ERIC WEBB

Nova Southeastern University

CISC 680 - Software Engineering - CRN – 21741

Assignment 4: (Question Set 2) - See Syllabus for assignment % and Due

1. What are the attributes of a good software test?

A good Software test should be able to communicate clearly their objectives and expected results. There should not be any argument or doubt on whether it is an effective test or what the results should be. The intent of a software test should be direct and easy to understand in such a way that a non-technical person can comprehend what the objective and tasks at hand are. A test should be significant that one can understand and grasp the importance of its results. They could be clear on the results, but one may not understand why these results are important. A good test should be isolated in the fact that it should not be dependent on others. If it depends on others then the results could be skewed depending on the results of the others. Tests should be automated so that excessive time is not spent creating and running a test. These automations should also be quick to write and quick to run. These test should be unique in such a way that the results provide a confidence that is not offered from others. A good software test typically only invokes a portion of code and does not use the code in its entirety. The test will usually deal with a certain portion of code like a single method or a particular piece of business logic. A good test should be order independent so it can be ran regardless of order of other tests, giving the tester a choice on which test to choose. It is no secret that a good test should be re-creatable and be able to be mimicked. It is said that a good test should be thought of in a manner similar to an end user. Thinking outside the box and asking questions are good attributes of a software test. Asking questions like why and why not something occurred or how something occurred can lead to better understanding of the applications functionality. Test results should be easily displayed and readable to the human eye. These are the many factors that go into a good software test.

2. Describe three control structure testing strategies.

Three examples of control structure testing strategies are Branch testing, Condition testing, and Data Flow testing. To begin, branch testing is sometimes synonymous with the term “decision testing.” Branch testing is based off of obvious decision statements such as for, while, if, etc. These branch tests can also include more subtle decision statements such as a Boolean, try-catch, and ternary expressions. Essentially they use decisions statement in the program. For these branch tests each condition needs to be executed at least once or exit. An example of this could be explained in pseudo code as follows, if dog is an instance of animal then perform bark test, if not instance of dag leave dog house. Next we have loop testing, loops are an integral part of most programs and need to be tested accordingly. There are four types of loops that need to be tested; simple, nested, concatenated, and unstructured. A loop can cause many instances to occur so one needs to make sure that each instance created is properly tested. One popular testing method when testing a program is an on change condition. For example, if a loop creates fifteen instances of a dog and the last instance changes to be a cat it alerts that a change has occurred in the loop. It is good to test simple loops by testing a few scenarios such as skipping the loop, the n-1 equivalent, and the 1+n equivalent. You should test nested loops by starting in the most inner loop and setting all other passes to their minimum values and then working from the inner loop outwards increasing passes till all loops are tested. If you are testing concatenated loops test them as simple loops if they are independent of each other, if they are dependent of each other test them as you would a nested loop. Lastly when it comes to unstructured loops it is best to redesign the looping structure or the algorithm. When it comes to basis path testing you will create a test case scenario for every method, statement, or function in your design at least once. This can be done in a flow graph notation such as basically a top down approach to testing, next you can use a flow graph notion that breaks the testing down into regions where nodes and edges meet. An edge being the control flow itself, a node being the procedural function. A node and edge have to bind and meet, this becomes a region. This way you can have different parts of you code tested that are similar to other parts of your code deriving away from the top down fashion of a flowchart. When testing independent program paths it is important to calculate the cyclomatic complexity of how many paths in your program to test for. One more useful tool for basis path testing is a graph matricie. These are a good rendition of nodes line weight along with the likely hood of how often they might occur. So in conclusion that is how Branch, Condition, and Data Flow testing make up the control structure testing strategies.

