Maintenance & Re-engineering

Chapter 29

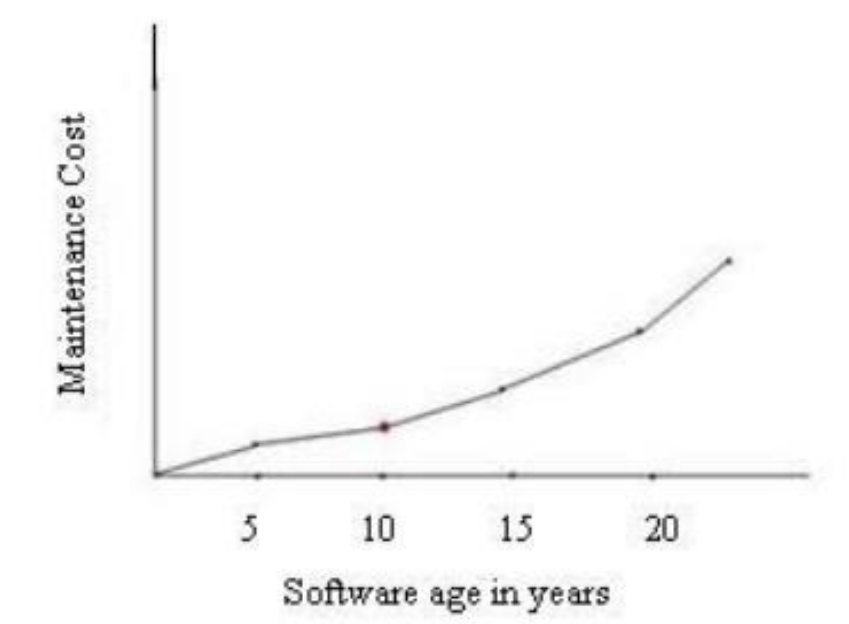
Roger Pressman 7th Edition

Software Maintenance

- Software is released to end-users, and
  - within days, bug reports filter back to the software engineering organization.
  - within weeks, one class of users indicates that the software must be changed so that it can accommodate the special needs of their environment.
  - within months, another corporate group who wanted nothing to do with the software when it was released, now recognizes that it may provide them with unexpected benefit. They'll need a few enhancements to make it work in their world.
- All of this work is *software maintenance.*

Software Maintainence

- Software maintenance starts after delivery of the software system. It goes on increasing with the increasing age of software as depicted in the following figure.



Types of Maintainence

- Software maintenance is of following four types
- 1. Corrective Maintenance: As its name implies, activities of correcting bugs in the software are included in this type of maintenance. It is for making the software system to conform to the real situation.
- 2. Adaptive Maintenance: It deals with making the software adjustable to the changed environment.
- 3. Preventive Maintenance: Modification of the software to detect and correct hidden faults (bugs) before becoming active.
- 4. Perfective Maintenance: It is modification of the software for better performance, maintenance and reliability. The activities related to updating the software are included in this type of maintenance.

Re-engineering

Software re-engineering is vital to restore and reuse the things inherent in the existing software, put the cost of software maintenance to the lowest in the control and establish a basis for the development of software in the future.

