IEEE Software
www.computer.org/software

Tests and Requirements, Requirements and Tests: A Möbius Strip

Robert C. Martin, Object Mentor Inc.
Grigori Melnik, University of Calgary and Microsoft

Vol. 25, No. 1
January/February 2008

Tests and Requirements, Requirements and Tests: A Möbius Strip

Robert C. Martin, *Object Mentor Inc.*

Grigori Melnik, *University of Calgary and Microsoft*

Writing acceptance tests early is a requirements-engineering technique that can save time and money and help businesses better respond to change.

Surprisingly, to some people, one of the most effective ways of testing requirements is with test cases very much like those for testing the completed system. —Donald C. Gause and Gerald M. Weinberg

When Donald Gause and Gerald Weinberg wrote this statement,[1] they were asserting that writing tests is an effective way to test requirements' completeness and accuracy. They also suggest writing these tests when gathering, analyzing, and verifying requirements—long before those requirements are coded. They go on to say, "We can use the black box concept during requirements definition because the design solution is, at this stage, a truly black box. What could be more opaque than a box that does not yet exist?"[1] Clearly, they value early test cases as a requirements-analysis technique.

Testing expert Dorothy Graham agrees. She recommends performing test-design activities "as soon as there is something to design tests against—usually during the requirements analysis."[2] According to Graham, designing tests highlights what users really want the system to do. If software professionals design tests early and with users' involvement, they can discover problems before building them into the system.

The testing community has also promoted early writing of acceptance tests,[3] but this remains at odds with much practice. Most development organizations don't write acceptance tests. The first tests they write are often manual scripts, written after the application starts executing. They base these regression tests on the executing system's behavior as opposed to the original requirements. Instead of manual tests, some organizations use record-and-playback tools to automate their tests. These tools record the tester's strategic decisions by watching the tester operate the current system and remembering how the system responds. Later, the tool can repeat the sequence and report any deviation. Although record-and-playback tests can be valuable,[4] they're written far later than Gause, Weinberg, and Graham suggest, and their connection to the original requirements is indirect at best.

We argue for early writing of acceptance tests as a requirements-engineering technique. We believe that concrete requirements blend with acceptance tests in much the same way as the two sides of a strip of paper become one side in a Möbius strip (see figure 1). In other words, requirements and tests become indistinguishable, so you can specify system behavior by writing tests and then verify that behavior by executing the tests.

0740-7459/08/$25.00 © 2008 IEEE

Test precision

Record-and-playback might occur late, but is it really appropriate to write tests as part of requirements definition? Yes, because writing tests is a way of being relentlessly precise. In essence, a test is a question that has a concrete answer, whereas a requirement is generally more abstract. Consider the following requirement: "The system shall acknowledge the number of tickets purchased." As figure 2 shows, a test could more concretely state this requirement.

This simple table is written in the FIT style (Framework for Integrated Testing, http://fit.c2.com). It shows three different ticket purchases, along with the acknowledgment that the system should output. The table provides a level of detail that's hard to put into the less formal language of "shall" statements. We see not only the precise message that acknowledges the purchase but also subtleties in grammar and the length of the play names.

David Parnas recognized the value of tabular specification as early as 1977 when he was working on the A-7 project for the US Naval Research Lab. In 1996, he wrote,

> *Tabular notations are of great help in situations like this. One first determines the structure of the table, making sure that the headers cover all possible cases, then turns one's attention to completing the individual entries in the table. The task may extend over weeks or months; the use of the tabular format helps to make sure that no cases get forgotten.*[5]

Indeed, as we examine figure 2, we can think of many additional headers, such as "Where will the play be held?" or "What time is the showing?" The table is far more suggestive than the original "shall" statement.

