Evolvability as a Quality Attribute of Software Architectures*

Selim Ciraci, Pim van den Broek
Software Engineering Group
Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente
PO Box 217
7500 AE Enschede
The Netherlands
Email: {s.ciraci, pimvdb}@ewi.utwente.nl

***Abstract*— We review the definition of evolvability as it appears on the literature. In particular, the concept of software evolvability is compared with other system quality attributes, such as adaptability, maintainability and modifiability.**

Keywords: Software evolvability, Software evolution, Quality Attributes.

I. Introduction

In recent years, IT industry has faced the problem of evolving their software products in order to stay on the market and to compete with similar products. For software systems to stay on market, they should inherit new requirements and adapt themselves to the changing environment. However, today's marketing trends do not allow the industries to work on changing their products for a long time. Thus the need for designs that can withstand and easily adapt new requirements and changes has emerged, which put the focus on evolvability to be considered as a software quality.

Two trends have been identified to allow software systems to become evolvable: component exchangeability and increase in component distance [2]. Architecting software products has allowed the designers to divide the system in consideration into components. These components exchange messages or request services of other components by means of connections between them. This decoupling of systems at the architecture level reflected itself into detailed design stages like Object Oriented design. In that case, components become classes and connections between components become inheritance, message passing and so on. Besides reducing the understandability of complex systems, this decoupling of components allowed exchangeability. System architects can easily replace a component in the architecture of a system by a better one. In code level design, this change is reflected by changes in the inheritance hierarchy, replacing a class by a newer one and updating dynamically linked libraries [2]. According to [2], development and improvement of networking technologies also contributed to evolvability of software systems. Networking allowed different components to be designed to work at different entities in a networking environment and these components exchanged messages by means of remote procedure calls (RPCs) and sockets. Thus a component can easily be changed or upgraded without affecting other components working with different entities. In summary, decoupling complex systems into components allowed software systems to become evolvable. However, decoupling caused other problems like keeping the same communication interface between components, in order that a component change doesn't affect other components. Studies have shown that evolution and maintenance are the longest and the most expensive phase of software life-cycle. This drew the attention to consider evolvability further in research.

Lehman et al. [8] focus on two different uses of the term evolution; as a noun and as a verb. The first and the largest group of researchers focuses on the question "how" to effectively and reliably evolve a software system; this includes the theories, abstractions, languages and methods. This group is considered to be using the term evolution as a verb. The second group of users ,of the term evolution, uses it as a noun and focuses on the question "what" to investigate and learn properties of software evolution. We think that evolvability research belongs to this group, since it is asking the question "what is evolvable".

In this paper, we present our view of the term evolvability and the operations in software evolution. In the next section the definition of evolvability is presented. Section 3 explains the reason behind considering evolvability as a quality attribute. In section 4, evolvability and some other quality attributes are compared in order to depict where evolvability stands. In the last section, conclusions and some research topics are provided.

II. Definition of Evolvability

In this section, we present the definition used in the literature for the term evolvability and then we present our definition of evolvability which explains our scope for the term. Evolution first appeared in the software engineering literature in 1970s by the study conducted in [3]. In that study

*This work has been carried out as a part of the DARWIN project under the responsibilities of the Embedded Systems Institute. This project is partially supported by the Netherlands Ministry of Economic Affairs under the Bsik program.

the authors have tried to measure the complexity, size, cost and maintenance, using the source code of 20 releases of Os/360 operating system. All measures have shown an increasing trend, which led the authors to compose the five laws of software evolution: Continuing Change, Increasing Complexity, Fundamental Law of Program Evolution, Conservation of Organizational Stability and Conservation of Familiarity.

Since then software evolution has been used to describe the ”long and broad view of change in software systems” [4]. Thus, from this definition of software evolution, the term evolvability is defined as: ”the capability of software products to be evolved to continue to serve its customer in a cost effective way” [4]. Although this definition gives a common ground on evolvability, it doesn't describe the scope of changes that are meant by the term evolution.

To address this problem, we begin constructing our evolvability definition from identifying the changes that cause systems to evolve. Currently, three sources of evolution have been identified [14]:

- Domain: covers the model of the real world considered by the system, i.e., the environment. Any change in that model may force the system change.
- Experience: The users of the system gain experience over time and they may require some suggestions for the system, which in turn may cause the system to evolve.
- Process: includes the organizations and methods that may also impact the system and cause it to change.

