# Software Design Assessment Testing
Effects and Benefits

Charles E. Matthews
Fifth Generation System, Ltd.
Markham, Ontario, Canada
charles.matthews@acm.org

**Abstract:**
Assessment testing is a common requirement for many fields before an individual can perform maintenance-related tasks. A notable exception to this practice is software maintenance.
This essay examines the impact that assessment testing can have on software product maintenance. Although no studies exist at this time that provide quantitative evidence for the amount of cost savings that assessment testing provides, there is substantial qualitative evidence that a positive return on investment exists. This paper presents that evidence and argues for the case that companies that choose to implement software assessment testing for their maintenance staff can improve the quality of their maintenance activity. For the purpose of this paper, assessment testing refers to assessing an individual's level of knowledge specific to a particular software design. It does not refer to assessing an individual's level of knowledge about any general software or engineering discipline.

## Assessment/Certification Testing:
Certification testing is mandatory for many fields. Physicians are not allowed to practice medicine until they have passed medical board exams – likewise, for nurses and pharmacists. Accountants are not allowed to perform certain jobs until they have passed their certification exams. In these cases, the subjects are tested for general knowledge within their field of study. However, there are also cases where subjects are tested for specific knowledge. Natural gas utility companies must certify their field technicians for their knowledge on how to inspect and repair oil and gas pipelines of particular types and under specific conditions. Aircraft mechanics are not allowed to repair a particular model of jet engine until they have been certified for that model. Thus, assessment or certification testing is a common requirement for many fields before an individual can perform maintenance-related activity in that field.

A notable exception to this practice is software maintenance. The Merriam-Webster dictionary defines the term *assess* as “to determine the importance, size, or value of” and *certify* as “to attest authoritatively”[1]. From a definitional stance, this problem seems straightforward - employ some process to determine the degree of value of an individual's understanding of a design in order to establish a belief that this individual can act authoritatively with respect to that design, i.e. perform maintenance on the design. Unfortunately, the aspect that makes this problem difficult is the speed with which a software design can change. The nature of software is such that it can undergo significant and continuous change. Furthermore, there is rarely much value in knowing what the design “used to be”. There is only value in knowing what the design actually is right now. Two consequences of this high change rate are that any test that can be presented to an individual will soon be obsolete and an individual's state of knowledge/expertise with a given design will also become obsolete unless he remains informed of the design changes.

This high rate of change does not exist in other fields where certification testing for specific knowledge

1 Webster's Seventh New Collegiate Dictionary.

is required, e.g. oil and gas pipeline field technicians or aircraft mechanics. In both of these cases, the state of knowledge that is being tested (the physical characteristics of pipelines or the design of a particular jet engine) remains static and unchanging long enough that assessment tests and procedures can be generated by traditional methods.

Ignoring, for the moment, the difficulties inherent with a constantly changing software design, what value would assessment testing provide for a company? Three characteristics stand out – accuracy, speed, and completeness. Assessment testing improves maintenance accuracy by providing feedback to the individual where a lack of knowledge exists. This feedback triggers further actions such as studying the UML diagrams or design specifications for that particular area. Assessment testing improves maintenance activity speed as a side-effect of improving an individual's knowledge about a design. Maintenance activity is a complex process that requires the utilization of numerous aspects of a program design. When an individual has existing, in-depth knowledge of the entire design, he can utilize that knowledge more effectively. In one of the earliest studies of programming as a psychological activity, Shneiderman stated, *"Instead of absorbing the program on a character-by-character basis, programmers recognize the function of groups of statements and then piece together these chunks to form ever larger chunks until the entire program is comprehended. ... Once the internal semantic structure of a program is developed by a programmer, this knowledge is resistant to forgetting and accessible to a variety of transformations. Programmers could convert the program to another programming language or develop new data representation or explain it to others with relative ease."* [2]

This ability to create conceptual, abstract chunks to represent the design model improves the completeness of maintenance tasks. A better understanding of the design reduces the number of "boundary value cases" that may not have been considered by the programmer when he performed the maintenance task.

