UNIT - IV

❖ Software Maintenance

Software maintenance is the process of changing, modifying, and updating software to keep up with customer needs. Software maintenance is done after the product has launched for several reasons including improving the software overall, correcting issues or bugs, to boost performance, and more.
Software maintenance is a natural part of SDLC (software development life cycle). Software developers don't have the luxury of launching a product and letting it run, they constantly need to be on the lookout to both correct and improve their software to remain competitive and relevant.

Using the right software maintenance techniques and strategies is a critical part of keeping any software running for a long period of time and keeping customers and users happy.

software maintenance important

Creating a new piece of software and launching it into the world is an exciting step for any company. A lot goes into creating your software and its launch including the actual building and coding, licensing models, marketing, and more. However, any great piece of software must be able to adapt to the times.

This means monitoring and maintaining properly. As technology is changing at the speed of light, software must keep up with the market changes and demands.

❖ Basics of Software Maintenance

Software maintenance is the process of changing, modifying, and updating software to keep up with customer needs. Software maintenance is done after the product has launched for several reasons including improving the software overall, correcting issues or bugs, to boost performance, and more.
Software maintenance is a natural part of SDLC (software development life cycle).

Types of software maintenance

The four different types of software maintenance are each performed for different reasons and purposes. A given piece of software may have to undergo one, two, or all types of maintenance throughout its lifespan.

The four types are:

1. Corrective Software Maintenance
2. Preventative Software Maintenance
3. Perfective Software Maintenance
4. Adaptive Software Maintenance

Corrective Software Maintenance

Corrective software maintenance is the typical, classic form of maintenance (for software and anything else for that matter). Corrective software maintenance is necessary when something goes wrong in a piece of software including faults and errors. These can have a widespread impact on the functionality of the software in general and therefore must be addressed as quickly as possible.

Many times, software vendors can address issues that require corrective maintenance due to bug reports that users send in. If a company can recognize and take care of faults before users discover them, this is an added advantage that will make your company seem more reputable and reliable (no one likes an error message after all).

Preventative Software Maintenance

Preventative software maintenance is looking into the future so that your software can keep working as desired for as long as possible.

This includes making necessary changes, upgrades, adaptations and more. Preventative software maintenance may address small issues which at the given time may lack significance but may turn into larger problems in the future. These are called latent faults which need to be detected and corrected to make sure that they won't turn into effective faults.

Perfective Software Maintenance

As with any product on the market, once the software is released to the public, new issues and ideas come to the surface. Users may see the need for new features or requirements that they would like to see in the software to make it the best tool

available for their needs. This is when perfective software maintenance comes into play.

Perfective software maintenance aims to adjust software by adding new features as necessary and removing features that are irrelevant or not effective in the given software. This process keeps software relevant as the market, and user needs, change.

Adaptive Software Maintenance

Adaptive software maintenance has to do with the changing technologies as well as policies and rules regarding your software. These include operating system changes, cloud storage, hardware, etc. When these changes are performed, your software must adapt in order to properly meet new requirements and continue to run well.

The Software Maintenance Process

The software maintenance process typically involves the following steps:

1. **Problem identification:** The first step is to identify the problem that needs to be addressed. This may be a bug, a performance issue, or a change in the environment.

2. **Analysis:** Once the problem has been identified, it is necessary to analyze the problem and determine the root cause. This may involve reviewing the code, testing the software, or talking to the users of the software.

3. **Design:** Once the root cause of the problem has been determined, it is necessary to design a solution. This may involve modifying the code, adding new features, or improving the user interface.

4. **Implementation:** The next step is to implement the solution. This may involve writing new code, testing the changes, and deploying the changes to the production environment.

5. **Testing:** Once the changes have been implemented, it is necessary to test the software to ensure that the problem has been fixed and that there are no new problems.

6. **Documentation:** The final step is to document the changes that have been made. This is important for future reference and for ensuring that the software is maintained properly.

