CSC 408F/CSC2105F Lecture Notes

These lecture notes are provided for the personal use of students taking CSC 408H/CSC 2105H in the Fall term 2004/2005 at the University of Toronto.

Copying for purposes other than this use and all forms of distribution are expressly prohibited.

Reading Assignment

van Vliet Chapter 14

Software Maintenance

This is where the money goes.

- Maintain control over systems day-to-day functions.
- Maintain control over system modifications.
- Perfect existing functionality.
- Prevent system performance from degrading to unacceptable levels.
- Well over half of the organizations resources expended on a software system are spent on maintenance.
- Design for ease of maintenance should be a primary consideration in the design of any software system.

Recall Lehman's Laws of Software Evolution

1. *Continuing Change* A program undergoes continual change or it becomes progressively less useful. Change or decay continues until it becomes more cost effective to replace the system.
2. *Increasing Complexity* As a program evolves, its structure deteriorates. Complexity increases unless effort is expended to reduce it.
3. *Fundamental Law of Program Evolution* Software systems exhibit regular behavior and trends that we can measure and predict.
4. *Conservation of Organizational Stability* Productivity is essentially constant over the lifetime of the software.
5. *Conservation of Familiarity* As software evolves, each release has a diminishing effect (change) on functionality.

The size, complexity and disorder of a system all increase over time.

Types of Software Maintenance

- *Corrective maintenance* - repair of errors in the software reported by users
- *Adaptive maintenance* - modify software interface to changing environment
- *Perfective maintenance* - modify software to increase functionality, improve internal structure, or improve performance
- *Preventive maintenance* - modify or rewrite software to improve structure and reliability and to enhance future maintainability

Maintenance Management

- Change Control as described previously
- Keep documentation up to date with changes
- Archive systematically
- Do preventive maintenance on software that is in really bad shape
- Monitor maintenance effort to track maintenance costs

  Beware of steadily rising maintenance costs, software rot is setting in

Maintenance Activities

1. Understanding the system.
2. Locating information in system documentation.
3. Keeping system documentation up to date.
4. Extending existing system functions to accommodate new or changing requirements.
5. Adding new functionality to the system.
6. Finding the source of system failures or problems.
7. Locating and Correcting faults.
8. Answering questions about the way the system works.
9. Restructuring system design and system code.
10. Rewriting system design and system code.
11. Deleting design and and code components that are obsolete.
12. Managing changes to the system as they are made.

Corrective Maintenance Cycle

- User discovers a possible failure of the software.
- User calls support hot-line and describes failure.
- Front line support personnel create a trouble ticket describing the call.

  Identify the release of the software involved.

  Try to get as much information as possible about the cause of the failure. Has user customized the software?
- Trouble tickets are sent to second level support personnel for assessment.
- Trouble ticket assessment.
  - Determine if trouble ticket corresponds to a known problem.
  - Try to group trouble tickets that *appear* to be the same fault together.
  - Try to determine if user is using the software correctly.
  - Determine severity of the failure (preliminary prioritization)

- Maintenance Management monitors outstanding trouble tickets.
  Assign tickets to maintenance programmers in priority order.
- Maintenance programmer receives new trouble ticket.
  - Retrieve documentation for specified release (from release archive).
  - Try to identify software modules responsible for failure.
  - Retrieve relevant source code (using system model and release archive).
  - Create test environment for suspect modules.
  - Try to replicate failure on test system.
  - Iterate steps above until failure is reproduced
    or it appears that failure is user caused.
  - From failure try to isolate fault causing the failure.
  - Prepare change request describing the fault and the probable steps required to correct it.

- Change committee prioritizes change requests and assigns change requests to support programmers in priority order.
- Support programmers implement and test changes.
- Change is integrated into next release of the software.
  Quick patch may be distributed to affected users.
- Updates required
  - Software design documents.
  - User and internal documentation.
  - System model.
  - Archive all modified software in version archive.
  - Add test cases to test suite.
  - Status of trouble ticket and change request.

Information to Aid Maintenance

- Complete and exact description of each software release.
  System model and version control archive.
- Complete and correct description of the process used to build the software.
- Complete and correct internal and external documentation for each release.
- Tools and hooks to discover user customization and/or modification of the software.
- Test suite for release from release archive.
- Debugging and testing output built into the system.
- Ability to compare system models for different releases.

