SYSTEMS DEVELOPMENT MANAGEMENT

SIZING SOFTWARE WITH TESTABLE REQUIREMENTS

Peter B. Wilson

INSIDE

Software Sizing Measures; Definition of Requirements; Testable Requirements; Testable Requirements as Metric; Testable Requirements and Test Cases; Application of Testable Requirements

According to a survey conducted by Mosaic, Inc., and the Quality Assurance Institute, only 23 percent of project managers measure the progress, effectiveness, and productivity of their software development efforts.[1] Since one cannot effectively manage what one cannot measure, why do so few software developers measure? Why, when the costs to develop, maintain, and fix software are so great, do software developers ignore this important activity? One important reason is that current software sizing measures are not flexible enough to meet the needs of today's software developers.

The purpose of this article is to describe a software sizing measure — testable requirements — first proposed by this author.[2] Testable requirements implies a new software measurement paradigm. As such, it has some very interesting attributes and applications.

SOFTWARE SIZING MEASURES

Sizing measures are used to normalize other measures so that valid comparisons can be made across (or within) systems. Without a sizing measure, productivity statistics cannot be computed. For example, barrels of oil and square feet are used by the oil and the real estate industries, respectively, to size their products.

A software sizing measure is fundamental to any software measurement program. While estimating cost and schedule is probably the most

PAYOFF IDEA

The two commonly used metrics for sizing software are lines of code and function points. However, these measures are not flexible enough to meet today's needs. Testable requirements offer a new paradigm for measuring the size of a system. Applications include measuring system scope and earned value, measuring changes in system size, and measuring the thoroughness of test cases.

Auerbach Publications

common use of a sizing measure, there are many other potentially valuable applications, including progress measurement, earned value, risk identification, and change management.

There are only two software sizing measures widely used today: lines of code (LOC) and function points (FP). And, although each is a sizing measure, the two actually measure different things and have very different characteristics.

LOC is a measure of the size of the system that is built. It is highly dependent on the technology used to build the system, the system design, and how the programs are coded. There are many well-documented problems and issues with LOC.[3,4] In fact, Capers Jones has stated[5] that anyone using LOC is "committing professional malpractice." Despite these problems, LOC is still frequently used by very reputable and professional organizations.

In contrast to LOC, function points (FP) is a measure of delivered functionality that is relatively independent of the technology used to develop the system. While FP addresses many of the problems inherent in LOC, and has developed a loyal following, it has its own set of issues.[2,4]

Because LOC and FP have been the only widely accepted ways to size a system, a software developer's measurement choices have been very limited. Most have opted not to measure at all.

REQUIREMENTS

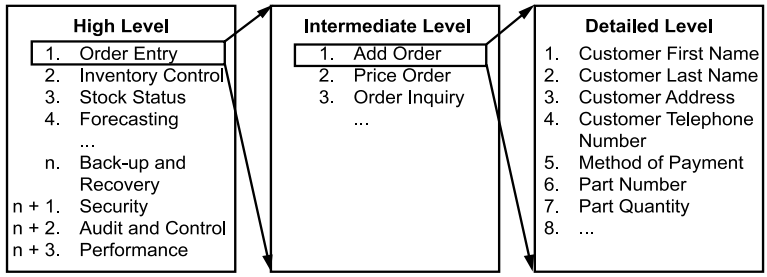
For this article, a "requirement" is defined as a condition or capability that is necessary for a system to meet its objectives. This definition is purposely broad. Many software developers mean the "user requirements" when they use the term "requirements." The above definition purposely includes the traditional user requirements as well as other categories such as technical design and operational requirements. The *IEEE Standard Glossary of Software Engineering Terminology* includes definitions of six different types of requirements: functional, design, implementation, interface, performance, and physical. Thus, it is important to size the user requirements and any other meaningful category of requirement.

It should be emphasized that frequently there are important differences between various types of requirements. Many problems occur, for example, when developers do not properly distinguish between user requirements and technical design requirements. For purposes of sizing, however, there is value in being able to size each type of requirement with the same measure. The above definition of requirements thus includes all types.

