EXTERNAL VALIDATION OF A METRICS-BASED QUALITY ASSESSMENT OF THE JWAM FRAMEWORK

Claus Lewerentz, Frank Simon,
Frank Steinbrückner,
Software Systems Engineering Research Group
Technical University Cottbus
e-mail: {cl, simon, fsteinbr}
@informatik.tu-cottbus.de

Holger Breitling, Carola Lilienthal,
Martin Lippert
Computer Science Department, SE group
University Hamburg
e-mail: {breitling, lilienthal,lippert}
@jwam.org

Abstract: Product metrics allow for an efficient way to get feedback about the quality of an existing software system. Beside the internal validation of metrics, i.e. their examination with respect to measurement theory, the external validation is important to examine the value of results based on metrics analysis. In this paper, a well-defined process to get a metrics-based quality assessment is applied to the Java Framework JWAM. For the external validation its results were analysed by the JWAM development team: Most aspects were valuable to them and the findings were classified as either *completely new*, *confirming rough knowledge*, or *well known*. Additionally, a second process walkthrough for the new JWAM-version showed that many detected quality defects have been removed.

1 Introduction

In today's software development one of the major goals is to produce high quality software. The concept of quality depends heavily on the view of the system. Whereas the end user of a system is primarily interested in the usability and friendliness of the product – which depends on criteria like ergonomic look and feel, consistency to other products or how easy it is to understand (cf. e.g. [ISO9241]) – the developer of a system is primarily interested in typical engineering goals like maintainability, efficiency and portability (cf. e.g. [ISO9126]). In correspondence to these two views, quality can be refined into *external quality* to cover the end users' interests and *internal quality* to cover the engineers' interests (cf. [GhJaMa91]).

The quality assessment of the JWAM-framework (and the external validation) described in this paper concentrates on internal quality because this is especially important for frameworks: There exist two separate groups of engineers: the developers of the framework itself and the programmers using it to build a software based on this framework. Both are primarily interested in internal quality.

One of the major approaches to ensure internal quality is of course to concentrate on quality aspects from the very first stages of the software development process. Many processes support this constructive quality assurance (e.g. eXtreme programming or cleanroom technique). Nevertheless, none of these techniques prevents the system from loosing structure during further evolution. This *law of increasing entropy* [BeLe79] covers the fact that the entropy of a system increases with time, unless special care is taken to maintain the system. In software industry this demand is contrasted by release pressure, bug fixing and resource shortage. Any management activity has to handle these two sides and depends on the following questions:

- What is the current quality of the system?
- When does the loss of structure of a system exceed a particular but not necessarily predefined limit?
- Where to apply a fixed effort for reengineering activities to get maximal benefit?

From our point of view one practicable technique to answer these questions is product measurement because

- its application can be highly automated,
- its application does not need much effort (it even can be outsourced),
- its application allows to identify the most anomalous parts with respect to a given quality model (cf. [FePf96]) and
- its application provides a very condensed quality-based view on the whole system.

One kind of empirical proof for this hypothesis is the *external validation* of the used measurement program, i.e. the demonstration of a consistent relationship between some metrics and some available empirical data (cf. [Zuse98], p. 536).
The following paper describes an external validation by applying a measurement-based quality assessment on a large software system and by empirically evaluating its results. These two distinct tasks were done by two separate groups:

- The software system engineering group at the Technical University Cottbus did the measurement-based quality assessment and
- the SE group of the computer science department of the University Hamburg provided the source for the examination and evaluated the results of the quality assessment.

To prevent random correlations, this assessment-evaluation-cycle was applied twice for different versions of the considered system. Thus, this work is compatible to the external validation process introduced by Schneidewind and its guideline for the validation of software metrics ([Schn92], [IEEE1061]). Additionally, the results of the first walkthrough were used for an improvement of the measurement program and the used metrics.

The paper is structured as follows: In Section 2 the measurement program and the used process to create a quality assessment are briefly explained. In Section 3 the examined system is introduced and some additional parameters for the quality assessment are presented. Section 4 shows the results of both the assessments and its evaluations. Additionally, the adjustments to the measurement program that were applied for the second process walkthrough are explained. Section 5 gives a summary and some outlooks.

2 Process for metrics-based quality assessment

The process for a measurement-based quality assessment used in this case study is similar to the one presented for a measurement-based review preparation in [KöRuSi98]. Figure 1 shows the structure of this process.

