**CH11: Maintaining the System**

- maintenance process can be difficult
- * The Changing System
- * The Nature of Maintenance
- * Maintenance Problems
- * Measuring Maintenance Characteristics
- * Maintenance Techniques and Tools
- * Software Rejuvenation

**The Changing System**

- Any work done to change the system after it is in operation is considered to be maintenance.
- Unlike hardware, software does not degrade or require periodic maintenance (?).
- Types of Systems:
  - S, P, E

**S-System: the solution of the problem is well-known**

- Real world
  - Problem
    - Requirements specification
      - System
        - Information
          - Comparison
          - Subject to change

**P-System: based on a practical abstraction of the problem**

- Real world
  - Problem
  - Abstraction
    - Requirements specification
      - System
        - Information
          - Comparison
          - Subject to change

**E-System: embedded in the real world and changes as the world does.**

- Real world
  - Problem
  - Abstraction
    - Requirements specification
      - System
        - Information
          - Comparison
          - Subject to change

**Change During the System Life Cycle**

- Everything Change
- Take change into consideration during system development.
- Make it easier to change during maintenance
The System Life Span
- Is it possible to build a system right the first time?
- Maintenance state of development is the evolutionary phase of the system.
- Legacy systems were built earlier when our needs and environment were different.

Development Time vs. Maintenance Time
- Typical development project takes between 1 and 2 years, but
- requires an additional 5 to 6 years of maintenance time!
- 80-20 Rule: Twenty percent of the effect is in development and eighty percent is in maintenance.

System Evolution vs. System Decline: When shall we discard the old system and build a new one to replace it?
- Is the cost of maintenance too high?
- Is the system reliability unacceptable?
- Can the system no longer adapt to further change, and within a reasonable amount of time?
- Is system performance still beyond prescribed constraints?
- Are system functions of limited usefulness?
- Can other systems do the same job better, faster, or cheaper?
- Is the cost of maintaining the hardware great enough to justify replacing it with cheaper, newer hardware?

“Laws of Software Evolution”
- Continuing change: changing till replacing by recreated version
- Increasing complexity: structure deteriorates and complexity increases
- Fundamental law of program evolution: subjected to self-regulating with statistically-determinable trends and invariances.
- Conservation of organizational stability (invariant work rate): rate in a programming project is statistically invariant.
- Conservation of familiarity (perceived complexity): release content of the successive releases is statistically invariant.

The Nature of Maintenance
- Corrective Maintenance: maintenance in respond to problems resulting from faults during day-to-day system usage.
- Adaptive Maintenance: a change introduced in one part of the system requires changes to other parts.
- Perfective Maintenance: make changes to improve some aspect of the system
- Preventive Maintenance: make change to prevent failures.

Use of Maintenance Time
- Perfective 50%
- Adaptive 25%
- Corrective 21%
- Preventive 4%
Maintenance Problems
- How do we balance the need for change with the need for keeping a system accessible for users?
- Staff Problems
- Technical Problems
- The Need to Compromise
- Maintenance Cost

Staff Problems
- Limited Understanding: 47% of software maintenance effort is devoted to understanding the software to be modified.
- User’s Limited Understanding: provide incomplete and misleading data when reporting a problem
- Management Priorities: view maintaining and enhancing as more (or less?) important than building new applications

Staff Problems: Morale
- 11.9% of the problems during maintenance result from low morale and productivity.
- During maintenance, 8% of the problems result from a programmer’s being pulled in too many directions at once, and thus being unable to concentrate on one problem long enough to solve it.

Technical Problems
- Number one problem! The y2k problem, “year 2000 problem,” is a good example of where simple but narrow design decisions can have a major effect on maintenance.
- Maintaining object-oriented programs can be problematic: the design often involves components that are highly interconnected by complex inheritance schemes.

More Technical Problems
- Inadequate design specifications and low-quality programs and documents account for almost 10% of maintenance effort.
- Testing Difficulties: Testing the system in use can be a problem.
  + Solution: tests are often run on duplicate system

The Need to Compromise
- Programmer may be forced to compromise elegance and design principles because a change is needed immediately.
- The team is forced to concentrate is resources on a problem about which it may have little understanding.
**Maintenance Cost**

- All the problems of maintaining a system contribute to the high cost of software maintenance.
- Maintenance costs may be up to 80% of a system’s lifetime costs.
  - A series of repairs and enhancements usually leads to fragmentation of system, poor document, ... -> more difficult and more effort needed for future maintenance.
- Staff Specialization: leads to exponential increase in resources devoted to maintenance.

