The Proceedings of the Eighth International Research Workshop on Advances and Innovations in Software Testing

The University of Memphis
Systems Testing Excellence Program
FedEx Institute of Technology

The workshop is organized by the:

Systems Testing Excellence Program - STEP (step.memphis.edu) at the University of Memphis. STEP is an interdisciplinary research program under the auspices of the FedEx Institute of Technology that brings together the Departments of Computer Science, Management Information Systems, Electrical & Computer Engineering.

step.memphis.edu
November 3-4, 2014
Memphis, Tennessee
Unit Cost of a Functional Test Case

Mark Gillenson¹
Thomas Stafford¹
Xihui “Paul” Zhang²
Jasbir Dhaliwal¹
Yao Shi¹

November 3, 2014

Introduction

Software testing is a critical aspect of software development. With the vagaries of requirements, the complexity of designing and writing code, and the translation issues in moving from one stage of software development to the next, testing software to make sure that it accurately does what it was intended to do is a necessary part of the process. Actually, the artifacts produced in each development stage must be tested. In the earlier stages of development, such as the requirements and systems analysis stages, testing generally means conducting reviews and inspections in which the authors of the artifacts present them for comment by a knowledgeable team. While the code itself can (and often should) be reviewed in this manner, historically most software testing effort has gone into executing the code with well-planned test cases.

All types of testing represent a significant cost to the software development organization. With the pressures that these organizations face to produce, maintain, and upgrade large numbers of applications, there is a natural tendency to invest their resources in code generation rather than in code testing. However, since the importance of testing cannot and should not be underestimated, the next step is to be able to manage it effectively. One aspect managing testing is being able to gauge its cost. This is not an easy task as there are quite a few cost components to testing. These include the costs of personnel, equipment, and the time delay of moving the application into production. Of course, we understand that these are necessary costs as the later cost of putting an application with defects into production can be much higher.

Code can be tested in order to check whether several aspects of its execution meet expectations. Functionality testing is designed to validate that the code accurately performs the functions for which it was intended. Performance testing determines whether the code's response time meets its goal given a specified level of throughput. Security or vulnerability testing checks whether the code can withstand attacks from hackers. Usability testing is a matter of the user-friendliness of the code. Globalization testing is a matter of verifying that the code will work well in other countries and with other languages. All of these types of testing require the development of test cases consisting of sets of input variable values with associated expected output values.

This paper focuses on one aspect of the cost of testing: the unit cost associated with a test case used in functionality testing. There are compelling reasons for an interest in this. The number of test cases used to test a piece of code can vary widely and is a function of the manner in which the test cases were developed, which in turn is often a function of the risk associated with the code when it is put into production. Test cases for functionality testing can be created in several ways with varying costs. They can be created manually as part of the requirements writing process. They can be created by any of several “black box” or “white box” heuristic techniques. They can also be introduced from actual production inputs of an existing system that is being modified or completely rewritten. As to the risk of the code being used as a guide to the creation of test cases, it stands to reason that the higher the perceived risk of the code, the larger the number of test cases that will be desired and the larger the number of different methods used for developing the test cases.

¹ University of Memphis
² University of North Alabama
Test Case Cost Components

A test case can be designed to test a small module of code, an entire application, or any piece of code in between. What is necessary and is in common among these is a well-defined set of input variables to the code and, for a given set of input variable values, an expected output based on the applicable business rules governing the application or part of the application being implemented by that code. Thus, a test case consists of a set of input variable values and an expected output value based on how the code is supposed to function.

We began with the premise that when considering the cost of a test case, one must take into account the cost of creating it, the cost of running it, the cost of determining its success or failure, and, possibly the cost of using it to fix a defect in the code that it discovered. Thus, as an initial way of evaluating the cost of a test case, we developed the following function:

The unit cost of functional testing is based on the following factors:

- Cost to create the test case input values (CIV)
- Cost to determine the expected output of the test case (CEO)
- Cost to run the test case (CRT)
- Cost to record the test results (CRR)
- Cost to evaluate the test results (CER)
- Cost to fix a defect if found (CFD)

Unit Cost = CIV + CEO + \( \sum_{1}^{n} (\text{CRT} + \text{CRR} + \text{CER} + \text{CFD}) \)

where \( n \) is the number of times the test case is run until the defect is fixed.

The concept was that this function recognizes that the cost to create the test case and to determine its expected output is a one-time cost, while the cost to run it and work with its results is repetitive with each time it is executed.

In the next step of this research, we presented this preliminary function to 28 software testing professionals from a total of seven different companies, explained it to them, and asked for their comments on it. The interviews were conducted either in person or by telephone and ranged in time from 45 to 90 minutes.

Based on these interviews and the comments received in them, we significantly revised the above function, including adding a variety of new factors. One major area of commentary was that there are a number of preparatory activities that have to be included in the cost of a test case. However, it must be recognized that these costs must be amortized across all of the test cases. Another point was that the limit “\( n \)” in the summation should not be the number of times the test case is run until the defect is fixed, because defects are discovered only some of the times that a test case is run. Furthermore, the cost to fix a test case has to be removed from the summation for the same reason. There is also a question of who should be responsible for resolving a failed test case based on who was responsible for the failure.

The revised function is as follows:

Unit Cost = Prep Costs + CIV + CEO + \( \sum_{1}^{n} (\text{CRT} + \text{CRR} + \text{CER} + \text{CME}) \) + \( \sum_{1}^{a} (\text{CMD} + \text{CFC} + \text{CRF}) \)

where \( n \) is the number of times the test case is run
where \( a \) is the number of times a test case fails
The full set of factors is as follows:

<table>
<thead>
<tr>
<th>No.</th>
<th>Category</th>
<th>Code</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Prep Costs</td>
<td>CTP</td>
<td>Cost to create test plan</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>CPR</td>
<td>Cost to review test plan</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>CRV</td>
<td>Cost to review and validate requirements</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>CPC</td>
<td>Cost to set up test environment, including hardware and software, as necessary</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>CTT</td>
<td>Cost to acquire, develop, or upgrade test tools</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>CAD</td>
<td>Cost to collect associated test data</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>CTS</td>
<td>Cost to write temporary code needed to test part of code before the rest of the code is ready</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>CTR</td>
<td>Cost to review test cases</td>
</tr>
<tr>
<td>9</td>
<td>Creation Costs</td>
<td>CIV</td>
<td>Cost to create the test case input values</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>CEO</td>
<td>Cost to determine the expected output of the test case</td>
</tr>
<tr>
<td>11</td>
<td>Run Costs</td>
<td>CRT</td>
<td>Cost to run the test case</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>CRR</td>
<td>Cost to record and report the test results</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>CER</td>
<td>Cost to evaluate the test results</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>CME</td>
<td>Cost to collect and record test metrics, if required</td>
</tr>
<tr>
<td>15</td>
<td>Failure Costs</td>
<td>CMD</td>
<td>Cost to manage a defect</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>CFC</td>
<td>Cost to determine failure category</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>CRF</td>
<td>Cost to resolve test case failure</td>
</tr>
</tbody>
</table>

The preparatory costs, as shown in the table above, include costs associated with test plans, requirements, and various costs associated with setting up the test environment. Again, these costs must be amortized across all of the test cases that they affect.

The input values to the test cases can be created in three different ways. They can be created manually from text, which is the most expensive way. They can be created manually from pseudo-code, if the requirements were stated that way, which is less expensive. They can be created automatically from appropriate software, which is the least expensive way except that set up costs would have to be incurred. They can also be created by any of several heuristic techniques. Finally, they can be adopted from actual inputs into production systems in the case where the current development effort either upgrades an existing system or completely replaces one. The associated cost, CIV, varies among all of these various techniques. The cost to determine the expected output of the test case, CEO, is zero if the test case comes from a production system and has not been affected by requirements changes. Otherwise, the activity is a manual one with appropriate costs.

The cost to run a test case, CRT, varies depending on whether the test case is run manually or in an automated fashion. If automated, there is a cost to build the test scripts which should be amortized across all test runs. CRT also
varies with the number of physical devices, operating systems, and combinations thereof, on which the software is to be tested. Additional costs associated with running a test case include the cost to record and report test results, CRR, the cost to evaluate the test results, CER, which varies depending on whether it is being done manually or in an automated fashion, and the cost to collect and record metrics, CME, if required. Notice that the upper limit, “n”, in the summation, now reflects the number of times the test case is run, independently of any issue of test case failures.

Assigning costs associated with the failure of a test case can be controversial. First, the second summation function, which is associated with test case failures, has an upper limit of “a” which is the number of times the test case fails. We assume that the cost to manage a defect, CMD, will always be charged to the testers. This activity includes tracking the failure through assigning it to the responsible party for correction, making sure the correction has been completed, and reintroducing the test case into the mix. The cost to determine the failure category should also be borne by the testers. There are four failure categories: a code error, an error in calculating the expected output of the test case, a hardware or software problem with the test environment, or an error in the intended input values (derived from requirements) leading to an unintended negative test case. The cost of resolving the test case failure, CRF, should be assigned to the party responsible for the error that caused the failure. A code error should certainly be charged to the developers. A problem with the test environment should be charged to the testers. Errors in calculating the test case input values or the expected output should be charged to whoever was responsible.

There are several other points to be made. First, a change in a test case in a regression suite due to a change in a requirement results in the revised test case being considered as a new test case. It would be too complex to try to figure the costs associated with such a situation into the cost of the original test case. Second, all costs are assumed to include appropriate personnel costs and machine costs. Third, alternatively, a total cost of all functional test cases could be calculated. In this case, the general costs would not have to be amortized across all test cases but the costs for the individual test cases would have to be summed for inclusion in the total cost.

At the time of writing this paper, we have an ongoing effort to process all of the interview comments through Nvivo software to further refine the function.

Future work will include considering not just the cost of a test case, but the value of a test case, as a function of its cost, risk, and priority.
Automated Test Case Generator

Chuck Morgan, FedEx Services Corp.
Son Bui, University of Memphis
Mark Gillenson, University of Memphis
Ted Lee, University of Memphis

Introduction

Every newly developed product of any kind has to be tested to ensure that it correctly performs the functions for which it was designed. This is true of an airplane, an oil refinery, or a washing machine. It is also true of software, whether it is systems software or application software. Software can vary from simple applications to very complex applications and systems, and it all must be tested. The fact that in today's business environment, software applications have to function on or be accessed from panoply of devices ranging from smart phones to mainframes (each with a variety of operating systems and versions) multiplies the testing problem considerably.

Testing software to validate its functionality requires the development of carefully crafted test cases. A test consists of executing a test case and comparing the actual result to the expected result. Broadly speaking, test cases can be developed in one of three ways. They can be developed algorithmically using techniques such as pairwise analysis, they can be taken (possibly with modifications) from data from an existing application that is being replaced or upgraded, or they can be developed from requirements. This paper will focus on developing test cases from requirements.

