ISSN (Online) 2249 - 054 X

International Journal of Computing and Corporate Research

Specialized and Refereed Journal for
Research Scholars, Academicians, Engineers and Scientists

http://www.ijccr.com

VOLUME 1 ISSUE 3 MANUSCRIPT 6 NOVEMBER 2011

QCR - A METHODOLOGY FOR SOFTWARE EVOLUTION ANALYSIS

Esha Bansal

Research Scholar

Dravidian University, Kuppam

Nisha Bansal

Assistant Professor, CGC Landaran

Mohali

ABSTRACT

In order to support re-engineering efforts we defined a new methodology called QCR for software evolution analysis. QCR is defined as an incremental methodology combining three complementary steps of analysis, where the results of each step are used as input to the next. First the Quantitative Analysis (QA) is utilized to give an overview of the entire system and to point out potential architectural insufficiencies within the software system. The results of this first step are taken as

ISSN (Online) 2249 - 054 X
International Journal of Computing and Corporate Research
Specialized and Refereed Journal for Research Scholars, Academicians, Engineers and Scientists
http://www.ijccr.com
VOLUME 1 ISSUE 3 MANUSCRIPT 6 NOVEMBER 2011

input to the further steps. The Change Sequence Analysis (CSA) helps to broaden the knowledge base. It supports the reasoning about the structural weaknesses discovered in the first analysis step. Additionally, CSA provides hints for new testimonies about the architecture of the entire system. As last step the Relation Analysis (RA) provides deeper insight into the structure of the software. This step, which was developed in the research of this thesis, improves the findings of the other steps. Despite the huge quantity of the logical couplings discovered with the help of RA, the graphical representation simplifies the assessment of the software architecture. The RA method allows receiving an overall notion about the evolutionary dependencies between different parts of the entire software system. It describes how the parts of the application are positioned to each other and which part has connections to which other parts.

KEYWORDS

QA- Quantitative Analysis

CSA- Change Sequence Analysis

RA- Relation Analysis

QCR- Quantitative, Change Sequence, Relation Analysis

Size- The size of each system, subsystem and module is defined as the number of classes it contains.

Growing rate- The growing rate is defined as the percentage of newly added classes.

QCR reveals problematic areas of the software for further maintenance;

ISSN (Online) 2249 - 054 X
International Journal of Computing and Corporate Research
Specialized and Refereed Journal for Research Scholars, Academicians, Engineers and Scientists
http://www.ijccr.com
VOLUME 1 ISSUE 3 MANUSCRIPT 6 NOVEMBER 2011

supports the understanding of the architecture, and improves evolvability for the satisfaction of new requirements. The larger a software product becomes the more important is the architecture of the entire system in order to support evolvability. Focusing on the analysis of software evolution has revealed a high potential for improvement of the software development process and, as a result, the software itself. Software evolution refers to the dynamic behavior of software systems as they are maintained and enhanced over their lifetimes. Software evolution is very important as systems become longer- lived. However, evolution is challenging to study due to the longitudinal nature of the phenomenon in addition to the usual difficulties in collecting empirical data [Kemerer and Slaughter, 1999].

First the concept of software evolution is introduced. This part provides also a definition of the term “software evolution.” A system has to exhibit certain characteristics in order to be maintainable. Therefore, we summarize a paper concerning evolvability, which requires other attributes such as changeability and testability. All these factors contribute to the quality of software. The concept of software architecture seems to be an appropriate concept for the evaluation of software quality. The architecture of a software system describes the elements and its communication paths. Hence, the dependencies between classes should be considered for the sake of maintainability. Metrics are often used for the assessment of complexity. In particular, code based metrics such as cohesion and coupling provide measurements of a system’s structural complexity.

