Selecting the Right Tests at the Right Time

A Guideline for Test Prioritization and Test Selection

TESTOMAT PROJECT
The Next Level of Test Automation

Version 1.2

Feedback of any kind is welcome! Contact us via Twitter @TestomatProject or using this **feedback** form: https://goo.gl/J5wnjm

Please take part in our **survey** on test prioritization! https://goo.gl/PCSpU6

CONTENT

Why Prioritize Your Tests Systematically?

The ideal situation for test is that you could test everything, but this is often not feasible for real-world systems. But it is often surprising how a statement like this, will allow major functionality to go untested, and communicates to people that you do not need to test (all) aspects (that you know). One must understand that it is feasible to cover almost everything at a decent cost, so aiming for zero-fault tolerance should be a goal for most test teams. By using prioritization and optimization techniques in a clever way - no unnecessary repetitions and waste of time is achieved. The more automated these processes are - the less time is spent on wasteful activities. Instead - Adding and complementing the test suite with new test cases addressing new aspects can be the main concern for the testers.

Prioritization of what you are testing can mean:

- **Importance:** To first make sure that within the time given, you have created tests for aspects of your software that are important to you

overall, or for a specific level or loop in the flow

- **Fast Feedback:** To reorder an existing test suite to increase its efficiency during test execution to e.g. do the changes to the software first, or the last bug-fixes first.
- **Execution schedule:** To choose when to run which tests at what time in the development and delivery cycle.
- **Clean-up:** To
- **Improve your Automation or Test-flow:** To select manual or semi-manual tests for full automation. One example could be to automate the test outcome and verdict, a difficult thing for non-functional testing that could involve several steps of post processing and analysis.

which allows you to...

- Save time, effort and cost
- Make sure you focus on what is important for you, your development speed, your users and customers
- Save on hardware utilization and test environment but not repeating unnecessary test cases

- Save on energy and power, be more sustainable to the environment
- Create a test suite that optimizes your test process (e.g. lower cost, less time, higher fault detection rate)
- Get an earlier feedback regarding faults or success e.g. when you have changed something or if you have very large test suites

The main goal is to avoid unnecessary repetition of test, e.g. if we already know the test will lead to a faulty verdict, it should not be repeated, or, if it is not expected to give any new information, e.g. nothing that impacts this particular test case has changed, it should not be re-executed.

In many modern systems, the traceability of the system to a test is lost and you are forced to re-test “everything”, - if you are also running in an agile way, this might mean you are really wasting resources. Here it could be beneficial to build up intelligence on what test cases are related to what particular code, and utilize learning to get “smart” test prioritization and scheduling schemes. It is simply smart to utilize your test suite as efficiently and effective as possible.

The different methods of test prioritization can be used and adapted to achieve various goals in various phases of the test

process. By using test prioritization methods like test case selection, removal, or scheduling, you will be able to improve the efficiency in your processes by reducing testing effort, time, and cost.

In the case of test case prioritization, which is a part of test case scheduling, a prioritized and reordered test suite allows you to increase the rate of fault detection in an early phase of testing without compromising on test coverage. This is especially interesting for large test suites, where an early feedback regarding faults is often beneficial.

Prioritizing Safety Critical code

It is noteworthy that when you optimize and prioritize testing using safety-critical code, it is often required by standards, e.g. ISO/IEC 26262 to state the reasons for how and in what way you did your test case selection, ordering and prioritization.

