Suitability of FIT User Acceptance Tests for Specifying Functional Requirements: Developer Perspective

Grigori Melnik, Kris Read, Frank Maurer

Department of Computer Science, University of Calgary
Calgary, Canada
{melnik,readk,maurer}@cpsc.ucalgary.ca

Abstract. The paper outlines an experiment conducted in two different academic environments, in which FIT tests were used as a functional requirements specification. Common challenges for functional requirements specifications are identified, and a comparison is made between how well prose and FIT user acceptance tests are suited to overcoming these challenges from the developer's perspective. Experimental data and participant feedback are examined to evaluate whether developers can use requirements in the form of FIT tests to create a design and implementation.

1. Introduction

It is common knowledge that two thirds of all software projects today fail (either by being terminated, going overtime, going over-budget, or because they deliver only partial functionality). Ambiguous or incomplete software requirements along with poor quality control are two of the biggest contributors to these failures [7].

Despite the fact that quality control is a major cause of project failure, it is still often overlooked by project teams. Eighty-three percent of organizations' software developers don't like to test code [2]. One of the reasons is simply a lack of time to perform diligent and proper testing, which is frequently the result of inadequate planning and time overruns in other activities. When testing is performed, often it is done at the level of unit tests by the development and/or testing team. However, the goals and mentality of testers may not entirely correspond with those of the customer. Acceptance tests are needed to ensure customer satisfaction with the final product. Acceptance tests also serve as regression tests, to ensure that previously working functionality continues to behave as expected. These tests are often created based on a requirements specification, and serve to verify that contractual obligations are met. This creates a dependency between the requirements specification and acceptance test suite, a dependency that may involve a great deal of overhead. Changes to one side necessitate changes to the other, and effort is needed to ensure that the written requirements correspond precisely to the expected test results (and vice versa). More-

over, this dependency means that problems in the requirements specification will directly impact quality control.

It is estimated that 85 percent of the defects in developed software originate in the requirements [9, 1]. “Irrespective of the format chosen for representing requirements, the success of a product strongly depends upon the degree to which the desired system is properly described” [8]. Most software requirements are not specified using formal languages, but instead are written as some form of business requirement document. Normally such documents are written using natural languages and pictures. There are several “sins” to avoid when specifying requirements, some of which are listed by Meyer[1] [6]. The first such sin is *noise*, which manifests as information not relevant to the problem, or a repetition of existing information phrased in different ways. Noise may also be the reversal or shading of previously specified statements. Such inconsistencies between requirements make up 13 percent of requirements problems [4]. A second hazard is *silence*, in which important aspects of the problem are simply not mentioned. Omitted requirements account for 29 percent of all requirements errors [4]. *Over-specification* can happen when aspects of the solution are mentioned as part of the problem description. Requirements describe what is to be done but not how they are implemented [3]. *Wishful thinking* is when prose describes a problem to which a realistic solution would be difficult or impossible to find. *Ambiguity* is common when natural languages allow for more than one meaning for a given word or phrase. Often this is problematic when jargon includes terms otherwise familiar to the other party [6]. Prose is also prone to *reader subjectivity* since each person has a unique perspective (based on their cultural background, language, personal experience, etc). *Forward references* mention aspects of a problem not yet mentioned, and cause confusion in larger documents. *Oversized documents* are difficult to understand, use and maintain. *Customer uncertainty* appears when an inability to express specific needs results in an inclusion of vague descriptions. This, in turn, leads to developers making assumptions about “fuzzy” requirements: it has been estimated that incorrect assumptions account for 49 percent of requirements problems [4]. Making requirements understandable to the customer and verifiable by the developer might lead to the creation of *multiple representations* of the same requirements. Preserving more than one document can then lead to maintenance, translation and synchronization problems. Requirements are sometimes lost, especially non-functional requirements, when the use of *tools for requirements capture* only supports a strictly defined format or template. Lastly, requirements documents are often poor when written with *little to no user involvement*, instead being compiled by requirements solicitors, business analysts, domain experts or even developers [7].

This paper examines the suitability of FIT as a format for communicating functional requirements to the developer, and explores whether this format helps mitigate the “sins” listed above. In this context, we define “suitability” as the degree to which the functional requirements are found to be unambiguous, verifiable, consistent, and usable by developers for designing and implementing the system.