3. Why is regression testing an important part of any integration testing procedure?

When testing code it is imperative to implement a form of regression testing. This is a part of any integration testing for good reason. Integration testing aims to test individual parts of code but as a whole. While testing components of code individually you might not get the same results as if you tested them all together. This can test from the main program down toward the more atomic pieces of code called top down integration, also it can be tested from the smallest units up towards the main program called bottoms-up integration. When it comes to Objected Oriented Programing this becomes more difficult since code is not ran in hierarchal fashion but can be resolved using methodologies such as thread and use-based testing. This helps verify the program works as a whole instead of verifying small individual parts to make up as the whole. Every time a new piece of code is added, new input output paths are created establishing new data flows. You must go back in your code and verify that your new changes not only work, but they do not break any existing logic or functionality. This is why regression testing is so crucial to integration testing because it verifies that new code has not broken the old code. This can be tested in a variety of ways within the regression test suite. For example, a test suite can use a representative test to verify all the programs functionality. They can perform additional tests on code that is more likely to be changed in a more specific format. From that they can also perform tests on components that have indeed been changed. If it were not for regression testing new functional code might be added but the now broken older functionality would not be discovered till it was too late. Imagine adding a nice new user interface to a banking system, but breaking the older functionality of allowing users to withdraw funds. This would cost the banking company a lot of grief with complaints and maybe even law suits. From this example you could see how regression testing is not something to be overlooked and sometimes you must go back and test very basic functionality of code no matter how simple or small. This is why regression testing is crucial to any integration testing.

4. What are the key differences between validation testing goals and acceptance testing goals?

When it comes to validation testing and acceptance testing sometimes they get referenced in similar fashions, when in fact they have very different goals. When it comes to validation testing the goal is to make sure that the user can not break the program from their input. If a user enters an input that does not make sense it can cause faulty functionality. For instance, if you are putting in input for a numerical value you want to make sure that there are no other characters besides numbers such as an improper input of a letter or symbol. Also you would not want a number to be below 0 or above a variable finite amount. This could be an example of validation testing. If validation testing was not present users could stumble upon inputs to code that could cause potential damage. That is why it is imperative to test as many inputs as possible to guard against this type of manipulation. A popular way mitigate this is by using test driven development. You can write many correct and incorrect inputs to be tested in your validation at once instead of testing each one individually at a time. If all your tests handle the correct inputs and block your incorrect inputs then your validation testing is complete. Although validation testing can become pretty extensive because users simply have so many options to enter information, software engineers must think ahead to protect their code from all these options. That is the importance of validation testing. When it comes to acceptance testing, you are not testing user input but rather if the code is right for the user at all! You may have designed a perfect form with validation for all inputs, but if the form doesn’t send an email to its intended recipient but just goes to the next page, then it has failed its user acceptance test. Acceptance testing aims to provide whether the functionality is working as intended, or is it doing something completely at random? Another feature of acceptance testing is how difficult it is for the user to use it. Does it work as smoothly as intended? Acceptance testing is not used to protect code from the user, but to make sure the user is protected from having improper, defective, unplanned code. That is the difference between acceptance testing and validation testing.

5. Describe how test cases are derived from behavior models to facilitate interclass testing?

Since object oriented programs are not meant to be tested in a traditional hierarchal fashion, analyzing behavioral models and testing their transitions is recommended for interclass testing. This is done by looking at the transitions between classes. It is recommended to do this in a breadth-first fashion. This allows you to test transitions consisting of only transitions that were previously used. For example, if you were to create a banking application with the following three behaviors: login, account, and logout. In a breadth-first fashion you would not be able to test account or logout without first using the login. This follows the logical pattern that you must first login to be able to test your account and you obviously must first be logged in to be able to test logging out. This is an example of how the breadth-first test transitions only contain transitions previously used. Without this previously used behavior, a tester might stumble upon unpredicted faulty functionality. Although a test might work correctly in one scenario with some behaviors it might not work the same with other behaviors. This is especially true with behavior that it is not related to it as well. It also might not work properly without any previous related behavior first initiated. Starting from the breadth and working you way outwards in the code using only the same transitions allows for a more accurate representation of your application rather than testing it with unrelated behavior. This is how test derived from behavioral models facilitate interclass testing.

6. List the components of a formal specification language and describe their roles.

When it comes to developing software it can be very helpful to define and model logic in a formal specification language. This could include formal specification languages such as LARCH, Object Constraint Language, and the Vienna Development method. The point of these formal specification languages is to help with system analysis, system design, and requirements analysis to better describe the system as a whole. The beauty of formal specification languages are they allow you to design the system at a much higher level that is then replicable via a traditional programming language. Formal specification languages are typically comprised of three main components. The first being the syntax of the specification language itself. If the formal specification language does not have proper syntax in place then there is no set definition on how the read, write, or interpret the language. This could cause confusion and arguments of interpretation among developers. The second component of a formal specification language is the semantic domain. The semantic domain are the objects within the system and allows for the formal specification language to define a universe of objects that can be used to describe the system. Without the semantic domain we would not be able to concretely define what the formal specification language it truly defining. Again, this leaves room for error and argument when designing if not applied. The last component in the formal specification language architecture is the relations component. This component is what modifies and manipulates the semantic component with the proper syntax. This is what defines the relations for the data and what we can do with it. Without the relations component we would only have semantic objects but would not provide any concrete functionality to manipulate them. This would intern leave the functionality and business logic of the formal specification language up for grabs. So to recap the three component are essentially they syntax, the objects, and their relations all in the form of the syntax, semantic domain, and relations components.

