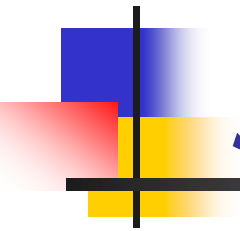
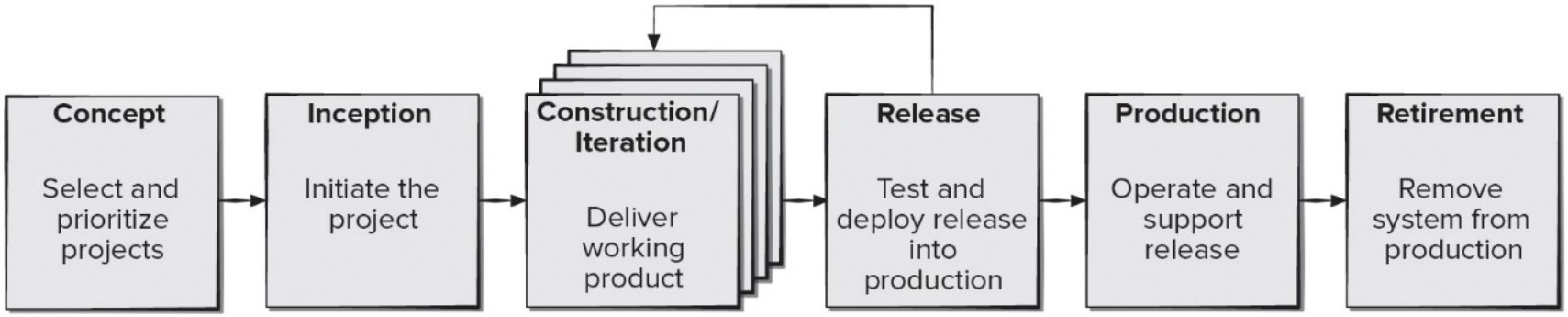


COMP 354: Introduction to Software Engineering

Software Support

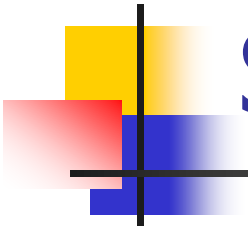
Based on Chapter 26 of the textbook

Prototype Evolution Process Model



Lehman's Laws of Software Evolution

- Law of continuing change (1974). Software implemented in a real-world will evolve over time and must be continually adapted.
- Law of increasing complexity (1974). As a system evolves its complexity increases unless work is done to maintain or reduce it.
- Law of conservation of familiarity (1980). As a system evolves all associated with it (all stakeholders) must maintain knowledge of its content and behavior to achieve satisfactory evolution. Excessive growth diminishes that knowledge.
- Law of continuing growth (1980). The functional content of systems must be continually increased to maintain user satisfaction.
- Law of declining quality (1996). The quality of systems will appear to be declining unless they are rigorously maintained and adapted to operational environment changes.

