Lecture 20: Software Maintenance

Software Evolution
- Software types
- Laws of evolution

Maintaining software
- types of maintenance
- challenges of maintenance

Reengineering and reverse engineering

Software Reuse

Program Types

Source: *Adapted from Lehman 1980, pp1061-1063*

S-type Programs ("Specifiable")

- problem can be stated formally and completely
- acceptance: Is the program correct according to its specification?
- This software does not evolve.
  - A change to the specification defines a new problem, hence a new program

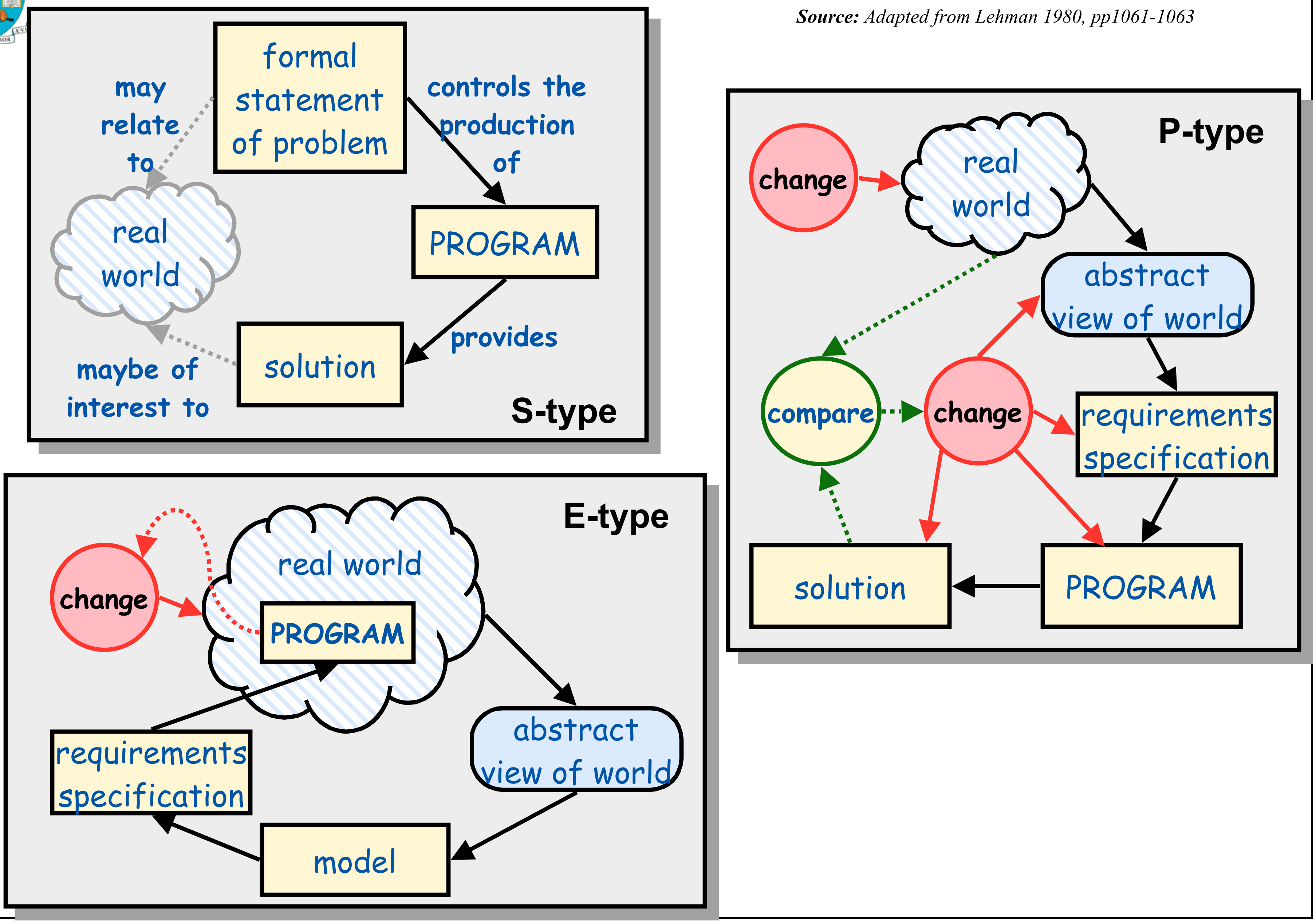
P-type Programs ("Problem-solving")

