**Week 5 Discussion 2: Project Test Plan**

Shaun Hoadley

CST499: Capstone for Computer Software Technology

Professor Amr Elchouemi

February 10, 2022

**Project Test Plan**

**1. Test Management Strategy**

**1.1 Test Teams**

It is important to consider test teams and an approach to each level of testing to achieve efficient development of the Student Course Enrollment System. Developers on the team should not test their own code, if at all possible,. According to Software testing foundations: A study guide for the certified tester exam (4th ed.), “because there is a tendency to be blind to our own errors, it is much more efficient to let different people perform testing and development and to organize testing as independently as possible from development” (Spillner, et al., 2014). It is for this reason, there should designated testers on the development team to perform the component testing. Additionally, a designated testing team should perform the integration testing. It could be beneficial if these testers include individuals from the business or IT staff. The testing team for system testing should include specialists: “Especially in system testing, it is often necessary to extend the test team by adding IT specialists, at least temporarily, to perform work for the test team” (Spillner, et al., 2014). By using dedicated testers this way allows the system to be viewed and tested from several perspectives.

**1.2 Test Roles**

Conducting the testing of the Student Course Enrollment System, the following test roles should be assigned to individuals: testing manager, testing designer, testing automator, testing administrator, and testers (Spillner, et al., 2014). The test manager requires experience in project management, software testing, quality management, and personnel management. The test designer needs a skillset that includes test methods and specification, testing, and software engineering and should also hold a degree in Computer Science or related field. The test automator requires experience in testing, programming, scripting, automation, and test tools. The test administrator needs a skillset involving setting up and supporting test environments, as well as system administration and networking. The testers should have experience following procedures, executing tests, reporting failures, and using test objects and testing tools (Spillner, et al., 2014).

**1.3 Exit Criteria**

Each test case should contain specific exit criteria in order to determine when a test is considered to be complete. Exit criteria are important because “They prevent tests from ending too early, for example, because of time pressure or because of resource shortages” (Spillner, et al., 2014). Test cases are prioritized and classified based on risk, then three metrics will be used to quantify when testing of the project is complete. The first metric is based on graphing the number of defects discovered over time. The rate of defects discovered should be high when testing first begins, and should decrease continually as the majority of defects are identified and managed. Testing will only stop once the defect find rate reaches 1 defect/1000 hours for high risk test cases, 1 defect/500 hours for medium risk, and 1 defect/100 hours for low risk. The second metric is defect seeding and estimation. Testing of the project will not stop until 100% of defects are found in high risk test cases, 80% of defects are discovered in medium risk test cases, and 70% of defects have been found in low risk test cases. For example, if a third-party seeds ten defects in the software amongst the others that were naturally occurring, then testing should stop when eight out of those ten defects have been found for medium risk test cases (Tsui, Karam, & Bernal, 2018). The last metric for testing the project, testing should only stop once 100% line coverage has been accomplished.

**1.4 Test Estimated Effort**

When planning software testing, it is important to consider both the costs of testing, as well as the costs of undetected defects. It is the responsibility of the test manager to initiate test effort estimation during the planning phase to ensure the proper assignment and distribution of resources. Estimating test effort for the project will be conducted by basing estimates on data from previous or similar projects, as this method provides the most reliable test effort estimates: “task-driven test effort estimation tends to underestimate the testing effort. Estimating based on experience data of similar projects or typical values usually leads to better results” (Spillner, et al., 2014).

**1.5 Test and Risk**

Risk-based prioritization will be implemented to ensure that the most significant defects within the project are revealed as early as possible. This helps prevent critical defects from having downstream affects and reduce the time and cost of handling critical defects that make it to production. Prioritization by risk is one of the best methods for selecting test cases, though test cases can be prioritized based on multiple factors, including severity, risk, complexity, frequency of function use, visibility, and priority of functional and non-functional requirements from the customer. According to Spillner, et al., “Risk based prioritization of the tests ensures that risky product parts are tested more intensively and earlier than parts with lower risk. Severe problems (causing much corrective work or serious delays) are found as early as possible” (2014).

**1.6 Incident Reporting**

Incident reporting will be used throughout the development and testing of the Student Course Enrollment System to document and manage incidents. All defects should be documented that are determined to be significant and legitimate and not the result of a poorly designed test. The template shown in Figure 1 (Spillner, et al., 2014) below will be used for documenting incidents. This allows the communication of incidents to be consistent so developers will be able to easily reproduce defects. Homès stresses the importance of clearly communicating defects: “Determining the impact in an understandable way for developers (data loss, functionality loss, software instability, etc.) and for customers and the hierarchy (impacts in financial terms or in usability terms, noncompliance to requirements, etc.) enables a quick recognition of the anomaly” (2012). The incident report contains the following information: the tested software, test environment, tester’s name, the class that contains the defect, prioritization of the defect, and information relevant to reproducing and locating the defect (Spillner, et al., 2014). Testers, developers, customers and users can report incidents. When corrections are made, the incident report should be updated in the database, helping the project team track and manage incidents.