**Assessment Costs:**
On the other hand assessment testing has associated costs. The cost of the time for the programmer to take the test may be considered negligible for a one-time event. Only if the company institutes a repetitive testing process with short periods between tests will the testing time become significant enough to track. The frequency of repetitive testing should be based upon one of three scenarios – a) the test is repeated because the subject failed to obtain a high enough score to demonstrate mastery, b) the test is repeated because the design has changed enough to warrant retesting, or c) the test is repeated because enough time has passed since the individual worked with the design that the individual has forgotten significant elements of the design.

In the first scenario, the repetition is necessitated by the need to acquire a sufficient level of knowledge. Once the subject achieves an acceptable threshold score, subsequent testing should occur only if the design changes significantly or the subject demonstrates that he has "forgotten" the design. In the second scenario, maintenance personnel may need to retest because the design changes. For this to happen the product must have undergone a new code release. For active, on-going projects the rule of thumb used within many product companies is a minimum of 6 months between product releases. Anything more frequent leads to product instability, code churn, and customer dissatisfaction. For the third scenario, the individual must have left the project for a long enough period of time to forget key aspects of the design. Unless the subject has unusually short retention memory, it is likely that most

2 Ben Shneiderman, Software Psychology, Winthrop Publishers, Inc., 1980, p. 53.

design knowledge will be retained for at least 3-4 months. Any absence from the project for more than 6 months should probably require retesting. If the company follows these guidelines, then it is unlikely that the cost for programmer testing time will be significant enough to warrant tracking.

Typically, the testing costs that a company tracks are the cost for preparing the assessment test and any overhead cost for administering the test. Overhead costs may be related to the cost for third-party software packages to track student progress. This cost will vary depending upon the sophistication of the third party software or even whether or not it is used at all.

Assessment test generation using traditional methods can be quite expensive. However, Zualkernan[3] demonstrated that product design assessment tests can be created automatically from UML diagrams which are normal by-products of the design process. By using this technique, the cost for creating the assessment test becomes negligible. Because this process uses actual design artifacts that should accurately model the design, then any design change that is updated in the UML diagrams can immediately result in the creation of a new suite of assessment tests.

Although there is a tendency in many software organizations to allow the product documentation (including UML diagrams) to lapse during the development phase of a release cycle, recent studies support the value of maintaining current UML documentation. A joint study of software maintenance tasks conducted by Carleton University in Ottawa, Canada and the Simula Research Laboratory of Lysaker, Norway, found support for maintaining accurate UML diagrams. *"In terms of correctness, despite the fact that the two experiments use different correctness measurement, both experiments show that for the most complex task, the subjects who used UML documentation performed significantly better than those who did not."* [4]

This same study also found a positive correlation between using the UML documentation during maintenance tasks and the quality of the resulting solutions. Therefore, it is beneficial for software companies to place a higher emphasis upon maintaining accurate UML documentation throughout the entire project life cycle than they have in the past.

## Design-Specific Knowledge:

How does one assess an individual's level of knowledge – more specifically, knowledge about a particular design? For that matter, what is knowledge itself? For the purpose of simplicity, consider the following illustration:

Data ==> Information ==> Knowledge

Data leads to Information which leads to Knowledge. Data elements represent a collection of facts. Information adds the concept of relationships to Data to show how different Data elements can be useful to others. Knowledge adds the concepts of process and priority to Information to show how Information can be organized and structured. For the following discussion we will focus upon Knowledge – specifically Knowledge about a design.

---

3 Imran A. Zualkernan, Salim Abou ElNaaj, Maria Papdopoulos, Budoor K. Al-Amoudi, *Automatic Generation of Just-In-Time Online Assessments From Software Design Models*, Project Report, 2006.

