International Journal of Engineering &Technology, 7 (3.12) (2018) 963 -967

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Software Change Management: a Quantified Perspective

Ankit Dhamija[1], Sunil Sikka[2]

[1]*Research Scholar*, [2]*Associate Professor,*
[1,2]*Amity School of Engineering & Technology, Amity University Gurgaon, Haryana*
**Corresponding Author Email:* [1]*adhamija@ggn.amity.edu* , [2]*ssikka@ggn.amity.edu*

Abstract

A systematic Change Impact Analysis (CIA) is being used for better change management of software. Also, CIA process is evolved continuously to make it more effective. Software metrics play an important role to evaluate CIA process. Two types of metrics are used to evaluate CIA. First types of metrics are the standard metrics used to evaluate the performance of CIA techniques for example Precision, Recall, F-measure etc. These are most commonly used by researchers. Second types of metrics are those which are used to quantify the change impact which is based on the code/design features. This paper is aimed at identification of these second types of metrics available in literature.

Keywords*: Change Impact Analysis, Software Metrics.*

1. Introduction

A very important activity in software development is Change Impact Analysis. It is a step by step process of analyzing the probable impact of a change on the whole software. While some of these changes are suggested by the clients, some are also unearthed by the developers themselves, some by the maintenance team. Thus, it becomes very important to treat each change request one by one and perform a careful change impact analysis to estimate the software artifacts that are going to be impacted by this change implementation. Some tough decisions like "whether to ignore or consider this change request?" or "how many artifacts are getting affected?" or "whether the quality of software will be improved after change implementation?" or "whether the change implementation will bring in adverse/ripple effects?", and so on, needs to be taken by the maintenance team. However, due to lack of time and short release cycles, this activity is not given required attention that it needs. This results in the release of faulty software with bugs. Therefore, a systematic Change Impact Analysis must be carried out. Figure 1 presents a systematic CIA process; it starts with the Change Set which includes the tentative impacted areas in source code that may be affected due to change introduction. Thereafter, the set for estimated change impact (EIS), a set for actual change impact (AIS), a set denoting over-estimation of Impacts called False Positive Impact Set (FPIS), a set denoting under-estimation of impacts called False Negative Impact Set (FNIS) are created and efforts are done to bring the difference between EIS and AIS to zero.

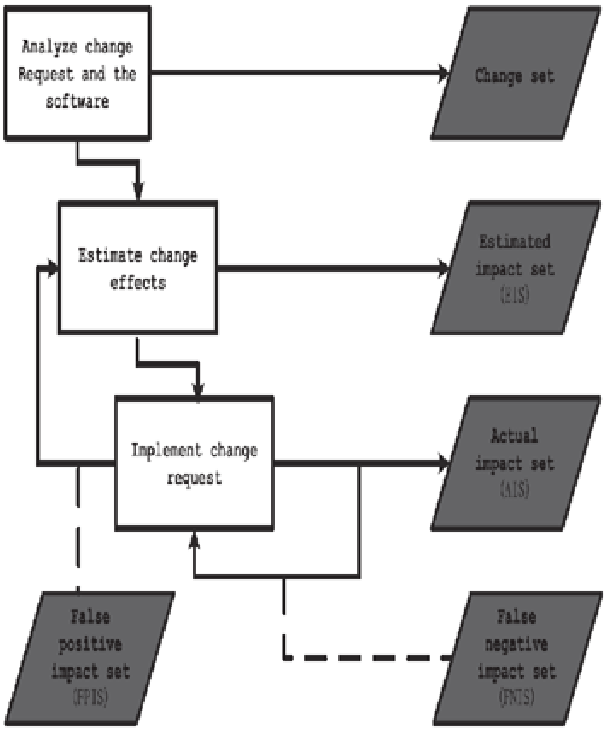

Fig. 1: Systematic CIA Process [1]

