Software Maintenance

- Introduction
- Process Models
- Program Understanding
- Configuration Management
- Management Issues
- Conclusion

Software Maintenance

- Management and control of changes to a software product after delivery
  - Bug fix, new features, environment adaptation, performance improvement
- Often accounts for 40-70% of the cost of the entire life-cycle of a software product
  - The more successful a software product is, the more time it spends on maintenance

Software vs Programs

Software Components	Examples	
Program	1. Source code 2. Object code	
	1. Analysis/Specification	(a) Formal specification (b) Data flow diagrams
	2. Design	(a) High-level design (b) Low-level design (b) Data model
Documentation	3. Implementation	(a) Source code (b) Comments
	4. Testing	(a) Test design (b) Test results
Operating Procedures	1. Installation manual 2. User manual	

Maintenance vs Development

- Maintenance must work within the parameters and constraints of an existing system
  - The addition of a new room to an existing building can be more costly than adding the room in the first place
- An existing system must be understood prior to a change to the system
  - How to accommodate the change?
  - What is the potential ripple effect?
  - What skills and knowledge are required?

Why Maintenance?

- To provide continuity of service
  - Bug fixing, recover from failure, accommodating changes in the environment
- To support mandatory upgrades
  - Government regulations, maintaining competitive edges
- To support user requests for improvements
  - New features, performance improvements, customization for new users
- To facilitate future maintenance work
  - Re-factoring, document updating

Lehman's Laws

- Law of continuing change: systems must be continually adapted
- Law of increasing complexity: as a system evolves, its complexity increases unless work is done to maintain or reduce it
- Law of continuing growth: functionality must be increased continually to maintain user satisfaction
- Law of declining quality: system quality will appear to decline unless rigorously adapted

Major Activities

- Change identification
  - What to change, why to change
- Program understanding
  - How to make the change, what is the ripple effect
- Carrying out the change and testing
  - How to actually implement the change and ensure its correctness
- Configuration management
  - How to manage and control the changes
- Management issues
  - How to build a team

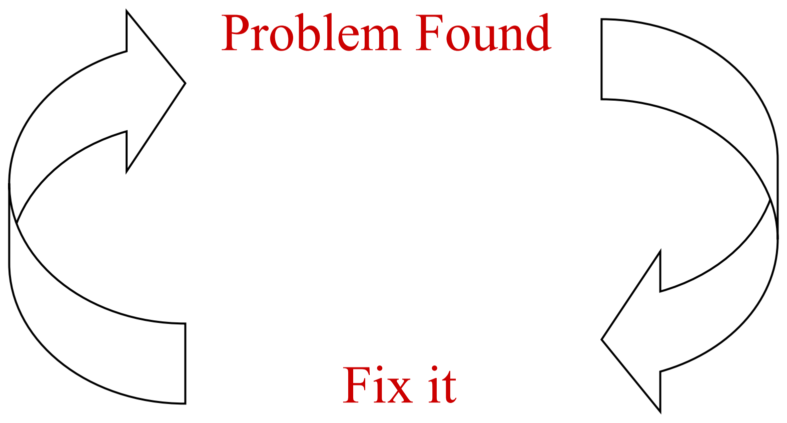
Software Maintenance

- Introduction
- Process Models
- Program Understanding
- Configuration Management
- Management Issues
- Conclusion

