EPiC Series in Computing
Volume 64, 2019, Pages 204–214
Proceedings of 28th International Conference on Software Engineering and Data Engineering

Documenting architectural rationale using source code annotations: An exploratory study

Santiago Hyun Dorado[1] and Julio Ariel Hurtado[1]
[1] Universidad del Cauca, Cauca, Colombia
santiagodorado@unicauca.edu.co, ahurtado@unicauca.edu.co

Abstract

The architectural rationale is the documentation of the reasons why certain software design decisions are made that satisfy quality needs in a system. On many occasions this rationale is implicitly found in precise sentences of the system's documentation or the same source code, making it difficult to understand and make decisions on the maintenance phase, leading to a deviation and erosion of the architecture and therefore aging of the software. In this paper, we discuss the utility of a tool based on code annotations as an alternative to document the architectural rationale with the source code. For this, a quasi-experiment with local industry software engineers is done, in order to answer the research question: Does the source code annotations, with information about the architectural rationale, improves the software maintenance? The quasi-experiment is done with software engineers who know Java language and notions of software architectures. It included 3 tasks that involve changes to the architecture and its documentation. After the results were analyzed using the t-student test, concluding that the participants who use the annotations with information of the Architectural Rationale achieve a better understanding of the architecture and its Rationale than those using a traditional way for documenting the rationale(documents). However, the efficiency and effectiveness of maintenance time do not depend on the Rationale specification. With the same problem, the variation was due to the ability of individuals to develop software, but the documentation of the architecture, in general, was very important to be able to make the changes within the limits.

1 Introduction

The scalability and maintainability of a software system are key aspects to develop a system with a much longer life cycle [1]. Due to this, a lot of effort is made in the initial phases of the development to design a good architecture, capable of allowing the construction of software systems more extensible in the future and much easier to modify, without having to change the source code already written [2]. However, a common problem in software development is to dedicate a lot of

F. Harris, S. Dascalu, S. Sharma and R. Wu (eds.), SEDE 2019 (EPiC Series in Computing, vol. 64), pp. 204–214

effort to the initial design of the architecture and not to update it while the software life cycle progresses, causing modifications to the system to be implicit in the code and cause the erosion of the architecture, changing its structure and decreasing its performance [3]. The correct documentation of architectural decisions allows designers to transmit knowledge through time [4]. This knowledge or justification of architectural decisions that meet quality needs is known as "Architectural Rationale" and it describes why a change is made in architecture [4]. Many times, this rationale is implicitly found in particular sentences of the source code, causing much more time, effort and money for understanding, maintain and extends the architecture [5]. The research goal is to analyze the documentation of Architectural Rationale through code annotations, to determine the value of the code annotations as a tool for documenting the architectural rationale. This value is analyzed in the context of the maintenance of small system respect to the maintenance of its architecture in terms of efficiency and effectiveness, as well as the architecture comprehension and its rationale, while architectural changes are being made by a new architect. To validate the research hypotheses, it was necessary to perform a quasi-experiment. The central hypothesis states the use of code annotations as a software tool to document the architectural rationale improves the maintainability of a system. In this experiment, we had one system developed in Eclipse IDE, 4 software engineers with Java experience of at least 2 years, the system documentation in a repository with version control, ARAT Eclipse Plugin (Architecture Rationale Annotations Tool) and libraries with methods of reflection[1]. The experiment consisted of four participants individually perform 3 tasks and complete a survey, the tasks that the groups did include architectural changes and their documentation, to document these changes and the Rationale involved through the documentation strategy randomly assigned to each group: with the annotation model or in a text / Word document. Finally, participants completed a survey that tries to confirm the understanding of the architecture and its rationale, in addition to knowing the usefulness of the model in a qualitative way. To determine the efficiency, effectiveness, and understanding of the architectural changes, it was necessary to measure the time taken to make the changes to each participant, review the correctness of each task, to finally contrast the results of the two groups, to determinate the usefulness of the developed annotation model. The rest of this document is organized as follows: The second section deals with the works related to the research objective. In section 3, the experiment is planned and designed. The execution, analysis, and discussion of the experiment are described respectively in sections 4, 5 and 6. Section 7 contains the conclusions and further work.

