Software Maintenance and Evolution

Overview

Jeff Offutt

SWE 437
George Mason University
2008

Thanks to Ian Sommerville, Susan Eisenbach, Jeff Lei, Hausi A. Müller, Oscar Nierstrasz, and Hiern Variava

Software Maintenance

"When the transition from development to evolution is not seamless, the process of changing the software after delivery is often called **software maintenance**"
– Sommerville, 2004

- **Modifying a program after it has been put into use**
- **Maintenance does not normally involve major changes to the system's architecture**
- **Changes are implemented by modifying existing components and adding new components to the system**
- **Maintenance requires program understanding**

Importance of Maintenance

- **Organizations have huge investments in their software systems - they are critical business assets**
- **To maintain the value of these assets to the business, they must be changed and updated**
- **The majority of the software budget in large companies is devoted to modifying existing software rather than developing new software**

What is Software Maintenance ?

- **"A program that is used undergoes continual change or it becomes progressively less useful"**
  - **Manny Lehman**
- **Software maintenance is the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment.**
  - **IEEE definition 1983**
- **Average system takes 1-2 years to write and is in operation 5-6 years**
  - **Most software development is maintenance**

Software Changes are Inevitable

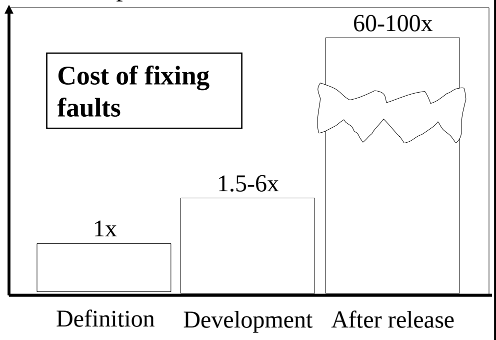
- **We cannot avoid changing software**
  - **New requirements emerge when the software is used**
  - **The business environment changes**
  - **Faults must be repaired**
  - **New computers and equipment is added to the system**
  - **The performance or reliability of the system may have to be improved**
- **Software is tightly coupled with the environment**
  - **When software is installed in an environment it changes that environment and therefore changes the software requirements**
- **A key problem for organizations is implementing and managing change to their existing software systems**

Management Myths

- ***Myth***: We already have a book that's full of standards and procedures for building software, won't that provide my people with everything they need to know?
  - ***Reality***: The book of standards may very well exist, but is it used? In many cases, the answer to the following questions is "no"
    - Are software practitioners aware of its existence?
    - Does it reflect modern software engineering practice?
    - Is it complete?
    - Is it streamlined to improve time to delivery while still maintaining a focus on quality?
- ***Myth***: If we get behind schedule, we can add more programmers and catch up
  - ***Reality***: Software development is not a mechanistic process like manufacturing. In the words of Brooks: "adding people to a late software project makes it later"
- ***Myth***: If I decide to outsource the software project to a third party, I can just relax and let that firm build it
  - ***Reality***: If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsources software projects

Customer Myths

- ***Myth:*** A general statement of objectives is sufficient to begin writing programs—we can fill in the details later
  - ***Reality:*** A poor up-front definition is the major cause of failed software efforts. A formal and detailed description of the information domain, function, behavior, performance, interfaces, design constraints, and validation criteria is essential. These characteristics can be determined only after thorough communication between customer and developer.
- ***Myth***: Project requirements continually change, but change can be easily accommodated because software is flexible
  - ***Reality***: It is true that software requirements change, but the impact of change varies with the time at which it is introduced



Practitioner's Myths

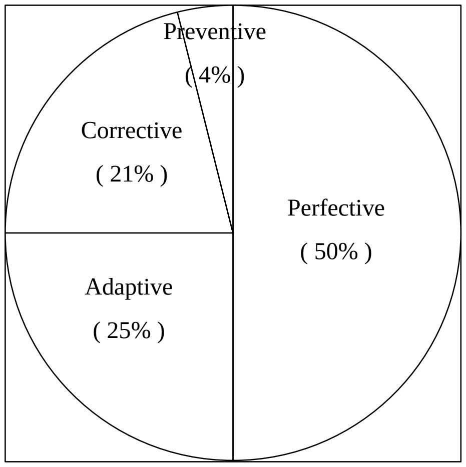
- ***Myth*: Once we write the program and get it to work, our job is done**
  - ***Reality*: Someone once said that "the sooner you begin 'writing code', the longer it'll take you to get done." Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.**
- ***Myth*: Until I get the program "running" I have no way of assessing its quality**
  - ***Reality*: One of the most effective software quality assurance mechanisms can be applied from the inception of a project—the formal technical review. Software reviews are more effective than testing for finding certain classes of software defects.**
- ***Myth*: The only deliverable work product for a successful project is the working program**
  - ***Reality*: A working program is only one part of a software configuration that includes many elements. Documentation provides a foundation for successful engineering and, more important, guidance for software support.**
- ***Myth*: Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down**
  - **Reality: Software engineering is not about creating documents. It is about creating quality. Better quality leads to reduced rework. And reduced rework results in faster delivery times.**

