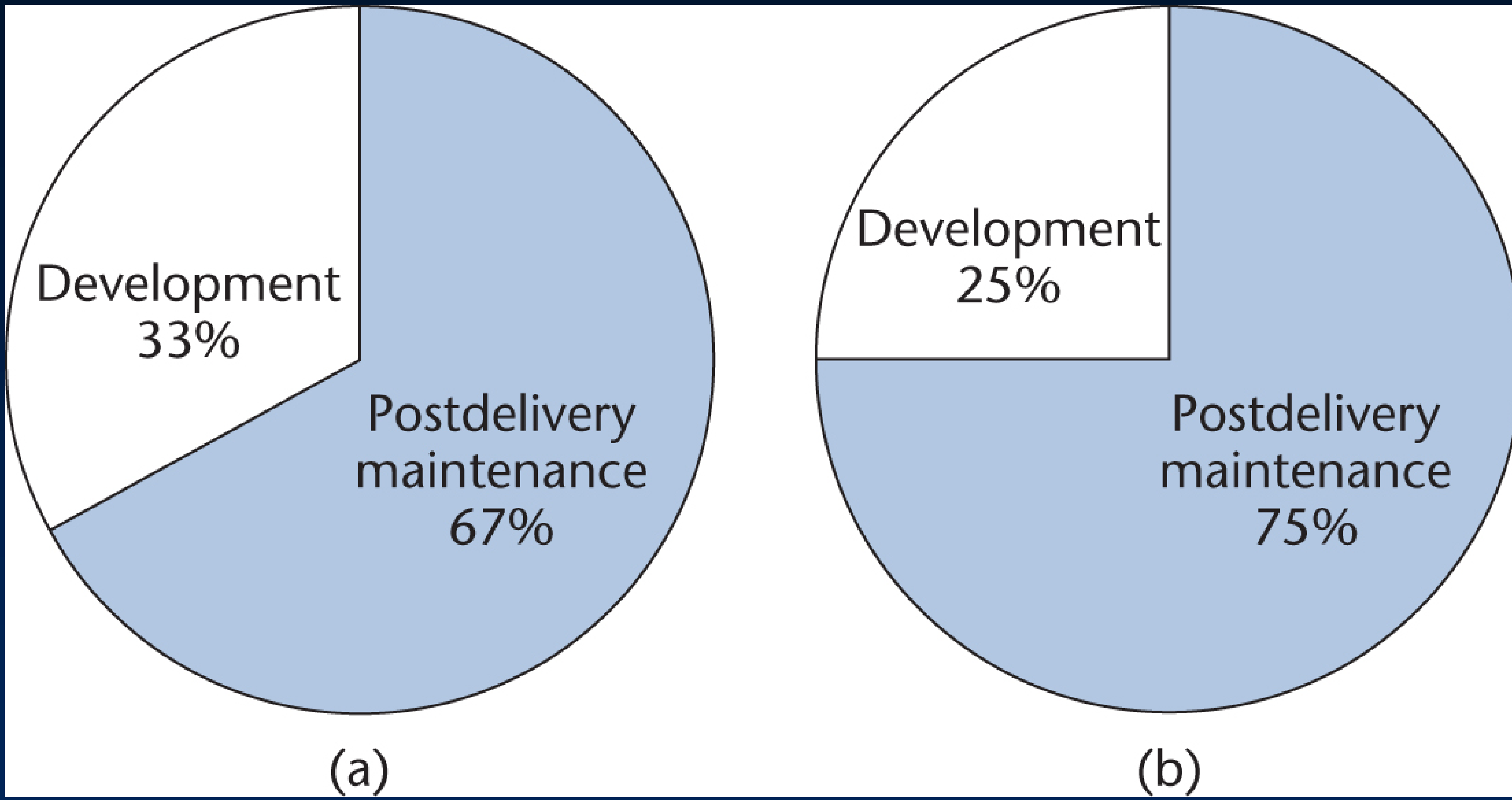

Figure 1.3

(a) Between 1976 and 1981

(b) Between 1992 and 1998

The Costs of the Classical Phases

- Surprisingly, the costs of the classical phases have hardly changed

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

Figure 1.4

Consequence of Relative Costs of Phases

- Return to CM_{old} and CM_{new}
- Reducing the coding cost by 10% yields at most a 0.85% reduction in total costs
  - Consider the expenses and disruption incurred
- Reducing postdelivery maintenance cost by 10% yields a 7.5% reduction in overall costs

