Software Engineering-BSCE-301L

Module 6:

Software Evolution

Dr . Saurabh Agrawal

Faculty Id: 20165

School of Computer Science and Engineering

VIT, Vellore-632014

Tamil Nadu, India

Outline

- Software Maintenance
- Types of Maintenance
- Software Configuration Management
- Overview – SCM Tools
- Re-Engineering
- Reverse Engineering
- Software Reuse

Software Maintenance

- Software Maintenance refers to the process of modifying and updating a software system after it has been delivered to the customer.
- It is a critical part of the software development life cycle (SDLC) and is necessary to ensure that the software continues to meet the needs of the users over time.

Software Maintenance

❑**What is Software Maintenance? :** Software maintenance is a continuous process that occurs throughout the entire life cycle of the software system.

❑The goal of software maintenance is to keep the software system working correctly, efficiently, and securely, and to ensure that it continues to meet the needs of the users.

❑This can include fixing bugs, adding new features, improving performance, or updating the software to work with new hardware or software systems.

❑It is also important to consider the cost and effort required for software maintenance when planning and developing a software system.

❑It is important to have a well-defined maintenance process in place, which includes testing and validation, version control, and communication with stakeholders.

❑It's important to note that software maintenance can be costly and complex, especially for large and complex systems. Therefore, the cost and effort of maintenance should be taken into account during the planning and development phases of a software project.

Software Maintenance

❑Key Aspects of Software Maintenance

1. **Bug Fixing:** The process of finding and fixing errors and problems in the software.
2. **Enhancements:** The process of adding new features or improving existing features to meet the evolving needs of the users.
3. **Performance Optimization:** The process of improving the speed, efficiency, and reliability of the software.
4. **Porting and Migration:** The process of adapting the software to run on new hardware or software platforms.
5. **Re-Engineering:** The process of improving the design and architecture of the software to make it more maintainable and scalable.
6. **Documentation:** The process of creating, updating, and maintaining the documentation for the software, including user manuals, technical specifications, and design documents.

Types of Software Maintenance

❑Types of Software Maintenance

1. **Corrective Maintenance:** This involves fixing errors and bugs in the software system.
2. **Patching:** It is an emergency fix implemented mainly due to pressure from management. Patching is done for corrective maintenance but it gives rise to unforeseen future errors due to lack of proper impact analysis.
3. **Adaptive Maintenance:** This involves modifying the software system to adapt it to changes in the environment, such as changes in hardware or software, government policies, and business rules.
4. **Perfective Maintenance:** This involves improving functionality, performance, and reliability, and restructuring the software system to improve changeability.
5. **Preventive Maintenance:** This involves taking measures to prevent future problems, such as optimization, updating documentation, reviewing and testing the system, and implementing preventive measures such as backups.

Types of Software Maintenance

- Maintenance can be categorized into proactive and reactive types.
- Proactive maintenance involves taking preventive measures to avoid problems from occurring, while reactive maintenance involves addressing problems that have already occurred.
- Maintenance can be performed by different stakeholders, including the original development team, an in-house maintenance team, or a third-party maintenance provider.
- Maintenance activities can be planned or unplanned.
- Planned activities include regular maintenance tasks that are scheduled in advance, such as updates and backups.
- Unplanned activities are reactive and are triggered by unexpected events, such as system crashes or security breaches.
- Software maintenance can involve modifying the software code, as well as its documentation, user manuals, and training materials.
- This ensures that the software is up-to-date and continues to meet the needs of its users.

Types of Software Maintenance

❑**Need for Maintenance :** Software Maintenance must be performed in order to:

1. Correct faults.
2. Improve the design.
3. Implement enhancements.
4. Interface with other systems.
5. Accommodate programs so that different hardware, software, system features, and telecommunications facilities can be used.
6. Migrate legacy software.
7. Retire software.
8. Requirement of user changes.
9. Run the code fast

Types of Software Maintenance

❑**Categories of Software Maintenance :** Maintenance can be divided into the following categories.

1. **Corrective Maintenance:** Corrective maintenance of a software product may be essential either to rectify some bugs observed while the system is in use, or to enhance the performance of the system.
2. **Adaptive Maintenance:** This includes modifications and updations when the customers need the product to run on new platforms, on new operating systems, or when they need the product to interface with new hardware and software.
3. **Perfective Maintenance:** A software product needs maintenance to support the new features that the users want or to change different types of functionalities of the system according to the customer's demands.
4. **Preventive Maintenance:** This type of maintenance includes modifications and updations to prevent future problems with the software. It goals to attend to problems, which are not significant at this moment but may cause serious issues in the future.

Software Configuration Management

❑The output of the software process is information that may be divided into three broad categories:

1. Computer programs (both source level and executable forms)
2. Work products that describe the computer programs (targeted at various stakeholders)
3. Data or content (contained within the program or external to it).

❑**The items that comprise all information produced as part of the software process are collectively called a software configuration.**

❑As software engineering work progresses, a hierarchy of software configuration items (SCIs)—a named element of information that can be as small as a single UML diagram or as large as the complete design document—is created.

❑If each SCI simply led to other SCIs, little confusion would result.

❑Unfortunately, another variable enters the process—change.

❑Change may occur at any time, for any reason. In fact, the First Law of System Engineering [Ber80] states: “No matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle.”

Software Configuration Management

❑What is the origin of these changes? The answer to this question is as varied as the changes themselves.