Developing test cases from requirements has several distinct advantages:

- The assurance that the software that supports every requirement is being tested.
- The involvement of the business personnel who requested the application in the testing process.
- Traceability of a defect, based on a test case failure, to the requirement and thus to the portion of the software that was written for the requirement.

However, there are also some potential issues with manually developing test cases from requirements:

- The process is labor-intensive and can be unacceptably time consuming.
- If the person writing the test cases is not one of the business people who developed the requirements (e.g. a tester) then there is a significant time element in reviewing and understanding the requirements.
- The requirements may not have been broken down into fine enough divisions for a comprehensive set of test cases to be developed.
- A test case developed from a requirement may not correctly test it.

This paper describes an Automated Test Case Generator (ATCG) that takes requirements statements as inputs and creates test cases as outputs.
Test Case Creation Improvement by Using an Automated Test Case Generator (ATCG)

ATCG is a software tool developed internally by FedEx that can read requirements documents from the business and development teams, for the purpose of automatically detecting and generating test cases. ATCG is run on the Windows desktop by software testing personnel. After a requirements document is selected for analysis using ATCG tool, a test case can be generated as follows:

1) The ATCG electronically ‘reads’ the document as a human would do by parsing lines of text.
2) When a sentence appears to contain a requirement, it is decomposed and analyzed for content, word-by-word:
   - An attempt is made to analyze the sentence by subject-verb-predicate order.
   - Action verbs and key words are looked for in the sentence, using user configurable word lists.
   - If the sentence proves to contain a test case, then the test case is created

Once the test cases are created, they are stored in a centralized database so that everyone can review. Test cases can be edited for revisions, printed, and imported to Quality Center and emailed to others. The architectural overview of ATCG is shown in Figure 1. A search feature allows the search for test cases that are common across multiple systems.

---

The detailed steps to automatically create the test cases

There are 5 steps to automatically generate test cases based on the requirements document. Each step will be discussed with relevant examples.

1) Step 1
The ATCG loads into memory the lists of key words and phrases that will help it to identify test cases. The lists by the ATCG are shown in Table 1.

<table>
<thead>
<tr>
<th>List Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action Verbs</td>
<td>Used for determining verbs that indicate some actionable phrase</td>
</tr>
<tr>
<td>Technology terms</td>
<td>Words that are commonly used technology terms in requirements</td>
</tr>
<tr>
<td>Nouns</td>
<td>Commonly used nouns that may appear as test inputs</td>
</tr>
<tr>
<td>Adjectives</td>
<td>Adjectives used to describe the types of test inputs</td>
</tr>
<tr>
<td>Prepositions</td>
<td>Prepositions that may be used to indicate placement of test inputs</td>
</tr>
<tr>
<td>Test Data values</td>
<td>A list of test data values that can be static, random, or a range</td>
</tr>
</tbody>
</table>

2) Step 2
The ATCG parses the requirements document to look for phrases that may contain requirements reference numbers. This is accomplished by identifying numerical values formatted in patterns that typically are used for software requirements numbering. If a reference is found, it will remember requirements reference number and apply to subsequent test cases generated (this is an optional step and if no reference is found, test cases can still be generated). The examples of requirements reference numbers are SR 390241, BR 99.0011, SRS 5932022, and 66.21.49 as shown in Table 2.

[Insert Table 2 here]

3) STEP 3
The ATCG parses the requirements document looking for complete sentences. A complete sentence must start with a capital letter and end with a period. Once a sentence is identified, the ATCG looks for an action verb in the sentence from a list of user-configurable verbs. It then separates the sentence into three parts: subject, verb, and predicate. The following is an example sentence found in a requirements document: “The user interface shall support 5 digit postal codes for US Locations.”

<table>
<thead>
<tr>
<th>Subject</th>
<th>Verb</th>
<th>Predicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>The user interface</td>
<td>shall support</td>
<td>5 digit postal codes for US locations.</td>
</tr>
</tbody>
</table>

4) STEP 4
Using the predicate found in the sentence, the ATCG searches through the vocabulary lists to look for phrases that could be expected test inputs. Once a test input is found, it looks in the list of test data values to obtain a value to use as the expected input.

If a sentence, “The user interface shall support 5 digit postal codes for US Locations,” is found, then “5 digit postal codes for US Locations” will be identified as a predicate, and the two test input names below will be found within the predicate along with matched test data values in test data list as follows:

<table>
<thead>
<tr>
<th>Expected Test Input Name</th>
<th>Test Data Value found In List</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 digit postal codes</td>
<td>38103</td>
</tr>
<tr>
<td>US Locations</td>
<td>Memphis, TN</td>
</tr>
</tbody>
</table>
Finally, a test case is generated, including the following:

- A sequential test case number is generated
- The requirements reference number identified in Step 2 is included
- The test case description is derived from the requirements sentence
- Test case type is specified as positive, negative, boundary analysis, etc.
- The test inputs that were identified in the requirements, along with test data values chosen from the test data lists available
- The expected results are created from the sentence found in the requirements by making it into a declarative sentence,

The following table shows an example of a generated test case:

<table>
<thead>
<tr>
<th>Test Case Number</th>
<th>03901</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference Number Found</td>
<td>SR39029</td>
</tr>
<tr>
<td>Test Case Type</td>
<td>Positive</td>
</tr>
<tr>
<td>Test Case Description</td>
<td>Test for support 5 digit postal codes for US Locations</td>
</tr>
<tr>
<td>Expected Test Inputs</td>
<td></td>
</tr>
<tr>
<td>Test Input Name</td>
<td>Test Data Value</td>
</tr>
<tr>
<td>5 digit postal code</td>
<td>38103</td>
</tr>
<tr>
<td>US Location</td>
<td>Memphis, TN</td>
</tr>
<tr>
<td>Expected Results</td>
<td>The user interface supports 5 digit postal codes for US Locations</td>
</tr>
</tbody>
</table>

**Correcting Expected Test Inputs**

If a generated test case has expected test inputs, but the ATCG cannot locate any equivalent test data to use in the list of test data input, then the user will be notified that the test case is missing some test data. The user will then be shown a list of expected test inputs that are missing test data, and will be prompted to edit the test data inputs list to provide test data for these expected test inputs. For example, suppose that ATCG cannot locate test data for an expected test input ‘US Locations’ as follows:
Test Data Inputs List (partial):

<table>
<thead>
<tr>
<th>Test Input Name</th>
<th>Test Data Value to Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>postal codes</td>
<td>38103</td>
</tr>
<tr>
<td>City</td>
<td>Chicago</td>
</tr>
<tr>
<td>Employee ID</td>
<td>99999</td>
</tr>
</tbody>
</table>

Expected Test Inputs for Test Case:

<table>
<thead>
<tr>
<th>Expected Test Input Name</th>
<th>Test Input Value found</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 digit postal codes</td>
<td>38103</td>
</tr>
<tr>
<td>US Locations</td>
<td>NO TEST DATA FOUND</td>
</tr>
</tbody>
</table>

If needed, the ATCG can generate an email to the requirements author to clarify the expected test input's purpose.

**Available Functions in Automated Test Case Generator (ATCG)**

To make ATCG more use-friendly and easier to use, the following features/functions are available as tabs on Windows interface: Documents, Requirements, Test Cases, Vocabularies, Tools, Preferences, Test Data, and Help. The full descriptions of these features/functions are provided in Table 3.

[Insert Table 3 here]

**Summary and conclusion**

Testing software to validate its functionality requires the development of carefully crafted test cases. Test cases can be developed algorithmically using techniques such as pairwise analysis, they can be taken (possibly with modifications) from data from an existing application that is being replaced or upgraded, or they can be developed from requirements. In this paper, we try to find a better, systematic way to develop test cases automatically from requirements in order to alleviate some potential issues with manually developing test cases from requirements as mentioned earlier in the introduction section.

To improve the process of creating test cases, ATCG is internally developed by FedEx that can read requirements documents from the business and development teams, for the purpose of automatically detecting and generating test cases. After a requirements document is selected for analysis using the ATCG, a test case can be generated as follows:

1) The ATCG Tool electronically ‘reads’ the document as a human would do by parsing lines of text.
2) When a sentence appears to contain a requirement, it is decomposed and analyzed for content, word-by-word.

Once the test cases are created, they are stored in a centralized database so that everyone can review. Test cases can be edited for revisions, printed, and imported to Quality Center and emailed to others.
The Automated Test Case Generator (ATCG) will be evaluated for use on a regular basis. It should be noted that the accuracy of test case generation is heavily dependent on the quality and standardization of the requirements documents. When used correctly, the ATCG can provide the following benefits:

- To improve the productivity of test case design
- To increase test coverage over manual test case writing methods
- To improve end-to-end testing by identifying test cases that affect multiple systems
- To save time in generating test data by having the TCG to automatically choose test data

In addition, we believe that an agile version of the automated test case generator could also be created that reads user stories, and then generates test cases continuously, as the user stories change.

As a future study, we need to conduct an empirical study by collecting some data to confirm whether the promised benefits are actually realized and to what extent.
<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Requirement Name</th>
<th>Requirement Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>6.7.21.1</strong></td>
<td><strong>SR 5421503</strong></td>
<td><strong>Support Variable Length Postal Codes</strong>&lt;br&gt;The system shall support alpha-numeric postal codes of variable length not to exceed the database limit of 5 characters.&lt;br&gt;BR 6.1.1</td>
</tr>
<tr>
<td><strong>6.7.21.2</strong></td>
<td><strong>SR 5421522</strong></td>
<td><strong>Support Alpha-Numeric Sector Codes</strong>&lt;br&gt;The system shall support two character alpha-numeric sectors on the user interface.&lt;br&gt;BR 6.1.2</td>
</tr>
<tr>
<td><strong>6.7.21.3</strong></td>
<td><strong>SR 5421527</strong></td>
<td><strong>Modify Time Segment</strong>&lt;br&gt;The system shall support minute-resolved time segments.&lt;br&gt;Current default: 15 min resolution&lt;br&gt;Proposed change: 1 minute resolution&lt;br&gt;BR 6.1.3</td>
</tr>
<tr>
<td><strong>6.18.1.8</strong></td>
<td><strong>SR 5421539</strong></td>
<td><strong>Support Mass Updates</strong>&lt;br&gt;The system shall provide a “mass update” capability to manage a one route to many relationship.&lt;br&gt;BR 6.1.4</td>
</tr>
</tbody>
</table>

**Table 2**
<table>
<thead>
<tr>
<th>Tab</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documents</td>
<td>View the list of documents that have been submitted for test case generation, and search all documents for test cases using key word search capability</td>
</tr>
<tr>
<td>Requirements</td>
<td>Create a requirements document profile to be analyzed for test cases, and begin the process of test case generation</td>
</tr>
<tr>
<td>Test Cases</td>
<td>View the test cases generated for the currently opened requirements document</td>
</tr>
<tr>
<td>Vocabularies</td>
<td>View/edit the action verb and terminology vocabularies used for test case analysis</td>
</tr>
<tr>
<td>Tools</td>
<td>Access online tools useful in the testing process (Requirements Management, etc.)</td>
</tr>
<tr>
<td>Preferences</td>
<td>Set the test case generation preferences to be applied when generating test cases</td>
</tr>
<tr>
<td>Test Data</td>
<td>View/edit the list of test data parameters used for automatic test data generation</td>
</tr>
<tr>
<td>Help</td>
<td>Perform a sentence analysis test of the test case generator to generate a test case</td>
</tr>
</tbody>
</table>

Table 3
W-Model Testing: Software Validation Using Requirements Simulation

Jong Seok Lee
Department of Management Information Systems
The University of Memphis
Memphis, Tennessee, USA
jslee4@memphis.edu

I. INTRODUCTION
Software validation is a critical element of software testing. Validation is often referred to as “are you building the right thing?” [1], as it is centered on evaluating whether a product, service, or system meets the needs of customers or stakeholders [2]. Software development can fail when a software product does not meet the needs of users, even if the product is built according to specifications.