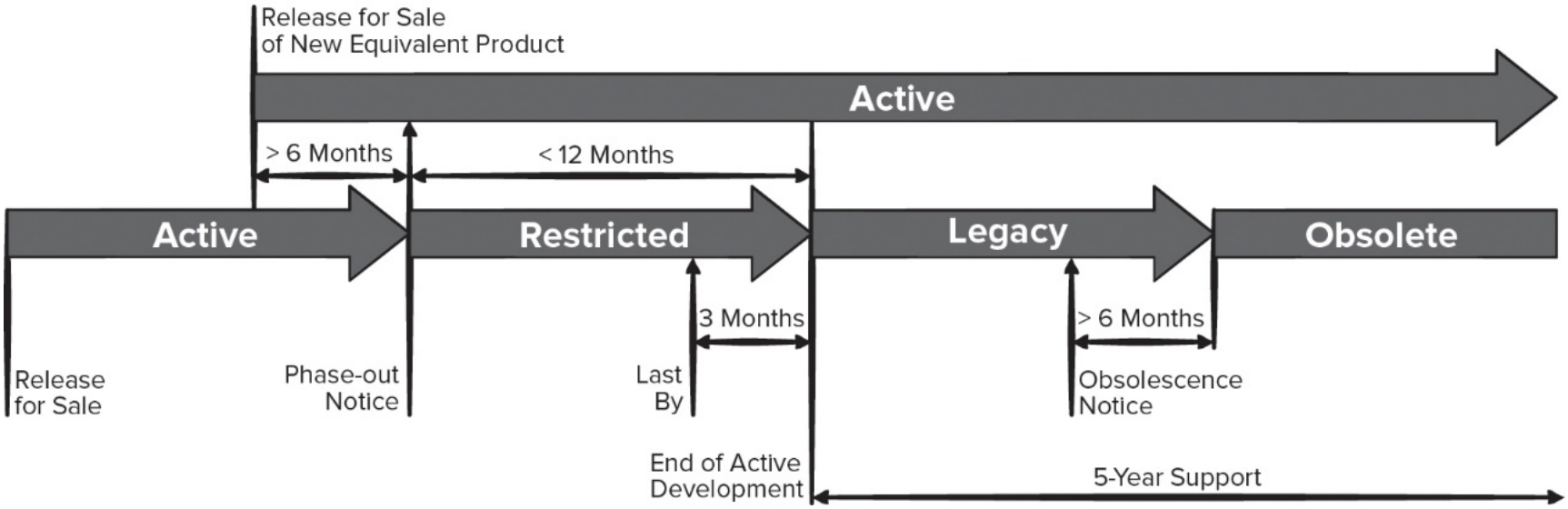


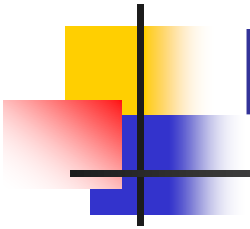
Software Support

Software support can be considered an umbrella activity that includes:

- Change management.
- Proactive risk management.
- Process management.
- Configuration management.
- Quality assurance.
- Release management.