4 Erik Arisholm, Lionel C. Briand, Siw Elisabeth Hove, and Yvan Labiche, "The Impact of UML Documenation on Software Maintenance: An Experimental Evaluation", IEEE Transactions on Software Engineering, June, 2006, pp. 365 – 381.

To understand how one can practically assess knowledge, you must be able to construct a test for it. Educational learning theory provides some guidance regarding how to construct these tests. In the mid 1950's Benjamin Bloom proposed that people develop intellectual skills according to a hierarchy of intellectual abilities. According to Bloom's taxonomy of learning[5], human cognition involves knowledge and the development of these intellectual skills. This includes the recall or recognition of specific facts, procedural patterns, and concepts that serve in the development of intellectual abilities and skills. There are six major categories of cognitive knowledge – starting from the simplest behavior to the most complex. The categories can be thought of as degrees of difficulties. That is, the first must be mastered before the next can take place.

1. **Knowledge**: Recall data or information
2. **Comprehension**: Understand the meaning, translation, interpolation, and interpretation of instructions and problems. State a problem in one's own words.
3. **Application**: Use a concept in a new situation or unprompted use of an abstraction. Applies what was learned in the classroom into novel situations in the work place.
4. **Analysis**: Separates material or concepts into component parts so that its organizational structure may be understood. Distinguishes between facts and inferences.
5. **Synthesis**: builds a structure or pattern from diverse elements. Put parts together to form a whole, with emphasis on crating a new meaning or structure.
6. **Evaluation**: Make judgments about the value of ideas or materials.

Different questions can test different categories based upon how the question is worded and structured. For example, “Does A occur before B?” is a knowledge question. It tests the subject's recall of data or information. The question, “How is A transformed into B?” is a comprehension question. It tests more than simple data recall. It requires the subject to understand how a sequence of steps occurs to move from A to B.

To assess an individual's understanding of a product design, the test questions should cover all six categories. Rather than testing general technical knowledge, the test questions should focus upon knowledge about the design. Different artifacts from the design process lead to different assessment questions. Activity diagrams are useful for testing the subject's knowledge about sequential program flow and conditional tests. Object diagrams are useful for testing the subject's understanding about the various relationships between objects in the design. Sequence diagrams are useful for testing the subject's understanding of parallel processes that may be occurring within the design. A critical factor for these questions must be the focus upon what the design actually does. In this way the test assesses the subject's knowledge about a particular design rather than the subject's knowledge about an engineering or software discipline.

**Evolving Designs:**

Because these tests assess the understanding for a particular design, the questions reflect a specific snap shot of the design at a given moment in time. However, many products have an evolving nature for their design that spans multiple releases over multiple years. The assessment process must be able to accommodate the time variant aspects of a product design life cycle.

The most reliable mechanism for handling the changing nature of a design is by constructing the assessment tests from the design model itself. To be successful, it is necessary to have a separate

5 Benjamin Bloom, Taxonomy of Educational Objectives, Handbook 1: The Cognitive Domain, David McKay Co Inc., New York, 1956.

representation of the design model that is accepted by everyone as the final authority. By necessity, this representation cannot be any type of reverse engineered artifact from the software code itself. Otherwise, the model and the implementation are basically the same entity. If this condition existed, then there would be no cognitive advantage gained by having access to a higher abstract model than the code itself. The design model and the code would be one and the same.

Likewise, the design model cannot exist ***only*** as a mental model in the mind of any individual on the project. It must exist as an independent entity and must be generally accepted as an accurate representation of the design itself. If errors in this design representation exist, then they must be regarded as errors in the actual design. When those errors are corrected, a new design exists that represents the corrected design model.

If the design model exists in a machine readable format, then it is possible to write tools to create assessment tests directly from the design model. This is the approach taken by Zualkernan's Sphinx[6] tool. With this tool, the process of generating a new suite of assessment tests is simply a matter of executing the tool against the design snap shot. If it is necessary to generate historical assessment tests against prior versions of the design, then the design model should be managed in a version control system in order to provide access to all past versions of the design model.

