# Fundamentals of Software Testing

# 3

**Key Topics**

Test planning
Test case design
Unit testing
System testing
Performance testing
Psychology of software tester
Acceptance testing
White box testing
Black box testing
Test tools
Test environment
Test reporting

## 3.1 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%) are detected by software inspections in a mature software organization, with the remainder detected by the



G. O'Regan, *Concise Guide to Software Testing*,
Undergraduate Topics in Computer Science,
https://doi.org/10.1007/978-3-030-28494-7_3

various types of testing carried out during the project. Testing has been defined in TMaps (2004) as:

> Testing is a process of planning, preparing, executing, and analysing, aimed at establishing the characteristics of an information system and demonstrating the differences between the actual status and the required status.

Software testing involves defining the test conditions and designing the test cases and then executing the test cases. This is followed by analysis and reporting of the results. Software testing provides confidence that the product is ready for release to potential customers, and the recommendation of the testing department plays a key role in the decision on whether to release the software product or not. The test manager highlights any risks associated with the product, and these are carefully considered to ensure that they can be managed on release. The test manager and test department are influential in an organization by providing strategic advice on product quality and in encouraging organization change to improve the quality of the software product through the use of best practice in software and system engineering.

The testers need a detailed understanding of the software requirements 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 through software inspections. The testers will generally participate in the review of the requirements, as the testing viewpoint is important in ensuring that the requirements are correct and 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 scope of the testing, the personnel involved, the resources and effort required, the key milestones, the definition of the test environment, any special hardware and test tools required, and the planned test schedule. There is a separate test specification plan for the various types of testing, which records the test cases, including the purpose of each test case, the inputs and expected outputs, and the test procedure for the execution of the particular test case.

Several 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 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 desired 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 more effective than a test group that is directly reporting to the development manager. The independence of the test group helps ensuring 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 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 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 project schedule due to commitments to the stakeholders. There are risks associated with shortening the test cycle, as the testers may be unable to complete the planned test activities. This means that there may be insufficient data 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. The test manager needs to be assertive in presenting the test status of the project and in clearly communicating the quality and associated risks. The recommendation of the test manager needs to be carefully considered by the project manager and other stakeholders prior to the release of the software.

## 3.2 Software 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. 3.1, and it will typically include:

– 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
– Test management
– Lessons learned and test process improvement.