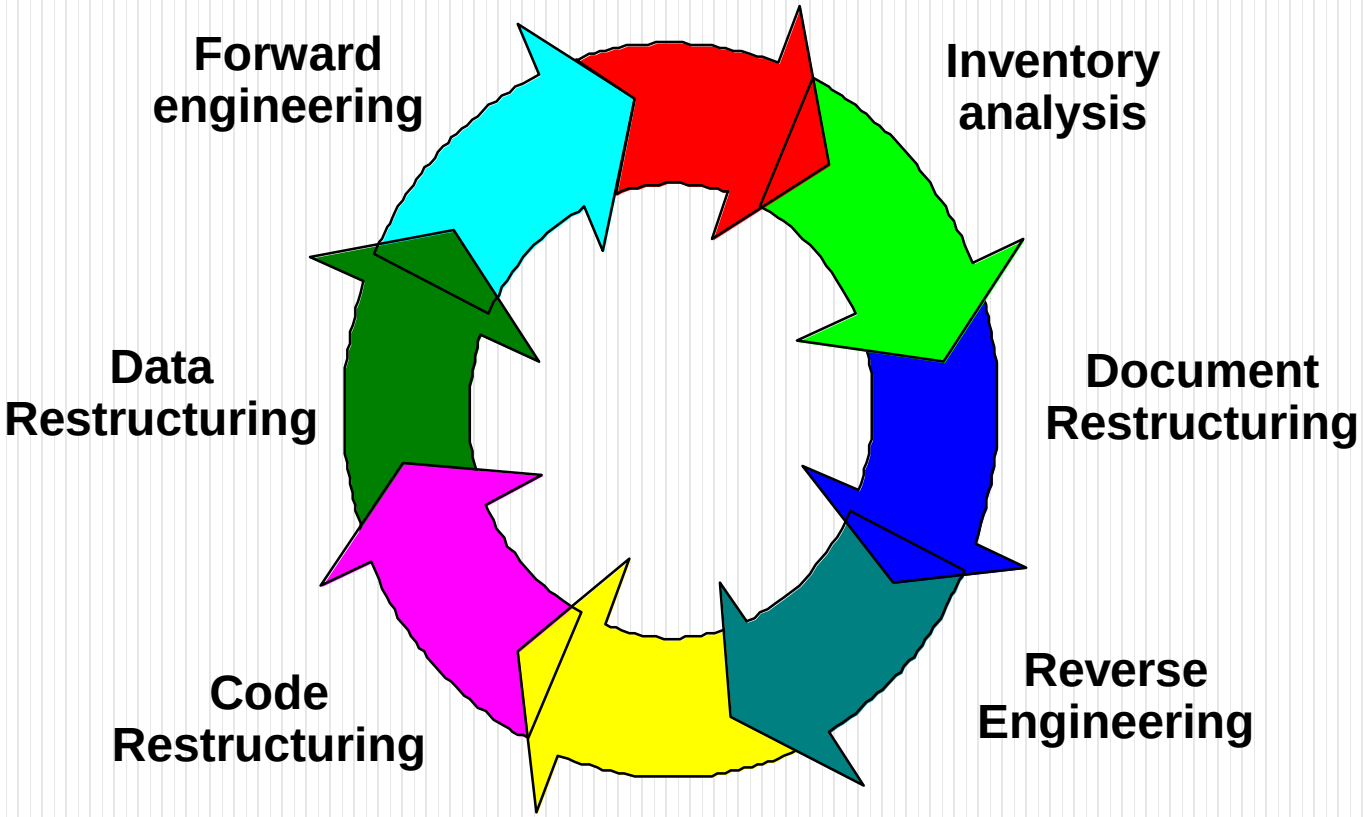
- When the system changes affect a subsystem and the subsystem that needs to be redesigned.
- When hardware and software support has become outdated and obsolete etc.

- Reengineering is the analysis of existing software system and modifying it to constitute into a new form.

Maintainence vs Re-engineering

- Maintenance and reengineering terms are closely coupled with each other.
- Maintenance and reengineering are two different areas in software engineering.
- Maintenance is for running the system till the age of the system where as the reengineering make the system new to work for another life span.
- Reengineering has more scope in the world of software than in the world of hard ware objects. Software systems and software objects do not wear and tear out like hardware objects in the real world.
- Maintenance is close to repair/mend where as reengineering is very close to new development.
- When maintenance cost is not feasible, we go for reengineering the software system. Reengineering makes the software system new.

