EXODUS | CONSULTING SERVICES

WHITE PAPER

MQDA: Model and Quality-Driven Assessment of Software Artifacts

Printed in the United States of America

Document Identifier: Exodus White Paper 03022013

Executive Summary

In this White Paper we present MQDA, Exodus' Model and Quality-Driven approach for the assessment of software artifacts. We discuss evidence-based results that support the adoption of early assessment practices. The elements that constitute MQDA and a comparison of our approach with others that are well-known are presented. We also discuss how MQDA can be tailored to your organization. Finally, cited references are listed, which readers can use to further their knowledge and understanding of fundamental Software Engineering concepts related to the assessment of software products and processes.

Motivation and Background

"What we have learned about fighting defects"

In 2002, a group of notable individuals from academic and industrial circles wrote a joint technical report[1] entitled: "*What We Have Learned About Fighting Defects*" [1]. Allow us to comment on some of the results found in this report, and, at the same time, to associate them with related results that have been published by other—equally notable—individuals and institutions.

"*More than half of all types of software systems enter use with defects that affect execution.*" This finding is further validated by a 2002 report[2] from the National Institute of Standards and Technology (NIST), which quantifies the business impact of these defects by estimating their cost to the US economy to be about $59.5 billion annually (or 0.6% of the gross domestic product) [2].

"*Finding and fixing a severe software problem after delivery is often 100 times more expensive than finding and fixing it during the requirements and design phase.*" This result is part of evidence-based Software Engineering mainstream wisdom, which warns us about how rapidly the cost of fixing a software defect grows, as a function of the time that elapses between the moment at which the defect is injected, and the moment at which an observed failure unveils the existence of such defect. It turns out that, as indicated by the NIST report: *"… more than a third of these costs, or an estimated $22.2 billion, could be eliminated by an improved testing infrastructure that enables earlier and more effective identification and removal of software defects.*"

"Assess early, assess often" as an evolution of "Test early, test often"

These two findings are summarized by the motto "*test early, test often.*" Unfortunately, since the term "testing" is often associated with executing source code, organizations that rely on traditional testing approaches, in order to detect and remove defects in the software they develop, might miss the "*early*" part of the motto.

Early assessment practices can be highly effective!

This observation has led the Software Engineering community to adopt practices that aim at assessing artifacts produced earlier in the software development lifecycle, upon which the final software product strongly depends. These "early assessment practices"—as we like to call them—include: *inspections*, *reviews*, *audits*, and

[1] Let us refer to this as the "USC report", for it is a technical report from the University of Southern California.
[2] Let us refer to this as the "NIST report".

walkthroughs. Although some software development organizations do use these terms interchangeably, they are not equivalent (see [3] for authoritative definitions of these terms).

In connection with early assessment practices, the USC report mentions: "*Reviews catch more than half of a product's defects regardless of the domain, level of maturity of the organization, or lifecycle phase during which they were applied*." Moreover, if—as proposed in the UCS report—one defines the effectiveness of early assessment practices as the ratio between the number of defects found early, and the total number of defects (some of which are found by testing the final product), then "*a valid heuristic is that reviews find 60-90% of defects.*" This finding has been validated by well-regarded software productivity and quality author (and practitioner) Capers Jones, in his 2008 book [4]. Additionally, Jones points out—in a different article—that omitting inspections is one of the top causes of planning failures that lead to project delays or cancellations [5].

So, if early assessment practices are effective, why are they not embraced by some software development organizations?

An *IEEE Software* paper published in 2007, and co-authored by two researchers associated with the USC report, begins by quoting the 60-90% effectiveness of early assessment practices [6], and then poses the question of why many software development organizations—even at the light of these impressive results—either drop inspections altogether, "cut corners" when applying them, or are simply hesitant to include them as part of their development plans. The authors of this paper offer the following three reasons to explain why early assessment practices have not been embraced by some software development organizations (although measurable results indicate they ought to).

First, there is a perceived lack of connection between the effort associated with inspections and measurable product quality. Therefore, there should be a way of connecting early assessment results with measurable improvements in product quality.