❑However, there are **four fundamental sources of change**:

1. New business or market conditions dictate changes in product requirements or business rules.
2. New stakeholder needs demand modification of data produced by information systems, functionality delivered by products, or services delivered by a computer-based system.
3. Reorganization or business growth/downsizing causes changes in project priorities or software engineering team structure.
4. Budgetary or scheduling constraints cause a redefinition of the system or product.

Software Configuration Management

- Software configuration management is a set of activities that have been developed to manage change throughout the life cycle of computer software.
- SCM can be viewed as a software quality assurance activity that is applied throughout the software process.
- In the sections that follow, I describe major SCM tasks and important concepts that can help you to manage change.

Software Configuration Management

❑Elements of a Configuration Management System: Susan Dart [Dar01] identifies four important elements that should exist when a configuration management system is developed:

1. **Component elements**—a set of tools coupled within a file management system (e.g., a database) that enables access to and management of each software configuration item.
2. **Process elements**—a collection of actions and tasks that define an effective approach to change management (and related activities) for all constituencies involved in the management, engineering, and use of computer software.
3. **Construction elements**—a set of tools that automate the construction of software by ensuring that the proper set of validated components (i.e., the correct version) have been assembled.
4. **Human elements**—a set of tools and process features (encompassing other CM elements) used by the software team to implement effective SCM.

Software Configuration Management

- These elements are not mutually exclusive.
- For example, component elements work in conjunction with construction elements as the software process evolves.
- Process elements guide many human activities that are related to SCM and might therefore be considered human elements as well.

Software Configuration Management

❑Baselines

- ❑Change is a fact of life in software development.
- ❑Customers want to modify requirements.
- ❑Developers want to modify the technical approach.
- ❑Managers want to modify the project strategy.
- ❑Why all this modification?
- ❑The answer is really quite simple.

Software Configuration Management

❑Baselines

❑As time passes, all constituencies know more (about what they need, which approach would be best, and how to get it done and still make money).

❑This additional knowledge is the driving force behind most changes and leads to a statement of fact that is difficult for many software engineering practitioners to accept: Most changes are justified!

❑A baseline is a software configuration management concept that helps you to control change without seriously impeding justifiable change.

❑ The IEEE (IEEE Std. No. 610.12-1990) defines a baseline as: "A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures".

Software Configuration Management

❑Baselines

❑Before a software configuration item becomes a baseline, changes may be made quickly and informally.

❑However, once a baseline is established, changes can be made, but a specific, formal procedure must be applied to evaluate and verify each change.

❑In the context of software engineering, a baseline is a milestone in the development of software.

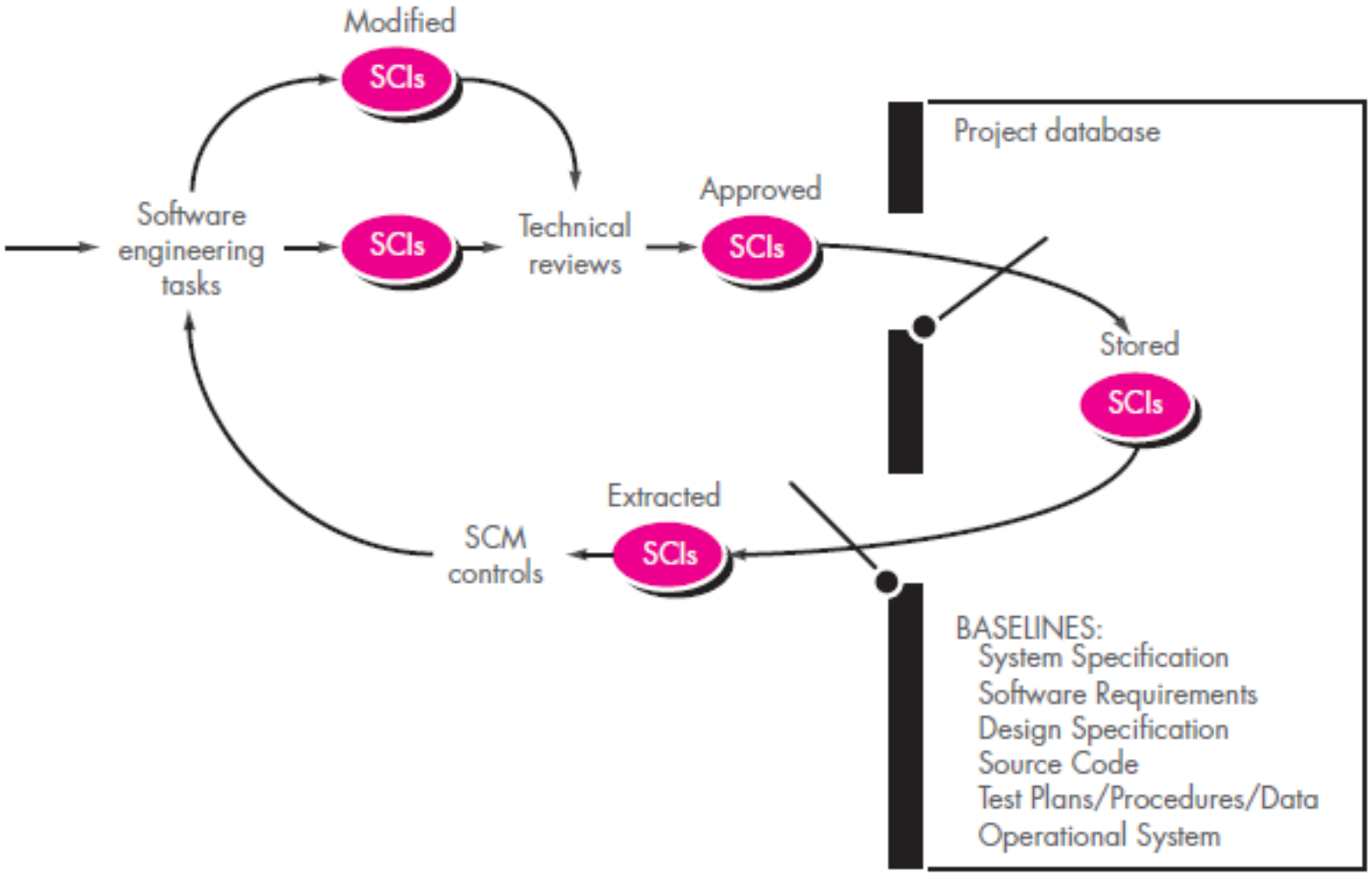
❑A baseline is marked by the delivery of one or more software configuration items that have been approved as a consequence of a technical review.

