3. REGRESSION TESTING, DEBUGGING, PROGRAM ANALYSIS

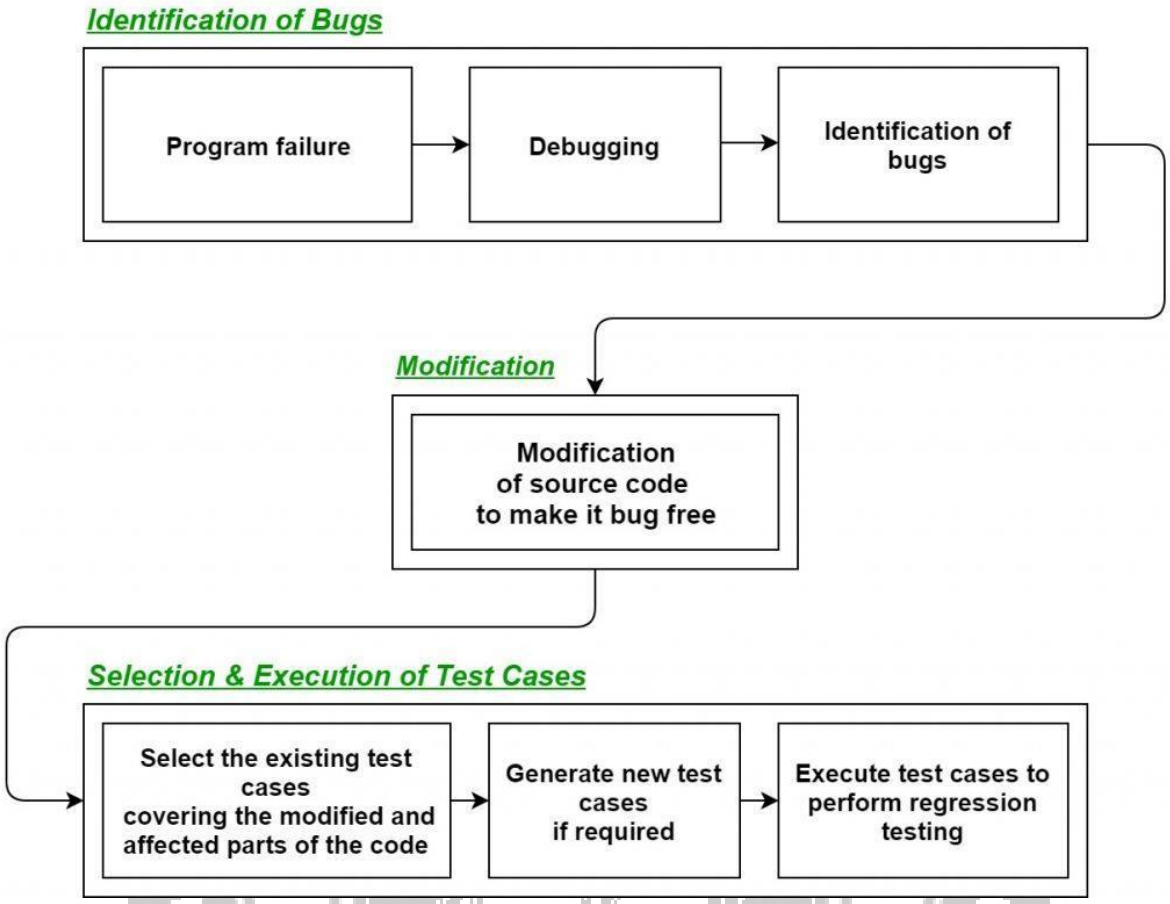
It is the process of testing the modified parts of the code and the parts that might get affected due to the modifications to ensure thatno new errors have been introduced in the software after the modifications have been made. Regression means the return of something and in the software field, it refers to the return of a bug.

When to do regression testing?

- When a new functionality is added to the system and the code has been modified to absorb and integrate that functionality with the existing code.
- When some defect has been identified in the software and the code is debugged to fix it.
- When the code is modified to optimize its working.