7. Describe the process of writing a formal specification for some system function.

It is important to exercise your ability to write formal specifications for system functions. For this example I will be demonstrating the process of writing a formal specification for a payroll website with a single Manager and multiple Employees as Users. We will add a User to this website. This will be conducted using Z formal specification. To begin let USER be the set of all people: [User]. Next, a USER has a set of Managers and Employees where the Manager is also an Employee. The first process for this Z notation example would go as follows, the Schema name Website will have two declarations, Multiple Employees and one Manger. The invariant states that a Manager is an employee. This is represented in Z notation below. Taking note of the U as a variable indicating multiplicity and the bottom representing that a Manager is a User with the symbol.

|  |
| --- |
| Website |
| Employees : U User Manager : User |  |
| Manager  User |  |

The Next process is adding an Employee to the website. To do this we need to add an Employee to a set of Users. When doing this we need to verify the Manager does not change when a new Employee is added. This is denoted in Z notation below. Take note of the  notation to include are original website schema. Also taking not of the  symbol, unioning are Employees with the new Employee that was added.

|  |
| --- |
| addUser |
| Website  New? : User |  |
| Employees’ = Employees  new?  Manager’ = Manager |  |

From the above process we can see how that to make a formal specification. Essentially a schema needs to be defined with a schema name, declarations, and invariants. Following that the schema needs to be manipulated via the declarations and predicates.

8. Technical testing metrics fall into two major categories. What are they?

There are two categories that fall within the technical testing metrics. To begin, one category is estimating the number of techniques needed to be tested. This predicts the likely number of tests that will be required at various testing levels. Sticking to a good estimation of testing requirements and techniques builds good rapport with clients and helps sustain a good reputation in the field. It is difficult to blindly place a finite estimation of how many tests are needed that is why experience is very beneficial because an experienced engineer will have worked on multiple projects and can more accurately estimate the number of tests needed. Another good way to get an estimation of tests would be to locate documents or artifacts online pertaining to your test environment. This could be a good lead to understanding what a good quantity of tests would be. Sticking to an excessive number of tests allows for almost of a forced testing situation where you might be able to test ten things in one scenario but it is required you write 1 test for each one instead so you are not to overlook minute details. The next technical test metric focuses on the test coverage for a given component. When testing components you must think in terms of OOP. It is important to remember and consider encapsulation and inheritance. When testing individual components you might unlock new data flows and I/O paths further downstream in the code. When testing individual components its allows you to think more directly on ways to test, in contrast to a broad test pattern of the code as a whole, this allows for more specific test cases in the component. So, essentially the main differences between the two testing metric categories is quality vs quantity, with quantity being the standard estimated number of test cases that should be performed vs the quality of testing that could be done in an individual component.

9. Describe the five activities associated with the software measurement process.

When it comes to software engineering it is important to assess the quality of the product to better understand the design of the system. This can be done in direct measures like analyzing the number of code lines produced or execution speed. Indirect measures would include concepts like functionality and reliability. These direct and indirect measures have five activities associated with them in the software measurement process. To begin, formulation plays a crucial role. Formulation considers the software and applies proper metrics to perform measurement on the system, effectively formulating somewhat of a game plan for your measurement process. Next we have collection, collecting data is imperative to being able to accurately derive the formulated metrics. Without collection you would not be able to properly derive your metrics because you would have no initial data to base your metrics off of. Third is the concept of analysis. This is done once you have formulated and derived your metrics. Analysis allows for you to use mathematical tools to calculate your metrics. You can think of analysis as the final data collected from initially collecting your data. This takes that initial data and applies the metrics to later be interpreted. This brings us to the fourth activity in the software measurement process, which is the concept of interpretation. Interpretation is performed once your metrics have been analyzed. Once you have a defined analysis model of metrics you can now interpret that information and discern its importance and relevance to the project at hand. This information can be negative or positive to the applications developments and must be reviewed accordingly. After the interpretation activity has made a review of the analyzed metrics it is then time to give some feedback. Feedback is the fifth step in this activity process and allows for information to flow back to the software team. The software team takes this feedback from the interpreted metrics and applies development accordingly. So In summary the five activities in the software measurement process are formulation, collection, analysis, interpretation, and feedback in that order.