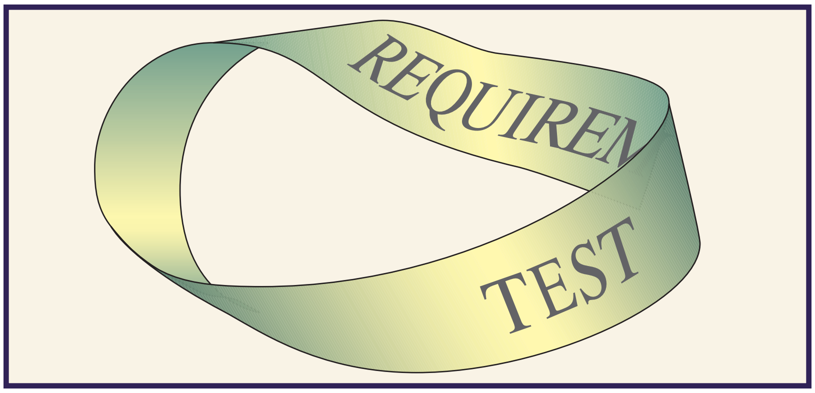


Figure 1. Writing requirements and testing are interrelated, much like the two sides of a Möbius strip.

Purchase tickets and check acknowledgment		
Play	Quantity	Acknowledgment?
Phantom	2	You have purchased 2 tickets to Phantom of the Opera.
Wizard	1	You have purchased 1 ticket to The Wizard of Oz.
Cats	1	The play 'Cats' is not currently playing in a theater.

Figure 2. A FIT (Framework for Integrated Testing) table for purchasing tickets.

The equivalence hypothesis

If you look closely at figure 2, you'll see that it's difficult to tell whether it's a requirement or a test. Clearly, it specifies the system behavior and could be viewed as a kind of Parnas-table requirement. On the other hand, you could imagine a tester using this table to verify system operation. Indeed, you might even imagine a simple software engine that reads the table, operates the system, and turns a light red or green, depending on whether the system's behavior matches the table's specification.

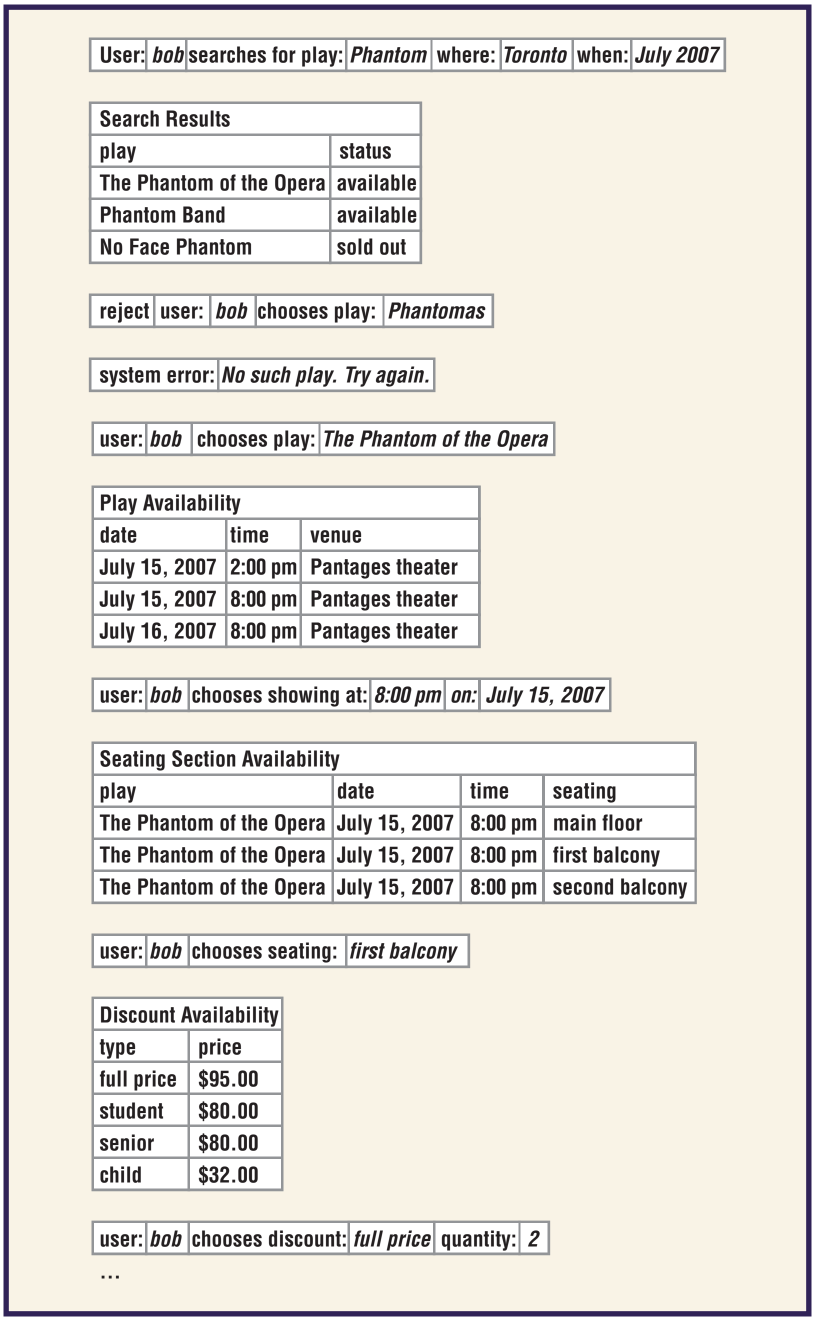
This fuzziness between requirements and tests suggests an idea that we call the equivalence hypothesis: "As formality increases, tests and requirements become indistinguishable. At the limit, tests and requirements are equivalent." In a practical sense, this hypothesis describes how many software development teams behave—typically, the passing of acceptance tests, as opposed to an examination of the requirements, is the final criterion for shipping a system.