Software Re-engineering



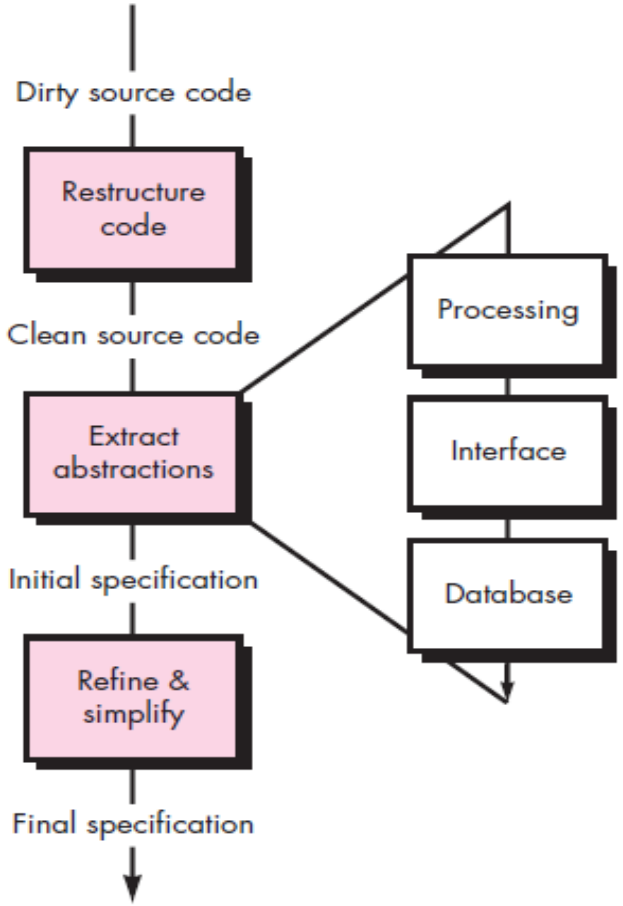
Inventory Analysis

- build a table that contains all the details of the application.
- establish a list of criteria, e.g.,
  - name of the application
  - year it was originally created
  - number of substantive changes made to it
  - total effort applied to make these changes
  - date of last substantive change
  - effort applied to make the last change
  - system(s) in which it resides
  - applications to which it interfaces, ...
- analyze and prioritize to select candidates for reengineering

Document Restructuring

- Weak documentation is the trademark of many legacy systems.
- But what do we do about it? What are our options?
- Options …
  - *Creating documentation is far too time consuming.* If the system works, we'll live with what we have. In some cases, this is the correct approach.
  - *Documentation must be updated, but we have limited resources.* We'll use a "document when touched" approach. It may not be necessary to fully re-document an application.
  - *The system is business critical and must be fully re-documented.* Even in this case, an intelligent approach is to pare documentation to an essential minimum.

Reverse Engineering



- Process of design recovery - analyzing a program in an effort to create a representation of the program at some abstraction level higher than source code.
- Reverse engineering does not change the system or create a new system; it is an inspection process does not change the overall functionality of the system.

Code Restructuring

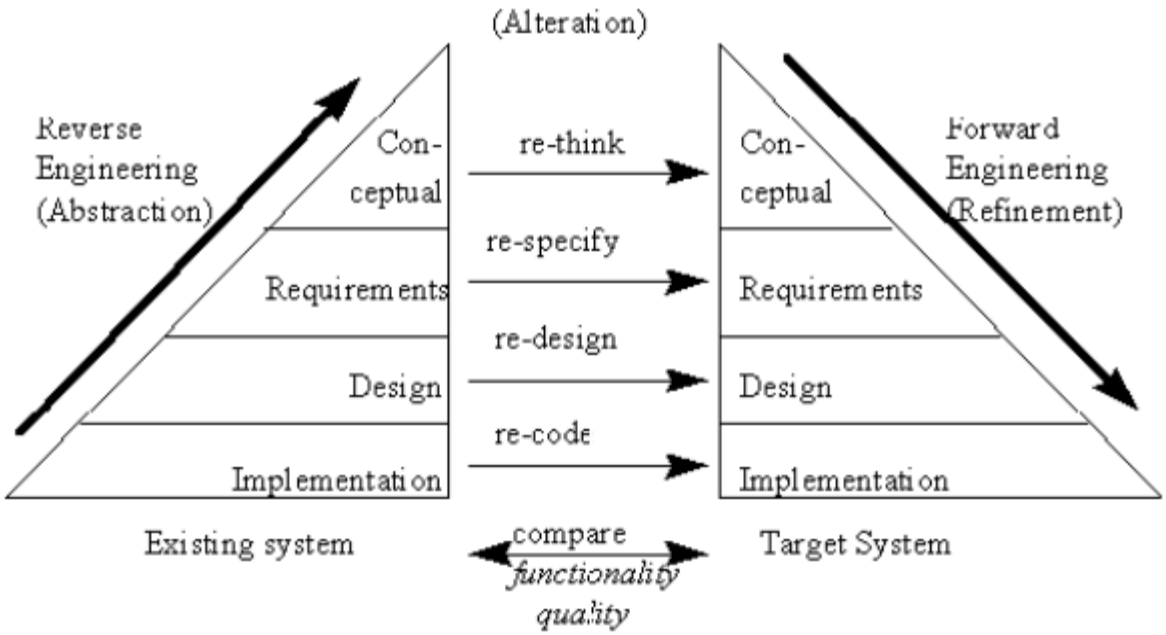
- Source code is analyzed using a restructuring tool.
- Poorly design code segments are redesigned
- Violations of structured programming constructs are noted and code is then restructured (this can be done automatically)
- The resultant restructured code is reviewed and tested to ensure that no anomalies have been introduced
- Internal code documentation is updated.