Software Release and Retirement



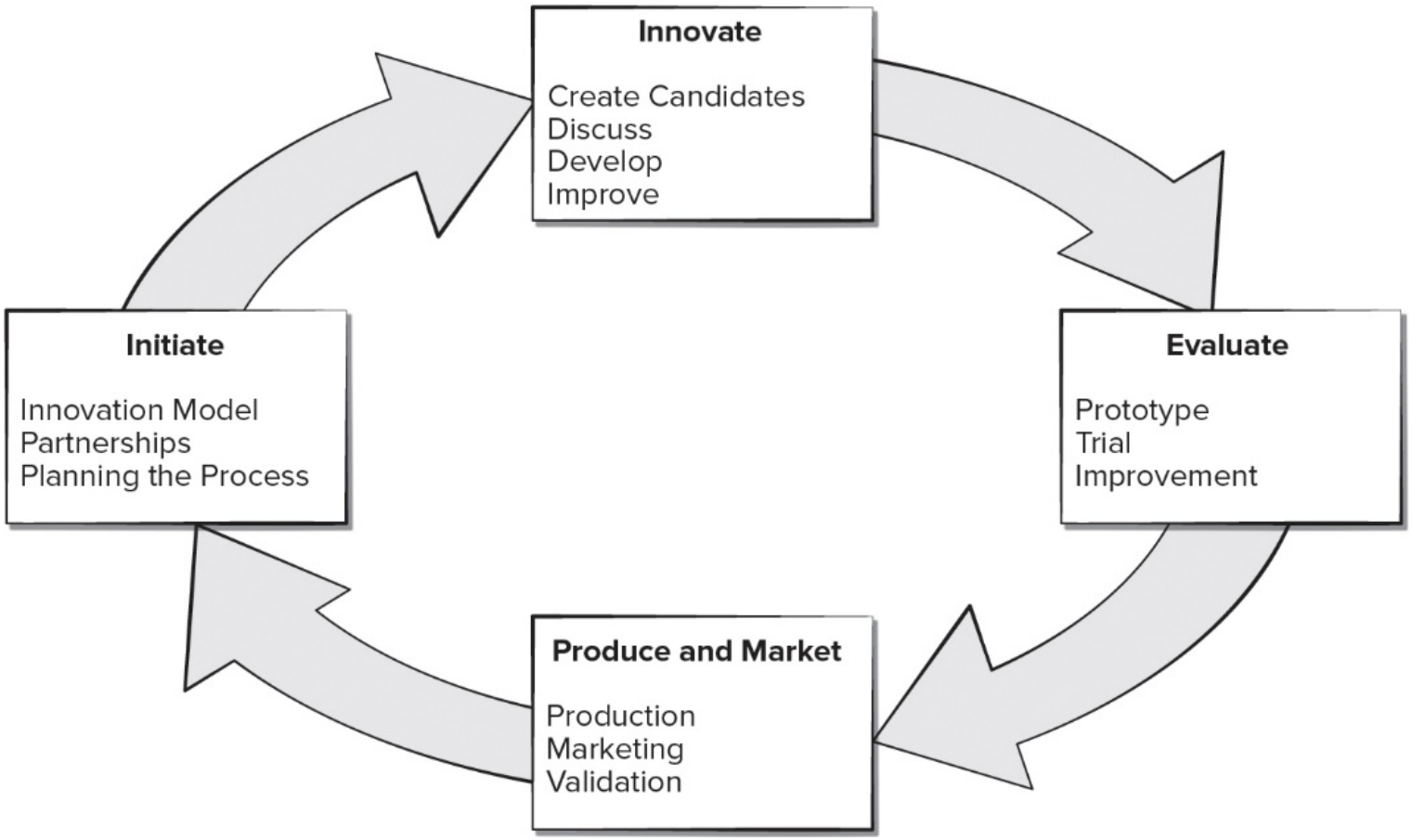


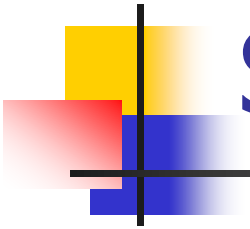
Release Management

Release management - process that brings high-quality code from developer's workspace to the end user includes:

- Code change integration.
- Continuous integration.
- Build system specifications.
- Infrastructure-as-code.
- Deployment and release.
- Retirement.

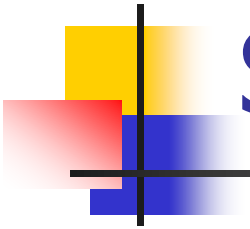
Iterative Software Support Model





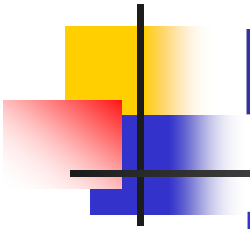
Software Supportability

- Capability of supporting software over its whole lifetime.
- Implies satisfying all necessary requirements and also the provision of resources, support infrastructure, additional software, facilities, and manpower needed to ensure software is capable of performing its functions.
- Software should contain facilities to assist support personnel when a defect is encountered in the operational environment.
- Support personnel should have access to a database containing records of all defects that have already been encountered—their characteristics, cause, and cure.