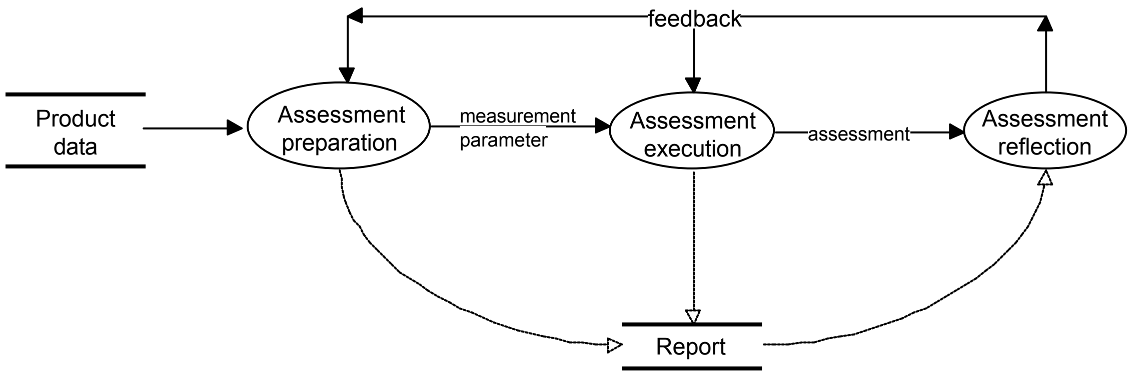

Fig. 1: Process for measurement based quality assessments

Since this process to some degree is just an extension of the ISO 14598 (cf. [PuSoTr97]) – which is concerned with the evaluation of software product quality – with a reflection and feedback action we explain it in a condensed way and focus on some particular process parameters we applied:

1 The *assessment preparation*, which corresponds to the three ISO-sub-processes analysis, specification and design. The ISO calls the process's output *evaluation plan.*
   1.1 *Preparation of the product data*: This step includes getting the product data, extracting relevant data and maybe some pre-processing with respect to some inheritance relations to consider the complete available functionality of a class (cf. "*flatten*"-technique in [SiBeLe01]).
   1.2 *Preparation of the measurement environment*: To get maximal benefit of the measurement technique, tool support is necessary. To consider different quality views a *quality profile* has to be created (cf. ISO 14598) which in turn should be used for adjusting the tool to the special evaluation context.
   1.3 *Planning the assessment*: To do the assessment in a well-organised manner a detailed resource plan has to be made explicit. It contains data about the time available for the assessment process and data about how much resources can be used (personal, money, computers, etc.).

2 The *assessment execution* (called *evaluation execution* in ISO 14598) consists of the following subtasks:

  2.1 *Measurement execution*: In this step the measurement environment calculates the measurement values for the metrics that are defined in the quality model (cf. 1.2).

  2.2 *Measurement visualisation*: To get an impression about the quality of the product and to find some anomalous parts the huge amount of data produced in step 2.1 has to be visualised in an intuitive way. For this dynamic, explorable 3D models are created representing the following data (cf. [LeLöSi99], [LeSiSt00]):

  – Structure elements like subsystems, files, classes or methods/attributes.
  – The distance between the displayed elements can be based on usage relations (the more interactions between two elements, the smaller their distance) and inheritance relations (the more similar the set of superclasses, the smaller their distance). Using the distance in this way allows for the easy detection of cohesive parts.
  – The size of a displayed element can be chosen in relation to some metrics (e.g. the size for a class can be chosen in relation to its number of public methods).
  – The colour of a displayed element is chosen in relation to the subsystem in which the element is defined: All elements of one subsystem have the same colour.
  – Additionally, it is possible to enrich the visualisation by the relations themselves (e.g. usage-relations or inheritance relations) which are displayed by directed lines between the corresponding elements.

  2.3 *Quality assessment*: The visualisation's exploration for an interpretation can be guided by looking for some of the following characteristics within the visualisation:

  – Find smallest/highest value objects with respect to one metric: These objects, which are easily detectable within the visualisation – because the size can be chosen in relation to one metric – are anomalous because they have the smallest/highest metric values.
  – Find clusters of objects: If some displayed objects are close to each other this cluster can be interpreted with respect to the chosen distance concept (cf. 2.2). For example a good design of a cohesive subsystem would yield a visualisation with many closely arranged classes, which all have the same colour, and a few facade classes providing the subsystem's functionality to outside. These facade classes would be arranged between the subsystem and its clients.
  – Find "anti-clusters" of objects: The opposite of the previous step is also interesting, i.e. to find objects (or groups of objects) that are separated from each other. According to the chosen cohesion concept this can give valuable insights into the system for the quality assessment (e.g. dead classes or widely spread subsystems that are only containers of classes with no meaningful semantic glue between them).