Software Maintenance Cost

The cost of software maintenance can be high. However, this doesn't negate the importance of software maintenance. In certain cases, software maintenance can cost up to two-thirds of the entire software process cycle or more than 50% of the SDLC processes.

The costs involved in software maintenance are due to multiple factors and vary depending on the specific situation. The older the software, the more maintenance will cost, as technologies (and coding languages) change over time. Revamping an old piece of software to meet today's technology can be an exceptionally expensive process in certain situations.

In addition, engineers may not always be able to target the exact issues when looking to upgrade or maintain a specific piece of software. This causes them to use a trial and error method, which can result in many hours of work.

There are certain ways to try and bring down software maintenance costs. These include optimizing the top of programming used in the software, strong typing, and functional programming.
When creating new software as well as taking on maintenance projects for older models, software companies must take software maintenance costs into consideration. Without maintenance, any software will be obsolete and essentially useless over time.

Software maintenance strategies

All software companies should have a specific strategy in place to tackle software maintenance in an effective and complete manner.

Documentation is one important strategy in software development. If software documentation isn't up to date, upgrading can be seemingly impossible. The documentation should include info about how the code works, solutions to potential problems, etc.

QA is also an important part of a software maintenance plan. While QA is important before an initial software launch, it can also be integrated much earlier in the process (as early as the planning stage) to make sure that the software is developed correctly and to give insight into making changes when necessary.

❖ Refactoring techniques Software version control

Refactoring or Code Refactoring is defined as systematic process of improving existing computer code, without adding new functionality or changing external behaviour of the code. It is intended to change the implementation, definition, structure of code without changing functionality of software. It improves extensibility, maintainability, and readability of software without changing what it actually does

code refactoring With example

Code refactoring is the process of restructuring to improve the existing code without changing its external behavior. The goal is to cleanse the internal structure of the code, making it more readable, maintainable, and efficient, while preserving its inherent functionality. It tidies up the codebase—making it more organized and easier to maintain. As per Deloitte, it can help modernize legacy applications without altering functionality. Python e.g.,

Original	Refactored

def factorial(n):
if n == 0:
return 1
else:
return n * factorial(n – 1)
	def factorial(n):
	result = 1
	for i in range(1, n + 1):
	result *= i
	return result

The refactored simple loop version removes the unnecessary “else” block, avoids a potential stack overflow, and provides a clearer understanding of the logic without losing the functionality of calculating the factorial.

Why should you refactor your code?

- Improves code clarity, making it easier to understand and navigate the codebase.

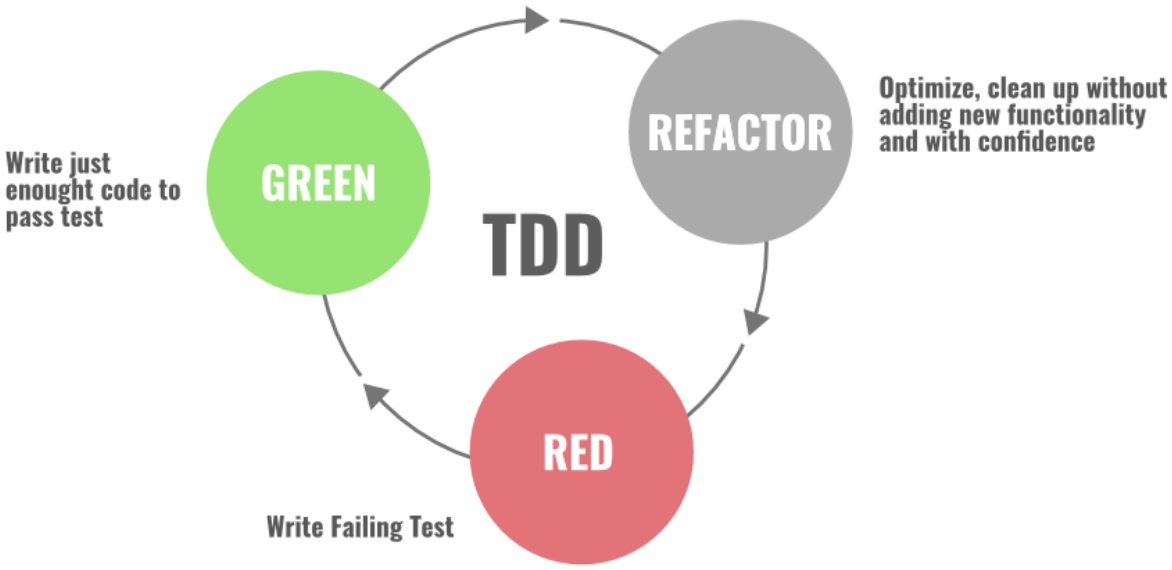
- Less convoluted code, facilitates easier maintenance and updates.
- Enables easier adaptation while scaling to adjust to the evolving requirements.
- Modularize code, making code components more reusable, testable, and adaptable.
- Detects and eliminates bottlenecks and bugs for crystal-clear code and efficiency.
- Enforces coding standards and consistency across the codebase.
- Improves collaboration, reduces communication overhead, and streamlines software development.

