5.1. Test Planning

5.1.1. Purpose and Content of a Test Plan

A test plan describes the objectives, resources and processes for a test project. A test plan:

- Documents the means and schedule for achieving test objectives
- Helps to ensure that the performed test activities will meet the established criteria
- Serves as a means of communication with team members and other stakeholders
- Demonstrates that testing will adhere to the existing test policy and test strategy (or explains why the testing will deviate from them)

Test planning guides the testers' thinking and forces the testers to confront the future challenges related to risks, schedules, people, tools, costs, effort, etc. The process of preparing a test plan is a useful way to think through the efforts needed to achieve the test project objectives.

The typical content of a test plan includes:

- Context of testing (e.g., scope, test objectives, constraints, test basis)
- Assumptions and constraints of the test project
- Stakeholders (e.g., roles, responsibilities, relevance to testing, hiring and training needs)
- Communication (e.g., forms and frequency of communication, documentation templates)
- Risk register (e.g., product risks, project risks)
- Test approach (e.g., test levels, test types, test techniques, test deliverables, entry criteria and exit criteria, independence of testing, metrics to be collected, test data requirements, test environment requirements, deviations from the organizational test policy and test strategy)
- Budget and schedule

More details about the test plan and its content can be found in the ISO/IEC/IEEE 29119-3 standard.

5.1.2. Tester's Contribution to Iteration and Release Planning

In iterative SDLCs, typically two kinds of planning occur: release planning and iteration planning.

Release planning looks ahead to the release of a product, defines and re-defines the product backlog, and may involve refining larger user stories into a set of smaller user stories. It also serves as the basis for the test approach and test plan across all iterations. Testers involved in release planning participate in writing testable user stories and acceptance criteria (see section 4.5), participate in project and quality risk analyses (see section 5.2), estimate test effort associated with user stories (see section 5.1.4), determine the test approach, and plan the testing for the release.

Iteration planning looks ahead to the end of a single iteration and is concerned with the iteration backlog. Testers involved in iteration planning participate in the detailed risk analysis of user stories, determine the testability of user stories, break down user stories into tasks (particularly testing tasks), estimate test effort for all testing tasks, and identify and refine functional and non-functional aspects of the test object.

5.1.3. Entry Criteria and Exit Criteria

Entry criteria define the preconditions for undertaking a given activity. If entry criteria are not met, it is likely that the activity will prove to be more difficult, time-consuming, costly, and riskier. Exit criteria define what must be achieved in order to declare an activity completed. Entry criteria and exit criteria should be defined for each test level, and will differ based on the test objectives.

Typical entry criteria include: availability of resources (e.g., people, tools, environments, test data, budget, time), availability of testware (e.g., test basis, testable requirements, user stories, test cases), and initial quality level of a test object (e.g., all smoke tests have passed).

Typical exit criteria include: measures of thoroughness (e.g., achieved level of coverage, number of unresolved defects, defect density, number of failed test cases), and completion criteria (e.g., planned tests have been executed, static testing has been performed, all defects found are reported, all regression tests are automated).

Running out of time or budget can also be viewed as valid exit criteria. Even without other exit criteria being satisfied, it can be acceptable to end testing under such circumstances, if the stakeholders have reviewed and accepted the risk to go live without further testing.

In Agile software development, exit criteria are often called Definition of Done, defining the team's objective metrics for a releasable item. Entry criteria that a user story must fulfill to start the development and/or testing activities are called Definition of Ready.

5.1.4. Estimation Techniques

Test effort estimation involves predicting the amount of test-related work needed to meet the objectives of a test project. It is important to make it clear to the stakeholders that the estimate is based on a number of assumptions and is always subject to estimation error. Estimation for small tasks is usually more accurate than for the large ones. Therefore, when estimating a large task, it can be decomposed into a set of smaller tasks which then in turn can be estimated.

In this syllabus, the following four estimation techniques are described.

Estimation based on ratios. In this metrics-based technique, figures are collected from previous projects within the organization, which makes it possible to derive "standard" ratios for similar projects. The ratios of an organization's own projects (e.g., taken from historical data) are generally the best source to use in the estimation process. These standard ratios can then be used to estimate the test effort for the new project. For example, if in the previous project the development-to-test effort ratio was 3:2, and in the current project the development effort is expected to be 600 person-days, the test effort can be estimated to be 400 person-days.