2 Related Work

The architecture of a system according to Jasnen A et al [4] is the composition of a set of architectural design decisions and it is generally based on decisions made in the requirements elicitation activity. Two types of important design decisions are the application of tactics and architectural patterns [6], the patterns are common architectural structures, which are well defined, documented and well understood for reusing [7], [8]. The architecture defines what the system must do through the selected architecture patterns [6]; Tactics are important design decisions because they allow us approaching quality attributes [9]. However, mistakes are often resulting in decision making without determining the scope, consider legal and business aspects, neither remove obsolete decisions [10]. Relating rationale issues with software architecture allows software engineers to reuse, make changes with less effort, improve the quality of the final product and support the transfer of knowledge through time [10]. The rationale is the justification for the decisions or actions that are taken in the development process. In software engineering, it is captured and managed to improve the comprehension of a system by the stakeholders [11]. The rationale is usually found in all the decisions

[1] Reflection: It is the ability of a computer program to auto examine itself and modify its behavior and structure at runtime

that are made in the analysis, design implementation and testing. In this work, we will refer to the design rationale, which is defined by Allen H. D. et al [11] as the reasons that determine the realization and the influences on the artifact design [12]. On the other hand, according to Milan N. et al [13], the code annotations are structured and declarative labels in the source code that allows developers to customize the functionality of a program. These annotations are placed on source code elements called annotation objectives, these can be global variables, function parameters, return values, methods, user-defined types, among others [14]. The implementation of the source code annotations generally has two approaches:1. As a plugin[2] to obtain source code information at compile time. 2. As a reflection mechanism for getting program information at runtime. The first approach requires the existence of a processor factory to handle each annotation declared and the second approach requires libraries allowing capture of metadata in the source code [13]. Archium is an extension of Java proposed by Van der Ven. et al [10] that uses an architectural model, defined by an ADL (Architectural Description Language) and a decision model which allows modeling design decisions with its rationale through DRL (Decision Representation Language), an argumentation scheme that allows to architect modeling a decision including decision problems, alternatives, objectives, claims, and groups. With the Van der Ven et al [10] approach it is possible to define design decisions in a structured way. However, the rationale is an element that depends considerably on the architect's mental model. The generated artifacts by Archium work as a bridge between architecture and design decisions, however, these are not directly related to the implementation of the system, neither specified as code elements. In this paper, we present a model of architectural rationale directly codified at the code source level. This rationale is located on architecturally relevant code elements.

3 Planning the experiment

The goal of this research is to analyze the architectural rationale documentation through code annotations, to determining the value of the code annotations as a tool to document the architectural rationale for achieving a maintainable architecture in terms of efficiency, effectiveness, and comprehension while architecturally relevant changes are realized by software architects. For the preparation of this document, the guide to report experiments in software engineering proposed by Andreas J. et al [15] was followed up.

3.1 Hypotheses

The overall hypothesis of this experiment states the architecture rationale codified as source code annotations improve the architecture maintainability aspects such as the efficiency, effectiveness, and architecture (including its rationale) comprehension at maintenance time. To confirm the general hypothesis, the null and alternative hypotheses are formulated for each aspect related to architecture maintenance:

H1: The effectiveness of achieving an architectural change in a system using rationale documented as code annotations:

- **Null:** it is not significantly different than the effectiveness when changes are realized without code annotations.
- **Alternative:** it is significantly greater than the effectiveness, when changes are realized without code annotations.

[2] Plugin: It is a complement for software that adds functionality

H2: The efficiency for achieving an architectural change in a system using rationale documented as code annotations:

- **Null:** is not significantly different than the efficiency when changes are realized without code annotations.
- **Alternative:** it is significantly greater than the efficiency when changes are realized without code annotations.

H3: The comprehension of the architecture and its rationale for achieving an architectural change in a system using rationale documented as code annotations:

- **Null:** it is not significantly different when changes are realized without code annotations.
- **Alternative:** it is greater than comprehension when changes are realized without code annotations.