Software Configuration Management

❑Baselines

FIGURE 22.1

Baselined SCIs and the project database



Software Configuration Management

❑SCM Process:

❑The software configuration management process defines a series of tasks that have four primary objectives:

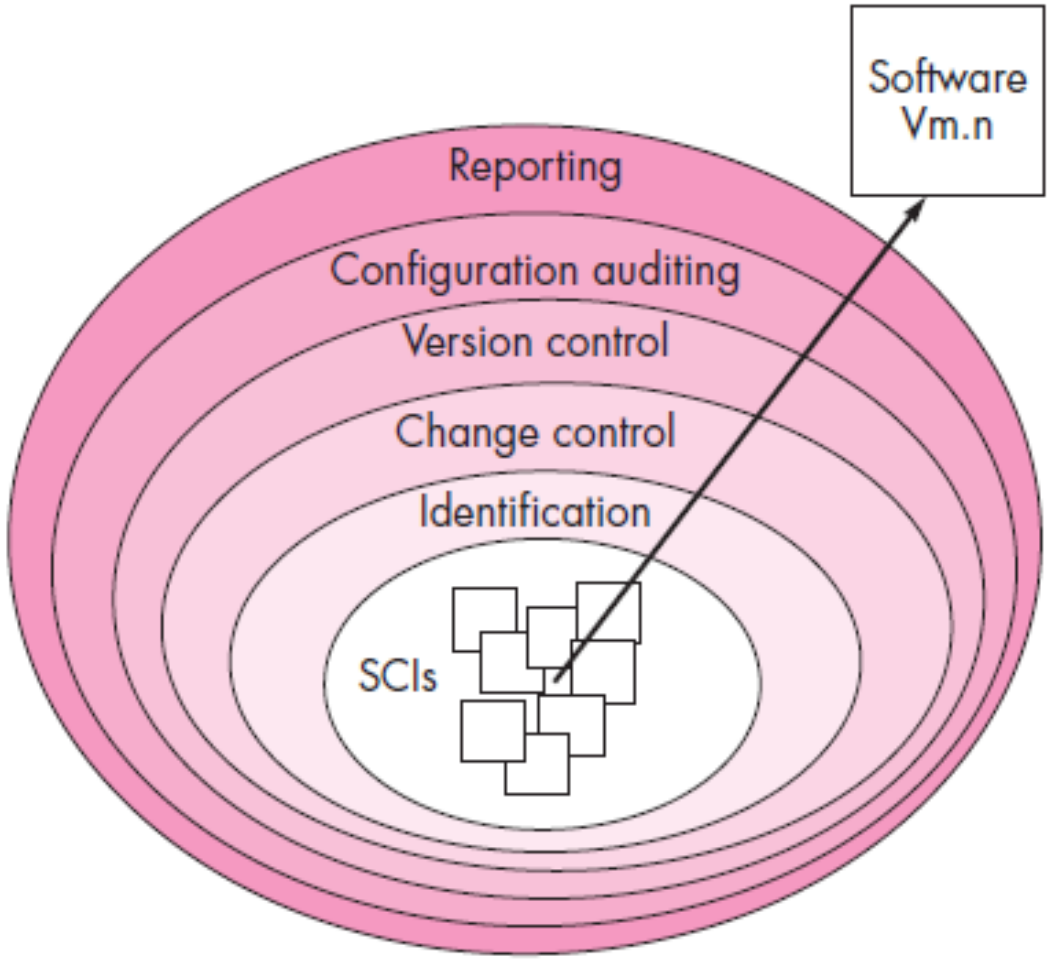
1. To identify all items that collectively define the software configuration
2. To manage changes to one or more of these items
3. To facilitate the construction of different versions of an application
4. To ensure that software quality is maintained as the configuration evolves over time.

Software Configuration Management

❑SCM Process:

FIGURE 22.4

Layers of the SCM process



Software Configuration Management

❑SCM Process:

- ❑Referring to the figure, SCM tasks can viewed as concentric layers.
- ❑SCIs flow outward through these layers throughout their useful life, ultimately becoming part of the software configuration of one or more versions of an application or system.
- ❑As an SCI moves through a layer, the actions implied by each SCM task may or may not be applicable.
- ❑For example, when a new SCI is created, it must be identified.
- ❑However, if no changes are requested for the SCI, the change control layer does not apply.
- ❑The SCI is assigned to a specific version of the software (version control mechanisms come into play).
- ❑A record of the SCI (its name, creation date, version designation, etc.) is maintained for configuration auditing purposes and reported to those with a need to know.
- ❑In the sections that follow, we examine each of these SCM process layers in more detail.

