Introduction

- Software maintenance:
  - any modifications to a software product after it has been delivered to the customer.
- Software maintenance is an important activity for many organizations.

Introduction

- Maintenance is inevitable for almost any kind of product.
- Most products need maintenance:
  - due to wear and tear caused by use.
- Software products do not need maintenance on this count.

Introduction

- Many people think
  - only bad software products need maintenance.
- The opposite is true:
  - bad products are thrown away,
  - good products are maintained and used for a long time.

Introduction

- Software products need maintenance for three reasons:
  - corrective
  - adaptive
  - perfective

Corrective

- Corrective maintenance of a software product:
  - to correct bugs observed while the system is in use.
  - to enhance performance of the product.

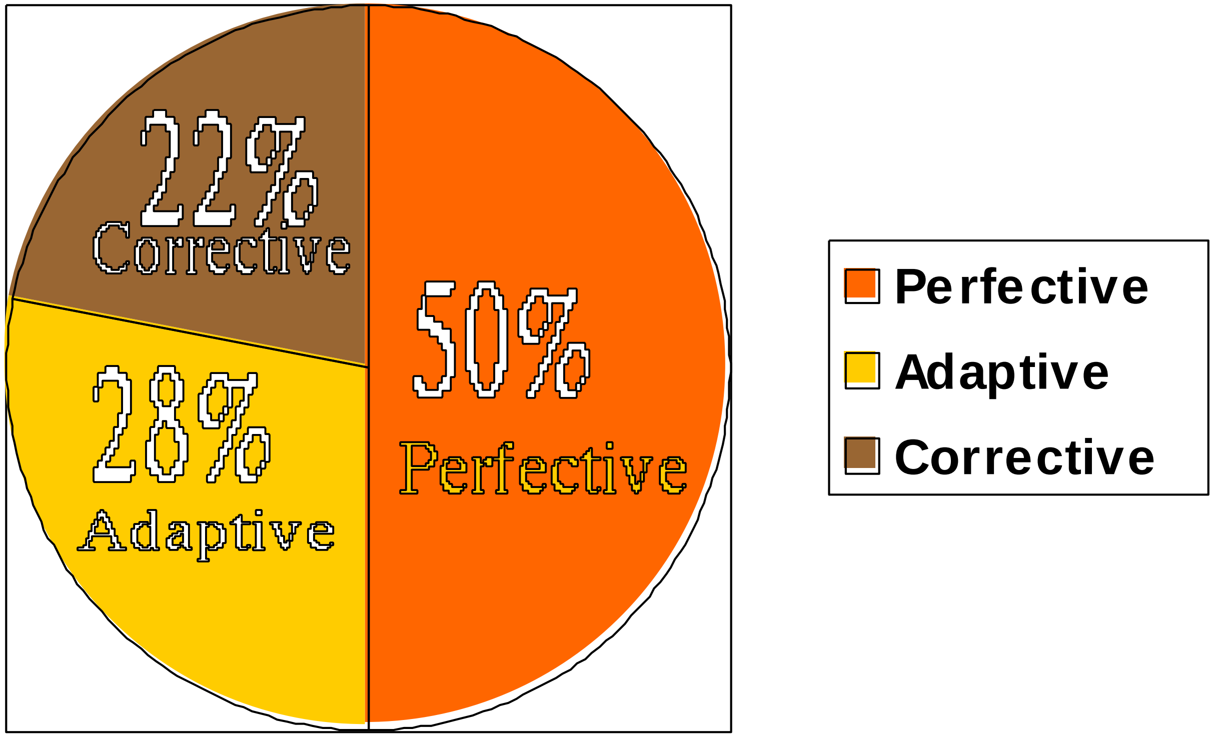
Adaptive

- A software product needs maintenance (porting) when customers:
  - need the product to run on new platforms,
    - or, on new operating systems,
  - need the product to interface with new hardware or software.

Perfective

- Perfective maintenance:
  - to support new features required by users.
  - to change some functionality of the system due to customer demands.

Maintenance Effort Distribution



Causes for maintenance

- Users want existing software to run on new platforms:
  - to run in new environments,
  - and/or with enhanced features.

Causes for maintenance

- Whenever other software it works with change:
  - maintenance is needed to cope up with the newer interface.
  - For instance, a software product may need maintenance when the operating system changes.

Laws of Maintenance

- ✓ There will always be a lot of old software needing maintenance.
- ✓ Good products are maintained, bad products are thrown away.

Laws of Maintenance

- Lehman's first Law:
- "Software products must change continuously, or become progressively less useful."

Laws of Maintenance

- Lehman's Second Law
- "When software is maintained, its structure degrades,
  - unless active efforts are made to avoid this phenomenon."

Laws of Maintenance

- Lehman's Third Law:
- "Over a program's life time,
  - its rate of development is approximately constant."