Software Maintenance

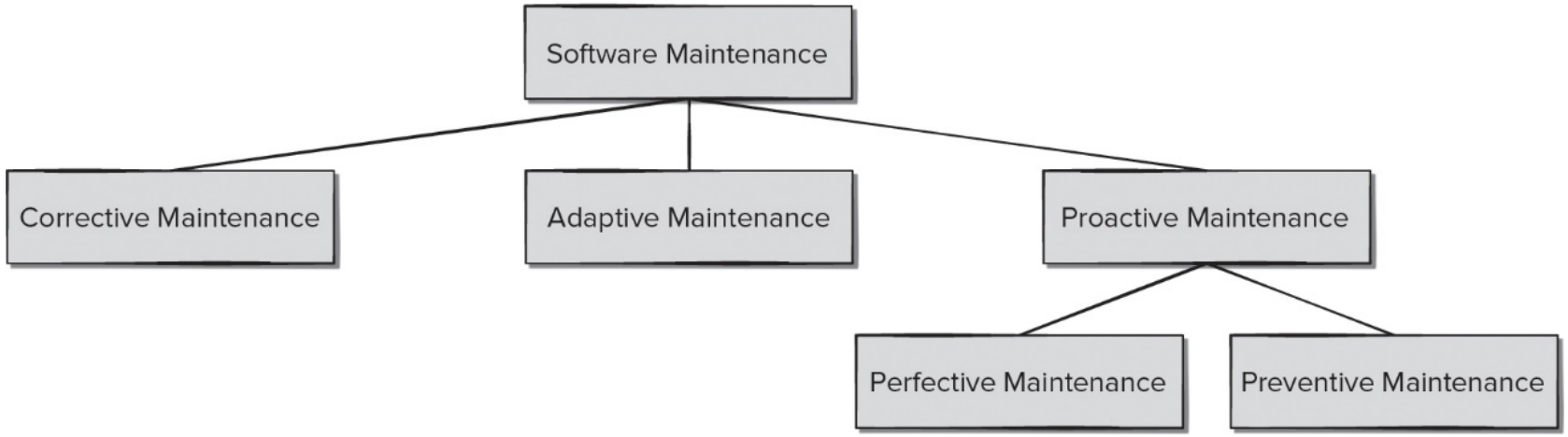
- Software is released to end-users, and:
  - Within days, bug reports filter back to the software engineering organization.
  - Within weeks, one class of users indicates that the software must be changed so that it can accommodate the special needs of their environment.
  - Within months, another corporate group who wanted nothing to do with the software when it was released, now recognizes that it may provide them with benefits and want few enhancements to make it work better for their needs.
- All of this work is software maintenance

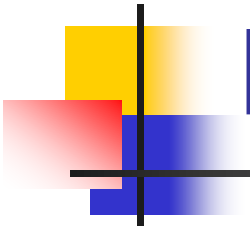


Maintainable Software

- Maintainable software exhibits effective modularity.
- It makes use of design patterns that allow ease of understanding.
- It has been constructed using well-defined coding standards, leading to understandable source code.
- It has undergone quality assurance techniques that uncovered maintenance problems before release.
- It was created by software engineers who recognize that they may not be around when changes must be made.
- The design and implementation of the software must "assist" the person who is making the change.

Maintenance Types

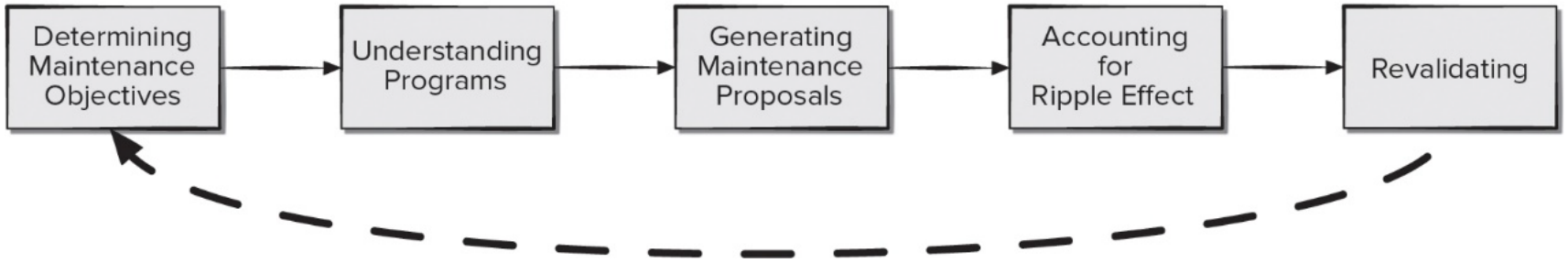


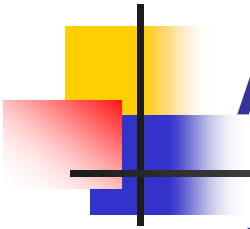


Maintenance and Support

- Reverse engineering - process of analyzing a software system to create representations of the system at a higher level of abstraction. Often used to rediscover and redocument system design elements prior to modifying the system source code.
- Refactoring - process of changing a software system to improves its internal structure without altering its external behavior. Often used to improve the quality of a software product and make it easier to understand and maintain.
- Reengineering (evolution) - process of taking an existing software system and generating a new system that has the same quality as software created using modern software engineering practices.