---

[1] Meyer’s classification is well known; we have added some additional difficulties to the traditional “seven sins”. Meyer’s classification has been frequently referenced (see pp.232-233 in [8] for example)

There are possibly other desirable properties of requirements. For example, from the customer's perspective, the ease of specifying and understanding the requirements by all stakeholders is important. However our paper focuses only on those properties listed above.

2. Acceptance Testing with FIT

By definition, acceptance tests assess whether a feature is working from the customer's perspective. Acceptance tests are different from unit tests in that the later are modeled and written by the developer, while the former is at least modeled and possibly even written by the customer. Acceptance tests can be specified in many ways, from prose-based user stories to formal languages. Because the execution of acceptance tests is time consuming and costly, it is highly desirable to automate this process. Automating acceptance tests gives an objective answer when functional requirements are fulfilled. At the same time, making the requirements too formal alienates the user, as in the case of definition using formal languages.

FIT was named from the thesaurus entry for "acceptable". The goal of FIT is an acceptance test that an ordinary person can read and write[2]. To this end, FIT tests come in two parts: tests are defined using ordinary *tables* (usually, written by customer representatives, see Fig. 1 and Fig. 2, left side), and later fit *fixtures* are written to execute code using the data from table cells (implemented by the developers, see Fig. 1 and Fig. 2, right side). By abstracting the definition of the test from the logic that runs it, FIT opens up authorship of new tests to anyone who has knowledge of the business domain.

eg.ArithmeticFixture					
x	y	x + y	x - y	x * y	x / y
200	300	500	-100	60000	0
400	20	420	380	8000	20

```
public class  ArithmeticFixture
              extends ColumnFixture {
  public int x;
  public int y;

  public int plus (){ return x + y; }

  public int minus(){ return x - y; }

  public int times(){ return x * y; }

  public int divide(){return x / y; }
}
```

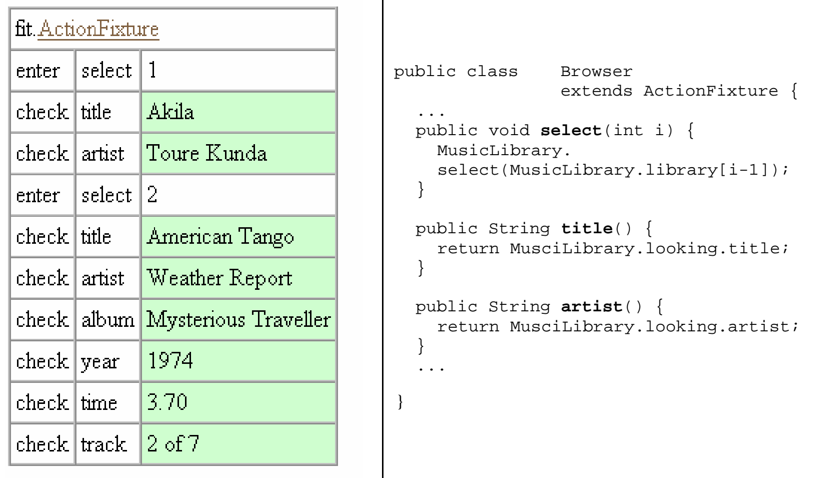
Fig. 1. Sample FIT table and ColumnFixture in Java. Excerpt from fit.c2.com

Deleted: Taken

FIT tables can be created using common business tools, and can be included in any type of document (HTML, MS Word, MS Excel, etc). This idea is taken one step fur-

[2] We leave it to a future experiment to show whether or not FIT tests can be easily read or written by customers. The present experiment focuses on whether developers can use functional requirements for their purposes when specified as FIT acceptance tests.

ther by FitNesse[3], a Web-based collaborative testing and documentation tool designed around FIT. FitNesse provides a very simple way for teams to collaboratively create documents, specify tests, and even run those tests through a Wiki Web site. The FitNesse wiki[4] allows anyone to contribute content to the website without knowledge of HTML or programming technologies.



fit.ActionFixture		
enter	select	1
check	title	Akila
check	artist	Toure Kunda
enter	select	2
check	title	American Tango
check	artist	Weather Report
check	album	Mysterious Traveller
check	year	1974
check	time	3.70
check	track	2 of 7

```
public class    Browser
                extends ActionFixture {
  ...
  public void select(int i) {
    MusicLibrary.
    select(MusicLibrary.library[i-1]);
  }

  public String title() {
    return MusciLibrary.looking.title;
  }

  public String artist() {
    return MusciLibrary.looking.artist;
  }
  ...
}
```