Table

Description automatically generated

**Figure 1: Incident Reporting Template (Spillner, et al., 2014)**

**1.7 Defect Classification**

Defect classification will be implemented for prioritizing incidents related to the project, where the severity level will reflect the level of impairment caused by the defect. The following 5 levels of severity will be used: 1-Fatal, 2-Very Serious, 3-Serious, 4-Moderate, and 5-Mild. The criteria for these levels can be seen in Figure 2 (Spillner, et al., 2014) below. Additionally, levels of priority will also be implemented to identify how rapidly problems should be addressed. The following four levels of priority will be used: 1-Immediate, 2-Next Release, 3-On Occasion, and 4-Open. The criteria assigned to each of these priority levels can be seen in Figure 3 (Spillner, et al., 2014) below.

Table

Description automatically generated

**Figure 2: Severity Levels (Spillner, et al., 2014)**

Graphical user interface

Description automatically generated with medium confidence

**Figure 3: Priority Levels (Spillner, et al., 2014)**

**1.8 Configuration Management**

Throughout the development of the Student Course Enrollment System, configuration management will be used in order to track the version history of the project, and to allow multiple developers to make contributions without interfering with work of each other. A number of avoidable problems such as developers overwriting each others code, the inability to integrate components due to unknown versions, and difficulty testing from changes to a component being untraceable or testers don’t know which test cases belong to which version of a test object, are all by-products of poor configuration management (Spillner, et al., 2014). Through implementing version management, configuration identification, incident and change status control, and configuration audits, these circumstances can be avoided. The use of configuration management tools allows all of these activities to be achieved. In *System for Efficient Storage and Version Control of Arbitrary File Collections,* it discusses some popular version control options: “Modern file version control systems, such as git, mercurial, or svn implement a concept of revision or commit. This concept may be crucial to store program source code, as it allows users to view previous versions of stored files” (Dmitriev, et al., 2020).

**2 Test Tools**

**2.1 Introduction**

Test tools are used to ease the burden of manual software testing. Since the Student Course Enrollment System is a new build, test tools should be acquired and utilized with the intent to increase test efficiency and reliability, reduce manual testing, and achieve load and performance testing (Spillner, et al., 2014). A cost-benefit analysis should be conducted before selecting test tools. Tools for testing the project may include those for test specification, test management and control, dynamic testing, static testing, and non-functional testing.

**2.2 Test Specification Tools**

Tools for the specification phase of testing should also be considered for the project because they help test designers generate test data. There are four types of test generators: code-based test generators, database-based test generators, interface-based test generators, and specification-based test generators. Code-based test generators use the source code to generate test data, but while these tools can be useful, they are unable to detect faults caused by missing code. Additionally, Spillner, et al., states, “The use of code as a test basis for testing the code itself is in general a very poor foundation” (2014). However, code-based test generators can be very useful in regression testing. According to Spillner, database-based test generators “process database schemas and are able to produce test databases from these schemas.” These tools aid in generating test data by filtering through databases. An example of database-based test generator tool is DatProf, which supports Windows operating systems and generates synthetic data using most common database technologies (Software Testing Help, 2020). Interface-based test generators identify parameter domains, which can then be used for boundary value analysis and equivalence class partitioning to generate test cases and are also useful when testing API’s and GUI’s (Spillner, et al., 2014). Specification-based test generators synthesize test data from formal specifications or models.

**2.3 Test Management and Control Tools**

Tools supporting test management and control allow testers and project managers to document, prioritize, list, and maintain test cases (Spillner, et al., 2014). A test management tool being considered for the development of the project is Jira. Test management tools can ensure that there are test cases to address each software requirement. Test tools with this functionality are often considered as a requirements management tool. Test execution tools are another form of test management tool that are used for executing test scripts automatically and documenting the results. Incident management tools are used for tracking software defects and their resolutions. Configuration management tools (such as Git) track versions and builds that will be tested. Finally, tool integration can be used to combine multiple test management tools into one (Spillner, et al., 2014). A cost-benefit analysis specific to the Student Course Enrollment System should be used for determining which tools will be the most beneficial to the project.

