Software Testing

7

Abstract

This chapter is concerned with software testing and discusses the various types of testing that may be carried out during the project. We discuss test planning, test case definition, test environment set-up, test execution, test tracking, test metrics, test reporting and testing in an e-commerce environment.

Keywords

Test planning · Test case design · Unit testing · System testing · Performance testing · e-commerce testing · Acceptance testing · White box testing · Black box testing · Test tools · Test environment · Test reporting

7.1 Introduction

Testing plays a key role in verifying the correctness of software and confirming that the requirements have been correctly implemented. It is a constructive and destructive activity in that while on the one hand it aims to verify the correctness of the software, on the other hand it aims to find as many defects as possible in the software. The vast majority of defects (e.g. 80%) will be detected by software inspections in a mature software organization, with the remainder detected by the various types of testing carried out during the project.

Software testing provides confidence that the product is ready for release to potential customers, and the recommendation of the testing department is crucial in the decision as to whether the software product should be released or not. The test manager highlights any risks associated with the product, and these are considered prior to its release. The test manager and test department can be influential in an organization by providing strategic advice on product quality, and in encouraging

G. O'Regan, *Concise Guide to Software Engineering*, Undergraduate Topics in Computer Science, DOI 10.1007/978-3-319-57750-0_7

organization change to improve the quality of the software product through the use of best practice in software engineering.

The testers need a detailed understanding of the software requirements to enable them to develop appropriate test cases to verify the correctness of the software. Test planning commences at the early stages of the project, and testers play a role in building quality into the software product and verifying its correctness. The testers generally participate in the review of the requirements, and the testing viewpoint is important during the review to ensure that the requirements are correct and are testable.

The test plan for the project is documented (this could be part of the project plan or a separate document), and it includes the personnel involved, the resources and effort required, the definition of the testing environment to enable effective testing to take place, any special hardware and test tools required, and the planned schedule. There is a separate test specification plan for the various types of testing, and it records the test cases, including the purpose of the test case, the inputs and expected outputs and the test procedure for the particular test case.

Various types of testing are performed during the project, including unit, integration, system, regression, performance and user acceptance testing. The software developers perform the unit testing, and the objective is to verify the correctness of a module. This type of testing is termed "*white box*" testing and is based on knowledge of the internals of the software module. White box testing typically involves checking that every path in a module has been tested, and it involves defining and executing test cases to ensure code and branch coverage. The objective of "*black box*" testing is to verify the functionality of a module (or feature or the complete system itself), and knowledge of the internals of the software module is not required.

Test reporting is an important part of the project, and it ensures that all project participants understand the current quality of the software, as well as understanding what needs to be done to ensure that the product achieves the required quality criteria. The test status is reported regularly during the project, and once the tester discovers a defect, a problem report is opened, and the problem is analysed and corrected by the software developers. The problem may indicate a genuine defect, a misunderstanding by the tester or a request for an enhancement.

An *independent test group* is generally more effective than a test group that is directly reporting to the development manager. The independence of the test group helps to ensure that quality is not compromised when the project is under pressure to make its committed delivery dates. A good test group will play a proactive role in quality improvement, and this may involve participation in the analysis of the defects identified during testing phase at the end of the project, with the goal of prevention or minimization of the reoccurrence of the defects.

Real-world issues such as the late delivery of the software from the developers often complicate the software testing. Software development is challenging and deadline-driven, and missed developer deadlines may lead to compression of the testing schedule, as the project manager may wish to stay with the original schedule. There are risks associated with shortening the test cycle, as the testers

may be unable to complete the planned test activities. This means that insufficient data are available to make an informed judgment as to whether the software is ready for release, leading to risks that a defect-laden product may be shipped to the customer.

Test departments may be understaffed, as management may consider additional testers to be expensive and may wish to minimize costs. The test manager needs to be assertive in presenting the test status of the project, stating the known quality and risks, and the recommendation of the test manager needs to be carefully considered by the project manager and other stakeholders.