Any reasonable hypothesis should be falsifiable; otherwise, it's little more than a feel-good statement. To falsify our equivalence hypothesis, you would have to construct a requirement and a test that diverge as the formality increases. In other words, as the system description becomes more precise, the requirement becomes less testable. This, it seems to us, is virtually reductio ad absurdum.

A hypothesis should also be predictive. Our equivalence hypothesis predicts that an informal system description won't be testable. Or, rather, you can't write unambiguous tests against an imprecise requirement. Consider the following requirement, written at a higher abstraction level: "The system shall acknowledge the number of tickets purchased." How should the system present this acknowledgement? As a green light? A verbal response? By mail? Also, when should it give it—immediately, or six months from now?

Executable test-based specifications

If our equivalence hypothesis is true, we should



User:	*bob*	searches for play:	*Phantom*	where:	*Toronto*	when:	*July 2007*

Search Results	
play	status
The Phantom of the Opera	available
Phantom Band	available
No Face Phantom	sold out

reject	user:	*bob*	chooses play:	*Phantomas*

system error:	*No such play. Try again.*

user:	*bob*	chooses play:	*The Phantom of the Opera*

Play Availability		
date	time	venue
July 15, 2007	2:00 pm	Pantages theater
July 15, 2007	8:00 pm	Pantages theater
July 16, 2007	8:00 pm	Pantages theater

user:	*bob*	chooses showing at:	*8:00 pm*	*on:*	*July 15, 2007*

Seating Section Availability			
play	date	time	seating
The Phantom of the Opera	July 15, 2007	8:00 pm	main floor
The Phantom of the Opera	July 15, 2007	8:00 pm	first balcony
The Phantom of the Opera	July 15, 2007	8:00 pm	second balcony

user:	*bob*	chooses seating:	*first balcony*

Discount Availability	
type	price
full price	$95.00
student	$80.00
senior	$80.00
child	$32.00

user:	*bob*	chooses discount:	*full price*	quantity:	*2*

...

Figure 3. A specification scenario for a ticket-sales system.

be able to specify system behavior using tests and then verify that the system behaves as specified by executing those tests. The open source tool FitNesse (http://fitnesse.org) provides a collaboration platform to flesh out requirements written as tests in the FIT style.[6] FitNesse also contains a test-running engine that can apply those tests to the system being specified. Here, we use this tool to specify a simple ticket-sales system.

As you can see from the specification episode in figure 3, we use both declarative and procedural tables. The declarative tables have column headers and rows that show a query result. The procedural tables specify user interactions with the system, including both happy and sad paths.

According to Ian Alexander and Neil Maiden, these "sequences of events in time are at the heart of our ability to construct meaning."[7] Furthermore, other researchers have recognized that "tables, unlike natural language, encode temporal sequences unambiguously."[8] Taken as a whole, we see an entire workflow, including user actions and system responses. The result looks close enough to an ordinary story that people can understand it without inordinate effort. What's more, specifiers can write plain text stories and then maneuver those stories into tables.