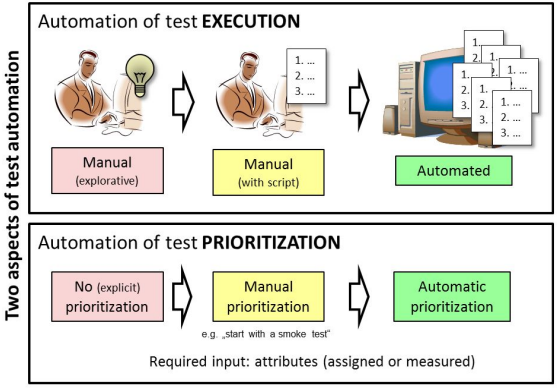

Figure 1. Two aspects of test automation

When to Prioritize Your Tests - in your Development & Test Process

Prioritization is done all the time, both with a long and short time-scale. When we talk about a long time, it could for a software systems entire life-cycle be years or decades, but the normal context for most software development and test activities it is done on a daily basis, and even every hour or “on demand”. This means that methods of test prioritization can be applied from the moment you design your requirements and choose what you are to test, from decisions to what test cases your are to automate, but most often we mean in the execution phase, or in the regression test phase: In other words, every time you run your tests during at every level of testing.

The absolute first prioritization is done when you are in the process of creating tests. How do you select what test cases should be written? What is the “best set” that would sufficiently test you software system given a specific quality criteria? The goal should be to test the best you can with the time given, and a multitude of techniques, ways of writing tests etc can aid you in that prioritization, which is called “Test Design”. We will not discuss these test design methods further in this booklet.

More important is to know that the earlier you start to plan for test prioritization in your testing process, the easier it will be to actually implement and use prioritization methods. To utilize test prioritization, it might be necessary to make changes to your existing test process, which means additional cost and man hours for implementation. Changing an existing test suite to allow for efficient prioritization can be a lot more work than having your test model optimized for prioritization from the beginning.

A very simple thing is that prioritized and main requirements with a corresponding key test cases, would from the beginning marked with “regression test” - since it must always work. One such example could be to prioritize test cases that traverses a large part of the system functionality, often called “End-to End test cases”.

So far, we only talked about when to start implementing test prioritization schemes. Another important topic is when to actually use your implemented test prioritization methods. This depends on various factors, for one the cost of utilizing it. Another factor that impacts cost is the actual size of the test suite. If you have thousands of developers adding code and test cases simultaneously - your prioritization will look very different than if you look at a small team owning one repository for one piece of software. Size matters.

Generally speaking, using a prioritization method will always be beneficial as soon as a full execution of an existing test suite will either be too expensive regarding cost or time. For example, if a full execution of a test suite takes several days and an early feedback regarding faults is necessary, it is beneficial to only execute the important test cases which are determined by the prioritization. A goal here could be to return information within minutes of a change - or for some systems within the hour. We all want answers immediately - but getting a result of a complex test suite, might be sometimes needed more computational time than available.

Alternatively, if the test suite is prioritized using adequate methods and metrics to increase the fault detection rate early in the testing process, the entire test suite can still be executed - but most of the faults will be reported in the first few hours of test execution (assuming that an efficient prioritization method has been chosen). A goal could be “small changes -fast feedback, big changes - takes a bit longer."

Prioritization Versus Development Methodologies

For software development different standard approaches exist in order to

achieve a certain quality. Depending on the project and its boundary conditions the waterfall approach, the Agile/Scrum approach, or the DevOps approach might be suiting. Regardless of the chosen approach a prioritization technologies are useful, as software systems is known to evolve over time.

Some Different Prioritizations

First you must decide what the key quality attributes are to know what to prioritize in your test of your system. It is often our first priority to test the functionality or features that our customer demands. This testing phase has its own requirements regarding the test process which are often different to those of later regression testing. Depending on where you are in your testing process (or which level of automation you have already reached), we can differentiate between different prioritization approaches:

1. Initial test prioritization (what requirements or aspects of requirements to prioritize
2. Regression testing or Retesting (when you have found a bug or done any slight change or update to your hardware or software in you system).
3. Automated prioritization for variable CI/CD testing - where you might have one too many “loops” of test case evaluation.

Each type has two major steps: The selection and the ordering of execution.

What to test?

The general question here is: What do I actually have to test? Meaning, we want to know what aspects we have to create test cases for; also called “defining the test scope”. This question can most of the time only be answered generally, because it really depends on what kind of software you are developing and what your customer requirements are.