- imprecise statement of a real-world problem
- acceptance: Is the program an acceptable solution to the problem?
- This software is likely to evolve continuously
  - because the solution is never perfect, and can be improved
  - because the real-world changes and hence the problem changes

E-type Programs ("Embedded")

- A system that becomes part of the world that it models
- acceptance: depends entirely on opinion and judgement
- This software is inherently evolutionary
  - changes in the software and the world affect each other

Source: Adapted from Lehman 1980, pp1061-1063



Laws of Program Evolution

Source: *Adapted from Lehman 1980, pp1061-1063. See also, van Vliet, 1999, Pp59-62*

Continuing Change

- Any software *that reflects some external reality* undergoes continual change or becomes progressively less useful
  - The change process continues until it is judged more cost effective to replace the system entirely

Increasing Complexity

- As software evolves, its *complexity* increases…
  - …unless steps are taken to control it.

Fundamental Law of Program Evolution

- Software evolution is self-regulating with statistically determinable trends and invariants

Conservation of Organizational Stability

- During the active life of a software system, the work output of a development project is roughly constant (regardless of resources!)

Conservation of Familiarity

- During the active life of a program the amount of change in successive releases is roughly constant

Types of Maintenance

Source: *Adapted from van Vliet, 1999, p449.*

Corrective Maintenance

- fixing latent errors
  - includes temporary patches and workarounds

Adaptive Maintenance

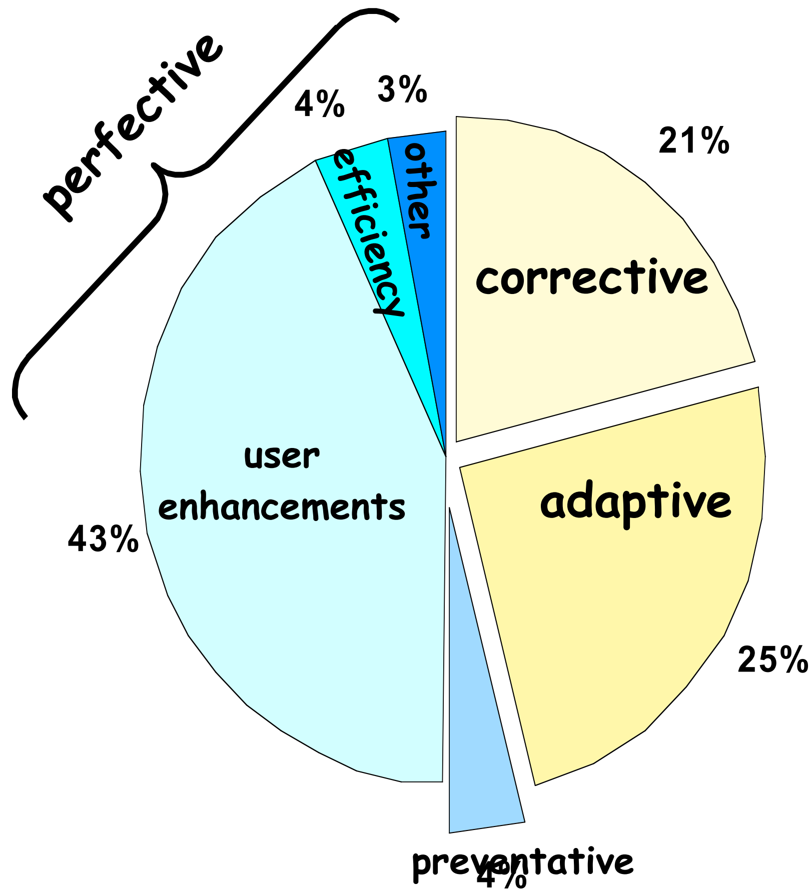
- responding to external changes
  - changes in hardware platform
  - changes in support software

Perfective Maintenance

- improving the as-delivered software
  - user enhancements
  - efficiency improvements

Preventative Maintenance

- Improves (future) maintainability
  - Documenting, commenting, etc.