Software Maintenance Issues

- Understanding *current* state of software after many versions and releases.
  Changes inadequately documented.
- Hard to determine the *process* that was used to create the software
- Very hard to understand someone else's program.
  Especially if code is poorly documented.
- Originator of the software may not be available for consultation
- Documentation may be poor .. non-existent.
  Documentation may not be current
- Software was not designed to be maintainable.
  Source files may be difficult to locate.
- Maintenance is unglamorous, maintenance group can become a dumping ground for least effective programmers

Maintenance Problems - Staff Related

- Limited understanding of the software being maintained.

  Maintenance programmers don't understand the system

  Users don't understand system. Increases maintenance workload.

- Management priorities.

  May not get enough time/budget to do maintenance well.

  Sloppy maintenance causes problems in the future.

- Morale

  Maintenance staff have low social/economic status in the organization.

- **Possible cures.**
  - Good documentation, kept up to date.
  - Educate management on importance (cost/benefit) of doing maintenance right.
  - Reward maintenance staff generously. Rotate staff between maintenance and development.

Maintenance Problems - Technical

- Artifacts & Paradigms

  Original design logic convoluted and/or hard to understand.

  Designers failed to anticipate (correctly) future changes.

  e.g Y2K problems.

  Object Oriented programs harder to maintain due to intricate object interconnectivity.

  Very intricate programs (e.g. high coupling, low cohesion) are inherently more difficult to maintain (correctly).

- Testing Issues

  May need expensive duplication of production environment for testing.

  May be difficult to generate test data for new hardware.

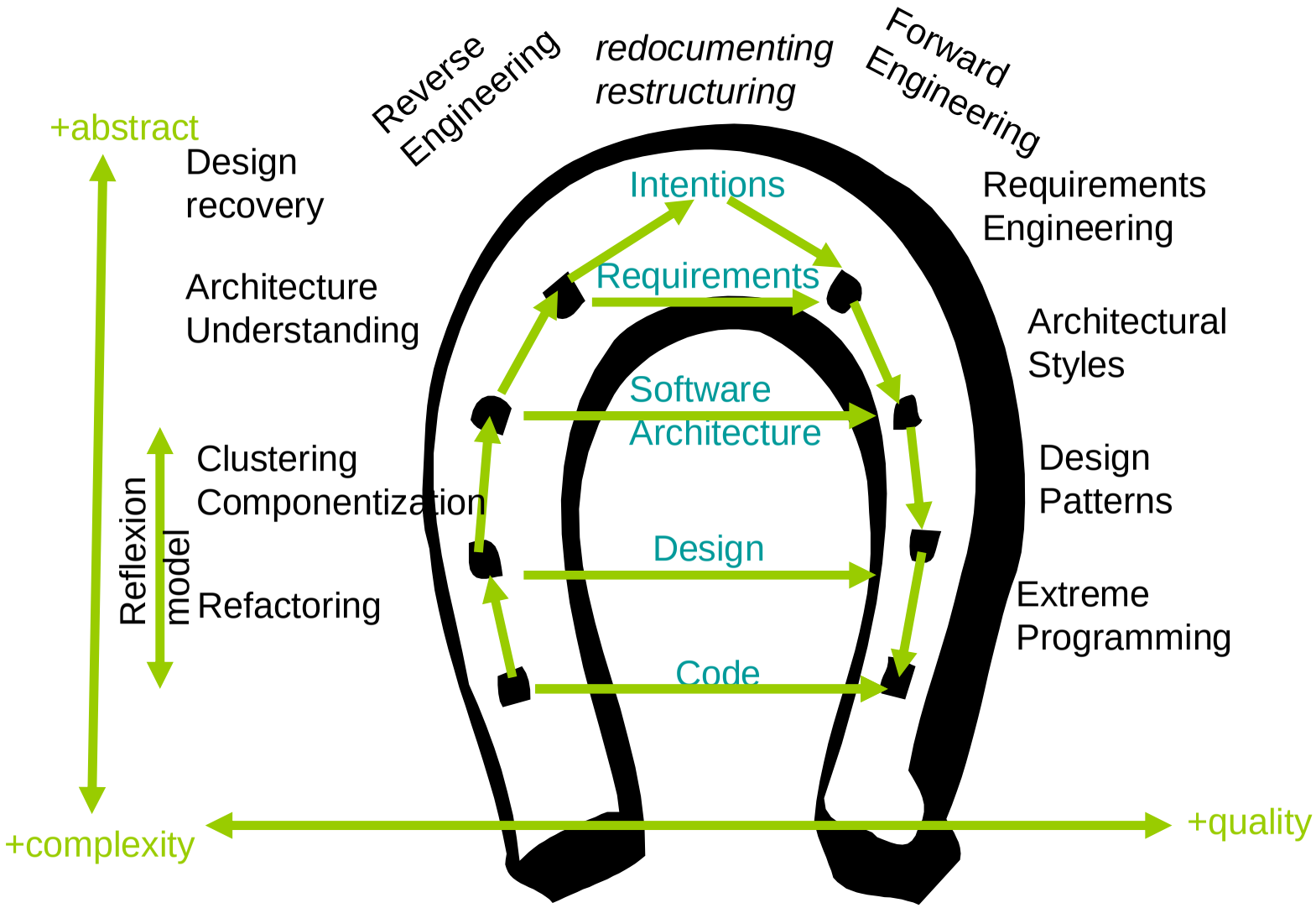
  May be difficult to determine how to devise tests for new change.