There can be a variety of factors that influence your decision - but in most of the cases, it should at least satisfy the stakeholders.

Example of stakeholders are:

- Customers
- Users of systems
- Company strategy
- Economic situation

Most other aspects can be summed up as TEST CRITERIA:

- Requirements/User Stories (if it was important to describe in a requirement - it is important to test that it fulfills the requirement)
- Non-functional aspects (e.g. using quality attributes from the ISO/IEC std 25010)

- Necessity of certification and standardization adherence (there could be some necessity test that must be executed)

We could also assume we already have a test suite existing. If it is a manual set of test cases, we will be forced to select carefully among which of these tests we have time to test again, in addition to the re-test necessary for tests that revealed faults. Such an existing test suite can be prioritized using the same prioritization metrics that are used for prioritization type 3 (see chapter: “Type 3: Prioritization for your variable CI/CD testing”). However, the goal of testing (increasing fault detection rate, decreasing risk etc.) might be a different one.

Manual Test Case Selection for Regression Testing

If you have automated your testing process - this choice would appear trivial: You simply test everything again. When you work e.g. agile with a continuous build and test manner, and have automated most of your tests, you are likely to encounter these problems quickly:

1. The time to run all tests is too long
2. The number of tests is too high (for the time given)
3. It is a waste of energy and resources to rerun every test

4. That currently run tests are not related or important for the moment

Then you need to select among your automated suite of tests to choose in which order you want to test them.

Select Test cases for Regression Testing

As mentioned, this assumes that you are working manually, otherwise you would go for our next automated test selection (as the test case is already implemented).

Various prioritization criteria may be applied to the regression test suite. Test cases can be prioritized in terms of:

- random
- total or additional coverage[1]
- failure rates
- fault exposing potential

of the test cases. Test case prioritization can help with detecting high risk faults earlier in the testing life cycle, improving the detection of regression errors related to code changes, enhancing the coverage of the code and making a system reliable. Test case prioritization thus helps in

[1] Note, there are many different coverage, statement, branch/decision, basic condition, MC/DC, Mutation, Model, etc... check out our other TESTOMAT Booklet on Coverage to learn more about this.

reducing the effort and therefore time and cost of testing [7].

Manual prioritization also covers the prioritization process when deciding which test cases are automated first. This first selection can be done by choosing test cases that:

- "cover" as many different aspects of the system under test as possible (code, model, system architecture, modules, functionality, requirements, non-functional aspects, etc.)
- need re-testing (that found a fault in the earlier version) and still has to be confirmed as "fault-free"[2] (note that this option does not necessarily mean you should automate for it)
- you think has a high probability of finding a fault (e.g error guessing)

Automated Prioritization For Variable CI/CD Testing

Changing the software to correct faults or add new functionality can cause the existing functionality to regress, introducing new faults. To avoid such kind of defects, software can be retested after

[2] It is important to note that in software testing, tests can be used to show the presence of bugs, but never to show their absence!

modification which is known as regression testing. Regression testing is known to be one of the most expensive parts of software maintenance and therefore, it is necessary to prioritize test cases in test suites. Several techniques are available for prioritizing the test cases to accelerate the rate of fault detection in regression testing. Some existing approaches rely on requirement coverage. These approaches consider prioritization as an unordered, independent and one-time model. They do not take into account the performance of test cases [6].

