Program Debugging

TVET ICT Technician Level 6 (DICT)

Introduction

- It is a systematic process of spotting and fixing the number of bugs, or defects, in a piece of software so that the software is behaving as expected.
- Debugging is harder for complex systems in particular when various subsystems are tightly coupled as changes in one system or interface may cause bugs to emerge in another.
- Testing a program against a well-chosen set of input tests gives the programmer confidence that the program is correct.

Cont.

- During the testing process, the programmer observes input-output relationships, that is, the output that the program produces for each input test case.
- If the program produces the expected output and obeys the specification for each test case, then the program is successfully tested.
- But if the output for one of the test cases is not the one expected, then the program is incorrect -- it contains **errors** (or **defects**, or "**bugs**").

Cont.

- In such situations, testing only reveals the presence of errors, but doesn't tell us what the errors are, or how the code needs to be fixed.
- In other words, testing reveals the **effects** (or **symptoms**) of errors, not the **cause** of errors. The programmer must then go through a debugging process, to identify the causes and fix the errors.
- The debugging process may take significantly more time than writing the code in the first place.
- A large amount (if not most) of the development of a piece of software goes into debugging and maintaining the code, rather than writing it.

Cont.

- Therefore, the best thing to do is to **avoid the bug when you write the program in the first place!**
- It is important to sit and think before you code: decide exactly what needs to be achieved, how you plan to accomplish that, design the high-level algorithm cleanly, convince yourself it is correct, decide what are the concrete data structures you plan to use, and what are the invariants you plan to maintain.
- All the effort spent in designing and thinking about the code before you write it will pay off later.

Cont.

- The benefits are twofold.
  - ❖First, having a clean design will reduce the probability of defects in your program.
  - ❖Second, even if a bug shows up during testing, a clean design with clear invariants will make it much easier to track down and fix the bug.
- Programmers should always use defensive programming to avoid bugs.
- Defensive programming means developing code such that it works correctly under the worst-case scenarios from its environment.
  - For instance, when writing a function, one should assume worst-case inputs to that function, i.e., inputs that are too large, too small, or inputs that violate some condition;
- The code should deal with these cases, even if the programmer doesn't expect them to happen under normal circumstances.

Classes of Defects

Syntax or type errors. They are errors that arise as a result of violating syntax rules. These errors are always caught by the compiler, and reported via error messages.

- Typically, an error message clearly indicates the cause of error; for instance, the line number, the incorrect piece of code, and an explanation.
- Such messages usually give enough information about where the problem is and what needs to be done.
- In addition, editors with syntax highlighting can give good indication about such errors even before compiling the program.

Classes of Defects Cont.

Logical errors. They arise from flaws in the algorithm used.

- They are difficult to fix, therefore, one has to closely examine the algorithm, and try to come up with an argument why the algorithm works.
- Trying to construct such an argument of correctness will probably reveal the problem.
- A clean design can help a lot figuring out and fixing such errors. In fact, in cases where the algorithm is too difficult to understand, it may be a good idea to redo the algorithm from scratch and aim for a cleaner formulation.

Debugging Looks like this!

Challenges Encountered During Debugging

- ***The symptoms may not give clear indications about the cause.*** In particular, the cause and the symptom may be remote, either in space (i.e., in the program code), or in time (i.e., during the execution of the program), or both. Defensive programming can help reduce the distance between the cause and the effect of an error.
- ***Symptoms may be difficult to reproduce.*** Replay is needed to better understand the problem. Being able to reproduce the same program execution is a standard obstacle in debugging concurrent programs.

