International Journal of u- and e- Service, Science and Technology
Vol.7, No.2 (2014), pp.91-100
http://dx.doi.org/10.14257/ijunesst.2014.7.2.09

# Fuzzy and Cognitive Attraction Based Solution for the Software Architectural Change Decision Problem

Zohair Chentouf and Mashael Al-Duwais

*College of Computer and Information Sciences*
*King Saud University, KSA*
*{zchentouf, malduwais}@ksu.edu.sa*

***Abstract***

*Software change is an ineluctable part of the software lifecycle. To avoid the degradation of the software quality, changes need to be performed in a controlled and rigorous manner. Most often, the management of software changes presents itself as a decision problem where software analysts have to decide whether to accept the change request and eventually how to implement the change if it is accepted. The present work's focus is on the architectural change management. We propose a solution to this problem, which requires the software analysts to assess the impact of the software change on a set of software attributes. This assessment uses the fuzzy sets theory. An algorithm is also designed to infer a common decision out of the analysts' very likely subjective and inaccurate individual assessments. This part of the solution employs the Cognitive Attraction Theory. The solution is validated through an industrial case study.*

**Keywords:** *Software Engineering, Software Change, Fuzzy Sets, Cognitive Attraction Theory*

## 1. Introduction

Software requirement change management is a continuous and inevitable process. Setting fixed and stable requirements is a challenge for software engineers because of changes in the business environment or the opinions of the project's stakeholders [1, 2]. The software change requests are formulated by the stakeholders and submitted to the Change Control Board (CCB) for approval. The CCB members then analyze the impact of the requested change, evaluate the cost of the implementation and non-implementation of the change, and decide either to accept or reject the change [3, 4].

The literature addressed the software change management problem. Some research works used statistical classification models and data about past software changes to predict if the requested change is defect prone [2, 5-10]. Other research works' interest went to classifying software changes based on the types of the implied code revision [3], document update [11], and code refactoring [12]. How change classification might be biased by the software team members' roles has been studied in [13]. Another set of research works focused on helping the CCB perform a successful software architectural change. Tools proposed in [14, 15] aim at providing representations of the software architecture with different levels of detail. The environment designed in [16] offers a framework which helps the CCB analyze the functional, organizational, short-term goals, and long-term goals aspects of the requested change. In [17], a tool is proposed to plan the replacement of a software module by analyzing the trace of its past interactions with other modules. Another subset of the literature focused on analyzing the impact of the requested change. More generally, the impact analysis (IA) has as objective to identify the effects of the change and/or how to implement it [18]. Under this perspective, IA may focus on the code. For example, [19-22] use program slicing techniques to identify the modules that may affect a given set of variables. In [19, 23-25], cross reference techniques are used in order to trace references to given variables or procedures. IA may consist to track the propagation of similar changes through the past versions of the software. Configuration management systems are of a particular usefulness [26, 29, 30]. IA may also consist to evaluate the scope of the requested change through identifying the affected artifacts [27, 28]. Some research works designed architectural IA techniques. For example, scaling the proposed code changes up to the architectural level [31], using dependency relationships among modules to trace the change propagation through software modules [7, 32], and using statistical association to determine the probability of change propagation from a given module to other ones [21].

The present study builds on the software change characterization framework of Williams and Carver [33] called the Software Architecture Change Characterization Scheme (SACCS). The latter contains a list of software attributes that can be affected by a software change. The aim of SACCS is to help software engineers better analyze requested architectural changes through analyzing the possible impacts on every one of those attributes. Our research is leveraged by the following software practice facts, which we will consider as requirements:

- The CCB members discuss whether to implement or reject the requested change and eventually which particular change solution to implement if the change is accepted [4] (R1).
- The CCB members might have different, subjective, and incompatible change impact evaluations [1,2] (R2).
- The CCB members need an automated way to infer a common decision from their individual evaluations (R3).