Fig. 2. Simple FIT table and ActionFixture in Java. Excerpt from fit.c2.com

Although acceptance tests are often written based on user requirements, we see that with FIT it is not necessary to create a written requirements document before creating an acceptance test. FIT tests are a tabular representation of customer expectations that can be understood by human beings. All that is needed to write a FIT table is a customer expectation and the ability to precisely and unambiguously write it down. In this way they are very similar to written functional requirements[5]. If the expectations themselves adequately explain the requirements for a feature, can be defined by the customer, and can be read by the developer, there may be some redundancy between the expression of those expectations and the written system requirements. Consequently, it may be possible to eliminate or reduce the size of prose requirements definitions. An added advantage to increased reliance on acceptance tests may be an increase in test coverage, since acceptance testing would both be mandatory and defined early in the project life cycle. To this end an experiment has been designed to evaluate the understandability of FIT acceptance tests for functional requirements specification.

[3] http://www.fitnesse.org and http://fit.c2.com
[4] http://wiki.org/wiki.cgi?WhatIsWiki
[5] http://c2.com/doc/xpu02/workshop.html

3. Instrument

The goal of our experiment was to determine the suitability of using FIT tests as the functional part of a requirements specification. A project was conceived to develop an online document review system (DRS). This system allows users to submit, edit, review and manage professional documents (articles, reports, code, graphics artifacts etc.) called submission objects (so). These features are selectively available to three types of users: Authors, Reviewers and Administrators. More specifically, administrators can create repositories with properties such as: title of the repository, location of the repository, allowed file formats, time intervals, submission categories, review criteria and designated reviewers for each item. Administrators can also create new repositories based on existing ones. Authors have the ability to submit and update multiple documents with data including title, authors, affiliations, category, keywords, abstract, contact information and bios, file format, and access permissions. Reviewers can list submissions assigned to them, and refine these results based on document properties. Individual documents can be reviewed and ranked, with recommendations (accept, accept with changes, reject, etc) and comments. Forms can be submitted incomplete (as drafts) and finished at a later time.

For the present, subjects were required to work on only a partial implementation concentrating on the submission and review tasks (Fig. 3). The only information provided in terms of project requirements was:

1. An outline of the system no more detailed than that given in this section.
2. A subset of functional requirements to be implemented (Fig. 3).
3. A suite of FIT tests (Fig. 4)

Deleted: above

Specification

1. Design a data model (as a DTD or an XML Schema, or, likely, a set of DTDs/XML Schemas) for the artifacts to be used by the DocumentReviewSystem. Concentrate on "Document submission/update" and "Document review" tasks for now.
2. Build XSLT sheet(s) that when applied to an instance of so's repository will produce a subset of so's. As a minimum, queries and three query modes specified in DrsAssignmentOneAcceptanceTests must be supported by your model and XSLT sheets.
3. Create additional FIT tests to completely cover functionality of the queries.

Setup files

drs_master.xml - a sample repository against which the FIT tests were written

DrsAssignmentOneAcceptanceTests.zip - FIT tests, unzip them into FITNESSE_HOME\FitNesseRoot\ directory.

Fig. 3. Assignment specification snapshot[6].

[6] http://mase.cpsc.ucalgary.ca/EB/Wiki.jsp?page=SENG513w04AssignmentOne

DRS Assignment One Acceptance Test Suite

Startswith Author Search

DrsAssignmentOneAcceptanceTests.FindByAuthorUnsorted
DrsAssignmentOneAcceptanceTests.FindByAuthorSortByTitle
DrsAssignmentOneAcceptanceTests.FindByAuthorSortByTitleDescending
DrsAssignmentOneAcceptanceTests.FindByAuthorSortByType
DrsAssignmentOneAcceptanceTests.FindByAuthorSortByDate
DrsAssignmentOneAcceptanceTests.FindByAuthorSortByDateDescending

Contains Author Search

DrsAssignmentOneAcceptanceTests.FindByAuthorContainsUnsorted
DrsAssignmentOneAcceptanceTests.FindByAuthorContainsSortByTitle
DrsAssignmentOneAcceptanceTests.FindByAuthorContainsSortByTitleDescending
DrsAssignmentOneAcceptanceTests.FindByAuthorContainsSortByType
DrsAssignmentOneAcceptanceTests.FindByAuthorContainsSortByDate

Fig. 4. Partial FIT Test Suite. The suite contains test cases and can be executed. For example, the test FindByAuthorUnsorted results in an unsorted list of items matching an author name.