Second, there is a perceived lack of customization of the adopted early assessment practices so they align with specific software development ecosystems, which are characterized by: (a) the people—e.g., culture, degrees of knowledge-related maturity, and cultural diversity; (b) the processes—e.g., software process, management process, and enterprise-wide processes; and (c) the tools that support the whole software development lifecycle. Therefore, the adopted assessment process should be easily and effectively adaptable to a wide variety of organizations.

And—finally—third, key stakeholders (e.g., developers, architects, project leaders, and customer advocates) perceive that early assessment practices are not part of their day-to-day tasks. In other words, these practices are perceived as a diversion from the main business plan. Therefore, early assessment advocates need to make the (measurably) case for the (properly customized) adoption of early assessment practices, as part of a well-defined software process.

What we offer …

Our approach, MQDA—which stands for *Model and Quality-Driven Assessment of Software Artifacts*—is an early assessment process that effectively addresses these challenges. The next section describes MQDA's key concepts.

Exodus' MQDA—Key Concepts

MQDA is quality-driven

We would like to explain the quality-driven aspect of MQDA with the aid of a simple example. Consider the problem of assessing the artifacts that constitute a use case model. For the sake of uniformity, we follow the terminology used by two popular and authoritative references [7, 8].

Although a use case model might be comprised of several artifacts[3], in this example we only concentrate on two classes, namely:

- Use case diagrams, which graphically characterize the functional behavior of a software system by showing its use cases, actors, and their relationships.
- Use case description documents, which describe the interactions between actors and use cases.

Suppose we need to assess a use case diagram. We say the assessment is quality-driven because:

(a) Measurable quality attributes (of interest) must be elicited and associated with the use case diagram; and
(b) A clear and effective procedure must be defined in order to evaluate each of the identified quality attributes.

As a matter of illustration, one possible quality attribute that can be associated with a use case diagram is *syntactic correctness*. An excerpt of an effective procedure that can be used to evaluate this quality attribute is defined in **Side Bar 1**. This procedure is, of course, derived from the Unified Modeling Language (UML) specification [9]. Notice that more complex conditions, not included in the

***Side Bar 1**: Excerpt of an effective procedure used to check the syntactic correctness of a use case diagram.*

1. *The only elements in the diagram are: actors, use cases, and relationships among these elements.*
2. *Actors can only be of two types: **human** and **system**.*
3. *Any two actors can be related only if they are of type **human** and the relationship is generalization.*
4. *The only relationship between a use case and an actor is represented by a solid line that connects them.*
5. *The only relationships between two use cases are: include, extend, and generalization.*

[3] For instance, the following elements are mentioned in [7]: use case model summary, use case diagrams, use case descriptions, glossary, domain model, and non-functional requirements.

sample procedure, might involve inspecting the use case diagram in conjunction with the associated use case documents. For instance, if the diagram shows use cases related by the *include* relationship; the corresponding use case documents must properly reflect the appropriate dependencies. This comment also applies when the *extend* and *generalization* relationships are used.

Other quality attributes that might be relevant in certain contexts include —albeit we do not elaborate on them in this white paper: conformance to prescribed style guidelines, semantic correctness, and the use of prescribed use case patterns and blueprints (see, for instance, [7] for an elaboration of these quality attributes).

MQDA is model-driven

In some cases, once a quality attribute has been associated with a software artifact, it is convenient to create an alternative model in order to effectively (and efficiently) evaluate such quality attribute. For instance, let us consider the example presented in [8], which discusses the refactoring of an actual use case description document.

The original document occupies 5 pages (single-spaced, using a 9-point font), with 44 steps in the basic flow, 10 global alternative flows, and 32 alternative steps associated with the basic flow (and a maximum depth level of 2). The refactored document occupies 3 pages (also single-spaced, using a 9-point font), with 7 steps in the basic flow, 1 global alternative flow, and 40 alternative steps associated with the basic flow (and a maximum depth level of 3).