Software validation is conducted most often during or at the end of the development process. IEEE Standard Glossary of Software Engineering Terminology indicates software validation is “the process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements.” [3, p. 80] However, this common practice, or perception that validation is conducted during or after the development process can be problematic, because validation errors detected during or after the development process can be problematic, because validation errors detected during or after the development process are very expensive and difficult to fix [1].

In order to prevent software validation errors that occur during or after development, many organizations conduct validation tests on some aspect of software at an earlier stage in the software development cycle; examples include validation of requirements and validation of designs. While such an approach has certain merit, it is difficult to test a software product that is yet to be developed based on requirements or design. Furthermore, in most software development projects ‘low tech’ tools, such as Word, Excel, and Visio are used for capturing requirements and design. Validating requirements and design based on such tools is very challenging, as these tools cannot simulate the actual behavior of software products. Against this backdrop, in this paper I propose W-Model testing based on requirements simulation, and discuss how W-Model overcomes the current challenges associated with software validation.

II. DEFINITION AND APPLICATIONS OF SIMULATION
According to Dictionary.com, simulation is defined as “imitation or enactment, as of something anticipated or in testing.” Simulation is centered on building an artifact, or a model that represents key characteristics, behaviors, or functions of the target object (e.g., a computer program). Essentially, creating a simulation involves mimicking the operations of a target object in a real-world situation. Simulation is used in a variety of contexts, including training pilots, testing the safety of an automobile, and predicting financial markets.

III. SIMULATION IN SOFTWARE DEVELOPMENT
In software development, in recent years simulation tools have been developed and become available. Simulation tools enable software development teams visually define and capture requirements (i.e., requirements simulation). Requirements simulation is similar to prototyping in the sense that it focuses on building an artifact that mimics actual behaviors and functions of a software product. However, unlike prototyping requirements simulation does not require programming. Indeed, current simulation tools have intuitive functions, and allow building a software simulation without writing a single line of code.

The fidelity of a simulation is the degree to which a simulation involves details of a software product in terms of business logics, procedures, rules, and interfaces. In other words, the more realistic a simulation is the higher
fidelity the simulation is. For example, one can build a simulation that implements an entire set of requirements, and looks exactly like a final product. A high-fidelity simulation takes longer and more effort to develop, but it tends to reduce errors and re-work during the development process.

IV. BENEFITS OF REQUIREMENTS SIMULATION
With simulation tools gaining momentum, using requirements simulation is believed to have several benefits. A report published by SAP suggests that there are three major benefits of using requirements simulation; 1) it helps align business and IT in the design phase, 2) it helps reduce the costs by facilitating requirements validation, enabling faster development, and reducing rework, and 3) it helps validity application usability and facilitating adoption [4].

V. W-MODEL TESTING
The V-model (Figure 1) is a well-known methodology that represents the software development process from a testing perspective. The V-model illustrates how each phase of analysis and design (left axis) corresponds to each phase of testing. The center pillar is development, which separates analysis & design activities from testing activities. One underlying assumption of the V-model is that meaningful testing activities (both verification and validation) cannot begin until development has been completed.

Contrary to the V-model, the W-model, which I propose in this paper, assumes that a meaningful software validation is possible before development, and is enabled by requirements simulation. Specifically, in the W-model requirements simulation can be used to validate a software product before the development process begins as requirements simulation allows analysts, and testers to see and experience actual behaviors and functions of an anticipated software product. Furthermore, analysts can get users involved and get them to interact with simulation. In other words, in the W-model, validation can include user acceptance testing as well as testing of features and business logics.

Fig. 1. V-Model Testing

Fig. 2. W-Model Testing
One consideration for the W-model is that more work may be required before development can begin. While simulation tool vendors suggest that using requirements simulation can reduce costs and effort associated with requirements analysis and design, building a requirement simulation can be an effortful process. In other words, while adopting a simulation tool may save time required to write text requirements and designs, it adds more time to the software development process as it involves building a simulation – particularly building a high-fidelity simulation may require a significant amount of time. Nevertheless, additional time and effort made into building a simulation can be beneficial in the end, as requirements simulation enables an early software validation (before development), and reduces costs and effort associated with potential errors and rework caused by gaps in requirements.

VI. CONCLUSION

Requirements simulation is a new trend, and further research is warranted to probe its effectiveness, and develop best practices. Requirements simulation holds promise as an enabler for early software validation. Particularly, W-based model proposed in this paper suggests that software validation can be conducted following building a simulation and before development, and such validation tests may include not testing software features and business logics, but also user acceptance testing.

REFERENCES

A Procedure for Ordering Requirements in Agile Development

Michael Racer
Marketing and Supply Chain Management Department
University of Memphis
Mark Gillenson
MIS Department
University of Memphis
John Dugan
Coroutine, Inc.

November 3, 2014

Abstract
One of the challenges of creating a successful project in agile development is the identification of the sequence for developing modules within the program based on the stated requirements. The challenge here is that the difficulty of finding the best solution is a function of the number of modules to be created plus other factors, giving rise to a NP complete problem. We will start with a simple example

The Problem
First, we present the content of the system and the basic definition of those units to be scheduled.

- 10 requirements A-J
- 10 software modules to be developed
- Capacity of each iteration: 16 person days
- Each requirement has:
  - Development cost and testing cost in person days
  - Priority, 1-5, 1 is highest, 5 is lowest
  - Risk, 1-5, 1 is lowest, 5 is highest
  - Testing cost is directly related to risk
  - Possible mandatory ordering among requirements

We will consider the following properties to influence the decision-making process.

- Some modules will be developed and tested in the same iteration. Any necessary rework will be done in the next iteration.
- Some modules will be developed in iteration n, tested in iteration n+1, and have any necessary rework done in iteration n+2.
- New requirements may be introduced during development.
Sort the Matrix

- Sort the rows of the matrix by highest priority then lowest risk, then lowest cost.
- This strategy prioritizes the most important items that are most likely to survive the testing process and actually make it to production within the given timeframe.
- However, the capacity remaining in an iteration may cause ordering changes.

<table>
<thead>
<tr>
<th>Req</th>
<th>Dev Cost</th>
<th>Pri</th>
<th>Risk</th>
<th>Test Cost</th>
<th>Total Cost</th>
<th>Mandatory Ordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>7</td>
<td>A must be in Iteration 1</td>
</tr>
<tr>
<td>E</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>9</td>
<td>E must be done before G &amp; H</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>F must be done before G &amp; H</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>H</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>A must be in Iteration 1</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>I</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>J</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>-</td>
</tr>
</tbody>
</table>

Beginning with this said we start by sequencing the units in order to minimize production costs. As we monitor the progress through each step of the production period. As we do this we will also consider the fact that conditions may change as development proceeds and other challenges discussions among the participants introduces.

This capability to respond to new conditions is one advantage of the agile approach framework as well as one of the challenges of it. We shall see that a wide set of conditions can change and new ones can be introduced as we proceed.

**Iteration 1**

- Iteration 1, begins by implementing the requirements with priority 1, Reqs A and F.
- Both their development and testing will be performed in It 1.
- Their combined total cost is 11, leaving another 5 person-days of capacity in It 1.
- There are two requirements with priority 2, Reqs D and H.
- D’s total cost of 7 exceeds the remaining capacity in It 1 and so does H’s total cost of 6.
- So, a decision is made to perform only the development but not the testing of H (H is lower in risk than D) in It 1 at a cost of 5 person-days.
• Development and testing conducted on the code for Reqs A and F during It 1 and possibly right after It 1 were successful.
• The requirements matrix going forward no longer includes Reqs A and F.
• Since the code for Req H was developed but not tested, it remains in the matrix but with a development cost of 0.

<table>
<thead>
<tr>
<th>Req</th>
<th>Dev Cost</th>
<th>Pri</th>
<th>Risk</th>
<th>Test Cost</th>
<th>Total Cost</th>
<th>Mandatory Ordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>9</td>
<td>E must be done before G</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

Iteration 2
• It 2 begins with the testing of Req H.
• Then it picks up the remaining priority 2 requirement, Req D, for development and testing.
• The total capacity needed is 8 person-days, leaving 8 person-days of additional capacity.

• The priority 3 requirements are C and I but only Req C, with a total cost of 3 for development and testing, will fit in the remaining capacity of It 2 and so it is included. Note that Req C is lower in risk than Req I.
Finally, only the development of Req I is included in It 2 due to the limitation of the remaining capacity in the iteration.

After Iteration 2, testing indicated that the code for Requirement C needed 1 more person-day of development time to fix discovered defects and 1 more person-day of testing. Also, the testing of Requirement H indicated that it needed more work: 2 more person-days of development and 2 more days of testing.

In addition, a new requirement, Req K, with Priority 1, was introduced into the project.

Since Req I was developed but not tested in It 2, it now appears in the matrix with development cost 0.

<table>
<thead>
<tr>
<th>Req</th>
<th>Dev Cost</th>
<th>Test Cost</th>
<th>Total Cost</th>
<th>Mandatory Ordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>H</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>I</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>J</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

Iteration 3

Since the new Req K is Priority 1, its code is developed and tested in the next iteration, It 3.

The rework on Req H is performed and so is the rework on Req C.

Finally, the testing of Req I, which was developed but not tested in the previous iteration is performed in It 3.
It 3 successfully completed the development (or rework) and testing of the code for Reqs K, H, and C.

The testing of the code for Req I indicated that it needed more work: 2 more person-days of development and 2 more person-days of testing.

It’s worth noting how the process of the method has allowed us to modify the production of tasks H and I.