Types of Maintenance

- Perfective maintenance - Enhancement - new operations and refinements to old functions (about 50%)
- Adaptive maintenance - Modifying the application to meet new operational circumstances (about 25%)
- Corrective maintenance - Eliminating errors in the program functionality (about 21% of all maintenance activity)
- Preventive maintenance - Modifying a program to improve its future maintainability (about 4%)
  - Why so little preventive maintenance?
  - Preventive programming is more constrained by time than other types

Distribution of Maintenance

Percent of Total Maintenance



Development vs. Maintenance

not directly linked to the real world	directly driven by the real world
freedom	constrained by existing system
defects have no immediate effect	defects can disrupt production when system is in use
current process and tools available	system does not use current process or tools
standards may be enforced	shifting standards, if any

Five Types of Software Maintenance

Corrective Maintenance
- Identify and remove defects
- Correct actual errors

Perfective Maintenance
- Improve performance, dependability, maintainability
- Add new functionality

Adaptive Maintenance
- Adapt to a new/upgraded environment (e.g., hardware, operating system, middleware)

Preventive Maintenance
- Identify and detect latent faults
- Systems with safety concerns

Emergency Maintenance
- Unscheduled corrective maintenance

(Risks due to reduced testing)

Two Types are Fault Repair

Corrective Maintenance
- **Identify and remove defects**
- **Correct actual errors**

Fault Repair

Preventive Maintenance
- **Identify and detect latent faults**
- **Systems with safety concerns**

Emergency Maintenance
- **Unscheduled corrective maintenance**

(Risks due to reduced testing)

Two Types are Migration

Post - Delivery

Perfective Maintenance
- **Improve performance, dependability, maintainability**
- **Add new functionality**

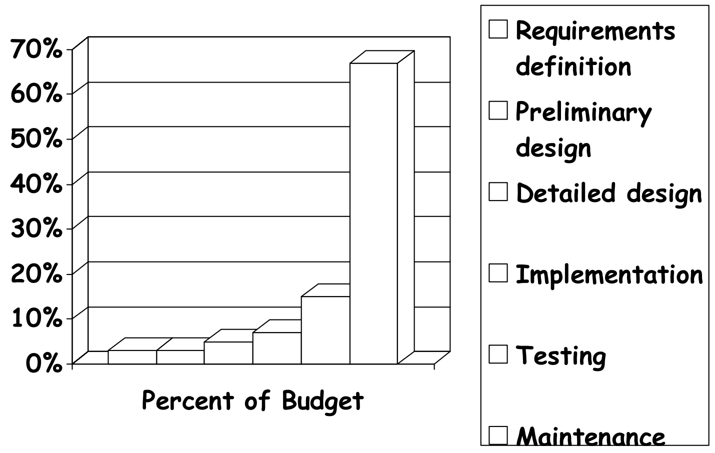
Adaptive Maintenance
- **Adapt to a new/upgraded environment (e.g., hardware, operating system, middleware)**

Development / Migration

Software Maintenance Problems

- Most computer systems are difficult and expensive to maintain
- Software changes are poorly designed and implemented
- The repair and enhancement of software often injects new faults that must later be repaired

Relative Costs of Maintenance



- Most of the software budget is devoted to maintaining systems
- Sommerville claims that 90% of all costs are evolution

Maintenance Costs

- Usually greater than development costs (2 – 100 times depending on the application)
- Affected by both technical and non-technical factors
- Increases as software is maintained
  - Maintenance corrupts the software structure, making further maintenance more difficult
- Aging software can have high support costs (old languages, compilers etc.)

Maintenance Cost Factors

- Team stability
  - Maintenance costs are reduced if the same staff stay involved
- Contractual responsibility
  - If the developers of a system may have no contractual responsibility for maintenance, there is no incentive to design for future change
- Staff skills
  - Maintenance staff are often inexperienced and have limited domain knowledge
- Program age and structure
  - As programs age, their structure is degraded and they become harder to understand and change

Technical Aspects of Maintenance

- Maintainability
- Impact Analysis
- Ripple Effect
- Traceability
- Legacy systems

Maintainability

The ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment [IEEE90]

What Affects Maintainability ?

