**Week 5 – Test Plan**

Cody Angeline

The University of Arizona Global Campus

CST 499: Capstone for Computer Software Technology

Dr. Amr Elchouemi

January 19, 2023

Table of Contents

[1 Introduction 3](#_Toc125042883)

[1.1 Unit Testing 3](#_Toc125042884)

[1.2 Functional Testing 3](#_Toc125042885)

[1.3 Integration Testing 4](#_Toc125042886)

[1.4 System Testing 4](#_Toc125042887)

[1.5 Models 4](#_Toc125042888)

[2 Testing Strategies 5](#_Toc125042889)

[2.1 Static Analysis 5](#_Toc125042890)

[2.2 Walkthrough Review 6](#_Toc125042891)

[2.3 Inspection Review 6](#_Toc125042892)

[2.4 Technical Review 6](#_Toc125042893)

[2.5 Informal Review 7](#_Toc125042894)

[2.6 Data Flow Analysis 7](#_Toc125042895)

[2.7 Control Flow Analysis 7](#_Toc125042896)

[2.8 Static Analysis: Testing Tools 7](#_Toc125042897)

[2.9 Black Box Testing 8](#_Toc125042898)

[2.10 White Box Testing 8](#_Toc125042899)

[2.11 Compliance Convention and Standards 8](#_Toc125042900)

[3 Test Management Strategies 9](#_Toc125042901)

[3.1 Test Teams 9](#_Toc125042902)

[3.2 Test Roles 9](#_Toc125042903)

[3.3 Test Exit Criteria 9](#_Toc125042904)

[3.4 Test Estimated Effort 9](#_Toc125042905)

[3.5 Test & Risk Evaluation 9](#_Toc125042906)

[3.6 Incident Reporting 9](#_Toc125042907)

[3.7 Defection Classification 10](#_Toc125042908)

[3.8 Configuration Management 10](#_Toc125042909)

[References 11](#_Toc125042910)

# Introduction

A software’s test plan is created at the start of a software project after the elicitation has been performed, and the software requirements specification has been designed. The test plan outlines the test policies of the organization as well as the goals, risks, and constraints of the software project (Spillner et al., 2014).

This test plan is for the University of Arizona’s online course registration website (UACRW). The document aims to outline the overall strategy for testing, assert the test environments automation techniques, as well as define the roles and management of the test effort.

## Unit Testing

Unit testing is meant to evaluate the functional correctness of discrete classes, methods, functions, or procedures (whose nomenclature varies by programming paradigm); therefore, these software units may be called as such “*units*”. Generally, unit tests are performed by the originator of the source code (developer). The unit testing framework that developers will use for UACRW is the test-driven development framework as outlined by Beck (2003) in his book *Test-Driven Development: By Example*.

## Functional Testing

Functional testing evaluates the correctness of discrete modules. Modules in software architecture terms, are groups of related units of software (Richards & Ford, 2020). During functional testing, the UACRW modules will be evaluated via black-box testing by third-party testers. Because the testing will be black-box, the success criteria will be based solely on the software requirements specification (SRS).

## Integration Testing

A component in software architecture terms is a group of related modules. Examples of components are Java archive (.JAR) files, or a dynamically linked library (.dll) in Microsoft .NET. During the integration phase of development, developers compose groups of components to form structural units and subsystems (Spillner et al., 2014). The purpose of integration testing is to ensure that all components collaborate as expected. During integration testing, the UACRW components will be evaluated via black-box testing by third-party testers.

## System Testing

System testing is meant to evaluate the non-functional requirements of the software products. System testing analyzes the software product from a customer/user perspective, and also incorporates acceptance testing. During this phase, the testers will determine if the system requirements are completely and appropriately implemented (Spillner et al., 2014). The UACRW system tests will be conducted by third-party and customer testers.

## Models

System modeling is meant to communicate the different views of a software system as an abstraction rather than an implementation (Sommerville, 2018). Models are traditionally used during the requirements engineering phase to help narrow down system functionality that engineers will implement. The SRS accompanying this test plan within the UACRW deliverable includes the abstractions necessary to appropriately design black-box tests.

# Testing Strategies

To design tests that produce data that can be analyzed and presented as information to decision makers, the SRS must be scrutinized in reviews. If a requirement is found to be too imprecise during the review, it must be reworked.