This ease of reading and authoring is critical for requirements engineering, because stakeholders who aren't technologically savvy often perform the specification. Furthermore, requirements are more credible and motivating if stakeholders write them—or at least help write them.

The notion that stakeholders can write tests doesn't imply that we should suddenly discard our years of testing experience in favor of customer-written tests. On the contrary, although writing tests in a language that customers can read—and even write—is valuable, testing professionals still must apply techniques such as boundary value analysis, path analysis, and state transition analysis.

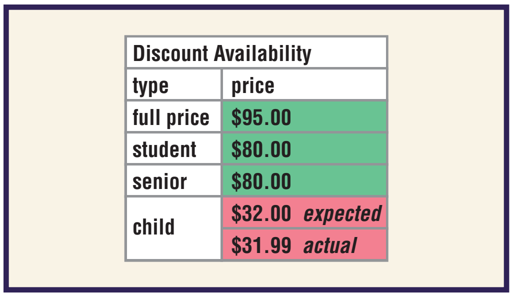
Benefits of narratives and sufficient formality

Notice how close the test in figure 3 is to a normal narrative or use case. If desired, a simple postprocessor or browser option could make the test look even more like a colloquial scenario. This closeness to human language means that stakeholders and developers can easily read these requirements and infer the same meaning from them. (A series of quasi-experiments and case studies support this.[9])

And yet, for all their readability, these tests have sufficient formality to allow an automatic engine to run them and validate that the system behaves as specified. In short, the human-readable requirements are also executable acceptance tests. For example, if our ticket-sales application miscalculates the price of a child's ticket, FitNesse will display the test results in figure 4.

The red and green highlighting makes evaluating these test results easy. As Cem Kaner points out, ease of evaluation is "valuable for all tests, but is especially important for scenarios because they are complex."[10] After all, we don't want bugs to be exposed by a test but not recognized by the engineering team.

Another important benefit of concrete examples



Discount Availability	
type	price
full price	$95.00
student	$80.00
senior	$80.00
child	$32.00 *expected*
	$31.99 *actual*

Figure 4. The test results FitNesse displays in correct cases (full, senior, and student prices) and in one erroneous case (miscalculation of a child's ticket price).

and FIT-style specifications is that they help developers, managers, testers, and stakeholders avoid misunderstandings. They help the parties agree on business needs and terminology through the creation and enhancement of a ubiquitous language—that is, a well-documented (through tests) shared language that can express the necessary domain information as a common medium of communication.[11]