Software Configuration Management

❑SCM Process:

1. **Identification of Objects in the Software Configuration:** To control and manage software configuration items, each should be separately named and then organized using an object-oriented approach.
2. **Version Control :** Version control combines procedures and tools to manage different versions of configuration objects that are created during the software process.
3. **Change Control:** Change control is vital. But the forces that make it necessary also make it annoying. We worry about change because a tiny perturbation in the code can create a big failure in the product. But it can also fix a big failure or enable wonderful new capabilities. We worry about change because a single rogue developer could sink the project; yet brilliant ideas originate in the minds of those rogues, and a burdensome change control process could effectively discourage them from doing creative work.

Software Configuration Management

❑SCM Process:

4. **Configuration Audit :** Identification, version control, and change control help you to maintain order in what would otherwise be a chaotic and fluid situation. However, even the most successful control mechanisms track a change only until an ECO is generated. How can a software team ensure that the change has been properly implemented? The answer is twofold: (1) technical reviews and (2) the software configuration audit.
5. **Status Reporting:** Configuration status reporting (sometimes called status accounting) is an SCM task that answers the following questions: (1) What happened? (2) Who did it? (3) When did it happen? (4) What else will be affected?

Software Configuration Management Tools

❑**Following are the some popular Software Configuration Management Tools:**

1. CFEngine: CFEngine is an older open source configuration management application that enables automation configuration for large computer systems, including unified administration of servers, systems, users, embedded networked devices, mobile devices, and other devices.

2. Puppet: One of the greatest sources for DevOps trends is Puppet's yearly "State of DevOps" report. For those working in operations, understanding the advantages and disadvantages of the Puppet platform is becoming more and more crucial.

3. Chef: Chef, along with Puppet, is a major heavyweight in the CM and automation platform industry. It oversees servers that are either on-premises, in the cloud, or in a hybrid environment. As you switch cloud providers, you may manage both the data center and cloud environments at once if you are cloud-agnostic.

4. Ansible: Ansible, which is newer than Chef or Puppet, is the best open-source configuration management, deployment, orchestration, and automation engine. Fedora and other well-known Linux distributions contain it.

Software Configuration Management Tools

❑**Following are the some popular Software Configuration Management Tools:**

5. SaltStack : SaltStack, the leading proponent of the "infrastructure-as-code" philosophy, has amassed a substantial following despite its relatively late entry into the market due to its numerous integrations with cloud providers such as Google Cloud, Amazon Web Services (AWS), and others.

6. Docker: Docker is a relative newcomer that has taken the DevOps and software development worlds by storm since its inception in 2013.

7. Rudder: Rudder is an open-source IT infrastructure management application that runs on top of CFEngine. Rudder's special asset-management feature can identify nodes and their attributes, which might be helpful when carrying out configuration management operations. Asset management is used by this CMT to identify configuration management nodes.

Software Re-Engineering

- Software Re-engineering is a process of software development that is done to improve the maintainability of a software system.
- Re-engineering is the examination and alteration of a system to reconstitute it in a new form.
- This process encompasses a combination of sub-processes like reverse engineering, forward engineering, reconstructing, etc.
- Re-Engineering, is the process of analyzing, designing, and modifying existing software systems to improve their quality, performance, and maintainability.

1. This can include updating the software to work with new hardware or software platforms, adding new features, or improving the software's overall design and architecture.
2. Software re-engineering, also known as software restructuring or software renovation, refers to the process of improving or upgrading existing software systems to improve their quality, maintainability, or functionality.
3. It involves reusing the existing software artifacts, such as code, design, and documentation, and transforming them to meet new or updated requirements.

Software Re-Engineering

❑Objective of Re-engineering

❑The primary goal of software re-engineering is to improve the quality and maintainability of the software system while minimizing the risks and costs associated with the redevelopment of the system from scratch.

❑Software re-engineering can be initiated for various reasons, such as:

1. To describe a cost-effective option for system evolution.
2. To describe the activities involved in the software maintenance process.
3. To distinguish between software and data re-engineering and to explain the problems of data re-engineering.

Software Re-Engineering

❑**The process of software re-engineering involves the following steps:**

1. **Planning:** The first step is to plan the re-engineering process, which involves identifying the reasons for re-engineering, defining the scope, and establishing the goals and objectives of the process.
2. **Analysis:** The next step is to analyze the existing system, including the code, documentation, and other artifacts. This involves identifying the system's strengths and weaknesses.
3. **Design:** Based on the analysis, the next step is to design the new or updated software system. This involves identifying the changes that need to be made and developing a plan to implement them.
4. **Implementation:** The next step is to implement the changes by modifying the existing code, adding new features, and updating the documentation and other artifacts.
5. **Testing:** Once the changes have been implemented, the software system needs to be tested to ensure that it meets the new requirements and specifications.
6. **Deployment:** The final step is to deploy the re-engineered software system and make it available to end-users.

Software Re-Engineering

❑Why Perform Re-engineering?

1. **To improve the software's performance and scalability:** By analyzing the existing code and identifying bottlenecks, re-engineering can be used to improve the software's performance and scalability.
2. **To add new features:** Re-engineering can be used to add new features or functionality to existing software.
3. **To support new platforms**: Re-engineering can be used to update existing software to work with new hardware or software platforms.
4. **To improve maintainability:** Re-engineering can be used to improve the software's overall design and architecture, making it easier to maintain and update over time.
5. **To meet new regulations and compliance**: Re-engineering can be done to ensure that the software is compliant with new regulations and standards.
6. **Improving software quality:** Re-engineering can help improve the quality of software by eliminating defects, improving performance, and enhancing reliability and maintainability.