1.4 Requirements, Analysis, and Design Aspects

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

Requirements, Analysis, and Design Aspects (contd)

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

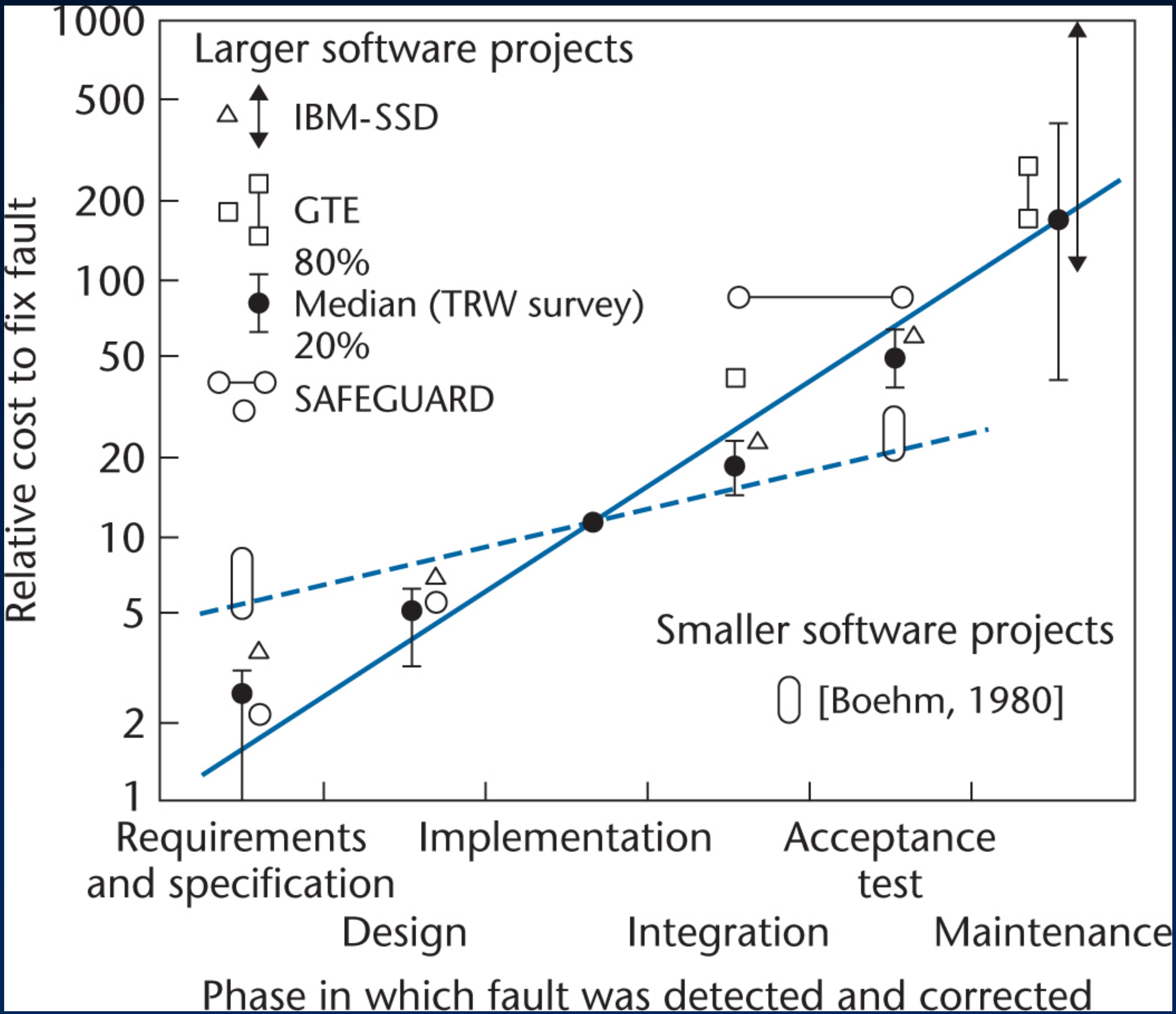

Figure 1.5

Requirements, Analysis, and Design Aspects (contd)