When choosing a regression test selection, a goal oriented approach is generally a good idea. Here, a test selection for regression testing can be chosen according to one or more of the following goals:

- Increase fault detection rate
- Decrease risk of faults
- Decrease testing time
- Decrease cost of testing
- Feature-based prioritization focused on testing of current development
- A mix of those described by an optimization problem: maximize test coverage for a limited available testing time

An automated prioritization of your regression testing could be based on one or several of the following prioritization metrics:

- Coverage (additional) based on
  - Configuration
  - Test Environment
  - System architecture, structure
  - Customer prioritization
  - Code Coverage in case of white box testing [1]
  - Model Coverage in case of black box testing
  - N-F Quality Attributes
- Time for execution of TC [8]
- Type of TC (Level, or specific quality attribute)
- Resources required
  - Availability of Test Environment (context) needed
- History of TC
  - Latest 1- x test results (pass/fail) [4]
  - How long ago a TC was last executed
  - When a TC was introduced (age)
- Code Churn (based on latest changes of the code) [19]
  - Dependability analysis
  - Static Analysis
  - Build information
- Dependency in group, If verdict/results reacts “together” in a group - only select one in a group of “similar” TC
- Code complexity or model complexity
- Requirements

    - Requirement changes [2]
    - Requirement complexity [2]
- Risk-based metrics [3]
- Error-Probability-based metrics
    - Based on Bayesian networks [5]
- Other constraints - e.g. goal to fulfill

Example of such selection criteria for different ordering or group's are:

- Time/Duration to execute the Test suite overall (or a specific TC)
- "group", e.g. Smoke tests, Functional tests, Non-functional tests

How Do You Implement Test Prioritization?

Implementing an effective and robust test prioritization process that doesn't jeopardize the software quality in the long term requires a certain amount of planning and a clear definition of prioritization goals. The benefit of prioritization however will outweigh the initial cost of setup by reducing test execution time and cost for regression testing. Comparing testing with an industrial process, parallels to lean manufacturing can be drawn. Instead of running all the tests all the time, test prioritization tries to identify those tests that add value (by finding faults and aiding in improving software quality) and reduce test cases which add only minimal or no

value at all. Comparing test prioritization with just-in-time manufacturing, it can be argued that it might be more beneficial to run only those tests at a time, which are actually necessary for reaching a certain testing goal.

While concrete prioritization methods differ a lot based on which testing goals they were designed for, often a first prioritization-like step can be identified. Prioritization helps in identifying all test cases which aid in reaching a certain testing goal. To give an example, imagine a test suite for which full execution takes a whole week. Prioritizing the test suite and running those test cases, which were identified as being the most important according to some metric first, will allow for a much earlier feedback to the testers. Test prioritization methods are therefore basically trying to identify the most important test cases based on some metric and rearrange the execution order of the test suite.

Consider the following example excerpted from [18] where a test suite is composed of 5 test cases (i.e., A, B, C, D, E). The ability of each test case to detect faults is presented in the following table.

Table 1. Fault Matrix for test cases

test	fault									
	1	2	3	4	5	6	7	8	9	10
A	x				x					
B						x	x			
C	x	x	x	x	x	x	x			
D					x					
E								x	x	x

Suppose we “randomly” sort test cases of the test suite in order A-B-C-D-E. We can measure how rapidly a test suite detects faults by using the weighted Average of the Percentage of Faults Detected (APFD). The values range from 0 to 100, while a higher APFD means faster fault detection rate. In this example, the APFD metric obtained from this “random” order is 50%. However, if the prioritized order is C-E-B-A-D, the obtained APFD metric is 84%. It can be observed that when the test suite is ordered based on suitable prioritization, faults are detected earlier.

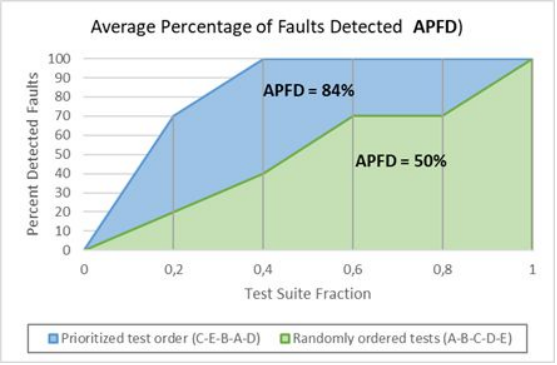

Figure 2. APFD-Graph for prioritized and randomly sorted test orders

A step further from simple prioritization methods are test selection techniques.