7.2 Test Process

The quality of the testing is dependent on the maturity of the test process, and a good test process will include test planning, test case analysis and design, test execution and test reporting. A simplified test process is sketched in Fig. 7.1, and the test process will include as follows:

- Test planning and risk management.
- Dedicated test environment and test tools.
- Test case definition.
- Test automation.
- Test execution.
- Formality in handover to test department.
- Test result analysis.
- Test reporting.
- Measurements of test effectiveness.
- Lessons learned and test process improvement.

Test planning consists of a documented plan defining the scope of testing and the various types of testing to be performed, the definition of the test environment, the required hardware or software for the test environment, the estimation of effort and resources for the various activities, risk management, the deliverables to be produced, the key test milestones and the test schedule.

The test plan is reviewed to ensure its fitness for purpose and to obtain commitment to the plan, as well as ensuring that all involved understand and agree to their responsibilities. The test plan may be revised in a controlled manner during the project. It is described in more detail in Sect. 7.3.

The test environment varies according to the type of business and project requirements. Large organizations may employ dedicated test laboratories, whereas a single workstation may be sufficient in a small organization. A dedicated test environment may require significant capital investment, but it will pay for itself in reducing the cost of poor quality, by identifying defects, and verifying that the software is fit for purpose.

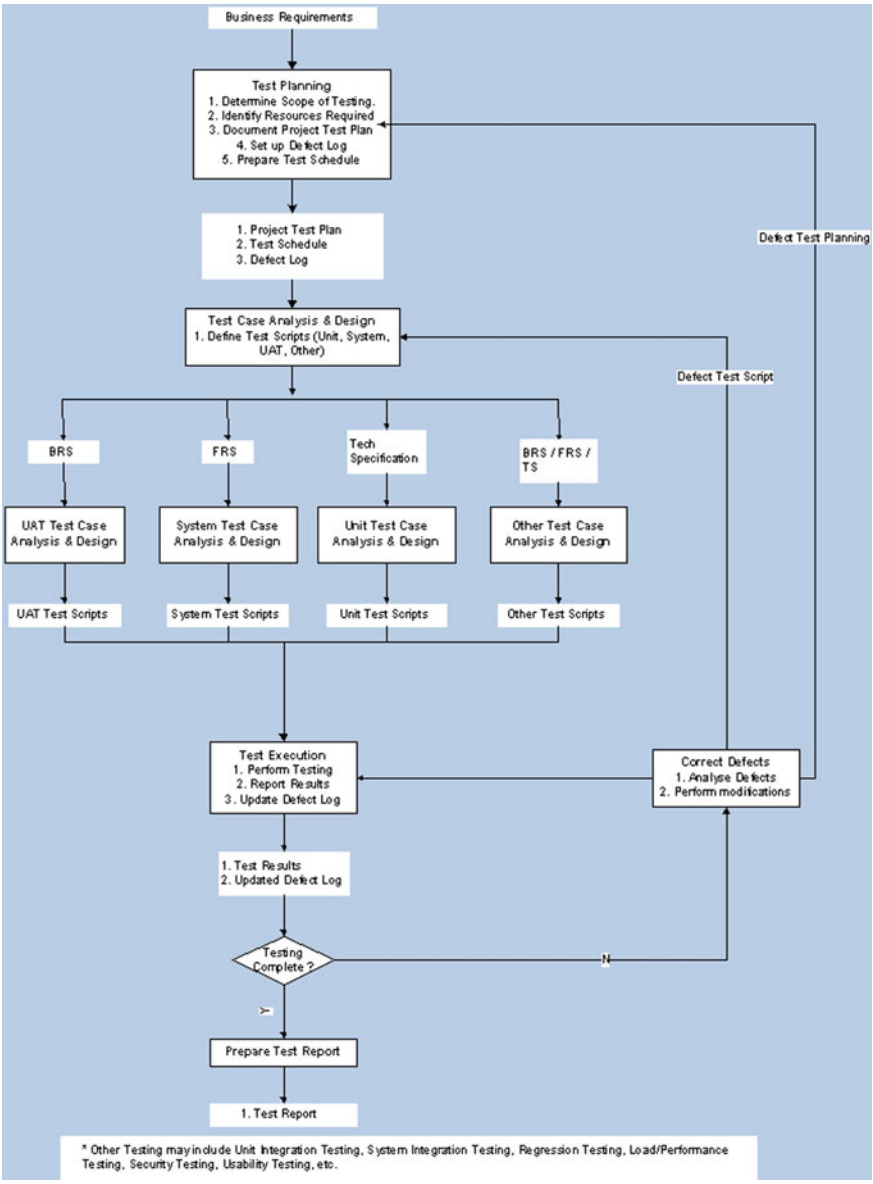

Fig. 7.1 Simplified test process

The test environment includes the hardware and software needed to verify the correctness of the software. It is defined early in the project so that any required hardware or software may be ordered in time. It may include simulation tools, automated regression and performance test tools, as well as tools for defect reporting and tracking.

The software developers produce a software build under configuration management control, and the build is verified for integrity to ensure that testing may commence. There is generally a formal or informal handover of the software to the test department, and a formal handover includes criteria that must be satisfied for the handover to take place. The test department must be ready for testing with the test cases and test environment prepared.

The various types of testing employed to verify the correctness of the software are described in Table 7.1. They may include:

The effectiveness of the testing is dependent on the definition of good test cases, which need to be complete in the sense that their successful execution will provide confidence in the correctness of the software. Hence, the test cases must relate or cover the software requirements, and we discussed the concept of a traceability matrix (that maps the requirements to the design and test cases) in Chap. 3 (Table 3.4). The traceability matrix provides confidence that each requirement has a corresponding test case for verification. The test cases will consist of a format similar to the following:

Table 7.1 Types of testing

Test type	Description
Unit testing	This testing is performed by the software developers, and it verifies the correctness of the software modules
Component testing	This testing is used to verify the correctness of software components to ensure that the component is correct and may be reused
System testing	This testing is (usually) carried out by an independent test group to verify the correctness of the complete system
Performance testing	This testing is (usually) carried out by an independent test group to ensure that the performance of the system is within the defined parameters. It may require tools to simulate clients and heavy loads, and precise measurements of performance are made
Load/stress testing	This testing is used to verify that the system performance is within the defined limits for heavy system loads over long or short periods of time
Browser compatibility	This testing is specific to web-based applications and verifies that the website functions correctly with the supported browsers
Usability testing	This testing verifies that the software is easy to use, and that the look and feel of the application is good
Security testing	This testing verifies that the confidentiality, integrity and availability requirements are satisfied
Regression testing	This testing verifies that the core functionality is preserved following changes or corrections to the software. Test automation may be employed to increase its productivity and efficiency
Test simulation	This testing simulates part of the system where the real system currently does not exist, or where the real live situation is hard to replicate
Acceptance testing	This testing carried out by the customer to verify that the software matches the customer's expectations prior to acceptance

– Purpose of the test case.
– Set-up required to execute the test case.
– Inputs to the test case.
– The test procedure.
– Expected outputs or results.

The test execution will follow the procedure defined in the test cases, and the tester will compare the actual results obtained with the expected results. The test completion status will be passed, failed or blocked (if unable to run at this time). The test results summary will indicate which test cases could be executed, which passed, which failed and which could not be executed.

The tester documents the test results including detailed information on the passed and failed tests. This will assist the software developers in identifying the precise causes of failure and the appropriate corrective actions. The developers and tester will agree to open a defect report in the defect-tracking system to track the successful correction of the defect.

The test status (Fig. 7.2) consists of the number of tests planned, the number of test cases run, the number that have passed and the number of failed and blocked tests. The test status is reported regularly to management during the testing cycle. The test status and test results are analysed and extra resources provided where necessary to ensure that the product is of high quality with all defects corrected prior to the acceptance of the product.