The implementation of the above listed requirements furthers the work of Williams and Carver. The result is a framework which contains a conceptual model and a decision algorithm. The conceptual model is mostly derived from [33]. The decision algorithm employs a combination of the fuzzy sets theory and the Cognitive Attraction Theory. As it will be explained in the sequel, the latter is used in order to fulfill the requirements R1 and R3, and the former is used to implement the requirement R2.

The rest of the paper is organized as follows. Section 2 presents the SACCS model. Section 3 details the software architectural change decision solution. Section 4 presents the validation of the solution. Section 5 contains a summary of the study and suggestions on potential directions.

## 2. The SACCS Model

Williams and Carver [33] surveyed the software architectural change impact literature and came out with a comprehensive list of the software attributes that might be affected by changes. They called their list: Software Architecture Change Characterization Scheme (SACCS). The goal of this model is to assist software professionals in analyzing the impact of architectural change requests. In the present work, we move a step forward; we elaborate a change decision algorithm whose inputs are the evaluations given by every CCB member to the expected impact of the request software change on every one of the software attributes. For this aim, we had to distinguish between software attributes that can be assessed and those which cannot. For example, SACCS contains the attribute "data access". We conjecture that analyzing the change's impact on data access is required but a CCB member cannot assess it as good or bad; he/she only can assess the emergent impact on software quality attributes like maintainability, reliability, efficiency, or portability, for example. The latter attributes are part of SACCS as well. Consequently, we consider two types of attributes organized in two layers: the first layer, called analysis layer, contains attributes that cannot be assessed. The second layer, the decision layer, contains the assessable software attributes. The attributes of the analysis layer are necessary to examine by the CCB because they contribute in the comprehension of the change impact. The attributes of the decision layer are meant to be assessed by the CCB members in order to allow the decision algorithm to infer a common verdict about the change request. Figure 1 illustrates the two layers of software attributes. The definition of every attribute is well presented in [33]. We added project management related attributes as they are very likely to be assessed by the CCB while analyzing the effects of a change. SACCS listed time attribute but we moved it under the same category as cost.