Most Common Code Refactoring Techniques

There are many approaches and techniques to refactor the code. Let's discuss some popular ones…

1. Red-Green Refactoring

Red-Green is the most popular and widely used code refactoring technique in the Agile software development process. This technique follows the "test-first" approach to design and implementation, this lays the foundation for all forms of refactoring. Developers take initiative for the refactoring into the test-driven development cycle and it is performed into the three district steps.



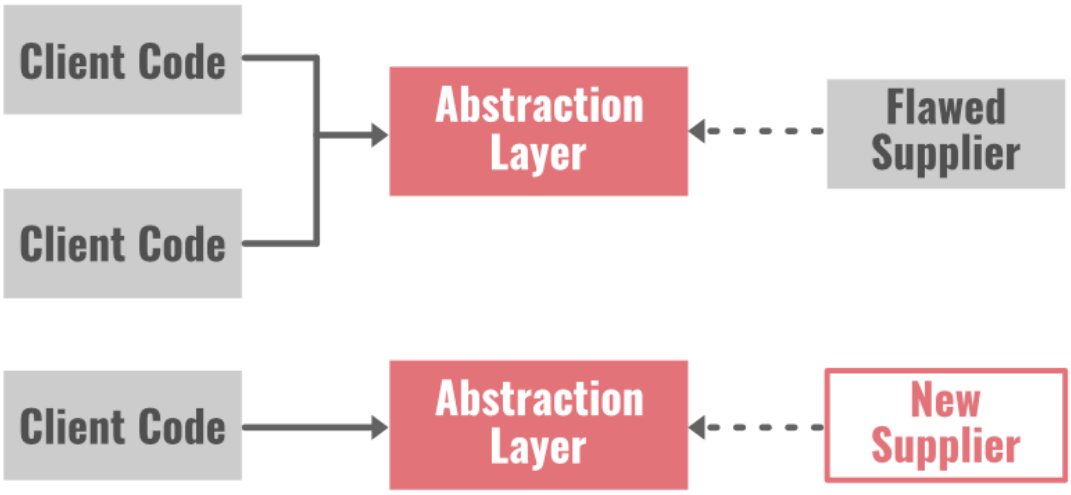
- **RED:** The first step starts with writing the failing "red-test". You stop and check what needs to be developed.

- **Green:** In the second step, you write the simplest enough code and get the development pass “green” testing.
- **Refactor:** In the final and third steps, you focus on improving and enhancing your code keeping your test green.

So basically this technique has two distinct parts: The first part involves writing code that adds a new function to your system and the second part is all about refactoring the code that does this function. Keep in mind that you’re not supposed to do both at the same time during the workflow.