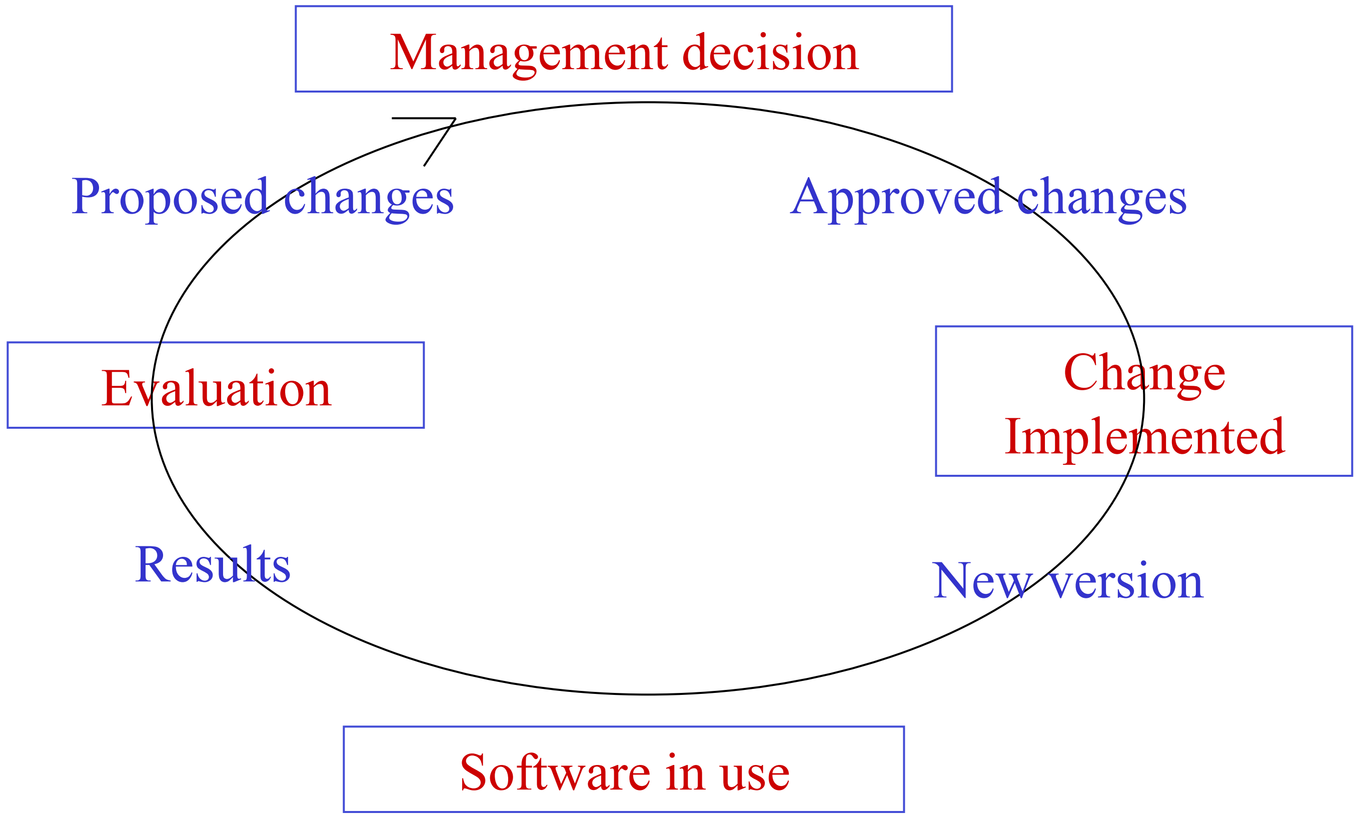
Development Models

- Code-and-Fix
  - Ad-hoc, not well-defined
- Waterfall
  - Sequential, does not capture the evolutionary nature of software
- Spiral
  - Heavily relies on risk assessment
- Iterative
  - Incremental, but constant changes may erode system architecture