Process of Regression testing:

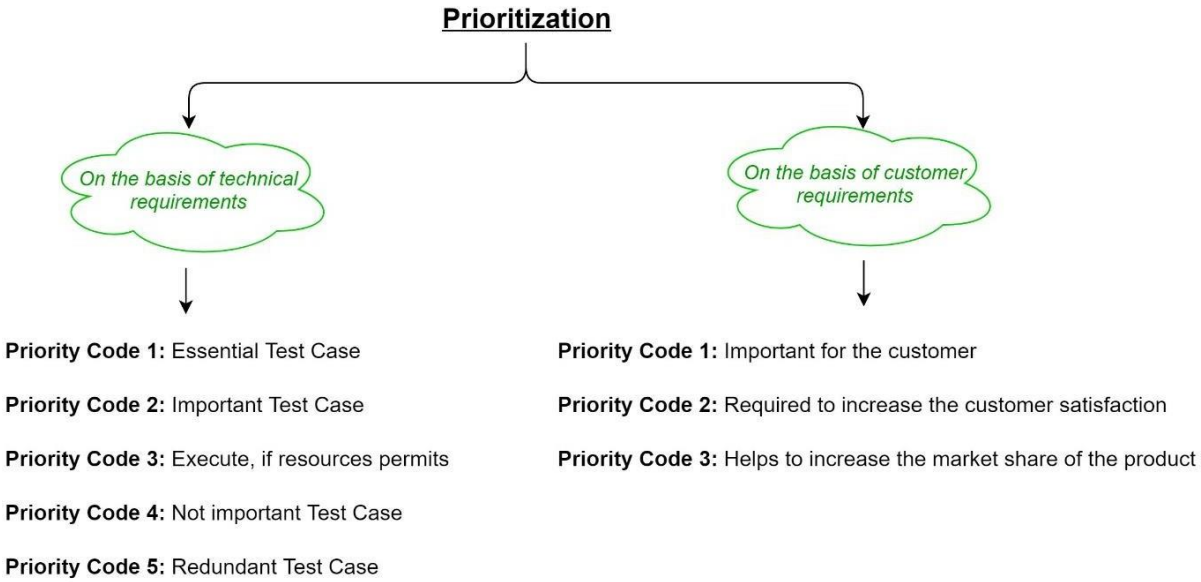
0 seconds of 17 seconds Volume 0% Firstly, whenever we make some changes to the source code for any reason like adding new functionality, optimization, etc. then our program when executed fails in the previously designed test suite for obvious reasons. After the failure, the source code is debugged in order to identify the bugs in the program. After identification of the bugs in the source code, appropriate modifications are made. Then appropriate test cases are selected from the already existing test suite which covers all the modified and affected parts of the source code. We can add new test cases if required. In the end, regression testing is performed using the selected test cases.



Techniques for the selection of Test cases for Regression Testing:

- **Select all test cases:** In this technique, all the test cases are selected from the already existing test suite. It is the simplest and safest technique but not much efficient.
- **Select test cases randomly:** In this technique, test cases are selected randomly from the existing test-suite, but it is only useful if all the test cases are equally good in their fault detection capability which is very rare. Hence, it is not used in most of the cases.
- **Select modification traversing test cases:** In this technique, only those test cases are selected which covers and tests the modified portions of the source code the parts which are affected by these modifications.
- **Select higher priority test cases:** In this technique, priority codes are assigned to each test case of the test suite based upon their bug detection capability, customer requirements, etc. After assigning the priority codes, test cases with the highest priorities are selected for the process of regression testing. The test case with the highest priority has the highest

rank. For example, test case with priority code 2 is less important than test case with priority code 1.



Tools for regression testing:

In regression testing, we generally select the test cases from the existing test suite itself and hence, we need not compute their expected output, and it can be easily automated due to this reason. Automating the process of regression testing will be very much effective and time saving. Most commonly used tools for regression testing are:

- Selenium
- WATIR (Web Application Testing In Ruby)
- QTP (Quick Test Professional)
- RFT (Rational Functional Tester)
- Winrunner
- Silktest

Advantages of Regression Testing:

- It ensures that no new bugs has been introduced after adding new functionalities to the system.
- As most of the test cases used in Regression Testing are selected from the existing test suite, and we already know their expected outputs. Hence, it can be easily automated by the automated tools.
- It helps to maintain the quality of the source code.