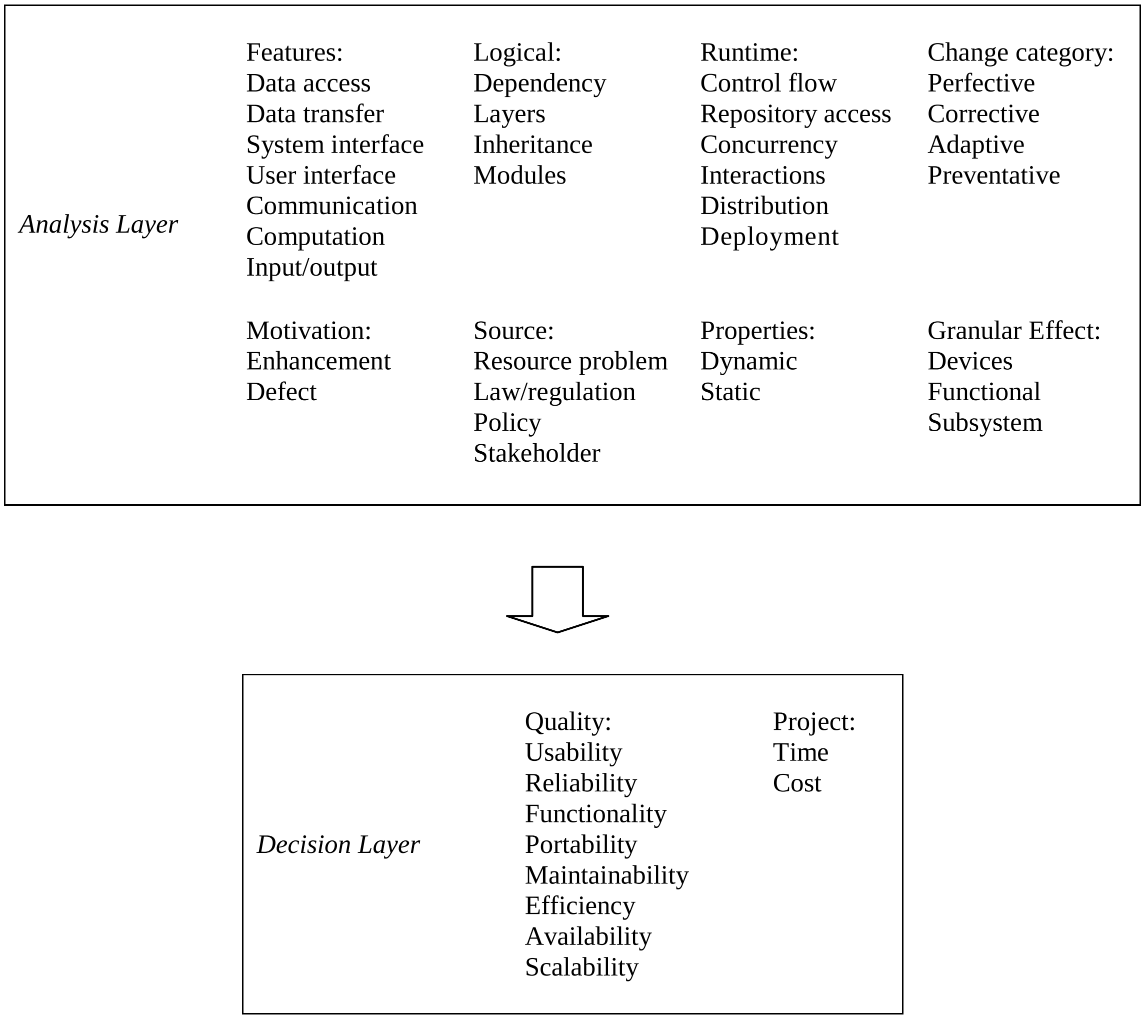
Analysis Layer
Features: Data access, Data transfer, System interface, User interface, Communication, Computation, Input/output
Logical: Dependency, Layers, Inheritance, Modules
Runtime: Control flow, Repository access, Concurrency, Interactions, Distribution, Deployment
Change category: Perfective, Corrective, Adaptive, Preventative
Motivation: Enhancement, Defect
Source: Resource problem, Law/regulation, Policy, Stakeholder
Properties: Dynamic, Static
Granular Effect: Devices, Functional, Subsystem
Decision Layer
Quality: Usability, Reliability, Functionality, Portability, Maintainability, Efficiency, Availability, Scalability
Project: Time, Cost

**Figure 1. The SACCS Model**

## 3. Architectural Change Decision Problem and Solution

A software change decision problem is encountered when the CCB members $M_k$, k=1, …, NM, have to choose among ND decisions $D_j$, j=1, …, ND: to reject the change request or to implement it in one of the ND-1 possible ways. The proposed decision algorithm that is to be presented in the current section contains three phases.

**Table 1. Linguistic Terms of Attribute Importance**

| Linguistic term | Fuzzy number | Semantics |
|---|---|---|
| Very Low (VL) | (0.0, 0.0, 0.1) | Very low importance/desirability |
| Low (L) | (0.0, 0.1, 0.3) | Low importance/desirability |
| Medium Low (ML) | (0.1, 0.3, 0.5) | Moderately low importance/desirability |
| Neutral (N) | (0.3, 0.5, 0.7) | Neutral |
| Medium High (MH) | (0.5, 0.7, 0.9) | Moderately high importance/desirability |
| High (H) | (0.7, 0.9, 1.0) | High importance/desirability |
| Very High (VH) | (0.9, 1.0, 1.0) | Very high importance/desirability |