Software Re-Engineering

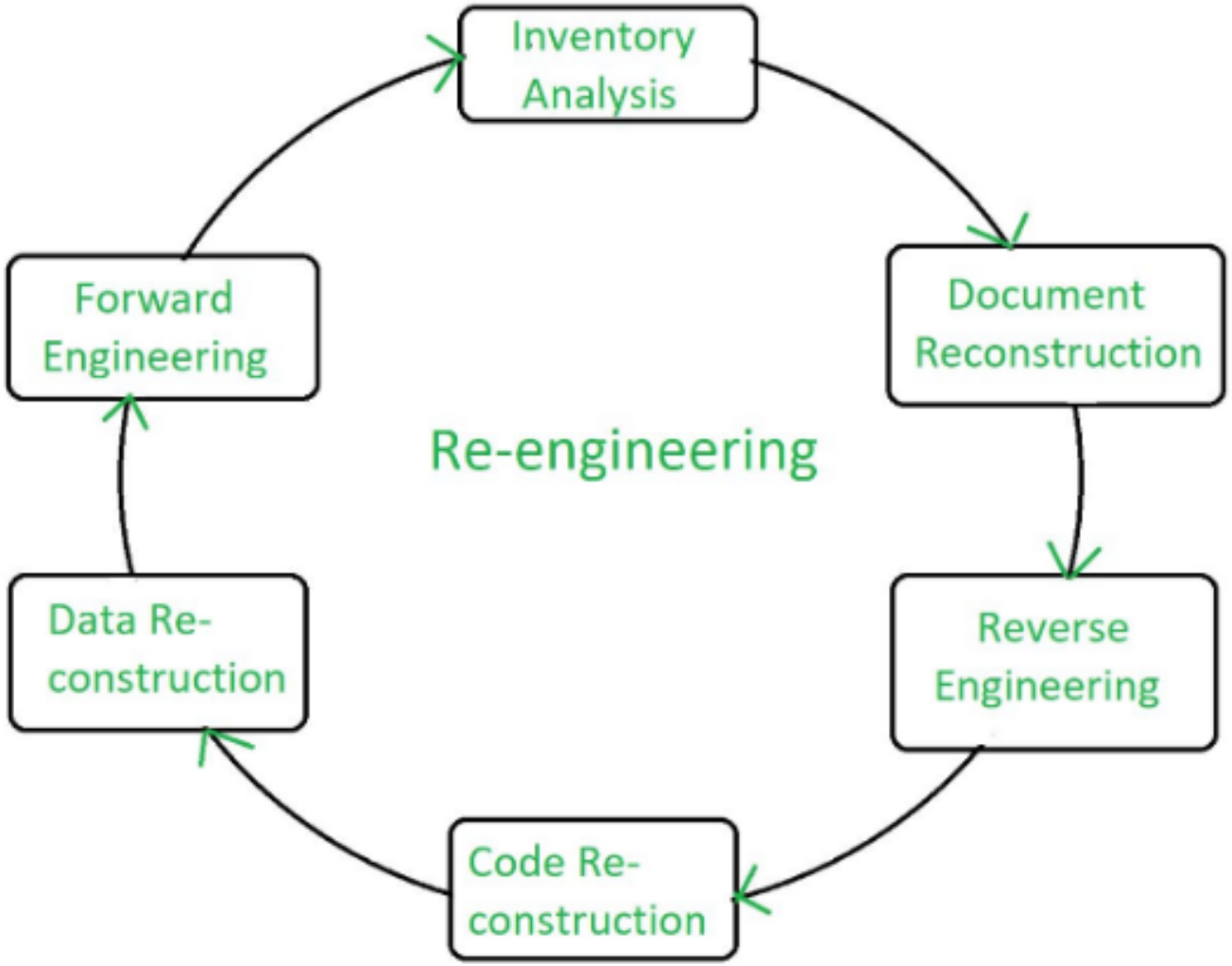
❑Why Perform Re-engineering?

7. **Updating technology:** Re-engineering can help modernize the software system by updating the technology used to develop, test, and deploy the system.
8. **Enhancing functionality:** Re-engineering can help enhance the functionality of the software system by adding new features or improving existing ones.
9. **Resolving issues:** Re-engineering can help resolve issues related to scalability, security, or compatibility with other systems.

Software Re-Engineering

❑Steps involved in Re-engineering

1. Inventory Analysis
2. Document Reconstruction
3. Reverse Engineering
4. Code Reconstruction
5. Data Reconstruction
6. Forward Engineering



Software Re-Engineering

❑Steps involved in Re-engineering

1. **Inventory Analysis:** Every software organization should have an inventory of all the applications. Inventory can be nothing more than a spreadsheet model containing information that provides a detailed description of every active application. By sorting this information according to business criticality, longevity, current maintainability, and other local important criteria, candidates for re-engineering appear. The resource can then be allocated to a candidate application for re-engineering work.
2. **Document reconstructing:** Documentation of a system either explains how it operates or how to use it. Documentation must be updated. It may not be necessary to fully document an application. The system is business-critical and must be fully re-documented.
3. **Reverse Engineering:** Reverse engineering is a process of design recovery. Reverse engineering tools extract data and architectural and procedural design information from an existing program.