Requirements in the FIT Test Suite of our experiment can be described generally as sorting and filtering tasks for a sample XML repository. Our provided suite initially consisted of 39 test cases and 657 assertions. In addition to developing the code necessary to pass these acceptance tests, participants were required to extend the existing suite to cover any additional sorting or filtering features associated with their model. An example FIT Test finding a document by exact match of author name, with results sorted by title in descending order is shown in Fig. 5.

Participants were given two weeks (unsupervised) to implement these features using XML, XSLT, Java and the Java API for XML Processing (JAXP). A common online experience base[7] was set up and all students could utilize and contribute to this knowledge repository. An iteration planning tool and source code management system were available to all teams if desired.

We hypothesized that:

A) FIT acceptance tests describe a customer requirement such that a developer can implement the feature(s) for that requirement.

B) Developers with no previous FIT experience will quickly be able to learn how to use FIT given the time provided.

C) 100% of developers will create code that passes 100% of customer provided tests.

D) More than 50% of the requirements for which no tests were given will be implemented and tested.

E) 100% of implemented requirements will have corresponding FIT tests.

---

[7] http://mase.cpsc.ucalgary.ca/EB/

fit.ActionFixture		
start	seng513.w04.drs.fixtures.FindActionFixture	
enter	repository	drs_master.xml
enter	querytype	findbyauthor
enter	querymode	isexact
enter	sortorder	title
enter	sortdirection	descending
enter	query	Maurer, Frank
press	find	
enter	select	3
check	author	Read, Kristopher
check	author	Maurer, Frank
check	author	Melnik, Grigori
check	author	Liu, Lawrence
check	title	Advantages of Tablet Usage by Software Development Teams
check	type	tex
check	dateSubmitted	2003-12-25
enter	select	2
check	author	Melnik, Grigori
check	author	Read, Kristopher
check	author	Maurer, Frank
check	title	Distributed Extreme Programming
check	type	rtf
check	dateSubmitted	2003-11-21
enter	select	1
check	author	Maurer, Frank
check	author	Chau, Thomas
check	title	Knowledge Management in Scrum
check	type	pdf
check	dateSubmitted	1999-03-15

Fig. 5. A sample FIT test (after execution).

4. Sampling

Students of computer science programs from the University of Calgary and the Southern Alberta Institute of Technology (SAIT) participated in the experiment. All individuals were knowledgeable about programming and testing, however, no individuals had any advance knowledge of FIT or FitNesse (based on a verbal poll).

Twenty five (25) senior undergraduate University of Calgary students were enrolled in the course *Web-Based Systems*[8], which introduces the concepts and techniques of building Web-based enterprise solutions and includes comprehensive hands-on software development assignments. Seventeen (17) students from the Bachelor of

[8] http://mase.cpsc.ucalgary.ca/seng513/W2004/

Applied Information Systems program were enrolled in a similar course, *Internet Software Techniques*[9], at SAIT. The material from both courses was presented consistently by the same instructor in approximately the same time frame. This experiment spans only the first of six assignments involving the construction of a document review system.

Students were encouraged to work on programming assignments following the principles and the practices of extreme programming, including test-first design, collective code ownership, short iterations, continuous integration, and pair programming.

The University of Calgary teams consisted of 4 to 5 members, and additional help was available twice a week from two teaching assistants. SAIT teams had 3 members[10] each; however they did not have access to additional help outside of classroom lectures. In total, there were 12 teams and a total of 42 students.

5. Observations

Our first hypothesis was that FIT acceptance tests describe a customer requirement such that a developer can implement the feature(s) for that requirement. Our experiment provided strong evidence that customer requirements provided using good acceptance tests can in fact be fulfilled successfully. On average (mean) 82% of customer-provided tests passed in the submitted assignments (SD=35%), and that number increases to 90% if we only consider the 10 teams who actually made attempts to implement the required FIT tests (SD=24%)[11] (Fig. 6). Informal student feedback about the practicality of FIT acceptance tests to define functional requirements also supports our first and second hypotheses. Students generally commented that the FIT tests were an acceptable form of assignment specification[12]. Teams had between 1 and 1.5 weeks to master FIT in addition to implementing the necessary functionality (depending on if they were from SAIT or the University of Calgary).