3.2 Variables

The variables of this experiment were divided into independent variables (Can be controlled) and dependent variables (Results obtained). In this work, the only independent variable is the specification or not of architectural rationale as source code annotations. The possible values are “Yes” or “No”, this variable is of categorical type. The dependent variables, efficiency, and effectiveness are calculated by measuring other variables such as time and the level of correctness in the performance of the task. Also, define the reference values was necessary to calculate the results of the efficiency and effectiveness variables as a percentage measure. A third dependent variable is the comprehension level of the architecture and its rationale, in a range between 0 and 100 according to an assessment protocol. Participants should perform 3 maintenance tasks. The assessment protocol allows us to measure the correctness level of each task, for the first, second and third tasks, an example for the protocol for measuring the correctness level is described in Table 1. The detailed description of all tasks can be found in section 3.5.

	CL1
Value	**Description**
100	The participant did not realize the change.
200	The participant realize the changes but she/he did not complete the task.
300	The participant completed the task but she/he did not maintain the established design.
400	The participant completed the task keeping the design, but it took longer than the established time.
500	The participant completed the task keeping the design in the established time.

Table 1: Correctness Level task 1

The sum of the levels of the correctness of all tasks is defined as ‘TCL’ expressed as follows:

$$\boldsymbol{TCL} = \sum_{i=1}^{n} CLi$$

Where ‘n’, is the total number of realized tasks.

The time in which a participant performs a task is denoted by the letter ‘t’. The time it takes to perform all tasks ‘T’ is defined as the sum of each of the times taken in each task, as expressed in the following equation:

$$\boldsymbol{T} = \sum_{i=1}^{n} ti$$

The estimated values that are defined for the total time and the total correctness are shown below:

- The estimated time for the first, second and third tasks is 1 hour, 50 minutes and 30 minutes respectively, with an estimated total time of **Te =** 140 minutes to perform all tasks.
- Total estimated correctness level in the execution of all assigned tasks:

  TCLe = 500 + 500 +500 = 1500 cl

Efficiency: Value measured concerning the correctness and total time obtained in carrying out the tasks, the absolute efficiency value 'E' is defined below as:

$$\boldsymbol{E} = \frac{TCL}{T}$$

Due to absolute efficiency values are not easy for comparing, so the efficiency value must be calculated in terms of percentage considering a reference point as the ideal performance of tasks, which is defined as:

$$\boldsymbol{Er} = \frac{TCLr}{Te}$$

Where the total level of correctness is: **TCLr**= 400 + 500 + 500 = 1400 cl. Which we get the following efficiency value referring 'Er':

$$\boldsymbol{Er} = \frac{1400}{140} = 10\ \frac{cl}{min}$$

Finally, we obtain the efficiency percentage for each participant:

$$\%\,\boldsymbol{Efficiency} = \frac{E}{Er} * 100$$

Effectiveness: Value measured by correctness in the performance of all tasks divided by the total expected correctness. In this work, we define the effectiveness of 'EF' as:

$$\boldsymbol{EF} = \frac{TCL}{TCLe}$$

Finally, we obtain the percentage of effectiveness of each participant by:

$$\%\,\boldsymbol{Effectiveness} = EF * 100$$

Comprehension level: It is a quantitative value evaluated by cross-referencing information between the responses of a survey and the documentation provided in a task of maintainability documentation. To calculate the value of this variable it is necessary to define the documentation level according to the understanding expressed by the participants in the different artifacts of the experiment. Table 2 shows the design to calculate the level of comprehension of the architecture and its Rationale, through the evaluation of the content in the performed task documentation and the writing in the survey responses. These auxiliary variables are in a range of 0 to 100 for a total maximum of 500. The level of comprehension denoted as ' C ' is the average of the values of the auxiliary variables.

$$\boldsymbol{C} = \frac{CDf + CDd + CDa + RDd + RDa}{5}$$

CDf: Changes Documentation at functional level, **CDd**: Changes Documentation at design level, **CDa**: Changes Documentation at architectural level, **RDd**: Rationale Documentation at design level, **RDa**: Rationale Documentation at architectural level