  For a more detailed description on how to do a quality assessment using this kind of visualisation, e.g. by detecting "visual anti-pattern", refer to [SiStLe01].

3 *Assessment reflection*: Both the results of the assessment preparation and the results of the assessment execution are stored in a report. All the interpretations are only based on knowledge that can be extracted from the measurement values, i.e. the persons doing the quality assessment do not have any detailed semantic knowledge about the system. For an external validation of the used metrics it is important to get developers feedback on the quality assessment, i.e. do they agree or disagree with the interpretations.

The results of the assessment reflection step should be taken to improve a next cycle of this process, i.e. the quality profile can be adjusted to special quality goals, the selection of software parts important for the quality can be refined, the interpretation can be made sensible to special design conventions etc.

3 Case study: JWAM

The case study described in this paper was done during the first months of 2000. The SE group of the computer science department of the University Hamburg provides a Java framework (cf. Section 3.1) which was to assess with respect to quality by external experts (the software system engineering group at the Technical University of Cottbus). The framework and the used measurement environment are briefly explained in the following subsections.

3.1 Characterisation of JWAM

JWAM is a Java framework supporting the development of large scale interactive software systems according to the tools & materials approach[1]. The foundations for the JWAM framework (see [JWAM]) were laid years ago by students at the University of Hamburg's software engineering department in their diploma theses. At the end of 1999, a spin-off company, APCON Workplace Solutions, was founded to ensure the framework's commercial utilisation.

Today, the JWAM framework consists of more than 1000 classes, a small kernel and several additional components. It is used within a number of professional software development projects by the company itself as well as by licensees. APCON Workplace Solutions has initiated the development of commercially relevant components like host-system adaptation or ERP-system integration.

The quality of a framework is the crucial factor in framework development. As a framework will be reused by a large number of application projects, errors and inappropriate functionality multiply. In addition, a framework provides the general architecture and design for applications. A lack of quality here is detrimental to the application system's architecture as well.

To ensure top quality of the JWAM-framework, a number of construction techniques are applied. These techniques ensured the system's high quality and helped to avoid the old mistake of separating construction from quality management. The techniques employed (many of them part of eXtreme programming, see [Beck99]) are:

- *Pair Programming*: This part of the eXtreme programming paradigm is used exclusively for the framework kernel. Other parts of the framework may be realised by a single developer, but kernel components are invariably integrated into the framework by a pair of programmers.
- *Design by Contract*: The well-known design paradigm by Bertrand Meyer [Meye97] ensures the quality of features by means of pre- and postconditions. The complete framework uses this paradigm, supported by a simple Contract class in the language extension layer.
- *Unit Testing using JUnit*: Currently 95% of all classes within the JWAM framework kernel have their own test class. The JWAM developers have extended JUnit by components that allow them to test the GUI and event handling of tools. In total, the JWAM framework contains about 200 test classes.
- *Aggressive Refactoring*: These refactorings are done in so-called JWAM Sprints, during which the JWAM developers concentrate all their resources on the framework for 3 consecutive days.

3.2 Actual parameters for a metrics-based quality assessment

The process explained in Section 2 was applied twice: One time for the JWAM 1.4 (cf. Section 4.1) and the second time for the JWAM 1.5 (cf. Section 4.2). In both versions the following parameters were used:

- *Assessment preparation*: For the measurement of source code the measurement group has developed a fully customisable metrics engine, called *Crocodile* (cf. [LeLöSi99]). For the first walkthrough, a default quality model with a default metrics set developed by the measurement group was used (cf. Section 4.1). The feedback of the first report was used to define an optimised quality model (with a modified metrics set) for the second walkthrough (cf. Section 4.2).

  The resources available for one assessment walkthrough were: 2 persons with together 25 hours within 2 weeks. The evaluation of the first walkthrough and the definition of an optimised quality model took additional 15 hours.
- *Assessment execution*: The calculation of the measurement data was done by Crocodile and took nearly 1 hour for the whole project[2]. For the three dimensional exploration of the data, a standard VRML-browser was used to simplify the exploration and navigation of the measurement values. The assessment itself took about half of the time (12 hours). The completion of the report, which consists of the following parts, took the other half (13 hours):
  - Short explanation of the goal of the report and the used techniques to create it,
  - short quantitative overview about the system,
  - for every metric:
    - a short explanation (e.g. its domain, range, scale, etc.),
    - an overview about the depending quality goals (according to the quality model),

[1] WAM is the German acronym for tools, automatons, materials. More information about WAM can be found in [RiZü95] and [Zuel98]. The framework can be downloaded from [JWAM].