This methodology – called QCR – is based on the historical data of a software system. The evolution of the studied software system was analyzed on the level of

ISSN (Online) 2249 - 054 X
International Journal of Computing and Corporate Research
Specialized and Refereed Journal for Research Scholars, Academicians, Engineers and Scientists
http://www.ijccr.com
VOLUME 1 ISSUE 3 MANUSCRIPT 6 NOVEMBER 2011

classes. The research on software engineering of the last few years suggests that not only the source code with its number of lines of code provides enough information about the complexity of a system, but we have to investigate on modules and programs for the measurement of software systems. Accordingly, classes as basic building blocks of object oriented systems provide a good decomposition level for the assessment of the size and evolution of a system. Additionally, this level can be used to evaluate functional enhancements.

Our methodology, QCR, investigates the historical development of classes. The time when new classes are added to the system and when the existing classes are changed has to be measured. Attributes related with changes of classes like the author of a change are an additional input for the software evolution analysis of this thesis. Changes applied to the classes of the studied software system were inspected in order to reveal common change behavior. The same change behavior of different parts of the system during the evolution is referred to as logical coupling. Through the assessment of classes we can evaluate modules or even the entire system, as they build up a hierarchical organization of classes. Thus, the software system as a whole can be analyzed, but also the subparts of the system may be regarded and related to each other.

All three techniques are examined during the evolution of the software within the inspection period. This period is divided into its twenty eight months and then each month can be compared with its successor. A short overview of the three methods is given, whereas the definition in greater detail can be found within the sections concerning each step:

ISSN (Online) 2249 - 054 X
International Journal of Computing and Corporate Research
Specialized and Refereed Journal for Research Scholars, Academicians, Engineers and Scientists
http://www.ijccr.com
VOLUME 1 ISSUE 3 MANUSCRIPT 6 NOVEMBER 2011

- *Quantitative Analysis*: The first technique of QCR is based on metrics such as the size and change rate of different parts of the software. These metrics are computed for the entire system and all of its subparts. As a result we can estimate whether the system reached a stable state, or whether improvements and functional enhancements dominate the evolution of the entire software system. Then we zoom in on the subparts in order to discover outliers that show a different historical development then the entire system. Such outlier point to structural flaws. As the results of the Quantitative Analysis are used as input to the other methods, we may get a deeper in- sight into different aspects of the software system.

- *Change Sequence Analysis*: The Change Sequence Analysis (CSA) identifies common change patterns. Change patterns refer to the same change behavior of different parts of the software during system evolution. Each change of a class is represented on system level, which allows comparison of classes between each other. The changes of a particular class are assembled into a change sequence. Based on these change sequences dependencies between classes and its modules may be recovered.

- *Relation Analysis*: The Relation Analysis (RA) is a technique that examines the data related with a particular change like the change description or author. RA enables a verification of the change patterns of CSA and the results of the Quantitative Analysis. Detecting similarities within the change behav-

ISSN (Online) 2249 - 054 X
International Journal of Computing and Corporate Research
Specialized and Refereed Journal for Research Scholars, Academicians, Engineers and Scientists
http://www.ijccr.com
VOLUME 1 ISSUE 3 MANUSCRIPT 6 NOVEMBER 2011

ior, like referring to the same author or the same change time, can confirm the logical dependency discovered with the help of CSA. Additionally, new logical coupling may be detected when inspecting the information related with single changes.

Quantitative Analysis (QA)

The first step in the methodology applied on the case study is a Quantitative Analysis (QA). This analysis step utilizes metrics like growth rates and change values in order to find potential failings within a software system. The metrics were computed at the scope of the whole application and for all its subparts. Thus, this approach provides a high level view of the system and allows zooming in on recovered weaknesses of the analyzed software.

Approach

The goal of the quantitative software evolution analysis is to recover potential shortcomings by tracking the historical development of the software system. Therefore outliers in the evolution of the system have to be identified.