Extrapolation. In this metrics-based technique, measurements are made as early as possible in the current project to gather the data. Having enough observations, the effort required for the remaining work can be approximated by extrapolating this data (usually by applying a mathematical model). This method is very suitable in iterative SDLCs. For example, the team may extrapolate the test effort in the forthcoming iteration as the averaged effort from the last three iterations.

Wideband Delphi. In this iterative, expert-based technique, experts make experience-based estimations. Each expert, in isolation, estimates the effort. The results are collected and if there are deviations that are out of range of the agreed upon boundaries, the experts discuss their current estimates. Each expert is then asked to make a new estimation based on that feedback, again in isolation. This process is repeated until a consensus is reached. Planning Poker is a variant of Wideband Delphi, commonly used in Agile software development. In Planning Poker, estimates are usually made using cards with numbers that represent the effort size.

Three-point estimation. In this expert-based technique, three estimations are made by the experts: the most optimistic estimation (a), the most likely estimation (m) and the most pessimistic estimation (b). The final estimate (E) is their weighted arithmetic mean. In the most popular version of this technique, the estimate is calculated as E = (a + 4*m + b) / 6. The advantage of this technique is that it allows the experts to calculate the measurement error: SD = (b – a) / 6. For example, if the estimates (in person-hours) are: a=6, m=9 and b=18, then the final estimation is 10±2 person-hours (i.e., between 8 and 12 person-hours), because E = (6 + 4*9 + 18) / 6 = 10 and SD = (18 – 6) / 6 = 2.

See (Kan 2003, Koomen 2006, Westfall 2009) for these and many other test estimation techniques.

5.1.5. Test Case Prioritization

Once the test cases and test procedures are specified and assembled into test suites, these test suites can be arranged in a test execution schedule that defines the order in which they are to be run. When prioritizing test cases, different factors can be taken into account. The most commonly used test case prioritization strategies are as follows:

- Risk-based prioritization, where the order of test execution is based on the results of risk analysis (see section 5.2.3). Test cases covering the most important risks are executed first.
- Coverage-based prioritization, where the order of test execution is based on coverage (e.g., statement coverage). Test cases achieving the highest coverage are executed first. In another variant, called additional coverage prioritization, the test case achieving the highest coverage is executed first; each subsequent test case is the one that achieves the highest additional coverage.
- Requirements-based prioritization, where the order of test execution is based on the priorities of the requirements traced back to the corresponding test cases. Requirement priorities are defined by stakeholders. Test cases related to the most important requirements are executed first.

Ideally, test cases would be ordered to run based on their priority levels, using, for example, one of the above-mentioned prioritization strategies. However, this practice may not work if the test cases or the features being tested have dependencies. If a test case with a higher priority is dependent on a test case with a lower priority, the lower priority test case must be executed first.

The order of test execution must also take into account the availability of resources. For example, the required test tools, test environments or people that may only be available for a specific time window.

5.1.6. Test Pyramid

The test pyramid is a model showing that different tests may have different granularity. The test pyramid model supports the team in test automation and in test effort allocation by showing that different goals are supported by different levels of test automation. The pyramid layers represent groups of tests. The higher the layer, the lower the test granularity, test isolation and test execution time. Tests in the bottom layer are small, isolated, fast, and check a small piece of functionality, so usually a lot of them are needed to achieve a reasonable coverage. The top layer represents complex, high-level, end-to-end tests. These high-level tests are generally slower than the tests from the lower layers, and they typically check a large piece of functionality, so usually just a few of them are needed to achieve a reasonable coverage. The number and naming of the layers may differ. For example, the original test pyramid model (Cohn 2009) defines three layers: “unit tests”, “service tests” and “UI tests”. Another popular model defines unit (component) tests, integration (component integration) tests, and end-to-end tests. Other test levels (see section 2.2.1) can also be used.