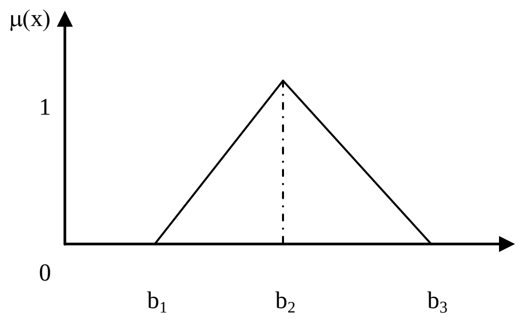

**Figure 2. Triangle Fuzzy Membership Function**

### 3.1. Fuzzy Evaluation of the SACCS Attributes

The whole decision process is leveraged by the SACCS decision attributes $T_i$, i=1, 2, …, NT (layer 2 in Figure 1). That is why, first, the CCB members are required to assess the importance of every SACCS attribute. Because of the subjectivity, vagueness, and inaccuracy of such assessments [1, 2], we adopt fuzzy evaluations using linguistic terms like very important, important, *etc*. (see Table 1). We will use triangle membership functions like illustrated by Figure 2.

The fuzzy importance $L_i^k$ of the attribute $T_i$ assigned by the CCB member $M_k$ is then translated into the corresponding fuzzy number $w_i^k$ using Table 1. The fuzzy numbers of Table 1 are illustrated by Figure 3.

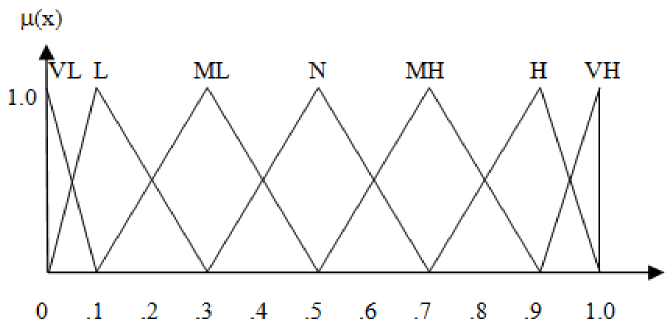

**Figure 3. The Fuzzy Numbers that Correspond to the Linguistic Terms**

The mean fuzzy importance $w_i$ of every attribute $T_i$ is calculated in order to aggregate the assessments of all the CCB members in only one:

$$w_i = \frac{1}{NM} \sum_{k=1}^{NM} w_i^k \qquad (1)$$

Let us take an example. Suppose the CCB contains three members $M_1$, $M_2$, and $M_3$. They have to decide among three change solution alternatives: to reject the change request ($D_1$), to implement it in a specific way ($D_2$), and to implement it in a second possible manner ($D_3$). For the sake of simplicity, we abstracted semantically related software change attributes into one. The resulted attributes $T_1$-$T_3$ are presented in Table 2. The fuzzy assessments of the importance of the attributes made by the CCB members are presented in Table 3 where $w_i$ are calculated too.

**Table 2. Aggregated Software Attributes**

| | |
|---|---|
| $T_1$ | Time, cost |
| $T_2$ | Reliability, availability, functionality, usability |
| $T_3$ | Efficiency, portability, maintainability, scalability |

**Table 3. Fuzzy Evaluation of Attributes**

| | $M_1$ | $M_2$ | $M_3$ | $w_i$ |
|---|---|---|---|---|
| $T_1$ | M | MH | H | (0.50, 0.70, 0.87) |
| $T_2$ | H | MH | MH | (0.57, 0.77, 0.93) |
| $T_3$ | ML | M | MH | (0.30, 0.50, 0.70) |

## 3.2. Fuzzy Evaluation of the Change Decisions' Impact

Given a change request and its related decisions $D_j$, the CCB members analyze every $D_j$ under the light of the SACCS analysis layer's attributes (first layer in Figure 1). This step will help them assign a linguistic evaluation $L_{ii}^k$ (see Table 4) to the possible impact of every $D_j$ on every attribute $T_i$ of the SACCS decision layer. Linguistic evaluations $L_{ii}^k$ are converted into fuzzy numbers $r_{ii}^k$ using Table 4.
Then, for every couple ($D_j$, $T_i$) the mean is calculated in order to aggregate the evaluations of all the CCB members into a single one:

$$r_{ji} = \frac{1}{NM} \sum_{k=1}^{NM} r_{ji}^k \qquad (2)$$

**Table 4. Linguistic Terms of Change Impact**

| Linguistic term | Fuzzy number | Semantics |
|---|---|---|
| Very Bad (VB) | (0.0, 0.0, 0.1) | Very bad/undesirable impact |
| Bad (B) | (0.0, 0.1, 0.3) | Bad/undesirable impact |
| Medium Bad (MB) | (0.1, 0.3, 0.5) | Moderately bad/undesirable impact |
| Neutral (N) | (0.3, 0.5, 0.7) | No impact |
| Medium Good (MG) | (0.5, 0.7, 0.9) | Moderately good/desirable impact |
| Good (G) | (0.7, 0.9, 1.0) | Good/desirable impact |
| Very Good (VG) | (0.9, 1.0, 1.0) | Very good/desirable impact |

For our example, the fuzzy evaluations of impacts $r_{ji}^{k}$ are presented in Table 5 and the means $r_{ji}$ in Table 6.

**Table 5. Fuzzy Evaluations of the Change Impacts**

| | $M_1$ | | | $M_2$ | | | $M_3$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | $D_1$ | $D_2$ | $D_3$ | $D_1$ | $D_2$ | $D_3$ | $D_1$ | $D_2$ | $D_3$ |
| $T_1$ | MB | G | MB | G | MG | MB | MG | MG | MG |
| $T_2$ | N | G | N | MG | MG | MB | G | G | MB |
| $T_3$ | MB | M | MB | M | MG | N | MB | MG | MB |

**Table 6. Mean Fuzzy Evaluations of the Change Impacts**

| | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|
| $T_1$ | (0.43, 0.63, 0.80) | (0.57, 0.77, 0.93) | (0.23, 0.43, 0.63) |
| $T_2$ | (0.50, 0.70, 0.87) | (0.63, 0.83, 0.97) | (0.17, 0.37, 0.57) |
| $T_3$ | (0.17, 0.37, 0.57) | (0.43, 0.63, 0.83) | (0.17, 0.37, 0.57) |

## 3.3. Common Decision Inference Using the Cognitive Attraction Theory

This phase of the proposed architectural change decision algorithm relies on the Cognitive Attraction Theory (CAT) [34, 35]. The latter underlines the ontological and epistemological (onto-epistemological) mechanisms that make human cognition emerge from neuronal dynamics. Adhering to the associanistic paradigm, CAT considers that mental entities (concepts, beliefs, ideas, etc.) are stored in the brain as an associative network. Associations are considered to represent the conceptual relations between mental entities. For example, a friend’s name and his/her face are memorized as having a link. Consequently, hearing the name evokes that friend’s face. As conjectured by CAT, mental associations are the emergence of the Hebb’s rule and they convey cognitive attractions. Based on this conjecture, CAT has been used in [35] to elaborate a moral decision algorithm, where moral goals cognitively attract moral decisions through the conceptual links that exist between every moral decision and the relevant moral goal. Such an attraction is numerically weighted by the empirical plausibility degree that expresses the extent to which the subject believes that the moral decision causes the fulfillment of the moral goal. It is also weighted by the subjective emotional valence of the moral goal. We here adapt this algorithm to solve the architectural change decision problem. For this aim, the attributes $T_i$ and the change decisions $D_j$ will replace the moral goals and the moral decisions, respectively. The weights of the empirical causality link (moral-decision, moral-goal) will be replaced by the fuzzy weight which is the expected impact $r_{ji}$ of the decision $D_j$ on the attribute $T_i$. The moral importance of the morals goal will be replaced by the importance $w_i$ of the attributes $T_i$. Figure 4 illustrates this model.