Software Re-Engineering

❑Steps involved in Re-engineering

4. **Code Reconstructing:** To accomplish code reconstruction, the source code is analyzed using a reconstructing tool. Violations of structured programming construct are noted and code is then reconstructed. The resultant restructured code is reviewed and tested to ensure that no anomalies have been introduced.
5. **Data Restructuring:** Data restructuring begins with a reverse engineering activity. The current data architecture is dissected, and the necessary data models are defined. Data objects and attributes are identified, and existing data structures are reviewed for quality.
6. **Forward Engineering:** Forward Engineering also called renovation or reclamation not only recovers design information from existing software but uses this information to alter or reconstitute the existing system to improve its overall quality.

Software Re-Engineering

❑Advantages of Re-engineering

1. **Reduced Risk:** As the software is already existing, the risk is less as compared to new software development. Development problems, staffing problems and specification problems are the lots of problems that may arise in new software development.
2. **Reduced Cost:** The cost of re-engineering is less than the costs of developing new software.
3. **Revelation of Business Rules:** As a system is re-engineered , business rules that are embedded in the system are rediscovered.
4. **Better use of Existing Staff:** Existing staff expertise can be maintained and extended accommodate new skills during re-engineering.
5. **Improved efficiency:** By analyzing and redesigning processes, re-engineering can lead to significant improvements in productivity, speed, and cost-effectiveness.

Software Re-Engineering

❑Advantages of Re-engineering

1. **Reduced Risk:** As the software is already existing, the risk is less as compared to new software development. Development problems, staffing problems and specification problems are the lots of problems that may arise in new software development.
2. **Reduced Cost:** The cost of re-engineering is less than the costs of developing new software.
3. **Revelation of Business Rules:** As a system is re-engineered , business rules that are embedded in the system are rediscovered.
4. **Better use of Existing Staff:** Existing staff expertise can be maintained and extended accommodate new skills during re-engineering.
5. **Improved efficiency:** By analyzing and redesigning processes, re-engineering can lead to significant improvements in productivity, speed, and cost-effectiveness.

❑Disadvantages of Re-engineering

1. **High costs:** Re-engineering can be a costly process, requiring significant investments in time, resources, and technology.
2. **Disruption to business operations:** Re-engineering can disrupt normal business operations and cause inconvenience to customers, employees and other stakeholders.
3. **Resistance to change:** Re-engineering can encounter resistance from employees who may be resistant to change and uncomfortable with new processes and technologies.
4. **Risk of failure:** Re-engineering projects can fail if they are not planned and executed properly, resulting in wasted resources and lost opportunities.
5. **Lack of employee involvement:** Re-engineering projects that are not properly communicated and involve employees, may lead to lack of employee engagement and ownership resulting in failure of the project.

Reverse Engineering

- Software Reverse Engineering is a process of recovering the design, requirement specifications, and functions of a product from an analysis of its code.
- It builds a program database and generates information from this.
- Reverse engineering can extract design information from source code, but the abstraction level, the completeness of the documentation, the degree to which tools and a human analyst work together, and the directionality of the process are highly variable.

Reverse Engineering

❑Objective of Reverse Engineering:

1. **Reducing Costs:** Reverse engineering can help cut costs in product development by finding replacements or cost-effective alternatives for systems or components.
2. **Analysis of Security:** Reverse engineering is used in cybersecurity to examine exploits, vulnerabilities, and malware. This helps in understanding of threat mechanisms and the development of practical defenses by security experts.
3. **Integration and Customization:** Through the process of reverse engineering, developers can incorporate or modify hardware or software components into pre-existing systems to improve their operation or tailor them to meet particular needs.
4. **Recovering Lost Source Code:** Reverse engineering can be used to recover the source code of a software application that has been lost or is inaccessible or at the very least, to produce a higher-level representation of it.
5. **Fixing bugs and maintenance:** Reverse engineering can help find and repair flaws or provide updates for systems for which the original source code is either unavailable or inadequately documented.