Challenges Encountered During Debugging

- ***Errors may be correlated.*** Therefore, symptoms may change during debugging, after fixing some of the errors. The new symptoms need to be re-examined. The good part is that the same error may have multiple symptoms; in that case, fixing the error will eliminate all of them.
- ***Fixing an error may introduce new errors.*** Statistics indicate that in many cases fixing a bug introduces a new one! This is the result of trying to do quick hacks to fix the error, without understanding the overall design and the invariants that the program is supposed to maintain. Once again, a clean design and careful thinking can avoid many of these cases.


when you trying
to fix your code
but ended up
making it worse

Steps Involved In Debugging

Steps Involved In Debugging

- **Identify the Error:** A bad identification of an error can lead to wasted developing time.
- It is usual that production errors reported by users are hard to interpret and sometimes the information we receive is misleading. It is import to identify the actual error.
- **Find the Error Location:** After identifying the error correctly, you need to go through the code to find the exact spot where the error is located.
- In this stage, you need to focus on finding the error instead of understanding it.

Steps Involved In Debugging

- **Analyze the Error:** In the third step, you need to use a bottom-up approach from the error location and analyze the code. This helps you in understanding the error.
- Analyzing a bug has two main goals, such as checking around the error for other errors to be found, and to make sure about the risks of entering any collateral damage in the fix.
- **Prove the Analysis:** Once you are done analyzing the original bug, you need to find a few more errors that may appear on the application. This step is about writing automated tests for these areas with the help of a test framework.

Steps Involved In Debugging

- **Cover Lateral Damage:** In this stage, you need to create or gather all the unit tests for the code where you are going to make changes. Now, if you run these unit tests, they all should pass.
- **6. Fix & Validate:** The final stage is the fix all the errors and run all the test scripts to check if they all pass.

A simple Summary of Debugging Process

The debugging stages can be broadly summarized into three processes:

1. Examine the error symptoms.
2. Identify the cause
3. Fix the error.

Debugging Strategies

- **Incremental and bottom-up program development.** One of the most effective ways to localize errors is to develop the program incrementally, and test it often, after adding each piece of code.
- It is highly likely that if there is an error, it occurs in the last piece of code that you wrote.
- With incremental program development, the last portion of code is small; the search for bugs is therefore limited to small code fragments.
- An added benefit is that small code increments will likely lead to few errors, so the programmer is not overwhelmed with long lists of errors.
- Bottom-up development maximizes the benefits of incremental development.

Debugging Strategies

- With bottom-up development, once a piece of code has been successfully tested, its behavior won't change when more code is incrementally added later.
- Existing code doesn't rely on the new parts being added, so if an error occurs, it must be in the newly added code (unless the old parts weren't tested well enough)

Debugging Strategies

- **Use debuggers.** If a debugger is available, it can replace the manual instrumentation using print statements or assertions.
- Setting breakpoints in the program, stepping into and over functions, watching program expressions, and inspecting the memory contents at selected points during the execution will give all the needed run-time information without generating large, hard-to-read log files.
- **Backtracking.** This option involves starting from the point where to problem occurred and go back through the code to see how that might have happened.

Debugging Strategies

- **Binary search.** The backtracking approach will fail if the error is far from the symptom.
- A better approach is to explore the code using a divide-and-conquer approach, to quickly pin down the bug.
- For example, starting from a large piece of code, place a check halfway through the code.
- If the error doesn't show up at that point, it means the bug occurs in the second half; otherwise, it is in the first half.
- Thus, the code that needs inspection has been reduced to half. Repeating the process a few times will quickly lead to the actual problem.

Debugging Strategies

- **Problem simplification.** A similar approach is to gradually eliminate portions of the code that are not relevant to the bug.
- For instance, if a function which has arguments (g, h, k) yields an error, try eliminating the calls to g, h, and k successively (by commenting them out), to determine which is the erroneous one.
- Then simplify the code in the body of buggy function, and so on. Continuing this process, the code gets simpler and simpler.
- The bug will eventually become evident. A similar technique can be applied to simplify data rather than code. If the size of the input data is too large, repeatedly cut parts of it and check if the bug is still present. When the data set is small enough, the cause may be easier to understand.

