Software Quality Model and Framework with Applications in Industrial Context

Lajos Schrettner
InfoPólus 2009 Ltd.
Gutenberg u. 14.
H-6722 Szeged, Hungary
schrettner@infopolus.hu

Lajos Jenő Fülöp
DEAK Plc.
Dugonincs tér 13.
H-6720 Szeged, Hungary
flajos@inf.u-szeged.hu

Árpád Beszédes, Ákos Kiss, Tibor Gyimóthy
University of Szeged
Department of Software Engineering
Árpád tér 2, H-6720 Szeged, Hungary
{beszedes,akiss,gyimi}@inf.u-szeged.hu

***Abstract*—Software Quality Assurance involves all stages of the software life cycle including development, operation and evolution as well. Low level measurements (product and process metrics) are used to predict and control higher level quality attributes. There exists a large body of proposed metrics, but their interpretation and the way of connecting them to actual quality management goals is still a challenge. In this work, we present our approach for modelling, collecting, storing and evaluating such software measurements, which can deal with all types of metrics collected at any stage of the life cycle. The approach is based on the Goal Question Metric paradigm, and its novelty lies in a unified representation of the metrics and the questions that evaluate them. It allows the definition of various complex questions involving different types of metrics, while the supporting framework enables the automatic collection of the metrics and the calculation of the answers to the questions. We demonstrate the applicability of the approach in three industrial case studies: two instances at local software companies with different quality assurance goals, and an application to a large open source system with a question related to testing and complexity, which demonstrates the complex use of different metrics to achieve a higher level quality goal.**

***Index Terms*—Software quality assurance, Metrics, Goal Question Metric, Quality model, Modelling, Data persistence.**

I. Introduction

Measurement is an essential constituent of any software quality assurance activity [1]. Without measuring different properties of the product and process it is impossible to assess quality issues, and prepare for their mitigation either in form of prevention, refactoring or any other kind of risk management. There is a significant body of work published about what kind of measurements and specific metrics should be used for this purpose, e.g. see [2]. These metrics are properties of the system or the process that can be expressed numerically. Also, there are different recommendations about how quality should be defined in the first place for a specific project and goal (e.g. see the ISO 9126 standard). In other words, what are those higher level quality attributes that should be managed (specified, tested and maintained). Finally, there are different standards about the overall quality assurance of software processes [3], [4].

This work is based on the Goal Question Metric (GQM) paradigm [5] in the sense that we suggest a method for combining metrics and questions in a unified model to bridge the gap between high level goals and low level metrics. The key idea of this paradigm is that a measurement must be defined in a top-down fashion. A bottom-up approach has several drawbacks because there are many characteristics in software, but selecting the important ones is not straightforward without well defined top-level goals. The paper makes two contributions.

First, we implemented the approach in a framework that supports the modeling, collecting, storing and evaluating software measurements, which can deal with all types of metrics collected at any stage of the life cycle. The model and its representation allows the definition of various complex questions involving different types of metrics, while the supporting framework enables the automatic collection of the metrics and the calculation of the answers to the questions. The novelty of the approach is the combined use of metrics and questions. Second, we present details about three industrial applications for which a monitoring environment has been set up using the model and quality framework.

There are some similar works ([6], [7], etc.) but none of them provided all the features we deemed important: (1) build on the basis of the GQM paradigm, (2) define models for data storage, (3) store and manage the questions together with the data and metrics, (4) should be extendible with regard to data uploading and model extension.

The paper is organized as follows. Section II describes the concepts and design decisions that led to the development of the Unified Quality Monitoring framework. In Section III, we give some details about the implementation of the framework, while Section IV is dedicated to details and experiences with the applications. Finally, we conclude in Section V.

II. Software Development Life Cycle Monitoring Approach

Principles of a possible solution: Based on the above we define high level concepts that describe the principles and key requirements of a possible solution. This concept is depicted in Figure 1.

The left hand side of the figure shows the concepts of measurement based on GQM. The right hand side of the figure shows the modelling concepts for the structural elements and relationships of the software under investigation. An important

detail in the GQM paradigm is that when the GQM model is defined, then the appropriate data collection techniques and tools have to be developed.We call this kind of tools *adapters*, as can be seen in Figure 1.

