**SOFTWARE EVOLUTION GROUP WORK**

**GROUP TWO MEMBERS**

NYANZI MATIA 19/U/0022/LCS

NAZZIWA SHARON 19/U/0021/LCS

LAYA GODFREY CLEMENT 19/U/0015/LCS

MUKASA JOSEPH SERWADDA 19/U/0093/LCS

NAULA ZAINABU EDRIS 19/U/0086/LCS

**QUESTIONS**

1. What is forward engineering, reverse engineering, re-engineering and reuse?
2. Discuss event-centered programming

**Forward engineering**

Forward engineering is the process of building from a high-level model or concept to build in complexities and lower-level details. This type of engineering has different principles in various software and database processes.

Or

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

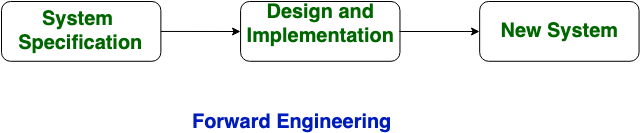
Forward engineering is also known as Renovation and Reclamation.

Forward engineering is required high proficiency skills and requires and takes more time to construct or develop an application

Forward engineering is a technique of creating high-level models or designs to make in complexities and low-level information. Therefore this kind of engineering has completely different principles in numerous package and information processes. Forward Engineering applies of all the software engineering process which contains SDLC to recreate associate existing application. It is near to full fill new needs of the users into re-engineering.

**Characteristics of forward engineering**:

* Forward engineering is a variety of engineering that has different principles in numerous package and information processes.
* Forward engineering is vital in IT as a result of it represents the ‘normal’ development process.
* Forward engineering deals with the conversion of business processes, services, and functions into applications.
* In this method business model is developed first. Then, a top-to-down approach is followed to urge the package from the model developed.
* Forward engineering tools are accustomed move from implementation styles and logic to the event of supply code.
* It essentially permits the user to develop a business model which may then be translated into data system components.
* These tools basically follow the top-to down approach. System creator and visual Analyst is a forward engineering CASE tool



**FORWARD ENGINEERING TOOLS**

Forward engineering tools are programs that move a program from a human-centric level of abstraction towards a machine-centric level of abstraction. Most programmers' main interface to the machine is the compilation environment.

The compilation environment takes as input one or more files in a high-level language such as C or Java, plus a number of supporting files such as resource files and libraries, and converts all of them into an executable for a particular execution environment, say Linux or Windows This is accomplished through a number of steps that involve individual programs:

* Each high-level language source file is compiled into assembly by a compiler for that high-level language.
* Each assembly language file, whether created by a compiler or directly by the programmer, is converted into a relocatable object file by an assembler program. The assembler is not concerned about which language was used to write the high-level source file. It is only concerned about which processor will execute the binary code. This is the first step where information can be lost, since the assembler may not see a lot of the information that is important to the programmer, such as local variable names and types.
* Each relocatable object file is combined together with a number of libraries that support the target execution environment by the linker. The linker may not care about the processor that will execute the program. It may only care about what information is required for the program to be loaded by the target operating system. The linker may decide to remove information from the generated binary file that it thinks will not be necessary to execute the program.

What's worse is that the programmer himself may instruct each tool to generate or remove valuable information. When using a compiler, the user may decide to:

* generate additional information to improve the debuggability of the code (the -g command line option of Unix compilers is used for this purpose)
* generate code that is more difficult to understand for humans, but is better executed by processors; that is, to generate optimized code through the -O1, -O2 or higher command line options. Optimizing compilers, through a number of transformations they perform on the generated instructions, make the final code less readable even when there is debugging information present in the final file and the original source is available and inspected through a debugger. Debugging optimized code is a worthy area of research in its own right, and will not be considered in this document, although many of the techniques can be applied to a 'de-optimizing debugger'.

When using any of the other tools, the user may also affect the operation of a decompiler, for example by instructing the linker to remove any symbolic information from the binary file.

From this point on, any tool that we can use to understand the program can be considered a reverse engineering tool.

1. **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.

1. **Assemblers.**

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

1. **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.

**REVERSE ENGINEERING TOOLS.**

Reverse engineering is the process of taking apart an object to understand its functions in order to duplicate or enhance its features or quality.

Reverse Engineering can also be defined as the process in which software, machines, aircraft, architectural structures and other products are deconstructed to extract design information from them.

**REVERSE ENGINEERING PROCESS**

There are three general steps common to all reverse-engineering efforts i.e., information extraction, modeling, and review. In software reverse-engineering this might require gathering source code and related design documents for study.

* **Information extraction.** The object being reverse-engineered is studied, information about its design is extracted and examined to determine how the pieces fit together. In software reverse-engineering this might require gathering source code and related design documents for study.
* **Modelling.** The collected information is abstracted into a conceptual model, with each piece of the model explaining its function in the overall structure. In software reverse-engineering this might take the form of data flow diagram or a structure chart.
* **Review.** Involves reviewing the model and testing it in various scenarios to ensure it is a realistic abstraction of the original object or system.

**TOOLS USED FOR REVERSE ENGINEERING:**

1. **Debuggers.** Debuggers are tools that are used for detecting and removing of existing and potential errors (also called as 'bugs') in a software code that can

cause it to behave unexpectedly or crash.

**Examples of debuggers**. Ida Pro, Ollydbg, Gbd, Immunity Debugger, Windbg

1. **Fault Injection Tools.** These tools can supply malformed or improperly formatted input to a target software process. The failures in programs can be analysed to determine whether errors exist in the targeted software. There are some failures that have security implications, such as failures that allow an attacker direct access to the host computer or network.

**Examples of fault injection tools:** Bstorm, The Mu Service Analyzer, Holodeck, Xception

1. **The Disassembler**. A disassembler is a tool reads the binary code and displays each executable instruction as text.