2. Refactoring By Abstraction

This technique is mostly used by developers when there is a need to do a large amount of refactoring. Mainly we use this technique to reduce the redundancy (duplication) in our code. This involves class inheritances, hierarchy, creating new classes and interfaces, extraction, replacing inheritance with the delegation, and vice versa.



Pull-Up/Push-Down method is the best example of this approach.

- **Pull-Up method:** It pulls code parts into a superclass and helps in the elimination of code duplication.
- **Push-Down method:** It takes the code part from a superclass and moves it down into the subclasses.

Pull up the constructor body, extract subclass, extract superclass, collapse hierarchy, form template method, extract interface, replace inheritance with the delegation,

replace delegation with Inheritance, push down-field all these are the other examples.

Basically, in this technique, we build the abstraction layer for those parts of the system that needs to be refactored and the counterpart that is eventually going to replace it. Two common examples are given below…

- **Encapsulated field:** We force the code to access the field with getter and setter methods.
- **Generalize type:** We create more general types to allow code sharing, replace type-checking code with the state, replace conditional with polymorphism, etc.

3. Composing Method

During the development phase of an application a lot of times we write long methods in our program. These long methods make your code extremely hard to understand and hard to change. The composing method is mostly used in these cases.

In this approach, we use streamline methods to reduce duplication in our code. Some examples are: extract method, extract a variable, inline Temp, replace Temp with Query, inline method, split temporary variable, remove assignments to parameters, etc.

Extraction: We break the code into smaller chunks to find and extract fragmentation. After that, we create separate methods for these chunks, and then it is replaced with a call to this new method. Extraction involves class, interface, and local variables.

Inline: This approach removes the number of unnecessary methods in our program. We find all calls to the methods, and then we replace all of them with the content of the method. After that, we delete the method from our program.

4. Simplifying Methods

There are two techniques involved in this approach…let's discuss both of them.

- **Simplifying Conditional Expressions Refactoring:** Conditional statement in programming becomes more logical and complicated over time. You need to simplify the logic in your code to understand the whole program.
  There are so many ways to refactor the code and simplify the logic. Some of them are: consolidate conditional expression and duplicate conditional

fragments, decompose conditional, replace conditional with polymorphism, remove control flag, replace nested conditional with guard clauses, etc.

- **Simplifying Method Calls Refactoring:** In this approach, we make method calls simpler and easier to understand. We work on the interaction between classes, and we simplify the interfaces for them.
  Examples are: adding, removing, and introducing new parameters, replacing the parameter with the explicit method and method call, parameterize method, making a separate query from modifier, preserve the whole object, remove setting method, etc.

5. Moving Features Between Objects

In this technique, we create new classes, and we move the functionality safely between old and new classes. We hide the implementation details from public access.

Now the question is… when to move the functionality between classes or how to identify that it's time to move the features between classes?

When you find that a class has so many responsibilities and too much thing is going on or when you find that a class is unnecessary and doing nothing in an application, you can move the code from this class to another class and delete it altogether.

Examples are: move a field, extract class, move method, inline class, hide delegate, introduce a foreign method, remove middle man, introduce local extension, etc.