- Application age : Older programs were probably worse written and have probably been patched more
- Size : Measured in KLOC, number of input/output files
- Programming language : Newer languages are supposed to yield more maintainable code than older languages
- Processing environment : Files harder to maintain than databases, real-time harder than non-real-time
- Analysis and design methodologies : Well designed software is supposed to be easier to maintain
- Formatting and documentation : Well written source is easier to understand and modify

What Affects Maintainability … ?

- Modularization : Small reasonably self contained pieces of code should be easier to maintain
  - Motivating factor behind OO — data abstraction, information hiding, inheritance, etc.
- Documentation generation : Maintenance of documentation is as expensive as maintenance of code
- End-user involvement : Some researchers believe when end users are more involved maintenance is less needed
- Maintenance management : Scheduling and the attitudes of management to affects productivity

Impact Analysis

Helps to determine the cost of making a change

- **Translate the change from requirements to implementation to decide if it is viable or should be rejected**
- **Determine the origin of the change and suggest solutions**
- **All solutions be investigated to determine they can be applied to all software components**
- **Make a decision on the best implementation route or to make no change**

Ripple Effect

Changes in one software location can impact other components

- **Ripple effects cannot be determined fully using static analysis of the source**
- **Therefore dynamic analysis must be used**
  - **We have to run the program to understand …**
- **Impact analysis is needed :**
  - **To ensure that the change has been correctly and consistently bounded**
  - **To identify all objects impacted by changes in the primary sector**

Traceability

The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor successor or master subordinate relationship to one another (IEEE91)

- **Traceability provides semantic links for impact analysis**
- **Some types of traceability links are very hard to determine**

Legacy Systems

A software system that is still in use but the development team is no longer active

- **A <u>legacy system</u> is a piece of software that :**
  - **Has been inherited**
  - **Is still valuable (or essential)**
- **Very hard to change !**

You can’t throw it away, but you can’t change it either !

Legacy Systems

- **Typical characteristics of legacy systems :**
  - **Very old and large**
  - **Has been heavily modified**
  - **Based on old technology**
  - **No documentation available**
  - **No original developers available**
  - **Often support very large quantities of live data**
  - **The software is often at the core of the business and replacing it would be a major expense**
- **Maintenance has often become almost impossible**
- **Dealing with legacy systems is hard but some options exist ...**

Legacy System Options

- **Keep using the old system, possibly subcontracting the maintenance**
- **Replace software with a package**
- **Re-implement from scratch**
- **Discard software and discontinue**
- **Freeze maintenance and bring in new system**
- **Encapsulate the old system in a new external "wrapper" interface**
  - **"web-enabled legacy system" is one of the latest buzzwords**
  - **Patriot web is a classic example**
- **Reverse engineer the legacy system and develop new software**

Maintenance Terminology (Perfective)

- **Reverse engineering : The process of transforming from source to a more abstract version of the product**
  - Used for forward engineering new systems and maintenance
  - 60% of maintenance time spent understanding code
- **Restructuring : The process of transforming a product to another product at the same level of abstraction**
  - Improving the structure and documentation
  - Changing variable definitions and scope
- **Design Recovery : A particular type of reverse engineering where a design for the system (with a connection to real-world requirements) is created**
- **Re-engineering : Reverse engineering followed by forward engineering adhering to good practice**

Software Reverse Engineering

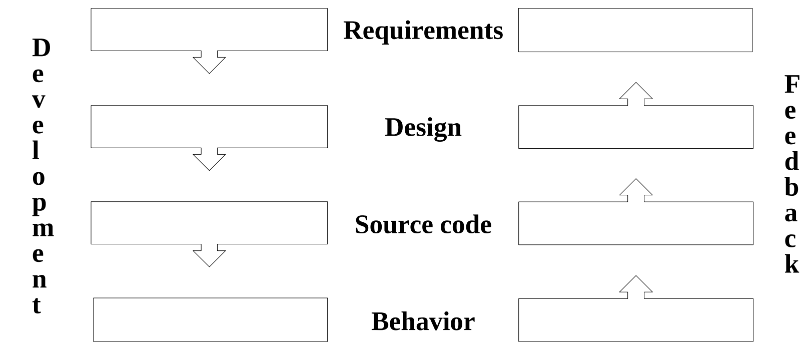
- **A two-step process**
  1. Information extraction
  2. Information abstraction
- **A three-step process**
  1. Information gathering
  2. Knowledge organization
  3. Information navigation, analysis, and presentation
- **Analyzing subject system**
  - to identify its current components and their dependencies
  - to extract and create system abstractions and design information
- **The subject system is not altered; however, additional knowledge about the system is produced**

Software Reverse Engineering ...