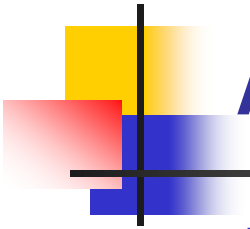
Maintenance Tasks





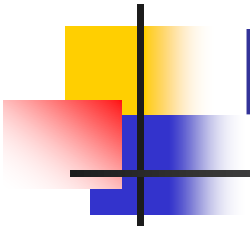
Agile Maintenance

- Use sprints to organize the maintenance work. Balance the goal of keeping customers happy with technical needs of the developers.
- Allow urgent customer requests to interrupt scheduled maintenance sprints, make time for them during sprint planning.
- Facilitate team learning by ensuring that more experienced developers are able to mentor less experienced team members.
- Allow multiple team members to accept customer requests as they coordinate their processing with maintenance team members.



Agile Maintenance

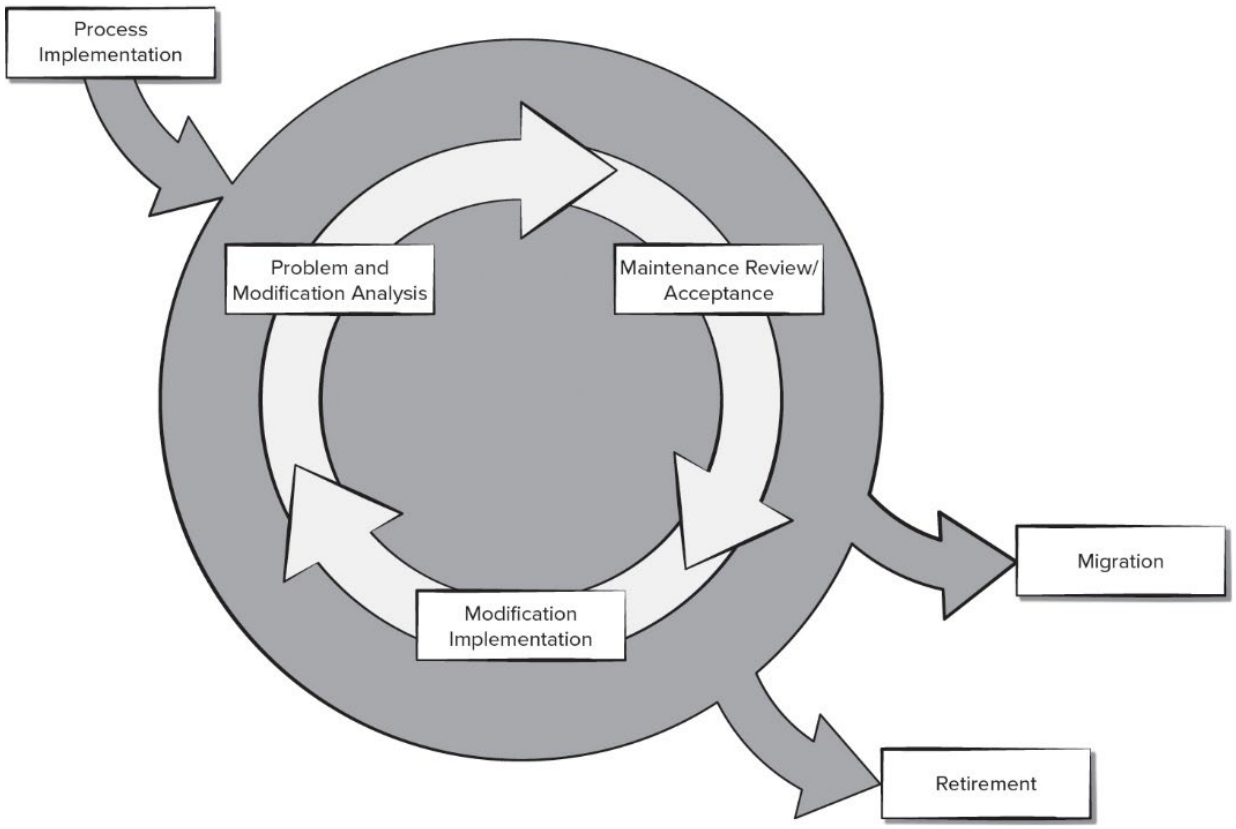
- Balance the use of written documentation with face-to-face communication to ensure planning meeting time is used wisely.
- Write informal use cases to supplement documentation being used for communications with stakeholders.
- Have developers test each other's work (both defect repairs and new feature implementations).
- Make sure developers are empowered to share knowledge with one another. Motivates people to improve the skills and knowledge.
- Keep planning meetings short, frequent, and focused.