**2.4 Dynamic Testing Tools**

Utilizing dynamic testing tools during development of the project can reduce the mechanical work involved with manual test execution. These tools provide the test object with input data, and record the output. Some examples of dynamic testing tools include debuggers, test drivers and test frameworks, test robots, simulators, comparators, dynamic analyzers, and tools for coverage analysis (Spillner, et al., 2014). Debuggers (like xDebug) can be found in popular text editors and IDE’s and can allow testers and developers to execute a program one line at a time. Programs can also be paused mid-execution for observing or manipulating variables. Test drivers and test frameworks are often used for component and integration tests, and are designed for specific programming languages and environments, providing control over the test object (Spillner, et al., 2014). Test robots log manual inputs such as keyboard strokes or mouse clicks, store them as a script, and then can execute the script to replay that sequence of inputs in an automated fashion and are especially helpful for regression testing. Simulators mirror the real system environment and are used when the tests cannot be executed there. Comparators identify deviations from expected results. Dynamic analyzers evaluate memory usage and allocation, as well as identify “memory leaks, wrong pointer allocation, or pointer arithmetic problems” (Spillner, et al., 2014). Lastly, coverage analysis tools track statement and branch coverage to ensure that as much code as possible is executed during testing. One example of a dynamic analysis tool is VectorCast. “VectorCAST ICover is used to test code coverage of source code through dynamic analysis” (Khalid, 2017).

**2.5 Static Testing Tools**

Static testing tools should be applied directly to the source code of the project because they support the evaluation of source code, they can aid in identifing defects early in the development process before code is even executed. Review tools are a form of static testing that serve like checklists helping testers plan, execute, and evaluate code reviews (Spillner, et al., 2014). Static analyzers can also be used to identify areas in the code that are complex, risky, inconsistent, violate programming standards, defect-prone, or pose portability issues, then these areas can then be refactored, and or prioritized for further testing. Model checkers can also be considered to analyze the UML models to check for “missing states, state transitions, and other inconsistencies in a model” (Spillner, et al., 2014). An example of a static analysis tool is Polyspace, released in 2014 consists of two components: Polyspace Code Prover and Polyspace Bug Finder. “The former identifies every code instruction by applying all possible values of variables, while the later is used to perform static analysis” (Khalid, 2017).

**2.6 Non-Functional Testing Tools**

Non-functional testing tools must also be considered for assessing the quality attributes of the Student Course Enrollment System, as often, these attributes can be difficult to evaluate using manual methods alone. According to Spillner, et al., “Load test tools generate a synthetic load (i.e., parallel data-base queries, user transactions, or network traffic”, allowing testers to determine how a system will behave under more realistic conditions. Additionally, performance tests may reveal the response time of a system. Load and performance tools are particularly useful when a system expects to recieve high traffic or needs to handle large numbers of parallel requests (Spillner, et al., 2014). They can also identify bottlenecks within the system. An example of a tool used for performance and load testing is Apache JMeter (Khandelwal, 2019). In addition to performance and load testing tools, other non-functional testing tools include those for testing security and assessing data quality. It is critical to identify security vulnerabilities early on in development. As Parizi, et al. states, “Developers cannot afford to believe that their security requirements during development are perfect and impenetrable, no matter how thorough their precautions might be” (2018). In addition to testing for security vulnerabilities, data quality assessment tools can ensure that data before and after migration or conversion is correct and complete. Using these testing tools, developers, testers, and project managers can produce a Student Course Enrollment System that is reliable and robust.

**References**

Dmitriev, S. O., Valter, D. A., & Kontsov, A. M. (2020). *System for Efficient Storage and Version Control of Arbitrary File Collections*. 2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), 2020 IEEE Conference Of, 295–298. https://doi-org.proxy-library.ashford.edu/10.1109/EIConRus49466.2020.9038922

Homès, B. (2012). Fundamentals of Software Testing, John Wiley & Sons, Incorporated. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/ashford- ebooks/detail.action?docID=1120766. Created from ashford-ebooks on 2022-02-10 21:16:36

Khalid, R. (2017). *Towards an automated tool for software testing and analysis*. 2017 14th International Bhurban Conference on Applied Sciences and Technology (IBCAST), Applied Sciences and Technology (IBCAST), 2017 14th International Bhurban Conference On, 461–465. https://doi- org.proxy- library.ashford.edu/10.1109/IBCAST.2017.7868094

Khandelwal, A. (2019). Top 10 Non -Functional Testing Tools | Pros and Cons. Retrieved from https://testinggenez.com/non-functional-testing-tools/

M. Parizi, R., Qian, K., Shahriar, H., Wu, F., & Tao, L. (2018). Benchmark Requirements for Assessing Software Security Vulnerability Testing Tools. 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Computer Software and Applications Conference (COMPSAC), 2018 IEEE 42nd Annual, COMPSAC, 01, 825–826. https://doi-org.proxy- library.ashford.edu/10.1109/COMPSAC.2018.00139

Software Testing Help. (2020). Top 10 Best Test Data Generation Tools in 2020. Retrieved from https://www.softwaretestinghelp.com/test-data-generation-tools/.

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.