TESTABLE REQUIREMENTS

Most methodologies (e.g., requirements, design, etc.) specify functional-

EXHIBIT 1 — Requirements Decomposition

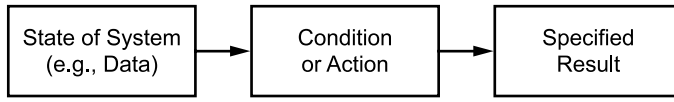


ity in increasing levels of detail. This is illustrated in Exhibit 1. The highest level is generally a major user requirement (e.g., order entry or claims processing). Once identified, high-level requirements can be decomposed into the next — "intermediate" — level of user requirements (e.g., reports or screens/windows). This level, and succeeding levels, can be further decomposed. Decomposition stops (usually after three or four levels) when the requirement is precisely defined; that is, when there is no ambiguity in the specification of the requirement.

The criterion that the requirement be precisely defined and unambiguous will be met only if the requirement is testable. A requirement is testable if someone is able to write one or more test cases that would validate whether the requirement has or has not been implemented correctly. This is the source of the term "testable requirement." For most requirements, a testable requirement can be described in terms of the (see Exhibit 2):

- state of the system
- data elements that are inputs (e.g., customer number or product number)
- condition or action that invokes the requirement (e.g., upon entry from the user, when the order is validated, or when the check is processed)
- expected result is described in terms of data elements (e.g., the customer number must be eight-digit numeric, or the product quantity must be greater than zero)

EXHIBIT 2 — Testable Requirements



TESTABLE REQUIREMENTS AS A SIZING MEASURE

The concept of a testable requirement has been used for years as a test of the quality and detail of system specifications. The main assertion of this article is that the number of testable requirements is a measure of the size of the system. This in turn has many practical applications for the measurement of the software development process. The "testability" attribute of a requirement is the key concept that normalizes the requirement, and makes the measure meaningful.

Because the above definition of requirements includes not only the traditional user requirements, but also other types of requirements, there is considerable flexibility to "size" either a complete system or selected aspects of a system. For example:

- user requirements
- design requirements
- requirements for a particular program
- portion of the system that is being changed by a maintenance release
- size of a software package
- size of modifications required to a software package

This raises the question: what do we really mean by the *size* of a system? The answer depends on one's perspective:

- for an end user, the size is the number of user requirements (e.g., FP)
- for a programmer, the size is the number of program requirements (e.g., LOC)
- for an estimator trying to project costs, the size is the total of all types of requirements

A NEW PARADIGM FOR SIZING SOFTWARE SYSTEMS

Testable requirements offers a new paradigm for measuring the size of a system. One way to illustrate the differences between lines of code, function points, and testable requirements is to consider how each would look at the differences between a system developed using a character-based

user interface (e.g., DOS or a 3270 mainframe) and the equivalent system developed using a graphical user interface (GUI) such as Windows.

- *LOC perspective:* It would be very difficult to make a comparison using LOC here because of the differences in the way the interfaces are developed. Because the "language" used to code screen or window functionality is not procedural, it is very difficult to measure LOC, let alone make valid comparisons between two different technical platforms.
- *FP perspective:* Function points would consider the interfaces equivalent because they provide the same functionality to the user (e.g., the same number of logical inputs and outputs). The value adjustment factor could be used to make the GUI a higher function point count. (The value adjustment factor is essentially a complexity factor used to adjust a raw function point count because of factors related to the complexity of the system.)
- *Testable requirements perspective:* Testable requirements would give a much higher count to a GUI than its character-based equivalent. There are simply more conditions (i.e., testable requirements) that are supported by a GUI than are supported by a character-based interface. For example, there is usually only one way to enter a command in a character-based interface: through the menu. In the equivalent GUI interface, there are usually at least three: the menu, the keyboard, and the icon.