For a multi-release product, it is normal for the product design to undergo some amount of change at every release point. The exception to this statement is when the product release is a relatively small maintenance release only. In this case the design model may remain stable through the release cycle. Another exception to the amount of change that the model undergoes at a release point is when the design is intentionally undergoing a massive redesign. This could occur when the product is given a major functional overhaul. However, this case should occur infrequently.

For any release which contains at least some functional enhancements to the product, the design model will typically undergo some amount of change. However, it is unusual for the design model to undergo a massive amount of change. Project designs that have a sound fundamental architecture tend to exhibit inertial characteristics. Although everything in the design model has the potential for changing, significant portions of the product design tend to remain stable from release to release. Design areas that have been stable tend to remain stable. Design areas that have been changing tend to continue to change until they reach a point of stability. This design stability is a characteristic of a good design model.

## Corporate Development Concerns:

In addition to assessing the level of design knowledge for maintenance staff, value exists in assessing the design knowledge for development staff – especially when the development organization has a significant level of staff turnover. When a corporation (or a department) chooses a set of tools to assist its development processes, it may use a variety of reasons to justify its choices. In most cases these choices occur in order to achieve specific benefits or objectives. Common objectives that often influence the tool selection process are:

- Reduce project complexity
- Reduce project schedule
- Reduce product cost (throughout the life cycle)
- Improve product reliability

6 Imran Zualkernan (et al).

- Provide end-to-end solution for business-driven development processes
- Improve SME (Subject Matter Expert) competency

***Project Complexity:***
By assessing a developer's knowledge throughout the project, the developer receives feedback on those areas of the design that he doesn't understand. If those modules are ones with which he must interact, then improving his understanding of that design reduces complexity by reducing the unknown.

Complexity has a dual nature. Typically, complexity is rated on the number of elements and the number of interactions between elements in the object. However, when relevant knowledge of the object is unknown, then complexity becomes infinite because developers are unable to make rational decisions about unknown interactions. By reducing the amount of unknown knowledge about a component, you reduce its complexity.

By reducing complexity, product quality typically improves. When the developer's knowledge of the component has improved, he has a better opportunity for assessing design correctness, identifying unnecessary elements of the design, and identifying a more complete set of scenarios for validating the design.

***Project Schedule:***
Development time is proportional to component complexity. Historical project experience suggests that this relationship is non-linear. When complexity doubles, development time usually increases significantly more than 2x. Historical studies of project schedule as a factor of various program complexity metrics (McCabe cyclomatic complexity, Halstead volume, function points, lines of code, etc) indicate that this relationship has an exponential growth factor. Therefore, reducing product complexity results in a significant improvement in development time.

A secondary effect of improving the design quality and the completeness of test scenarios is a reduction in the number of defects to be fixed prior to release and after product release. Because every defect correction has a certain probability of introducing a new defect, fewer defect corrections imply fewer defects introduced.

***Product Cost:***
Product cost differs from project cost. Total product cost typically spans multiple projects for the product – maintenance releases, feature enhancements, product line growth, etc. Improving product quality reduces maintenance costs – service calls and maintenance releases. Many enterprise-type products adhere to an informal schedule of two maintenance releases per year. Each release cycle costs the software company money – not only for the development cost but also in fixed overhead costs that are necessary for any product release. In addition, the product customers have their own costs associated with updating their product installation. The customer's on-going costs are attributed to the product company. A company which reduces the customer's support costs appears more attractive when compared to its competitors. This perception can translate into a distinct marketing advantage. For example, the Porsche company states that around 70 percent of all Porsche automobiles ever built are still being driven today[7]. This is a statement that no other major automobile manufacturer can make.

7 http://www.porsche.com/canada/aboutporsche/porschephilosophy/whoweare/aps1248/

***Product Reliability:***
End users normally interpret product reliability as a measure of how well a product accomplishes the tasks that it is supposed to accomplish. Although software does not degrade as a function of time in the same way that hardware does, many product organizations still measure product reliability in terms of MTBF (mean time between failures). However, precisely because software products do not degrade as a function of time, they exhibit the characteristic that once a defect is fixed, it is fixed forever. While this is true for the individual defect, it isn't necessarily true for the software product as a whole.