6. Preparatory Refactoring

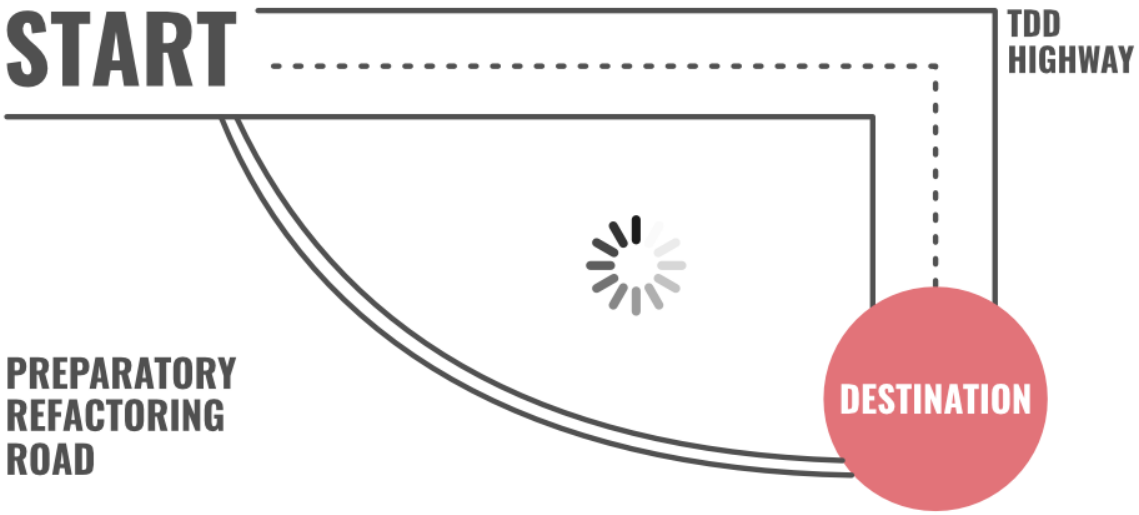
This approach is best to use when you notice the need for refactoring while adding some new features in an application. So basically it's a part of a software update with a separate refactoring process. You save yourself with future technical debt if you notice that the code needs to be updated during the earlier phases of feature development.

The end-user can not see such efforts of the engineering team eye to eye but the developers working on the application will find the value of refactoring the code when they are building the application. They can save their time, money, and other resources if they just spend some time updating the code earlier.

"It's like I want to go 100 miles east but instead of just traipsing through the woods, I'm going to drive 20 miles north to the highway and then I'm going to go 100 miles

east at three times the speed I could have if I just went straight there. When people are pushing you to just go straight there, sometimes you need to say, 'Wait, I need to check the map and find the quickest route.' The preparatory refactoring does that for me."

Jessica Kerr (Software Developer)



7. User Interface Refactoring

You can make simple changes in UI and refactor the code. For example: align entry field, apply font, reword in active voice indicate the format, apply common button size, and increase color contrast, etc.

Final Words

You need to consider the code refactoring process as cleaning up the orderly house. Unnecessary clutter in a home can create a chaotic and stressful environment. The same goes for written code. A clean and well-organized code is always easy to change, easy to understand, and easy to maintain. You won't be facing difficulty later if you pay attention to the code refactoring process earlier.

Two of the most influential software developers Martin Fowler and Kent Beck have devoted their time to explain the code refactoring process and the techniques of it. They have also written a complete book on this subject Refactoring: Improving the Design of Existing Code. This book describes various refactoring techniques with a clear explanation of working on these refactoring process. We recommend you to read this book if you want to go in-depth with the code refactoring process.

❖ Code Review

The code review is a methodical process where a group of developers work together to analyze and check another developer's code to detect errors, give suggestions, and confirm if the developed code is as per the standards. The objective of code review is to enhance the quality, maintainability, stability, security etc of the software which bring positive results to the project. Also, the findings from the code review promote sharing knowledge and learnings among the team members.

The code review is done for the reasons listed below –

- It helps to detect errors, defects, issues etc in the code prior to being deployed to production. Thus a code review helps to fix bugs at the initial phases of software development life cycle (SDLC).
- It motivates developing clean, maintainable, and effective code. The reviewers pass feedback and comments so that the code is as per the standards and best practices.
- It implements consistency in coding among all the developers which enables easy maintenance and understanding of the code base.
- The findings from the code reviews can be shared across teams which propagate domain knowledge and coding guidelines.
- The code reviewers take partial ownership of the code they review thereby increasing collective responsibility towards ensuring quality.
- The code reviewers can work together and collaborate to improve the entire review process which helps in enhancing the overall software quality.
- It can be a part of documentation.
- It is an integral part of ensuring the software quality. By doing code reviews, the team can confirm if the software meets all the functional and non-functional requirements.
- It helps to adopt continuous improvement for the team. By following the suggestions, feedback, and findings from the code review, the team can work up on them, then gradually improve.