10. Describe the role of class-oriented metrics in assessing the quality of an OO system.

When designing an OO system it is important to understand the role of class–oriented metrics. A class is a fundamental unit in an OO system and encapsulates data and functions that manipulate this data. These classes can effect each other’s functions and data sets from within other parts of the code and it is important to create metrics with this in mind. One good practice for developing a metric is the weighted methods per class style, essentially the more numerous and complex the methods in a class are then the more amount of effort will be need to test that class. Another role of class-oriented metric is establishing how far down the code a class is from the initial root of the program. This is called Depth of the Inheritance tree. DIT essentially means the farther downstream a class is the more likely it will inherit functionality and will be more difficult to predict behavior for testing. Another role of class-oriented metrics is defining the number of children that will inherit from that class to better understand the possibilities of data flow downstream. Coupling obviously plays a factor in these metrics and as collaboration between classes increases the likely hood of the class will be reusable decreases. This in turn makes modifying and testing the code more difficult. Responsiveness is another role in assessing the quality of an OO system. The responsiveness for a class allows you to create a metric based on how many potential methods can be activated from a triggered event within that class. If the functionality has a potential to execute many methods then the difficulty of testing goes up. Lastly, cohesion also plays an important role because it allows you to develop a metric based on how many potential methods can modify data within that class. Do multiple methods have access to changing the value of a variable? And if so, then the testing will need to be increased for that situation. So to conclude, the role of class-oriented metrics are to define the quality and quantity of tests needed in regards to how large a class is, how complex a class is, the inheritance of a class, and how a class interacts with others.

11. Why is it important for software developers to make use of measurement to guide their work?

Measurements are important for software developers when developing an application because it provides a clarity as well as a baseline for their work. These measurements enable software developers to provide a quantitative way to analyze the quality of the product before it is even built. This provides the data to be able to develop effective requirements, designs models, code, and proper tests. If the developers did not make use of these measurements they would not have a solid foundation to base their metrics off of. This in turn would lead to improper metric interpretation that could be fed back to the developers. It is important to set measurement goals, when a measurement goal is defined developers can then create a set of questions to be developed. Answers to these questions provide the stakeholders and software team with a determination whether proper measurement was achieved. When defining a measurement it is important to not set a definition for a measurement that is to complex. If it is to complex, real world professionals will have a hard time understanding them. Along with being too complex measurements should not lead to a bizarre mix of units. Keeping the measurement simple and atomic keeps the measurement in good clarity and more easily interpreted by the developers. So it is essentially important for software developers to make use of measurements to guide them because it allows them to create better metrics based off of the data at hand.

12. Why is the "make-buy" decision and deciding whether to outsource software development an important part of the software planning process?

The make-buy decision and deciding to outsource are very important concepts for software development. The make-buy decision aims to answer whether software should be made internally or bought off-shelf to help complete a project. If a decision is used to buy third party software, an end result might be the project gets completed quicker, but at a greater financial cost. Another factor is whether the store bought software is customizable and distributable, or will it cost less to develop it in house? Another important aspect is the concept of supporting and maintaining the code. Will the off the shelf software have a service license agreement, or maintenance contract? Or could maintenance of the code be performed better internally instead of depending on a third party? Maybe the application that was purchased is of better quality then what is believed to be achieved by the current resources and times constraints of the current project. Answering questions like these help aid in the outcome of the “make-buy” decision. This is important because choosing the wrong decision of the “make-buy” can cost the project time, money, and effect quality. Similarly to deciding whether an application component could or should be bought off the shelf, is the concept of outsourcing talent and environments at the cost of quality or financial gain. Outsourcing jobs and resources brings to question whether saving money is worth the qualitative risk. In a perfect situation, outsourcing jobs and working environments saves the project money while also hopefully improving quality. Although, this obviously is not always the case. When outsourcing it is important to keep in mind you are essentially giving up control of your project. If your servers need to be rebooted and the environments are outsourced somewhere you are now dependent on that outsourced provider to perform the reboot. Similarly, when outsourcing jobs it is arguable that a full time employee typically tends to be more vetted in their work, then a contract position that is only staying around a short while. Because the “make-buy” and outsourcing decisions can have both positive and negative impacts on finances and quality of the project, it is important to weigh the measures within good reason.