Legacy code--- Major maintenance problems

- Unstructured code (bad programs)
- Maintenance programmers have:
  - insufficient knowledge of the system or the application domain.
  - Software maintenance has a bad image.
- Documentation absent, out of date, or insufficient.

Insufficient knowledge

- Maintenance team is usually different from development team.
  - even after reading all documents
    - it is very difficult to understand why a thing was done in a certain way.
  - Also there is a limit to the rate at which a person can study documents
    - and extract relevant information

Bad image of maintenance?

- Maintainers are skilled not only in writing code:
  - proficient in understanding others' code
  - detecting problems, modifying the design, code, and documentation
  - working with end-users

Maintenance Nightmares

- Use of gotos
- Lengthy procedures
- Poor and inconsistent naming
- Poor module structure
- Weak cohesion and high coupling
- Deeply nested conditional statements
- Functions having side effects

How to do better maintenance?

- Program understanding
- Reverse engineering
- Design recovery
- Reengineering
- Maintenance process models

Maintenance activities

- Two types of activities:
  - Productive activities:
    - modification of analysis, design, coding, etc.
  - Non-productive activities:
    - understanding system design, code, etc.

Software Reverse Engineering

- By analyzing a program code, recover from it:
  - the design and the requirements specification.

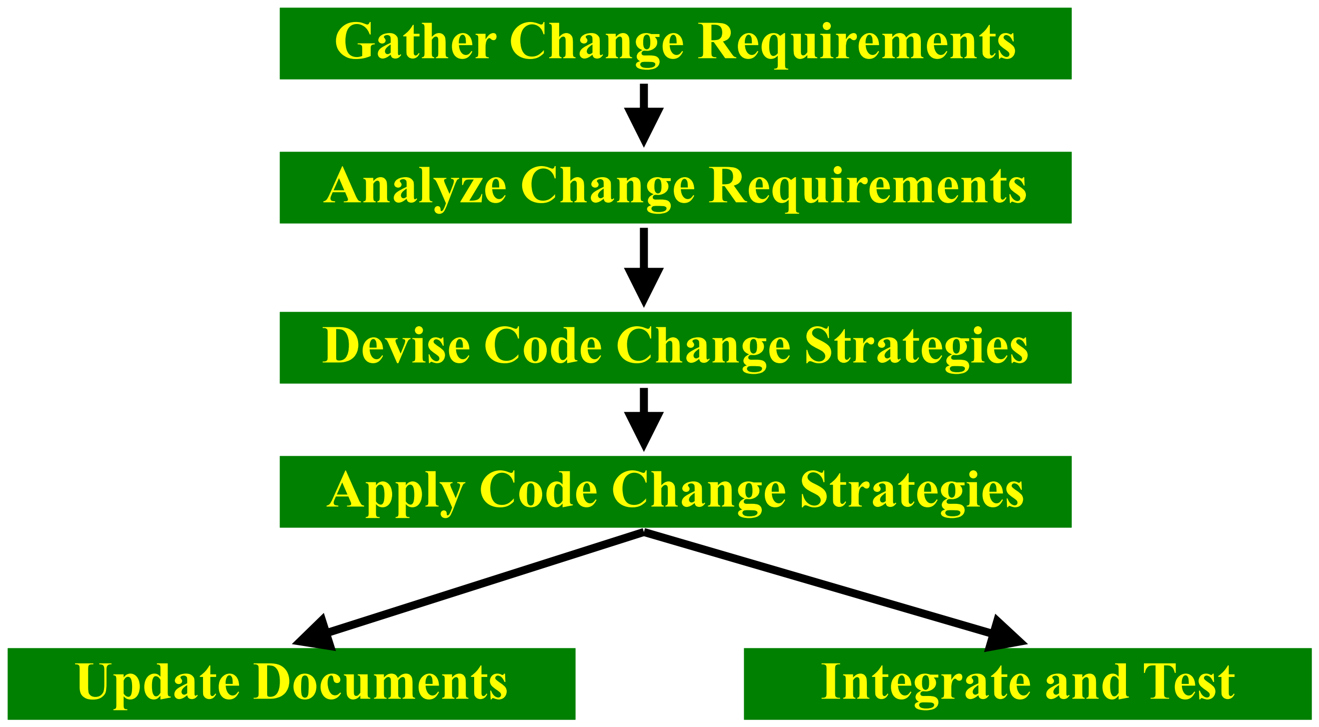
Software Reverse Engineering

- In order to extract the design:
  - fully understand the code.
- Automatic tools can be used to help derive:
  - data flow and control flow diagrams from the code.

Software Maintenance Process Model - 1

- When the required changes are small and simple:
  - the code can be directly modified
  - changes reflected in all relevant documents.
  - more elaborate activities are required when required changes are not trivial.