Types of Code Review

Pull Requests (PR):

In Git, the developers raise a PR to incorporate changes to the code. It should be reviewed prior to the changes being merged with the base code.

Pair-Programming:

It is a type of review in which two developers work on the same computer. One of them writes the code and the other one reviews it in real time. It is a highly interactive form of code review.

Over the Shoulder Review:

It is the type of review in which one developer in the team is requested to review the code of another developer by sitting together and going through the code on the computer.

Tool Aided Reviews:

It is a type of review conducted by tools like Github, GitLab, BitBucket, Crucible etc.

Email Based Reviews:

It is a type of review in which the code changes sent over email for review. The feedback of the code review is also delivered in email.

Checklist Reviews:

It is a type of review in which the reviewers follow the list of checklist items for the review process.

Ad Hoc Review:

It is an informal way of review. A developer may be requested to have a quick look at the code and provide feedback not formally.

Formal Inspection:

It is a type of review in which an already existing process is followed. It is mostly done by an inspection team and is guided by proper documentation.

The code review is done by following the processes listed below –

Step 1 – The developers complete the code and create a review request or inform the team about the same.

Step 2 − Single or multiple code reviewers are selected based on their experiences and skills to review the code correctly.

Step 3 − The code reviewers have the required tools or IDE that enable them to get hold of the code, review it and pass the feedback.

Step 4 − The reviewers may follow checklists or guidelines while reviewing the code to maintain consistency.

Step 5 − The reviewers perform code inspections covering the logical, syntax, performance, security etc issues so that there is stability, scalability, and good performance in the code.

Step 6 − The reviewers' comments, feedback, and suggestions are recorded in the review tools. They should be clear, constructive, and to the point so that the author can easily understand them.

Step 7 − A detailed discussion between the reviewers and author is done regarding the expected code changes.

Step 8 − The author incorporates the changes and may have multiple discussions with the reviewers till all the issues are addressed and resolved in the code.

Step 9 − Once the reviewers are satisfied with the code changes, the code is approved for merging.

Step 10 − The code is merged using version control tools like Git.

Step 11 − Once the code is merged to the production, it is checked if the new changes in the code are working fine and they have not impacted any part of the existing code.

Step 12 − This entire code review process is documented for future references. All the relevant comments, feedback, and suggestions are also included in the documentations.

❖ Code Inspection

The software development life cycle (SDLC) consists of multiple stages. Each and every stage of it plays an important role towards software development. Inspection is a critical step for the complete building of the software.

The focus is not only in creating the software but also in verifying the entire code used for building it and detecting faults in them. This is known as code verification. It is of two types listed below −

- **Dynamic Technique** − It is done by running the software by feeding some inputs to it. Then the output it generates is examined to find issues in the code.
- **Static Technique** − It is done by running the software conceptually without any data and inputs. The static techniques include reading through the code, static analysis, code reviews, inspections etc.

The code inspection is done to review the code of the software and detect errors in it. It reduces the probability of fault multiplication and defects being detected at the later stages of SDLC by streamlining the bug identification procedures. Thus code inspection is a part of code review.

How Does the Code Inspection Work

The moderator, author, reader, and recorder forms a part of the code inspection team. All relevant documents are made available to this team to plan for the future course of actions in this regard. If the inspection team is unaware of the project, the author gives an introduction of the project and the outline of code to the inspection team.

The inspection team then checks every piece of code as per the inspection checklists. Once inspection has been completed, the inspection team informs the findings of the reviewed code to the respective team members.