[2] The calculation was done on a Windows® NT computer with 128MB Ram and a 600MHz Pentium III

  - a distribution diagram showing the distribution of all values of this metric for the whole project,
  - a list of measured objects with the 10 highest and lowest measurement values,
  - a list of quality impressions with respect to the metric and its dependencies within the quality model and
  - a list of restructuring recommendations, i.e. quality impressions including one possible solution to improve the system's quality with respect to the measured property.
- *Assessment reflection*: For the reflection of the single impressions and restructuring recommendations within the report a questionnaire was filled out by the *JWAM Architecture Group*: This group of eight framework architects meets weekly and discusses strategic concerns of the version-in-progress as well as construction details. With its members having both conceptual and detail knowledge of the JWAM Framework, the Architecture Group was the natural forum for discussing the metrics-based quality assessment that had been done for the framework. Every single impression/restructuring recommendation should be assessed by this group on a 5 step scale: "Completely right", "mostly right", "indifferent", "mostly wrong", "completely wrong". The both ratings "completely right" and "mostly right" had to be refined with respect to the degree of novelty into one of the following classes: "completely new", "confirming rough knowledge" and "well known".

4 External validation of the JWAM quality assessment

To prevent random correlations, the measurement group applied the assessment-evaluation-cycle twice. Thus, this work is compatible to the external validation process introduced by Schneidewind and its guideline for the validation of software metrics ([Schn92], [IEEE1061]). The result of the first walkthrough is described in Section 4.1. Its feedback was used to optimise some of the measurement parameters for the second walkthrough that is described within Section 4.2.

4.1 First process walkthrough: Quality assessment of JWAM 1.4

The first walkthrough was done for the JWAM release version 1.4. The three tasks of the assessment, i.e. preparation, execution and reflection were applied in the following manner:

- *Assessment preparation*: Because JWAM is public domain software the measurement group downloaded the version 1.4 from the official software server. While extracting the relevant data for the Crocodile tool the following system size metrics were identified: JWAM 1.4. has 100 subsystems, 922 files, 1287 classes, 655 inheritance relations, 7818 methods, 2235 attributes, 11039 call-relations between methods and 5656 use-relations between methods and attributes. Because there was no additional knowledge the complete JWAM was measured, i.e. all software components (e.g. including all test class, cf. Section 3.1) were selected for the quality assessment.
  For the first walkthrough a default quality model for large JAVA-projects coming from own experience and literature research was applied. The 18 metrics at its leafs can be grouped as follows:
  - *Size-metrics*: In correspondence to the later visualisation and its abstraction levels (cf. Section 2) size metrics were defined on different levels of the containment-based hierarchy: subsystems→files→classes→methods/attributes. On each level it is possible to define metrics counting elements of levels below. The following 9 size metrics were defined:
    - For every subsystem: *number of classes*.
    - For every file: *number of lines*, *number of delimiter* to approximate number of commands, *number of commands modifying the control flow* to approximate cyclometric complexity and *number of comment lines*.
    - For every class: *number of public methods*, *number of public attributes*, *number of overridden methods* and *number of characters needed to implement all methods of one class.*
  - *Coupling-metrics*: Two kinds of coupling are considered: *Interaction coupling* as defined by Stevens et al. ([StMyCo74]), i.e. the coupling by direct usage or calling, and *inheritance coupling* ([EdKaSc93]), i.e. one component directly or indirectly inherits from another-one. These couplings can be extended to every level above the level where they are defined, i.e. a subsystem X has interaction coupling to another subsystem Y, if X has got a class containing a method that uses a method or attribute of another class that is defined within subsystem Y. The following 9 coupling metrics were defined:

- For every subsystem: *number of subsystems a subsystem has incoming/outgoing interaction couplings from/to*.
- For every class: *number of classes a class has incoming/outgoing interaction couplings from/ to*, *number of classes a class has outgoing inheritance couplings to*.
- For every method: *number of methods a method has incoming/outgoing interaction couplings from/to*.
- For every attribute: *number of methods an attribute has incoming interaction couplings from*. One time only those methods are considered that are defined within the same class like the attribute and one time only foreign methods are considered.

Additionally, the usage-based and inheritance-based distances for the later visualisation were calculated (cf. Section 2).