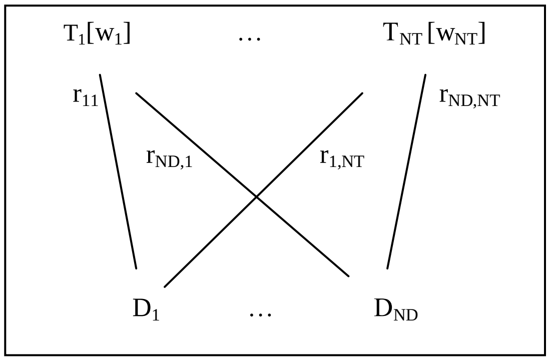

**Figure 4. CAT Approach of the Software Architectural Change Problem**

According to CAT, the attraction $A_j$ performed by all the attributes on the decision $D_j$ has to be calculated as:

$$A_j = \sum_{i=1}^{NT} r_{ij} \times w_i \qquad (3)$$

The best decision is the one with the greatest attraction valence. This can be interpreted as favoring the decision that has the greatest desirable impact on the software attributes.

Let us resume our example. After using Table 6 and Equation 3, the normalized fuzzy attraction strength values are the following: $A_1$=(0.24, 0.50, 0.83), $A_2$=(0.34, 0.65, 1.0), $A_3$=(0.11, 0.33, 0.64).

In order to rank the three fuzzy attraction strength values, we use the ranking method described in [36]. Define the fuzzy ideal positive value $P^*$=(1,1,1) and the fuzzy ideal negative value $P^-$=(0,0,0). Then, for every attraction $A_i$ compute the distances $D^*$ and $D^-$ between $A_i$ and $P^*$ and $P^-$, respectively. The distance between two fuzzy numbers A=($a_1$, $a_2$, $a_3$) and B=($b_1$,$b_2$,$b_3$) is the following:

$$D(A,B) = \sqrt{\frac{(a_1-b_1)^2+(a_2-b_2)^2+(a_3-b_3)^2}{3}} \qquad (4)$$

Then derive the closeness coefficient:

$$CC_i = \frac{D_i^-}{D_i^*+D_i^-} \qquad (5)$$

The higher closeness coefficient, the stronger attraction and better change decision.

Applying Equations 4-5 on our example results in Table 7. These results show that decision $D_2$ is subject to the strongest attraction from the software attributes and thus it is the best decision.

**Table 7. Distances and Closeness Coefficients**

| | $D^*$ | $D^-$ | CC |
|---|---|---|---|
| $D_1$ | 0.53 | 0.58 | 0.52 |
| $D_2$ | 0.43 | 0.56 | 0.56 |
| $D_3$ | 0.67 | 0.43 | 0.38 |

## 4. Empirical Study

A Java implementation of the proposed solution has been developed and submitted to a team of freelance software developers to test it. The respondents fed the solution with data of a Web application project they have recently delivered. A draft of the present paper (Sections 1-3) has been given to the developers. In the solution implementation, we chose to abstract the decision attributes into three attributes like presented in Table 2. The developers chose to use data about the most important changes they performed during that project, which happened to be accept/reject decisions, with only one implementation option in case the change were accepted. The developers ran the implemented tool with fuzzy evaluations like explained in Sections 2-3. They recorded the tool's decision and their own decision. They have been asked to evaluate how they judge the actual impact they noticed of any change they implemented, using the linguistic terms of Table 4 (columns 1 and 3). Besides, they have been required to tell whether they agree with the tool's output for every change case.

Table 8 summarizes the experiment's results. Six change cases have been assessed. As already mentioned, they all represent accept/ reject decisions.

**Table 8. Results of the Empirical Study**

| | Actions | Team's decision | Tool's decision | Actual impact | Tool is right? |
|---|---|---|---|---|---|
| Change 1 | Accept/ Reject | Accept | Reject | MB | Y |
| Change 2 | Accept/ Reject | Accept | Reject | B | Y |
| Change 3 | Accept/ Reject | Accept | Reject | VB | Y |
| Change 4 | Accept/ Reject | Accept | Accept | G | Y |
| Change 5 | Accept/ Reject | Accept | Accept | G | Y |
| Change 6 | Accept/ Reject | Accept | Accept | VB | N |

In cases 1-3 (50%), the team implemented the change but the actual impact was not satisfactory. In these three change cases the tool has rejected the change request and the team agrees with this decision. This leads to conclude that the tool outperformed the team in 50% of the change cases. In cases 4-5 (33%) the team implemented the change and the impact was actually satisfactory. The tool inferred the same decision and the team agrees with it. Therefore, in 33% of the change cases, the tool performed as good as the team. In case 6 (17%) the team implemented the change but the actual impact was not satisfactory. The tool inferred the same bad decision. The developers think that the tool made a bad decision just like they did.