Disadvantages of Regression Testing:

- It can be time and resource consuming if automated tools are not used.
- It is required even after very small changes in the code.

DEBUGGING

Debugging is the process of identifying and resolving errors, or bugs, in a software system. It is an important aspect of software engineering because bugs can cause a software system to malfunction, and can lead to poor performance or incorrect results. Debugging can be a time-consuming and complex task, but it is essential for ensuring that a software system is functioning correctly.

There are several common methods and techniques used in debugging, including:

1. **Code Inspection:** This involves manually reviewing the source code of a software system to identify potential bugs or errors.
2. **Debugging Tools:** There are various tools available for debugging such as debuggers, trace tools, and profilers that can be used to identify and resolve bugs.
3. **Unit Testing:** This involves testing individual units or components of a software system to identify bugs or errors.
4. **Integration Testing:** This involves testing the interactions between different components of a software system to identify bugs or errors.
5. **System Testing:** This involves testing the entire software system to identify bugs or errors.
6. **Monitoring**: This involves monitoring a software system for unusual behavior or performance issues that can indicate the presence of bugs or errors.
7. **Logging**: This involves recording events and messages related to the software system, which can be used to identify bugs or errors.

It is important to note that debugging is an iterative process, and it may take multiple attempts to identify and resolve all bugs in a software system. Additionally, it is important to have a well-defined

process in place for reporting and tracking bugs, so that they can be effectively managed and resolved.

In summary, debugging is an important aspect of software engineering, it's the process of identifying and resolving errors, or bugs, in a software system.

There are several common methods and techniques used in debugging, including code inspection, debugging tools, unit testing, integration testing, system testing, monitoring, and logging. It is an iterative process that may take multiple attempts to identify and resolve all bugs in a software system.

In the context of software engineering, debugging is the process of fixing a bug in the software. In other words, it refers to identifying, analyzing, and removing errors.

This activity begins after the software fails to execute properly and concludes by solving the problem and successfully testing the software.

It is considered to be an extremely complex and tedious task because errors need to be resolved at all stages of debugging.

A better approach is to run the program within a debugger, which is a specialized environment for controlling and monitoring the execution of a program.

The basic functionality provided by a debugger is the insertion of breakpoints within the code. When the program is executed within the debugger, it stops at each breakpoint. Many IDEs,such as Visual C++ and C-Builder provide built-in debuggers.

Debugging Process: The steps involved in debugging are:

- Problem identification and report preparation.
- Assigning the report to the software engineer defect to verify that it is genuine.
- Defect Analysis using modeling, documentation, finding and testing candidate flaws, etc.
- Defect Resolution by making required changes to the system.
- Validation of corrections.

The debugging process will always have one of two outcomes:

1. The cause will be found and corrected.
2. The cause will not be found.

Later, the person performing debugging may suspect a cause, design a test case to help validate that suspicion, and work toward error correction in an iterative fashion.

During debugging, we encounter errors that range from mildly annoying to catastrophic.

As the consequences of an error increase, the amount of pressure to find the cause also increases. Often, pressure sometimes forces a software developer to fix one error and at the same time introduce two more.

Debugging Approaches/Strategies:

1. **Brute Force:** Study the system for a longer duration to understand the system. It helps the debugger to construct different representations of systems to be debugged depending on the need. A study of the system is also done actively to find recent changes made to the software.
2. **Backtracking:** Backward analysis of the problem which involves tracing the program backward from the location of the failure message to identify the region of faulty code. A detailed study of the region is conducted to find the cause of defects.
3. **Forward analysis** of the program involves tracing the program forwards using breakpoints or print statements at different points in the program and studying the results. The region where the wrong outputs are obtained is the region that needs to be focused on to find the defect.
4. **Using** A debugging**experience** with the software debug the software with similar problems in nature. The success of this approach depends on the expertise of the debugger.
5. **Cause elimination:** it introduces the concept of binary partitioning. Data related to the error occurrence are organized to isolate potential causes.
6. **Static analysis:** Analyzing the code without executing it to identify potential bugs or errors. This approach involves analyzing code syntax, data flow, and control flow.
7. **Dynamic analysis:** Executing the code and analyzing its behavior at runtime to identify errors or bugs. This approach involves techniques like runtime debugging and profiling.
8. **Collaborative debugging:** Involves multiple developers working together to debug a system. This approach is helpful in situations where multiple modules or components are

involved, and the root cause of the error is not clear.