**Measuring Maintenance Characteristics**

- External view of maintainability: records the following info for each problem:
  - the time at which the problem is reported
  - any time lost due to administrative delay
  - the time required to analyze the problem
  - the time required to specify which changes are to be made
  - the time needed to make the change
  - the time needed to test the change
  - the time needed to document the change

**Internal Attributes Affecting Maintainability**

- the more complex the code, the more effort required to maintain it
- measuring how complex:
  - **Cyclomatic Number** is a metric that captures an aspect of the structure complexity of source code by measuring the number of linearly independent paths through the code.

**Finding the Cyclomatic number**

\[ \text{Cyclomatic Number} = \text{edges} - \text{nodes} + 2 \]

- \( \text{edges} \) = number of separated segments in the graph
- \( \text{nodes} \) = number of decision statements in the code + 1
- Cautions:
  - there are other attributes that contribute to complexity that are not captured by its structure

**Graphs for Cyclomatic number calculation**

**Readability**

- Text: \( \text{Readability} = \frac{1}{\text{Fog Index}} \)
  - Fog Index = 0.4 * number of words / number of sentences + percentage of words of 3 or more syllables
- Software: \( \text{Readability of source code} = 0.295 \times (\text{average normalized length of variables}) - 0.499 \times (\text{number of lines containing statements}) + 0.13 \times (\text{Cyclomatic number}) \)
Maintenance Techniques and Tools

- **Configuration Management** keeps track of changes and their effects on other system components.
- **Impact analysis** is the evaluation of the many risks associated with the change, including estimates of effects on resources, effect, and schedule.

Software maintenance Activities

- Preventive
  - Perfective
  - Corrective
  - Perfective

Workproducts and Traceability

- A workproduct is any development artifact whose change is significant.
- Vertical traceability expresses the relationship among the parts of the workproduct.
- Horizontal traceability addresses the relationships of the components across collections of workproducts.

Horizontal traceability

Complexity as result of maintenance

- If the number of paths in the graph increases after the change,
  - then the system is likely to be more unwieldy and difficult to maintain.
- If the number of edges going into the node (in-degrees) and number of edges going out the node (out-degrees) increase substantially,
  - then the system may be harder to maintain in the future.
Tools for Maintenance
- Text Editors
- File Comparator
- Compilers and Linkers
- Debugging Tools
- Cross-reference Generators
- Static Code Analyzers
- Configuration Management repositories

Software Rejuvenation
- addresses maintenance challenge by trying to increase the overall quality of an existing system
- Redocument: analysis of the source code to provide more info to assist maintenance.
- Restructure: transforming ill-structured code into well-structured one.

Reverse engineer
- Reverse engineer: recreating design and specification information from the code.
- Reengineering: reverse engineer then “forward engineer” to change the specification and design

Taxonomy of software rejuvenation

Redocumentation
Leading to additional information about:
- Internal representation of the code
- Code complexity and size data
- Program flow
- Data usage
- Component calling hierarchy
- Cross-reference tables
- Data interface tables
- Component and variable tables
- Source code

Restructuring
- Structured code
- Internal representation of the code
- Simplify internal representation
- Regenerate structured code
- Reverse engineer code
- Manage and modify representation
- Based on acceptable software methods
- Forward engineer: complete and modify representation
- Regenerate code
- Reverse engineer: produces design and specification based on accepted software methods
- Reports on structure, complexity, volume, data, etc.
- Not based on software methods
- Iteratively simplify structure and eliminate dead code
- Regenerate code
- Internally represent through process
- From code:
- From code
- From code
- Forward progression represents
- Reverse engineering
---

**Concluding Remark:**

**What this course means for you**

- Software development processes
- Large software projects
- Team Work
- Documentation and communication
- Learn the terms of the trade
- What to expect when you work in software development co. ($$$) (???)

• “Live Long and Prosper”

---