Instead of only reordering the test suite, the number of total test cases that are actually run is reduced by selecting only the most important test cases by some metric. A basic selection technique is designed by introducing restrictions to a prioritization method. The goal for a selection method could for example be to maximize test coverage with the least amount of test cases. Another goal could be to run test cases based on some metric derived from fault history until a certain testing coverage has been reached.

Test minimization techniques can be similar to test selection. However, for test minimization the goal is to identify test cases which are in some sense redundant or no longer needed and therefore removed from the test suite permanently.

A drawback of test selection is that before running a test, it is unknown if the test case will detect a fault. By omitting certain test cases, these faults will not be detected. To counter this problem, test selection can be combined with a test scheduling strategy. Basically, test scheduling defines at which times tests are run. In combination with a test selection method, this allows to run test cases that were e.g. selected by different prioritization metrics at different times. Additionally, full execution of the test suite can be planned at certain intervals to prevent missing certain faults completely.

Classification of Test Prioritization Methods

In the following, we classify test prioritization methods as discussed in the previous section. Starting with basic prioritization of tests, we discuss methods for deciding which test cases are better or more important according to some prioritization metric. We then continue with minimization and selection techniques which are usually based on some prioritization. Additionally, test scheduling based on different sets of selected test cases and also different quality levels is discussed.

Test Prioritization (Sorting)

Test case prioritization techniques involve scheduling over test cases in an order that improves the performance of regression testing. It is not efficient to re-execute every test case (a.k.a retest all [12]) for every program function if a change occurs, but this depends on the size of the test suite. Overall the concept of CI/CD test in agile methods assumes that all tests are to be repeated every time. In a more sustainable world, it is worth to save energy by not retesting unnecessarily. Test case prioritization techniques

organize the test cases in a test suite by ordering such that the most beneficial are executed first thus allowing for an increase in the effectiveness of testing. One of the goals is a measure of how quickly faults are detected during the testing process. In Test Case prioritization, each test case is assigned a priority. Priority is set according to some criterion and test cases with highest priority are scheduled first. For instance, criterion may be that the test case which has faster code coverage gets the highest priority [9]. A list of possible prioritization metrics is given in chapter 3 in this booklet. For ordering the test cases for a given criterion, several search algorithms can be used. Sorting is of course easy for simpler metrics, were you simply sort from highest to lowest value or reverse (e.g. Total coverage metrics). For some metrics, more advanced search algorithms might be useful (e.g. Additional coverage metric) [1].

Test minimization

Different techniques have been proposed in literature for test case minimization. The techniques include Heuristic H, GRE, and Divide and conquer approach, Genetic algorithm, selective redundancy, TestFilter, Integer Linear Programming based DILP, Cluster analysis, set theory etc. Many of them generated significant reduction in test suite, but it is harder to tell which one performs best. Heuristic

based approach produced significant reduction but less fault detection effectiveness. ILP based approach guaranteed minimal set but more complex and increased cost. For a technique to be efficient it should be good in both - reduced test suite size and improved fault detection efficiency [10]. Table shows the quick overview of common techniques:

Table 2. Test case minimization techniques [10]

Technique	Result
Heuristic H	Produced smaller size reduced set
Heuristic GRE	Produced optimal representative set
Concept Analysis of relation between test cases and requirements	Reduced sets were either same or less in size than greedy approach
Branch coverage and all-uses coverage obtained by data-flow analysis	Larger test suites but better fault detection capability
Statement coverage as weight	This can find redundant test cases and reduced cost
Dynamic call trees for reducing and prioritizing test cases	Constructing call trees increase 13% testing time
Combination of distribution-based and coverage-based	Reduced test suites with less fault detection loss

techniques	
Hybrid, multi objective genetic algorithm with greedy approach	More efficient testing decisions
Genetic algorithm	Produced optimal sized test-suite taking execution time and coverage factors into account
Set theory, Greedy algorithm	All requirements are covered, reduced set same as greedy