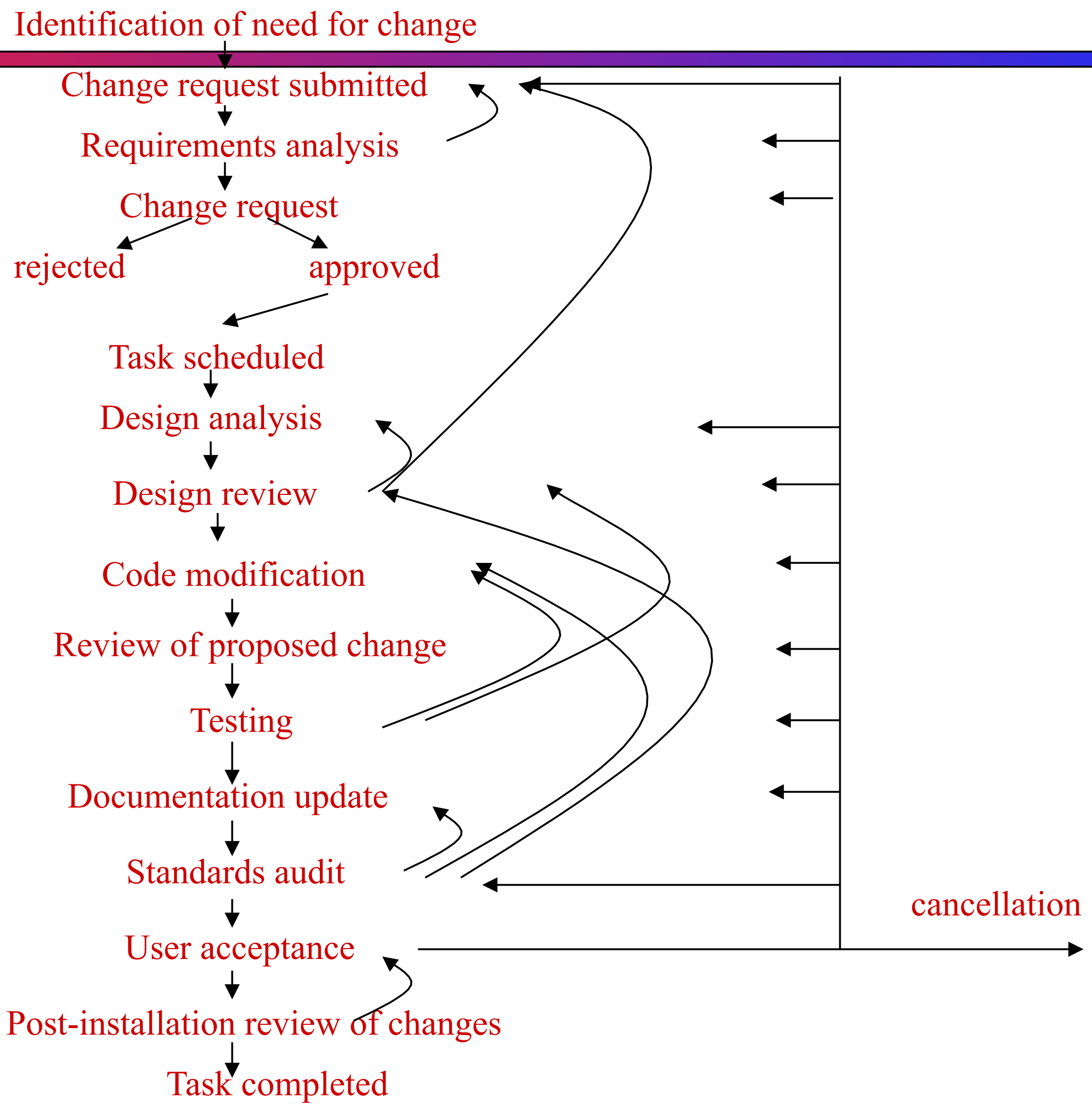
Quick-Fix Model



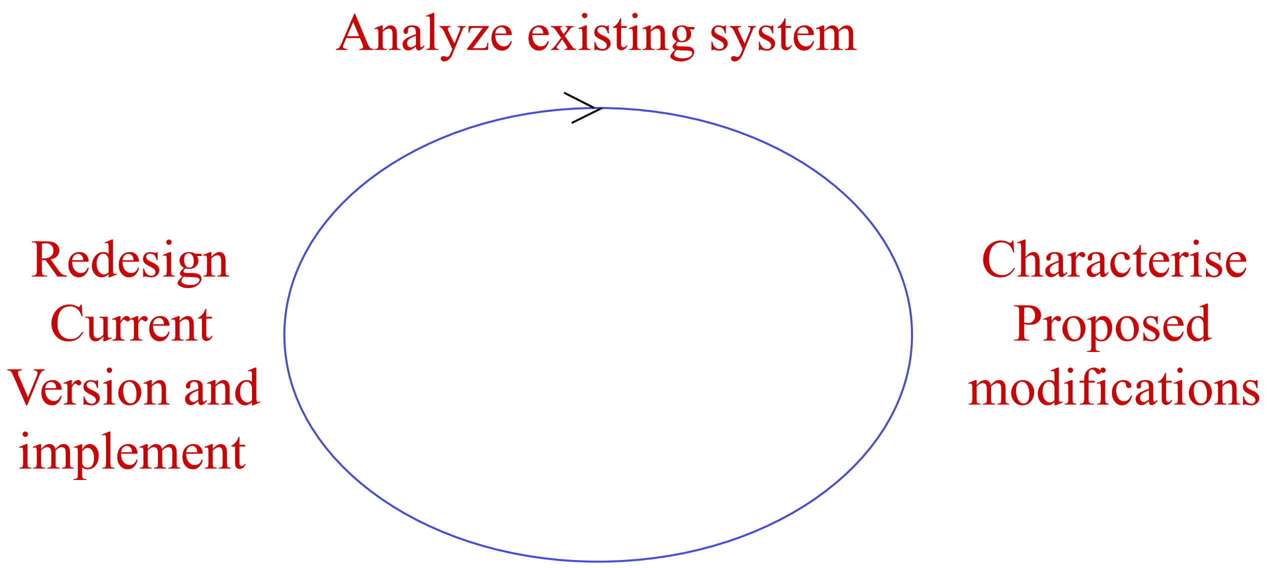
Boehm's Model



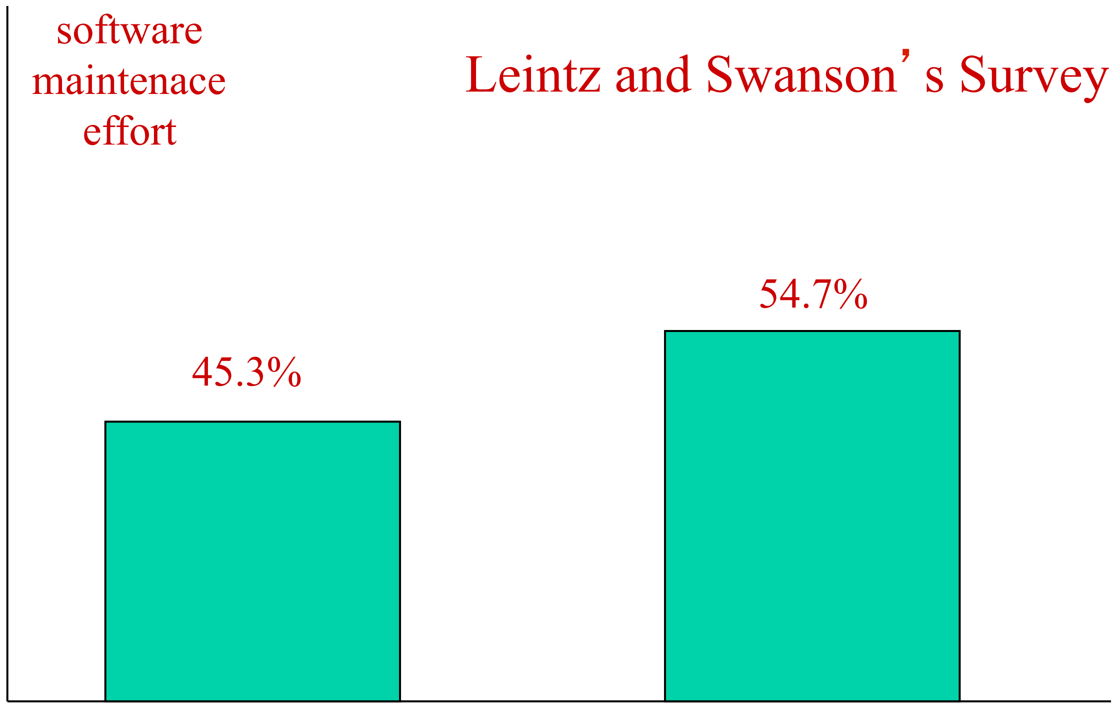
Osborne' s Model



Iterative Enhancement Model


Analyze existing system
Characterise Proposed modifications
Redesign Current Version and implement