The following part will give definitions for the properties of the system, which were used for the QA method. It is necessary to mention that the research of the last few years suggests that not the source code with its number of lines of code provide sufficient information about the complexity of a software system. Instead we have to investigate the number of modules, programs or classes (in object oriented systems) for complexity measurements. With increasing size of the analyzed system,

ISSN (Online) 2249 - 054 X
International Journal of Computing and Corporate Research
Specialized and Refereed Journal for Research Scholars, Academicians, Engineers and Scientists
http://www.ijccr.com
VOLUME 1 ISSUE 3 MANUSCRIPT 6 NOVEMBER 2011

this statement becomes even more relevant. As the aim of this thesis was to improve the evolvability of the software system, the analysis concentrated only on source code files and neglected others, such as files containing icons or configuration information. The following metrics take only files into account, which contain source code.

The interval between two measurements was chosen to be one month which is quite a fine granularity on time and helps to identify even small evolutionary outliers. The time line of the case study was divided into separate periods, where the length of a period was chosen to be one month.

The quantitative software evolution analysis has the following attributes:

- Easily computable: As the Quantitative Analysis utilizes simple metrics in order to discover outliers within the entire application, the application of the QA method does not bear great difficulties. Thus, it should be easy to execute the Quantitative Analysis on many different software systems.

- Scalable: Measures like size, growing rate, and changing rate do not demand high computational power. Furthermore, the amount of data during the computation of the appropriate values is quite low. Thus, the Quantitative Analysis is well scalable on large software packages.

- Practicable on different levels: The QA method is practicable on all levels of decomposition. As a result it is possible to zoom in on smaller units like

ISSN (Online) 2249 - 054 X
International Journal of Computing and Corporate Research
Specialized and Refereed Journal for Research Scholars, Academicians, Engineers and Scientists
http://www.ijccr.com
VOLUME 1 ISSUE 3 MANUSCRIPT 6 NOVEMBER 2011

modules and submodules.

- Outliers can be easily spotted: As the computed metrics can be represented as single numerical values, the different parts of software can be compared with each other. Additionally, the detection of outliers can be automated to allow human users to concentrate on the interpretation of the results.

- Progression of values as support: The used metrics may be computed on different time intervals. Thus, a progression of values may be calculated in order to represent the evolution of an analyzed part. Such progressions provide additional hints for the interpretation of the results.

- Overview: The Quantitative Analysis provides a good overview of the entire system by calculating metrics for all classes of the system. Subunits like subsystems may also be measured. Thus, an overall picture may be drawn, which enhances the global understanding of the system.

- Additional analysis: With the help of the Quantitative Analysis we receive a global understanding of the architecture. Nevertheless, additional analysis steps would be helpful to get a deeper insight of the software. The detection of logical coupling supports the reasoning of structural failings.

- Domain knowledge: In order to get significant results and explanations for structural weaknesses the integration of domain knowledge is essential.

ISSN (Online) 2249 - 054 X
International Journal of Computing and Corporate Research
Specialized and Refereed Journal for Research Scholars, Academicians, Engineers and Scientists
http://www.ijccr.com
VOLUME 1 ISSUE 3 MANUSCRIPT 6 NOVEMBER 2011

Change Sequence Analysis (CSA)

The Change Sequence Analysis reveals dependencies between different parts of the analyzed software system by comparing the change evolution of the system blocks. The changes done on each part of the software system are arranged in sequences by the date of the changes. Thus, entities that changed in the same months of the evaluation period are related to each other.

Approach

In order to compare changes of different classes each change of a class is represented on the system level. Similar to the Quantitative Analysis (QA) the Change Sequence Analysis (CSA) divides the evaluation period into single months. Thus, on system level months are enumerated with sequential numbers. The changes of all classes are projected on the system level. The projected changes of a single class are gathered into a change sequence. As a result each class is represented as a change sequence that shows the months of the evaluation period in which the class was changed. For example if a class has changed in the first, second and twentieth month of the evaluation period, the change sequence would contain these three months: [1, 2, 20].After building the change sequences of all classes, these change sequences are compared with each other in order to find dependencies between parts of the software system. We search for change sequences of classes that are equal with change sequences of other classes. Thus, if we consider the previous example where a class has a change sequence [1, 2, 20], a common change sequence <1,2, 20> is established, if another class