Genetic algorithm with time constraints	Reduced test suite and low running time

Test Selection

Test selection is an approach that aims at selecting a subset of test cases within a specific domain according to a certain test goal. They are key for the definition of an efficient test strategy that aims to eliminate redundant or unnecessary test cases and maximize fault detection for a test run, usually during regression testing. For example, test selection techniques can be used to reduce the effort of testing by reducing the number of test cases per test run and therefore reducing test time and cost while keeping the quality of testing at an acceptable level. However, unlike minimization techniques, test selection techniques do not aim to remove test cases permanently from the test suite. Both academia and industry have made a great effort to propose effective techniques for the selection of test cases. This effort has been reflected in numerous studies and reviews [12, 13, 14, 15, 16, 17]. Most of the selection methods found in literature are based on heuristics, which are not deterministic and based on experience or judgement of human beings. Therefore there is no guarantee that the adoption of a selection method build on certain assumptions of the heuristic approach will always result in an effective test case selection. The most usual heuristic approaches for test case selection are usually either based on random selection, coverage-based

selection or similarity-based selection. For random selection, the assumption is that random selection of test cases can result in an effective test that requires low computational cost. For coverage-based methods, the aim is to maximize test coverage with the least amount of test cases. Similarity-based methods are build on the assumption that the diversification of test cases tends to maximize the fault detection ability, because dissimilar test cases tend to reveal different faults in a program. Usually algorithms for test case selection are first prioritizing the given set of test cases for a test goal or test criteria (e.g. coverage, similarity, cost, risk, time based metrics) and then selecting test cases by introducing a certain restriction (e.g. cost, time, coverage, number of tc). Consequently, all test cases with lower priority are removed until the given restriction is met. However, other approaches for test case selection exist and their practicability is dependent on the test goal as well as the software domain they are applied in (Reactive systems, mobile devices, web services, embedded systems, GUI, etc.).

Usual methods for TCS found in literature are:

- Random TCS
- Adaptive Random TCS
- Genetics Algorithms for TCS
- Clustering for TCS
- Greedy-Algorithms for TCS

- Coverage Relationship Model based TCS
- Particle Swarm Optimization for TCS
- Pareto Efficient TCS
- Fuzzy Logic for TCS
- Model/Requirement based TCS

Additionally, test case selection is often optimized for the different flow stages, which is increased for more complex systems. Optimization is based based on a series of criterion at each point. E.g. if you have inner-loop testing, focus is on churn and coverage - also new test cases are given preference. Here a variety of technologies are used. E.g. if you have a traceable set of TC and you know exactly what test cases are affected (and should be selected for regression test) when you change a piece code, test selection will be trivial. For more complex test selection criteria and test goals or if traceability of TC is not given, test case selection becomes a less trivial problem.

Test Scheduling in different “loops” or test levels (Quality levels) e.g. in an Agile Development Flow

When you have larger software system, with more integration points, with using and testing dependent of different configurations (set up), phases or levels of test. In a CI/CD continuous development, build and

test. In this context, there are often several “levels” of tests that is “hidden”. Ways to manage this are e.g. separation of e.g. “inner loop” tests (unit or structural low level testing). When testing on a short, inner loop way, the essential aspect is to get feedback for the developer that has just created and submitted code - to check if it fulfilled the quality. In a CI/CD tool chain, the aim is to qualify and add test cases at different levels, e.g. functional and use case testing can then be added, as well as specific system test cases. Early levels are often simulated to add speed to the process and support fast feedback. Some system test aspects might only make sense to test in real environment, or in a semi-simulated, environment. Testing virtually/or simulated can be very efficient. Usually in an Agile automated CI/CD flow, the first level is to ensure (qualify, test) is that the code churn (the changed part of the code) has not broken the system. If using TDD you submit the test cases first, and then write code to “pass” them - Testing as a “requirement specification” (which still means you need to complement your test suite. You want your test prioritization to select your suite for fast d, and for most developers the main goal is to check that your testing has not altered anything else, e.g. e.g. legacy code, has not been broken. Assuring a specific change has not altered anything else can be very

difficult, in modern parallel, distributed systems.