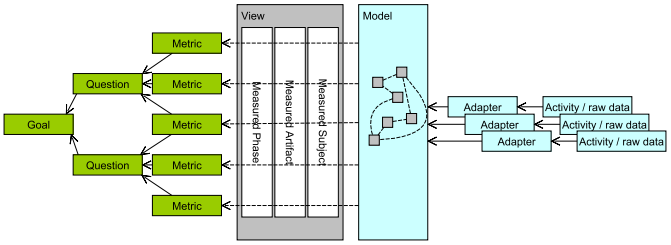

Figure 1. High level concepts for a GQM- and model-based software measurement solution

An important part of this figure is the *View*, which is located in the middle, between the *Model* and the *Metrics*. *View* defines the viewpoint of the model from three aspects: *measured phase* (eg. design, implementation), *measured artifact* (eg. design documents, source code), and *measured subject*. The latter deals with the categorization of the measured metrics: *product* (eg. lines of code) and *process* (eg. average time to fix a bug).

The interpretation and the need for a viewpoint are usually specific to a certain project. For example, in the case of *Measured Phase* it is possible to measure phases such as operation and maintenance that are parallel to each other, i.e. may be carried out at the same time. Therefore, the viewpoints have to be selected first, then the metrics should be determined, taking into account the viewpoints.

Data model: We developed a model based on the principles above. The most important part of the developed model, which we call *library*, contains the common general elements of the whole model. Among the elements there are general items from which specific measurable entities can be derived, items from which specific metrics can be derived, and top level elements that are required in every application (such as identifying the system under observation, and the versions of the system). The library is defined via a static UML diagram, but due to space constraints its details cannot be presented here. We defined a *reference process* to extend and customize the library to suit the needs of a specific field or application. The reference process consists of four steps: (1) defining the structure of the measurement, (2) identifying the measured elements, (3) defining software quality metrics, and (4) determining constraints for the new measurement. The process does not deal with the high level quality attributes, because the extension process is general, while goals and questions are domain-dependent or project specific. While the questions should determine the type of metrics to use, the library can be seen as a set of possible metrics at our disposal.

III. Implementation

Based on the principles defined in the previous section we developed the Unified Quality Monitoring (UQM) application. The UQM application integrates the collection of quality data, high level query management and reporting features. It has been implemented on top of our Model Based Persistence Server (MBPS) framework, which is capable of supporting clients that benefit from viewing their persistent data as a collection of nodes (and connections) instead of as records of a relational database. Its main distinguishing feature is that besides serving as a vehicle for model-based data storage, it supports the integration of queries into the models. Our MBPS framework provides the basic data abstraction and persistence infrastructure on which the higher level quality monitoring functions are built.

The UQM application is still under development, but it has already been used with success in several academic and industrial projects. Quality data accumulates in the model instance by the use of special client components, called adapters (Figure 2). Adapters are data transformation devices that collect data from various sources, restructure them as dictated by the model, then forward the resulting elements to the persistence server. There are a predefined set of adapter types that are prepared to be able to collect data from sources that have been encountered so far in different projects. Adapter types and supporting classes are arranged in an inheritance hierarchy, so new adapter types can be included into the system relatively easily if the need arises.

In a particular installation of the UQM application, those adapters that are determined to be necessary for the operation of the system have to be instantiated. Most adapters require that parameters be provided at instantiation time to be able to connect to their data source. Instantiated adapters can be set up to operate under the supervision of a scheduler that activates them at preconfigured moments, or they can be activated manually.

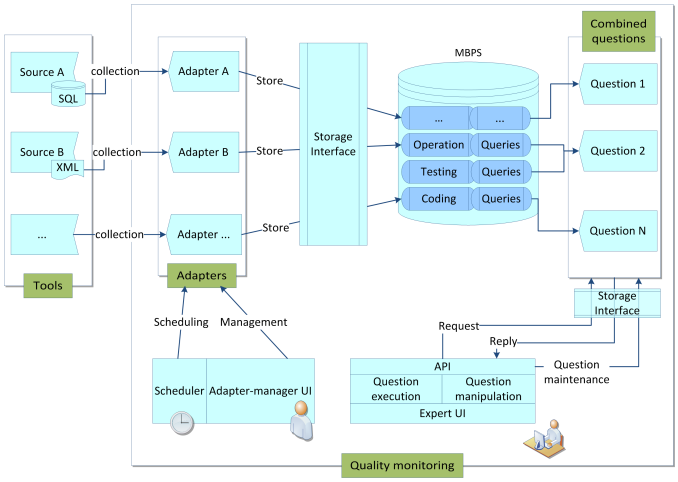

Figure 2. UQM architecture

Data collected through adapters accumulates in the persistent storage, in the current model instance, where it can be queried from. Queries are stored next to the model, and can have parameters that should be filled in before execution. The parameters together with the constraints that are described inside the queries determine a subset of the nodes and connections of the model instance. This way they are similar to

relational queries, except that they produce a subgraph, not a series of records. Queries can encode a limited number of aggregation functions (e.g. count, sum, average), and it is also possible to attach custom post-processing Java code to them. The results of an executed query can be viewed textually in the simplest case, or they can appear on a diagram if the output type of the query is one of those for which diagrams are predefined. The current set of diagrams consists of several two and three-dimensional bar charts, timelines and other reports.