Participant	CDf	CDd	CDa	RDd	RDa	C
Participant 1	100	100	100	100	90	98

Table 2: Example of Comprehension level in documentation

3.3 Participants

To obtain the participation of people, emails were sent, inviting potential participants to the experiment. The participants full fit the characteristics as follow:

1. Basic knowledge about software architectures.
2. Experience as Java developer for at least two years.

For selecting the participants, the guidance of Andrew J. et al [16] is followed. The participants were software and electronic engineers currently related to the local software industry, so all the selected people know fields of computer science and related fields. All were men over twenty-five (25) years, counting with experience between two (2) and twelve (12) years in the industry and playing different roles such as analyst, product engineer, software developer and architect. Also, one of the participants has active work as software engineering professor. The experiment and the process rubric was explained to the participants, introducing the experiment and obtain their written permission.

3.4 Experimental material

The system used in the experiment, is a family of games ‘n' in a- row, which consists of a game in which each player must vertically, horizontally or diagonally place on a board, n consecutive chips in an arrow. This game has many variants, one of them is called 'Connect four': this variant is a game on a board of seven columns wide and six rows high. Each player uses tiles of a particular color and, alternating the turn, places tiles on the board. The winner is the first achieving four chips of their color in a line, either horizontally, vertically or diagonally. The source code[3] has about 1664 lines of code, 3 tiers and four main components of manage games, data and presentation ways.

Some participants (experimental subjects) count with the code annotations, including information about the previous architectural rationale, specified at the source code level, using the library in the folder 'lib' called 'ARAT.jar' (Architectural Rationale Annotations Tool[4]). It has documented architectural reasons and shows a report with the location and information marked as annotations. All participants had access to the Software Architecture Document SAD, using different architecture perspectives: scenarios, logic, development, process and physical. Additionally, the SAD includes different models such as use cases, classes, sequence, packages, components, and deployment diagrams.

3.5 Tasks

The experiment consisted of three tasks to be performed by each participant:

1. Additive maintenance (New game 'PopOut'): the tiles can be placed in a full column, in this case, the player inserts the tile and all the tiles descend one position, eliminating the tile below. As a result of this new rule, other situations appear, for example, both players can, at the same movement, achieve connect four, and then the game must continue. Also, the player who places a tile could lose the game. This would happen if the tile is placed in a full column and gets your opponent to keep your tiles in line.
2. Corrective maintenance (Architectural change): the result of a game must be stored locally in a file located in a directory of the system, this directory can be accessed by anyone who knows where the project files are located. It is a new requirement where the system includes

[3] Source code: https://github.com/zahydo/cuatroenlinea
[4] ARAT: https://zahydo.github.io/arat-V1.0/

the capacity for delegating the responsibility for managing the files with the information of the results to a server. It would wait for requests from the instances of the game. To keeping the data safe, considering a separate system from the game, so that they cannot be modified and these can be accessed by several instances of the game concurrently.

3. Architectural Rationale Documentation (Documentation): to verify the comprehension of the architecture and its Rationale, the participants documented the changes realized while the tasks before mentioned were performed, according to the documentation strategy assigned (including or not the ARAT model). Participants also wrote quality attributes to be achieved and potential techniques or strategies to be used. In the course of this task, the participants reviewed different software artifacts such as the SAD Software Architecture Document (Software Architecture Document), the source code and the architectural rationale code annotations when applied.

3.6 Process

The participants were automatically grouped into 2 pairs, using a web tool [17], which allows us to create random groups. The system source code was located on the 'src' folder, with annotations for one group and without annotations for the other; the reflection library and the annotation model library was located on the 'lib' folder. The 'ExampleSockets' folder includes an example of a sockets based implementation suggested to the second task. Additionally, each participant had access to the Software Architecture Document in PDF format.