The mindset of the agile environment has allowed us to take on the mindset that an operation is not necessarily 100% complete the assigned iteration and we can be flexible in addressing the problem and possibly using that opportunity to further explore the costs within this environment.

In addition the introduction of Katie at this step illustrates one large student in the agile process that is a capability that can be absorbed as the procedure goes on. And we will study further in later steps how can modify the process even more by taking more information and more challenges into account.

### Iteration 4

- Iteration 4 will begin with the needed rework of Requirement I.
- It will also include the development and testing of Requirement B.
Because of the stated mandatory ordering between Requirements E and G (plus the higher priority of Requirement E), and the limited capacity left in Iteration 4, the development of the code for Requirement E will be done in Iteration 4, but not its testing as Iteration 4 does not have enough remaining capacity for Requirement E’s testing.

- The code for Req I was tested successfully and so it was completed.
- The code for Req E must still be tested, since it was only developed in the previous iteration.
- The testing of Req B indicated that it needed more work: 2 more person-days of development and 3 more person-days of testing.

**Iteration 5**

- It 5 will include the rework of the code for Req B.
- The testing of Req E.
- The development and testing of Req G.

<table>
<thead>
<tr>
<th>Req</th>
<th>Dev Cost</th>
<th>Pri</th>
<th>Test Risk</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>J</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Reqs B, E, and G, were successfully tested in It 5 and so are completed.

Since only 5 iterations were planned, it was decided that the development and testing of Req J will be held over for a later release.

*Consider how the path could change*

Now that we have a solution let’s consider the fact that other options were available to us as we proceeded to this point.
One of the biggest changes in the system came when we introduced task K. We started the procedure without having initial consideration of task K. Yet during the process it was deemed to be worthwhile to consider. Given this we should consider some new angles that are introduced. Decision-making must be guided by more information.

What's new?
- What is the value of each component in our system?
- What happens if something has to be left out of the finished product?
- How do we decide how to finish?

To reduce the complexity let’s begin by supposing that we’re just focusing on an iteration and then take a different position.

Let’s begin by recognizing that there are two main issues we must consider in the process - time and cost.

Focus on iteration five
- We don’t have the capacity to implement every module
- So we have to postpone one module
- Using our basic model, we ended by eliminating J.

This reality that we can’t do it all suggests that decision-making is a challenge for us. And we really must consider the possibilities on how to make that choice.

Possible issues
- Value of a module
- Interactions with other modules
- Timing restrictions
- What about extending the deadline
- What about incurring additional costs

Consider the options.

New Options Iteration Five

With this simple look at the last iteration, we must realize that we can carry this problem back

But there’s more
- Remember, K was introduced early in the process.
- So we could back up our analysis to the previous iteration as well.
Picture in your heads a large network of paths and nodes and you can realize that there are quite a few pathways that lead to the final solution. This complexity would increase dramatically if we step back one more iteration in the process. This would result in the introduction of eight more alternatives to be considered. Briefly we will discuss the complexity of the problem as well as how it can be addressed by the decision-makers.

**Complexity**
- Notice the rapid increase in the number of options for evaluation
- Two approaches to progressing through a graph
  - Depth first
  - Crossword

**Summarizing**
Objective agile offers the set of decision-makers the opportunity to explore the options available to them.
Increase the flexibility with respect to time and cost in its way. These two factors are most extreme important for us to consider both the postulates for introducing new opportunities and exploring the challenges for selecting our final destination.
And because all contributors in the system are participating throughout the process, decisions based on money and time can therefore be shared amongst all participants.
A well structured approach allows the development team to investigate
- time restrictions
- resource availability
- risks
- mandatory ordering
- costs
Abstract— Software has evolved as a critical aspect of the medical devices. Apart from being used in making decisions or collecting clinical data, software is now used to control the medical devices. However, malfunctioning software in the medical device can cause life threatening situations. The software development community has been facing complex and ever increasing design challenges to harmonize software in or as medical devices while mitigating any associated risks. This paper highlights the importance of an effective software design along with validation considerations within the framework of IEC 62304:2006. The key concepts (design and testing) along with challenges in the development of software in or as medical devices are explained in this paper.

Keywords—software; medical device; design and testing; risk; regulations

I. INTRODUCTION

The medical device industry has evolved a lot over the past several decades and the role of software has become increasingly important as in any other growing industry. Software, pervasive to any industry, is created for various medical purposes as well as non-medical purposes in the healthcare industry. Software is used to make clinical decisions, collect clinical data, and even to control medical devices among various other applications. Other non-medical purposes of software include administrative support, financial management, collaboration and information sharing. We can broadly classify the software in medical device industry into two main categories: 1) Software in a medical device; and 2) Software as a medical device.

A. Software in a medical device

Software in a medical device can be considered as a component that is embedded in or is a part of the medical device. The medical device may have several components making up its functionality and a software program is one of these components that are used to assemble this device. Implantable devices such as pacemakers, implantable cardioverter defibrillators (ICDs), neurosimulators, and implantable drug pumps are all programmable devices and these use embedded computers to monitor patient’s conditions and automatically introduce therapy in the patient when needed [1]. These smart devices have revolutionized the level of care provided to patients with chronic cardiovascular or neurological disorders. As an example, we will elaborate on the working of ICDs and how software plays a critical role in its effective functionality. ICD monitors and responds to heart activity by using the inbuilt modes of pacing – sending small electrical stimulus to the heart periodically. In the event of an abnormal heart rhythm, the device can send a larger shock for rhythm restoration. ICD is typically implanted below
patient’s clavicle and close to the skin where electrical leads are connected to the heart muscles to monitor heart’s activity. Through an external software program, a physician can actually perform diagnostics, read and write private data or adjust the therapy settings in the device (in other words – can reprogram). Malfunctioning software in the medical device can cause life threatening conditions by not sending the electrical stimulus when it is needed or by sending a large shock when it is not needed by the patient.

B. Software as a medical device (SaMD)

According to the definition provided by the International Medical Device Regulators Forum (IMDRF) [2], the term Software as a Medical Device (SaMD) is defined as “software intended to be used for one or more medical purposes that perform these purposes without being part of a hardware medical device”. A medical device is a machine or an instrument that is intended for in-vitro use for a variety of medical purposes including but not limited to diagnosis, prevention, life-support, monitoring and treatment of disease. Some of the in vitro diagnostic (IVD) medical devices can come under the framework of the SaMD as these devices don’t require hardware for any functionality. In the example of ICDs mentioned above, the diagnostic program which is used to connect with the actual implanted device can also constitute SaMD.

The software development community has been facing complex and ever increasing design challenges and therefore requires greater focus and emphasize on the software design and validation (testing). The software development process is generally guided by the available standard IEC 62304:2006 on medical device software [3]. The IEC 62304:2006 defines the standard or the set of processes that should be followed for developing medical device softwares as part of the software development lifecycle. It is an international harmonized standard that has been adopted by United States and European Union. The adoption of the IEC 62304:2006 standard for software design ensures that the end software product has been developed using a controlled and well-defined process and has met all the process requirements based on the safety class of that software. According to this standard, each life cycle process is divided into a set of activities that need to be carried out. Most of these activities are further divided into set of tasks within the standard. This report will mainly focus on the software design and validation considerations for the software developers within the framework of software development process outlined in IEC 62304:2006. Focusing on the software in medical device industry, we will explain the key concepts along with some challenges in the development of software in or as medical devices.

II. SOFTWARE DESIGN FOR MEDICAL DEVICES

An effective medical device is often supported by an equally effective piece of software. Software plays a crucial role in device’s operation as well as in ensuring patient’s safety. Software can provide multiple functionalities to a medical device and some of the examples are monitoring heart or brain signals, controlling electric pulses, and recording and analyzing data. Some of these devices can also be lifesaving and hence the software efficacy becomes even more critical.

Most softwares operate on an underlying hardware platform and with the technology advancements, periodic updates are needed for both hardware and software. However, performing multiple surgeries to update the hardware is not practical as most of the implantation procedures, especially for the permanent medical devices are invasive. Because of this, software for such devices needs to be designed in a more robust manner to be able to perform well even on an outdated hardware. This requirement often leads to a complicated software design which requires even more structured testing.

The following sub-topics explain the key design considerations and tools for the software designers while designing the software in a medical device / healthcare industry.

A. Design Principles

The medical device software products should not only be designed keeping in mind the basic software design principles like separation of concerns, modularity, abstraction, reuse etc. but also take
into consideration the special regulations that the product is subjected to within a particular country [4]. Some of the design principles are elaborated below:

- **Separation of concerns**: Design principle separation of concerns involves separating the various software features into distinct sections, each addressing a separate concern. The sections should be implemented in a way which results in high cohesion and low coupling in between these sections. When concerns are separated into segments, each segment can be worked upon or modified independently and thus increasing overall efficiency in the development process.

- **Modularity**: The principle of modularity is further specialization of the separation of concerns principle. It involves separating software chunks according to their functionality or responsibility and each component or module is responsible for only a specific feature or functionality.

- **Plan ahead for Reuse**: Reuse principle is using a piece of software for multiple uses whether it’s in the same program or across multiple programs. Programmers can reuse tested code segments, functions, classes, templates etc. whenever appropriate. Software The reuse of software can reduce repeated work and thus can save time and energy. Software designers can create software libraries with custom code and reuse the libraries for similar software applications thus reducing the development and testing time significantly.

B. **Risk Analysis**

The errors made by the designers or programmers in software development project or process can result in the software malfunction or failure thus introducing risks with the software products. According to the FDA, as many as one-third of the reported medical device failures are failures of medical device use rather than the device itself [5]. In 2011, the recalls in medical devices attributed to software failures reached over 20%. The key to manage and mitigate these risks lies in knowing the basic concepts of software risk management which include risk index, risk analysis and risk assessment [6].

Risk Analysis in medical device software is a critical step as any failure in the software can sometimes lead to the catastrophic situations. Risk analysis is a structured tool that can help predict potential issues in any process including the use of a medical device. Identifying and analyzing potential risks in any software or medical device design at an early stage can save a lot of production cost as well as improve quality of the end product [7]. Some regulatory bodies have a mandate on medical device companies to perform and document a thorough risk analysis during the design phase. IEC 62304:2006 provides guidelines to classify software in medical devices based on the software risk and potential hazard that software could result in. These classifications are [8]:

- Safety class A: No injury or damage to health is possible
- Safety class B: Non-serious injury is possible
- Safety class C: Death or serious injury is possible

Risk analysis is a part of the design validation process, which is done to ensure that the product is well designed. The design validation process is very similar to the software validation which tests software against its requirements. Risk analysis has been included as a part of the software design based on some real incidents. For example, between 1985 and 1987, a radiation therapy machine that was controlled by a computer overdosed six people leading to highly toxic results [9]. Risk classification is another important concept in software risk management. It deals with analyzing risks and grouping similar risks together into classes. Some of the software risk classes include software requirement risks, software design risks, software cost risks, software scheduling and planning risks, software quality and safety risks, and software business risks. If a proper risk management process is taken into consideration for the software development process, future problems could be minimized or even completely eliminated.