There are three views in the unified user interface of the system, two for administrative tasks (Adapter management, Question manipulation) and another one for the end users, i.e. the experts who would like to monitor the quality of the observed system(s).

IV. Application experiences

In the following three subsections we present experiences with three industrial applications.

A. Code Complexity and Regression Testing in WebKit

WebKit is an actively developed open source web browser engine [8] consisting of about 1.8 million lines of C/C++ code with a regression test suite consisting of about 20000 test cases that are run after every commit to the WebKit source code repository. In principle, the layout regression tests must pass before any patches can land in the repository, but unfortunately this requirement is often violated. Developers often skip the full testing process before the commits, because regression testing is a complex task that requires a lot of time and other resources.

We have set up a quality monitoring environment in which data acquisiton is triggered after each commit and formulated research questions that we would like to answer using our quality framework. Specifically, we have set up a measurement environment in which we are able to:

- Analyze the source code of any revision of the project and compute a set of code metrics for that revision.
- Instrument the code and measure procedure level code coverage during regression testing.
- Extract Passed/Failed outcome information from the regression test suite.

Using the Unified Quality Monitoring approach, we gather data from the development and the regression test phases of the WebKit life cycle. Two adapters need to be instantiated for WebKit, one for the development phase, and another one for the testing phase. The primary goal of our research in connection to WebKit is centered around investigating the connection between complex methods and regression errors.

Here, we deal with research problems that occur in the implementation and testing phases of the software development life cycle, therefore we specialized the necessary components of the *library* model (see Section II) to accomodate the data gathered from WebKit. GQM [5] introduces a summary table format which we also use in this paper. Table I summarizes a GQM triplet in connection with WebKit.

Goal	Purpose	Investigating the connection between complex methods and regression errors.
	Issue	What is the most common cause of regression errors
	Object	in the system under development?
	Viewpoint	Software quality assurance
Question	Q1	Is it true that in a revision the most complex methods of the system are responsible for (some of the) regression errors
Metrics	M1	Complexity of methods (e.g. McCabe, NOI, NII)
	M2	Set of modified methods between two revisions
	M3	Set of methods that were covered during regression testing of a revision
	M4	Regression test outcomes in a revision

Table I
GOAL: RELATE COMPLEX METHODS AND REGRESSION ERRORS

The answer for the question included in Table I is given by the UQM application as a diagram where 3 values are assigned to each revision in the given interval: (1) number of complex methods, (2) number of methods that caused a test case to fail, and (3) number of elements in the intersection of the set of complex methods and the set of failing methods.

Preliminary results show that there is indeed a connection between complex methods and regression errors. Using our framework, we collected data from 26 revisions. We found that the ratio of complex methods among the ones that cause any test case to fail is consistently higher than the ratio of complex methods among all methods. Depending on the definition of "complex", complex methods occur 2 to 5 times more frequently among failing methods. For example, if we regard a method "complex" if its NII (Number of Incoming Invocations) metric is greater or equal to 10, then 3% of all methods are complex, but 15% of them take part in executions that lead to a test case to fail. For technical reasons, the number of revisions we took into account were somewhat limited, and coverage information was not available for all revisions, but the flexibility of the UQM application and more specifically the way the answer is calculated enables us to continue this promising line of research. We would like to continue to work on these and other (eg. impact analysis) issues, but they are out of the scope of this paper, as the aim was to demonstrate the use of the UQM application.

B. Usability Testing in the DEAK Project

In this case study we chose one of the joint projects with our industrial partner DEAK [9]. The project is about usability testing of large and complex web applications in the domain of library administration. Usability information is collected from deployed systems, therefore we developed a specific model (ie. a specific measurement) for this project, which models the operation phase of the software development life cycle.

Some of the major goals of the DEAK project are to improve the less usable parts of the systems, and determine the unnecessary and confusing parts. We worked on developing automatic methods and questionnaires, for which purpose our quality framework is perfectly suitable. To facilitate the above, the systems are extended with capabilities to record the user interactions and store them in specific database tables. The

Goal	Purpose Issue Object Viewpoint	Improve the less usable part of the systems from the users viewpoint
Question	Q1	What functionalities require the longest execution time?
Metrics	M1 M2	Recursive time Existing time
Question	Q2	Which functionalities are the most difficult to use?
Metrics	M3 M4 M5	Number of errors Number of steps Number of help use

Table II
GOAL: IMPROVE THE LESS USABLE PARTS OF THE SYSTEMS

Goal	Purpose Issue Object Viewpoint	Characterizing the response delays in the observed safety critical system (e.g. teller machine) for the executive manager (of a bank).
Question	Q1	Were there any slowdowns during the last 2 weeks?
Metrics	M1 M2	Processes (and related process steps) ClearTime, ExistingTime

Table III
GOAL: CHARACTERIZING DELAYS IN A SAFETY CRITICAL SYSTEM.

data from these database tables are then transferred by a specialized adapter to the UQM application.