Debugging Strategies

- **Bug clustering.** If a large number of errors are being reported, it is useful to group them into classes of related bugs (or similar bugs), and examine only one bug from each class.
- The intuition is that bugs from each class have the same cause (or a similar cause).
- Therefore, fixing a bug with automatically fix all the other bugs from the same class (or will make it obvious how to fix them).

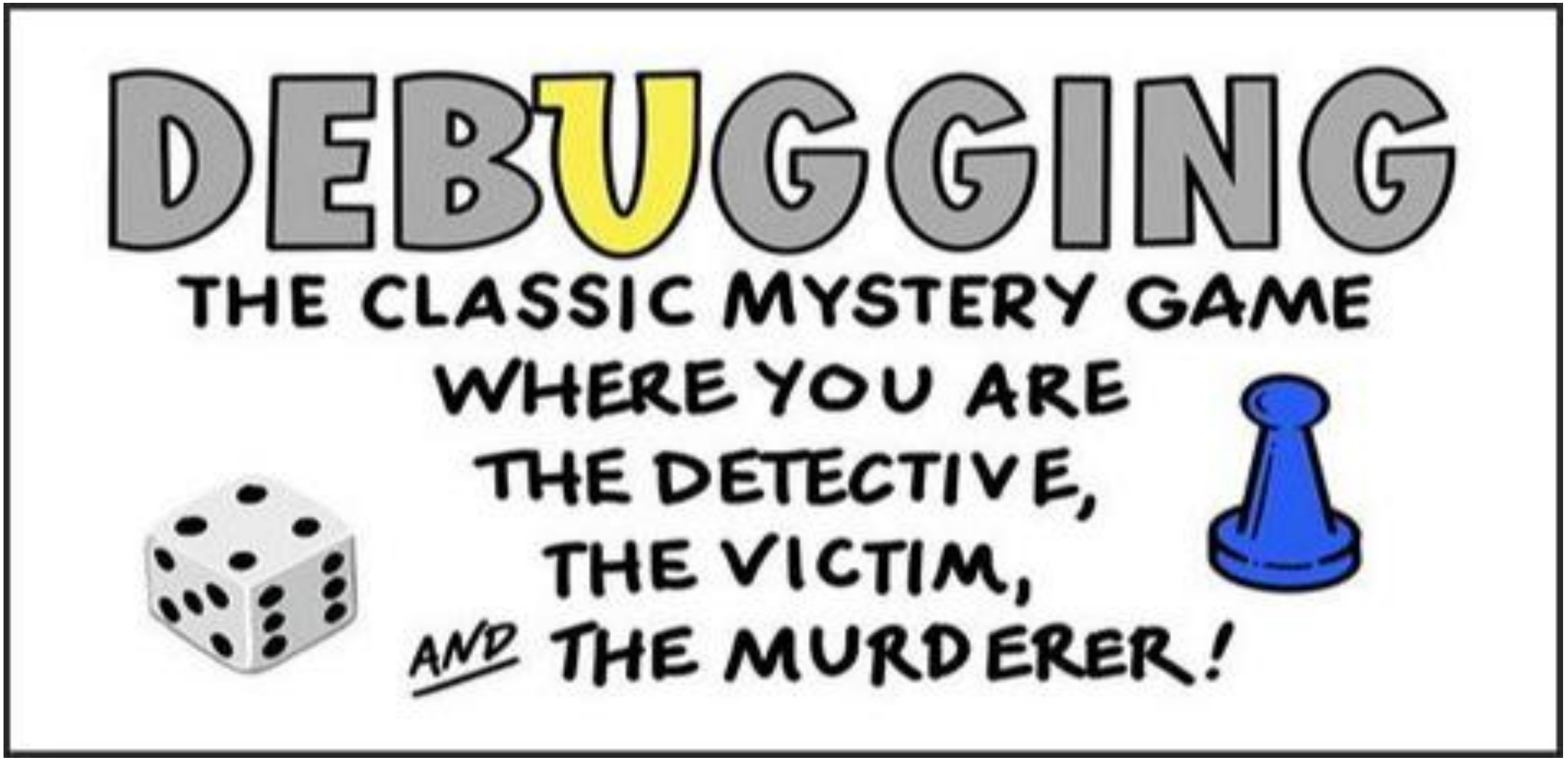
Debugging Strategies

- **Error-detection tools.** Such tools can help programmers quickly identify violations of certain classes of errors.
- For instance, tools that check safety properties can verify that file accesses in a program obey the open-read/write-close file sequence;
  - that the code correctly manipulates locks;
  - or that the program always accesses valid memory.
- Such tools are either dynamic (they instrument the program to find errors at run-time), or use static analysis (look for errors at compile-time).

Debugging Strategies

- For instance, **Purify** is a popular dynamic tool that instruments programs to identify memory errors, such as invalid accesses or memory leaks.
- Examples of static tools include **ESC Java** and **Spec#**, which use theorem proving approaches to check more general user specifications (pre and post-conditions, or invariants);
- Such tools can dramatically increase productivity, but checking is restricted to a particular domain or class of properties.
- Currently, there are relatively few such tools and this is more an (active) area of research.

The End


DEBUGGING
THE CLASSIC MYSTERY GAME
WHERE YOU ARE
THE DETECTIVE,
THE VICTIM,
AND THE MURDERER!

Software Maintenance

ICT Technician Level 6

What is software Maintenance?

- Software Maintenance is the process of modifying a software product after it has been delivered to the customer.
- The main purpose of software maintenance is to modify and update software applications after delivery to correct faults and to improve performance.
- 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.

Need for Software Maintenance

- Software maintenance must be performed in order to.
  - ✓Correct faults.
  - ✓Improve the design.
  - ✓Implement enhancements.
  - ✓Interface with other systems.
  - ✓Accommodate programs so that different hardware, software, system features, and telecommunications facilities can be used.