ISSN (Online) 2249 - 054 X
International Journal of Computing and Corporate Research
Specialized and Refereed Journal for Research Scholars, Academicians, Engineers and Scientists
http://www.ijccr.com
VOLUME 1 ISSUE 3 MANUSCRIPT 6 NOVEMBER 2011

produces the same change sequence [1, 2,20]. This common change sequence matches the change sequence of both participating classes. When two classes build up a common change sequence we postulate a logical coupling between these classes and the associated parts of the soft ware system. One important aspect of this thesis is the maintainability of the analyzed software system. The performed software evolution analysis should help to improve the evolvability of the software. These considerations had consequences on the computation of the change sequences:

- Only change sequences with more than three months were considered, because shorter sequences do not reveal strong problems during the system maintenance.

- The longer a change sequence is the higher is the probability of a real dependency between the analyzed classes. Therefore, the search for interesting change sequences concentrated on the longer sequences.

Similar to the Quantitative Analysis, for CSA some queries were defined in order to discover architectural limitations of the analyzed software system. The queries were applied on the computed change sequences and their participating classes. The following section describes the supported queries:

- Are there any classes that change in every release?
- What are the longest change sequences?

ISSN (Online) 2249 - 054 X
International Journal of Computing and Corporate Research
Specialized and Refereed Journal for Research Scholars, Academicians, Engineers and Scientists
http://www.ijccr.com
VOLUME 1 ISSUE 3 MANUSCRIPT 6 NOVEMBER 2011

- Which change sequences include the most classes?
- Are there any modules which produce significantly more common sequences then the average?
- Are some modules more often related by common change sequences than the average?
- Which common change sequences include the last few month of the inspection period?
- Are there any classes which show absolutely equal changing behavior?
- Which modules are noticeable in the QA, but may not be found within the common change sequences?
- Which modules were not recognized in the first analysis step, but provide long change sequences?

Observations

The following section describes the observations that could be done during the Change Sequence Analysis (CSA). First the attention is directed towards the results of the Quantitative Analysis (QA). With the help of CSA this thesis refines the discovered high change rates found in the Quantitative Analysis step. Outliers of the changing rate and changing intensity of QA were investigated in more detail based on their change sequences, in order to spot the classes that caused the high values in the first analysis step. Hence CSA utilizes change sequences, only reasons for high changing rates and changing intensities of the Quantitative Analysis may be verified with the help of the CSA method. However, QA brought many out-

ISSN (Online) 2249 - 054 X
International Journal of Computing and Corporate Research
Specialized and Refereed Journal for Research Scholars, Academicians, Engineers and Scientists
http://www.ijccr.com
VOLUME 1 ISSUE 3 MANUSCRIPT 6 NOVEMBER 2011

liers to light with the help of changing measures. The growing rate produced less interesting results.

Lessons learned

- Levels of decomposition: The Change Sequence Analysis indicates logical coupling on many levels of decomposition. Dependencies could be discovered for subsystem as well as for submodules, which are system blocks on a lower level of decomposition. Related system blocks on different levels could be revealed too.

- Domain knowledge: In order to discover real dependencies and to be able to draw conclusions about the structural limitations of the system, the analysis should be supported by human knowledge about the domain of the application.

- Different levels of precision: Identical change sequences reveal very strong dependencies between parts of the system. Common change sequences provide a broader base to reason about the architecture of the studied software. Even weaker dependencies could be measured, when only parts of a change sequence of a system block would be compared with parts of a change sequence of another system block.

- More attributes necessary: The CSA method sometimes leads to false positives. The results can be probably improved by adding additional attributes

ISSN (Online) 2249 - 054 X
International Journal of Computing and Corporate Research
Specialized and Refereed Journal for Research Scholars, Academicians, Engineers and Scientists
http://www.ijccr.com
VOLUME 1 ISSUE 3 MANUSCRIPT 6 NOVEMBER 2011

to the analysis. For example the author of a change may be an indication for relations between different parts of the system.