Considering these sources, we describe evolution as changes in a system's environment (domain), requirements (experience) and implementation technologies (process). Then we define evolvability as a system's ability to survive changes in its environment, requirements and implementation technologies. It is important to notice here that this definition of evolution modifies the original focus presented in [3] to include the evolution that may occur during the initial development of the system, since these sources of changes may also occur during initial development. For example, during development of the initial system, new implementation technologies may be developed which may cause changes in the requirements of the system.

III. Evolvability as a Quality Attribute

Most of the research on software evolution is focused on analyzing the properties of evolution on the source code level. The majority of these studies try to capture the properties of software evolution by analyzing the changes on the source code in release cycles of software systems. The research on this field has mainly considered size (number of modules) as a principal measure for evolvability [5]. However, studies show that using different metrics may result in different distributions. Kemerer and Slaughter [1] list some of these studies and continue with conducting time, sequence and gamma analysis on two different software systems. An important observation from this study is that these software systems start their evolution cycle with similar activities, such as addition of new modules.

Besides their help on understanding the properties of software evolution, such empirical studies may also provide estimates on the future changes that the source code of a software system is going to face and predict the cost of these changes, such the number of module additions on the next release and the cost of adding these. However, the problem with these estimates is that they do not provide information on how evolvable the initial system is. The system may be designed without considering the changes, so that adding or removing components from it may be very costly. Thus, we believe that the research on evolution should raise the level of abstraction so that systems are designed in a way that they can withstand changes. In other words, evolvability should be a non-functional requirement of a system.

The IEEE 1061 standard [9] defines software quality as the degree to which the software system fulfills a selected group of attribute requirements. In [7] a quality attribute is defined as a non-functional characteristic of a component or a system. Since evolvability is a non-functional requirement of system, it can also be considered to be a characteristic of the system; thus one can conclude that evolvability is a quality attribute. Bennet and Rajlich [6] also mention the importance of raising the abstraction level and point out two research topics on this subject:

- Architecting systems in a way that they allow changes without damaging the integrity of the system
- Constructing architectures which can be evolved in a controllable way.

To measure how evolvable a system is, it is desirable to have a mechanism that evaluates the system at high levels of abstraction. Currently, there are methods that can estimate how a system meets certain quality requirements and we believe that some of these methods can be adapted to measure evolvability. For example, evaluation techniques based on scenarios can be easily adapted to evaluate designs with respect to evolvability; SAAM [11] may be specialized to work with evolution scenarios and ATAM [12] can be used to measure the trade-off between evolvability and other quality attributes. Although scenario based techniques may supply great value of information for many quality attributes, they may not be very useful for quality attributes that deal with future changes. This is due to the limitation of the scenario generation process to the generators' view of the future. For example, when evaluating with respect to evolvability most of evolvability scenarios may be missed by the scenario generators (most of the time the stakeholder) which may result in wrong judgments about the current architecture. It is obvious that a model based evaluation technique may be more suitable for evolvability; though a great deal of work has to be conducted in order to find metrics for evolvability.

Currently, ISO/IEC 9126 standard [10] derives metrics for evolvability based on the goal-question-metric (GQM) [4], [13]. The steps that are taken during an evolution request act as the goal (e.g., analyze the current system). Then, for each goal a set of questions is generated and for each question a

metric is associated (e.g., what is the time required to the find the changes that need to be done in order to analyze XYZ change in the requirements?). Finally, from these metrics the architectures average response to an evolution request is evaluated; which in turn may give some insight about the evolvability of the current architecture.

IV. Evolvability and Other Quality Attributes

This section of the paper tries to depict where evolvability stands with respect to other quality attributes that deal with "changes" in a system and tries to distinguish these attributes from evolvability. In the literature, most of the time the term evolution is used with maintenance, and evolvability is used to mean maintainability or modifiability. This is because the changes that evolution refers to are not identified and since, in this paper, we identify these changes, we should also provide means of distinguishing evolvability from other quality attributes. Before going to the further details of comparing these attributes, we first present their definitions.

The ISO/IEC 9216 standard [10] defines maintainability as the set of attributes that have a bearing on the effort needed to make specified modifications. These modifications include corrections, improvements and adaptations to the changing environment [7]. Modifiability is defined as the ability to make changes quickly and cost effectively [10]. These changes include addition of new requirements (extensibility), deleting unwanted capabilities and portability. Evans and Marciniak [15] define adaptability as the ease with which software satisfies differing system constraints and user needs.