	University of Calgary						SAIT				
Team	1	2	3	4	5	6	1	2	4	5	6
Customer Tests Pass Ratio	100%	100%	0%	100%	100%	100%	79%	26%	100%	100%	100%

Fig. 6. Customer test statistics by teams.

[9] http://mase.cpsc.ucalgary.ca/apse504/W2004/

[10] SAIT teams had fewer members so that we would have an equal number of teams at each location.

[11] One team's data was removed from analysis because of a lack of participation from team members. One other team (included) delivered code but did not provide FIT fixtures.

[12] It should be noted that an academic assignment is not the same as a real-world requirements specification.

Seventy-three percent (73%) of all groups managed to satisfy 100% of customer requirements. Although this refutes our second hypothesis, our overall statistics are nonetheless encouraging. Those teams who did not manage to satisfy all acceptance tests also fell well below the average (46%) for the number of requirements attempted in their delivered product (Fig. 7).

	University of Calgary						SAIT				
Team	1	2	3	4	5	6	1	2	4	5	6
% of Require-ments Attempted	87%	55%	42%	77%	42%	68%	32%	10%	59%	32%	35%

Fig. 7. Percentage of attempted requirements. An attempt is any code delivered that we evaluate as contributing to the implementation of desired functionality.

Unfortunately, no teams were able to implement and test at least 50% of the additional requirements we had expected. Those requirements defined loosely in prose but given no initial FIT tests were largely neglected both in terms of implementation and test coverage (Fig. 8). This disproves our hypothesis that 100% of implemented requirements would have corresponding FIT tests. Although many teams implemented requirements for which we had provided no customer acceptance tests, on average only 13% of those new features were tested (SD=13%). Those teams who did deliver larger test suites (for example, team 2 returned 403% more tests than we provided) mostly opted to expand existing tests rather than creatively testing their new features.

	Team	*Number New Tests*	*New Test Pass Ratio*	*Number New Assertions*	*New Assertions Pass Ratio*	*% Additional Tests*	*% Additional Assertions*	*% New Features Tested*	*% Attempted Features Tested*
University	1	19	100%	208	100%	49%	32%	32%	67%
	2	157	100%	5225	100%	403%	795%	26%	100%
	3	0	0%	0	0%	0%	0%	0%	0%
	4	116	100%	2218	100%	297%	338%	32%	75%
	5	9	100%	99	100%	23%	15%	16%	100%
	6	41	93%	616	95%	105%	94%	37%	100%
SAIT	1	0	0%	0	0%	0%	0%	0%	80%
	2	0	0%	0	0%	0%	0%	0%	100%
	4	56	100%	1085	100%	144%	165%	11%	66%
	5	0	0%	0	0%	0%	0%	0%	100%
	6	5	100%	64	100%	13%	10%	5%	100%

Fig. 8. Additional features and tests statistics.

Customers do not always consider exceptional cases when designing acceptance tests, and therefore acceptance tests must be evaluated for completeness. Even in our own scenario, all tests specified were positive tests; tests confirmed what the system should do with valid input, but did not explore what the system should do with invalid entries. For example, one test specified in our suite verified the results of a search by file type (.doc, .pdf, etc.). This test was written using lowercase file types, and nowhere was it explicitly indicated that uppercase or capitalized types be permitted (.DOC, .Pdf, etc). As a result, 100% of teams wrote code that was case sensitive, and 100% of tests failed when given uppercase input.

6. Conclusions

Our hypotheses (A and B) that FIT tests describing customer requirements can be easily understood and implemented by a developer with little background on this framework were substantiated by the evidence gathered in this experiment. Considering the short period of time allotted, we can conclude from the high rate of teams who delivered FIT tests (90%) that the learning curve for reading and implementing FIT tests is not prohibitively steep, even for relatively inexperienced developers.