- Tool support simplifies evaluation dramatically: In order to compare the change sequences of all classes with each other, appropriate programs had to be developed. These programs supplied necessary measures for the evaluation of CSA. A large amount of common change sequences could be discovered.

- Many results within group of subsystems: Especially the group of subsystem was a fruitful source of common change sequences. As a result with the help of the Change Sequence Analysis many logical couplings in conjunction with subsystems could be brought to light.

- Complement to Quantitative Analysis: The results of the Change Sequence Analysis show potential for approaches in order to reveal logical dependencies between parts of the software system. These results help to clarify the picture drawn by the Quantitative Analysis, which utilizes especially growing and changing values. Thus, CSA is a complementary process to QA.

Relation Analysis (RA)

The RA method tries to incorporate more details of changes applied to each piece of software in order to gain improved results of the analysis. Thus, within RA the comparison of changes is not only based on the time of check in, but also on the

ISSN (Online) 2249 - 054 X
International Journal of Computing and Corporate Research
Specialized and Refereed Journal for Research Scholars, Academicians, Engineers and Scientists
http://www.ijccr.com
VOLUME 1 ISSUE 3 MANUSCRIPT 6 NOVEMBER 2011

author that actually carried out the particular change. As we will see with the help of RA we can verify some of the results of CSA and even get more and fine-grained results. The logical coupling between different parts of a system points to structural shortcomings. Especially, the relations between separate modules may be an architectural weakness, because a module should encapsulate particular aspects of a software system.

Approach

While examining the system for similar change patterns the attention was drawn to the modularity of the software system. The evolvability can be preserved or improved with a well formed architecture composed of self-contained software components. Thus, an ideal situation would allow changing each component independently of the others. If changes are necessary, the smallest possible set of components should be involved in a particular change.

In the Relation Analysis (RA) single classes and their historical development are regarded in more detail. The evolution of classes is compared to find the points where classes were changed together. Therefore, the changes of each class are compared with the changes of other classes. This comparison is based on the date and time of the check in of a particular change and the author who carried out the change. The division of the evaluation period into monthly sequences was dropped and each change was considered based on the exact date and the author of the change. This selection was based on the fact that the analyzed software system was developed with *strong ownership of code*. A developer was responsible for a

ISSN (Online) 2249 - 054 X
International Journal of Computing and Corporate Research
Specialized and Refereed Journal for Research Scholars, Academicians, Engineers and Scientists
http://www.ijccr.com
VOLUME 1 ISSUE 3 MANUSCRIPT 6 NOVEMBER 2011

particular part of the software system. Thus, a necessary change for a requested improvement of the software was fulfilled by a single developer. As a result the comparison of the authors of changes should lead to good results. Additionally, within RA the date of each change of a class is compared with the dates of changes of other classes in order to discover equal change dates. Dates are compared on equality, but a time window of four minutes was chosen, because a check in of a large module takes a while, the according files may get different time stamps.

All changes that are done on the same date and by the same author point to a logical coupling. The more such correspondences can be found between a particular group of classes the stronger is the postulated relationship. This logical coupling can be represented as number of common changes, which defines the *strength of the logical connection*.

Lessons learned

The previous parts described the Relation Analysis (RA) together with possible findings and evaluations. These parts reveal interesting logical coupling. Next we are going to outline certain attributes of RA based on the experiences during the application of this new technique on the case study:

- Combines all levels of decomposition: The Relation Analysis is based on the information concerning changes applied to single classes. With these smallest building blocks it is possible to relate parts of the system on different de-

ISSN (Online) 2249 - 054 X
International Journal of Computing and Corporate Research
Specialized and Refereed Journal for Research Scholars, Academicians, Engineers and Scientists
http://www.ijccr.com
VOLUME 1 ISSUE 3 MANUSCRIPT 6 NOVEMBER 2011

composition levels with each other. As the sizes of the different parts on different levels have large deviations it seems promising to compare different levels of decomposition with each other, in order to receive a better insight into the structure of the software system.