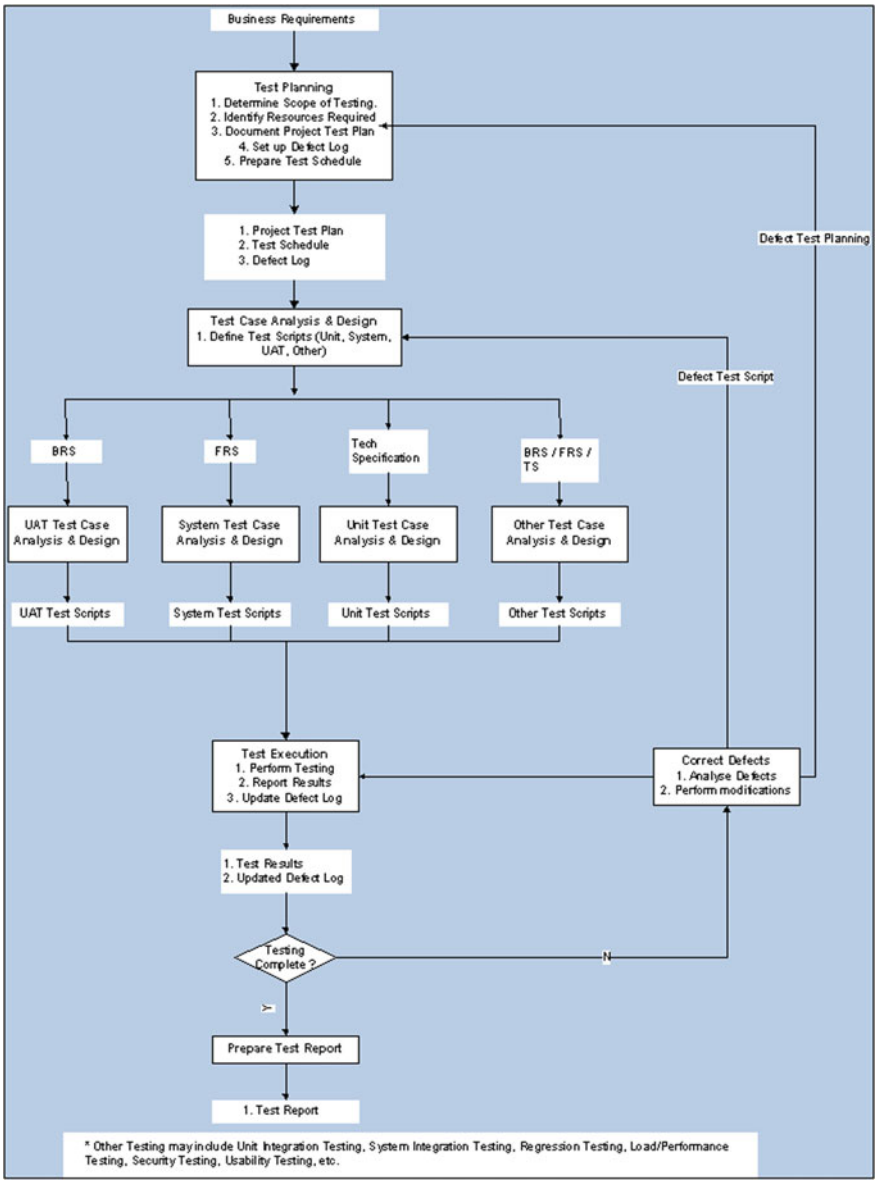


**Fig. 3.1** Simplified test process

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. 3.3.

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

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

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, which includes criteria that must be satisfied for testing to commence. The test department must be ready to carry out the testing, with the test cases defined and the test environment set-up.

Several types of testing employed to verify the correctness of the software are described in Table 3.1.

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 cover the software requirements, and a traceability matrix may be employed to map the requirements to the design and test cases. The traceability matrix provides confidence that each requirement has a corresponding test case for verification. The test cases will include the purpose of the test case, the set-up required, inputs, the procedure, and expected outputs.

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 test cases could not be executed.

The test results are documented including detailed information on the passed and failed tests. This will assist the software developers in identifying the precise causes of failure and determining an appropriate solution. The developers and tester will agree to open a defect report in the defect tracking system to track the correction of each defect.

*Test levels* refer to a group of testing activities that are organized and managed together. A test level is linked to the responsibilities in a project (Table 3.2).

**Table 3.1** Types of testing

| Test type | Description |
|---|---|
| Unit testing | This testing is performed by the software developers to verify the correctness of the software modules |
| Component testing | This testing is performed by the software developers to verify the correctness of the software components, i.e. to ensure that the component is correct and may be reused |
| System testing | This testing is generally carried out by an independent test group to verify the correctness of the complete system |
| Performance testing | This testing is generally 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-life situation is hard to replicate |
| Acceptance testing | This testing carried out by the customer to verify that the software is fit for purpose and matches the customer's expectations |

**Table 3.2** Test levels

| Test level | Description |
|---|---|
| Component testing | Each component is tested separately prior to integration with others. It may include functional, non-functional, and structural tests. The test cases are based on software design and code structure |
| Integration testing | The integrated components are tested together with functional, non-functional, and structural tests |
| System testing | The integrated system is tested by a dedicated test team using functional and non-functional tests (sometimes structural testing—e.g. page navigation) |
| Acceptance testing | This testing is the responsibility of the customer, and the goal is to verify that the software is fit for purpose and matches the customer's expectations. It involves user acceptance testing and operational acceptance testing. It may involve contractual and compliance acceptance testing and alpha/beta testing |

The test status (Fig. 9.11) 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 agreed 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 increased productivity. Tool selection 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 and to the various types of testing.

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

## 3.3 Software Test Planning and Scheduling

Testing is a subproject of a project and needs to be managed as such, and so good planning and monitoring and control are required. The IEEE 829 standard includes a template for test planning, which 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 people to the tasks; defining the schedule; and identifying any risks to the testing and managing them.

The tracking of the testing involves monitoring progress and taking corrective action to ensure quality and schedule are achieved; replanning when the scope of the testing has changed; preparing test reports to give visibility to management (including the number of tests planned, executed, passed, blocked, and failed); retesting corrections to the failed or blocked tests; managing risks; and providing a final test report with a recommendation to go to acceptance testing.