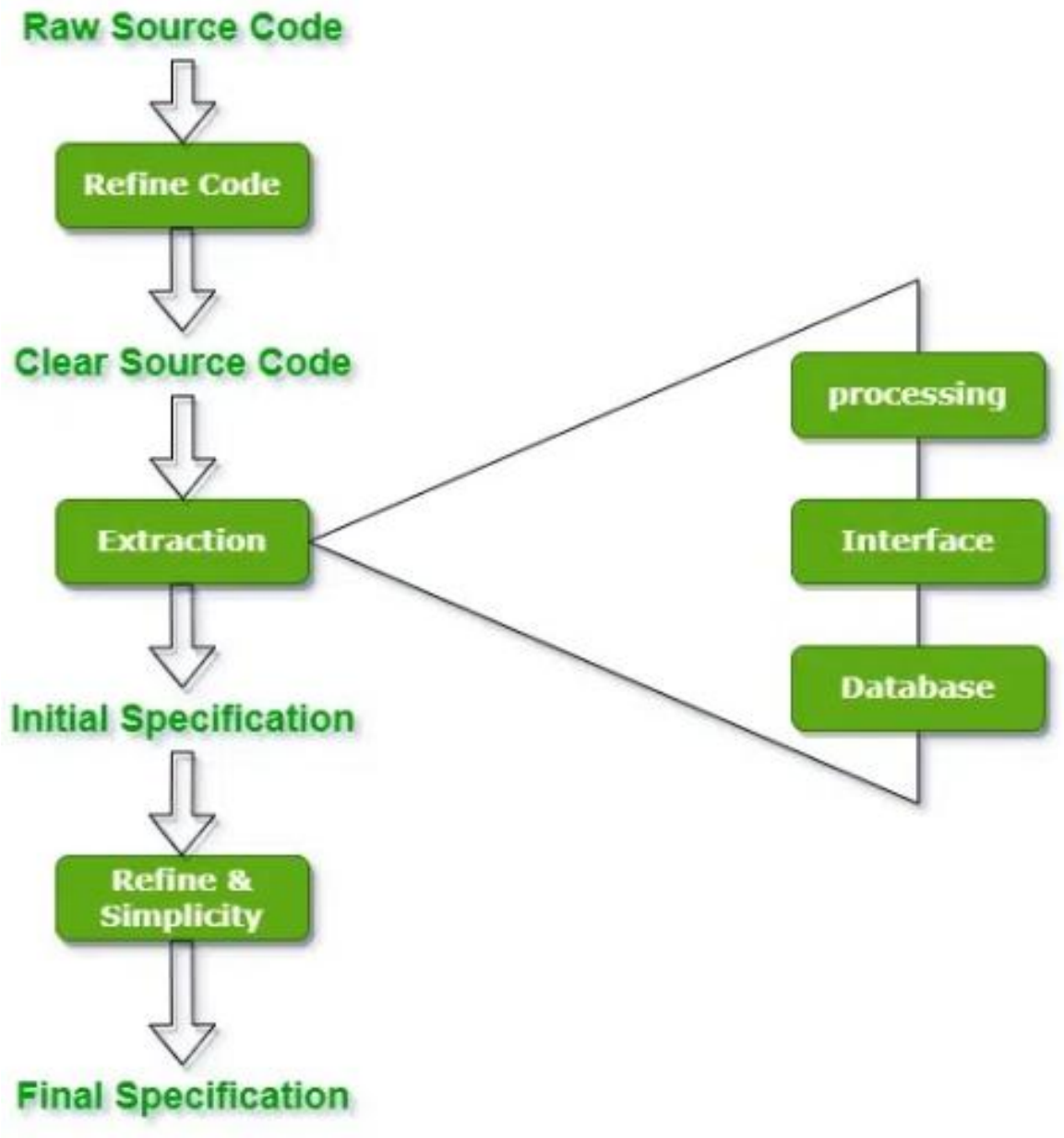
Reverse Engineering

❑Reverse Engineering Goals:

1. **Cope with Complexity:** Reverse engineering is a common tool used to understand and control system complexity. It gives engineers the ability to analyze complex systems and reveal details about their architecture, relationships and design patterns.
2. **Recover lost information:** Reverse engineering seeks to retrieve as much information as possible in situations where source code or documentation are lost or unavailable. Rebuilding source code, analyzing data structures and retrieving design details are a few examples of this.
3. **Detect side effects:** Understanding a system or component's behavior requires analyzing its side effects. Unintended implications, dependencies, and interactions that might not be obvious from the system's documentation or original source code can be found with the use of reverse engineering.
4. **Synthesis higher abstraction:** Abstracting low-level features in order to build higher-level representations is a common practice in reverse engineering. This abstraction makes communication and analysis easier by facilitating a greater understanding of the system's functionality.
5. **Facilitate Reuse:** Reverse engineering can be used to find reusable parts or modules in systems that already exist. By understanding the functionality and architecture of a system, developers can extract and repurpose components for use in other projects, improving efficiency and decreasing development time.

Reverse Engineering

❑**Reverse Engineering Goals:**



Reverse Engineering

❑Steps of Software Reverse Engineering:

1. **Collection Information:** This step focuses on collecting all possible information (i.e., source design documents, etc.) about the software.
2. **Examining the Information:** The information collected in step-1 is studied so as to get familiar with the system.
3. **Extracting the Structure:** This step concerns identifying program structure in the form of a structure chart where each node corresponds to some routine.
4. **Recording the Functionality:** During this step processing details of each module of the structure, charts are recorded using structured language like decision table, etc.
5. **Recording Data Flow:** From the information extracted in step-3 and step-4, a set of data flow diagrams is derived to show the flow of data among the processes.
6. **Recording Control Flow:** The high-level control structure of the software is recorded.
7. **Review Extracted Design:** The design document extracted is reviewed several times to ensure consistency and correctness. It also ensures that the design represents the program.
8. **Generate Documentation:** Finally, in this step, the complete documentation including SRS, design document, history, overview, etc. is recorded for future use.

Software Reuse

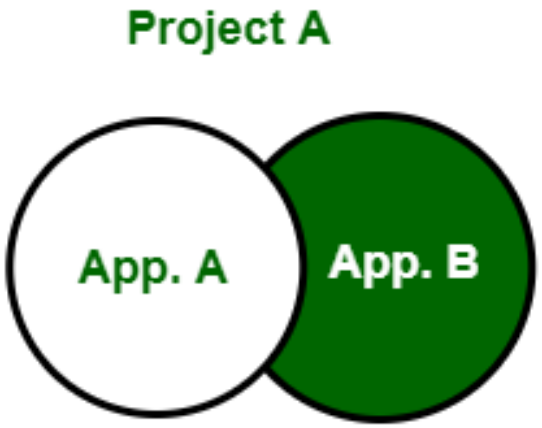
❑**Software Reuse** is the process of creating software systems from existing software systems, rather than building software system from scratch.”

❑**Ruse Maturity Model :** It is based on the experience within software development organization as well as experience with and observations of other development organizations.

Software Reuse

❑**LEVEL-1 : Single Project Source Based Reuse –** At the very first maturity level, organizations placed all their source code within a single project.

❑After this, the single pool of source code will hold multiple applications as depicted in the following figure:



❑Project A is home to two applications A and B. There is no need to copy source files or compiled files from one project to another because there is only one project.

❑But there is a limit that how well this practice will scale.