Automated spec ≠ autogenerated spec

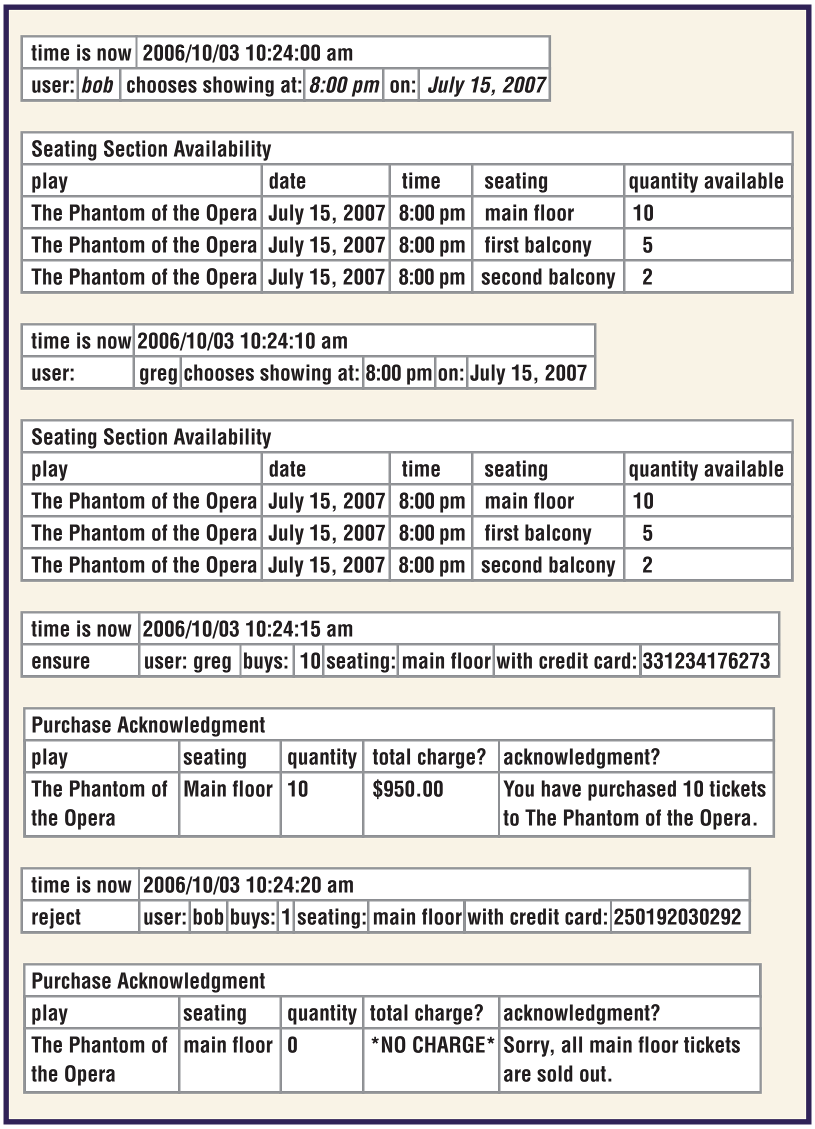
Writing requirements as tests in the FIT style shouldn't be confused with some earlier approaches that autogenerated test scripts from requirement specifications, finite-state machines, activity diagrams, and so forth. These approaches weren't very successful in practice. According to Klaus Weidenhaupt and his colleagues, "the main problem was that the scenarios developed during requirements engineering and system design were out of date at the time the system was going to be tested."[12]

Also, don't confuse FIT-style requirements tests with "operational specifications" that support formal reasoning (such as Gist, Statemate, or PAISley). These specifications are powerful, but they're quite cryptic for an ordinary business person.

When using the FIT style, the specification itself is a test suite. The requirements and tests evolve with the system. Indeed, in an environment where continuous integration and rigorous testing are practiced,[13] a FIT-style requirements document would never be out of sync with the application itself. This is because any disagreement between the requirements and the code would cause the build to fail!

A more complex scenario

Software is hard. Most tools break when you want to do something a bit more complicated than their designers expected—that is, when you most need them. Can we use the FIT specification style in situations that are more complex than simple interaction scenarios?



time is now	2006/10/03 10:24:00 am					
user:	*bob*	chooses showing at:	*8:00 pm*	on:	*July 15, 2007*	

Seating Section Availability				
play	date	time	seating	quantity available
The Phantom of the Opera	July 15, 2007	8:00 pm	main floor	10
The Phantom of the Opera	July 15, 2007	8:00 pm	first balcony	5
The Phantom of the Opera	July 15, 2007	8:00 pm	second balcony	2

time is now	2006/10/03 10:24:10 am				
user:	greg	chooses showing at:	8:00 pm	on:	July 15, 2007

Seating Section Availability				
play	date	time	seating	quantity available
The Phantom of the Opera	July 15, 2007	8:00 pm	main floor	10
The Phantom of the Opera	July 15, 2007	8:00 pm	first balcony	5
The Phantom of the Opera	July 15, 2007	8:00 pm	second balcony	2

time is now	2006/10/03 10:24:15 am							
ensure	user: greg	buys:	10	seating:	main floor	with credit card:	331234176273	

Purchase Acknowledgment				
play	seating	quantity	total charge?	acknowledgment?
The Phantom of the Opera	Main floor	10	$950.00	You have purchased 10 tickets to The Phantom of the Opera.

time is now	2006/10/03 10:24:20 am							
reject	user:	bob	buys:	1	seating:	main floor	with credit card:	250192030292

Purchase Acknowledgment				
play	seating	quantity	total charge?	acknowledgment?
The Phantom of the Opera	main floor	0	*NO CHARGE*	Sorry, all main floor tickets are sold out.