5.1.7. Testing Quadrants

The testing quadrants, defined by Brian Marick (Marick 2003, Crispin 2008), group the test levels with the appropriate test types, activities, test techniques and work products in the Agile software development. The model supports test management in visualizing these to ensure that all appropriate test types and test levels are included in the SDLC and in understanding that some test types are more relevant to certain test levels than others. This model also provides a way to differentiate and describe the types of tests to all stakeholders, including developers, testers, and business representatives.

In this model, tests can be business facing or technology facing. Tests can also support the team (i.e., guide the development) or critique the product (i.e., measure its behavior against the expectations). The combination of these two viewpoints determines the four quadrants:

- Quadrant Q1 (technology facing, support the team). This quadrant contains component and component integration tests. These tests should be automated and included in the CI process.
- Quadrant Q2 (business facing, support the team). This quadrant contains functional tests, examples, user story tests, user experience prototypes, API testing, and simulations. These tests check the acceptance criteria and can be manual or automated.
- Quadrant Q3 (business facing, critique the product). This quadrant contains exploratory testing, usability testing, user acceptance testing. These tests are user-oriented and often manual.
- Quadrant Q4 (technology facing, critique the product). This quadrant contains smoke tests and non-functional tests (except usability tests). These tests are often automated.

5.2. Risk Management

Organizations face many internal and external factors that make it uncertain whether and when they will achieve their objectives (ISO 31000). Risk management allows the organizations to increase the likelihood of achieving objectives, improve the quality of their products and increase the stakeholders' confidence and trust.

The main risk management activities are:

- Risk analysis (consisting of risk identification and risk assessment; see section 5.2.3)
- Risk control (consisting of risk mitigation and risk monitoring; see section 5.2.4)

The test approach, in which test activities are selected, prioritized, and managed based on risk analysis and risk control, is called risk-based testing.

5.2.1. Risk Definition and Risk Attributes

Risk is a potential event, hazard, threat, or situation whose occurrence causes an adverse effect. A risk can be characterized by two factors:

- Risk likelihood – the probability of the risk occurrence (greater than zero and less than one)
- Risk impact (harm) – the consequences of this occurrence

These two factors express the risk level, which is a measure for the risk. The higher the risk level, the more important is its treatment.

5.2.2. Project Risks and Product Risks

In software testing one is generally concerned with two types of risks: project risks and product risks.

Project risks are related to the management and control of the project. Project risks include:

- Organizational issues (e.g., delays in work products deliveries, inaccurate estimates, cost-cutting)
- People issues (e.g., insufficient skills, conflicts, communication problems, shortage of staff)
- Technical issues (e.g., scope creep, poor tool support)
- Supplier issues (e.g., third-party delivery failure, bankruptcy of the supporting company)

Project risks, when they occur, may have an impact on the project schedule, budget or scope, which affects the project's ability to achieve its objectives.

Product risks are related to the product quality characteristics (e.g., described in the ISO 25010 quality model). Examples of product risks include: missing or wrong functionality, incorrect calculations, runtime errors, poor architecture, inefficient algorithms, inadequate response time, poor user experience, security vulnerabilities. Product risks, when they occur, may result in various negative consequences, including:

- User dissatisfaction
- Loss of revenue, trust, reputation
- Damage to third parties
- High maintenance costs, overload of the helpdesk
- Criminal penalties
- In extreme cases, physical damage, injuries or even death

5.2.3. Product Risk Analysis

From a testing perspective, the goal of product risk analysis is to provide an awareness of product risk in order to focus the testing effort in a way that minimizes the residual level of product risk. Ideally, product risk analysis begins early in the SDLC.

Product risk analysis consists of risk identification and risk assessment. Risk identification is about generating a comprehensive list of risks. Stakeholders can identify risks by using various techniques and tools, e.g., brainstorming, workshops, interviews, or cause-effect diagrams. Risk assessment involves: categorization of identified risks, determining their risk likelihood, risk impact and level, prioritizing, and proposing ways to handle them. Categorization helps in assigning mitigation actions, because usually risks falling into the same category can be mitigated using a similar approach.

Risk assessment can use a quantitative or qualitative approach, or a mix of them. In the quantitative approach the risk level is calculated as the multiplication of risk likelihood and risk impact. In the qualitative approach the risk level can be determined using a risk matrix.