Testing is performed on both documentation and the software product itself. In order to perform this, logical test cases must identify the exact inputs and outputs expected from the test unit. For each test case, preconditions must be accurately described and it must be clear what environmental conditions must be present. Testing of the UACRW will be comprised of static analysis, and dynamic testing (white & black box).

## Static Analysis

Static analysis aims to identify defects, or areas that may be prone to defects in parts of a document through reviews. The types of artifacts analyzed during reviews are the technical requirements, architecture diagrams, and class diagrams found within the SRS. Static analysis via the compiler, and complexity analysis should be performed before or during functional or integration testing to ensure organizational programming guidelines and conventions are upheld (Spillner et al., 2014).

## Walkthrough Review

A walkthrough is an informal review that aims to find defects, ambiguities, and problems in written documents. Walkthroughs are non-time-limited meetings, and have little emphasis placed on preparation. This technique is most beneficial for non-critical documents (Spillner et al., 2014). Management and stakeholders of UACRW project will perform walkthroughs after each sprint during the software development lifecycle.

## Inspection Review

An inspection is the most formal of all reviews. Everyone in attendance will actively participate, and will have explicitly stated functions during the meeting. The goal for this type of review is to identify defects, ambiguities, problems in written documents in order, improve the quality of the document review process (Spillner et al., 2014). Management and technical members of UACRW will perform inspections at the end of each major revision made to the SRS.

## Technical Review

A technical review focus on specification/standards compliance, and applicability of SRS. All reviewers must be technically qualified. Management does not participate in these types of reviews (Spillner et al., 2014). The development and test team will perform technical reviews at the conclusion of every integration test.

## Informal Review

The informal review process is very generic. Planning is limited, and there is no meeting prior to follow up on an exchange of findings. Informal reviews involve very little effort, and often lead to more discussion over documents than evaluation (Spillner et al., 2014). The entire team of the UACRW project will perform informal reviews on an ad-hoc basis.

## Data Flow Analysis

Data flow analysis is used to reveal defects in the usage of variables, instantiation of objects, and use of modules (Spillner et al., 2014). Data flow analysis is primarily performed with the compiler of the source code. The analysis detects anomalies that have the potential to introduce failures in the operation of the software, but do not necessarily do so in any immediately observable way. Thus, the results of the analysis are detections of risks.

## Control Flow Analysis

Control flow analysis examines the changes in the execution of the program activated by control structures. Control flow analysis can be performed with software that creates control flow graphs from source code, or by proxy on compilation of source code (Spillner et al., 2014).

## Static Analysis: Testing Tools

Reviews are the primary form of static analysis used by managers and stakeholders, while the compiler is a primary analysis tool used by developers. All compilers carry out the static analysis function when checking that the syntax of the language is upheld, as well as detecting control and data flow anomalies (Spillner et al., 2014).

Software characteristics such as testability and maintainability can be estimated by using tools such as the cyclomatic complexity metric. The number generated from this metric indicates the volume of testing that will need to be performed in order to meet the coverage criteria specified by decision makers (Spillner et al., 2014).

## Black Box Testing

Black box testing is meant to analyze the software without any considerations of its inner workings. The test cases are formulated based purely on the information found within the SRS. Tests are designed via equivalence class partitioning and boundary value analysis to determine the inputs and expected outputs that a software should expect/generate given that it is interacted with in a specific way. There are a variety of black box tests, but some of the more common are use-case, state, logic, and pairwise testing.

## White Box Testing

White box testing is meant to analyze the inner workings of a software systems units, modules and components. The first step in performing white box testing is to have already analyzed certain static complexity metrics. The various forms of white box testing are statement, decision, condition, and path testing (Spillner et al., 2014).

## Compliance Convention and Standards

Standards exist to set requirements on design to ensure that parties adhering to the guidelines will yield reproducible products. Quality managers are tasked with defining in the context of their product, which standards, rules and legal directives are applicable to their initiatives, and thus must be tested or adhered to (Spillner et al., 2014).

# Test Management Strategies

In order to ensure that the software product has been properly tested, a decision must be made as to whom will do the testing, and that testing is well coordinated with the development effort. Because developers have a tendency to turn a blind eye towards their software’s deficiencies, it is much more desirable to have a third party perform some or all of the testing. The decision is left with management and stakeholders whether the development team is responsible for testing, but developers test one another’s code. Another possibility is that a tester is integrated into the development team and is solely responsible for all of the testing. If neither of these situations is desirable, independent test specialists or entirely separate organizations may be used (Spillner et al., 2014).