- Feedback loops in life cycle models (e.g., waterfall or spiral model) are opportunities for reverse engineering
- Related terms
  - Abstraction and composition
  - Design recovery and concept assignment
  - Re-documentation
  - Inverse engineering
  - Static and dynamic analysis
  - Summarizing resource flows and software structures
  - Change and impact analysis
  - Maintainability analysis
  - Migration analysis
  - Portfolio analysis
  - Economic analysis

Forward Engineering

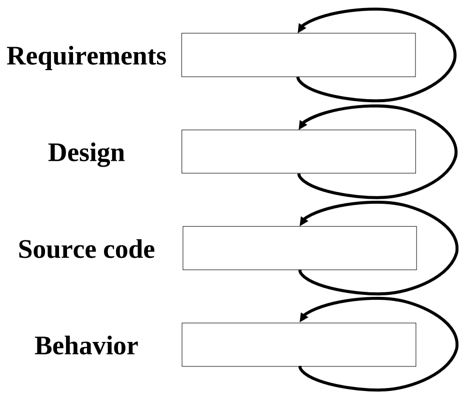
Traditional software process of moving from high-level abstractions and logical implementation-independent designs to the physical implementation of a system



Restructuring

Transforming from one representation to another at the same relative abstraction level, while preserving the subject's system external behavior

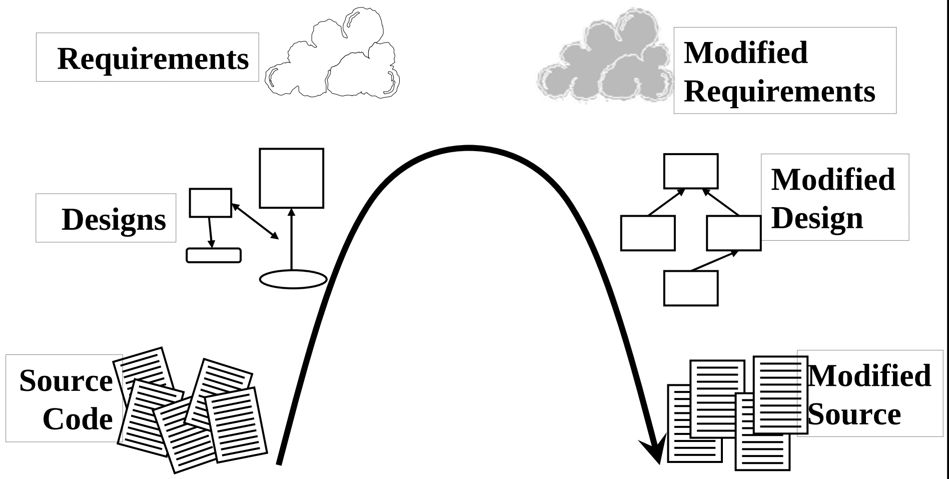
Intra-level feedback cycles



Reengineering Categories

- **Automatic restructuring**
- **Automatic transformation**
- **Semi-automatic transformation**
- **Design recovery and reimplementation**
- **Code reverse engineering and forward engineering**
- **Data reverse engineering and schema migration**
- **Migration of legacy systems to modern platforms**

The Reengineering Horseshoe Model



Reengineering Categories ...

- **Automatic restructuring**
  - **To obtain more readable source code**
  - **Enforce coding standards**
- **Automatic transformation**
  - **To obtain better source code**
  - **Web-enabling of source code**
  - **Simplify control flow (e.g., dead code, gotos)**
  - **Refactoring and re-modularizing**
- **Semi-automatic transformation**
  - **To obtain better engineered system (e.g., re-architect code and data)**
  - **Semi-automatic construction of structural, functional, and behavioral abstractions**
  - **Re-architecting or re-implementing the subject system from these abstractions**

‼ Reality Check ‼

Sorry to say, but …

All of the previous information comes from publications in the 1990s …

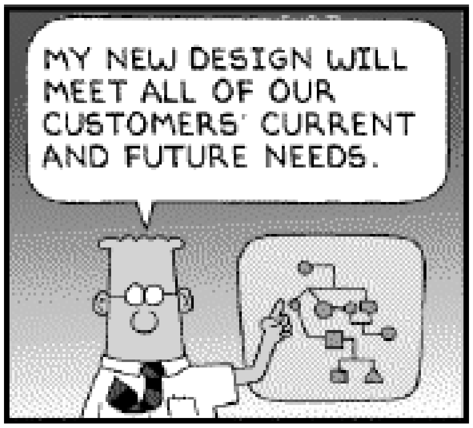
Based on knowledge from the 1980s …

When our software was "single-building size" !