C. **Human Factor**

Food and Drug Administration’s (FDA) historical perspective on the “human factors” defines it as “a discipline that seeks to improve human performance in the use of equipment by means of hardware and software design that is compatible with the abilities of the user population” [10]. Most of the software
products that are designed for the medical device industry involve human interaction. A user’s behavior is often influenced by the design and characteristics of the medical device or the equipment he/she has been operating and any user-interface that has ambiguity or lacks logic can lead to patient injuries including deaths. It will be important to reiterate that as many as one-third of the reported medical device failures are failures of medical device use rather than the device itself [5]. Hence, it becomes really important to include and analyze the interaction between the operating environment, user capability and the device during its design phase. There are various dimensions to human abilities that can play a role in the software design and need to be considered as a human factor.

- **Physical and Sensory ability**: Physical and sensory characteristics in humans include vision, hearing, strength, reach etc. Various software design features can alter the human performance and make the device prone to malfunctioning. Some of these design decisions include text font size and clarity of various symbols used, level of volume for audible components etc.

- **Perceptual and cognitive ability**: Users can vary in their ability to perceive and process any sensory input and understanding this important ability can play a great role in a software design. Software designers can consider using existing conventions that people are used to reacting based on their past habits and make it easy for the person to use a device. For example - color red symbolizes danger and green indicates safe and this information can be used for alerts / notifications.

- **Home use**: Some medical devices or equipment that are meant to be used by the doctors and trained professionals actually end up being used by patients in the premise of their homes, which may not provide the ideal operating environment or conditions and may thus lead to wrong interpretations or incorrect use of the equipment. The device design should include this factor and built with proper safety keeping in mind that it can be used by non-professionals.

Many of the software design related errors especially those relating to human factors can be prevented by following some of the below given important design considerations.

- Consistency and clarity in displaying information, abbreviations, and formats is very important.
- Designer should try to consider the existing conventions for symbols and language and exploit users’ past experience with these instead of building new conventions or challenging the user’s expectations.
- The user should not be overwhelmed with information that can be difficult to read or perceive. E.g. text that is not properly formatted or densely packed information or information not displayed for sufficient amount of time can be an issue.
- Use reminders and prompts to inform the user for important steps.
- Always keep the user informed about the current device status and whenever possible have dedicated sections on the device to display any critical information.

**D. Prototyping**

Prototyping is a tool that software designers can utilize to get feedback from the users during the early phases of the project. A prototype can be a mock-up or proof of concept or a simulation of the actual product. The users of the software can evaluate the software design by evaluating the prototype at an early stage. The users can try out the software functionality, enabling them to change/improve the requirements, and also to suggest any improvements if there see any early issues. There are various kinds of prototyping that can enhance the quality of a software product. Some of them are:

- **Throwaway prototyping**: In this prototyping technique, the developed prototype is discarded and never used in the final product. The method used in building the prototype is quite informal and the prototype is developed very quickly after a relatively short investigation.

- **Evolutionary prototyping**: In evolutionary prototyping, a robust model is built in a highly structured manner and is constantly revised as needed. The prototype then becomes an important part of the final product and the software system is continuously refined and rebuilt.

- **Incremental prototyping**: In this technique, several prototypes are built separately and these separate prototypes are then incrementally merged to get an overall design of the product.

- **Extreme prototyping**: Extreme prototyping is mainly used in web development and consists of three sequential phases: the first phase is a static prototype comprising of HTML pages; second phase deals with
data processing using a simulated services layer; and in the final phase the services are implemented and integrated into the prototype.

The main advantage of prototyping is the ability for early troubleshooting thus reducing time and costs for the project along with better user engagement. Also, any functionality that might have been missed can be identified and the users get a better understanding of the system being developed. However, software prototyping must be carefully dealt with because it comes with many limitations. Some disadvantages of prototyping include insufficient requirement analysis, confusion among users, increased scope and increased costs if not monitored properly.

E. Agile Software Methodology

The medical device industry has to continuously deal with the challenge of quality, associated costs and timing of producing the medical devices. There can be high quality medical device products that are cost-effective but are too late to enter the market. Also, quality products that are released on-time may end up being very costly for the manufacturers and hence don’t stand the competition. Classic software development methodology has been the waterfall model where all requirements need to be analyzed before design can start, and design needs to be finalized before development can begin and so on. This approach is sequential in nature and requires a lot of rework if requirements change in the middle of the development. This may not be the most efficient approach to develop software and is being replaced by newer principles like Agile software development. In Agile approach, requirements and software design can change in real time and development always happens in increments. Software is developed relatively quicker and iteratively with extensive customer/end-users involvement. This may be a more effective and efficient approach depending upon the situation and the needs of the software.

Traditionally thought, agile software methodology aims to achieve speed while compromising quality. However, this notion is being challenged and has shown to be incorrect [11]. It has also been reported that using the conventional development model (waterfall) a developer can spend between one-third and half their time on troubleshooting and rework which is reduced significantly with the agile methodology [11]. Agile model however is designed to reduce this rework time and in a way that quality can be ensured right from the beginning. In the traditional waterfall approach, coding and testing only begins when the requirements are fixed. However, often the requirements change and this leads to rework and slippage in schedule. In agile approach, chunks of code are written and thoroughly tested, relatively quicker and a continuous improvement of the software takes place in real time. Individual pieces of code/functionality are continuously integrated and automated unit tests continuously perform the testing. Best practices are followed by highly skilled programmers and team sizes are generally small and co-located for better communication. Shared ownership and requirement of working pieces of code at each iteration by the business help prevent any surprises and also balance out the pace at which the product is developed [11].

There are a lot of myths surrounding agile practices and organizations that have adopted this methodology but the use of genuine agile approach have defeated many of these myths. Changing existing processes and adopting new ones is always difficult and is often resisted by organizations especially when there are time constraints involved. However, these new evolving practices advocate ability to respond quickly which the biggest competitive advantage can be in today’s highly competitive industry.

F. Software Security

The software security risks associated with medical devices have increased significantly as more and more of these devices are now being connected to internet, internal networks, other medical devices and even smart phones. Failure in medical device can be due to various reasons ranging from hardware failure, software failure or improper use of the device. While it is important to consider these criterions because they play a significant role in the device’s functioning, it is equally important to consider security-related failures in medical devices and come up with software designs that are robust and
secure. The FDA is recommending the medical device manufacturers and health care providers to take appropriate steps in ensuring that the correct safeguards are in place when it comes to software security, to reduce the risk of cyber-attacks or unauthorized access to the systems.

While computer-related failures leading to injuries or deaths are reported as per FDA’s guidelines, however, no such national databases exist when it comes to security-related failures [12]. Still, the evidence that is available indicates that numerous security threats and incidents occur in medical devices each year, as reported by the hospitals. Medical device failure due to security vulnerabilities can lead to various consequences including disciplinary actions, monetary fines and loss of reputation which is another reason why such incidents are not widely reported. Thus the actual incidences of security failures may be much larger than the available statistics. For that reason, medical device manufacturers and software designers need to be careful and deliberate in considering security related hazards and take the software security as one of the important design considerations. Some guidelines that should be considered in processes and the product development thus limiting the opportunities for such attacks are listed below:

- Access to medical device and/or software involved in medical devices should be controlled and steps to limit unauthorized device access should be taken. Some examples include user authentication via username and password, biometrics, physical locks on data centers, card readers, access monitors etc.
- Design approach that retains the device’s critical functionality during an attack must be adopted to avoid life-threatening consequences for the patients.
- Deployment of security patches and methods that restrict other softwares from modifying the authenticated code should be implemented whenever possible.
- An incident response plan that considers disaster recovery and retention when the security is compromised will help analyze the situation and can even avoid further consequences.

G. User Interface (UI) Design

Communicating, understanding, designing and managing requirements for a software medical device user interface is a difficult task for the medical device manufacturers. Many times a good interface design for a software product is not entirely clear at the beginning of the project. Human factors (as discussed above) play an important role in design and development of the user interface in a software product. However, there are many factors besides human factors that are equally important in the optimal design of the user interface. Many diversified groups of people including lawyers, marketing personnel, clinical specialists should be included in the UI design process. These groups of people can have varying inputs affecting the user interface of medical software. While human factor experts can help in the optimal design and content layout and navigation schemes on a UI, but they may not be aware of patents owned by other competitors that can control certain aspects of the interface. Also, some devices have to be designed with legacy features in mind so that the users are not overwhelmed or confused with the new-generation software. This useful insight generally comes up when a team of software engineers, project managers and human factor engineers work together on the product.

Communicating and documenting software requirements related to user interface can be a challenging task for many. There is generally not one medium that adequately addresses and communicates all the requirements in different phases of the product. So experts recommend using different tools and other sources for defining various stages of the interface design [13]. In an early stage of the project, the requirements can be captured in text documents and/or verbalized by the team. In the later stages, storyboarding, presentations and other visual mediums should be used to communicate/document the look and feel of the interface. And in the final stages, professional software can be used to create mock-ups and screen samples for an effective communication. Rapid prototyping (as described above) may also prove useful in collecting and communicating these requirements to/from the user. Whatever medium is used to communicate the requirements, emphasis should be given to the understandability of the requirements by all team members; the requirements should be flexible for any changes and should represent what the user will see on the screen or the device display.
III. SOFTWARE TESTING FOR MEDICAL DEVICES

Verification and Validation are two terms that are most commonly used for testing a piece of software. It is important to understand the meaning and the differences between these two activities that complement each other. Verification is an activity that checks whether the software product was built right, whereas validation is used to make sure that the right software product was built for the user. The FDA considers software validation to be “confirmation by examination and provision of objective evidence that software specifications conform to user needs and intended uses, and that the particular requirements implemented through software can be consistently fulfilled”.

The FDA offers multiple guidelines for validating software in the medical devices. Various aspects like planning verification, testing, traceability, configuration management, change management, risk analysis among many other aspects are part of the software validation process.

In each of the development phases of a project, completion of certain tasks can conclude that the end product is validated. Of course, the order of these tasks and the timing of their performance will depend on the software development model that is chosen and also on the risk/complexity of the overall system. These tasks include quality planning, requirements, design, construction, and various types of testing. Below we shall briefly describe some of these different aspects of the software validation process along with discussion on the various challenges that need to be addressed for a validated software product.

