Minimization and Prioritization of test cases in regression testing

In regression testing, minimization and prioritization are two distinct but complementary strategies used to optimize the testing process, saving time and resources while maximizing fault detection. Minimization permanently reduces the size of a test suite by eliminating redundant or obsolete test cases, while prioritization orders test cases to execute the most important ones first.

Test case minimization

The goal of test case minimization (or test suite reduction) is to reduce the size of a test suite by permanently removing redundant or obsolete test cases. This creates a smaller, more efficient suite that maintains the same level of code coverage.

Techniques for minimization:

- **Greedy algorithms:** These algorithms use a heuristic approach to find a minimal set of test cases that cover all testing requirements. They work by repeatedly selecting a test case that covers the most currently uncovered requirements until all requirements are met.
- **Integer Linear Programming (ILP):** This method uses a mathematical equation to find the smallest possible representative test set. While it can produce the most optimal result, it is also highly complex and resource-intensive.
- **Hybrid approaches:** Some techniques combine multiple methods. For example, a genetic algorithm might be used to find an optimal test suite that considers execution time and coverage, or a multi-objective approach might combine different criteria.
- **Call-stack coverage:** This technique focuses on reducing the test suite by ensuring that the remaining test cases cover the same unique call stacks as the original suite.

Considerations:

- A key risk of minimization is that a discarded test case might have been the one that could have detected a specific fault.
- The effectiveness of a minimization strategy should be measured by how much it reduces the test suite while preserving fault-detection capability.

Test Case Prioritization

test case prioritization refers to prioritizing test cases in the test suite based on different factors. Factors could be code coverage, risk/critical modules, functionality, features, etc.
Why should test cases be prioritized?

As the size of software increases, the test suite also grows bigger and requires more effort to maintain the test suite. To detect bugs in software as early as possible, it is important to prioritize test cases so that important test cases can be executed first.

Types of Test Case Prioritization :

- **General Prioritization:** In this type of prioritization, test cases that will be useful for the subsequent modified versions of the product are prioritized. It does not require any information regarding modifications made to the product.
- **Version-Specific Prioritization:** Test cases can also be prioritized such that they are useful on a specific version of the product. This type of prioritization requires knowledge about changes that have been introduced in the product.

Prioritization Techniques

1. Coverage-based Test Case Prioritization :

This type of prioritization is based on code coverage i.e. test cases are prioritized based on their code coverage.

- **Total Statement Coverage Prioritization -** In this technique, the total number of statements covered by a test case is used as a factor to prioritize test cases. For example, a test case covering 10 statements will be given higher priority than a test case covering 5 statements.
- **Additional Statement Coverage Prioritization -** This technique involves iteratively selecting a test case with maximum statement coverage, then selecting a test case that covers statements that were left uncovered by the previous test case. This process is repeated till all statements have been covered.
- **Total Branch Coverage Prioritization -** Using total branch coverage as a factor for ordering test cases, prioritization can be achieved. Here, branch coverage refers to coverage of each possible outcome of a condition.
- **Additional Branch Coverage Prioritization -** Similar to the additional statement coverage technique, it first selects a text case with maximum branch coverage and then iteratively selects a test case that covers branch outcomes that were left uncovered by the previous test case.
- **Total Fault-Exposing-Potential Prioritization -** Fault-exposing-potential (FEP) refers to the ability of the test case to expose fault. Statement and Branch Coverage Techniques do not take into account the fact that some bugs can be more easily detected than others and also that some test cases have more potential to detect bugs than others.
- **FEP depends on:**
  1. Whether test cases cover faulty statements or not.
  2. Probability that faulty statement will cause test case to fail.

2. Risk-based Prioritization

This technique uses risk analysis to identify potential problem areas which if failed, could lead to bad consequences. Therefore, test cases are prioritized keeping in mind potential problem areas. In risk analysis, the following steps are performed :

- List potential problems.
- Assigning probability of occurrence for each problem.
- Calculating the severity of impact for each problem.

After performing the above steps, a risk analysis table is formed to present results. The table consists of columns like Problem ID, Potential problem identified, Severity of Impact, Risk exposure, etc.

3. Prioritization using Relevant Slice

In this type of prioritization, **slicing technique** is used – when program is modified, all existing regression test cases are executed in order to make sure that program yields same result as before, except where it has been modified. For this purpose, we try to find part of program which has been affected by modification, and then prioritization of test cases is performed for this affected part. There are 3 parts to slicing technique :

- **Execution slice -** The statements executed under test case form execution slice.
- **Dynamic slice -** Statements executed under test case that might impact program output.
- **Relevant Slice -** Statements that are executed under test case and don't have any impact on the program output but may impact output of test case.

4. Requirements - based Prioritization

Some requirements are more important than others or are more critical in nature, hence test cases for such requirements should be prioritized first. The following factors can be considered while prioritizing test cases based on requirements :

- **Customer assigned priority -** The customer assigns weight to requirements according to his need or understanding of requirements of product.
- **Developer perceived implementation complexity -** Priority is assigned by developer on basis of efforts or time that would be required to implement that requirement.
- **Requirement volatility -** This factor determines frequency of change of requirement.
- **Fault proneness of requirements -** Priority is assigned based on how error-prone requirement has been in previous versions of software.

Minimization vs. prioritization: a comparison

Aspect	Test Case Minimization	Test Case Prioritization
Strategy	Permanently eliminate redundant test cases to reduce the overall size of the test	Reorder the execution sequence of all test cases to run the most important ones first.

	suite.	
Goal	Reduce the number of tests and execution time while maintaining a specific coverage level.	Maximize the rate of fault detection during the early stages of testing.
Test suite size	The overall test suite size is permanently reduced.	The size of the test suite is not reduced, but a subset may be selected for early execution.
Primary risk	A discarded test case could have been crucial for finding a future fault.	If testing is terminated early, lower-priority tests may not be executed, potentially missing less-critical bugs.
Application	Useful for reducing long-term maintenance costs for large test suites.	Valuable in fast-paced or agile projects with limited time for complete test execution.

How to use both strategies together

Minimization and prioritization are not mutually exclusive and can be used in combination for a highly efficient regression testing strategy.

1. **Perform initial minimization:** Use a minimization technique to permanently remove obsolete or highly redundant test cases from the test suite to reduce maintenance overhead.
2. **Prioritize the remaining suite:** For each regression test cycle, apply a prioritization technique (e.g., risk-based or version-based) to the minimized test suite.
3. **Execute the prioritized subset:** Run the most important test cases first. This ensures high-risk areas are tested early and provides faster feedback to developers.
4. **Decide on further execution:** Based on the results and available time, you can decide whether to continue executing lower-priority tests or stop testing once the primary objectives are met.