- *Assessment execution*: This step produced a 43 page report with 27 quality impressions and 28 restructuring recommendations. In the following two examples are given (one impression and one recommendation):
  - Impression 28: Nearly all classes offer a moderate number of public functionality. This supports the impression that the classes are created by a well done decomposition of the domain problem. The moderate size simplifies the understanding and usage of classes.
  - Restructuring recommendation 31: The class `ConfigurationStandard` contains 18 public attributes; this is a strong violation of the principle of data encapsulation. Additionally, these attributes are only used by methods of `ConfigurationStandard`, so their visibility should be reduced to protected. The same hint holds for some attributes of the class `ClassUseMapper`.
- *Assessment reflection*: The evaluation of the questionnaire considering the degree to which the impressions and recommendations match the knowledge of the JWAM Architecture Group yielded the following data:

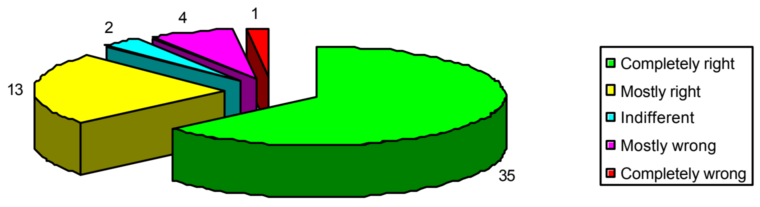

Fig. 2: Assessment of the report of JWAM 1.4

The result of this first walkthrough, in which 87% of the metrics-based impressions and recommendations match the detailed programmers' knowledge, was very promising. Most of the statements that were regarded as mostly or completely wrong did in fact comment on interesting anomalies in the framework while drawing false or too extreme conclusions. One example is restructuring recommendation no. 16 that commented on the fact that the JWAM Framework contains several method-free classes named `TestHook` in packages otherwise class-empty. The purpose of those classes is to be 'hooks' for the testing tool in use that takes a class and runs all unit tests for the classes in the same package and all packages below (in the hierarchy). Thus, the `TestHook` classes allow empty packages to be starting points for the traversal by the test tool. The restructuring recommendation no. 16 suggested to replace all but one of the `TestHook` classes by textual references to the one remaining class. Whereas the existence of the empty `TestHook` classes certainly was odd and may have implied to modify or replace the testing tool, the hint given in the assessment report clearly missed the point and was therefore considered to be *mostly wrong*. Nevertheless, even the statements rated as mostly or completely wrong were interesting inputs for the Architecture group.
Apart from the question whether some recommendations were considered more or less correct, it is interesting to learn to which degree the insights gained from the report were new to the Architecture Group. With respect to the degree of novelty the 48 completely and mostly right recommendations were rated as follows: 2 as completely new, 9 as confirming rough knowledge and 37 of them were well known.

This leads to the conclusion that nearly one quarter of the correct impressions and recommendations were at least valuable reminders for the group that there was something to do or to overthink. Some possible reasons for the other three quarters, i.e. impressions and recommendations that were well known, are given in Section 4.3.

4.2 Second optimised walkthrough: Quality assessment of JWAM 1.5

The evaluation of the filled feedback questionnaire and some discussions via e-mail were used to adjust the measurement environment (assessment preparation) and the way of interpretation (assessment execution). On the one hand the modifications had to be compatible with the wanted empirical validation of the metrics and the used process, i.e. the used metrics and the applied process should be similar. On the other hand the modifications should improve the report's quality, i.e. more impressions and restructuring recommendations should be ranked as correct.

The three tasks of the assessment were applied as for JWAM 1.4 (cf. Section 4.1):

- *Assessment preparation*: The results of the first reflection motivated to set a *measurement focus*, i.e. to select particular packages and classes for the further measurement process, e.g. some example packages and classes were excluded because they do not have a direct impact on the JWAM quality. The system size metrics of the reduced system are (numbers in brackets show the values for the complete JWAM 1.5 to allow a comparison with JWAM 1.4): 72 (93) subsystems, 787 (963) files, 1159 (1456) classes, 1054 (1297) inheritance relations, 6746 (7953) methods, 1883 (2546) attributes, 14738 (16674) call-relations between methods and 5223 (5722) use-relations between methods and attributes.