Test tools and test automation are used to support the test process and lead to improvements in quality, reduced cycle time and productivity. Tool selection (see Chap. 17) needs to be performed in a controlled manner, and it is best to identify the requirements for the tool first and then to examine a selection of tools to determine which best meets the requirements. Tools may be applied to test management and reporting, test results management, defect management and to the various types of testing.

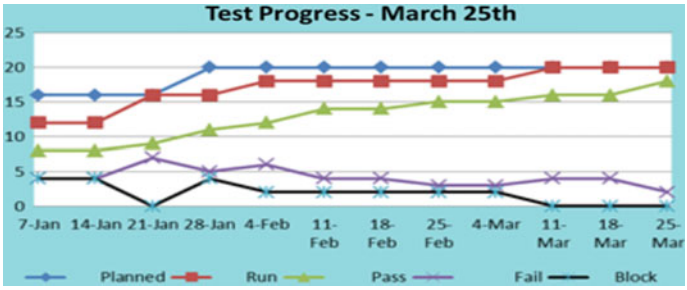

Fig. 7.2 Sample test status

A good test process will maintain measurements to determine its effectiveness, and an end of testing review is conducted to identify any lessons that need to be learned for continual improvement. The test metrics employed will answer questions such as:

- What is the current quality of the software?
- How stable is the product at this time?
- Is the product ready to be released at this time?
- What are the key risks and are they all managed?
- How good was the quality of the software that was handed over?
- How does the product quality compare to other products?
- How effective was the testing performed on the software?
- How many open problems are there and how serious are they?
- How much testing remains to be done?

7.3 Test Planning

Testing is a sub-project of a project and needs to be managed as such, and so good project planning and monitoring and control are required. The IEEE 829 standard includes a template for test planning, and test planning involves defining the scope of the testing to be performed, defining the test environment, estimating the effort required to define the test cases and to perform the testing, identifying the resources needed (including people, hardware, software and tools), assigning the resources to the tasks, defining the schedule, and identifying any risks to the testing and managing them.

The monitoring and control of the testing involves tracking progress and taking corrective action, replanning as appropriate where the scope of the testing has changed, providing test reports to give visibility of the test status to the project team (including the number of tests planned, executed, passed, blocked and failed), retesting corrections to the failed or blocked test cases, taking corrective action to ensure quality and schedule are achieved, managing risks and providing a final test report with a recommendation to go to acceptance testing. Test management involves as follows:

- Identify the scope of testing to be done.
- Determine types of testing to be performed.
- Estimates of time, resources, people, hardware, software and tools.
- Determine how test progress and results will be communicated.
- Define how test defects will be logged and reported.
- Provide resources needed.
- Definition of test environment.
- Assignment of people to tasks.

Table 7.2 Simple test schedule

Activity	Resource name(s)	Start date	End/Re-plan date	Comments
Review requirements	Test team	15.02.2017	16.02.2017	Complete
Project test plan/review	J. DiNatale	15.02.2017	28.02.2017	Complete
System test plan/review	P. Cuitino	01.03.2017	22.03.2017	Complete
Performance test plan/review	L. Padilla	15.03.2017	31.03.2017	Complete
Regression plan/review	P. Cuitino	01.03.2017	15.03.2017	Complete
Set-up test environment	P. Cuitino	15.03.2017	31.03.2017	Complete
System testing	P. Cuitino	01.04.2017	31.05.2017	In progress
Performance testing	L. Padilla	15.04.2017	07.05.2017	In progress
Regression testing	L. Padilla	07.05.2017	31.05.2017	In progress
Test reporting	J. DiNatale	01.04.2017	31.05.2017	In progress

- Define the schedule.
- Identify and manage risks.
- Track progress and take corrective action.
- Provide regular test status of passed, blocked, failed tests.
- Re-plan if scope of the project changes.
- Conduct post-mortem to learn any lessons.

Table 7.2 presents a simple test schedule for a small project, and the test manager will often employ Microsoft Project (or a similar scheduling tool) for planning and tracking of larger projects (e.g. Fig. 2.2). The activities in the test schedule are tracked and updated accordingly to record the tasks that have been completed, and dates are rescheduled as appropriate. Testing is a key sub-project of the main project, and the project manager will track the key test milestones and will maintain close contact with the test manager.