How out-of-date is this information for building integrated collections of continuously evolving cities ???? (that is … 2008)

Very !!

(Over) Confidence


MY NEW DESIGN WILL MEET ALL OF OUR CUSTOMERS' CURRENT AND FUTURE NEEDS.

Knowing we're right

Arrogance : based on hopes and dreams

Confidence : based on experience, knowledge, ability

Maintenance & Evolution

- Software Maintenance
  - consists of the activities required to keep a software system operational and responsive after it is accepted and placed into production
- Software Evolution
  - a continuous change from a lesser, simpler, or worse state to a higher or better state

Premise for 2000s

- Like death and taxes, software evolution is inevitable
- Unfortunately, software evolution is still poorly understood and supported
- We need techniques and tools to facilitate graceful software evolution

Software Evolution

"Software development does not stop when a system is delivered but continues throughout the lifetime of the system"
– Sommerville, 2004

- **The system changes relate to changing needs – business and user**
- **The system evolves continuously throughout its lifetime**
- **The process is a spiral process involving requirements, design & implementation throughout the lifetime of the system**

Lehman's Laws of Software Evolution

1. **Law of Continuing Change (1974)**
   – **Software that is used in a real-world environment must change or become less and less useful in that environment**
2. **Law of Increasing Complexity (1974)**
   – **As an evolving program changes, its structure becomes more complex, unless active efforts are made to avoid this phenomenon**
3. **Law of Self Regulation (1974)**
   – **Program evolution is a self-regulating process. System attributes such as size, time between releases, and the number of reported errors are approximately invariant for each system release**
4. **Law of Conservation of Organizational Stability (1980)**
   – **Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development**

Lehman's Laws of Software Evolution

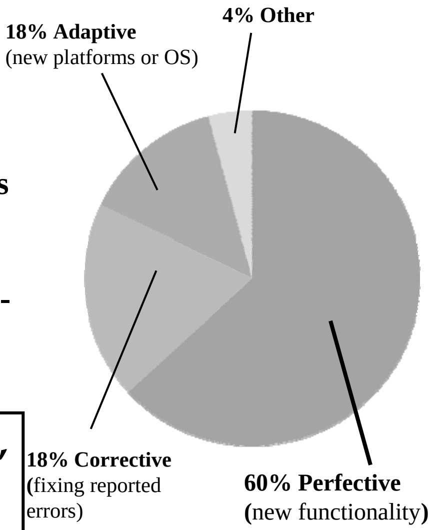
5. **Law of Conservation of Familiarity (1980)**
   – **Over the lifetime of a system, the incremental system change in each release is approximately constant**
6. **The Law of Continuing Growth (1980)**
   – **The functionality offered by systems has to continually increase to maintain user satisfaction**
7. **The Law of Declining Quality (1996)**
   – **The quality of systems will appear to be declining unless they are adapted to changes in their operational environment**
8. **The Feedback System Law (1996)**
   – **Evolution processes incorporate multi-agent, multi-loop feedback systems and you have to treat them as feedback systems to achieve significant product improvement**

Lehman, Ramil, Wernick, Perry and Turski, "Metrics and Laws of Software Evolution—The Nineties View," ***Proceedings of the 4th International Software Metrics Symposium (METRICS '97),*** **IEEE, November 1997, pp 20-32 (http://www.ece.utexas.edu/~perry/work/papers/feast1.pdf)**

The Myth of "Software Maintenance"

- **60% of "maintenance" is new functionality**
- **50-75% of development effort is "maintenance," that is, continuous development**
- **Modern methods lead to longer-lived systems, hence more maintenance**

No software maintenance, just continuous software evolution



The Pace of Change is Increasing

- Hardware advances lead to new, bigger software applications
- The rate of change (that is, new features) is increasing

How can we deal with the spiraling need to handle change ?

SWE 437

We will discuss specific ways to deal with this problem over the next few weeks