Test prioritization in industry

Ericsson/Telecommunication

Prioritization Problem

As Ericsson test suites are many, and at many levels of tests - often with test flow integrating many products, serving a multitude of agile teams - In summary, Ericsson is executing and thus managing millions of test cases. For reasons as said in this booklet, fast feedback, finding faults early, saving cost and energy, since we are testing often, prioritization and smart scheduling is a necessity at every level of test in the DevOps flow.

Prioritization Metrics and Goals

Ericsson focuses on machine learning based multi-objective algorithms for test prioritization, which differ in the different loops in the CI flow. On lower level, giving feedback to the developer on the latest changes are prioritized - specifically focused if it breaks the existing build. In later stages of testing, prioritization could be on specific test beds. Overall, the goal is to have efficient, minimal test cases, that runs regularly, but that all test cases are travers regularly in the flow.

Prioritization Methods and Tools

Ericsson has a set of proprietary tools and algorithms to optimize the flow. Most of these tools are using machine-learning algorithms, with multi-objective goals, varying at the stage of testing, level, and flow.

Tools on prioritization must be combined with different cost aspects, e.g. configurations and hardware. Extensive work is done to optimize configurations and combinations of equipment. In TESTOMAT Project, we have made an overview of successful and less successful approaches - and tried to create a better roadmap of how to optimize our different types of selection, scheduling and prioritization. Where the most difficult prioritization is take place up front: To provide sufficient test cases to cover requirements “in full” and in all aspects of all characteristics.

References:

[1] S. Elbaum, A. Malishevsky and G. Rothermel, "Test Case Prioritization: A Family of Empirical Studies," in *IEEE Transactions on Software Engineering, Vol. 28, No. 2*, 2002.

[2] H. Srikanth, L. Williams and J. Osborne, "System Test Case Prioritization of New and Regression Test Cases," in *2005 International Symposium on Empirical Software Engineering*, Raleigh, NC, 2005.

[3] H. Stahlbaum, A. Metzger and K. Pohl, "An Automated Technique for Risk-based Test Case Generation and Prioritization," in *AST '08 Proceedings of the 3rd international workshop on Automation of software test*, Leipzig, 2008.

[4] C. T. Lin, C. D. Chen, C. S. Tsai and G. M. Kapfhammer, "History-based Test Case Prioritization with Software Version Awareness," in *18th International Conference on Engineering of Complex Computer Systems*, Singapore, 2013.

[5] S. Mirarab , L. Tahvildari , A prioritization approach for software test cases based on Bayesian networks, Fundam. Approaches Softw. Eng. Springer Berlin Heidelberg 4422 (2007) 276–290 .

[6] A. Khalilian, M.A. Azgomi, Y. Fazlalizadeh, "An improved method for test case prioritization by incorporating historical test case data," *Science of Computer Programming Vol. 78, (2012), pp.93–116.*

[7] A. Ansari, A. Khan, A. Khan, K. Mukadam, "Optimized Regression Test using Test Case Prioritization," Procedia Computer Science, Vol. 79 (2016), pp. 152 – 160.

[8] Kristen R. Walcott, Mary Lou Soffa, Gregory M. Kapfhammer, and Robert S. Roos. Timeaware test suite prioritization. In Proceedings of the 2006 International Symposium on Software Testing and Analysis, ISST*A '06, pages 1--12, New York, NY, USA, 2006. ACM.*

[9] Srivastava, P. R. (2008). "Test case prioritization," Journal of Theoretical and Applied Information Technology , vol. 4, pp. 178-181.