Product risk analysis may influence the thoroughness and scope of testing. Its results are used to:

- Determine the scope of testing to be carried out
- Determine the particular test levels and propose test types to be performed
- Determine the test techniques to be employed and the coverage to be achieved
- Estimate the test effort required for each task
- Prioritize testing in an attempt to find the critical defects as early as possible
- Determine whether any activities in addition to testing could be employed to reduce risk

5.2.4. Product Risk Control

Product risk control comprises all measures that are taken in response to identified and assessed product risks. Product risk control consists of risk mitigation and risk monitoring. Risk mitigation involves implementing the actions proposed in risk assessment to reduce the risk level. The aim of risk monitoring is to ensure that the mitigation actions are effective, to obtain further information to improve risk assessment, and to identify emerging risks.

With respect to product risk control, once a risk has been analyzed, several response options to risk are possible, e.g., risk mitigation by testing, risk acceptance, risk transfer, or contingency plan (Veenendaal 2012). Actions that can be taken to mitigate the product risks by testing are as follows:

- Select the testers with the right level of experience and skills, suitable for a given risk type
- Apply an appropriate level of independence of testing
- Conduct reviews and perform static analysis
- Apply the appropriate test techniques and coverage levels
- Apply the appropriate test types addressing the affected quality characteristics
- Perform dynamic testing, including regression testing

5.3. Test Monitoring, Test Control and Test Completion

Test monitoring is concerned with gathering information about testing. This information is used to assess test progress and to measure whether the test exit criteria or the test tasks associated with the exit criteria are satisfied, such as meeting the targets for coverage of product risks, requirements, or acceptance criteria.

Test control uses the information from test monitoring to provide, in a form of the control directives, guidance and the necessary corrective actions to achieve the most effective and efficient testing. Examples of control directives include:

- Reprioritizing tests when an identified risk becomes an issue
- Re-evaluating whether a test item meets entry criteria or exit criteria due to rework
- Adjusting the test schedule to address a delay in the delivery of the test environment
- Adding new resources when and where needed

Test completion collects data from completed test activities to consolidate experience, testware, and any other relevant information. Test completion activities occur at project milestones such as when a test level is completed, an agile iteration is finished, a test project is completed (or cancelled), a software system is released, or a maintenance release is completed.

5.3.1. Metrics used in Testing

Test metrics are gathered to show progress against the planned schedule and budget, the current quality of the test object, and the effectiveness of the test activities with respect to the objectives or an iteration goal. Test monitoring gathers a variety of metrics to support the test control and test completion.

Common test metrics include:

- Project progress metrics (e.g., task completion, resource usage, test effort)
- Test progress metrics (e.g., test case implementation progress, test environment preparation progress, number of test cases run/not run, passed/failed, test execution time)
- Product quality metrics (e.g., availability, response time, mean time to failure)
- Defect metrics (e.g., number and priorities of defects found/fixed, defect density, defect detection percentage)
- Risk metrics (e.g., residual risk level)
- Coverage metrics (e.g., requirements coverage, code coverage)
- Cost metrics (e.g., cost of testing, organizational cost of quality)

5.3.2. Purpose, Content and Audience for Test Reports

Test reporting summarizes and communicates test information during and after testing. Test progress reports support the ongoing control of the testing and must provide enough information to make modifications to the test schedule, resources, or test plan, when such changes are needed due to deviation from the plan or changed circumstances. Test completion reports summarize a specific stage of testing (e.g., test level, test cycle, iteration) and can give information for subsequent testing.

During test monitoring and control, the test team generates test progress reports for stakeholders to keep them informed. Test progress reports are usually generated on a regular basis (e.g., daily, weekly, etc.) and include:

- Test period
- Test progress (e.g., ahead or behind schedule), including any notable deviations
- Impediments for testing, and their workarounds
- Test metrics (see section 5.3.1 for examples)
- New and changed risks within testing period
- Testing planned for the next period

A test completion report is prepared during test completion, when a project, test level, or test type is complete and when, ideally, its exit criteria have been met. This report uses test progress reports and other data. Typical test completion reports include:

- Test summary
- Testing and product quality evaluation based on the original test plan (i.e., test objectives and exit criteria)
- Deviations from the test plan (e.g., differences from the planned schedule, duration, and effort).
- Testing impediments and workarounds
- Test metrics based on test progress reports
- Unmitigated risks, defects not fixed
- Lessons learned that are relevant to the testing

Different audiences require different information in the reports, and influence the degree of formality and the frequency of reporting. Reporting on test progress to others in the same team is often frequent and informal, while reporting on testing for a completed project follows a set template and occurs only once.