The experiment begins with an introduction to the participants about the architectural rationale and related concepts. Then, the researchers presented the source code annotations, making an example for understanding the structures, use, and some constraints. After, the researchers presented the annotation model with architectural rationale information. Additionally, they explained as change tasks must be performed. Corresponding material was delivered to each group and timestamps were taken at finish each. After carrying out all the tasks and delivering the results, each participant completed a survey with the following questions:

1. Describe the reasons why the code was organized under the structure you proposed.
2. Describe the reasons why your architectural structure had to be altered to meet the requested changes.
3. Was the rationale information provided in this experience useful for making the modifications?
4. Why do you think it is important to document the Architectural Rationale in software development?

3.7 Analysis procedure

The systems with the modifications were received for evaluating the level of correctness in the accomplishment of the tasks. In order to determine the results of the experiment, a hypothesis test was carried out to confirm that the use of code annotations with information from architectural rationale improves the maintainability of Architecture in terms of efficiency, effectiveness, and comprehension of the architecture and its rationale, when changes are made with an impact at an architectural level. The test of the hypothesis used is the t- student test, this allows us to determine if there is a statistical significance between two variables, this means that the presence of the independent variable positively affects the results of the dependent variables. For this test was necessary to have two equal samples and the same variance must be assumed between the samples of each group. In this work, the groups are divided into: with annotations group (With) and without annotations group (Without).

4 Execution

The configuration of the project is done in the NetBeans development environment and the files were downloaded from the source code of the experimental material. The participants began to perform the tasks and the completion times of each task for each participant were captured. As each participant completed the tasks, the survey was delivered, for gathering qualitative information that contributes to the definition of the final structure of the annotation model for the architectural rationale. Below some of the most accurate responses to the survey made to the participants are presented:

1. The system was organized in this way to facilitate the maintainability and extensibility of the architecture.
2. Because a new functionality was required in a new component where the data will be stored for security.
3. In general terms, the participants answered that the documentation strategy that corresponded to them was useful.
4. At a general level, the answer to participants was what is important to facilitate the maintainability of a system to subsequent people to those who develop it.
5. In this question, the participants named the following aspects that seemed important to them: Type of requirement to satisfy, requirement priority, rationale modifications history.

Table 3 shows the results for the efficiency, effectiveness, and comprehension of the Architecture and its Rationale in the task realization that involve changes in the Software Architecture.

	% Efficiency		**% Effectiveness**		**% Comprehension**	
Participant	**With**	**Without**	**With**	**Without**	**With**	**Without**
1	83.3	53.7	80	53.3	90	50
2	86.1	103.7	86.7	93.3	80	62

Table 3: Variables results

5 Analysis

5.1 Descriptive statistics

Each of the hypotheses considers an independent variable on a nominal scale with two levels (with annotations, without annotations) and a dependent variable (effectiveness, efficiency or comprehension of the architecture and its rationale). To confirm the hypotheses a t-student test was used assuming a normal distribution, variances and sample size equal, for two independent measurements. Because we are interested in knowing if the results are above the normal distribution, a one-tailed t student test is used.

5.2 Data set preparation

To determine the value of the dependent variables, it was necessary to transform the task completion times into minutes. To calculate the comprehension level of the architecture and its rationale it was necessary to make a manual review of the results of task 3 of each participant, in addition, it was necessary to evaluate the answers of the survey that was given to each participant, which confirm the comprehension of the architecture and rationale involved, as well as giving a critical value to the annotation model.

5.3 Hypotheses testing

Once the dependent variables have been calculated, the hypotheses were tested through the t student test with a level of significance equal to 0.05.

Efficiency: The average efficiency percentage for the group with code annotations was 84.7%, unlike the group without code annotations, which had an average efficiency percentage of 78.7%. However, the p-value was 0.4913 which is much higher than alpha (0.05), therefore, we accept the null hypothesis and reject the alternative hypothesis.

Effectiveness: The results show that the effectiveness average for the group with code annotations was 83.3% and the average percentage for the group without code annotations was 73.3%. However, with the p-value of 0.4369 greater than 0.05, the alternative hypothesis must be rejected and the null hypothesis accepted.