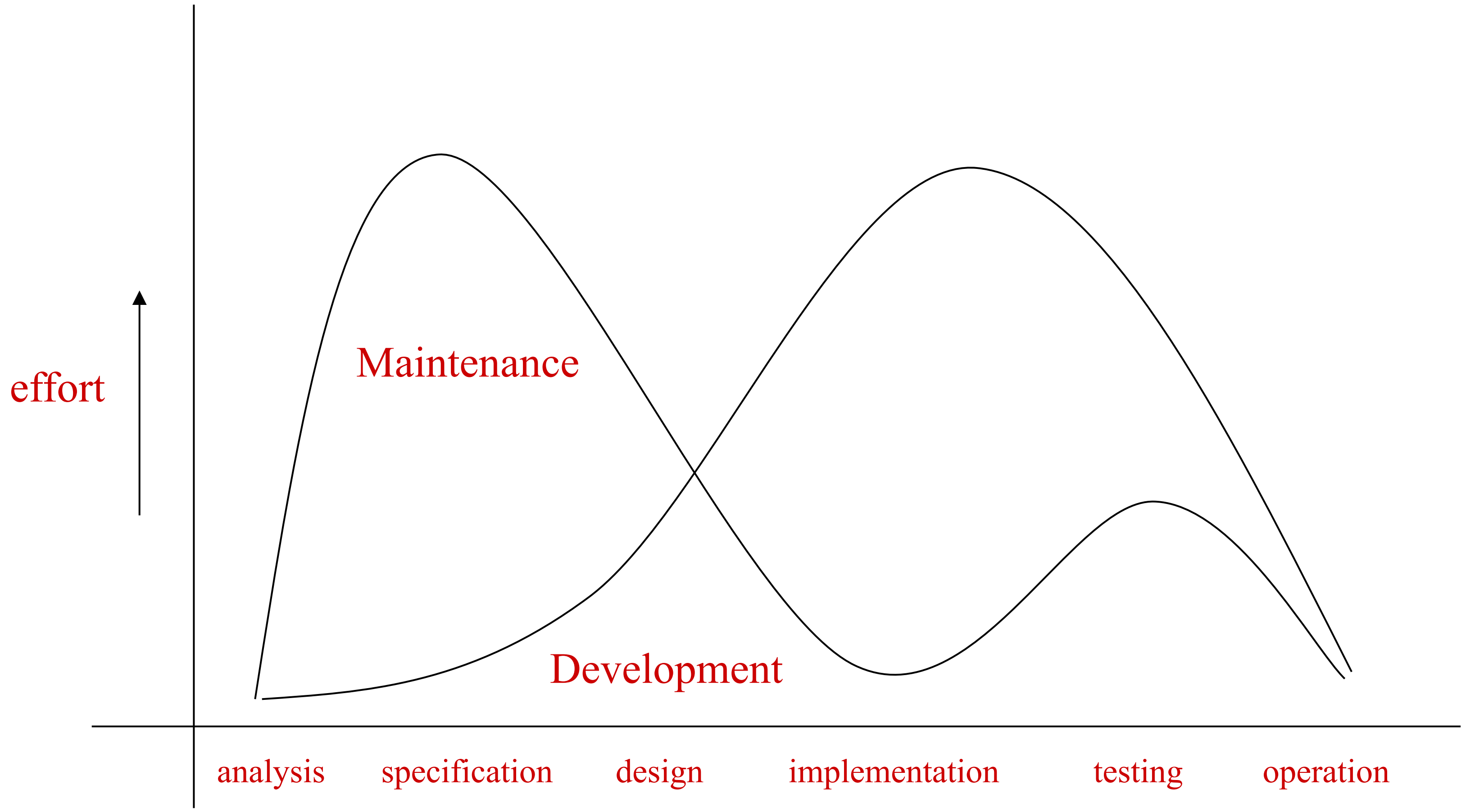
Maintenance Effort (1)



Non discretionary Maintenance
- emergency fixes
- debugging
- changes to input data
- changes to hardware

Discretionary maintenance
- enhancements for users
- documentation improvement
- improving efficiency

Maintenance Effort (2)