Conversely, our hypotheses that 100% of participants would create code that passed 100% of customer provided tests (C), that more than 50% of the requirements for which no tests were given would be tested (D), and that 100% of implemented requirements would have corresponding FIT tests (E) were not supported. In our opinion, the fact that more SAIT teams failed to deliver 100% of customer tests can be attributed to the slightly shorter time frame and the lack of practical guidance from TA's. Given more time and advice we believe that a higher rate of customer satisfaction can be achieved. The lack of tests for new features added by teams may, in our opinion, be accredited to the time limitations placed on students, the lack of motivation to deliver additional tests, and the lower emphasis given to testing in the past academic experiences of these students[13]. At the very least, our observation that feature areas with fewer provided FIT tests were more likely to be incomplete supports the idea that FIT format functional requirements are of some benefit.

The fact that a well defined test suite was provided by the customer up front may have instilled a false sense of security in terms of test coverage. The moment the provided test suite passed, it is possible that students assumed the assignment was complete. This may be extrapolated to industry projects: development teams could be prone to assuming their code is well tested if it passes all customer tests. It should be noted that writing FIT tests is simplified but not simple; to write a comprehensive suite of tests, some knowledge and experience in both testing and software engineering is desirable (for example, a QA engineer could work closely with the customer). It is vital that supplementary testing be performed, both through unit testing and additional acceptance testing. The role of quality assurance specialists will be significant even on teams with strong customer and developer testing participation. Often dia-

[13] Despite the fact that the importance of testing was repeatedly emphasized, students are not accustomed to writing test code. Students were aware that the majority of marks were not being assigned based on new tests.

bolical thinking and knowledge of specific testing techniques such as equivalence partitioning and boundary value analysis are required to design a comprehensive test suite.

From the outcome of our five hypotheses, along with our own observations and feedback from the subjects, we can suggest how FIT acceptance tests perform as a specification of functional requirements in relation to the criteria stated in our introduction. We believe that *noise* is greatly reduced when using FIT tests to represent requirements. Irrelevant information is more difficult to include in well structured tables than in prose documents. Also, tests which shade or contradict previous tests are easily uncovered at the time of execution (although there is no automatic process to do so). Acceptance tests can be used as regression tests after they have passed in order to prevent problems associated with possible noise. We discovered that *silence* is not well addressed by the FIT framework, and may even become a more serious problem. This was well demonstrated by the failure of our teams to test at least 50% of the requirements for which no tests were given. Our example of case-sensitive document types also clearly demonstrates how a lack of explicit tests can lead to assumptions and a lack of clarifications. Prose documents may be obviously vague, and by this obviousness incite additional communication. *Over-specification* is not a problem since FIT tests do not allow any room for embedded solutions in the tests themselves. FIT tables are only representations of customer expectations, and the fixtures become the agents of the solutions. Although it can be argued that specifying an ActionFixture describes a sequence of actions (and therefore a solution), when writing FIT tables these actions should be based on business operations and not code-level events. *Wishful thinking* is largely eliminated by FIT, since defining tests requires that the customer think about the problem and make very specific decisions about expectations.

	Noise	Silence	Over-specification	Wishful Thinking	Ambiguity	Forward References	Oversized Documents	Reader Subjectivity	Customer Uncertainty	Multiple Representations	Use of Tools	User Involvement
FIT Effectively Addresses	✓		✓	✓	✓	✓	✓	✓	✓	✓		✓

Fig. 9. Evaluation of FIT for requirements specification. Check marks indicate that FIT effectively addresses the issue (although it could be only partial).

Ambiguity may still be a problem when defining requirements using FIT tests if keywords or fields are defined in multiple places or if these identifiers are open to multiple interpretations. However, FIT diminishes ambiguity simply because it uses fewer words to define each requirement. *Forward references* and *oversized documents* may still be an issue if large numbers of tests are present and not organized into meaningful test suites. In our experiment, the majority of groups categorized their own tests without any instruction to do so. *Reader subjectivity* is greatly reduced by FIT tests. Tables are specified using a format defined by the framework (*ActionFix-*

ture, *ColumFixture*, etc). As long as tests return their expected results when executed, the developer or customer knows that the corresponding requirement was correctly interpreted regardless of the terminology used. *Customer uncertainty* may manifest as the previously mentioned problem of silence, but it is impossible for a defined FIT test not to have a certain outcome. FIT tests are executable, verifiable and easily readable by the customer and developer, and therefore there is no need for *multiple representations* of requirements. All necessary representations have effectively merged into a suite of tables. Requirements gathering *tools* can be problematic when they limit the types of requirements that can be captured. FIT is no exception; it can be difficult to write some requirements as FIT tests, and it is often necessary to extend the existing set of fixtures, or to utilize prose for defining non-functional requirements and making clarifications. However, FIT tests can be embedded in prose documents or defined through a collaborative wiki such as FitNesse, and this may help overcome the limitations of FIT tables.