The code inspection is done for the reasons listed below −

- It detects faults in the software code.
- It identifies if there are any needs for process improvements.
- It validates if the correct coding standards are followed in the project.
- It involves peer review of the code.
- It records and documents all the bugs in the code.

The common code inspection checklists are listed below −

- If the code is readable.
- If the code is maintainable.
- If there is efficient coding.
- If the code is incorporating all the software requirements.
- If the code has been created keeping in mind all the security features.
- If the code has the correct formatting, indentions, comments etc.
- If the code has been unit tested.
- If the code is as per the standards.
- If the code has all the relevant documentations and references.

The errors that are generally detected during the code inspections are listed below −

Data Errors − Some of the data errors are −

- The variables are not initialized correctly before actually using them.
- The constants have no proper names.
- There is buffer overflow.

Control Errors − Some of the control errors are −

- The conditions in the conditional statements are not correct.
- The loop is not ending correctly.
- All the code snippets do not correct brackets.

Input/Output Errors − Some of the input/output errors are −

- All the input variables remain unutilized.
- All the output variables are not assigned values.
- Some of the input values get corrupted.

Interface Errors − Some of the interface errors are −

- The methods and functions do not have proper parameters.
- The formal and actual parameters are not matching.
- The parameters are not appearing in the correct sequence.
- The parameters are not sharing the same memory structure.

Advantages of Code Inspections

The advantages of code inspection are listed below –

- It is done to enhance the software quality.
- It identifies bugs in the software code.
- It suggests various process improvements in the project.
- Educates the team how to leverage from past mistakes.
- It detects inefficiency in the code and project.

Disadvantages of Code Inspections

The disadvantages of code inspection are listed below –

- It is a time-consuming process.
- It requires extensive planning and execution.

❖ Software Evolution

Software evolution is the ongoing process of updating and improving software to keep up with changing needs, boost performance, and stay relevant. It ensures that the software keeps working properly, stays secure, and meets user expectations as circumstances and technology change.

Here are some more points about Software Evolution :

Regular Updates
It keeps software useful and efficient by fixing bugs and adding features.
Adaptability
It ensures compatibility with new technologies and user needs.
Enhanced Security
It also protects against new security threats and vulnerabilities.

Types of Software Evolution

There are several types of Software Evolution :

1. Corrective Evolution
2. Adaptive Evolution
3. Perfective Evolution
4. Preventive Evolution

Here are the types of software evolution explained in simple words:

- **Corrective Evolution:** This means fixing bugs and errors in the software after it's been released to make sure it works correctly and meets its requirements.
- **Adaptive Evolution:** This involves updating the software so it stays useful as the environment changes, like when new operating systems or hardware come out.
- **Perfective Evolution:** This is about adding new features or improving existing ones based on user feedback and changing needs.
- **Preventive Evolution:** This focuses on making the software easier to maintain in the future by cleaning up the code, updating documentation, and optimizing the system to prevent future problems.