**Table 3.3** Simple test schedule

| Activity | Resource name(s) | Start date | End/replan date | Comments |
|---|---|---|---|---|
| Review requirements | Test Team | 15.02.2019 | 16.02.2019 | Complete |
| Project test plan and review | J. DiNatale | 15.02.2019 | 28.02.2019 | Complete |
| System test plan/review | P. Cuitino | 01.03.2019 | 22.03.2019 | Complete |
| Performance test plan/review | L. Padilla | 15.03.2019 | 31.03.2019 | Complete |
| Regression plan/review | X. Yun | 01.03.2019 | 15.03.2019 | Complete |
| Set-up test environment | X. Yun | 15.03.2019 | 31.03.2019 | Complete |
| System testing | P. Cuitino | 01.04.2019 | 31.05.2019 | In progress |
| Performance testing | L. Padilla | 15.04.2019 | 07.05.2019 | In progress |
| Regression testing | L. Padilla | 07.05.2019 | 31.05.2019 | In progress |
| Test reporting | J. DiNatale | 01.04.2019 | 31.05.2019 | In progress |

Table 3.3 presents a simple test schedule for a small project, and the test manager will usually employ Microsoft Project for scheduling and tracking of larger projects (e.g. Fig. 5.2). The activities in the schedule are tracked and updated to record progress, and dates are revised as appropriate. The project manager will track the key test milestones and will maintain close contact with the testing manager.

It is prudent to consider risk management early in test planning, to identify risks that could potentially arise during the testing, and to manage them accordingly. The probability of occurrence of each risk and its impact is determined.

## 3.4 Test Case Design and Definition

Several types of testing that may be performed during the project were described in Table 3.1, and there may be 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 so on), 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. A test script generally includes:

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

## 3.5 Test Execution

The software developers will carry out the unit and integration testing as part of the normal software development activities. They 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 from the development manager), and the test activities will usually include 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.

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 management, 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.

## 3.6 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 including:

- 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 to consider) is the cumulative arrival rate (Fig. 9.12) 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 overtime 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, Fig. 9.12 indicates that testing halted on 13 May with no testing since then and that would explain why the defect arrival rate per week is zero. Careful investigation needs to be done before the interpretation of a measurement is made, and often several measurements rather than one are employed to make a sound decision.

## 3.7 Testing and Quality

Testing allows the quality of the software to be measured in terms of the defects found. It provides confidence in the quality of the software, and a properly designed test case that passes reduces the level of risk with the system. Further, the quality of the system generally increases (reliability growth) once defects identified during testing have been corrected and verified.

The recommendation of the test manager is carefully considered in the decision on whether to release the software product or not. Decision-making is based on objective facts, and measurements are generally employed to assess the quality of the software.

The open-problem status (Figs. 9.7 and 9.9), the problem arrival rate (Fig. 9.12), and the cumulative problem arrivals' rate (Fig. 9.13) give an indication of the quality and stability of the software product and may be used with other measures to decide on whether it is appropriate to release the software or whether further testing should be done.

### 3.7.1 What Is a Software Defect?

A *defect* is a flaw in the software that causes the software to fail to perform its required function. A defect that is encountered during program execution leads to a software failure.

A defect arises due to a developer making an error that produces a defect in the code. The code where the defect is present may be executed, which results in the software failing to do what it is required (i.e. failure). Not all defects result in failure as some defects may be on rarely used execution paths.

The defect density of the software is the number of defects in the software divided by the number of lines of code.

### 3.7.2 Is Exhaustive Testing Possible?

It may seem like a reasonable approach to perform exhaustive testing of the software for black box and white box testing. However, exhaustive black box testing would involve using every possible test input condition (valid and invalid), and the

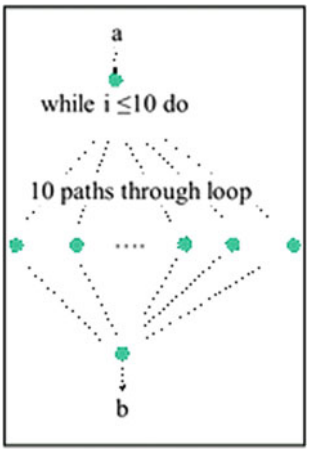


**Fig. 3.2** Number of paths through a trivial program

number of test cases rapidly becomes astronomical (potentially infinite). Similarly, it is unrealistic in white box testing to perform exhaustive path testing and to execute all possible paths through the program. This is since the number of paths through a relatively simple program rapidly becomes astronomical and so exhaustive path coverage is unrealistic.