9. **Logging and Tracing:** Using logging and tracing tools to identify the sequence of events leading up to the error. This approach involves collecting and analyzing logs and traces generated by the system during its execution.
10. **Automated Debugging:** The use of automated tools and techniques to assist in the debugging process. These tools can include static and dynamic analysis tools, as well as tools that use machine learning and artificial intelligence to identify errors and suggest fixes.

Debugging Tools:

A debugging tool is a computer program that is used to test and debug other programs. A lot of public domain software like gdb and dbx are available for debugging. They offer console-based command-line interfaces. Examples of automated debugging tools include code-based tracers, profilers, interpreters, etc. Some of the widely used debuggers are:

- Radare2
- WinDbg
- Valgrind

Difference Between Debugging and Testing:

Debugging is different from testing. Testing focuses on finding bugs, errors, etc whereas debugging starts after a bug has been identified in the software. Testing is used to ensure that the program is correct and it was supposed to do with a certain minimum success rate. Testing can be manual or automated. There are several different types of testing unit testing, integration testing, alpha, and beta testing, etc. Debugging requires a lot of knowledge, skills, and expertise. It can be supported by some automated tools available but is more of a manual process as every bug is different and requires a different technique, unlike a pre-defined testing mechanism.

Advantages of Debugging:

Several advantages of debugging in software engineering:

1. **Improved system quality:** By identifying and resolving bugs, a software system can be made more reliable and efficient, resulting in improved overall quality.
2. **Reduced system downtime:** By identifying and resolving bugs, a software system can be made more stable and less

likely to experience downtime, which can result in improved availability for users.

3. **Increased user satisfaction**: By identifying and resolving bugs, a software system can be made more user-friendly and better able to meet the needs of users, which can result in increased satisfaction.
4. **Reduced development costs:** Identifying and resolving bugs early in the development process, can save time and resources that would otherwise be spent on fixing bugs later in the development process or after the system has been deployed.
5. **Increased security**: By identifying and resolving bugs that could be exploited by attackers, a software system can be made more secure, reducing the risk of security breaches.
6. **Facilitates change:** With debugging, it becomes easy to make changes to the software as it becomes easy to identify and fix bugs that would have been caused by the changes.
7. **Better understanding of the system:** Debugging can help developers gain a better understanding of how a software system works, and how different components of the system interact with one another.
8. **Facilitates testing:** By identifying and resolving bugs, it makes it easier to test the software and ensure that it meets the requirements and specifications.

In summary, debugging is an important aspect of software engineering as it helps to improve system quality, reduce system downtime, increase user satisfaction, reduce development costs, increase security, facilitate change, a better understanding of the system, and facilitate testing.

Disadvantages of Debugging:

While debugging is an important aspect of software engineering, there are also some disadvantages to consider:

1. **Time-consuming:** Debugging can be a time-consuming process, especially if the bug is difficult to find or reproduce. This can cause delays in the development process and add to the overall cost of the project.

2. **Requires specialized skills:** Debugging can be a complex task that requires specialized skills and knowledge. This can be a challenge for developers who are not familiar with the tools and techniques used in debugging.
3. **Can be difficult to reproduce:** Some bugs may be difficult to reproduce, which can make it challenging to identify and resolve them.
4. **Can be difficult to diagnose:** Some bugs may be caused by interactions between different components of a software system, which can make it challenging to identify the root cause of the problem.
5. **Can be difficult to fix:** Some bugs may be caused by fundamental design flaws or architecture issues, which can be difficult or impossible to fix without significant changes to the software system.
6. **Limited insight**: In some cases, debugging tools can only provide limited insight into the problem and may not provide enough information to identify the root cause of the problem.
7. **Can be expensive:** Debugging can be an expensive process, especially if it requires additional resources such as specialized debugging tools or additional development time.

In summary, debugging is an important aspect of software engineering but it also has some disadvantages, it can be time-consuming, requires specialized skills, can be difficult to reproduce, diagnose, and fix, may have limited insight, and can be expensive.

PROGRAM ANALYSIS

Program Analysis Tool is an automated tool whose input is the source code or the executable code of a program and the output is the observation of characteristics of the program.