To achieve high level of accuracy, many CIA approaches such as traceability and dependency analysis are available. Traceability analysis may be requirements based, structural and knowledge based and implicit or explicit [2, 3]. Dependency analysis may be static, dynamic or hybrid [4-8].
Most of the proposed techniques have been evaluated for accuracy and effectiveness using software metrics. Software metrics are the measure of software characteristics which are quantifiable or

countable [9]. These are helpful in measuring the performance of the software, measuring its productivity, planning work items etc.
There are two types of metrics used in CIA related research. First types of metrics are used to assess how CIA techniques are performing. This is judged using precision and recall metrics. On the other hand second types of metrics are used to quantify the effect of a change. The second types of metrics are not much focused upon by the researchers. However these metrics can play a significant role in evaluating a CIA technique and finding out the possible impact a change will have on the whole system. Therefore, this motivates the author(s) to discover the second types of metrics for CIA from the literature.
The remaining paper is structured into three sections. Section II covers the literature review of various CIA metrics where a comparative analysis of recent work done in the field is presented. Section III concludes the paper and Section IV presents the references.

2. Literature Review

Many authors have proposed various metrics to quantify the CIA. These metrics are discussed in this section.
Pfleeger and Bohner [10] proposed two metrics named horizontal traceability metrics (HTM) and vertical traceability metrics (VTM) to identify the potentially impacted workproducts. Authors used directed graphs of software lifecycle objects to establish a relationship between requirements, design, code and test procedures to determine horizontal and vertical traceability links and created various impact analysis metrics to address traceability dependencies. For VTM, they used the characteristics of the vertical traceability graph like size (including the number of nodes, the in degree and out degree) and complexity (cyclomatic complexity) to assess the vertical traceability changes. For HTM, the workproducts associations and the way they are related to process was utilized. Various relationship graphs were created which were measured for size and complexity. Thus, their approach works by measuring the graph characteristics of the primary workproducts and the change effect. The metrics proposed by them are summarized in Table 1.

Table 1: Horizontal and Vertical Traceability Metrics

Metrics Category	Metrics
Vertical Traceability Metrics	**Product Metrics: Size & Complexity**
	Complexity within each workproduct: Cyclometic complexity.
	Size within each workproduct: counting the number of nodes (requirement components, design components, code components etc) Node Degree (No. of edges)- In degree and Out degree
Horizontal Traceability Metrics	Process Metrics: Relationship graphs, size & complexity
	Measuring relationships among workproducts through Relationship Graphs for size & complexity
	Defining a Tracing path for Horizontal Traceability Graph

Thereafter Zhang et al [11] proposed change metric suite for AspectJ programs to measure the change impacts during software evolution. They defined a terminology for Aspect oriented (AO) system to measure the explicit change impact and global change impact that included definition(s) of Aspect oriented system and its ancestors, the modules of an aspect and a class, attributes of aspects and classes, addressing changes in Aspect Oriented software that included adding, deleting and modifying module changes and attribute changes. Thereafter, they proposed 28 *Explicit impact metrics* and 06 *Global impact metrics* where the explicit impact metrics were used to measure the direct impact of adding, deleting or modifying a program element such as the addition of an aspect, a piece of advice, an intertype declaration, or a method. The Global impact metrics were used to evaluate the overall change impact in the system level. A common feature among all the metrics was that the larger is the value of a metric, the wider is the impact the changes have. Table 2 defines some of the metrics for adding changes in Aspect Oriented Software.

Table 2: A catalogue of Adding Change Metrics in AO software

Metrics	Definition
M1	Sum of highest path length from aspect a to the aspect hierarchy root in the set.
M2	Sum of highest path length from class c to the class hierarchy root in the set.