Therefore, it is reasonable to assume the refactored artifact is an example of a medium-complexity use case description document that can be found in actual software development scenarios.

Suppose this artifact is to be assessed according to the following quality attribute: *determine if all elicited transaction scenarios have been captured; and characterize each of the scenarios with a logical condition* (i.e., the scenario is executed if and only if the condition holds). We agree with those who suggest that an effective way to evaluate this attribute is by using a graph-based representation of the use case description document (e.g., an activity diagram [9]), instead of using the document itself [10].

It is in this sense that we refer to MQDA as being model-driven.

When defining an auxiliary model M to represent a software artifact A, in order to evaluate quality attribute Q (defined for A), we recommend following these steps:

1. Clearly specify how A is mapped to M (according to a properly derived invariant).
2. Find Q' (defined for M), such that evaluating Q on A is equivalent to evaluating Q' on M.

For the example we just discussed, the first step calls for defining how textual elements in the use case description document map to nodes, directed edges, and conditions in the activity diagram. The invariant to which the first step refers is, in this case, that there must be a one to one correspondence between each transaction scenario captured in the use case description document, and a simple path[4] from the initial to the final node in the activity diagram (a.k.a. begin-to-end path).

The second point calls for defining Q' as finding all begin-to-end paths in the activity diagram, and the conditions that characterize them. In general, the idea is that computing Q' on M should be more effective and efficient, when compared to computing Q directly on A. There are, of course, cases for which M is A itself (i.e., there is no need to come up with an alternative model), and therefore Q' is Q itself.

So, in summary …

MQDA approaches the assessment of software artifacts by:

1. Defining measurable quality attributes, and effective procedures that can be used to evaluate these quality attributes (therefore, MQDA is quality-driven).
2. Constructing auxiliary models in order to evaluate the quality attributes effectively and efficiently, when appropriate (therefore, MQDA is model-driven).

We have illustrated the key concepts associated with MQDA by means of a simple example. All the elements we used in the example (software artifacts, quality attributes, effective procedures, auxiliary models, and invariants) were chosen for illustrative purposes, and for the sake of clarity, simplicity, uniformity, and brevity.

When adapting MQDA to your organization, these elements are chosen so they align with the strategic values of the software project at hand, your software development ecosystem (process, people, and tools), and your business bottom line. This pertains to how MQDA is implemented, which is discussed in the next section.

[4] For the sake of simplicity, let us assume the graph is acyclic. Otherwise, the invariant and Q' must be properly defined.

Implementing MQDA

The MQDA process

1. **Initial Contact**: Exodus representatives meet with your IT team in order to elicit and document the software assessment needs of the organization, and its ecosystem (people, process, and tools). IT liaisons are identified and appointed.
2. **Project Proposal (which includes the adaptation of MQDA to your organization)**: Exodus presents a project proposal to IT liaisons. Some of the elements in the proposal include (but are not limited to): timeline, fixed project cost, assessment activities (which entail working with artifacts, quality attributes, auxiliary models, transformations to auxiliary models, assessment procedures, use and/or development of tools, and metrics to be gathered—all of which are specifically tailored to your organization), regular progress reports, and final report with recommendations. Pertinent changes are incorporated into the proposal.
3. **Agile Execution of Agreed upon Project Proposal**: Exodus staff works on executing the project as per the agreed upon plan, in collaboration with the IT liaisons. Periodic meetings with the IT team (e.g., once every week) are in place to assess the progress of the project. These meetings allow Exodus to introduce appropriate refinements into the agreed upon plan.
4. **Findings, Recommendations, and Post-Mortem Meeting**. Exodus prepares a document for—and presents it to—the IT team, with the results, analyses, and recommendations associated with the assessment project. When appropriate, recommendations might lead to further professional engagements in the form of training, coaching, and software process definition and/or refinement.