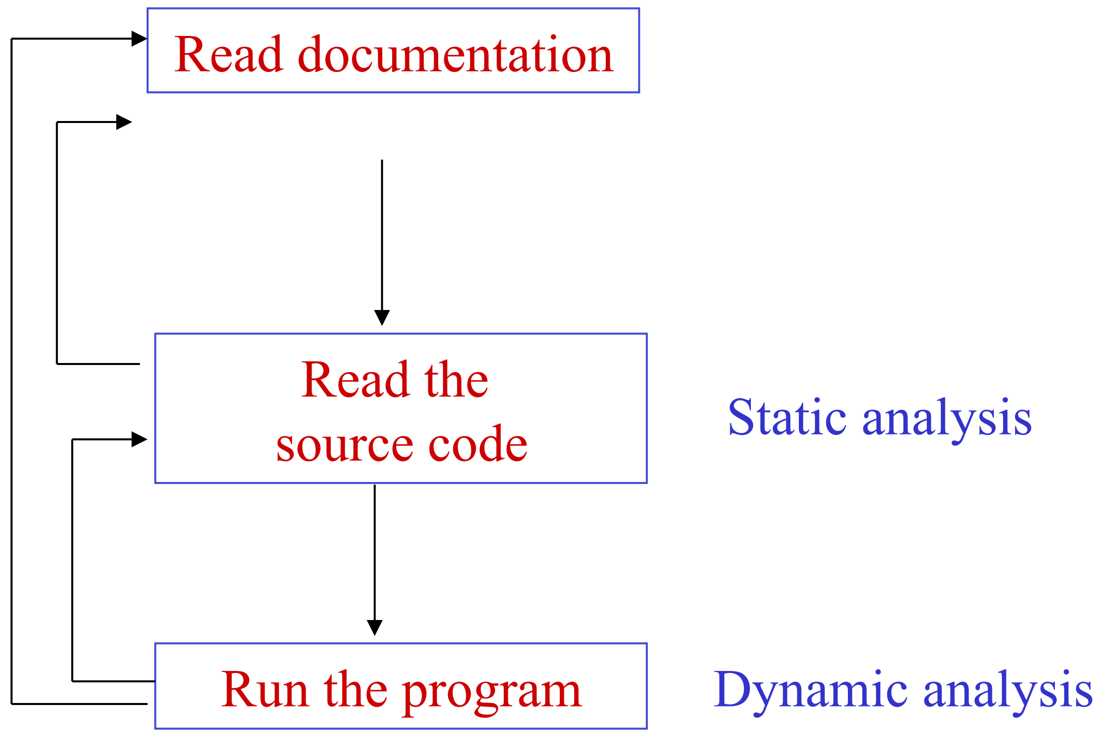
Software Maintenance

- Introduction
- Process Models
- Program Understanding
- Configuration Management
- Management Issues
- Conclusion

What to understand

- Problem Domain
  - Capture necessary domain knowledge from documentation, end-users, or the program source
- Execution Effect
  - Input-output relation, knowledge of data flow, control flow, and core algorithms
- Cause-Effect Relation
  - How different parts affect and depend on each other
- Product-Environment Relation
  - How the product interacts with the environment

Comprehension Process



Comprehension Strategies

- Top-down: start with the big picture, and then gradually work towards understanding the low-level details
- Bottom-up: start with low-level semantic structures, and then group them into high-level, more meaningful structures
- Opportunistic: A combination of top-down and bottom-up

Factors affecting understanding

- Expertise: Domain knowledge, programming skills
- Program structure: modularity, level of nesting
- Documentation: readability, accuracy, up-to-date
- Coding conventions: naming style, design patterns
- Comments: quality, shall convey additional information
- Program presentation: good use of indentation and spacing