A. Requirements and Specifications

An important aspect of software validation is well documented software specifications and requirements that define the “intended use” of the medical device or equipment. The software requirements should not only cover the features/functionalities of the medical device software, but also document requirements related to performance, quality, error handling, security, device startup and shutdown, alarms, and business logic etc. A requirement is an expectation for a system or a way its software should operate from the business / end user perspective. On the other hand, a specification is defined as a document that states the requirements [14] . A document that specifies the requirements serves as a baseline for performing verification and validation activities. Requirements deal with identification, analysis and documenting the information about the software and/or the medical device and how it is supposed to function. Some areas that are covered are user characteristics, anticipated tasks/functionalities, risks and hazards. Requirements cover system inputs as well as system outputs, various functions that a system will perform, the ways user will interact with the system, all data ranges and limits, the intended operating environment, how errors will be handled, the various system interfaces and interactions, and the performance expectations of the system among others. Typical tasks that take place during requirements phase are risk analysis, traceability analysis, describing user characteristics, listing characteristics and limitations of the system, user interface requirement analysis, system test plan, acceptance test plan and ambiguity analysis.

Software requirements should also be validated to make sure that there are no inconsistencies within the requirements, all performance requirements should be clearly laid out, safety and security requirements should be thoroughly covered and are correct, all requirements should be measurable and testable and finally the project analysts should ensure that the software requirements adequately address the safety hazard of the medical equipment.

B. Verification and Validation

Verification and Validation of a software product is essential to assure the quality of the product and to ensure that all the software operations meet the product requirements. Several medical device recalls in the 1990s were attributed to software faults or defects after which the FDA required software validation as part of quality system regulation [15]. According to the 21 CFR part 11 compliance, any software that is a part of a medical device or used to automate device production or any system that
evaluates the quality of the product will be required to be validated. A number of systems are subjected to comply with this regulation and hence it is important to understand the importance and processes behind software verification and validation.

Verification activities (including technical reviews, walkthroughs, software inspections, requirements traceability, unit testing, integration testing, system testing, acceptance testing, and audits etc.) provide objective evidence that the outputs of all phases of software development lifecycle has met the specifications written for that phase. It looks for consistency, completeness and correctness of the software during its entire development process and subsequently provides support for the validated software. Validation activities on the other hand can be conducted either during the software development lifecycle or at the end of the software development. The scope of the software verification and validation need to be defined as it is not possible to test the software for the continuous duration. Hence, the goal of software validation should be to develop a “level of confidence” that the device meets all its requirements. Various measures like defect analysis, defect prevention, testing coverage, including other techniques are used to establish this level of confidence before the final product is released. The extent to which these activities are performed depends on the level of risk associated with the software product that is being developed.

C. IQ/OQ/PQ

- **Installation Qualification (IQ):** A documented proof that all equipment and resources have been delivered and successfully installed and are in accordance with the statutory regulations constitutes an installation qualification. The qualification plan generally consists of an IQ test plan along with an IQ report. The IQ test plan contains tests and test steps taken to make sure the installation was correct and complied with the regulations. An IQ report is a document that summarizes the activities conducted as well as the results/finding of the installation qualification. Any deviations along with procedure/steps taken to correct those are also documented.

- **Operational qualification (OQ):** Operational qualification is done to evaluate the functioning of a device of software product. An OQ ensures that the system meets all its specifications and all items in the test plan are successfully executed and documented. A successful OQ is an essential prerequisite for the technical acceptance of any device and hence marked as a critical process in the validation. The proof for successful OQ generally consists of an OQ test plan along with an OQ report. The test plan contains detailed test cases that match the device specifications whereas the OQ report is used to summarize and evaluate the results of an OQ. All deviations are noted and analyzed to eliminate them until the product is termed defect free.

- **Performance Qualification (PQ):** The purpose of PQ is to verify and document that the device/equipment has met the stipulated performance criteria and is working within the specified working range. PQ comprises of proof that the device produces the expected defined results, physical tests that prove that the specified limits for stability are met within the product and validation done via several repetitions and reproduction of the processes.

D. Types of Testing

Software testing can be classified into several categories based on different parameters. Below we have covered a few most common types of software testing:

- **Unit testing:** In unit testing, software developers/testers test individual units of software for their functionality. A software unit is the smallest piece of software that can be tested. This ensures that each part of the program meets the technical specifications and helps in identifying any issues early during the developments stage. Unit testing if done the right way can simplify integration testing using a bottom-up testing approach. Typically, unit testing is done using white-box testing methodology in which test cases are designed to test the internal structures of a unit and the developer/tester tries to cover inputs that go through each path of the program to determine the appropriate outputs.

- **Integration testing:** In integration testing, individual tested units of program are combined together and tested as a group. Test cases are designed to test the interactions between various pieces of code usually using the black-box testing methodology. Various approaches can be taken for integration testing including bottom-up, top-down and sandwich testing which is a combination of the top-down and bottom-up approaches. In black box testing, program is tested for different inputs that should give the expected outputs. Tester is
generally not aware of the internal working of the program and basically only knows about the functionality of the software without knowing the working details.

- **System testing**: System testing is done to test the complete system as a whole and to ensure that the system meets its stipulated requirements. System testing takes all integrated software components that have passed the integration testing as an input to detect any inconsistencies between the various components. There are various types of system testing like user interface testing, performance testing, load testing, security testing etc.

- **Regression testing**: Regression testing is performed to find defects in existing working software especially after a change in the code has been made. The main reason to perform such a testing is to make sure that a new change in code has not affected other working pieces of code.

- **Acceptance testing**: Acceptance testing, also called user acceptance testing, is performed by the customer/user of the software as part of the software rollout or handover. This is usually done in the client machine and in the customer environment.

- **Functional vs. non-functional testing**: Functional testing is done to test a particular functionality or specific action of a software whereas non-functional testing focus on the aspects that are not directly related to a specific function e.g. performance, security, scalability etc.

- **Alpha testing**: Alpha testing is type of an operational testing by the user/customer done at the developer’s site during initial rollout of the software. It is a form of internal acceptance testing that is performed before launching the product to a broader customer base.

- **Beta testing**: Beta testing takes place after completion of alpha testing and is a form of external user acceptance testing. The software product is launched to solicit feedback from a limited audience involved in the testing of the software. This helps in identifying any issues which could have been missed out by the software developers.

**E. Software verification and validation tools**

The software tools also known as “Computer aided software engineering (CASE)” tools help in software verification and validation process in a various ways. These tools help in reducing the effort required to repeatedly test the software and thus giving more time for thorough testing. Also, the accuracy of software validation can be measured using these tools resulting in improved software quality and productivity. Apart from the general administrative tools like word processors, electronic mail, conferencing systems, and notes systems, tools like static analyzers, configuration management tools, reverse engineering tools, tracing tools, and testing tools help tremendously in the software validation. Some of these tools are briefly explained below.

- **Static analyzers**: Static analysis is performed to evaluate a system based on its form, structure or documentation [15]. The analysis is done using reviews or software inspections and includes activities like control flow analysis, data-use analysis, range-bound analysis, information flow analysis, code volume analysis, code standards analysis, and complexity analysis among others.

- **Configuration management tools**: Configuration management tools help in change management and have the ability to store various versions of the software being developed. They also help in maintaining traceability along with build/release management. Access control (check-in / check-out) is usually a valuable feature of these tools which enable teams to effectively work together on a common software product/application.

- **Reverse Engineering tools**: Reverse engineering tools can provide useful information during the product development process, which can verify whether the final built product conforms to its design.

- **Tracing tools**: Tracing tools are used to ensure that the identifiers used adhere to the naming conventions set by the project. These tools are used to make sure that the identifiers used in the program are unique, to store traceability records, and to provide change history etc. Overall, tracing tools allow easy and efficient navigation throughout the software.

- **Testing tools**: A number of testing tools are in the market (free and paid) that can help in various testing activities including automated testing, generating test cases, analyzing code, debuggers etc. More tools are available for executing the test cases and analyzing the results but there are only limited tools that can be used to design a test or for test case generation.

**F. Regulations**

There are a number of regulations including but not limited to ISO 13485, ISO 14791, 21 CFR 820 and 21 CFR Part 11 regulating the medical devices. Unlike IEC 62304:2006, the regulations are not guidance
documents but provide regulations for the developers or manufacturers to comply with and can result in fines or other penalties in case of a non-compliance. Medical device manufacturing companies are regulated by the FDA (in US) in two different ways. Firstly, the company must prove the safety and efficacy of the device for its intended use to the FDA. And secondly, all quality system regulations are met by the manufacturer. The software that is embedded in a medical device is particularly of interest to the FDA due to increased risk introduced for having software built into the device. The software failures can be hard to detect and can have unfavorable consequences and thus validation of the device software and the system that measures software quality is critical. It is noteworthy that the FDA has published several guidelines and offer training courses that can help companies understand and comply with these regulations. Some of the relevant regulations for the medical device software are listed below.

- **21 CFR Part 11**: This regulation is part of Code of Federal Regulations in USA and defines the criteria under which FDA considers electronic records and signatures to be trustworthy, reliable and equivalent to the paper records [16].
- **21 CFR 820.30**: The regulation, again a part of the Code of Federal Regulations in USA, focus on the design controls of the medical devices including software on or as medical device. The regulation requires each manufacturer of any class III or class II device and selected class I devices to establish and maintain procedures to control the design of the device in order for it to meet the design requirements set for it [17].
- **Cybersecurity for medical devices**: FDA recommends that medical device manufacturers and healthcare industry take appropriate measures to safeguard the software systems from cyber or network vulnerabilities [18].
- **Regulations for clinical trials**: FDA has several guidelines as well as regulations which have to be kept in mind when designing or maintaining software systems that are used for clinical trials [19]. Data collected in clinical trials has to be validated and follow various other compliances including HIPAA.

There are a lot of challenges that the device manufacturers have to deal with in order to comply with these regulations. Any change in technology brings about a change in the regulatory environment as well. The regulations and guidelines that apply to software development generally assume a traditional lifecycle model such as waterfall model which is a sequential model of developing software. The newer software development models like “Agile” do not really fit well in these existing guidelines and thus require special efforts on the part of the software developers. However, by understanding the reasons behind these changes, can actually help device manufacturers to be ahead of the curve and for this it is imperative for them to keep up and understand the FDA’s regulations [20]. The companies should adopt a pro-active approach to make sure that their products are compliant.

**IV. CONCLUSION**

The software design and validation are critical aspects of the software in the medical device industry. The design and testing of software in medical devices requires special efforts on part of the developers due to associated risks and challenges involved. In this report, we highlighted some of the key design and testing concepts to be considered by the developers working on the software in medical device or software as medical device.

**REFERENCES**


Testing of Software Designs in Health Bio-medicine

Son Ngoc Bui
Department of Management Information Systems
Fogelman College of Business and Economics
University of Memphis
Memphis, TN 38152
snbui@memphis.edu

Robin Poston
Department of Management Information Systems
Fogelman College of Business and Economics
University of Memphis
Memphis, TN 38152
rposton@memphis.edu

Abstract – Testing of software designs in health bio-medicine has been come central to the success of developing bio-medicine devices. The current software testing literature has very limited knowledge on the factors influencing software designs in health bio-medical devices. Draw from socio-technical theory, our literature review suggests that (1) social factors, and (2) technical factors affect the quality of testing software designs for bio-medical devices. Implication for theory and practices are also discussed.