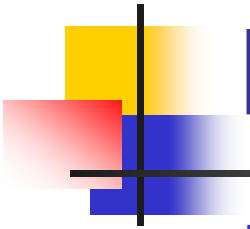
Reverse Engineering

- Reverse Engineering to Understand Data - first reengineering task often begins by constructing UML class diagram.
- Reverse Engineering of Internal Data Structures - focuses on the definition of object classes.
- Reverse Engineering of Database Structure - reengineering one database schema into another requires an understanding of existing objects and their relationships.
- Reverse engineering to understand processing - attempts to understand procedural abstractions in source code.
- Reverse engineering to understand user interfaces - may need to be done as part of the maintenance task (for example, adding a GUI).

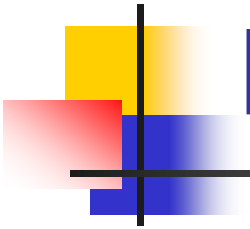
Proactive Software Support Model



Software Analytics and Proactive Maintenance

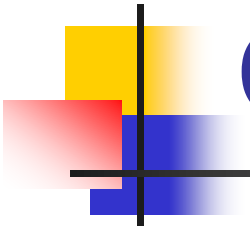


- Be sure you are using analytics to identify meaningful development problems, or you will get no buy-in from the software engineers.
- The analytics must make use of application domain knowledge to be useful to developers (this implies the use of experts to validate the analytics).
- Developing analytics requires iterative and timely feedback from the intended users.
- Make sure the analytics are scalable to larger problems and customizable to incorporate new discoveries made over time.
- Evaluation criteria used needs to be correlated to real software engineering practices.



Role of Social Media

- Many online stores allow users to provide feedback on the apps by posting ratings or comments.
- The feedback found in these reviews may contain usage scenarios, bug reports, or feature requests.
- Mining these reports can help developers identify potential maintenance and software evolution tasks.
- Many companies maintain Facebook pages or Twitter feeds to support their user communities.
- Some companies encourage product users to send program crash information for analysis by the support team members.
- Some companies use the questionable practice of tracking how and where products are used by customers without their knowledge.



Cost of Support

Nine parameters are defined:

- P1 = current annual maintenance cost for an application.
- P2 = current annual operation cost for an application.
- P3 = current annual business value of an application.
- P4 = predicted annual maintenance cost after reengineering.
- P5 = predicted annual operations cost after reengineering.
- P6 = predicted annual business value after reengineering.
- P7 = estimated reengineering costs.
- P8 = estimated reengineering calendar time.
- P9 = reengineering risk factor (P9 = 1.0 is nominal).
- L = expected life of the system.

Cost of Support

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

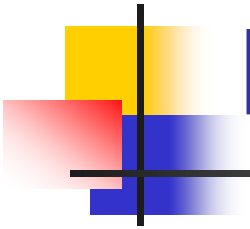
$$C_{maint} = [P_3 - (P_1 + P_2)] \times L$$

- 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:

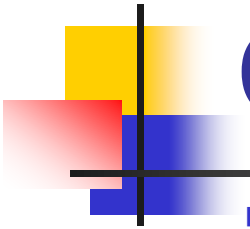
$$\text{Cost benefit} = C_{reeng} - C_{maint}$$