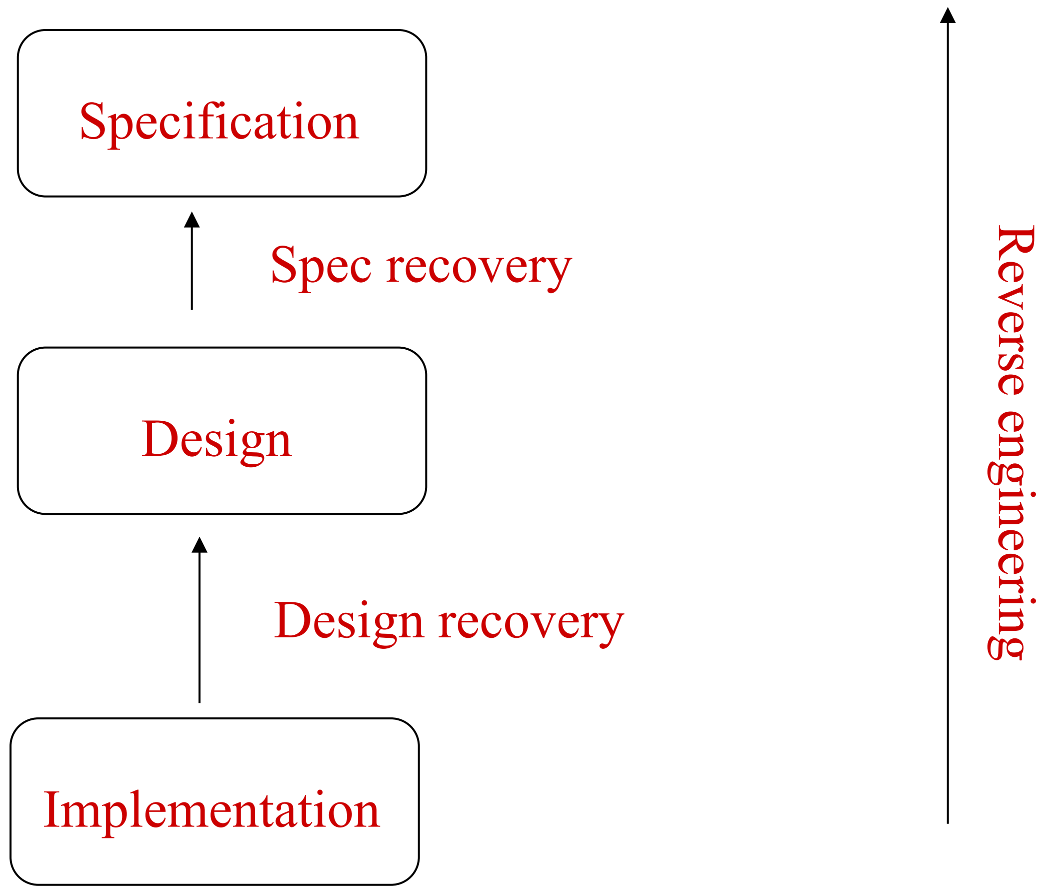
Reverse Engineering

- The process of analyzing the source code to create system representations at higher levels of abstraction
- Three types of abstraction
  - Function abstraction: what it does, instead of how
  - Data abstraction: abstract data type in terms of operations that can be performed
  - Process abstraction: communication and synchronization between different processes

Why RE?

- Allow a software system to be understood in terms of what it does, how it works and its architectural representation
  - Improve or provide documentation
  - Cope with complexity
  - Extract reusable components
  - Facilitate migration between platforms
  - Provide alternative views

Levels of RE


Specification
Spec recovery
Design
Design recovery
Implementation
Reverse engineering

Software Maintenance

- Introduction
- Process Models
- Program Understanding
- Configuration Management
- Management Issues
- Conclusion

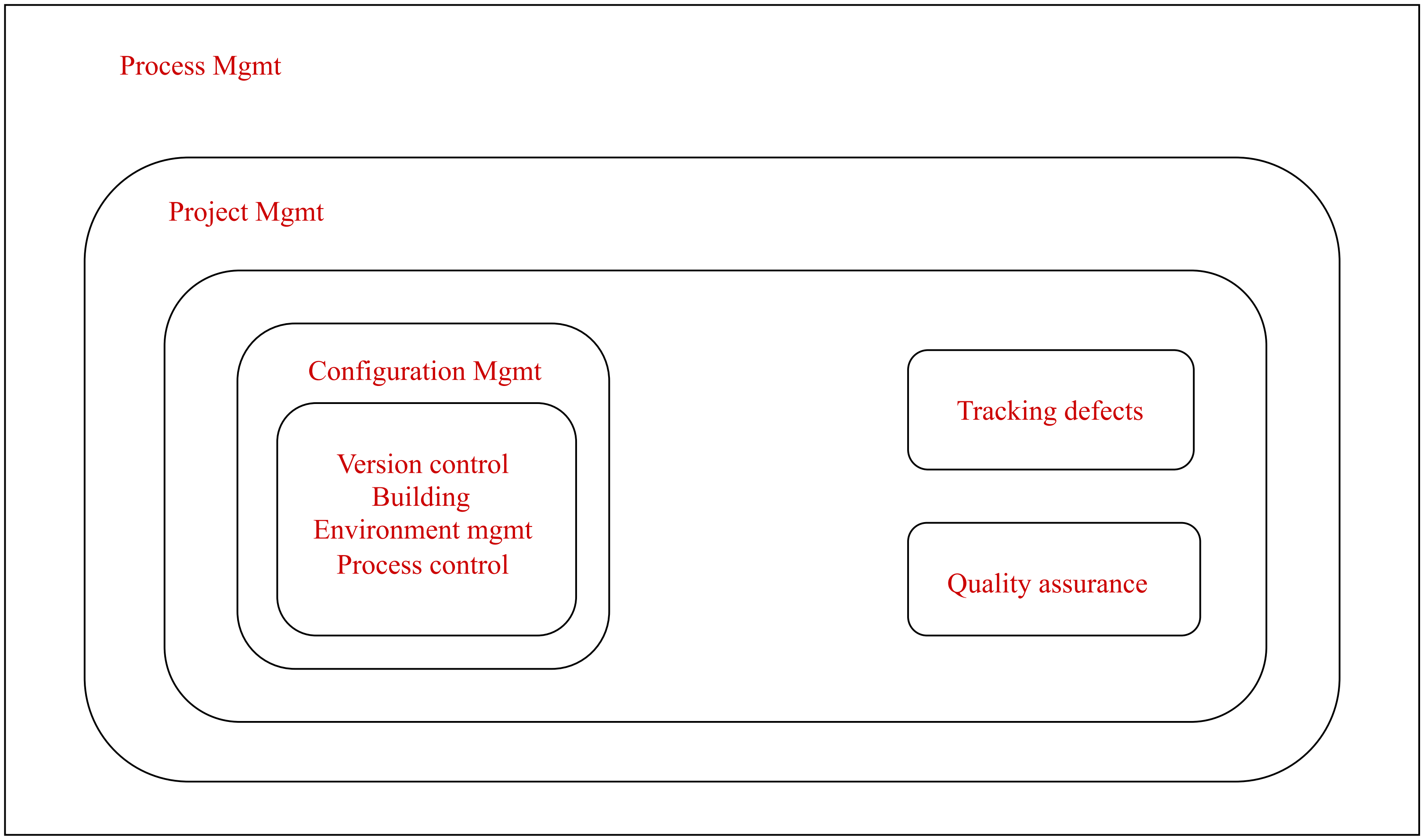
Why CM?