Figure 5. The specification of a typical concurrency issue.

Consider the specification of a typical concurrency issue in figure 5. When we view this specification as a message-sequence chart (see figure 6), we see that this is a classic race condition. Bob took his time selecting his seat, while Greg jumped in and got all the remaining main floor tickets.

Notice how these tests treat time as something that can be controlled. Indeed, time is just another system input that impacts the system state. If, as figure 5 shows, our tests can specify the flow of time, then we can describe concurrency issues and specify how the system with multiple synchronous stimuli and responses should behave.

Specifying and testing performance is conceptually no more difficult than the concurrency test we presented. The amount of time an operation takes is merely another system output, which we can specify and test just like any other output. It wouldn't be difficult to make the test shown in figure 7 execute.

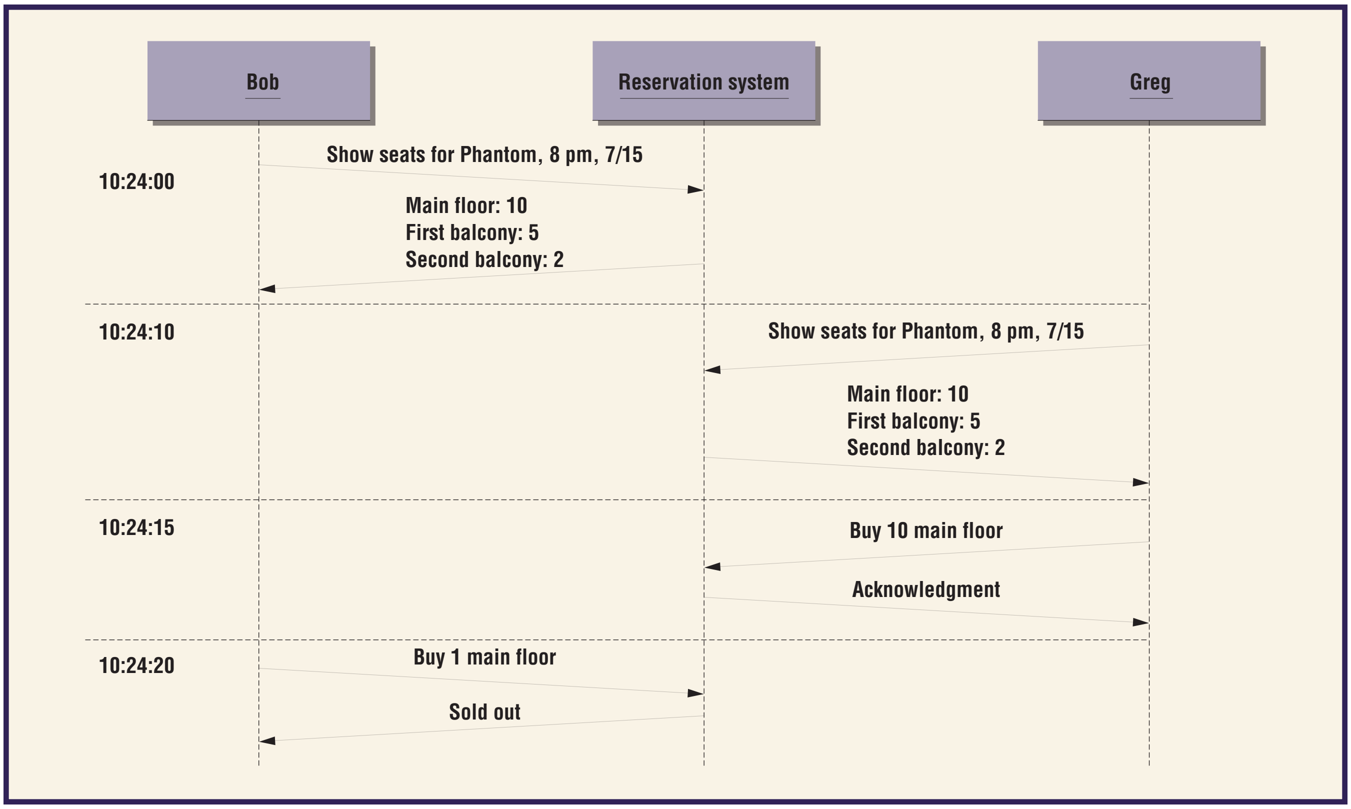
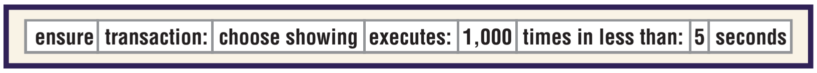