Wong and Gokhale [12] proposed distance metrics for code analysis based on features to ascertain the features' closeness. They first defined some acronyms and definitions, presented the mathematical equations to compute the distance and examined it using the following metric in Equation 1:

$$DIST\alpha\beta = |B\alpha\oplus\beta| / |B\alpha\cup\beta| \quad (1)$$

- α and β =Features of Program P.
- $B_{\alpha\oplus\beta}$ = Set of blocks in either B_α or B_β, but not both, i.e., $B_{\alpha\oplus\beta}$ equals $(\overline{B}_\alpha \cap B_\beta) \cup (B_\alpha \cap \overline{B}_\beta)$ where $\overline{B}_\alpha$ and $\overline{B}_\beta$ are the complements of B_α and B_β in the set of blocks in P, respectively, $\overline{B}_\alpha \cap B_\beta$ contains the blocks in B_β but not in B_α and $B_\alpha \cap \overline{B}_\beta$ contains the blocks in B_α but not in B_β.
- $B_{\alpha\cup\beta}$ =Set of blocks in the union of B_α and B_β.
- $DIST_{\alpha\beta}$ = Distance between features $_\alpha$ and $_\beta$.

I song et al [13] proposed a model to find out the risk associated with a change request and also further related that risk with fault proneness. This is found out using three CK software metrics: response for a class (RFC), coupling between objects (CBO) and Depth of Inheritance (DIT) in Equation 2:

$$(1/\text{Class FP}) = (-37.124 + 2.938(\text{CBO}) + 0.2(\text{RFC}) + 2.214(\text{DIT})) \quad (2)$$

They stated that a more than 50 % risk will make a class fault prone. They developed CCRecommender tool which quantifies the associated risks while changes are being made in a class which is prone to faults, in the impact set before the actual change implementation. Depending up on the risk value produced, the tool displays a specific color to emphasize the quantum of risk.
Salman et al [14] used Formal Concept Analysis (FCA) based on lattice theory for Software Product Line (SPL) to propose a feature level CIA technique for predicting the affected features for change management. They proposed two metrics *Impact Probability Metric (IDM)* and *Changeability Assessment Metric (CAM)* to be precise. While IDM measured the extent to which a particular feature may be affected, CAM metric finds out the fraction of features which are affected by a given change. Features which have high IDM values have a high likelihood of being affected. Figure 2 shows the equation to compute IDM where, the IDM value of the affected feature F is calculated by taking the intersection of I (the classes/intent of a lattice concept) having F as an extent and the impact set.

$$\text{IDM (F)} = [|\{I\} \cap \{\text{IMPACT SET}\}| / |\{I\}|] * 100\% \quad (3)$$

$$\text{CAM} = (\#\text{Affected Features}/\#\text{All Features}) * 100\% \quad (4)$$

The equation for computing CAM is presented in Equation 3 and 4. The Affected Features signify a set of potentially affected features given by a class impact set. All Features correspond to total features. The higher CAM values represent the sensitivity of features with respect to a proposed change request. Three case studies were used to perform the experimental evaluation of the technique of diverse sizes and domains.

Oliva et al [15]'s work included the evolution of workflow repositories where they proposed a CIA approach based on metrics and visualization. Two metrics namely *change scattering and impact* were proposed by them to understand the relationships between workflows where:

Scattering(F_i) = Number of *possibly affected* flows when a change is introduced in a particular workflow. (5)

Impact(F_i,*p*) *of a flow* F_i = Quantity of flows having a *high probability* of getting impacted when a particular flow is changed.

Thus, $Impact(F_i,p) \leq Scattering(F_i)$. (6)

They also defined some color code where Red denotes High Scattering & High Impact; Green denotes Low Scattering and Low Impact and so on. Their results showed that repositories vary substantially in size, number and percentage of flows and their approach improves the flexibility & reliability of workflow repositories.

Maazoun et al [16] proposed a novel method for change management in Software Product Lines by analyzing the evolution of feature model and then tracing its impact on the design of SPL. They proposed 18 new metrics based on CK metrics suite to find out the effort required for managing the change impact. They proposed 18 Change Impact metrics related to a feature, adding a feature and for removing a feature. These are listed in Table 3. Their results depict the high quality of feature models generated after evolution.