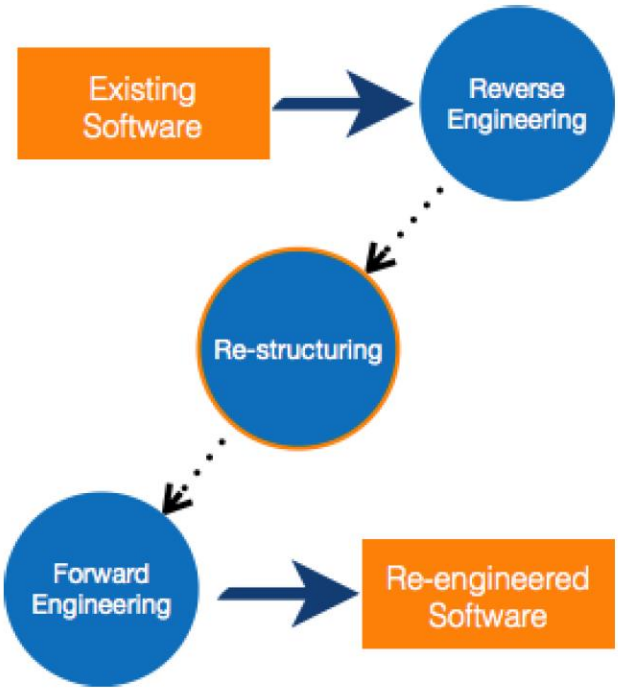
❖ Software Re-engineering

When we need to update the software to keep it to the current market, without impacting its functionality, it is called software re-engineering. It is a thorough process where the design of software is changed and programs are re-written.

Legacy software cannot keep tuning with the latest technology available in the market. As the hardware become obsolete, updating of software becomes a headache. Even if software grows old with time, its functionality does not.

For example, initially Unix was developed in assembly language. When language C came into existence, Unix was re-engineered in C, because working in assembly language was difficult.

Other than this, sometimes programmers notice that few parts of software need more maintenance than others and they also need re-engineering.



Re-Engineering Process

- • **Decide** what to re-engineer. Is it whole software or a part of it?

• **Perform** Reverse Engineering, in order to obtain specifications of existing software.

- **Restructure Program** if required. For example, changing function-oriented programs into object-oriented programs.
- **Re-structure data** as required.
- **Apply Forward engineering** concepts in order to get re-engineered software.

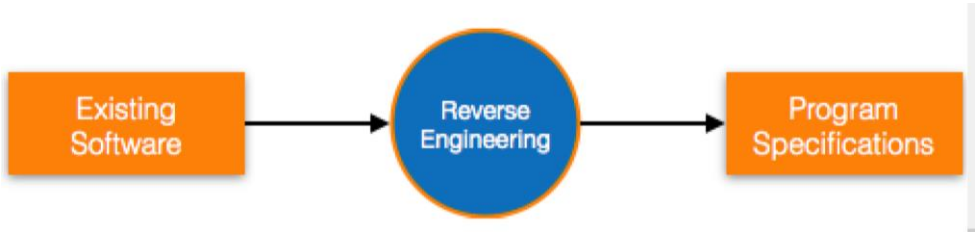
There are few important terms used in Software re-engineering

Reverse Engineering

It is a process to achieve system specification by thoroughly analyzing, understanding the existing system. This process can be seen as reverse SDLC model, i.e. we try to get higher abstraction level by analyzing lower abstraction levels.

An existing system is previously implemented design, about which we know nothing. Designers then do reverse engineering by looking at the code and try to get the

design. With design in hand, they try to conclude the specifications. Thus, going in reverse from code to system specification.



Program Restructuring

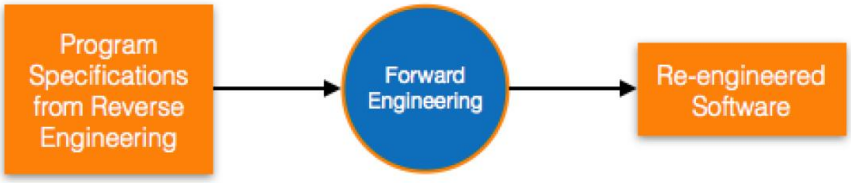
It is a process to re-structure and re-construct the existing software. It is all about re-arranging the source code, either in same programming language or from one programming language to a different one. Restructuring can have either source code-restructuring and data-restructuring or both.

Re-structuring does not impact the functionality of the software but enhance reliability and maintainability. Program components, which cause errors very frequently can be changed, or updated with re-structuring.

The dependability of software on obsolete hardware platform can be removed via re-structuring.

Forward Engineering

Forward engineering is a process of obtaining desired software from the specifications in hand which were brought down by means of reverse engineering. It assumes that there was some software engineering already done in the past.



Forward engineering is same as software engineering process with only one difference – it is carried out always after reverse engineering.