  - ✓Migrate legacy software.
  - ✓Retire software.

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:
  - ✓Preventative Software Maintenance
  - ✓Corrective Software Maintenance
  - ✓Perfective Software Maintenance
  - ✓Adaptive Software Maintenance

Preventive Software Maintenance

- Preventive maintenance involves performing activities to prevent the occurrence of errors.
- It tends to reduce the software complexity thereby improving program understandability and increasing software maintainability.
- It comprises documentation updating, code optimization, and code restructuring.
- Documentation updating involves modifying the documents affected by the changes in order to correspond to the present state of the system.
- Code optimization involves modifying the programs for faster execution or efficient use of storage space.

Cont. Preventive Maintenance

- Code restructuring involves transforming the program structure for reducing the complexity in source code and making it easier to understand.
- Preventive maintenance is limited to the maintenance organization only and no external requests are acquired for this type of maintenance.
- Preventive maintenance accounts for only 5% of all the maintenance activities.
- Preventative software maintenance is looking into the future so that your software can keep working as desired for as long as possible.

Types of Preventive Maintenance

Usage-based preventive maintenance

- Usage-based preventive maintenance is triggered by the actual utilization of an asset.
- This type of maintenance takes into account the average daily usage or exposure to environmental conditions of an asset and uses it to forecast a due date for a future inspection or maintenance task.

Calendar/time-based preventive maintenance

- Calendar/time-based preventive maintenance occurs at a scheduled time, based on a calendar interval.
- The maintenance action is triggered when the due date approaches and necessary work orders have been created.

Cont. Types of Preventive Maintenance

Predictive maintenance

- Predictive maintenance is designed to schedule corrective maintenance actions before a failure occurs.
- The team needs to first determine the condition of the equipment in order to estimate when maintenance should be performed.
- Then maintenance tasks are scheduled to prevent unexpected equipment failures.