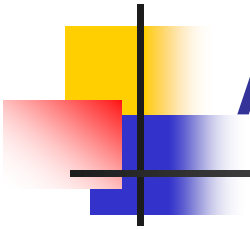
Data Refactoring

- Data refactoring should be preceded by source code analysis.
- Data analysis requires the evaluation of programming language statements containing data definitions, file descriptions, I/O, and interface descriptions are evaluated.
- Data redesign involves a data record standardization which clarifies data definitions to achieve consistency among data item names or physical record formats, data name rationalization ensures that data naming conventions conform to local standards.
- When refactoring moves beyond standardization and rationalization, physical modifications to existing data structures are made to make the data design more effective.
- This may mean a translation from one file format to another, or in some cases, translation from one type of database to another.

Code Refactoring



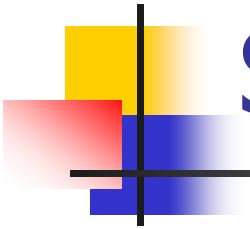
- Code refactoring is performed to yield a design that produces the same function but with higher quality than the original program.
- The objective is to take "spaghetti-bowl" code and derive a design that conforms to the quality factors defined for the product.
- Another approach relies on the use of anti-patterns to identify bad code design practices and suggest possible solutions.
- Code refactoring can alleviate immediate problems associated with debugging or small software changes; it is not reengineering.
- Real benefit of code refactoring is achieved only when data and architecture are refactored as well.



Architectural Refactoring

Architectural refactoring is one of the design trade-off options for dealing with a messy program:

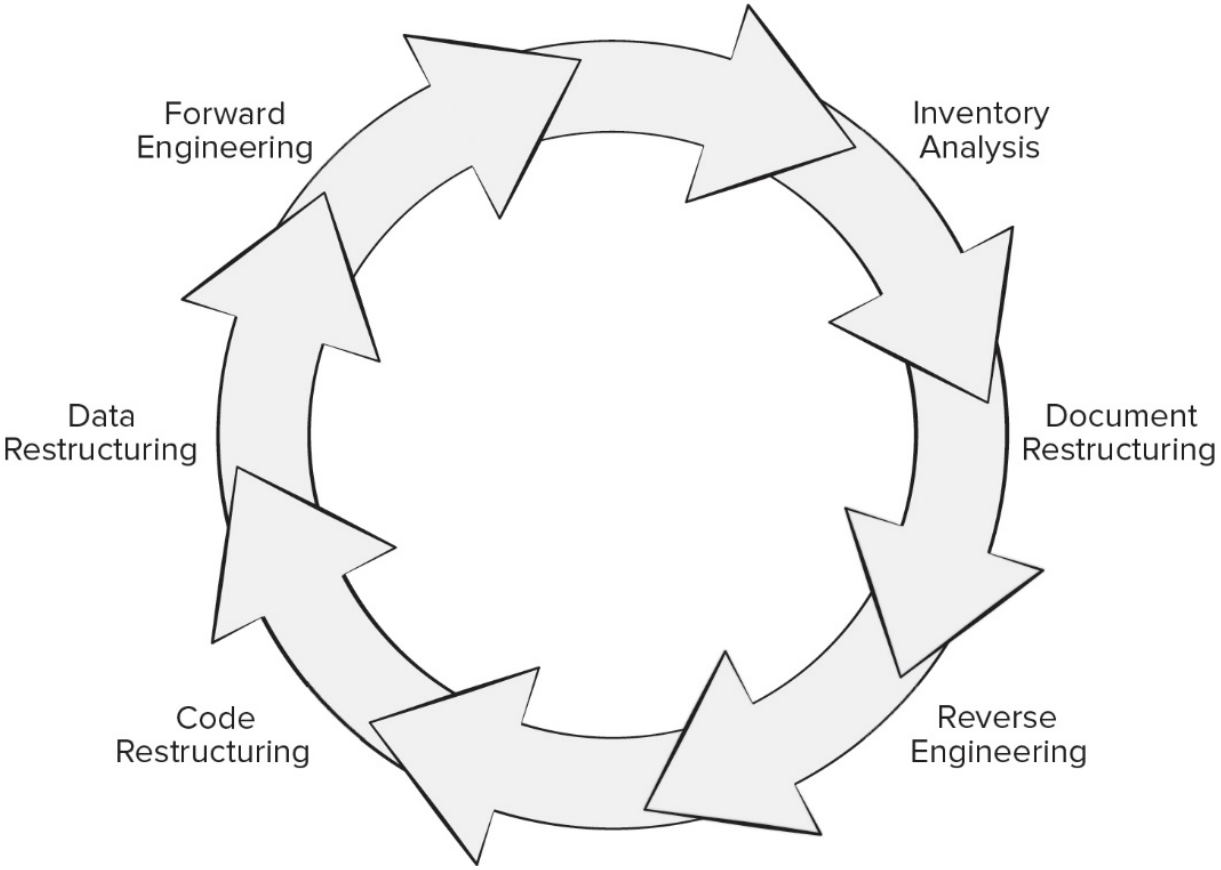
- You can struggle through modification after modification, fighting the ad hoc design and tangled source code to implement the necessary changes.
- You can attempt to understand the broader inner workings of the program to make modifications more effectively.
- You can revise (redesign, recode, and test) those portions of the software that require modification, applying a meaningful software engineering approach to all revised segments.
- You can completely redo (redesign, recode, and test) the program, using reengineering tools to understand the current design.