  The major modification of the second process walkthrough covered the consideration of inheritance and its possible impacts on the used metrics (cf. [SiBeLe01]). Both, the questionnaire's evaluation and the discussions showed that the concept of inheritance was not well examined. For the second walkthrough many metrics were applied in a different way: By setting all classes having subclasses into a so-called *inheritance-context* (cf. [SiBeLe01]) all subclasses include inherited attributes and methods; additionally, all couplings of the inherited class members are inserted into the subclass. Furthermore, polymorph calling structures, which may be modelled as coupling to interface(s), are extended to *potential couplings* to all classes implementing the interface. For calculation and interpretation of the metrics values both views on the system are relevant: The values of the default view and the values of the inheritance considering view (including the differences between them). This second view was additionally used for applying the following metrics:
  - all size-metrics on the class-level
  - all coupling-metrics on the subsystem and class-level

  Like in the first walkthrough the usage-based and inheritance based distances for the later visualisations were calculated (cf. Section 2).
- *Assessment execution*: This step produced a 45 page report with 37 quality impressions and 32 restructuring recommendations. From the point of view of the measurement group the non-quantifiable impression regarding the comparison between both reports allows for the following classification of the single statements within JWAM 1.5 and JWAM 1.4:
  - Many impressions and recommendations that showed quality weakness and that were ranked as completely right disappeared because either the component was deleted or modified.
  - Because of the improvement of many quality weaknesses detected in version 1.4 some new impressions and recommendations that didn't attract the attention within the first process walkthrough could be identified because they moved up and were most anomalous now (e.g. they had the highest metric values).
  - Some new impressions and recommendations were based on the additional view on the system.
  - A few impressions and recommendations ranked as right could be identified again.
- *Assessment reflection*: Like for the JWAM 1.4 report a filled questionnaire considering the degree to which the impressions and recommendations match the knowledge of the JWAM Architecture Group was requested. The results are:

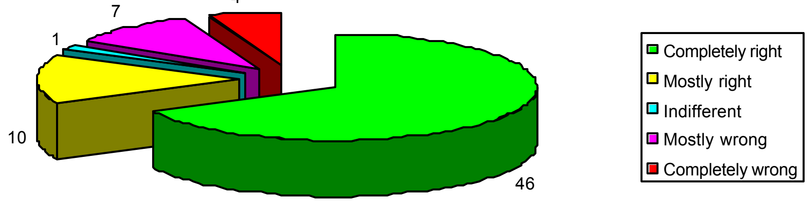

Fig. 3: Assessment of the report of JWAM 1.5

One impression given by the measurement group was not understood and thus could not be ranked. However, about 82% of the 68 metrics-based impressions and recommendations matches the programmers knowledge. All four statements ranked as completely wrong were restructuring recommendations. They all covered the special way inheritance is used in JWAM 1.5: It is a major goal to separate the interface with the implementation and to put both parts into separate subsystems. Due to the introduction of potential couplings many subsystems and classes had much more couplings to other subsystems and classes than in the default view. This special kind of design convention was not considered for the adjustment of the quality model.

With respect to the degrees of novelty a similar questionnaire had to be filled. In contrast to the first one a new ranking level had to be introduced: "Known from previous report", i.e. impressions or recommendations from the report JWAM 1.4 that are still valid in JWAM 1.5. The ratings are as follows: 1 as completely new, 11 as confirming rough knowledge, 40 as well known and 4 of them were known from the previous report.

4.3 Lessons learned

This Section summarises what has been learned during the validation from both involved sides:

4.3.1 The JWAM-Team:

They way of viewing the software system was totally new for the JWAM Architecture Group. They knew the general possibilities provided by metrics but they didn't use any of them before. So the assessment presented in this paper provided a totally new way of analysing the framework. Especially the first walkthrough has given valuable information. Some of them led to refactorings which improved the quality of the system. Other metrics presented interesting information about the size and the structure of the framework, which are also very interesting for the development group but did not lead to improvements of the code.

Important information derived from this experiment is:

- *Avoiding redundancy*: To make a series of assessments as interesting and valuable as the first one each assessment should be modified taking into account the previous results and comments. If one assessment led to a wrong result the next walkthrough should not contain the same (wrong) recommendation. Otherwise future assessments will contain less and less interesting facts.
- *Essence filtering*: The JWAM team draws the conclusion that the assessment report should be "filtered" by one or two of the developers of the framework for a concentrated presentation with the most important facts from the report. These should then be discussed in the Architecture Group. The decisions made should be put down on especially marked story cards that refer to the assessment report.
- *Lifecycle integration*: Another conclusion concerns the point in time when such an assessment is most productive. In the view of the JWAM team, the metrics-based quality assessment should take place some time (one month) before the release of a new version. This ensures that the main weaknesses that can be found based on the assessment report can be considered for refactorings that help to provide the best release version possible. Compared to this, a quality assessment of a newly released version is not as helpful, because some of the anomalies found will refer to parts of the framework that have to be changed anyway due to functional requirements.