Figure 6. The specification from figure 5, shown as a message-sequence chart.



ensure	transaction:	choose showing	executes:	1,000	times in less than:	5	seconds

Figure 7. A sample test that would be easy to execute.

Clearly, getting these specifications to execute as tests requires some behind-the-scenes magic. What's remarkable is the comparatively tiny amount of effort required to cast that magic spell. The glue code behind the scenes is small, tightly encapsulated, highly reusable, and very easy to write.

Potential business impact

If our equivalence hypothesis is true, and software professionals write their requirements in the form of acceptance tests, this could cut a lot of time and money from the project's test planning phase. The concrete nature of the test-based specifications could reduce the number of pointless features and code, making the project more agile. Furthermore, the development team would be able to handle requirements changes more adequately and efficiently.

In this article, we purposely avoided describing the detailed syntax of FIT to demonstrate that knowledge of that syntax isn't required to read and understand the tests as requirements. This could lead you to believe that there is no syntax and that the tests are simply ad hoc conversions of narratives to tables. In fact, the syntax is sufficiently formal for a computer program to interpret and execute unambiguously.

Requirements written in the FIT style are also tests. They form a Möbius strip that appears to have two sides but, on careful inspection, has only one. The result is that the requirements become tangible. There can be no ambiguity about a requirement if that requirement can turn a light red or green.

References

1. D. Gause and G. Weinberg, *Exploring Requirements*, Dorset House, 1989, p. 249.
2. D. Graham, "Requirements and Testing: Seven Missing-Link Myths," *IEEE Software*, vol. 19, no. 5, 2002, pp. 15–17.
3. B. Hetzel, *The Complete Guide to Software Testing*, QED Information Sciences, 1983.
4. G. Meszaros, "Agile Regression Testing Using Record & Playback," *Companion of the 18th Ann. ACM SIGPLAN Conf. Object-Oriented Programming, Systems, Languages, and Applications* (OOPSLA 03), ACM Press, 2003, pp. 353–360.
5. R. Janicki, D. Parnas, and J. Zucker, *Tabular Representations in Relational Documents*, Communications Research Laboratory, McMaster University, 1995, p. 5.
6. R. Mugridge and W. Cunningham, *Fit for Developing Software*, Prentice Hall, 2005.
7. I. Alexander and N. Maiden, *Scenarios, Stories, Use*

Cases through the Systems Development Life-Cycle, John Wiley & Sons, 2004, p. 5.
8. C. Potts, K. Takahashi, and A. Antón, "Inquiry-Based Requirements Analysis," *IEEE Software*, vol. 11, no. 2, 1994, pp. 21–32.
9. G. Melnik, F. Maurer, and M. Chiasson, "Executable Acceptance Tests for Communicating Business Requirements: Customer Perspective," *Proc. Agile 2006 Conf.*, IEEE CS Press, 2006, pp. 35–46.
10. C. Kaner, "On Scenario Testing," *Software Testing and Quality Eng. Magazine*, Sept./Oct. 2003, pp. 16–22.
11. E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*, Addison-Wesley, 2004, p. 376.
12. K. Weidenhaupt et al., "Scenarios in System Development: Current Practice," *IEEE Software*, vol. 15, no. 2, 1998, pp. 34–45.
13. M. Fowler, "Continuous Integration," www.martinfowler.com/articles/continuousIntegration.html, May 2006.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.

About the Authors

Robert C. Martin (Uncle Bob) is founder and president of Object Mentor. His research interests are agile programming, extreme programming, UML, object-oriented programming, and C++ programming. He has authored many books, including *Designing Object Oriented C++ Applications Using the Booch Method* (Prentice Hall, 1995), *Agile Software Development: Principles, Patterns, and Practices* (Prentice Hall, 2002), and *UML for Java Programmers* (Prentice Hall, 2003). He served three years as the editor in chief of the *C++ Report*, and he served as a founder and first chairman of the Agile Alliance. Contact him at unclebob@objectmentor.com.