In addressing the characteristics of suitability (as defined in Introduction), our findings demonstrate that FIT tests as functional requirements specifications are in fact unambiguous, verifiable, and usable (from the developer's perspective). However, insufficient evidence was gathered to infer consistency between FIT tests.

Although our results did not match all of our expectations, valuable lessons were learned from the data gathered. When requirements are specified as tests, there is still no guarantee that the requirements will be completed on-time and on-budget. Time constraints, unexpected problems, lack of motivation and poor planning can still result in only some requirements being delivered. As with any type of requirements elicitation, it is vital that the customer is closely involved in the process. FIT tests can be executed by the customer or in front of the customer, and customers can quickly evaluate project progress based on a green (pass) or red (fail) condition. In conclusion, our study provides only initial evidence of the suitability of FIT tests for specifying functional requirements. This evidence directly supports the understandability of this type of functional requirements specification by developers. There are both advantages and disadvantages to adopting FIT for this purpose, and the best solution is probably some combination of both prose-based and FIT-based specifications.

8. Validity

This paper provides only initial evidence supporting the use of FIT tests to communicate functional requirements to developers. There are several possible threats to the validity of this experiment that should be reduced through future experiments. One such threat is the limitation of our experiment to a purely academic environment. Although we spanned two different academic institutions, industry participants would be more relevant. Another threat is our small sample size, which can be increased through repeated experiments in future semesters. Moreover, all of the FIT tests provided in this experiment were written by expert researchers, which would not be the case in an industrial setting. Although this was an academic assignment, it was not conducted in a controlled environment. Students worked in teams on their own time without proper invigilation.

9. Future work

This experiment is the first in a series of six FIT-related experiments planned for the next eight months. Given more time and advice, we believe, that a higher rate of customer satisfaction can be achieved. This will be investigated using the same teams as the experiment continues this semester. All insights gained from the analysis of our observations will be verified and validated with additional trials on the current teams as well as new trials with a new sampling of subjects.

An upcoming experiment will have the subjects refactor current tests to adapt to new and changing requirements. In addition, there will be increased emphasis on more complete, negative testing. In a third experiment, subjects will be asked to specify a suite of FIT requirements for a remote team at a different institution.

An experiment with industry practitioners is part of our ongoing research. It will test the understandability of functional requirements specified as FIT tables. We invite any interested party to contact the authors for further discussion. FIT training, on-site or off-site, will be provided free of charge.

Acknowledgements

We would like to thank all participants from the University of Calgary and SAIT who participated in this study and provided us with their valuable feedback. This ongoing research is partially sponsored by NSERC and iCore.

References

1. Ben-Menachem, M., Marliss, G. *Software Quality: Producing Practical, Consistent Software*, International Thomson Publishing, London, UK, 1997.
2. CenterLine Software, Inc. A Survey of 240 Fortune 1,000 companies in North America and Europe, Cambridge, MA, 1996. Online http://www.computerworld.com/news/1997/story/0,11280,17522,00.html. Last accessed February 29, 2004.
3. Davis, A. *Software Requirements Revision Objects, Functions, & States*, Prentice Hall PTR, Englewood Cliffs, NJ, 1994.
4. Hooks, I., Farry, K. *Customer-Centered Products: Creating Successful Products Through Smart Requirements Management*. American Management Association, New York, NY, 2001.
5. Jones, C. Patterns of *Software Systems Failure and Success*. International Thompson Computer Press, Boston, MA, 1996.
6. Meyer, B. On Formalism in Specifications. *IEEE Software*, 2(1):6–26, 1985.
7. *The CHAOS Chronicles*. The Standish Group International, West Yarmouth, MA. Online http://www1.standishgroup.com//chaos/intro2.php. Last accessed January 20, 2004.
8. Van Vliet, H. *Software Engineering: Principles and Practice*, 2/e, John Wiley & Sons, Chichester, UK, 2000.
9. Young, R. *Effective Requirements Practices,* Addison-Wesley, Boston, MA, 2001.