- Critical to the management and maintenance of any large system
  - Suppose that a customer reports a bug. Without proper control, it may be impossible to address this problem. (Why?)
  - CM allows different releases to be made from the same code base
  - CM also allows effective team work, auditing, and accounting

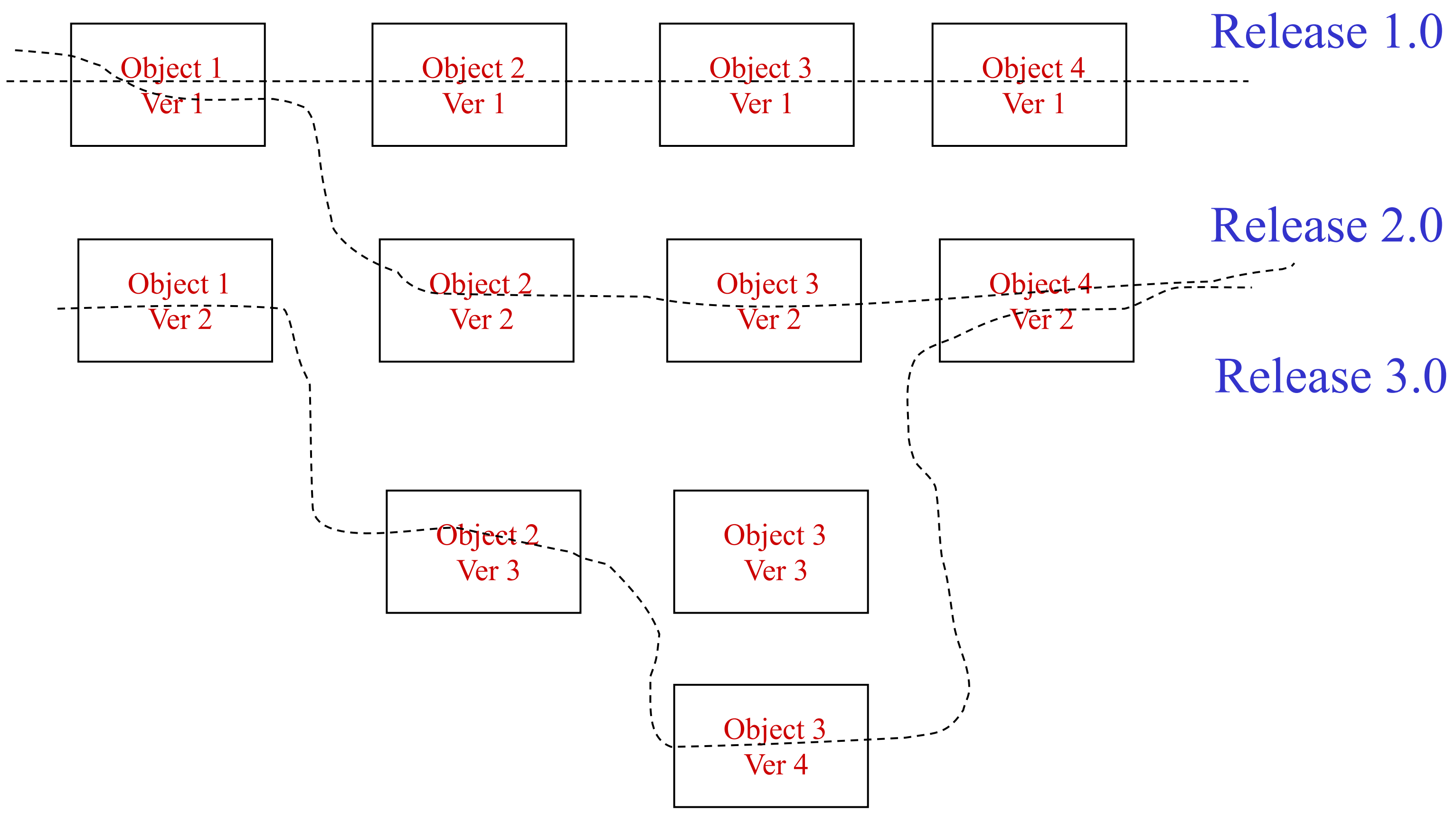
Major activities

- Identification of components and changes
- Control of the way changes are made
- Status accounting – recording and documenting all activities that have taken place
- Auditing changes – making the current state visible