Data Restructuring

- Unlike code restructuring, which occurs at a relatively low level of abstraction, data structuring is a full-scale reengineering activity
- In most cases, data restructuring begins with a reverse engineering activity.
  - Current data architecture is dissected and necessary data models are defined.
  - Data objects and attributes are identified, and existing data structures are reviewed for quality.
  - When data structure is weak (e.g., flat files are currently implemented, when a relational approach would greatly simplify processing), the data are reengineered.
- Because data architecture has a strong influence on program architecture and the algorithms that populate it, changes to the data will invariably result in either architectural or code-level changes.

Forward Engineering

- Also called reclamation or renovation, recovers design information from existing source code and uses this information to reconstitute the existing system to improve its overall quality and/or performance.



Economics of Re-engineering

- A cost/benefit analysis model for reengineering. Nine parameters are defined:
  - P_1 = current annual maintenance cost for an application.
  - P_2 = current annual operation cost for an application.
  - P_3 = current annual business value of an application.
  - P_4 = predicted annual maintenance cost after reengineering.
  - P_5 = predicted annual operations cost after reengineering.
  - P_6 = predicted annual business value after reengineering.
  - P_7 = estimated reengineering costs.
  - P_8 = estimated reengineering calendar time.
  - P_9 = reengineering risk factor (P_9 = 1.0 is nominal).
  - L = expected life of the system.

Economics of Re-engineering

- The cost associated with continuing maintenance of a candidate application (i.e., reengineering is not performed) can be defined as

$$C_{maint} = [P_3 - (P_1 + P_2)] \times L$$

- The costs associated with reengineering are defined using the following relationship:

$$C_{reeng} = [P_6 - (P_4 + P_5) \times (L - P_8) - (P_7 \times P_9)]$$

- Using the costs presented in equations above, the overall benefit of reengineering can be computed as

$$\text{cost benefit} = C_{reeng} - C_{maint}$$

Maintainence vs. Re-engineering

Software issues	Maintenance	Reengineering
When software is delivered	yes	No
When software is young	yes	no
When software is old	No	Yes
Cost comparison (software is not old)	Less	more
Cost comparison (software is old)	More	Less
Cost comparison (Transitional state)	equal	equal
Software age	Age increases	New life span started
Subject Area	Software engineering	Software engineering
Architecture	No change	change
Business process	No change	change
Addition of attributes	yes	yes
Additions of objects	yes	yes
System change	renovate	fresh (New)
Type of activity	Repair	development
Man power	Skilled	Highly skilled
Scope	equal	equal
Origin	Hardware objects	Hardware objects
Software managers (Interest)	More	Less
functioning	More	less
No. faulty objects less than half	yes	no
No. faulty objects greater than half	no	yes

THANK YOU!!

Any Questions???