Table 3: CI Metrics related to features

CI metrics related to a feature	
Metrics	**Definition**
NF	Counts the number features in a feature model
FNOP	Counts the number of packages in a feature
FNOC	Counts the number of classes in a feature
FNOM	Counts the number of methods in a feature
FNOA	Counts the number of attributes in a feature
FNOAs	Counts the number of associations in a feature
CI metrics to add a feature	
NF_added	Counts the number of added features
FNOP_added	Counts the number of packages added in a feature
FNOC_added	Counts the number of classes added in a feature
FNOM_added	Counts the number of methods added in a feature
FNOA_added	Counts the number of attributes added in a feature
FNOAs_added	Counts the number of associations added in a feature
CI metrics to remove a feature	
NF_removed	Counts the number of removed features
FNOP_removed	Counts the number of packages removed in a feature
FNOC_removed	Counts the number of classes removed in a feature
FNOM_removed	Counts the number of methods removed in a feature
FNOA_removed	Counts the number of attributes removed in a feature
FNOAs_removed	Counts the number of associations removed in a feature

Zhifei et al[17] proposed the use of metric based detection approaches for detecting Python code smells and their relation with changes and faults. They used 05 generic code smells and 05 additional never studied code smells and associated various code smell metrics with them. Eight new metrics are defined by the authors. Table 4 presents the smells and associated metrics.

Thereafter they applied threshold based filters; statistics based filters and turing machine filters, and obtained the threshold values. In the empirical study done by them, their results indicated that the code smell detection method based on metrics performs well in detecting bad smells in Python.

Table 4: Code Smells and Metrics

Smell	**Metrics**
LPL	PAR- NO. OF PARAMETRES
LM	MLOC- METHOD/FUNCTIONLINES OF CODE
LSC	DOC-DEPTH OF CLOSURE
LC	CLOC-CLASS LINE OF CODE
LMC	LMC-LENGTH OF MESSAGE CHAIN
LBCL	NBC- NO OF BASE CLASSES
LLF	NOC- NO OF CHARACTERS
LTCE	NOL- NO OF LINES
CCC	NOO- NO OF OPERATORS AND OPERANDS
MNC	LEC- LENGTH OF ELEMENT CHAIN

Choudhary et al [18] investigated the effect of change metrics on software fault prediction by using 16 existing change metrics and defining 20 new change metrics.

Using these metrics and machine learning algorithms, they build fault prediction models. Table 5 presents some of the new change metrics that were proposed by them:

Table 5: New Change Metrics

Metrics	**Definition**
LOC-WORKED-ON	Lines of code added plus lines of code deleted
MAX_COMMITS	Maximum no of commits made by developer
MAX_LOC_DELETED	Maximum lines of code deleted by developer
MAX_CODECHURN	Maximum CODECHURN by a developer over all developers

Then, they collected experimental data from GIT repositories and performed experiments on WEKA platform using K-Nearest Neighbor (KNN), Decision Tree (J48), and Random Forest (RF) classifiers. Precision, Recall and F-Measure metrics were used for performance evaluation. Their outcome stated that new change metrics enhances the model's performance & ensure development of fault predictors with high-performance. Also, with new metrics, 10% increase in recall is witnessed when compared with static code based metrics and about 23% above already existing metrics.

Kumar et al [19] used 62 source code metrics that included metrics related to size(7), cohesion(18), coupling(20) and inheritance(17) for building a change-proneness prediction model.

Then they used various machine learning algorithms and ensemble techniques to measure the effectiveness of metrics. Their results show much better results when the model is created using some selected set of source code metrics by taking any feature selection technique as input rather than taking all source code metrics. Furthermore, the model based on change-proneness reflected superior results as compared with other dimension metrics.
A review of the above mentioned work is presented in Table 6.