* **Examples of disassemblers** Binary Ninja, DEBUG, Interactive Disassembler (IDA), Ghidra, Hiew, Hopper Disassembler, PE Explorer Disassembler.

1. **The Reverse Compiler or Decompiler**: is a tool that converts assembly code or machine code into source code in a higher-level language such as C. These tools are extremely helpful in determining higher level logic such as loops, switches, and if-then statements and are much like disassemblers but take the process one step further which is really important.

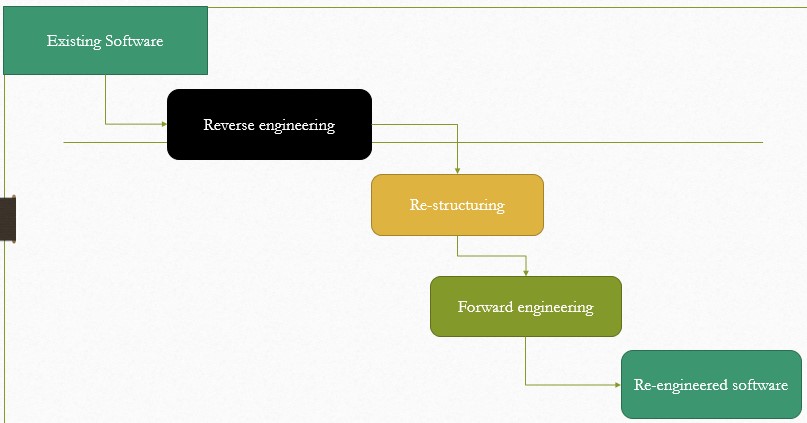


**Why reverse engineering**

* 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.

**RE ENGINEERING**

* This 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
* Its majorly aimed at adding a new functionality into the software



**Example**

* Facebook and you tube were first written in php. After emergence of new trends on market like AI , which php cannot do, facebook and youtube were re-engineered to suit the market trends using new powerful programming languages such as python.

**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

**SOFTWARE REUSE**

Software reuse is the process of creating software systems from existing software rather than building software systems from scratch.

Software reuse is a term used for developing the software by using the existing software components. Some of the components that can be reused are; source code, Design and interfaces, Software Documentation, Software requirement specifications and many more.

**What is reuse software engineering?**

Reuse software engineering is based on guidelines and principles for reusing the existing software.

What are 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.

**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.

**Downside of software reuse**

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

**EVENT-CENTERED PROGRAMMING**

**An event**: something that happens or takes place, especially one of importance.

**Centered**: occur mainly in or around (a specified place).

In computer programming, 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.

**Event-Driven Programming**

Most programs and devices like a cellphone respond to *events* — things that happen. For example, you might move your mouse, and the computer responds. Or you click a button, and the program does something interesting. In this chapter we’ll touch very briefly on how event-driven programming works.

**Key press events**

Here’s a program with some new features. Copy it into your workspace, run it. When the turtle window opens, press the arrow keys and make tess move about!

We can refer to keys on the keyboard by their character code (as we did in line 26), or by their symbolic names. Some of the symbolic names to try are Cancel (the Break key), BackSpace, Tab, Return(the Enter key), Shift\_L (any Shift key), Control\_L (any Control key), Alt\_L (any Alt key), Pause, Caps\_Lock, Escape, Prior (Page Up), Next (Page Down), End, Home, Left, Up, Right, Down, Print, Insert, Delete, F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, Num\_Lock, and Scroll\_Lock.

**Mouse events**[**¶**](https://openbookproject.net/thinkcs/python/english3e/events.html#mouse-events)

A mouse event is a bit different from a keypress event because its handler needs two parameters to receive x,y coordinate information telling us where the mouse was when the event occurred.

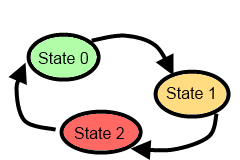
Automatic events from a timer .Alarm clocks, kitchen timers, and thermonuclear bombs in James Bond movies are set to create an “automatic” event after a certain interval. The turtle module in Python has a timer that can cause an event when its time is up.

An example: state machines

A state machine is a system that can be in one of a few different states. We draw a state diagram to represent the machine, where each state is drawn as a circle or an ellipse. Certain events occur which cause the system to leave one state and transition into a different state. These state transitions are usually drawn as an arrow on the diagram.

This idea is not new: when first turning on a cellphone, it goes into a state which we could call “Awaiting PIN”. When the correct PIN is entered, it transitions into a different state — say “Ready”. Then we could lock the phone, and it would enter a “Locked” state, and so on.

A simple state machine that we encounter often is a traffic light. Here is a state diagram which shows that the machine continually cycles through three different states, which we’ve numbered 0, 1 and 2.



**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.

**Principles of Event-Driven Programming**

A set of functions that handle events. Depending on the implementation these can be blocking or non-blocking.

A mechanism for binding the registered functions to events.

The main loop (or loops, if you are brave) which constantly polls for new events and calls the matching event handler(s) when a registered event is received.

**Why is Event-Driven Programming Object Oriented?**

Visual Basic is not an "Object-Oriented Language" its "Object-based Language" because, in VB we cannot implement Object-Oriented Concepts like **Inheritance**, **Polymorphism**, **Encapsulation**, and **Abstraction**.

But it is an Event drive programming, easy to develop applications. VB is used to Develop Graphical User Interface Applications (GUIs).

VB is Object-Based Event-Driven Programming Language as we use Objects to create our Applications such as Text Box, Command Button on Form and it is also called Event Driven Language because anything we do on VB application such as Click Mouse on command Button to execute any command or enter the text in a text box, etc.

Everything is an event that's why we call it Object-Based, Event-Driven Programming