For the simple program in Fig. 3.2, we have a while loop that executes 10 times, and there are 10 possible paths through each iteration of the loop. This means that there are a total of $10^{10}$ possible paths (10 billion paths) through this simple program. Suppose the tester writes a test in one second, then it would take one tester over 300 years to write $10^{10}$ tests to check each possible path.

Clearly, exhaustive testing is not feasible and instead alternative approaches such as risk-based approaches to testing need to be employed.

### 3.7.3 How Much Testing Should Be Done?

The amount of software testing to be done depends on the level of business and technical risks, as well as the project risks and the constraints on time and budget. Testing should provide sufficient information to allow the stakeholders to make informed decisions as to whether further testing should be done or whether the product is ready to be released.

### 3.7.4 Testing and Quality Improvement

Test defects are valuable in the sense that they provide valuable information that allows 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. 9.

Figure 9.10 indicates that the project had a phase containment effectiveness of approximately 54%. That is, the developers identified 54% of the defects (in software inspections and unit/integration testing), 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. 4. Training on inspections will be required, and the effectiveness of software inspections 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. For example, the test process may be improved by using more effective test tools for various areas of testing (e.g. performance testing). 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 in detecting the defect earlier in the 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, and so software testing plays an important role in quality improvement.

## 3.8 Psychology of Software Tester

The mindset for reviewing and testing is quite different from that of analysing and developing software (Fig. 3.3). There are several skills required to be an effective software tester (e.g. good attention to detail, destructive creativity), and the mindset of the tester influences the outcome of the testing. For example, if the tester is also the developer of the software, then the tester's objectives tend to be focused on demonstrating the correctness of the software, rather than in seeking defects in the software.

The effectiveness of the testing is influenced by the degree of independence of the testing, as professional testers are specialists in finding defects in the software. There are several degrees of independence in testing varying from low level of

**Fig. 3.3** Psychology of software tester. Public domain

independence where the author of the code tests to complete independence where the testing is outsourced to an external organization.

– Tests developed by the person who wrote the software
– Tests designed by another person in the same development group
– Tests designed by a person from a dedicated test group
– Tests developed by an external test organization.

Myers observed that the psychology of the person carrying out the testing influences the testing (Myers 1979). For example, if the tester believes that the purpose of the testing is to demonstrate that the software works correctly (*constructive mindset*), then the tester is likely to focus on proving this point and using inputs for which correct results will be obtained. Similarly, if the tester believes that the purpose of testing is to show that the software does not work (*destructive mindset*), then this approach helps in identifying most of the defects in the software.

However, in practice it is not possible to find all defects in commercial software, and a risk-based approach is often employed. Finally, if the tester believes that the purpose of testing is to detect as many defects as possible and to minimize the risks associated with the release of the software, then this approach is often optimal in achieving good results.

The software tester and developer share a common goal in that they both desire high-quality software, and so they need to collaborate closely to achieve the best possible outcome. The identification of defects during testing may be perceived as a criticism of the product and developers, and so it is essential that defects are

communicated in a constructive and professional way. Otherwise, there is a danger that there could be bad feelings between the testers and developers, and so the relationship needs to be professional with the tester clearly communicating the known defects with the software (as well as the steps to reproduce each defect) and the developer investigating and correcting the defects.

The decision to release the software may be based on several factors such as internal measurements of its quality, business factors, time constraints, and so on. However, it is essential that the risks with the software are known and that they can be managed effectively. Software testing is a means of reducing risk in software engineering.

## 3.9 Test-Driven Development

Test-driven development (TDD) ensures that there is an emphasis on testability of the code from the earliest part of the development. The approach is to write the test cases early, and the software code is then written to pass the test cases. It is a paradigm shift from traditional software engineering, where traditionally unit tests are written and executed after the code has been written.