Grigori Melnik is a senior product planner in the patterns and practices group at Microsoft. Prior to that (and when this article was written), he was a researcher and faculty member at the University of Calgary. His research areas include agile methods, empirical software engineering, executable acceptance-test-driven development, and domain-driven design. He served as a guest editor of the *IEEE Software* special issue on Test-Driven Development and is program chair of the Agile 2008 World Conference. Contact him at grigori.melnik@microsoft.com.

ADVERTISER INDEX JANUARY/FEBRUARY 2008

Advertiser/Product	Page Number
Classified Advertising	**49**
East Carolina University	**9**
ESRI	**12**
Gannon University	**7**
John Wiley & Sons, Inc.	**Cover 2**
SD West 2008	**Cover 3**
Seapine Software, Inc.	**Cover 4**

*Boldface denotes advertisements in this issue.

Advertising Personnel

Marion Delaney
IEEE Media,
Advertising Director
Phone: +1 415 863 4717
Email: md.ieeemedia@ieee.org

Marian Anderson
Advertising Coordinator
Phone: +1 714 821 8380
Fax: +1 714 821 4010
Email: manderson@computer.org

Sandy Brown
IEEE Computer Society,
Business Development Manager
Phone: +1 714 821 8380
Fax: +1 714 821 4010
Email: sb.ieeemedia@ieee.org

Advertising Sales Representatives

Mid Atlantic (product/recruitment)
Dawn Becker
Phone: +1 732 772 0160
Fax: +1 732 772 0164
Email: db.ieeemedia@ieee.org

New England (product)
Jody Estabrook
Phone: +1 978 244 0192
Fax: +1 978 244 0103
Email: je.ieeemedia@ieee.org

New England (recruitment)
John Restchack
Phone: +1 212 419 7578
Fax: +1 212 419 7589
Email: j.restchack@ieee.org

Connecticut (product)
Stan Greenfield
Phone: +1 203 938 2418
Fax: +1 203 938 3211
Email: greenco@optonline.net

Southwest (product)
Steve Loerch
Phone: +1 847 498 4520
Fax: +1 847 498 5911
Email: steve@didierandbroderick.com

Northwest (product)
Lori Kehoe
Phone: +1 650 458 3051
Fax: +1 650 458 3052
Email: l.kehoe@ieee.org

Southern CA (product)
Marshall Rubin
Phone: +1 818 888 2407
Fax: +1 818 888 4907
Email: mr.ieeemedia@ieee.org

Northwest/Southern CA (recruitment)
Tim Matteson
Phone: +1 310 836 4064
Fax: +1 310 836 4067
Email: tm.ieeemedia@ieee.org

Midwest (product)
Dave Jones
Phone: +1 708 442 5633
Fax: +1 708 442 7620
Email: dj.ieeemedia@ieee.org
Will Hamilton
Phone: +1 269 381 2156
Fax: +1 269 381 2556
Email: wh.ieeemedia@ieee.org
Joe DiNardo
Phone: +1 440 248 2456
Fax: +1 440 248 2594
Email: jd.ieeemedia@ieee.org

Southeast (recruitment)
Thomas M. Flynn
Phone: +1 770 645 2944
Fax: +1 770 993 4423
Email: flynntom@mindspring.com

Midwest/Southwest (recruitment)
Darcy Giovingo
Phone: +1 847 498-4520
Fax: +1 847 498-5911
Email: dg.ieeemedia@ieee.org

Southeast (product)
Bill Holland
Phone: +1 770 435 6549
Fax: +1 770 435 0243
Email: hollandwfh@yahoo.com

Japan (recruitment)
Tim Matteson
Phone: +1 310 836 4064
Fax: +1 310 836 4067
Email: tm.ieeemedia@ieee.org

Europe (product)
Hilary Turnbull
Phone: +44 1875 825700
Fax: +44 1875 825701
Email: impress@impressmedia.com