The ISO/IEC/IEEE 29119-3 standard includes templates and examples for test progress reports (called test status reports) and test completion reports.

5.3.3. Communicating the Status of Testing

The best means of communicating test status varies, depending on test management concerns, organizational test strategies, regulatory standards, or, in the case of self-organizing teams (see section 1.5.2), on the team itself. The options include:

- Verbal communication with team members and other stakeholders
- Dashboards (e.g., CI/CD dashboards, task boards, and burn-down charts)
- Electronic communication channels (e.g., email, chat)
- Online documentation
- Formal test reports (see section 5.3.2)

One or more of these options can be used. More formal communication may be more appropriate for distributed teams where direct face-to-face communication is not always possible due to geographical distance or time differences. Typically, different stakeholders are interested in different types of information, so communication should be tailored accordingly.

5.4. Configuration Management

In testing, configuration management (CM) provides a discipline for identifying, controlling, and tracking work products such as test plans, test strategies, test conditions, test cases, test scripts, test results, test logs, and test reports as configuration items.

For a complex configuration item (e.g., a test environment), CM records the items it consists of, their relationships, and versions. If the configuration item is approved for testing, it becomes a baseline and can only be changed through a formal change control process.

Configuration management keeps a record of changed configuration items when a new baseline is created. It is possible to revert to a previous baseline to reproduce previous test results.

To properly support testing, CM ensures the following:

- All configuration items, including test items (individual parts of the test object), are uniquely identified, version controlled, tracked for changes, and related to other configuration items so that traceability can be maintained throughout the test process
- All identified documentation and software items are referenced unambiguously in test documentation

Continuous integration, continuous delivery, continuous deployment and the associated testing are typically implemented as part of an automated DevOps pipeline (see section 2.1.4), in which automated CM is normally included.

5.5. Defect Management

Since one of the major test objectives is to find defects, an established defect management process is essential. Although we refer to "defects" here, the reported anomalies may turn out to be real defects or something else (e.g., false positive, change request) - this is resolved during the process of dealing with the defect reports. Anomalies may be reported during any phase of the SDLC and the form depends on the SDLC. At a minimum, the defect management process includes a workflow for handling individual anomalies from their discovery to their closure and rules for their classification. The workflow typically comprises activities to log the reported anomalies, analyze and classify them, decide on a suitable response such as to fix or keep it as it is and finally to close the defect report. The process must be followed by all involved stakeholders. It is advisable to handle defects from static testing (especially static analysis) in a similar way.

Typical defect reports have the following objectives:

- Provide those responsible for handling and resolving reported defects with sufficient information to resolve the issue
- Provide a means of tracking the quality of the work product
- Provide ideas for improvement of the development and test process

A defect report logged during dynamic testing typically includes:

- Unique identifier
- Title with a short summary of the anomaly being reported
- Date when the anomaly was observed, issuing organization, and author, including their role
- Identification of the test object and test environment
- Context of the defect (e.g., test case being run, test activity being performed, SDLC phase, and other relevant information such as the test technique, checklist or test data being used)

- Description of the failure to enable reproduction and resolution including the steps that detected the anomaly, and any relevant test logs, database dumps, screenshots, or recordings
- Expected results and actual results
- Severity of the defect (degree of impact) on the interests of stakeholders or requirements
- Priority to fix
- Status of the defect (e.g., open, deferred, duplicate, waiting to be fixed, awaiting confirmation testing, re-opened, closed, rejected)
- References (e.g., to the test case)

Some of this data may be automatically included when using defect management tools (e.g., identifier, date, author and initial status). Document templates for a defect report and example defect reports can be found in the ISO/IEC/IEEE 29119-3 standard, which refers to defect reports as incident reports.