Testable requirements can also be used to measure and analyze a system in ways that are not possible with other measures. Because testable requirements can measure external user requirements as well as internal technical requirements, it is possible not only to size the user requirements, but also to quantify their impact on the technical design. Performance requirements, for example, may contribute relatively few testable requirements when viewed from an end-user perspective. However, when viewed from a technical perspective, meeting the performance requirements may require a complex, real-time technical design, which in turn requires many testable requirements.

Similarly, some types of systems such as process control and scientific systems may have relatively few external requirements, but have a very complicated internal technical design. The testable requirement paradigm does not require a complexity factor to account for this. Rather, the complexity will manifest itself in the size of the design, or the size of individual programs. Complexity, in essence, means that there are more testable requirements somewhere.

TESTABLE REQUIREMENTS VERSUS TEST CASES

While the testability of a requirement is fundamental to the concept of sizing with testable requirements, it should be noted that sizing a system using testable requirements is *not* the same as counting test cases. The number of testable requirements may be very different from the number of test cases. There are a number of reasons for this, including:

- A testable requirement may require more than one test case to validate it.
- Not all testable requirements will have associated test cases. (Most systems test less than half the testable requirements.)
- Some test cases may be designed to validate more than one testable requirement.

EXHIBIT 3 — System Size

High-Level Requirement	Testable Requirements
Business	
Accounting	942
Administration	120
Audit and control	225
Editing functions	526
Update functions	450
Maintenance	789
Conversion	2500
Security	569
Reporting	3590
Subtotal	9711
Technical	
Communication	1500
Interface1	225
Interface2	369
Interface3	235
Interface4	436
Subtotal	2765
Operations	
Backup	2340
Recovery	3589
Installation	890
Subtotal	6819
Total:	**19,295**

APPLICATIONS OF TESTABLE REQUIREMENTS

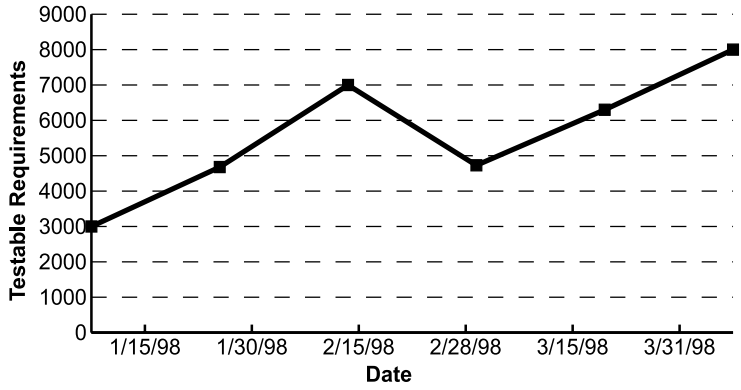
The following examples, which are based on this author's experience, show a few of the ways testable requirements can be used tactically by software developers.

Measuring System Scope and Earned Value

Exhibit 3 illustrates how the size of a system can be summarized. It also shows how the relative size of high-level requirements can be communicated.

Earned value is a measure of completed work expressed in terms of the budget assigned to that work. Although it is widely used on large government procurement projects, its application to software projects has been limited. In large part, this is due to the difficulty in determining the relative size (i.e., value) of the deliverables required to develop a system. Testable requirements appear to have the granularity and flexibility to make earned value a practical tool for software developers.

EXHIBIT 4 — System Size over Time



Measuring Changes In System Size

Understanding, controlling, managing, and communicating system scope is critical to the success of many projects. Exhibit 4 illustrates how changes in a system's size were tracked using testable requirements. The shape of this curve is typical, and illustrates the dynamics of measuring the system size over time. At first, the size will be significantly underestimated, in part because the size estimate will only include what is known by the people sizing the system, and in part because the complexity of some components will be underestimated. As more is learned about the system, the size estimate will grow. Eventually, management will become concerned about scope and some of the requirements will be eliminated. After that, the usual scope creep will cause the size to grow again.