- The previous figure redrawn on a linear scale

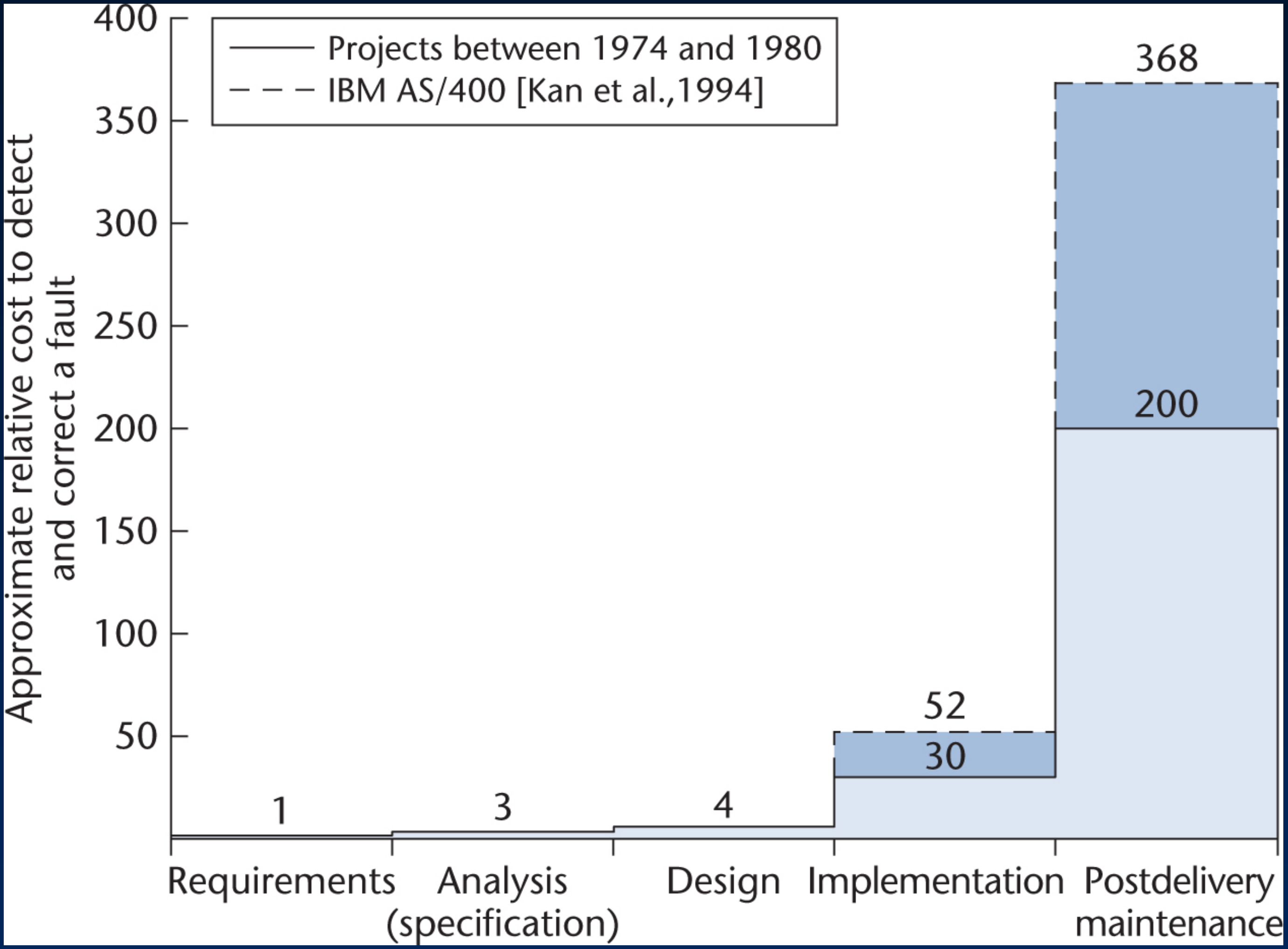

Figure 1.6

Requirements, Analysis, and Design Aspects (contd)

- 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)

- 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

- 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

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

- 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 Classical Paradigm

- 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 Classical Paradigm

- 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

1.11 Terminology

- Client, developer, user
- Internal software
- Contract software
- Commercial off-the-shelf (COTS) software
- Open-source software
  - Linus's Law

Terminology (contd)

- Software
- Program, system, product
- Methodology, paradigm
  - Object-oriented paradigm
  - Classical (traditional) paradigm
- Technique

Terminology (contd)

- Mistake, fault, failure, error

- Defect

- Bug
  - “A bug crept into the code”

    instead of
  - “I made a mistake”

1.12 Ethical Issues

- 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