[10] Singh R. and Santosh M. (2013. "Test Case Minimization Techniques: A Review," International Journal of Engineering Research & Technology (IJERT), Vol. 2, pp. 1048-1056.

[11] Z. Li, M. Harman und R. M. Hierons, "Search Algorithms for Regression Test Case Prioritization," in *IEEE Transactions on Software Engineering, vol. 33, no. 4*, 2007

[12] E. Engström, P. Runeson, M. Skoglund, "A systematic review on regression test selection techniques" in *Information and Software Technology*, Volume 52, Pages 14-30, 2010

[13] T.L. Graves, M.J. Harrold, J-M. Kim, A. Porter, G. Rothermel, "An empirical study of regression test selection techniques" in *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Volume 10, Pages 184--208, 2001

[14] S. Yoo, M. Harman, “Regression testing minimization, selection and prioritization: a survey“ in *Software Testing, Verification and Reliability*, Volume 22, Pages 67--120, 2012
[15] S. Biswas, R. Mall, M. Satpathy, S. Sukumaran, “Regression test selection techniques: A survey“ in *Informatica*, Volume 35, 2011
[16] R.H. Rosero, O.S. Gomez, and G. Rodriguez, “15 years of software regression testing techniques—A survey“ in *International Journal of Software Engineering and Knowledge Engineering*, Volume 26, Pages 675--689, 2016
[17] R. Kazmi, D. Jawawi, R. Mohamad, I. Ghani, “Effective regression test case selection: A systematic literature review“ in *ACM Computing Surveys (CSUR)*, Volume 50, 2017
[18] S. Elbaum, A. Malishevsky, G. Rothermel, “Incorporating varying test costs and fault severities into test case prioritization“ in *Proceedings of the 23rd International Conference on Software Engineering*, Pages 329--338, 2001
[19] M. Campbell, K. Martin, F. Bozóki and M. Atkinson, "Dynamic Test Selection Using Source Code Changes," in *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, Prague, 2017, pp. 597-598.

Further Reading

[1] B. Jian, W.K. Chan, “Input-based adaptive randomized test case prioritization: A local beam search approach,” *Journal of Systems and Software, 105, (2015), pp. 91-106.*
[2] S. Sharma, P. Gera, “Test Case Prioritization in Regression Testing using Various Metrics,” *International Journal of Latest Trends in Engineering and Technology, Vol. 4 Issue 2, 2014.*
[3]R. Pradeepa, K. VimalaDevi, “Effectiveness of Testcase Prioritization using APFD Metric: Survey,” *International Conference on Research Trends in Computer Technologies (ICRTCT - 2013).*

Abbreviations & Terminology

CI = Continuous Integration
N-F = Non-Functional (or Extra-functional) test (based on quality attributes listed other than Functional Suitability in ISO/IEC Standard 25010)
SUT = System Under Test

TC = Test Case or Test Cases
TCS = Test Case Selection

This booklet was produced by a research collaboration between the following partners:

AKKA

ERICSSON

exploe

ifak

ALERION

Mondragon Unibertsitatea

PARASOFT

Acknowledgements:
This booklet is produced by EUREKA ITEA3 TESTOMAT PROJECT
The Next Level of Test Automation
Find out about us on the web:
https://www.testomatproject.eu/

Follow us on Twitter @Testomatproject

The Testomat Project is sponsored by:

MINISTERIO DE ECONOMÍA, INDUSTRIA Y COMPETITIVIDAD

CDTI Centro para el Desarrollo Tecnológico Industrial

TÜBİTAK

AGENTSCHAP INNOVEREN & ONDERNEMEN

BUSINESS FINLAND

VINNOVA

Bundesministerium für Bildung und Forschung

Disclaimer: The content of this booklet is true to the best of our current knowledge. The authors, publishers, participating partners of the project as well as the funding agencies disclaim any liability in connection to use of this information.