**Forward engineering**

**Def.**

Forward engineering is the method of creating or making application with the help of given requirements

**Characteristics**

* It has different principles in numerous package and information processes.
* It represents the ‘normal’ development process.
* It converts business processes, services, and functions into applications.
* Involves development of model followed by a top-to-down approach to urge the package from the model developed.
* It is accustomed to move from implementation styles and logic to the event of supply code.
* User develop business model which is then be translated into data system components.

**Reasons for** **reverse engineering**

* To find out how an object or system works.
* To find a way to create similar product.
* Reconstructing a product that is outdated.
* Discovering any product vulnerabilities.
* Inspiring creative minds with old ideas.
* Exploring existing designs.

**Reverse engineering tools**

**Compiler.**

Compiler, computer software that translates (compiles) source code written in a high-level language (e.g., C++) into a set of machine-language instructions that can be understood by a digital computer's CPU. Compilers are very large programs, with error-checking and other abilities.

**Assemblers.**

An assembler is a program that converts assembly language into machine code. It takes the basic commands and operations from assembly code.

**Linkers.**

Linker is a computer program that takes one or more object files generated by a compiler and combines them into one, executable program. Computer programs are usually made up of multiple modules that span separate object files, each being a compiled computer program

**SOFTWARE REUSE**

**Def:** Software reuse is the process of creating software systems from existing software rather than building software systems from scratch

**Advantages of software reuse**

**Less effort**. Software reuse requires less effort because many components use in the system are ready made components.

**Time-saving**. Re-using the ready-made components is time saving for the software team.

**Reduce cost**. Less effort, and time saving leads to the overall cost reduction.

**Increase software productivity**. When you are provided with ready-made components, then you can focus on the new components that are not available just like ready-made components.

**Utilize fewer resources**.  Software reuse save many sources just like effort, time, money etc.

**Leads to a better-quality software**.  Software reuse save our time and we can consume our more time on maintaining software quality and assurance.

**Disadvantages (Downside) of software reuse**

* Maintenance cost increases.
* It takes time to select reusable software components.
* Software tools may become obsolete.

**Stages of reuse-oriented software engineering**

**Requirement specification:**

First of all, specify the requirements. This will help to decide that we have some existing software components for the development of software or not.

**Component analysis**

Helps to decide which component can be reused and where.

**Requirement updating / modifications.**

If the requirements are changed by the customer, then still existing components are helpful for reuse or not.

**Reuse System design**

If the requirements are changed by the customer, then still existing system designs are helpful for reuse or not.

**Development**

Existing components are matching with new software or not.

**Integration**

Can we integrate the new systems with existing components?

**System validation**

To validate the system that it can be accepted by the customer or not.

**EVENT DRIVEN PROGRAMMING**

**Def:** Event-driven programming is a programming paradigm in which the flow of the program is determined by events such as user actions (mouse clicks, key presses), sensor outputs, or message passing from other programs or threads.

**Advantages of Event-Driven Programming**

**1. Flexibility**

 Programmers that use event-driven can be altered easily if the programmer wants something to be changed. This paradigm allows the programmer to produce a form of their requirements.

Programmers who are event-driven can be put together without too many problems and also the code and design can be easily altered because if something isn't right.

**2. Suitability for Graphical Interfaces**

 Event-driven allows the user to select different tools from the toolbar to directly create what they need such as buttons, radio buttons, etc.

This also allows people to put objects where they want them and can directly edit. Some people find it easier to directly click on the thing they want to edit.

**3. Simplicity of Programming**

Event-driven can make programming easier for some by being able to directly edit the object you want the code for.

Another thing that can make the programming easier is that when using an event driven language such as visual basic it usually has predictive coding so when the user is coding it will predict what you want to do from what you are typing.

**4. Easy to Find Natural Dividing Lines**

it is easy to find natural dividing lines for unit testing infrastructure.

**5. Highly Compostable**

It is highly compostable.

**6. Simple and Understandable**

It allows for a very simple and understandable model for both sides of the DevOps Bridge.

**7. Purely Procedural and Purely Imperative**

Both purely procedural and purely imperative approaches get brittle as they grow in length and complexity.

**8. A good way to Model Systems**

It is one good way to model systems that need to be both asynchronous and reactive.

**9. Allows for more Interactive Programs**

It allows for more interactive programs. Almost all modern GUI programs use event-driven programming.

**10. Using Hardware Interrupts**

It can be implemented using hardware interrupts, which will reduce the power used by the computer.

**11. Allows sensors and other hardware**

It allows sensors and other hardware to easily interact with software.

**Disadvantages Event-Driven Programming**

**1. Complex**

For simple programs, event-driven programming is often more complex and cumbersome than batch programming.

**2. Less Logical and Obvious**

The flow of the program is usually less logical and obvious

**3. Difficult to find Error**

Errors can be more difficult to spot than with simpler, procedural programs.

**4. Slower**

Programs with complex GUIs may be slower to load and run than simpler programs particularly if **RAM** is insufficient.

5. **Confusing**

Programs with too many forms can be very confusing and/or frustrating for the user

**6. Tight Coupling**

Possible tight coupling between the event schema and the consumers of the schema.

**7. Blocking**

Reasoning about **blocking** operations might be becoming more difficult.

**RE ENGINEERING**

**Def:** is a process of updating a software to keep it to the current market without impacting its functionality.

It involves change of the design of the software and programs re-written

It’s majorly aimed at adding a new functionality into the software

**Re-engineering process**

* Decide what to re-engineer. The whole or part of software
* Perform reverse engineering to obtain specifications of existing software.
* Restructure program if required.
* Re-structure data as required
* Apply forward engineering to get a re-engineered software

**PARALLELISM AND LEGACY SYSTEMS**

**Parallelism.**

The term Parallelism is the  techniques use to make programs faster by performing several computations at the same time

**Types of Parallelism in Processing Execution**

* Data Parallelism. Data Parallelism means concurrent execution of the same task on each multiple computing core.
* Task Parallelism. Task Parallelism means concurrent execution of the different task on multiple computing cores.
* Bit-level parallelism.
* Instruction-level parallelism.

**Types of Legacy Systems**

* End of life: Are legacy systems from the vendor’s perspective, are now past the usage stage. As a result, the vendor discontinues the product.
* No updates available: while this relates closely to end of life, you can often replace an end of life legacy system with a similar but updated solution or, as in the case of windows a vendor may offer a newer version that performs similarly.
* Unable to scale: Some software cannot scale sufficiently to support, for instance, larger streams of data or a bigger volume of financial transaction, the software has already become
* Heavily patched: The more patches that a software has required in the past, the more difficult it becomes to keep up with security concerns. And this make the software become vulnerable.
* Lack of qualified developers: If a company has developed or altered software in-house, it may be difficult or nearly impossible to find the qualified developers who can maintain the software.

**Reasons why companies continue to use legacy systems**

* Investment
* Fear
* Difficulty

**Problems caused by legacy systems**

* Maintenance is costly
* Data is stuck in silos
* Compliance is much harder.
* Security gets weaker day by day
* New systems don’t integrate

**The key to updating legacy systems**

* Ssuccessful data migration

**A successful data migration includes:**

* Extracting the existing data
* Transforming data so it matches the new formats
* Cleansing the data to address any quality issues
* Validating the data to make sure the move goes as planned
* Loading the data into the new system