MQDA and other assessment approaches

We conceptualize the process of assessing software artifacts as the composition of the following orthogonal axes: *Verticals*, *Values*, *Logistics*, and *Practices*. These axes are informed by known assessment approaches including (but not limited to): Fagan Inspection Process [11], Wiegers' Peer Reviews [12], Gilb and Graham's Software Inspections [13], Parnas and Weiss' Active Design Reviews [14], Denger and Shull's TAQtIC [6], the IEEE Standard for Software Reviews and Audits [3], and recommendations put forth by Capers Jones [4, 5, 15].

The *Verticals* axis refers to the application domains associated with your software development organization (e.g., banking, insurance, transportation and logistics, etc.) The *Values* axis refers to issues related to human factors (e.g., not using assessment results punitively against developers). The *Logistics* axis refers to issues related to

how assessment activities are executed from an organizational standpoint (e.g., defining clear roles, responsibilities, and deliverables in connection with all assessment activities). Finally, the *Practices* axis refers to the actual assessment activities that are executed (e.g., analyzing software artifacts using effective procedures to evaluate quality attributes).

MQDA contributes to this body of knowledge in the area of *Practices*, by proposing an approach based on early assessment of software artifacts through the evaluation of measurable quality attributes, using model-based effective procedures. However, we are cognizant of the fact assessment practices must be—in some cases—highly specialized. A short list of examples include: evaluation of software architectures [16], threat modeling for security assessments [17], formal methods [18], and model checking [19].

Addressing your concerns

Connecting Early Assessment with Product Quality: After Exodus has conducted the first two steps of the MQDA process in cooperation with the IT liaisons (i.e., *Initial Contact*, and *Project Proposal*), our characterization of your development ecosystem (people, process, and tools) will inform the mining of all the quality-based information that is relevant to your organization, including: software artifacts to be assessed, quality attributes to be evaluated, evaluation procedures to be used, auxiliary models to be constructed, and tools that will be used and/or developed.

Customizing MQDA to your Organization: In addition to the elements mentioned in the previous point—defined as part of our *Project Proposal*—other elements taken from the four axes of well-known approaches (*Verticals*, *Values*, *Logistics*, and *Practices*) are tailored to fit your organization's ecosystem. For instance, in cooperation with the IT liaisons we might adopt a medley between Fagan Inspection Process [11] and the IEEE Standard for Software Reviews and Audits [3], supplemented by the capturing of some of the metrics suggested by Jones [4].

Relevance of MQDA to your Business Bottom Line: A software development organization that gets in touch with us typically belongs to one of the following categories.

(a) The organization is aware of the direct connection between early assessment practices, product quality, and its business bottom line. Moreover, the organization follows a well-defined software process that includes the quantification of this connection.
(b) The organization is aware of the direct connection between early assessment practices, product quality, and its business bottom line. However, the organization

follows a software process (even if it is a de-facto process) that does not include the quantification of this connection.

(c) The organization has an intuition that it must increase its awareness of the connection between early assessment practices, product quality, and its business bottom line. However, the organization needs assistance so it can initiate this improvement process.

Regardless of the category your software organization belongs to (and even if it belongs to a category not listed here!) we will work with you to establish, refine, and quantify the connection between the quality of the products you produce and your business bottom line.

Let's talk!

Please do not hesitate to contact us for a free-of-charge, one-hour, onsite visit so we can explore how Exodus can help your organization to assess and measure quality aspects of your software process and products. Send us an email to Solutions@ExodusSoftServices.com.

References

[1] Forrest Shull, Victor Basili, Barry Boehm, and others: "*What We have Learned About Fighting Defects*". Technical Report USC-CSE-2002-517. Center for Systems and Software Engineering. University of Southern California.

[2] National Institute for Standards and Technology (NIST): "*The Economic Impacts of Inadequate Infrastructure for Software Testing*". Planning Report 02-2 (May 2002).

[3] Computer Society of the Institute for Electronic and Electronic Engineers (IEEE): "*IEEE Standard for Software Reviews and Audits*". IEEE Std 1028™-2008 (August 2008).