Key word – software design, software testing, socio-technical theory, health biomedicine.

Introduction
Over the last decade, bio-medical organizations have spent relatively large amount of resources toward software-driven, custom hardware devices in applied research such as brain-computer-interfaces, implantable/telemetry monitoring devices, smart biomaterials, and advanced biosensors due to Patient Protection & Affordable Care Act signed in 2010 [6]. In fact, the market reached $65.6 million in 2012, and expected to grow more than 4% annually. Some bio-medical solutions such as Electronic Medical Record devices was even expected to grow more than 15% [6]. By its nature, bio-medical devices are developed and influenced by bio-medical research and software development fields [1]. Many devices are used for clinical purposes to improve patient care. Those clinical purposes range from telemedicine to e-Healthcare systems [7, 11]. Given the rising demand for designing bio-medical devices, the need for highly skilled bio-medical engineers and technicians to understand bio-mechanics and biological systems while being able to write good hardware and software systems requirements when modeling and analyzing alternatives designs for developing new bio-medical devices is a critical issue. Recent medical device recalls that purportedly met efficacy, safety, and security regulations are examples of how poor quality of software development can influence the failure of bio-medical devices [4, 10]. Healthcare and bio-medical experts are found to lack strong systems lifecycle skills [3, 8, 12, 13] One primary barrier to health care industry improvements is the lack of technical capabilities in systems-engineering approaches, which involves the ability to define the problem, model the system, analyze alternatives, implement options and assess what worked [2]. The challenge today is how engineers trained in bio-mechanics and biological systems can be trained to write good hardware and software systems requirements when modeling and analyzing alternatives designs for developing new bio-medical devices. Our literature review examines the quality of testing software design in health bio-medicine. We suggest that (1) social factors and (2) technical factors influence to the success of bio-medical devices. Our study
contribute to this research stream by proposing additional factors contributing to the long-term values of medical devices.

**Literature Review**

Bio-medical devices are machines which are used for diagnose of disease and intended to affect structure and functions of human body [1]. The United States Food and Drug Administration (FDA) has identified that developing bio-medical devices are complex software development processes which requires the understanding of multiple discipline, including engineering design, software development, and software testing of bio-medical devices [1]. Given limited guidance, developing a high quality software designs for bio-medical devices is still a biggest challenge for many organizations.

Drawing from socio-technical theory, we identify two major factors influencing the success of bio-medical devices: (1) social factors, and (2) technical factors. The social factors comprise the doctors, nurses, patients, and any values, skills, and knowledge that contribute to the health bio-medical environment. The technical factors comprise the bio-medical devices, techniques and any bio-medical systems that enhance patients’ health. The theory suggests that software testers need to understand the unique fit between social factors and technical factors in bio-medical environments to better test software design of bio-medical devices [5].

Table 1 summarizes selected studies for the two proposed factors.

<table>
<thead>
<tr>
<th>Independent Variable</th>
<th>Dependent Variable</th>
<th>Findings</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usability of software testing [3]</td>
<td>Customer satisfaction</td>
<td>Propose steps to design usability:</td>
<td>Social factor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Identify the usability problems and propose solutions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Agree on process to change usability design</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Define workflow models and usability roles to redesign usability</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Integrate usability into software development.</td>
<td></td>
</tr>
<tr>
<td>Physiological data by using wireless technology in body sensor [9]</td>
<td>Communication between doctors and patients</td>
<td>Provide benefits of how body sensor can be used to capture data, and improve communication between doctors and patients.</td>
<td>Social factor</td>
</tr>
<tr>
<td>Healthcare testing for interoperability [12]</td>
<td>Quality assurance</td>
<td>Introduce a new testing methodology for interoperability of healthcare systems:</td>
<td>Technological factor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The new testing method is dynamic to run in multiple test system interface by using multiple protocols</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The new testing method make use of existing database collected from clinical and hospital environments.</td>
<td></td>
</tr>
<tr>
<td>Model-Based Testing [13]</td>
<td>Interoperability of multiples interfaces from different vendors</td>
<td>Provide some benefits and challenges of applying model-based testing to generate test cases for Siemens healthcare software systems:</td>
<td>Technological factor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Healthcare products, particularly those that are highly integrated, pose significant challenges on the overall model design to address the scalability.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Healthcare products that embed a large volume of domain knowledge and business</td>
<td></td>
</tr>
</tbody>
</table>
### Proposed Model

The purpose of this study is to identify potential factors influencing the success of testing software design in bio-medical devices. To achieve this goal, we use socio-technological theory to identify two main factors contributing to the success of testing bio-medical devices. Figure 1 below represents our suggested model:

<table>
<thead>
<tr>
<th>Social Factors</th>
<th>Technological Factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient Use</td>
<td>Interoperability</td>
</tr>
<tr>
<td>Doctor Use</td>
<td>Safety</td>
</tr>
</tbody>
</table>

#### Social Factors

- Attitude of medical staff toward healthcare IT adoption is the most important factor.
- Perceived service risks were the second most important factor.
- Effect of perceived service benefits on healthcare IT adoption were not significant.

#### Technological Factors

Propose how a real-time virtual heart model has been created to help monitor and meet the safety guideline for electrophysiological operation of the functioning and malfunctioning heart.

---

|---------------------------------------------------------------------------|---------------------|--------------------------------------------------------------------------------------------------------------------------|

---

**Figure 1. Proposed Conceptual Model**

---

**Social Factors**
I. THE SOCIAL FACTORS EMPHASIZE THE IMPORTANCE OF LEARNING HOW PEOPLE USE BIO-MEDICAL DEVICES TO THE SUCCESS OF TESTING SOFTWARE DESIGNS. IN THIS STUDY, WE SUGGEST THAT UNDERSTANDING (1) PATIENTS’ USE, AND (2) DOCTOR’S USE CAN IMPROVE THE TESTING PROCEDURE.

II. IN BIO-MEDICINE, DEVICES CAN BE USED TO DIAGNOSE DISEASES OR EVEN CHANGE HUMAN BODY’S STRUCTURE [1]. FOR SOME CASES, THE DEVICES COULD ALTER PATIENTS’ DAILY BEHAVIORS OR EVEN COMPLETELY PREVENT CERTAIN ACTIVITIES TO OCCUR. BECAUSE OF THE DISRUPTION OF HUMAN’S BEHAVIORS CAUSED BY THE DEVICES, TESTING FOR PROPER SOFTWARE DESIGN IS VERY VITAL TO THE USAGE OF BIO-MEDICAL DEVICES. IF THE DEVICES ARE NOT APPROPRIATELY TESTED THEIR DESIGN, THE DATA CAPTURED BY THE DEVICES MIGHT BE INACCURATE OR EVEN MISSING. IN SOME INSTANCES, THE MISSING/INACCURATE DATA EVEN CAN LEADS TO WRONG TREATMENTS OR MISDIAGNOSE DISEASES. THUS, IT IS IMPORTANT FOR SOFTWARE TESTERS AND DOCTOR TO KNOW HOW THE PATIENTS CAN USE THE DEVICES. LEARNING THE PATIENT’S USE OF BIO-MEDICAL DEVICES CAN HELP SOFTWARE TESTERS TO FIND A BETTER WAY TO TEST THE DEVICES.

Besides patient’s use, understanding how doctor use bio-medical data is also crucial to software testing. For different treatments, doctors normally require various data captured from their patients [1]. Some data is very basic and can be easily obtained. Some data might require a long period of time to acquire. Thus, it is important for software testers to discuss with doctors to learn how the data can be used and help patients. Learning how doctor use the data captured from bio-medical devices can shorten the testing procedure of software design, leading to better quality of the devices.

Technical Factors
The technical factors emphasize the technical issues of bio-medical devices. In this study, we suggest that (1) interoperability, and (2) safety of devices have strong impacts on the testing quality of software design. In bio-medical settings, devices are manufactured by multiple parties. Those devices are typically support different functions, have various data output, and cannot communicate with other systems/devices [12]. That leads to interoperability issues which limit the exchange of data and communication across different bio-medical devices. While testing for bio-medical devices, software testers should pay attention to the standard protocols used in those devices. If those standard protocols are poorly tested, it will likely to limit the information exchange among devices, resulting in missing data or even completely loss of data.

Besides interoperability issues, the safety of the devices are also important to software testing. Many devices are used to alter human’s body structure for treatment purposes [1]. They plays a critical roles to support the treatment or even to cure many diseases. Because of their impact on patient’s health, the safety of the devices is probably the biggest concern with many parties, including patients, health care providers, and governments. With many recent recalls from US FDA, testing for safety of bio-medical devices must be met the FDA’s guideline.

Conclusion and Implications
In this research, we used socio-technical theory to suggest the two main factors influencing the success of software testing in bio-medical devices: (1) social factors, and (2) technical factors. This study has paid attention to the use of user acceptance theory in testing stage. Much of the extant research has examined the theory in usage stage, and has limited accumulated knowledge in design and testing phrases. Our research provides the potential usage of user acceptance theory in earlier stage to better design and test bio-medical devices. By understanding the potential data usage of doctor and patients, we can have better testing and design procedures for bio-medical devices.

Our study also suggests the consideration of testing theory. In testing literature, there are limited testing theory can be applied in bio-medical testing context. This research is one of the first steps providing the
needs to develop testing theory in bio-medical environments. The testing theory can be an important milestone to improve the testing actives and overall high-quality product outcomes. Besides the theoretical contribution, this study also provide several practical contributions. First, understand patient’s usage of medical devices is critical for software testing. It can provide software testers a better procedure to test the devices, and improve the products quality. Second, learning how the data is used by doctors is also vital for software testing in bio-medical settings. Doctors require different data captured from their patients, and use data differently to diagnose and treat their patients. Thus, software testers can shorten the testing procedure of software design by understanding how the data is used by doctors. Finally, interoperability and safety issues are big issues in bio-medical environments. Software testers need to test for standardized protocols used in bio-medical devices, and need to meet the guidelines provided by FDA.

References
Medical Device Systems and 510 (k) Submission Testing

Dr. Robin Poston
Management Information Systems
The University of Memphis
Memphis, TN
rposton@memphis.edu

Seungho Choi & Aneta Dziemianczyk
Management Information Systems
The University of Memphis
Memphis, TN
Schoi8@memphis.edu & kdzmnczy@memphis.edu

Abstract—The purpose of this paper is to clarify some of the uncertainties and explain how the process of filing and submission of the 510 (k) application works. It provides the reasons of application rejections and delays and gives guidance how to avoid the mistakes that can cause those rejections and/or delays. It also proves that adding testing step into the process could improve it significantly.