The assessment also showed that the JWAM team has a relatively good knowledge base about their own system. Many of the recommendations and detected anomalies were well known by the Architecture Group of the framework. The lived eXtreme programming paradigm, especially the collective ownership idea of XP, is the major reason for this solid knowledge about the framework.

4.3.2 The measurement group:

The experiment described in this paper was the first external validation of the quality assessment process and the used metrics. It confirmed that the metrics based approach allows efficient and detailed quality assessments of large OO-systems. The specific conclusions derived from this experiment are:

- *Scalability*: The assessment process developed by the measurement group leads to specific quality impressions and restructuring recommendations even for medium and large sized OO-systems. Despite the systems complexity, the measurement group was able to locate very detailed anomalies of the system and to get an impression about the software's structure in shortest time (25h, cf. Section 3.2).

- *Correctness*: The found impressions and recommendations consider only the technical software engineering viewpoint. Since the development team agreed to more than 80% of the statements in both walkthroughs, it can be concluded that metrics allow for an efficient and correct quality assessment. However, the assessment group does not have any semantic information about the system, which sometimes led to statements that are basically worth discussing but which, on the other hand, were well known to and accepted or even intended by the development team. To keep the process cost-effective the measurement group is not primarily interested in embedding anomalies of the software system into its semantic context.
- *Efficiency*: Since the knowledge about large systems usually is limited, e.g. there might be quality anomalies that are not known neither by the development team nor by any external assessment group, statements about the efficiency of this quality assessment process are difficult to make. Considering it as *diagnostic test* the following qualitative assessment might be possible:
  - *Sensitivity*: Within the assessment phase the JWAM-group was asked for quality anomalies that the measurement group did not detect. After the first process walkthrough this feedback was used for the extension of the used quality model. After the second process walkthrough all quality anomalies known to the developers were identified (well known or just confirming rough knowledge). So the number of *false negative errors* is low and the sensitivity high.
  - *Specificity*: Most discrepancies between the assessments and the developer knowledge are *false positive*, i.e. quality anomalies identified by the assessment group were rejected by the developers. Like in medical diagnostic this kind of errors is less severe than *false positive*. The high number of *true positive* impressions and recommendations suggest a high specificity.
- *Co-operation*: To derive as much interesting information as possible from the quality assessment, continuous communication between the development team and the assessment group is necessary. Thus, to avoid statements that are based on technical engineering knowledge but that do not contribute to the software's internal quality from the development team's viewpoint, it is important to define quality models depending on specific quality criteria (cf. [FePf96]) and to exclude parts of the system that are not interesting to the development team (e.g. Classes `TestHook`, cf. 4.1).

5 Summary / Outlook

The successful external validation of the metrics based quality assessment, i.e. the demonstration of a consistent relationship between some metrics and some available empirical data is demonstrated in this case study. A well-defined process for a metrics based quality assessment was defined and applied for two different versions of the JWAM-framework. To show the relationships the JWAM-team had to judge about every quality impression and restructuring recommendation that was part of the assessment report. In both cases most impressions and recommendations were ranked as right (>80%). Since the quality assessment is based only on software metrics it is confirmed that metrics are a powerful support in the area of software quality examination, i.e. there exists a relationship between the metrics we used and the empirical data known to the JWAM-engineers. The described process to use a metrics-based quality assessment of large object-oriented systems was applied to two more industrial projects and produced similar results. The reports containing the assessment provided new information about the system as well as reminded the developers of weaknesses of the framework that were known to some developers but probably suppressed from their memory. The relatively low degree of novelty can be explained by the special environment of the JWAM-team; the assumption that the use of eXtreme Programming guarantees the high dissemination of detailed knowledge about the system has to be examined in further experiments. Another interesting point is to examine the qualitative evolution of a system: Currently we analyse additional JWAM-versions before the versions described in this paper to allow a long time trend analysis, which allows statements about the evolution of the system (and its subsystems/classes etc.) regarding its quality. If the corresponding trends can be mapped onto different engineering techniques, which for example is the case for the JWAM-version 1.3 that was the first version developed using eXtreme programming, some interesting hypotheses about the correlation between used technique and quality of its output can be tested.