Comprehension: The average comprehension for the group with the code annotations was 85%, unlike the group without annotations which is 56%. When the p-value is calculated, it gives 0.0327, which is less than our alpha 0.05, therefore, we accept the alternative hypothesis and reject the null hypothesis. This means that the use of code annotations with architectural rationale information does not improve the efficiency or effectiveness in making changes with an architectural impact, but we can affirm that the increase in the understanding of the architecture and its rationale is due to the use of source code annotations with structured rationale information. In Figure 1 we can observe the distribution of the data with respect to the average of the efficiency and effectiveness of each group, where we can see that the efficiency and effectiveness average is higher in the group with code annotations, however, the results were kept within the range of the group without code annotations, due this we can affirm that the increase in the median for the group with annotations is due to other factors and not to the use of code annotations with information from architectural rationale. Finally, we can observe that the values for the group with code annotations were greater and were outside the range of the results of the group without code annotations, for this reason, there is a statistic significance in the results about the architecture and its rationale documentation comprehension.

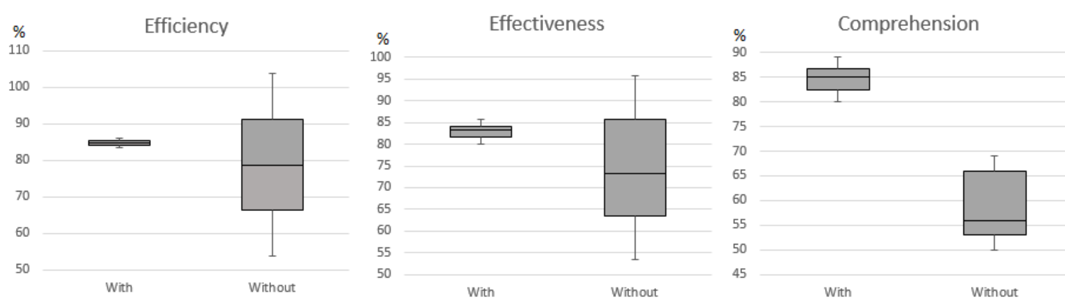

Figure 1: t-student results

6 Discussion

According to the results obtained from this quasi-experiment, the documentation of the architectural rationale through the use of code annotations has a positive effect on the comprehension of the architecture and its rationale, however, it does not guarantee efficiency or effectiveness for executing maintenance tasks at the same system. When making changes with an architectural impact, which does not indicate that the efficiency and effectiveness of making changes with an architectural impact do not depend on the rationale specification, but there are other factors such as the ability to develop, the experience, the development environment among other factors. The results showed that some participants performed better at a functional level than others, this is because some of them were currently working on software development and others have not developed software for some period. The latter is engaged in other engineering tasks, such as requirements analysis and systems design. The participants who read the information from the annotations expressed better the reasons for which the system was constructed in that way and the underlying reasons for the new modifications. This quasi-experiment in addition to testing the usefulness of the code annotations in comprehension the architecture and the decisions behind it, allowed us to capture comments and advice that contribute to the definition and improvement of the annotation model, through a survey in which the participants freely express their impressions of the architectural rationale, for example, several of them maintained that they would make use of the annotation model in the industry, since this facilitates the transmission of knowledge over time and of the people who are responsible for maintenance. It is important to have the participants motivated with the experiment, this allows them to be much more active in the development of tasks and have full interest in experimenting. The training on architectural rationale should be addressed with more time, since it is an aspect that is not usually taught in the academy or the industry, but which, according to the participants, should be addressed to avoid future problems in the maintenance of a software system. The use of code annotations is also a subject that requires training and exemplification time, this technology is frequently used in many known use cases, however, declaration and use of personalized code annotations is little known and requires an additional effort to generate value as a complementary tool.