Cont. Types of Preventive Maintenance

Prescriptive maintenance

- Prescriptive maintenance doesn’t just show that failure is going to happen and when, but also why it’s happening.
- This type of maintenance helps analyze and determine different options and potential outcomes, in order to mitigate any risk to the operation

Benefits of Preventive Maintenance

- **Extends asset life:** Systematically schedule maintenance and inspections to ensure assets achieve their full lifecycle and warranties are kept up to date.
- **Reduces maintenance:** Better insight into your operations and assets helps you make a significant reduction to maintenance costs.
- **Boosts productivity:** Preventive maintenance ensures that assets are always ready for use thus eliminating time wastage during breakdowns
- **Optimal functionality of software:** Good preventive maintenance ensures that every aspect of the software is working perfectly thus enhancing the usability of the software.

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).

Types of Corrective Maintenance: Planned Corrective Maintenance

There are two ways that corrective maintenance can be planned.

- Corrective maintenance is planned when a run-to-failure maintenance strategy is used.
  - This is when an asset is allowed to run until it breaks down and is then repaired or replaced.
  - This type of corrective maintenance only works with non-critical assets that are easily and cheaply repaired or replaced, or with systems that have redundancies.
- Corrective maintenance is planned when it’s performed as part of preventive maintenance or condition-based monitoring.
  - Both preventive and condition-based maintenance attempt to find problems before they cause equipment failure. If a problem is found, maintenance can be planned and scheduled.

Unplanned corrective maintenance

- Corrective maintenance is unplanned when a preventive maintenance schedule is in place, but a breakdown occurs between scheduled maintenance actions.
  - Maintenance can be may be performed immediately or at a later date, depending on the availability of tools, parts, and personnel.
- Corrective maintenance can also be unplanned when an asset shows signs of potential failure or reaches failure unexpectedly.
  - In this scenario, there are no planned maintenance actions to catch the failure before it happens or to address it after it happens.

When to use corrective maintenance?

- When preventive maintenance tasks identify potential faults
- When condition-based monitoring finds machine anomalies that signal potential failure
- When non-critical assets can be allowed to run to failure and are inexpensive and easy to repair or replace.
- When asset failure doesn’t affect safety
- When a system has redundancies that allow it to operate properly even if a part fails
- When an unplanned software failure occurs

Perfective Maintenance

- Perfective maintenance mainly deals with implementing new or changed user requirements.
- Perfective maintenance involves making functional enhancements to the system in addition to the activities to increase the system's performance even when the changes have not been suggested by faults.
- This includes enhancing both the function and efficiency of the code and changing the functionalities of the system as per the users' changing needs.

Cont. Perfective Maintenance

- Examples of perfective maintenance include modifying the payroll program to incorporate a new union settlement and adding a new report in the sales analysis system.
- Perfective maintenance accounts for 50%, that is, the largest of all the maintenance activities.
- 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

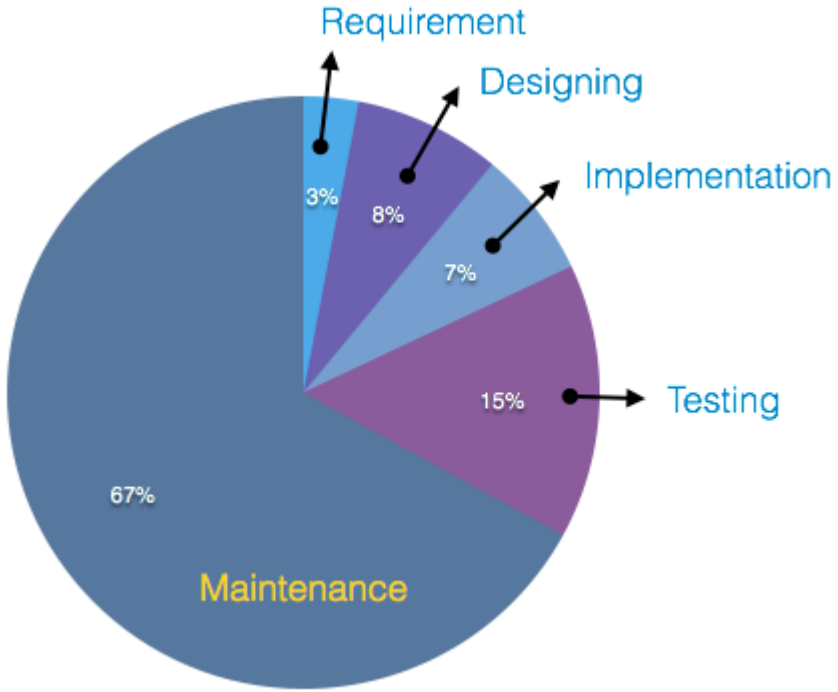
- Adaptive maintenance is the implementation of changes in a part of the system, which has been affected by a change that occurred in some other part of the system.
- Adaptive software maintenance has to do with the changing technologies as well as policies and rules regarding your software.
- These 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.
- Adaptive maintenance consists of adapting software to changes in the environment such as the hardware or the operating system.