6 References

[Beck99] Kent Beck: *"Extreme Programming - Embrace Change"*, Addison-Wesley, 1999

[BeLe79] L.A. Belady, M.M. Lehman: *"Characteristics of large systems"*, in "Research Directions in Software Technology", P.Wegner (editor), MIT-Press, Cambridge, Massachusetts, 1979

[EdKaSc93] J. Eder, G. Kappel, M. Schrefl: *"Coupling and Cohesion in Object-Oriented Systems"*, Technical Report of University Klagenfurt, Institute of Computer Science, 1993

[FePf96] Norman Fenton, Shari Lawrence Pfleeger: *"Software Metrics: A Rigorous and Practical Approach"*, Thomson Computer Press, second edition, Cambridge, 1996

[GhJaMa91] Carlo Ghezzi, Mehdi Jazayeri, Dino Mandrioli: *"Fundamentals of Software Engineering"*, Prentice-Hall International, London, 1991

[ISO9241] DIN EN ISO 9241-10, *"Ergonomic requirements for office work with visual display terminals (VDTs) – Part 10: Dialogue principles"*, Beuth-Verlag, Berlin, 1996

[ISO9126] ISO/IEC 9126, *"Information technology – Software product evaluation – Quality characteristics and guidelines for their use"*, Beuth-Verlag, Berlin, 1991

[IEEE1061] IEEE Standard *„Software Quality Metrics Methodology"*, IEEE Std. 1061-1992, New York, 1993.

[JWAM] The JWAM-Framework web-site: http://www.jwam.org

[KöRuSi98] Gerd Köhler, Heinrich Rust, Frank Simon: *"An Assessment of large object oriented Software Systems: A metrics based process"*, in proceedings of the Object-Oriented Product Metrics for Software Quality Assessment Workshop on 12th European Conference on object-oriented programming, CRIM Montreal, p 16-23, 1998

[LeLöSi99] Claus Lewerentz, Silvio Löffler, Frank Simon: *"Distance based cohesion measuring"*, in proceedings of the 2nd European Software Measurement Conference (FESMA) 99, Technologisch Instituut Amsterdam, 1999

[LeSiSt00] Claus Lewerentz, Frank Simon, Frank Steinbrückner: *"Multidimensionale Mess- und Strukturbasierte Softwarevisualisierung"*, published in proceedings of 2th workshop "Reengineering" in Bad Honnef, Fachberichte Informatik 8/2000, pp. 47-50, University Koblenz-Landau, 2000

[Meye97] Bertrand Meyer: *"Object-Oriented Software Construction"*, New York, London: Prentice Hall, Second Edition, 1997.

[PuSoTr97] Teade Punter, Rini van Solingen, Jos Trienekens: *„Software Product Evaluation"*, in Proceedings of 4th European Conference on Evaluation of Information Technology (EVIT'97), Delft, 1997

[RiZü95] Dirk Riehle, Heinz Züllighoven: *„A Pattern Language for Tool Construction and Integration Based on the Tools&Materials Metaphor"*, in: J.O. Coplien, D.C. Schmidt (eds.): „Pattern Languages of Program Design", Chapter 2, pp. 9-42, Addison-Wesley, Massachusetts, 1995

[Schn92] Norman F. Schneidewind: *"Methodology for validating software metrics"*, in Transactions on Software Engineering, Vol. 18, No. 5, pp. 410-422, 1992

[SiBeLe01] Frank Simon, Dirk Beyer, Claus Lewerentz: *"Impact of Inheritance on Metrics for Size, Coupling, and Cohesion in Object Oriented Systems"*, in Dumke, Abran (Eds): "New Approaches in Software Measurement", Lecture Notes on Computer Science 2006, pp. 1-17, Springer-Verlag, 2001

[SiStLe01] Frank Simon, Frank Steinbrückner, Claus Lewerentz: *"Metrics Based Refactoring"*, in Proceedings of the 5th European Conference on Software Maintenance and Reengineering (CSMR2001), pages 30-38, IEEE Computer Socienty Press, 2001

[StMyCo74] W.P. Stevens, G.J. Myers, L.L. Constantine: *"Structured Design"*, in IBM Systems Journal, Vol. 13, No. 2, 1974

[Zuel98] Heinz Züllighoven: *"Das objektorientierte Konstruktionshandbuch nach dem Werkzeug- & Material-Ansatz"*, dPunkt-Verlag, Heidelberg, 1998

[Zuse98] Horst Zuse: *„A Framework of Software Measurement"*, Walter de Gruyter, Berlin, 1998