Test Case Coverage Measurements

Testable requirements can also be used to measure the thoroughness of test cases, as illustrated in Exhibit 5. In this example, the first column lists

the high-level requirements; the second column lists the number of testable requirements associated with the corresponding high-level requirement; the third column lists the number of requirements in the second column that are validated by test cases; and the fourth column lists the percentage of testable requirements that are "covered" by test cases. This measure of requirements coverage is very powerful and offers important management insight into the adequacy of the testing.

EXHIBIT 5 — Coverage Status

High-Level Requirement	Testable Requirements	Number Tested	Percent Covered
Business			
Accounting	942	222	24
Administration	120	24	20
Audit and Control	225	98	44
Editing Functions	526	89	17
Update Functions	450	54	12
Maintenance	789	234	30
Conversion	2500	1200	48
Security	569	0	0
Reporting	3590	2400	67
Subtotal	9711	4321	44
Technical			
Communication	1500	17	1
Interface1	225	180	80
Interface2	369	200	54
Interface3	235	175	74
Interface4	436	220	50
Subtotal	2765	792	29
Operations			
Backup	2340	250	11
Recovery	3589	200	6
Installation	890	350	39
Subtotal	6819	800	12
Total:	**19,295**	**5913**	**31**

Experience Using Testable Requirements

Mosaic has found testable requirements to be an intuitive, flexible measure that is a very useful tool for communicating issues to users and management. Difficulties using the measure center around two issues:

1. *The newness of the measure.* There is not a large body of industry data and experience with the measure. It is very easy to underestimate the size of a system because certain types of requirements are not properly incorporated in the size. For example, in contrast to traditional mainframe systems, client/server systems can introduce significant new complexity in the user interface (e.g., the GUI), the database technology (e.g., the need to synchronize a database on a

server with a database on the mainframe system), or numerous other ways. World Wide Web applications are introducing other types of requirements. With experience, we are learning how to identify and size these types of requirements.

2. *Understanding the requirements.* System specifications are frequently vague and incomplete. This can make it difficult to size a system with *any* sizing measure. Sizing with testable requirements will quickly focus attention on areas of the system that are not well-understood or well-specified. While this can be frustrating from a sizing perspective, it provides a valuable early warning of potential problems if the specifications are not corrected.

SUMMARY

The high development, maintenance, and repair costs of software are well-known. Managing software requires measuring it, but too few software developers measure the progress, effectiveness, and productivity of their system development efforts. One reason is that the current software sizing measures are not flexible enough for today's systems. The two most common measures — lines of code and function points — fall far short.

Still, a more quantitative approach to developing software is essential. To achieve this, the software profession needs a sizing measure that is understood and accepted by software developers, users, and executive management. A practical, flexible sizing measure is needed not only to develop more reliable estimates, but also to better communicate with technical and nontechnical management on issues of scope, change, complexity, and the like.

This article shows that the concept of testable requirements meets this need. While the testable requirements measure has been used for years to test the quality and detail of system specifications, the number of testable requirements is also a measure of the size of the system. This new paradigm for measuring the size of a system enables system analysis in ways not possible with other measures. Some applications include measuring system scope and earned value, measuring changes in system size, and measuring the thoroughness of test cases. Difficulties in using testable requirements as a sizing metric include the newness of the measure and understanding the requirements.

Peter B. Wilson is executive vice president of Mosaic, Inc., Naperville, Illinois.

Notes

1. Quality Assurance Institute, *Establishing a Software Defect Management Process*, Quality Assurance Institute Research Report #8, 1995.

2. Wilson, P., Testable requirements — An alternative software sizing measure, *Journal of the Quality Assurance Institute*, October 1995, 3–11.
3. Jones, C., *Applied Software Measurement*, McGraw-Hill, New York, 1991.
4. Jones, C., Sizing up software, *Scientific American*, December 1998.
5. Chicago Quality Assurance Association presentation, November 22, 1996, Chicago, IL.