- Reveals many couplings: With the help of RA it was possible to find logical dependence. Despite this high number of findings it seems that no false positives could be revealed. Many discoveries help in the architectural reasoning of a broad range of system blocks, which may contain structural weaknesses and should draw the attention to the parts that should be developed carefully.

- Different types of results: The evaluation of the results gives rise to the assumption that many kinds of structural deficiencies may be discovered with the help of RA. Examples of such findings are spaghetti code, bad inheritance hierarchies, and poorly designed interfaces.

- Most findings based on submodules: In the case study most couplings that were discovered with the help of RA were located on submodule level. Based on the strong deviations of size on different levels it is interesting to find submodules, which are related based on their historical development.

- Frequent dependencies between system blocks: RA revealed many internal couplings within the regarded parts of the system. However, often even more

ISSN (Online) 2249 - 054 X
International Journal of Computing and Corporate Research
Specialized and Refereed Journal for Research Scholars, Academicians, Engineers and Scientists
http://www.ijccr.com
VOLUME 1 ISSUE 3 MANUSCRIPT 6 NOVEMBER 2011

external relations could be discovered. Internal links are likely to point out limitations within classes. External couplings are even more interesting, because they may bring to light limitations of the architecture of the entire system.

- Simplifying navigation: Due to the huge base of results as output of the RA method, additional visualization of the findings would improve the navigation to the system blocks of interest. This visualization could be supported by appropriate tools.

- Metrics necessary: In order to spot outliers easily, metrics based on the attributes used within RA are helpful. For the evaluation the number of common check-ins was taken into account. In addition the average number of check-ins within the inspected part of the software is interesting. Other such metrics could be developed to enhance the Relation Analysis.

- Domain knowledge: For the evaluation of the findings resulting form RA the integration of domain knowledge into the analysis technique is essential. By applying RA certain problematic points in the architecture can be found. Then human knowledge has to be incorporated into the method to assess the necessity to revise the discovered part of the software system.

REFERENCES

1. [Bieman and Kang, 1995] J.M. Bieman, B.-K. Kang, “Cohesion and reuse in an

ISSN (Online) 2249 - 054 X

International Journal of Computing and Corporate Research

Specialized and Refereed Journal for
Research Scholars, Academicians, Engineers and Scientists

http://www.ijccr.com

VOLUME 1 ISSUE 3 MANUSCRIPT 6 NOVEMBER 2011

object-oriented system.

2. [Brown et al. 1998] W. J. Brown, R. C. Malveau, H. W. McCormick, T. J. Mowbray, “AntiPatterns: Refactoring Software Architectures and Projects in Crisis,” John Wiley & Sons, March 1998.

3. [Chidamber and Kemerer, 1991] S. R. Chidamber, C. F. Kemerer, “Towards a Metrics Suite for Object Oriented Design,” Proc. Conference Object-Oriented Programming: Systems, Languages, and Applications (OOPSLA’91), October 1991.

4. [Cook et al., 2000] S. Cook, H. Ji, R. Harrison, “Software Evolution and Software Evolvability,” 2000.

5. [Kafura, 1987] D. Kafura, "The use of software complexity metrics in software mainte- nance," IEEE Transitions Software Engineering, vol. 13, pp. 335-343, 1987.

6. [Lehman and Belady, 1985] M. M. Lehman, L. A. Belady, “Program Evolution – Processes of Software Change,” Academic Press, London and New York, 1985.

7. [Pressman, 1992] R. Pressman, “Software Engineering – A Practitioner’s Approach,” McGraw Hill, 1992

8. Rum Baugh, J., et al. Object-Oriented Modelling and Design, Prentice – Hall, 1991.
9. Schaum’s Oulines “ Software Engineering” Tata McGraw Hill Book Company, UK, 1998.

10 K.K. Aggarwak & Yogesh Singh., “Software Engineering”New Age International.