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. 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 improving the code without changing the functionality and restructuring the code where appropriate to improve its efficiency. The test cases are rerun 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. 12.

## 3.10 E-Commerce Testing

There has been an explosive growth in electronic commerce, and website quality and performance are a key concern. A website is a software application and so standard software engineering principles are employed to verify its quality. 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 un-predictable
- Users may be unknown
- Security threats may be from anywhere
- Rapidly changing technologies.

Often a rapid application development model such as RAD/JAD or Agile is employed to design a little, implement a little, and test a little, and the standard waterfall lifecycle model is rarely employed for the front end of a Web application and. 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 generally used for Web development. The various types of Web testing include:

- 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 following deployment at the customer site, 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 could potentially be incurred in the case of a major outage.

## 3.11 Traceability of Requirements

The objective of requirements traceability 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 6.5). 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 traceability matrix provides confidence that each requirement number has been implemented in the software design and tested via the test plan. Requirement traceability is discussed in more detail in Sect. 6.5.

## 3.12 Software Maintenance and Evolution

Software maintenance is the process of changing a system after it has been delivered to the customer, and it involves correcting any defects that are present in the software and enhancing the system to meet the evolving needs of the customer. The defects may be due to coding, design, or requirements' errors, with coding defects less expensive to fix than requirements' defects. The resolution to the defects involves identifying the affected software components and modifying them and verifying that the solution is correct and that no new problems have been introduced.

Software systems often have a long lifetime (e.g., some systems have a lifetime of 20-30 years), and so the software needs to be continuously enhanced over its lifetime to meet the evolving needs of the customer. Software evolution is concerned with the continued development and maintenance of the software after its initial release, with new releases of the software prepared each year. Each new release includes new functionality and corrections to the known defects.

Maintenance testing plays a key role in verifying that the new release is fit for purpose, and the testing performed depends on the changes made to the system and the associated risks.

## 3.13 Software Test Tools

Test tools are employed to support the test process and to enhance quality and increase productivity. Tool selection needs to be planned, and the selection of a particular tool involves defining the requirements of the proposed tool and identifying candidate tools to evaluate against the requirements. Each tool is assessed to yield an evaluation profile, and the results analysed to enable an appropriate choice to be made.

There are various tools to support testing such as test planning and management tools; defect tracking tools; regression test automation tools; performance tools; and so on (Fig. 3.4). There are tools available from various vendors such as Compuware, Software Research, Inc., HP, LDRA, McCabe and Associates, and IBM Rational.

*Test Management Tools*

There are various test management tools available and their main features are:

- Management of entire testing process
- Test planning
- Test status and reporting
- Graphs for presentation
- Defect tracking system
- Support for many testers
- Audit trail proof that testing has been done
- Test automation
- Support for various types of testing.

*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 they analyse source code files to


**Fig. 3.4** Automated testing tools. Creative Commons

report on areas of code that were not executed at run time, thereby facilitating the identification of missing test data. They generally provide visual reports of the code areas that were executed.

Regression testing involves rerunning existing test cases to verify that the software remains correct following changes made to correct defects or implement new technology. It is often automated with capture and playback tools, and these tools capture, verify, and replay user interactions and allow regression testing to be automated.

The purpose of performance testing 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, disc space used, memory used). Performance testing tools allow 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 if can support 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. Tools to support software testing are discussed in Chap. 10.

## 3.14 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 are their benefits?
6. Describe an effective way to evaluate and select a test tool.
7. What characteristics make e-commerce testing unique from other domains.
8. Discuss the influence of the test manager.
9. Explain test-driven development.

## 3.15 Summary

This chapter discussed the fundamentals of software testing and how testing is used to verify that the software is of high quality and fit to be released to 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.

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

The mindset of the software tester is important where a destructive mindset helps in detecting as many defects as possible, whereas a constructive mindset is often focused on confirming correctness rather than finding defects. Often a mixture of a destructive mindset and destructive helps in minimizing the risks associated with the release of the software.

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.

We discussed tools to support the testing process, and tool selection and evaluation should be done formally. 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.

## References

Myers G (1979) The art of software testing. Wiley, Hoboken
Tmaps (2004) TMap home pages. Sogeti Nederland B.V., Amersfoort. http://www.tmap.net