In the following, we give two example questions in this domain, presented in GQM tabular form. The questions in Table II are typical usability testing questions, and they illustrate the connection between metrics and questions well.

The filled questionnaires and the automatic answers of the framework gave similar results. This way the most problematic parts (e.g. window resizing, saving search forms, etc.) of the investigated library applications had been discovered, and then fixed by developers.

C. Quality Platform of an Industrial Consortium

A large-scale cooperation among local software companies and the University of Szeged resulted in a project whose aim is to develop a common platform for software quality assurance with related models, methodologies and tools. The companies of the *InfoPólus* consortium [10] are interested in the platform in order to reduce their software quality assurance costs. This particular UQM application uses multiple sources, while question vary in their scope: some are related to only a single measurement, but there are ones that are based on several measurements. The initial development of both the quality model and the framework were heavily influenced by collaborating with the partners of the *InfoPólus* consortium.

There are a lot of different areas involved in this big project including code quality assurance, test efficiency measurement, data migration, safety critical systems operation monitoring, and others. For the purpose of illustrating the model in this paper, we selected the latter, which we characterize with the question in Table III. Here *ClearTime* is interpreted as the measured completition time of certain phases of the operation of a teller machine (Recognition of inserted card, waiting time for PIN number, etc.), while *ExistingTime* is interpreted as the maximum allowed completition time for the same phases. When executed, the question above produces a diagram showing the number of occasions each phase of operation took longer than it was allowed to in the last two weeks.

V. CONCLUSIONS

We presented an approach to aid software quality management by modelling and tooling. We think that our approach provides a common basis for any task related to measurement in software quality assurance since it can serve as an infrastructure for any kind of metrics and, following the GQM paradigm, evaluation of the metrics for specific purposes.

The present article showed only a portion of the models and tools that we are constantly developing in the scope of our R&D activities. We gave some details about our solution, both on conceptual and implementation level, and gave three concrete examples of the application in very diverse contexts.

We will make some parts of the core model publicly available so that other researchers and practitioners could also benefit from it. In the future, we plan different activities regarding the development of the approach and its publication. We plan to publish the details of our model (library) and further details about the industrial applications. We will develop further questions, data collection and visualization tools as we move forward with our projects, so we intend to publish details about these as well. Finally, according to GQM, *"Learning process: GQM models need always refinement and adaptation..."*, hence we will constantly focus on the evolution of our models to keep their usefulness on a maximum level.

ACKNOWLEDGEMENT

The authors would like to thank György Hegedűs for his help with the WebKit case study. This research was supported, in part, by the Hungarian national grants OTKA K-73688, GOP-1.2.1-08-2009-0005 and GOP-1.1.2-07/1-2008-0007.

REFERENCES

[1] T. DeMarco, *Controlling software projects: management, measurement & estimation*, ser. Yourdon Press computing series. Yourdon Press, 1982.

[2] L. M. Laird and M. C. Brennan, *Software Measurement and Estimation: A Practical Approach*. Wiley-Interscience, 2006.

[3] M. B. Chrissis, M. Konrad, and S. Shrum, *CMMI Guidlines for Process Integration and Product Improvement*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[4] *Systems and software engineering – Software life cycle processes*, ISO/IEC 12207:2008 ed., International Standards Organization, 2008.

[5] V. R. Basili, G. Caldiera, and H. D. Rombach, "The Goal Question Metric Approach," in *Encyclopedia of Software Engineering*. Wiley, 1994.

[6] Christian Hein and Tom Ritter and Michael Wagner, "Model-driven tool integration with modelbus," in *Workshop Future Trends of Model-Driven Development*, 2009.

[7] F. Deissenboeck, L. Heinemann, M. Herrmannsdoerfer, K. Lochmann, and S. Wagner, "The quamoco tool chain for quality modeling and assessment," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: ACM, 2011, pp. 1007–1009.

[8] "The WebKit homepage." [Online]. Available: http://www.webkit.org/

[9] "The DEAK homepage." [Online]. Available: http://www.gop.deakszeged.hu/language/en

[10] "The InfoPólus homepage." [Online]. Available: http://www.infopolus.hu/index.php?lang=en