Because the act of fixing a software defect changes the code itself, two mechanisms exist that contribute to future failures. In some cases the modification is an incomplete fix for the defect. If the developer fails to understand the interactions of the design, the incorrect behavior for only some scenarios is corrected. Other scenarios with other interactions may still behave incorrectly. Unless the developer understands all aspects of the design that contribute to the error behavior, this type of oversight is common.

Another factor that contributes to future defects is the possibility that the act of modifying the software may, itself, introduce new error behavior. In this case the probability that new error behavior is introduced is not only a function of the quality of the original correction but also a function of the number of software corrections that occur. Any reduction in the number of defects that need to be fixed will result in a lower probability that future errors will be introduced into the product.

***End-to-End Solution:***
Within the context of a software development organization, the term “end-to-end solution” means that the development organization will provide all resources and perform all activities to meet the customer's requirements and no other supplier need be involved. An end-to-end solution is often a project objective that is satisfied with varying degrees of success for any particular project.

Historically, assessment of product design knowledge has not been an explicit requirement for any development organization. The assumption exists that the developing organization will acquire any and all product knowledge necessary in order to satisfy the product requirements. As will be discussed in the following section, the corporation experiences an economic risk when it assumes that the appropriate level of design knowledge exists and *is available for use when needed* when the developing organization lies outside of the corporation. However, when assessment testing of product knowledge is incorporated within the corporation's development processes, this risk will be minimized. As experience with out-sourced development grows, corporations must take active steps to protect their intellectual property, and assessment testing is a tool that assists them with this task.

***Improve SME Competency:***
A major concern for the “knowledge worker” today is the improvement of the expertise for the Subject Matter Expert (SME). The SME is regarded within an organization as an authority for a given technical subject. The ability for any individual to remain credible as a subject matter expert relies directly upon his ability to make accurate statements about the particular subject domain. If the SME makes incorrect statements based upon a flawed knowledge of a design, then the credibility of this individual falls. Likewise, the perception for an individual who repeatedly makes accurate statements rises. One could argue that the entire economic value of any “expert” is intrinsically tied to the ability to make

consistently accurate statements about a subject, e.g. a product design. Stupid SMEs provide little value for a company.

In today's corporations, the value of the in-house product expert increases exponentially when the corporation uses out-sourced development resources to produce products. Those individuals who perform the product development tasks become the SMEs for that product. Their individual level of expertise across the entire spectrum of the product design domain is directly related to the individual's level of exposure to development tasks across multiple aspects of the design. Unless the corporation takes specific action to train in-house personnel as subject matter experts for the product, the corporation will forfeit that knowledge and, potentially, never be able to acquire it.

The outsourcing company who is developing the product design SMEs rarely has any economic incentive for retaining individuals purely for their product expertise. The value of those individuals to their employer is unrelated to the product design knowledge they have acquired. Unless the original corporation takes explicit action to train their own experts, they will lose significant portions of their intellectual property. Assessment testing is a useful tool that a company can use to verify that it is retaining the product knowledge that is essential to a healthy product lifetime.

**Summary:**

Software design assessment testing is not commonly performed in product companies at this time. However, those companies which begin a process of assessment testing for their development and maintenance staff will see a corresponding improvement in their products. Individual projects can improve by reducing project complexity and project schedule. On a broader company scope which may include multiple project release cycles, benefits for the software product line would be: a) reduced cost across the product lifetime and b) improved product reliability. The existence of tools that can create assessment tests automatically from design documents and models lowers the cost of assessment test creation to an insignificant level. For a company that chooses to implement assessment testing as another tool within its software development process, the most visible effects would be improved accuracy, speed, and completeness for its maintenance processes.