Table 6: Review of the Existing Work

Title	Target	Focus	Contribution	Evaluation	Tool	Reference
A Framework for Software Maintenance Metrics	Software Maintenance Process Models	Software Change Management	A new Software Maintenance Process Model. Vertical and Horizontal Traceability Metrics	Example	No	(Pfleeger and Bohner, 1996)
Metrics for Measuring Change Impacts in AspectJ Software Maintenance and Reuse	Aspect Oriented Systems	A change metrics suite for AO software.	28 Global Impact Metrics and 06 Explicit Impact Metrics	Experimental. Empirical study on seven AspectJ benchmarks.	Cemeta	(Zhang et al.,2008)
Static and dynamic distance metrics for feature-based code analysis	Feature Based Code Analysis	Numerical Example for computing the distance between features.	Distance Metrics	Experimental	Case Study on SHARPE	(Wong and Gokhale,2005)
Supplementing Object Oriented Software Change Impact Analysis with Fault proneness Prediction	Impact Analysis with Fault-proneness Prediction	A model for predicting Fault Proneness	Utilized CK metric suite	Descriptive Statistics and Binary Logistic Regression	CCRecommender	(Isong et al., 2016)
Feature-Level Change Impact Analysis Using Formal Concept Analysis	Software Product Line Engineering	Formal Concept Analysis	Impact Probability Metric (IDM) and Changeability Assessment Metric (CAM)	Experimental using Case Studies		(Salman et al., 2015)
A Static Change Impact Analysis Approach based on Metrics and Visualizations to Support the Evolution of Workflow Repositories	Workflow Management Systems	Workflow repositories	CIA metrics- Change scattering and Impact	Experimental and Exploratory	Approach implemented as a Java 2 SE library	(Oliva et al.,2016)
Change impact analysis for software product lines	Software Product Line	Analyzing feature model evolution and tracing their impact on the SPL design	18 CIA metrics corresponding to feature addition and deletion.	Experimental	Evo-SPL Tool	(Maazoun et al., 2016)
Understanding metric-based detectable smells in Python software: A comparative study	Python software	Metric-based detection method for code smells	Define 08 metrics to quantify best the symptoms of additional code smells	Experimental Empirical	Pysmell	(Zhifei et al.,2018)
Empirical Analysis on Effectiveness of Source Code Metrics for Predicting Change-Proneness	Source Code Metrics, Feature Selection Techniques	Change-proneness prediction	Usage of 08 learning algorithms to develop a change proneness model	Experimental and Empirical	No	(Kumar et al., 2017)
Empirical analysis of change metrics for software fault prediction	Eclipse Projects, GIT repositories	Using change metrics and code metrics to improve the performance of fault prediction models.	20 new Change Metrics	Experimental and Empirical	No	(Choudhary et al., 2018)

3. Conclusion

Quantifying change impact is an important area in the field of software metrics. Recently researchers have proposed metrics for quantifying change impact based on Aspect Oriented Systems, Software Product Line Engineering, and Workflow Management Systems etc.

This paper reviews various metrics proposed by many researchers and observed that only few of the metrics are available to evaluate the change impact. Moreover, the metrics proposed are not validated. So there is a need of standard set of metrics to quantify the CIA. Also it is required to validate the existing standard metrics in context to CIA.
Some of the researchers have also proposed tools like CEMETA, Pysmell, Evo-SPL, CCRecommender etc. which are specific to their work. Also, the tools work only on certain types of inputs

which don't fit in every context. Further research can be done for detailed study of various Change Impact Tools.

References

[1] Bohner SA. Software change impacts—an evolving perspective. Proceedings of the International Conference on Software Maintenance, Montréal, Canada, 2002; 263–272.

[2] De-Lucia A, Fasano F, Oliveto R. Traceability management for impact analysis. Proceedings of the International Conference on Software Maintainence, Beijing, China, 2008; 21–30.