It is prudent to consider risk management early in test planning, to identify risks that could potentially arise during the testing, to estimate the probability of occurrence of the risk and its impact should it occur, and to identify (as far as is practical) actions to mitigate the risk or a contingency plan to address the risk if it materializes.

7.4 Test Case Design and Definition

Several types of testing that may be performed during the project were described in Table 7.1, and there is often a separate test plan for unit, system and UAT testing. The unit tests are based on the software design, the system tests are based on the

system requirements, and the UAT tests are based on the business (or user) requirements.

Each of these test plans contains test scripts (e.g. the unit test plan contains the unit test scripts), and the test scripts are traceable to the design (for the unit tests), and for the system requirements (for the system test scripts). The unit tests are more focused on white box testing, whereas the system test and UAT tests are focused on black box testing.

Each test script contains the objective of the test script and the procedure by which the test is carried out. Each test script includes as follows:

- Test case ID
- Test type (e.g. unit, system, UAT)
- Objective/description
- Test script steps
- Expected results
- Actual results
- Tested by

Regression testing involves carrying out a subset of the defined tests to verify that the core functionality of the software remains in place following changes to the system.

7.5 Test Execution

The software developers will carry out the unit and integration testing as part of the normal software development activities. The developers will correct any identified defects, and the development continues until all unit and integration tests pass, and the software is fit to be released to the test group.

The test group will usually be *independen*t (i.e. it has an independent reporting channel), and it will usually perform the system testing, performance testing, usability testing and so on. There is usually a formal handover from development to the test group prior to the commencement of testing, and the handover criteria need to be satisfied in order for the software to be accepted for testing by the test group.

The handover criteria will generally require that all unit and integration tests have been run and passed, that all known risks have been identified, that the test environment is ready for independent testing, and that the system, performance and all other relevant test scripts are available, and that all required resources required for testing are available.

Test execution then commences and the testers run the system tests and other tests, log any defects in the defect-tracking tool and communicate progress to the test manager. The test status is communicated to the project team, and the developers correct the identified defects and produce new releases. The test group retests

the failed and blocked tests and performs regression testing to ensure that the core functionality remains in place. This continues until the quality goals for the project have been achieved.

7.6 Test Reporting and Project Sign-Off

The test manager will report progress regularly during the project. The report provides the current status of testing for the project and includes as follows:

- Quality status (including tests run, passed and blocked).
- Risks and issues.
- Status of test schedule.
- Deliverables planned (next period).

The test manager discusses the test status with management and highlights the key risks and issues to be dealt with. The test manager may require management support to deal with these.

The test status is important in judging whether the software is ready to be released to the customer. Various quality metrics may be employed to measure the quality of the software, and the key risks and issues are considered. The test manager will make a recommendation to release or not based on the actual test status. One useful metric (one of many) is the cumulative arrival rate (Fig. 7.3) that gives an indication of the stability of the product.

The slope of the curve is initially steep as testing commences and defects are detected. As testing continues and defects are corrected and retested, the slope of the curves levels off, and over time the indications are that the software has stabilized and is potentially ready to be released to the customer.

However, it is important not to rush to conclusions based on an individual measurement. For example, the above chart could possibly indicate that testing halted on May 13th with no testing since then, and that would explain why the defect arrival rate per week is zero. Careful investigation and analysis needs to be done before the interpretation of a measurement is made, and usually several measurements rather than one are employed to make a sound decision.