[4] Capers Jones: "*Applied Software Measurement*". Third Edition. McGraw-Hill Osborne Media (2008). ISBN: 0071502440.

[5] Capers Jones: "*Software Project Management Practices: Failure Versus Success*". Crosstalk—The Journal of Defense Software Engineering. October 2004 issue.

[6] Christian Denger and Forrest Shull: "*A Practical Approach to Quality-Driven Inspections*". IEEE Software. March/April 2007 issue. Pages 79-86.

[7] Gunnar Övergard, and Karin Palmqvist: "*Use Cases—Patterns and Blueprints*". Addison-Wesley Professional (2005). ISBN: 0131451340.

[8] Alistair Cockburn: "*Writing Effective Use Cases*". Addison-Wesley Professional (2000). ISBN: 0201702258.

[9] Object Management Group (OMG): "*UML Superstructure Specification*" (version 2.4.1). Available from http://www.omg.org/spec/UML/2.4.1/Superstructure

[10] Paul Ammann, and Jeff Offutt: "*Introduction to Software Testing*". Cambridge University Press (2008). ISBN: 9780521880381.

[11] Michael Fagan: "*A History of Software Inspections*". In Manfred Broy and Ernst Denert (Editors) "*Software Pioneers—Contributions to Software Engineering*". Springer (2002). ISBN: 3540430814.

[12] Karl Wiegers: "*Peer Reviews in Software*". Addison-Wesley Professional (2001). ISBN: 0201734850.

[13] Tom Gilb, and Dorothy Graham: "*Software Inspection*". Addison-Wesley Professional (1994).

[14] D. L. Parnas, and D. M. Weiss: "*Active design reviews: Principles and practices*". Journal of Systems and Software. Volume 7, Issue 4, December 1987, Pages 259–265.

[15] Capers Jones: "*Software Engineering Best Practices*". McGraw-Hill Osborne Media (2009). ISBN: 007162161X.

[16] Paul Clements, Rick Kazman, and Mark Klein: "*Evaluating Software Architectures: Methods and Case Studies*". Addison-Wesley Professional (2001). ISBN: 020170482X.

[17] Frank Swiderski, and Window Snyder: "*Threat Modeling*". Microsoft Press (2004). ISBN: 0735619913.

[18] Daniel Jackson: "*Software Abstractions*". MIT Press (2006). ISBN: 9780262101141.

[19] Christel Baier, and Joost-Pieter Katoen: "*Principles of Model Checking*". MIT Press (2008). ISBN: 9780262026499.

About Exodus

We are a Software Engineering consulting firm constituted by highly qualified professionals who share a passion for problem solving that requires a combination of ingenuity and the informed use of appropriate Computing practices.

We are grateful we have been blessed with having had the opportunity of working with diverse platforms, environments, frameworks, and software processes.

Since we are keenly aware of the importance of staying current in this our fast-paced discipline, we continue to be actively engaged in learning from others, and sharing our knowledge with others. We accomplish this by participating in activities such as: publishing peer-reviewed articles, books, and chapters of books associated with well-regarded organizations; and also by organizing/participating in professional workshops and tutorials.

Thanks to our education and to these activities, we have been exposed to advanced Computing concepts such as Data Structures, Algorithms, Numerical/Statistical Analysis, Machine Learning, Data Mining, Ontologies, Service-Oriented Architectures, Domain-Specific Languages, Computer Security, and Model-Driven Development, to name a few.

We are always ready to—and enjoy the challenge associated with—"rolling our sleeves up" in order to rapidly understand, apply, and extend new technologies in order to assist you in getting your job done on time, and with the levels of quality that you require. We also immensely enjoy combining seemingly disparate technologies, and—excuse the cliché—thinking outside of the box.

If you feel your development team is currently stuck and/or needs a breather, get in touch with us! Exodus Consulting will offer an exit strategy that is refreshingly affordable, independent, timely, organic, and with the appropriate breadth and depth that aligns with your organization's bottom line.

Visit our website at ExodusSoftServices.com

Send us email to Solutions@ExodusSoftServices.com