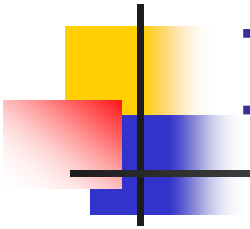


Software Evolution

- The cost to maintain one line of source code may be 20 to 40 times the cost of initial development of that line.
- Redesign of the software architecture (program and/or data structure), using modern design concepts, can greatly facilitate future maintenance.
- Because a prototype of the software already exists, development productivity should be much higher than average.
- Tools for reengineering will automate some parts of the job.
- Users now have experience with the software so new requirements and change direction can be ascertained with greater ease.
- A complete software configuration (documents, programs and data) when the evolutionary preventive maintenance is done.

Reengineering Process Model

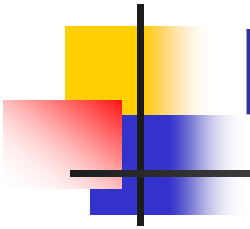




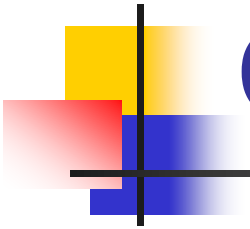
Inventory Analysis

- Every software organization should have an inventory of all applications.
- The inventory can be a spreadsheet model containing information that provides a detailed description (For example size, age, business criticality) of every active application.
- Sorting this information according to business criticality, longevity, current maintainability, and other criteria, helps to identify candidates for reengineering.
- Resources can then be allocated to candidate applications for reengineering work.
- The inventory should be revisited on a regular basis.

Document Restructuring

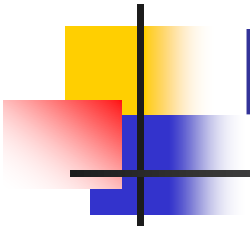


- Weak documentation is the trademark of many legacy systems.
- In some cases, creating documentation when none exists is simply too costly.
- In other cases, some documentation must be created, but only when changes are made.
- If a modification occurs, document it - try to reign in technical debt. There are situations in which a critical system must be fully documented, but documents to an essential minimum.



Code Restructuring

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



Data Restructuring

- Data refactoring is a full-scale reengineering activity.
- The current data architecture is analyzed 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 (for example, flat files are currently implemented, when a relational approach would greatly simplify processing), the data are reengineered.
- Data architecture has a strong influence on program architecture and the algorithms that populate it, changes to the data result in architectural or code-level changes.

Forward Engineering

- In an ideal world, applications would be rebuilt using an automated reengineering engine.
- Forward engineering recovers design information from existing software and uses this information to alter or reconstitute the existing system to improve its overall quality.
- Reengineered software re-creates the function of the existing system and adds new functions and/or improves overall performance.
- Forward engineering does not simply create a modern equivalent of an older program - the redeveloped program extends the capabilities of the older application.