Keywords—510(k), testing, FDA process, medical device

I. INTRODUCTION

Submission of applications for the FDA is not only quite complicated and time consuming but also risky due to the fact that not all applications are accepted and then processed. Properly filled and submitted forms are the key to the success and although it seems obvious and easy, many submitters have problems with them and wonder what the correct answer should be. Medical device system approval is facing many challenges related to the submission of the 510 (k) applications and the acceptance rate is not as high as it could be. At the same time, certain steps have to be thoroughly followed in order to achieve the goal.

II. BODY

First step in the process requires in-depth understanding of what type of device and why is going to be marketed. Before marketing, devices have to pass the number of tests and approvals; however, the U.S. Food and Drug Administration provides the option for developers to file so-called 510 (k) application which is nothing but the “premarketing submission made to the FDA to demonstrate that the device to be marketed is a safe and effective, that is, substantially equivalent (SE), to a legally marketed device that is not subject to premarket approval (PMA).3

To successfully file the 510 (k) application, the device has to be considered a “medical device” which means it needs to be an instrument, apparatus, implement, machine, contrivance, implant, in vitro reagent (chemical reactions outside the organism), or other device that is recognized in the official National Formulary, intended for use in the diagnosis of disease, cure, treatment or prevention of the disease, or intended to affect the structure or any function of the body. It is crucial to follow the steps provided by the FDA.

3http://www.fda.gov/MedicalDevices/ProductsandMedicalProcedures/DeviceApprovalsandClearances/
One of the great ways to improve the medical devices and healthcare solutions is the use of the technology that we know in today’s world is helpful and almost necessary. One of the examples of the technology used is the VitalConnect platform. VitalConnect was the first wireless monitoring system that monitors physiological data and captures clinical-grade biometric measurements in a continuous and configurable manner. VitalConnect was created in 2011 and is a pioneer in development of the technologies that help with the most challenging healthcare issues the world is facing today. Its wearable biosensor platform, that has a potential of being integrated into a variety of medical applications, received the FDA approval in July 2014 through the 510(k) process. That biosensor captures the biometric measurements, such as heart rate, respiratory rate, skin temperature, electrocardiography, and then is sent to the central location for analysis. That allows the authorized parties access the data and react when the parameters are outside of the normal range. Receiving FDA clearance for the Vital Connect platform is an important milestone in the future of the United States healthcare system,” said Vital Connect CEO Nersi Nazari. We are very proud of this achievement and look forward to working with partners to develop new and impactful methods for integrating the Vital Connect platform into a variety of clinical health solutions.

Although some devices are successfully approved and cleared by the FDA, there are many of them that are not. Certain steps need to be taken to improve those rates. On average, about eighty percent of applications were accepted by the FDA in the last decade, which is the equivalent of about three thousand clearances. Below is the graphical presentation of the revision time by the FDA. We can see that since 2006 it has been increasing steadily, and although it has been slowly decreasing in the last three years, it still takes about five and a half months.

The graph below shows that the type of the device also matters when calculating the processing time. We see that in general, the radiology devices are cleared within the period of four months while dental devices take two more months to clear.

Usually, once the FDA receives the 510(k) submission, within a week it will send the either the Acknowledgement Letter or a Hold Letter to the submitter if there are some unresolved issues with the eCopy or fees. By the 15th day, the FDA will conduct the Acceptance Review and inform the submitter if the 510(k) was accepted for Substantive Review or placed on the Refuse to Accept hold. By day 60, the Substantive Review is conducted and the FDA uses the Substantive Interaction to communicate with the submitter if either it will proceed with the Interactive Review or if the 510(k) will be placed on hold and Additional Information are needed. By day 90, FDA sends final Medical Device User Fee Amendments (MDUFA) Decision on 510(k). If MDUFA Decision is not reached by Day 100, FDA provides Missed MDUFA Decision Communication that identifies outstanding review issues.

Before the submission and acceptance can occur, the device has to be categorized. The factor used for the classification involves the risk of the device. FDA established three categories of devices, low-risk, medium-risk, and high-risk and premarket approval devices. The 510(k) process involves the medium-risk category of medical devices. The devices from the first low-risk class, are those that do not have to be approved by the FDA because they are not designed in any way to treat potentially fatal conditions and if misused, are unlikely to cause life-threatening harm. The third group considered the high-risk group of medical devices is also not eligible for the 510 (k) clearances because those devices need first to pass the Premarket Approval. The general description of those devices involves those that are implanted in the body, are designed, and intended to support, sustain and even save lives, and at the same time present the high risk of injury or death is used improperly. The medium-risk class devices, those called 510(k) devices, must be approved by the FDA through the 510(k) process and are usually not intended to treat potentially fatal conditions, but they can cause harm if misused. Some of the examples of the class II devices are acupuncture needles, powered wheelchairs, infusion pumps, and surgical drapes.

The next graph presents the summary of the 510(k) process. The testing step, assigned to it what would be implemented on every level of the application submission is the step we believe would not only improve the submitters to better understand the process as a whole but also hopefully increase the number of the applications properly and fully prepared for the 510(k) process that would increase the odds of passing 510(k). As presented, each step/ question is extremely important and needs to be fully answered or else the rejection or the delay of the process will be expected. As mentioned before, the first question for the submitter is to establish that the device he is going to present to the market is, in fact, a medical device. Once that is known, the next question is whether the device is a subject for the 510(k) process, meaning whether there is substantially equivalent device already on the market and that
the device falls under the category of class II medium-risk class of devices. The new and the predicament devices need to match in certain ways and be equivalent mainly in the areas as intended use, material, design, and the differences have to be specified. Does the new device raise new safety or effectiveness questions? Does it have the same or different technological characteristics and if so, are they affecting the safety and effectiveness of the new device? The submitter has to provide information about any other safety and effectiveness issues the new device raises and whether there are any scientific methods that exist for assessing effects. Those methods have to be accepted by the FDA. And finally, the application has to include the performance data that shows the devices equivalence.

<table>
<thead>
<tr>
<th>Step</th>
<th>Details</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Is the product a medical device?</td>
<td>Medical device – an instrument, apparatus, implement, machine, contrivance, implant, or in vitro reagent, or other device: • Recognized in the official National Formulary; • Intended for use in the diagnosis of disease or other condition, or in the cure, mitigation, treatment, or prevention of disease; • Intended to affect the structure or any function of the body</td>
<td>Document Set</td>
</tr>
<tr>
<td>2. Is the device subject to 510(k)?</td>
<td>• Is there a legally marketed device that has roughly the same safety and effectiveness characteristics? • Check the risk of the new product – if a high risk of harm or death – device should be class III and require FDA approval</td>
<td>Comparison Test Characteristics</td>
</tr>
<tr>
<td>3. Does the new device have the same indication statement as the predicate?</td>
<td>Compare new product with the predicate one - intended use, indications for use, studies, design, materials</td>
<td>Comparison Test Characteristics</td>
</tr>
<tr>
<td>4. Do the differences in the indication statement raise new issues of safety or effectiveness?</td>
<td>Yes – class III and NSE</td>
<td>Expert Review</td>
</tr>
<tr>
<td>5. Does the new device have the same technological characteristics as the predicate?</td>
<td>Defined in section 513(j) of the FDC Act</td>
<td>Comparison Test Characteristics</td>
</tr>
<tr>
<td>6. Could the new technological characteristics affect safety or effectiveness?</td>
<td></td>
<td>Expert Review</td>
</tr>
<tr>
<td>7. Are the descriptive characteristics precise enough to ensure equivalence?</td>
<td></td>
<td>Expert Review</td>
</tr>
<tr>
<td>8. Are there any new types of safety or effectiveness questions?</td>
<td>Eyes = NSE</td>
<td>Expert Review</td>
</tr>
<tr>
<td>9. Are there any accepted scientific methods that exist for assessing effects of the new characteristics?</td>
<td>Methods may include validation or test methods for material fatigue, hardness, or other. If method not accepted - NSE (Non Substantial Equivalence)</td>
<td>Test Plan</td>
</tr>
<tr>
<td>10. Are performance data available?</td>
<td>Develop a comprehensive test plan to assess equivalence and the worst case scenarios that can occur with new features of the product, if not – NSE</td>
<td>Test Results</td>
</tr>
<tr>
<td>11. Do the data demonstrate equivalence?</td>
<td>Substantially equivalent when: • Has the same intended use AND has the same technological characteristics OR • Has the same intended use AND has different technological characteristics and the information DOES NOT raise new questions on safety and effectiveness AND demonstrates that the new device is at least as safe and effective as the legally marketed device</td>
<td>Test Evaluation</td>
</tr>
</tbody>
</table>

Adding the testing step to the submission of the 510(k) process brings a hope for the success, given that many mistakes are made by submitters on every level of the process. In 2013, sixty percent of new 510(k) medical device submissions were rejected by the FDA. The reasons for rejection varied from very trivial ones like failure to locate materials within the document or improper wording of required
statements, failure to provide the particular test report, simply following the check list rather than using common sense to others like failure to fully address FDA’s original questions, responding to the questions in the way that raises new safety concerns. Those reasons counted for about 88 percent of rejection cases. About 16 percent of rejections came from the presentation of poorer performance of the device than its predicate. About 60 percent of the applications were prohibited from proceeding to a substantive review until the identified deficiencies were addressed and majority missed shelf live and biocompatibility information. For the purpose of weeding out the applications that lacked the basic information to preserve the Agency’s limited resources, the Refuse to Accept Policy for 510(k) came into effect in 2012. At the same time, the policy was created to help submitters by providing them with the feedback within the fifteen days after the FDA received the submission. The quality if data and details are not reviewed at that moment, but rather just to catch the administrative mistakes that give the submitter a chance to correct them and resubmit the application without paying the application fees for the second time.

Below we present the 510(k) submission cleared by the FDA for the last several years.

![Graph showing 510(k) submissions cleared by the FDA for various years](http://www.emergogroup.com/research/fda-510k-review-times-research)

### III. CONCLUSION

Again, to receive a positive response from the FDA, the submitter has to remember to follow and address any and all guidance documents and standards, perform the required tests, provide the clinical data for devices, fully describe the intend for use of the device and its technical standards and remember that the 510(k) process is only intended for the devices that have the same indications for use as a device already on a market and that the differences in the indications for use between the device and the predicate must not alter the intended use of the new one.

It is crucial to master the regulatory processes since the number of the medical devices introduced is constantly growing. Medical devices’ testing is heavily manual and needs to move toward automated environments. That requires the thorough training of people, capturing the requirements of procedures in sufficient detail, being ready to provide all required proof and test results, holding internal audits and controlling, tracking, and managing data electronically.

### REFERENCES

[http://www.emergogroup.com/research/fda-510k-review-times-research](http://www.emergogroup.com/research/fda-510k-review-times-research)