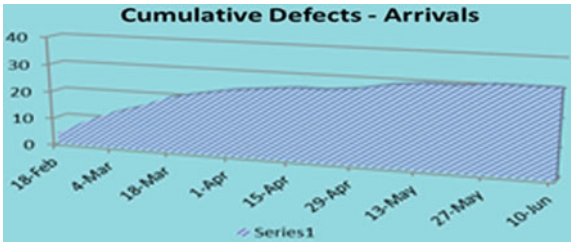

Fig. 7.3 Cumulative defects

7.7 Testing and Quality Improvement

Testing is an essential part of the software development process, and the recommendation of the test manager is carefully considered in the decision to release the software product. Decision-making is based on objective facts, and measurements are employed to assess the quality of the software. The open-problem status (Figs. 10.16 and 10.17), the problem arrival rate (Fig. 10.18) and the cumulative problem arrival rate (Fig. 7.3) give an indication of the quality and stability of the software product and may be used in conjunction with other measures to decide on whether it is appropriate to release the software, or whether further testing should be performed.

Test defects are valuable in the sense that they provide the organization the opportunity to improve its software development process to prevent the defects from reoccurring in the future. A mature development organization will perform internal reviews of requirements, design and code prior to testing. The effectiveness of the internal review process and the test process may be seen in the phase containment metric (PCE), which is discussed in Chap. 10.

Figure 10.19 indicates that the project had a phase containment effectiveness of approximately 54%. That is, the developers identified 54% of the defects, the system-testing phase identified approximately 23% of the defects, acceptance testing identified approximately 14% of the defects, and the customer identified approximately 9% of the defects. Many organizations set goals with respect to the phase containment effectiveness of their software. For example, a mature organization might aim for their software development department to have a phase containment effectiveness goal of 80%. This means that 80% of the defects should be found by software inspections.

The improvement trends in phase containment effectiveness may be tracked over time. There is no point in setting a goal for a particular group or area unless there is a clear mechanism to achieve the goal. Thus to achieve a goal of 80% phase containment effectiveness, the organization will need to implement a formal software inspection methodology as described in Chap. 6. Training on inspections will be required, and the effectiveness of software inspections was monitored and improved.

A mature organization will aim to have 0% of defects reported by the customer, and this goal requires improvements in its software inspection methodology and its software testing methodology. Measurements provide a way to verify that the improvements have been successful. Each defect is potentially valuable as it, in effect, enables the organization to identify weaknesses in the software process and to target improvements.

Escaped customer defects offer an opportunity to improve the testing process, as it indicates a weakness in the test process. The defects are categorized, causal analysis is performed, and corrective actions are identified to improve the testing process. This helps to prevent a reoccurrence of the defects. Thus, software testing plays an important role in quality improvement.

7.8 Traceability of Requirements

The objective of requirements traceability (as discussed in Chap. 3) is to verify that all of the requirements have been implemented and tested. One way to do this would be to examine each requirement number and to go through every part of the design document to find any reference to the particular requirement number, and similarly to go through the test plan and find any reference to the requirement number. This would demonstrate that the particular requirement number has been implemented and tested.

A more effective mechanism to do this is with a traceability matrix (Table 3.4). This may be a separate document or part of the test documents. The idea is that a mapping between the requirement numbers and the associated test cases is defined, and this provides confidence that all of the requirements have been implemented and tested.

A requirement number may map on to several test cases, i.e., the mapping may be one to many with several test cases employed to verify the correctness of a particular requirement. Traceability provides confidence that each requirement number has been implemented in the software design and tested via the test plan.

7.9 Test Tools

Test tools are employed to support the test process and are used to enhance quality, reduce cycle time and increase productivity. Tool selection needs to be planned, and the evaluation and selection of a particular tool involves defining the requirements for the proposed tool and identifying candidate tools to evaluate against the requirements. Each tool is then evaluated to yield an evaluation profile, and the results are analysed to enable an informed decision to be made. Tools to support the various software engineering activities (including testing) are described in Chap. 17.

There are various tools to support testing such as test planning and management tools, defect-tracking tools, regression test automation tools, performance tools. There are tools available from various vendors such as Compuware, Software Research, Inc., HP, LDRA, McCabe and Associates, and IBM Rational.

7.9.1 Test Management Tools

There are various test management tools available (e.g. the Quality Center tool from HP), and the main features of such a tool are as follows:

- Management of entire testing process.
- Test planning.
- Support for building and recording test scripts.