Factors Affecting Maintenance Effort

- Type of application.
- Novelty of the software.
- Turnover and maintenance staff availability.
- Expected system life span.
- Dependence on a changing environment.
- Hardware characteristics.
- Design quality.
- Code quality.
- Documentation quality.
- Testing quality.

Maintain vs. Replace

- Is the cost of maintenance too high?
- Is the system reliability unacceptable?
- Has the system lost the ability to adapt to further change at a reasonable cost and schedule?
- Is the system performance beyond prescribed constraints?
- Are system functions of limited usefulness?
- Can other systems do the same job better, faster, cheaper?
- Has hardware maintenance cost become excessive, so that hardware should be replaced?



The Horseshoe model of Software Reengineering and related research directions

Software Redocumentation

- Develop new documentation to assist maintenance efforts.
- Use static analysis tools to process the source code
  - Component calling relationships
  - Data dictionary information
  - Data flow information
  - Control flow information
  - Design recovery
  - Possible test paths
  - Cross reference information, functions & variables

Software Restructuring

- Restructure software to make it easier to understand and change.
- Use tools to analyze source code
  - Improve modularity through clustering analysis.
  - Try to reduce coupling and increase cohesion.
  - Try to impose good structure on ill-structured code.

Software Refactoring

Resources.

Martin Fowler's influential book: *Refactoring – improve the design of existing code*. and a portal `http://www.refactoring.com/`.

Tom Mens et al. *A Survey of Software Refactoring*. TSE 30(2), 2004

- Restructuring existing code by altering its internal structure without changing its external behavior.
- Are following activities a refactoring technique?
  - Adding new functionalities
  - Fixing functionality bugs
  - Tuning performance
  - Patching security

Refactoring Rythms

- Development = Adding feature, Refactoring
- Refactoring = (testing, small steps)*
- Small steps =

```
  Extract Methods
| Move Methods
| Rename Methods
| Replace Temp with Query
| Replace conditional with polymorphism
| Replace Type code with State/Strategy
| Self Encapsulate Field
 ...
```

[Some examples were discussed in Tutorial 8] *Any fool can write code that a computer can understand. Good programmers write code that humans can understand.*

Bad code smells

Putting things together when changes are together

- Duplicate code (clones): feature envy
- Complex control, Long method Use Hammock graph: single entry/single exit
  - comments signal semantic distance
  - conditional and loops
- Complex data, Long parameter list
- OO specific: large class, switch statements, parallel inheritance, middle man, message change, temporary fields, data class, etc.

Software Reverse Engineering

- Extract specification and design information from existing source code.
- Analyze data usage and calling patterns
- Attempt to identify and then specify underlying functionality.
- Reverse engineering difficulty increases with complexity of the software.

Software Re-engineering[a]

- Software quality and structure decays over time
  - Maintenance by ”patching” the software
  - Documentation of patches may be poor or non-existent
  - Source code gets lost due to programmer turnover, inadequate archives, organizational changes
  - Long term maintenance may introduce errors and inefficiencies into the software
- Software re-engineering involves the reimplementation of all or key parts of important software systems
- Goal of re-engineering is to reduce maintenance costs and to improve software quality
- Re-engineering may require *design recovery* from *binary (!!)* programs
  - Because source code or original design have been lost
  - Because patched software doesn’t correspond to any documentation

[a] Also known as the *Dusty Deck* problem