[3] M. Kilpinen. The Emergence of Change at the Systems Engineering and Software Design Interface - An Investigation of Impact Analysis. PhD thesis, Cambridge University, Engineering Department, 2008.

[4] Badri L, Badri M, Yves SD. Supporting predictive change impact analysis: a control call graph based technique. Proceedings of the Asia-Pacific Software Engineering Conference, Taipei, Taiwan, China, 2005; 167–175.

[5] Mund GB, Mall R. An efficient interprocedural dynamic slicing method. Journal of Systems and Software 2006; 79(6):791 806.

[6] Hewitt J, Rilling J. A light-weight proactive software change impact analysis using use case maps. Proceedings of the International Workshop on Software Evolvability, Budapest, Hungary, 2005; 41–46.

[7] Huang LL, Song YT. Precise dynamic impact analysis with dependency analysis for object-oriented programs. Proceedings of the International Conference on Advanced Software Engineering and Its Applications, Hainan Island, China, 2008; 217–220.

[8] Apiwattanapong T, Orso A, Harrold MJ. Efficient and precise dynamic impact analysis using execute-after sequences. Proceedings of the International Conference on Software Engineering, St. Louis, Missouri, USA, 2005; 432–441.

[9] W. Frakes and C. Terry. Software reuse: Metrics and models. ACM Computing Surveys, 28(2):415–435, 1996.

[10] Pfleeger SL, Bohner SA. A Framework for software maintenance metrics. Proceedings of the International Conference on Software Maintenance, Washington, DC, 1990; 320–327.

[11] Zhang, S., Shen, H., and Zhao, J. 2008. Metrics for Measuring Change Impacts in AspectJ Software Maintenance and Reuse. Technical Report. Center for Software Engineering, SJTU.

[12] W.E. Wong, S.S. Gokhale, "Static and Dynamic Distance Metrics for Feature-Based Code Analysis", J. Systems and Software, vol. 74, no. 3, pp. 283-295, 2005.

[13] B. Isong, O. Ifeoma and M. Mbodila, "Supplementing Object-Oriented software change impact analysis with fault-proneness prediction", 15th International Conference on Computer and Information Science (ICIS' 16), IEEE Computer Society, pp. 1--8, Okayama, Japan, 26-29 June 2016.

[14] Hamzeh Eyal Salman, Abdelhak-Djamel Seriai, and Christophe Dony, Int. J. Soft. Eng. Knowl. Eng. 25, 69 (2015).

[15] Gustavo Ansaldi Oliva, Marco Aurélio Gerosa, Fabio Kon, Virginia Smith and Dejan Milojicic, "A Static Change Impact Analysis Approach based on Metrics and Visualizations to Support the Evolution of Workflow Repositories", International Journal of Web Services Research Volume 13 • Issue 2 • April-June 2016, pp 74-103.

[16] Jihen Maâzoun, Nadia Bouassida, and Hanêne Ben-Abdallah. Change impact analysis for software product lines. J. King Saud Univ. Comput. Inf. Sci., 28:364– 380, Oct 2016.

[17] Chen Zhifei, Chen Lin, Ma Wanwangying, Zhou Xiaoyu, Zhou Yuming, Xu Baowen, "Understanding metric-based detectable smells in Python software: A comparative study, Information and Software Technology 94 (2018) pp 14–29

[18] Garvit Rajesh Choudhary, Sandeep Kumar, Kuldeep Kumar, Alok Mishra, Cagatay Catal, "Empirical analysis of change metrics for software fault prediction", Computers and Electrical Engineering 67 (2018) pp 15–24.

[19] Lov Kumar, Santanu Kumar rath, Ashish Sureka, "Empirical Analysis on Effectiveness of Source Code Metrics for Predicting Change-Proneness", ISEC '17, February 05-07, 2017, Jaipur, India.