Accompanying the decision of whom will perform the testing is the criterium for test exit, and the associated test effort. A risk analysis should then be performed to identify the acceptable number of tests performed given limited time and resource. Once test cases have been executed and the results analyzed, incident reports are generated and submitted to a central database. An incident database enables test and project managers to gain a complete understanding of the number and types of problems the software is experiencing. Furthermore, test databases help determine when test testing can be concluded. The reports generated by the test database are formatted according to different internationally accepted standards. The test reports generated during the development of the UACRW will be done so according to IEEE 1044.

## Test Roles

In order to facilitate the testing of UACRW, all unit testing will be performed by developers, while a dedicated tester will exist within the test team to facilitate the functional and integration tests. For all other testing, a third-party will be used. A test manager will be responsible for writing all test policy, developing test approaches, updating this test plan to ensure that its artifacts remain traceable to requirements, and writing test reports.

## Test Exit Criteria

Test cases will be prioritized such that if testing ends prematurely, the most usable amount of data is gathered. Given the constraints of time and budget, the highest priority test cases will be executed first to include regressions. For the UACRW, the test exit criteria will be reached according to the following tables.

**Table 1**.

*Failures/test hour.*

|  |  |  |  |
| --- | --- | --- | --- |
| Low | Medium | High | Critical |
| 0.3 | 0.25 | 0.2 | 0.15 |

**Table 2**.

*General test exit criteria.*

|  |  |  |  |
| --- | --- | --- | --- |
| Requirements | Test Coverage | Failed Test Cases | Budget |
| 100% | 95% | 5% | $0.00 |

## Test Estimated Effort

In order to produce a test effort, estimate a list of all tasks involved in the development activity can be presented to experienced members of the team, or be rooted in data from prior projects (Spillner et al., 2014). The test approach to be used in the UACRW project will be risk-based and the test effort is based on prior projects along with lines of code (LOC). Based on projects similar to this, it is estimated that the project will demand 250,000 LOC.

## Test & Risk Evaluation

During test evaluation, the test object is analyzed against the test exit criteria. Based on this, the decision to terminate or introduce additional tests will be made. An adequate test exit criterium is rooted in risk (Spillner et al., 2014). The risk associated with the UACRW product is considered to be high because of the financial nature of the system. The figure below identifies the key areas of risk that drive the test exit criteria.

**Table 3**.

*Risk categories.*

|  |  |  |  |
| --- | --- | --- | --- |
| Email | Scheduling | Financial | PII |
| Low | Medium | High | Critical |

*Note*. Personally identifiable information (PII). The term “*Scheduling*” is used to define inconveniences associated with incorrect class assignments or removals that may or may not result in personal grievance.

## Incident Reporting

The test incident reporting for the UACRW project will be submitted to a central database and contain the following information:

* Software module, component, or subsystem name.
* Test environment information.
* Tester name.
* Defect classification.

The test report produced by the central database will be formatted according to IEEE 1044. The reports will present all necessary information for reproducing and localizing the potential fault, and whose most notable features will be the defect classification and the incident status.

## Defection Classification

The defect classifications will be based on the risk analysis categories produced in section 3.4. These categories will be expanded upon if a determination is reached that they are to ambiguous or incomplete. This will be determined during a walkthrough review phase of development.

## Configuration Management

The implementation of configuration management for the project requires knowledge of the business policy and procedure of the particular sector of operation the company exists in. Based on this, various tools will be used that facilitate the management of artifacts produced during development, as well as the framework for management deployed. The software version control used for the project will be Git. There is no requirement to manage build control, and the IDE used for development is Brackets, which does not incorporate software version or build control. The framework to be used will be the naming model.

References

Beck, K. (2003). *Test-driven development*. Pearson Education. ISBN 0-321-14653-0

Richards, M., Ford, N. (2020). *Fundamentals of software architecture*. O’Reilly Media Inc. http://oreilly.com/catalog/errata.csp?isbn=9781492043454

Sommerville, I. (2018). *Software engineering* (10th Edition). Pearson.

Spillner, A., Linz, T., & Schaefer, H. (2014). *Software testing foundations: A study guide for the certified tester exam* (4th ed.). Rocky Nook.

Tsui, F., Karam, O., & Bernal, B. (2018). *Essentials of software engineering* (4th ed.). Jones & Bartlett Learning.