❑When number of applications or number of developers increases, to maintain the single source code will become difficult.

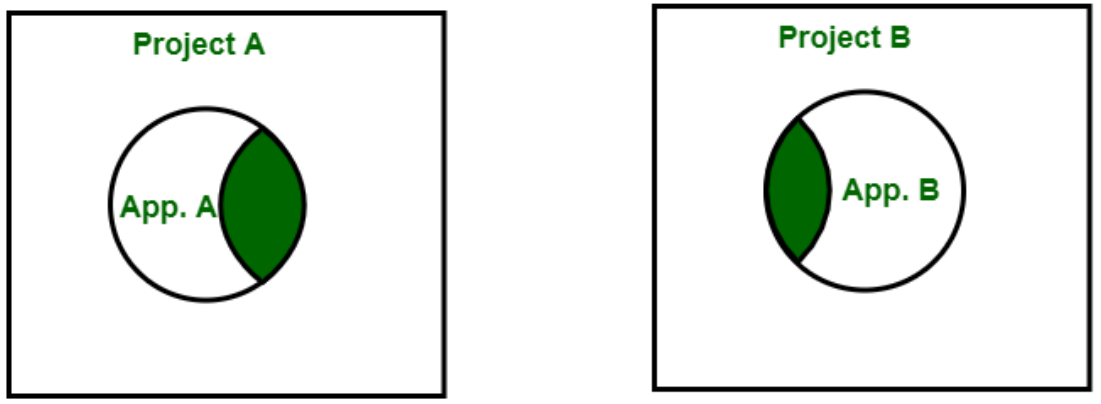
Software Reuse

❑**LEVEL-2 : Multi Project Source Based Reuse –** In this stage, source code is divided into multiple projects and practice source-based reuse between projects.

❑In this scenario, source-based reuse copies source developed in one project into another project.

❑Main target of such reuse is utilities, which can be developed in one project and can be used in the another project.

❑It is as shown in the following figure :



❑Project A is home to application A and project B is home to application B.

❑The code which is common to both the projects is copied into both projects. Problem with this is that, after copying, reused source code has to link back to the original.

❑This cause all sorts of maintenance problems, since bug fixes will have to be applied to every project that reuse copied code. Also, two host projects evolve reused code will evolve with them, which makes maintenance more difficult.

Software Reuse

❑**LEVEL-3 : Ad hoc Binary Reuse –** This is next step after level 2.

❑Organizations are advance to level and will realize drawback of level 2.

❑Under this approach, project boundaries realign and there is no longer mirror boundaries.

❑Projects at this level, can correspond to applications.

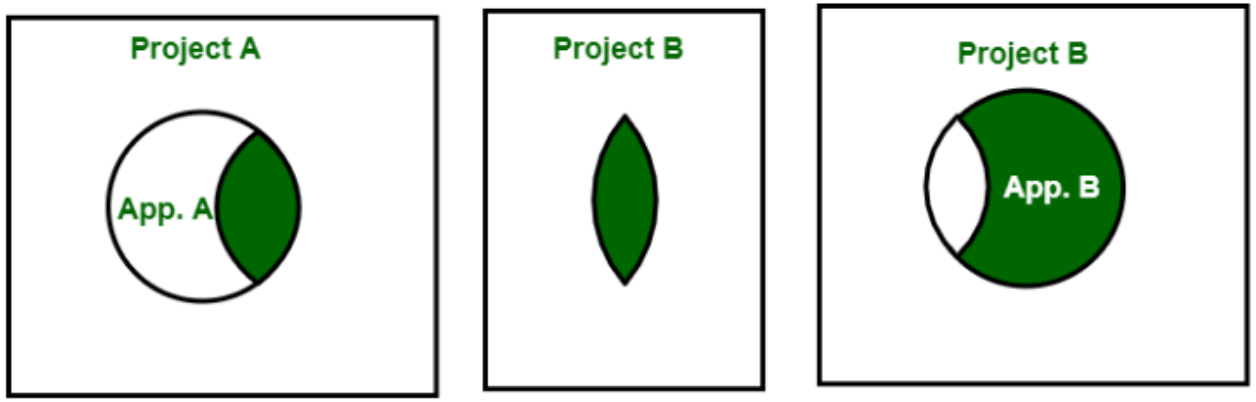
❑Utility source code that was copied from one project to another at previous level is now placed in its own project and has its own lifecycle independent of the application projects.

❑Application projects include binary artifacts of utilities project, and a dependency relationship between projects is established.

❑Maintenance of utilities project is greatly simplified because rather than maintaining multiple diverging copies of utilities, only a single version needs to b maintained.

❑LEVEL-3 : Ad hoc Binary Reuse:

❑Also application requires additional features, thus application features become available to all applications using utilities as shown in the following figure :



❑But in this, there are no release procedures or dependency management procedures.

❑Adding new features makes it impossible to know exact version of utilities used in application.

❑Also, with this when a bug is discovered it is difficult to know what version of code based= contained bug.

❑After a fix is implemented, it is difficult to know whether newest version will be compatible with all applications projects that need utilities library.

Software Reuse

❑**LEVEL-4 : Controlled Binary Reuse and the Reuse/Release Equivalence Principle –** It is based on ad hoc binary reuse.

❑The project boundaries remain same, with application projects and component projects.

❑At this level, each release of a project is controlled and tracked with a version number.

❑At this level, when the bug is discovered, the exact version of the component with the bug can be identified.

Note for Students

- This power point presentation is for lecture, therefore it is suggested that also utilize the text books and lecture notes.