Cont. Adaptive Software Maintenance

- The term environment in this context refers to the conditions and the influences which act (from outside) on the system.
- For example, business rules, work patterns, and government policies have a significant impact on the software system.
- Adaptive maintenance accounts for 25% of all the maintenance activities.

Cost of Software Maintenance

- Reports suggest that the cost of maintenance is high.
- A study on estimating software maintenance found that the cost of maintenance is as high as 67% of the cost of entire software process cycle.

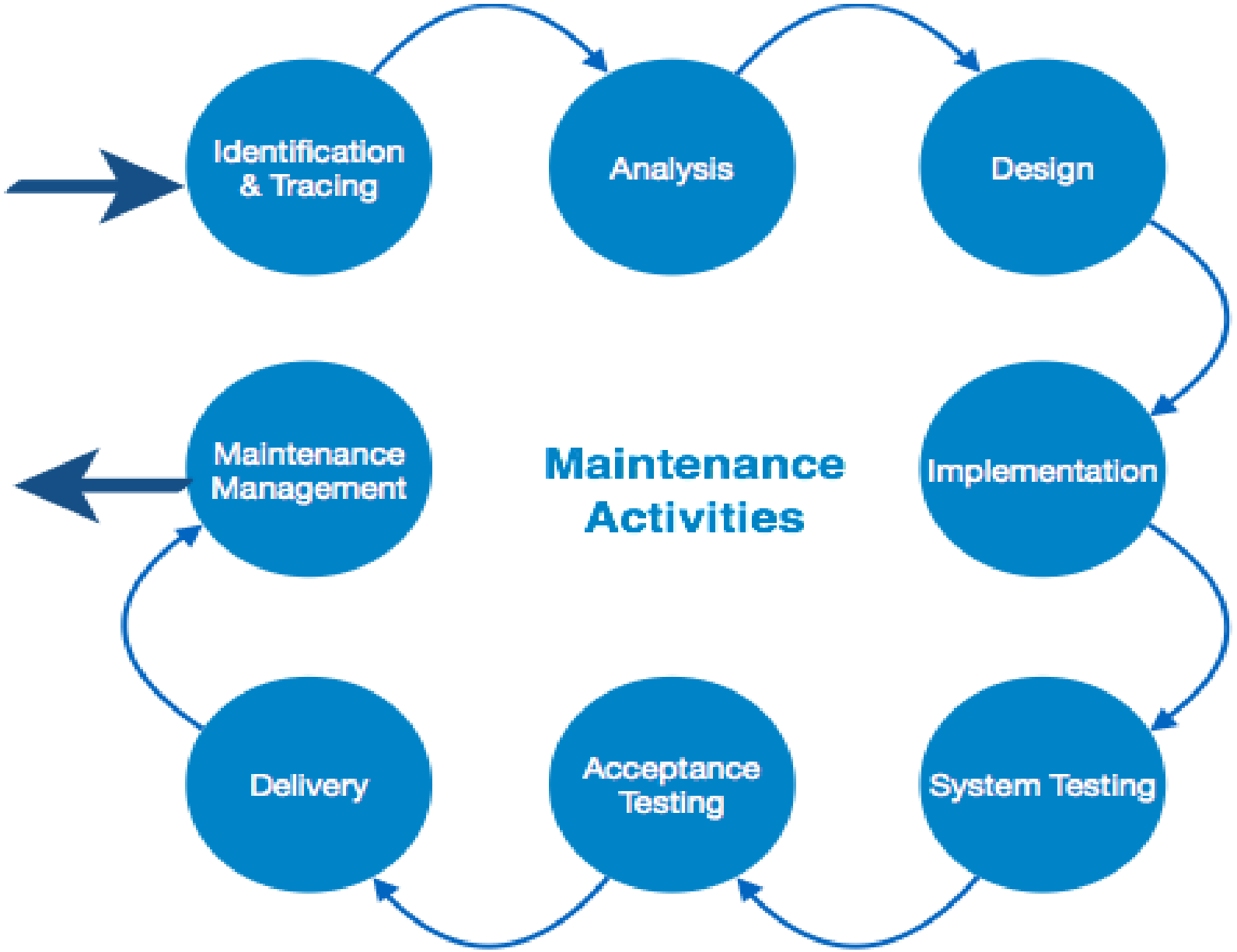


Real-world factors affecting Maintenance Cost

- The standard age of any software is considered up to 10 to 15 years.
- Older software, which were meant to work on slow machines with less memory and storage capacity cannot keep themselves challenging against newly coming enhanced software on modern hardware.
- As technology advances, it becomes costly to maintain old software.
- Most maintenance engineers are newbie and use trial and error method to rectify problem.
- Often, changes made can easily hurt the original structure of the software, making it hard for any subsequent changes.
- Changes are often left undocumented which may cause more conflicts in future.

Maintenance Activities

- Institute of Electrical and Electronic Engineering (IEEE) provides a framework for sequential maintenance process activities.
- It can be used in iterative manner and can be extended so that customized items and processes can be included.
- The activities of the maintenance process are shown I the figure below.


Identification & Tracing
Analysis
Design
Implementation
System Testing
Acceptance Testing
Delivery
Maintenance Management
Maintenance Activities

Cont.

- **Identification & Tracing** - It involves activities pertaining to identification of requirement of modification or maintenance.
- It is generated by user or system may itself report via logs or error messages. Here, the maintenance type is classified also.