- Test status and reporting.
- Graphs for presentation.
- Defect control system.
- Support for many testers.
- Support for large volume of test data.
- Audit trail proof that testing has been done.
- Test automation.
- Support for various types of testing.

The Quality Center™ tool standardizes and manages the entire test and quality process, and it is a web-based system for automated software quality management and testing. It employs dashboard technology to give visibility into the process.

It provides a consistent repeatable process for gathering requirements, planning and scheduling tests, analysing results and managing defects. It supports a high level of collaboration and communication between the stakeholders. It allows the business analysts to define the application requirements and testing objectives. The test managers and testers may then design test plans, test cases and automated scripts. The testers then run the manual and automated tests, report results and log the defects.

The developers review and correct the logged defects. Project and test managers can create status reports and manage test resources. Test and product managers decide objectively whether the application is ready to be released.

7.9.2 Miscellaneous Testing Tools

There is a wide collection of test tools to support activities such as static testing, unit testing, system testing, performance testing and regression testing.

Code coverage tools are useful for unit testing, and, for example, the LDRA Testbed is able to analyse source files to report on areas of code that were not executed at run-time, thereby facilitating the identification of missing test data. Code coverage tools are useful in identifying the sources of errors, as they will typically show the code areas that were executed through textual or graphic reports.

Regression testing involves rerunning existing test cases to verify that the software remains correct following the changes made. It is often automated with capture and playback tools, and the Winrunner tool[1] that was developed by Mercury (now part of HP) captures, verifies and replays user interactions, and allows regression testing to be automated. Effort is required to set-up the tests for automation, but the payback is improvements in quality and productivity.

The purpose of performance testing is to verify that system performance is within the defined limits, and it requires measures on the server side, network side and client side (e.g. processor speed, disk space used, memory used,). It includes load testing and stress testing. Mercury's LoadRunner (now called HP Loadrunner)

[1]The Winrunner tool has been replaced by HP Unified Functional Testing Software.

tool allows the software application to be tested with hundreds or thousands of concurrent users to determine its performance under heavy loads. It allows the scalability of the software system to be tested, to determine whether can support the predicted growth.

The decision on whether to automate and what to automate often involves a test process improvement team. It tends to be difficult for a small organization to make a major investment in test tools (especially if the projects are small). However, larger organizations will require a more sophisticated testing process to ensure that high-quality software is consistently produced.

7.10 e-commerce Testing

There has been an explosive growth in electronic commerce, and website quality and performance is a key concern. A website is a software application, and so standard software engineering principles are employed to verify the quality of a website. e-commerce applications are characterized by:

- Distributed system with millions of servers and billions of participants.
- High availability requirements (24 * 7 * 365).
- Look and feel of the website is highly important.
- Browsers may be unknown.
- Performance may be unpredictable.
- Users may be unknown.
- Security threats may be from anywhere.
- Often rapid application development is required.
- Design a little, implement a little and test a little.
- Rapidly changing technologies.

The standard waterfall life cycle model is rarely employed for the front end of a web application, and instead, RAD/JAD/Agile models are employed. The use of lightweight development methodologies does not mean that anything goes in software development, and similar project documentation should be produced (except that the chronological sequence of delivery of the documentation is more flexible). Joint application development allows early user feedback to be received on the look and feel and correctness of the application, and the method of design a little, implement a little and test a little is valid for web development. The various types of web testing include as follows:

- Static testing.
- Unit testing.
- Functional testing.
- Browser compatibility testing.
- Usability testing.

- Security testing.
- Load/performance/stress testing.
- Availability testing.
- Post-deployment testing.

Static testing generally involves inspections and reviews of documentation. The purpose of static testing of websites is to check the content of the web pages for accuracy, consistency, correctness and usability, and also to identify any syntax errors or anomalies in the HTML. There are tools available (e.g. NetMechanic) for statically checking the HTML for syntax correctness.