13. Describe the process of building a risk table.

Risk is an everyday threat to projects and it is important to keep in mind that risk should be evaluated in its likelihood to occur versus the problems that can occur from this risk. That is why developing a risk table to better illustrate these relations is recommend. The Risk table is a simple but effective was to illustrate risk projection. Creating a risk table goes as follows. First create a table and in the first column define all the risks associated in the project no matter how large or miniscule. The second column will be the category that risk is associated too. This could be risks such as a business risk or project size risk, after defining the risks and their categories you create a third column representing the probability this risk might occur. Since this number is a bit ambiguous these numbers can very. One common example of coming up with risk probability is to get an estimation from multiple people on a team and average out the answers to be the probability you will assign to that risk. The fourth column defines the impact of the risk if something negligent were to happen. This could be from a low risk value to a high one such a minimal to catastrophic. Now that the table is complete it is time to create a risk table. To do so you simply put the high risks with high probability at the top of the table, while adding the low risk low probability to the bottom of the table. This effectively orders your risks of importance and probability. Once created you can draw a line through the middle to indicate a cut off line. Risk above the cut off line should be mitigated while risk below the cut off line can be held off. This is why risk tables are useful, because they help define the value and probability of risk and what the cut off limit is for mitigating them.

14. What is forward engineering?

When inheriting legacy systems one might be asked to forward engineer the project for better maintainability and integration of new technology. Older code bases are typically hard to incorporate new software technologies and larger old code bases can make forward engineering more robust. Forward engineering is not just recreating older code with an efficient new design, but it is actually extending the features and capabilities of the original legacy code. Forward engineering aims to improve two components of the software design. The first portion being the business process. When forward engineering the business process it allows you to improve your capability to be competitive in the market. The other portion being the reengineering of the actual information systems and technologies. This is done so that the new code will be reconstructed to exhibit higher qualities. When deciding to perform forward engineering it is important to perform a cost-benefit analysis. Questions like the annual cost of operation and the business value after reengineering, come into the decision making process. If the legacy system is set to be terminated soon then it might not be useful to forward engineer the system. If the legacy system needs to be maintained and that level of service becomes more difficult as the product scales then one might want to forward engineer the project to be more scalable with new technologies and structuring. All in all, forward engineering aims to add functionality on top of restructured legacy code to effectively improve business and system processes while making the code more maintainable and easier to integrate with new technology.

15. What characteristics need to be exhibited by organization to improve its software process?

As organizations continue to develop and maintain projects, they should be learning from these processes while simultaneously incorporating new processes to be more efficient. This is the main characteristic exhibited by organizations to improve their software process. Every time they make or modify code, they should be getting better at making and modifying the next code! This is the basic principle of improving software processes. These processes contain a myriad of umbrella functions. These functions can include activities, actions, and tasks that are mandatory within the organization. The lack of these functions can leave loop holes, blocking the knowledge transfer of the work done. This is included to improve future processes. After observing these functions, one can then analyze the data to derive a process pattern. These process patterns can reveal flaws in the software process, and lead to better understanding of these processes within the organization. Some literal characteristics in the organization to improve software analysis is to perform assessment and gap analysis. Assessment analysis from an organization would include whether processes are consistent across the organization, whether theses process are sophisticated in their performance, if the processes are widely accepted across their domain, and if the management is committed to these processes. Gap analysis performed by the organization would include topics such as the current functionality comparted to the expected end result. When the final product is made the organization will perform a gap analysis to verify all functionality was included that was intended. If not, they can derive what happened and if processes can be improved to mitigate future failures. This is why it is important for companies to display characteristics of software process improvement by performing analysis to better improve processes while simultaneously staying vigilant on these improvements.

**Certification of Authorship**



Submitted to: Professor Jeffrey Kane

Student’s Name: Eric Webb

Date of Assignment: 12/01/2019

Title of Assignment: Assignment No. 4 – Question Set Two

Certification of Authorship: I hereby certify that I am the author of this document and that any assistance I received in its preparation is fully acknowledged and disclosed in the document. I have also cited all sources from which I obtained data, ideas, or words that are copied directly or paraphrased in the document. Sources are properly credited according to accepted standards for professional publications. I also certify that this paper was prepared by me for this course.

Student's Signature: ERIC WEBB