It gives various characteristics of the program such as its size, complexity, adequacy of commenting, adherence to programming standards and many other characteristics. These tools are essential to software engineering because they help programmers comprehend,

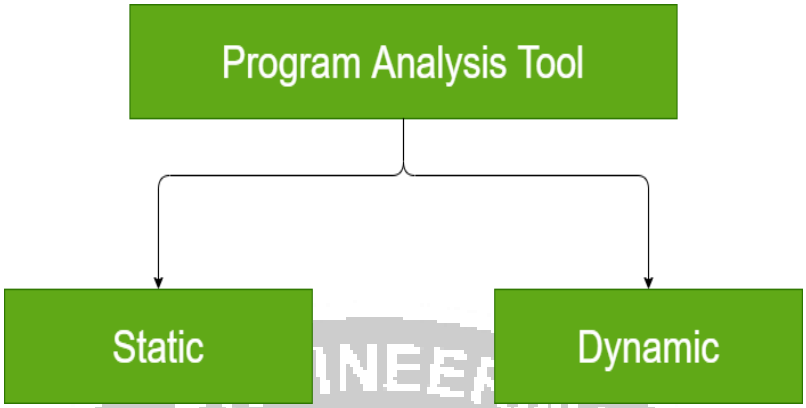
Improve and maintain software systems over the course of the whole development life cycle.

Importance of Program Analysis Tools

1. **Finding faults and Security Vulnerabilities in the Code:** Automatic programme analysis tools can find and highlight possible faults, security flaws and bugs in the code. This lowers the possibility that bugs will get it into production by assisting developers in identifying problems early in the process.
2. **Memory Leak Detection:** Certain tools are designed specifically to find memory leaks and inefficiencies. By doing so, developers may make sure that their software doesn't gradually use up too much memory.
3. **Vulnerability Detection:** Potential vulnerabilities like buffer overflows, injection attacks or other security flaws can be found using programme analysis tools, particularly those that are security-focused. For the development of reliable and secure software, this is essential.
4. **Dependency analysis:** By examining the dependencies among various system components, tools can assist developers in comprehending and controlling the connections between modules. This is necessary in order to make well-informed decisions during refactoring.
5. **Automated Testing Support:** To automate testing procedures, CI/CD pipelines frequently combine programme analysis tools. Only well-tested, high-quality code is released into production thanks to this integration, helping in identifying problems early in the development cycle.

Classification of Program Analysis Tools

Program Analysis Tools are classified into two categories:



1. Static Program Analysis Tools

Static Program Analysis Tool is such a program analysis tool that evaluates and computes various characteristics of a software product without executing it. Normally, static program analysis tools analyze some structural representation of a program to reach a certain analytical conclusion.

Basically some structural properties are analyzed using static program analysis tools. The structural properties that are usually analyzed are:

1. Whether the coding standards have been fulfilled or not.
2. Some programming errors such as uninitialized variables.
3. Mismatch between actual and formal parameters.
4. Variables that are declared but never used.

Code walkthroughs and code inspections are considered as static analysis methods but static program analysis tool is used to designate automated analysis tools. Hence, a compiler can be considered as a static program analysis tool.

2. Dynamic Program Analysis Tools

Dynamic Program Analysis Tool is such type of program analysis tool that require the program to be executed and its actual behavior to be observed. A dynamic program analyzer basically implements the code. It adds additional statements in the source code to collect the traces of program execution.

When the code is executed, it allows us to observe the behavior of the software for different test cases. Once the software is tested and its behavior is observed, the dynamic program analysis tool performs a post execution analysis and produces reports which describe the structural coverage that has been achieved by the complete testing process for the program.

For example, the post execution dynamic analysis report may provide data on extent statement, branch and path coverage

achieved. The results of dynamic program analysis tools are in the form of a histogram or a pie chart. It describes the structural coverage obtained for different modules of the program.

The output of a dynamic program analysis tool can be stored and printed easily and provides evidence that complete testing has been done. The result of dynamic analysis is the extent of testing performed as whitebox testing. If the testing result is not satisfactory then more test cases are designed and added to the test scenario. Also dynamic analysis helps in elimination of redundant test cases.