These experimental results have the merit to show the potential of the proposed solution; the tool outperformed the team in 50% of the change cases. They cannot be considered as a final verdict, however, because of the very small sample size (six change decision cases) and the very specific test environment (one software project and one software team).

## 5. Conclusion

This article presented an automated solution for the software architectural change decision problem. It consists of an adapted version of the software characterization model called SACCS, and a change decision algorithm. The latter combines the fuzzy set theory and the Cognitive Attraction Theory (CAT). The SACCS model provides the software architectural attributes on which the possible impact of the requested change has to be fuzzily evaluated by the Change Control Board (CCB) members. The use of CAT allows the proposed algorithm to infer a common decision from the subjective and often incompatible assessments of the CCB members. To validate the proposed solution, an industrial case study has been performed where a team of software developers used an implementation of the solution. Results demonstrated an interesting potential.

We believe that there is room for improving the proposed change management solution in the future. Larger empirical utilization of the solution is required to accurately verify its performance compared with human professionals' performance. The choice of the membership functions and the fuzzy ranking method can also be revised to explore the possibility of further amelioration.

## References

[1] S. Kim, E. Whitehead, and Y. Zhang, Classifying Software Changes, IEEE Transactions on Software Engineering, 34, 2 **(2008),** pp. 181-196.

[2] L. Briand, Y. Labiche, L. O'sullivan, and M. Sûwka, Automated impact analysis of UML models, Journal of Systems and Software, 79, 3 **(2006)**, pp. 339–352.

[3] R. Conradi, M. Nguyen, and A. Wang, Planning support to software process evolution, International Journal of Software Engineering, Knowledge Engineering, 10, 1 **(2000),** pp. 31–47.

[4] I. Sommerville, Software Engineering, Addison-Wesley, New-York **(2011)**.

[5] S. Harker, K. Eason, and J. Dobson, The change and evolution of requirements as a challenge to the practice of software engineering. Proc. of the IEEE International Symposium on Requirements Engineering **(2003)**, March, pp.266-272.

[6] J. Ferzund, S. Ahsan, and F. Wotawa, Software change classification using hunk metrics. Proc. of the IEEE International Conference on Software Maintenance **(2009)**, June, pp.471-474.

[7] M. Stoerzer, B. Ryder, X. Ren, and F. Tip, Change Classification and Its Applications to Program Development and Testing. Technical Report DCS-TR-05-566, Department of Computer Science **(2005),** Rutgers University.

[8] A. Porter and R. Selby, Empirically guided software development using metric-based classification trees, IEEE Software Journal, 7, 2 **(1990)**, pp. 46-54.

[9] M. Shaw and D. Garlan, Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall, Washington **(1996)**.

[10] L. Briand and V. Basili, A classification procedure for the effective management of changes during the maintenance process. Proc. of the the Conference on Software Maintenance **(1992)**, pp.328–336.

[11] L. Briand, Y. Labiche, and L. O'sullivan, Impact analysis and change management of UML models. Proc. of the International Conference on Software Maintenance **(2003)**, pp.256–265.

[12] P. Weissgerber and S. Diehl, Identifying refactorings from source-code changes. Proc. of the IEEE/ACM International Conference on Automated Software Engineering **(2006)**, pp.231–240.

[13] N. Madhavji, The prism model of changes. Proc. of the International Conference on Software Engineering **(1991)**, pp.166–177.

[14] R. Fiutem and P. Tonella, A cliche-based environment to support architectural reverse engineering. Proc. of the Working Conference on Reverse Engineering **(1996)**, pp.319–328.