Problems facing maintainers

Source: *Adapted Pfleeger 1998, p423-424. See also, van Vliet, 1999, pp464-467*

Top five problems:

- (Poor) quality of documentation
- user demand for enhancements and extensions
- competing demands for maintainers' time
- difficulty in meeting scheduled commitments
- turnover in user organizations

Limited Understanding

- 47% of software maintenance effort devoted to understanding the software
  - E.g. if a system has m components and we need to change k of them…
  - …there are k*(m-k) + k*(k-1)/2 interfaces to check for impact
- also, >50% of effort can be attributed to lack of user understanding
  - I.e. incomplete or mistaken reports of errors & enhancements

Low morale

- software maintenance is regarded as less interesting than development

Approaches to maintenance

Source: *van Vliet,1999, pp473-475*

Maintenance philosophies

- "throw-it-over-the-wall" - someone else is responsible for maintenance
  - investment in knowledge and experience is lost
  - maintenance becomes a reverse engineering challenge
- "mission orientation" - development team make a long term commitment to maintaining the software

Basili's maintenance process models:

- Quick-fix model
  - changes made at the code level, as easily as possible
  - rapidly degrades the structure of the software
- Iterative enhancement model
  - Changes made based on an analysis of the existing system
  - attempts to control complexity and maintain good design
- Full-reuse model
  - Starts with requirements for the new system, reusing as much as possible
  - Needs a mature reuse culture to be successful

Software Rejuvenation

Source: *van Vliet, 1999, Pp455-457*

Redocumentation

- Creation or revision of alternative representations of software
  - at the same level of abstraction
- Generates:
  - data interface tables, call graphs, component/variable cross references etc.

Restructuring

- transformation of the system's code without changing its behavior

Reverse Engineering

- analyzing a system to extract information about the behavior and/or structure
  - also Design Recovery - recreation of design abstractions from code, documentation, and domain knowledge
- Generates:
  - structure charts, entity relationship diagrams, DFDs, requirements models

Reengineering

- Examination and alteration of a system to reconstitute it in another form
- Also known as renovation, reclamation

Reuse

Source: *van Vliet, 1999, Chapter 17*

Software reuse aims to cut costs

- Developing software is expensive, so aim to reuse for related systems
  - Successful approaches focus on reusing knowledge and experience rather than just software products
  - Economics of reuse are complex as it costs more to develop *reusable* software

Libraries of Reusable Components

- domain specific libraries (e.g. Math libraries)
- program development libraries (e.g. Java AWT, C libraries)

Domain Engineering

- Divides software development into two parts:
  - domain analysis - identifies generic reusable components for a problem domain
  - application development - uses the domain components for specific applications.

Software Families

- Many companies offer a range of related software systems
  - Choose a stable architecture for the software family
  - identify variations for different members of the family
- Represents a strategic business decision about what software to develop

References

van Vliet, H. "Software Engineering: Principles and Practice (2nd Edition)" Wiley, 1999.

Chapter 14 is a very good introduction to the problems and approaches to software maintenance. Chapter 17 covers software reuse in far more detail than we'll go into on this course.

Lehman, M.M. "Programs, Life Cycles, and Laws of Software Evolution". Proceedings of the IEEE, vol 68, no 9, 1980.

Lehman was one of the first to recognise that software evolution is a fact of life. His experience with a number of large systems led him to formulate his laws of evolution. This paper is included in the course readings. It is widely cited.

Pfleeger, S. L. "Software Engineering: Theory and Practice" Prentice Hall, 1998.

Pfleeger's chapter 10 provides some additional data on the costs of maintenance.