The purpose of unit testing is to verify that the content of the web pages corresponds to the design, that the content is correct, that all the links are valid and that the web navigation operates correctly.

The purpose of functional testing is to verify that the functional requirements are satisfied. It may be quite complex as e-commerce applications may involve product catalogue searches, order processing, credit checking and payment processing, and the application may liaise with legacy systems. Also, testing of cookies, whether enabled or disabled, needs to be considered.

The purpose of browser compatibility testing is to verify that the web browsers that are to be supported are actually supported. The purpose of usability testing is to verify that the look and feel of the application is good and that web performance (loading web pages, graphics, etc.) is good. There are automated browsing tools which go through all of the links on a page, attempt to load each link and produce a report including the timing for loading an object or page. Usability needs to be considered early in design and is important in GUI applications.

The purpose of security testing is to ensure that the website is secure. The purpose of load, performance and stress testing is to ensure that the performance of the system is within the defined parameters.

The purpose of post-deployment testing is to ensure that website performance remains good, and this may be done as part of a service level agreement (SLA). A SLA typically includes a penalty clause if the availability of the system or its performance falls outside the defined parameters. Consequently, it is important to identify performance and availability issues early before they become a problem. Thus, post-deployment testing includes monitoring of website availability, performance and security and taking corrective action. e-commerce sites operate 24 h a day for 365 days a year, and major financial loss is incurred in the case of a major outage.

7.11 Test-Driven Development

Test-driven development (TDD) was developed by Kent Beck and others as part of extreme programming, and it ensures that test cases are written early with the software code written to pass the test cases. It is a paradigm shift from traditional

software engineering, where unit tests are written and executed after the code has been written.

The set of test cases is derived from the requirements, and the software is then written to pass the test cases. Another words, the test-driven development of a new feature begins with writing a suite of test cases based on the requirements for the feature, and the code for the feature is then written to pass the test cases.

Initially, all tests fail as no code has been written, and so the first step is to write some code that enables the new test cases to pass. This new code may be imperfect (it will be improved later), but this is initially acceptable as the only purpose is to pass the new test cases. The next step is to ensure that the new feature works with the existing features, and this involves executing all new and existing test cases.

This may involve modification of the source code to enable all of the tests to pass and to ensure that all features work correctly together. The final step is refactoring the code, and this involves cleaning up and restructuring the code. The test cases are rerun during the refactoring to ensure that the functionality is not altered in any way. The process repeats with the addition of each new feature. TDD is described in more detail in Chap. 18.

7.12 Review Questions

1. Describe the main activities in test planning.
2. What does the test environment consist of? When should it be set-up?
3. Explain the traceability of the requirements to the test cases?
4. Describe the various types of testing that may be performed.
5. Investigate available test tools to support testing? What areas of testing do they support and what are their benefits?
6. Describe an effective way to evaluate and select a test tool.
7. What are the characteristics of e-commerce testing that make it unique from other domains.
8. Discuss test reporting and the influence of the test manager in project sign-off.
9. Explain test-driven development.

7.13 Summary

This chapter discussed software testing and how testing may be used to verify that the software is of a high quality and fit to be released to potential customers. Testing is both a constructive and destructive activity, in that while on the one hand it aims to verify the correctness of the software, on the other hand it aims to find as many defects as possible.

Various test activities were discussed including test planning, setting up the test environment, test case definition, test execution, defect reporting, and test management and reporting.

We discussed black box testing and white box testing, unit and integration testing, system testing, performance testing, security and usability testing. Testing in an e-commerce environment was considered.

Test reporting enables all project participants to understand the current quality of the software and to understand what needs to be done to ensure that the product meets the required quality criteria.

Various tools to support the testing process were discussed, and a methodology to assist in the selection and evaluation of tools is essential. Metrics are useful in providing visibility into test progress and into the quality of the software. The role of testing in promoting quality improvement was discussed.

Testing is often complicated by the late delivery of the software from the developers, and this may lead to the compression of the testing schedule. The recommendation of the test manager on whether to release the product needs to be carefully considered.