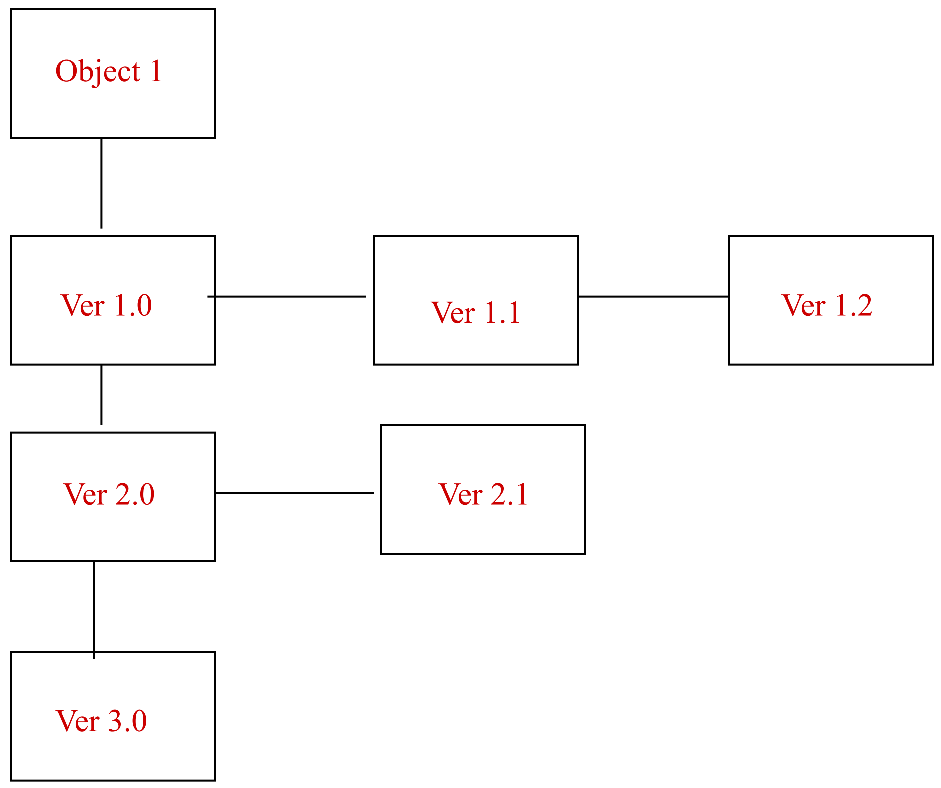
The Big Picture



Version Control (1)



Version Control (2)



Build

- One of the most frequently performed operations
- Incremental building: only rebuild objects that have been changed or have had a dependency change
- Consistency: must use appropriate versions of the source files
- Makefiles are often used to declare dependencies between different modules

Change Control

- Decide if a requested change should be made
  - Is it valid? Does the cost of the change outweigh its benefit? Are there any potential risks?
- Manage the actual implementation of the change
  - Allocate resources, record the change, monitor the progress
- Verify that the change is done correctly
  - Ensure that adequate testing be performed

Change Request Form

Name of system:
Version:
Revision:
Date:
Requested by:
Summary of change:
Reasons of change:
Software components requiring change:
Documents requiring change:
Estimated cost:

Software Maintenance

- Introduction
- Process Models
- Program Understanding
- Configuration Management
- Management Issues
- Conclusion

Responsibilities

- Maximize productivity
  - Personnel management
    - Choose the right people, motivate the team, keep the team informed, allocate adequate resources
  - Organizational mode
    - Combined or separate development/maintenance teams, module ownership, change ownership

Motivating the Team

- Rewards: financial rewards, promotion
- Supervision: technical supervision and support for inexperienced staff
- Assignment patterns: rotate between maintenance and development
- Recognition: properly acknowledge one's achievements
- Career structure: provide room for career growth

Education and Training

- Objective: To raise the level of awareness
  - Not a peripheral activity, but at the heart of an organization
- Strategies
  - University education
  - Conferences and workshops
  - Hands-on experience

Module Ownership

- Pros
  - The module owner develops a high level of expertise in the model
- Cons
  - No one is responsible for the entire system
  - Workload may not be evenly distributed
  - Difficult to implement changes that require collaboration of multiple modules

Change Ownership

- Pros:
  - Tends to adhere to standards set for the entire software system
  - Integrity of the change is ensured
  - Changes can be code and tested independently
  - Changes inspection tends to be taken seriously
- Cons:
  - Training of new personnel can be difficult
  - Individuals do not have long-lasting responsibilities

Software Maintenance

- Introduction
- Process Models
- Program Understanding
- Configuration Management
- Management Issues
- Conclusion

Conclusion

- Maintenance is a critical stage in the software lifecycle, and must be managed carefully
- Unlike new development, maintenance must work within the constraints of the existing system
- Central to maintenance is the notion of change. Changes must be managed and controlled properly.
- Not only the code needs to be maintained, but also the documentation.