Software Maintenance Process Model - 1


Gather Change Requirements
Analyze Change Requirements
Devise Code Change Strategies
Apply Code Change Strategies
Update Documents
Integrate and Test

Software Maintenance Process Model -2

- For complex maintenance projects, software reengineering needed:
  - a reverse engineering cycle followed by a forward engineering cycle.
  - with as much reuse as possible from existing code and other documents.

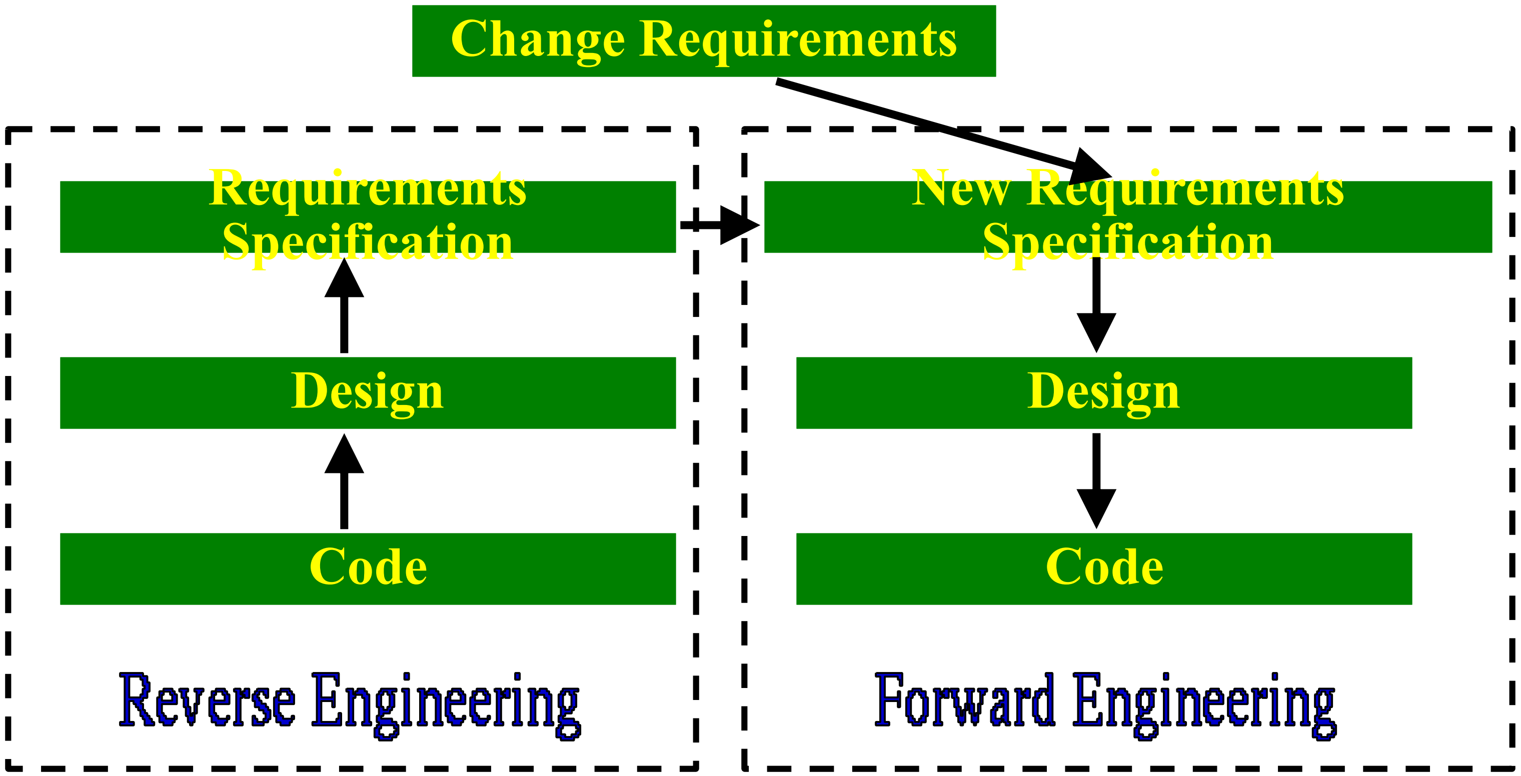
Maintenance Process Model 2

- Preferable when:
  - amount of rework is significant
  - software has poor structure.
- Can be represented by a reverse engineering cycle:
  - followed by a forward engineering cycle.

Software reengineering

- Forward engineering is carried out to produce the new code.
- During design, module specification, and coding:
  - substantial reuse is made from the reverse engineered products.

Process model for Software reengineering



Software reengineering

- Advantages of reengineering:
  - produces better design than the original product,
  - produces required documents,
  - often results in higher efficiency.

Software reengineering

- Reengineering is preferable when:
  - amount of rework is high,
  - product exhibits high failure rate.
  - product difficult to understand.