7 Conclusions

In this quasi-experiment, we evaluated whether the code annotations with architectural rationale information has a positive impact on the efficiency, effectiveness, and comprehension of the architecture and its rationale, managing some architectural changes tasks. For this, a game system was changed which participants had some modifications and documented the reasons why the system changed its structure. As the central measurement trends shows, the average and the median was better for the annotation model with respect to the efficiency, effectiveness, and comprehension of the architecture and its rationale, however, when the hypothesis confirmation tests are carried out, the use of the code annotations with architectural rationale information only benefits the comprehension of the architecture and its rationale. However, the efficiency and effectiveness of making changes with an architectural impact do not depend on the way as the rationale is specified, because there are other affecting factors such as the ability to develop, the experience, the development environment, and other factors. Finally, a survey was carried out to obtain the opinion of the participants regarding the documentation of the rationale and its importance, to which the participants agree, that it facilitates the maintainability work and the comprehension of the architecture of a system implemented by others, for instance, third party development. As future work, we intend to make modifications to the annotation model based on the observations and results of this quasi-experiment, to develop a controlled experiment with larger samples of engineers and with a second system closer to commercial development problems. In other future work, we want to raise the annotation model to the free development community to have more observations by developers from around the world, to detail the annotation model with real needs of programmers and designers. As a long-term work, we want to establish a model of annotations supporting the modeling through UML, which can identify, using a metaphor, the concept of the architectural rationale.

References

[1] A. E. Sabry, “Decision Model for Software Architectural Tactics Selection Based on Quality Attributes Requirements,” *Procedia Comput. Sci.*, vol. 65, no. Icc, pp. 422–431, 2015.

[2] M. Shahin, P. Liang, and Z. Li, “Do architectural design decisions improve the understanding of software architecture? two controlled experiments,” *Proc. 22nd Int. Conf. Progr. Compr. - ICPC 2014*, pp. 3–13, 2014.

[3] L. De Silva and D. Balasubramaniam, “Controlling software architecture erosion: A survey,” *J. Syst. Softw.*, vol. 85, no. 1, pp. 132–151, 2012.

[4] A. Jansen and J. Bosch, “Software Architecture as a Set of Architectural Design Decisions,” *5th Work. IEEE/IFIP Conf. Softw. Archit.*, pp. 109–120, 2005.

[5] J. E. Burge, J. M. Carroll, R. McCall, and I. Mistrík, *Rationale-Based Software Engineering*, Springer. Springer-Verlag Berlin Heidelberg, 2008.

[6] N. B. Harrison and P. Avgeriou, “How do architecture patterns and tactics interact? A model and annotation,” *J. Syst. Softw.*, vol. 83, no. 10, pp. 1735–1758, 2010.

[7] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software architecture*, 1st ed. Germany, 1996.

[8] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects*, John Wiley., vol. 2. West Sussex PO19 1UD, England: John Wiley & Sons, Ltd, 2000.

[9] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Second. Addison Wesley, 2003.

[10] J. S. van der Ven, A. G. J. Jansen, J. A. G. Nijhuis, and J. Bosch, “Design Decisions: The Bridge between Rationale and Architecture,” in *Rationale Management in Software Engineering*, 2006.

[11] A. H. Dutoit, R. McCall, I. Mistrík, and B. Paech, “Rationale Management in Software Engineering: Concepts and Techniques,” in *Rationale Management in Software Engineering*, SPI Publisher Services, Pondicherry, 2006.

[12] L. Bratthall, E. Johansson, and B. Regnell, “Is a Design Rationale Vital when Predicting Change Impact? -- A Controlled Experiment on Software Architecture Evolution,” in *Product Focused Software Process Improvement*, 2000, pp. 126–139.

[13] M. Nosáľ, M. Sulír, and J. Juhár, “Source Code Annotations as Formal Languages,” *Comput. Sci. Inf. Syst. (FedCSIS), 2015 Fed. Conf.*, vol. 5, no. 1, pp. 953–964, 2015.

[14] M. Das *et al.*, “Source Code Annotation Language,” vol. 2, no. 12, 2009.

[15] A. Jedlitschka, M. Ciolkowski, and D. Pfahl, “This is a preliminary version of a chapter in Reporting Experiments in Software Engineering,” 2007.

[16] A. J. Ko, T. D. LaToza, and M. M. Burnett, “A practical guide to controlled experiments of software engineering tools with human participants,” *Empir. Softw. Eng.*, vol. 20, no. 1, pp. 110–141, 2015.

[17] “Echalo a la suerte.” [Online]. Available: https://echaloasuerte.com/draw/new/groups/.