- **Analysis** - The modification is analyzed for its impact on the system including safety and security implications.
- If probable impact is severe, alternative solution is looked for.
- A set of required modifications is then materialized into requirement specifications. The cost of modification/maintenance is analyzed and estimation is concluded.

Cont.

- **Design** - New modules, which need to be replaced or modified, are designed against requirement specifications set in the previous stage. Test cases are created for validation and verification.
- **Implementation** - The new modules are coded with the help of structured design created in the design stage. Every programmer is expected to do unit testing in parallel.
- **System Testing** - Integration testing is done among newly created modules. Integration testing is also carried out between new modules and the system. Finally the system is tested as a whole, following regressive testing procedures.

Cont.

- **Acceptance Testing** - After testing the system internally, it is tested for acceptance with the help of users. If at this state, user complaints some issues they are addressed or noted to address in next iteration.
- **Delivery** - After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system. The final testing takes place at client end after the software is delivered.
- Training facility is provided if required, in addition to the hard copy of user manual.
- **Maintenance management** - Configuration management is an essential part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.

Critical Questions

- How can software maintenance cost be reduced?
- Which do you think are the best practices in software maintenance?
- What is the role of programmers in software maintenance cost reduction?
- What are some of the ethical concerns that must be considered during software maintenance?
- What are some of the security issues/loopholes that may arise during maintenance?

Software Maintenance