[15] J. Grundy and J. Hosking, High-level static and dynamic visualisation of software architectures. Proc. of the IEEE International Symposium on Visual Languages **(2000)**, pp.5–12.

[16] J. Nedstam, E. Karlsson, and M. Host, The architectural change process. Proc. of the International Symposium on Empirical Software Engineering **(2004)**, pp.27–36.

[17] T. Feng and J. Maletic, Applying dynamic change impact analysis in component-based architecture design. Proc. of the International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/ Distributed Computing **(2006)**, pp.43–48.

[18] R. Arnold and S. Bohner, Impact analysis-Towards a framework for comparison. Proc. of the Conference on Software Maintenance **(1993)**, pp.292-301.

[19] L. Chen L and B. Xu, Slicing Java Generic Programs Using Generic System Dependence Graph, Wuhan University Journal of Natural Sciences, 14, 4 **(2009)**, pp. 304-308.

[20] S. Horwitz, T. Reps, and D. Binkley, Inter-procedural slicing using dependence graphs. Proc. of the ACM SIGPLAN Conference on Programming Language Design and Implementation **(1998)**, pp.35-46.

[21] B. Ryder and F. Tip, Change impact analysis for object-oriented programs. Proc. of the ACM Workshop on Program Analysis for Software Tools and Engineering **(2001)**, pp.46–53.

[22] J. Zhao, Change impact analysis for aspect-oriented software evolution. Proc. the International Workshop on Principles of Software Evolution **(2002)**, pp.108–112.

[23] A. Orso, Leveraging field data for impact analysis and regression testing. Proc. of the ACM SIGSOFT Symposium on Foundations of Software Engineering **(2003)**, pp.128–137.

[24] J. Law and G. Rothermel, Incremental dynamic impact analysis for evolving software systems. Proc. of the ISSRE **(2003)**, pp.430–441.
[25] J. Law and G. Rothermel, Whole program path-based dynamic impact analysis. Proc. of the International Conference on Software Engineering **(2003)**, pp.308–318.
[26] G. Avellis, CASE support for software evolution: a dependency approach to control the change process. Proc. of the International Workshop on Computer-Aided Software Engineering **(1992)**, pp.62-73.
[27] J.S. O'Neal and D. Carver, Analyzing the impact of changing requirements. Proc. of the IEEE International Conference on Software Maintenance **(2001)**, pp.190-195.
[28] L. Lavazza and G. Valetto, Enhancing requirements and change management through process modelling and measurement. Proc. of the International Conference on Requirements Engineering **(2000)**, pp.106-115.
[29] J. Hewitt and J. Rilling, A light weight proactive change analysis using use case map. Proc. of the International Workshop on Software Evolvability **(2005)**, pp.12-28.
[30] J. Hassine, J. Rilling, and H. Hewitt, Change impact analysis for requirement evolution using use case maps. Proc. of the International Workshop on Principles of Software Evolution **(2005)**.
[31] L. Tahvildari, R. Gregory, and K. Kontogiannis, An approach for measuring software evolution using source code features. Proc. of the Asia Pacific Software Engineering Conference **(1999)**, pp.10–17.
[32] P. Bengtsson and J. Bosch, Architecture level prediction of software maintenance. Proc. of the EuroMicro Conference on Maintenance and Reengineering **(1999)**, pp.139–147.
[33] B. Wlliams and J. Carver, Characterizing software architecture changes: A systematic review, Information and Software Technology, 52, 3 **(2010)**, pp. 31-51.
[34] Z. Chentouf, Homo Informaticus, L'Harmattan, Paris **(2000)**.
[35] Z. Chentouf, Cognitive Attraction Theory and Moral Judgment, Psychology, 4, 1 **(2013)**, pp. 38-43.
[36] H. Chang, C. Wei, and R. Lin, Model for Selecting Product Ideas in Fuzzy Front End, Concurrent Engineering, 16, 2 (**2008**), pp. 121-128.