As it can be seen from the definitions of these quality attributes, it is difficult to distinguish evolvability from them; however, by considering the laws of evolution the difference between them becomes clearer. The tasks included in adaptability and maintainability, for example, disobey the increasing complexity law, since when a system is corrected or adapted to another environment, the complexity of the system does not change, although the complexity of these operations on the system may be too high.

We think that the definition given for modifiability is too broad; the changes included for adaptability and maintainability can easily be fitted to modifiability. This is also true with our definition of evolvability. For this, we view modifiability as a superset of all quality attributes that deal with changes in a system. Then one can easily say that a modifiable system is also evolvable; we think further attention has to be paid in order to understand the relation between the evolvability and modifiability quality attributes.

V. Conclusions and Future Work

In the literature, most of the research on evolvability focuses on source code level evolvability analysis; though, we believe that evolvability should be considered while designing the initial system. For this, we will develop techniques that can evaluate architectures with respect to evolvability. In order to achieve this goal, we first need to define evolvability and in this paper we present our definition of evolvability.

Our next step, towards pursuing the goal of finding techniques that can evaluate architectures with respect to evolvability, is identifying the operations involved in evolvability and conducting empirical analysis on these operations. To do so, we are going to use the architecture of a system that has been evolving for years. This empirical analysis study is going to be similar to the ones conducted on the source code level; however, it is going to allow us to understand the evolution at the architecture level. For example, from this study we may see a relationship for the number of addition operations done over time; we can use this relationship to estimate how many component additions will be made in the next release.

Then we are going to focus on identifying metrics for evolvability so that we can reason about the evolvability of an architecture. For example, let us assume that the number of connections to a component is our metric for evolvability. Obviously, removing or making additions to a component with many connections would be very costly. Furthermore, if the architecture is composed of such components then the system would not be very evolvable.

References

[1] C. F. Kemerer and S. Slaughter: An Empirical Approach to Studying Software Evolution. IEEE Trans SE 25(4) pp 493-509 (1999).

[2] C. Ler, D. Rosenblum, and A. van der Hoek: The Evolution of Software Evolvability, International Workshop on the Principles of Software Evolution: 131-134 (2001)

[3] L.A. Belady and M.M. Lehman: A model of large program development, IBM Sys. J. vol. 15, no. 1, pp. 225-252 (1976).

[4] S. Cook, H. Ji and R. Harrison: Software evolution and evolvability. Technical Report, University of Reading, UK (2000)

[5] M.M. Lehman and L.A. Belady, Program evolution - processes of software change. London: Academic Press (1985).

[6] K. H. Bennet, V. T. Rajlich: Software Maintenance and Evolution: a Roadmap. International Conference on Software Engineering. Proceedings of the Conference on the Future of Software engineering, pp. 73-87 (2000).

[7] L. Dobrica, E. Niemel: A Survey on Software Architecture Analysis Methods. IEEE Trans. Software Eng. 28(7): 638-653 (2002).

[8] M M Lehman, J F Ramil and G Kahen, Evolution as a Noun and Evolution as a Verb, Workshop on Software and Organisation Coevolution, Imp. Col., London (2000).

[9] IEEE Standard 1061-1992, Standard for Software Quality Metrics Methodology, New York: Institute of Electrical and Electronics Engineers (1992).

[10] ISO/IEC91Int'l Organization of Standardisation and Int'l Electrotechnical Commission, Information Technology and Software Product Evaluation and Quality Characteristics and Guidelines for Their Use, ISO/IEC 9216, (1991).

[11] R. Kazman, L. Bass, G. Abowd, and M. Webb: SAAM: A Method for Analyzing the Properties of Software Architectures. Proc. 16th Int'l Conf. Software Eng., pp. 81-90 (1994).

[12] R. Kazman, M. Klein, M. Barbacci, H. Lipson, T. Longstaff, and S.J. Carriere: The Architecture Tradeoff Analysis Method. Proc. Fourth Int'l Conf. Eng. of Complex Computer Systems (1998).

[13] V. R. Basili, G. Caldiera and H. D. Rombach: Goal Question Metric Paradigm. In: "Encyclopedia of Software Engineering", Volume 1, pp. 528-532, (1994).

[14] D.E. Perry: Dimensions of Software Evolution. Procedings Conf. Software Maintenance, (1994).

[15] W, M. Evans and J. Marciniak: Software Quality Assurance and Management. New York, NY: John Wiley & Sons, Inc. (1987).

[16] T. Mens,J. Buckley,M. Zenger and A. Rashid: Towards a Taxonomy of Software Evolution. International Workshop on Unanticipated Software Evolution, Warsaw, Poland (2003).