A Change Impact Analysis Tool: Integration Between Static and Dynamic Analysis Techniques

Nazri Kama[1(✉)], Saiful Adli Ismail[1], Kamilia Kamardin[1], Norziha Megat Zainuddin[1], Azri Azmi[1,2], and Wan Shafiuddin Zainuddin[2]

[1] Advanced Informatics School, Universiti Teknologi Malaysia, 54100 Kuala Lumpur, Malaysia
{mdnazri, saifuladli, kamilia, norziha.kl, azriazmi}@utm.my
[2] CyberSecurity Malaysia, Sapura@Mines, 43300 Seri Kembangan Selangor, Malaysia
wanshafi@cybersecurity.my

Abstract. Accepting too many software change requests could contribute to expense and delay in project delivery. On the other hand rejecting the changes may increase customer dissatisfaction. Software project management might use a reliable estimation on potential impacted artifacts to decide whether to accept or reject the changes. In software development phase, an assumption that all classes in the class artifact are completely developed is impractical compared to software maintenance phase. This is due to some classes in the class artifact are still under development or partially developed. This paper is a continuous effort from our previous work on combining between static and dynamic analysis techniques for impact analysis. We have converted the approach to an automated tool and call it a CIAT (Change Impact Analysis Tool). The significant achievements of the tool are demonstrated through an extensive experimental validation using several case studies. The experimental analysis shows improvement in the accuracy over current impact analysis results.

Keywords: Software development · Change impact analysis · Impact analysis

1 Introduction

Managing software changes is crucial in meeting the evolving needs of customers and later, satisfying their requirements [1]. On one hand taking risk by accepting huge number of changes will lead to delay in delivering project deadline. On the other hand, rejecting the changes contribute to customers unsatisfactory. Looking at this scenario, it is a challenge for software project manager to make a decision when software changes occur during software development. One type of inputs that can assist the software project manager to make an effective decision is through an early prediction on the number of impacted artifacts (or classes) by the changes. The prediction can be done by performing impact analysis or change impact analysis [2].

H. Fujita and G. Guizzi (Eds.): SoMeT 2015, CCIS 532, pp. 413–424, 2015.
DOI: 10.1007/978-3-319-22689-7_32

Referring to [3–5], impact analysis is defined as a process of investigating potential consequences of making a change, or estimating what are the artifacts that will be affected to accomplish a change. In other words, the impact analysis is an activity of identifying software artifacts that are potentially to be affected by a change. Impact analysis has been widely used in software maintenance phase rather than software development phase [6–9]. This is because the current developed impact analysis solutions assume that all classes or class artifacts are completely developed. Most solutions use dynamic analysis techniques [10–13] for their impact analysis implementation.

Our previous works [6–9] have shown that there is a clear difference on change impact analysis implementation between software maintenance and software development phases. This is due to the existence of partially developed classes in the software development phase. This existence causes the current implementation of dynamic analysis techniques are impractical to be implemented in the software development phase. The dynamic analysis technique uses method execution path model as a source of analysis. This model is developed through reverse engineering from source code [10–13]. The technique tends to produce inaccurate results because some method execution paths that involve partially developed classes [7, 8] are not visible due to they have yet to be implemented. This will led to inaccuracy of the generated results.

This paper is a continuous works on change impact analysis approach to support software development activity [6–9]. To note, this paper has close related to our newly published work in [8]. The difference is that this paper focuses on our experiment in automating the previously developed manual impact analysis approach whereas in [8] we extend the automated approach to support change effort estimation. In few recent studies [14, 15], the combination of static and dynamic approaches has indicated some noticeable advantages from both worlds. In this paper, we have extended our work to developing a prototype tool to support the previously developed approach. This paper will give more explanation on the developed tool rather than the concept of change impact analysis approach itself. Details explanation on the approach can be found in [6–9].

This paper is presented: Sect. 2 related work, Sect. 3 explanation on the prototype tool main screen, Sect. 4 and Sect. 5 provide explanation on evaluation procedure and its results. Lastly, Sect. 6 concludes and position our future works.

2 Related Work

Based on our literature, impact analysis has two categories that are static analysis technique and dynamic analysis technique. Our previous definitions have said that the static analysis technique generates a set of potential impacted classes from software artifacts. The dynamic analysis technique conversely builds a set of potential impacted classes through source code execution.

2.1 Static Analysis

Two most related existing static analysis techniques are selected as comparative to the new proposed approach which are the Use Case Maps (UCM) technique [16] and the class interactions prediction with impact prediction filters (CIP-IPF) technique [17, 18].

The UCM technique perform impact analysis on the functional requirements and the high level design models when all the functional requirements have been completely identified and the high level design models have been fully developed. Nevertheless, the main limitation of this technique is there is no traceability link between the functional requirements and the high level design models to the actual source codes. This technique only makes an assumption that the content of these two artifacts are reflected to the class artifacts in which any affected elements in the UCM models are indirectly reflected to the affected class artifacts.

Next, CIP-IPF technique [17, 18] uses the class interactions prediction model in order to define the impacted class artifacts. The advantage of the technique over the UCM technique is that it has a traceability link between the requirements artifacts and the class artifacts. Impact of changes at the requirement level to the class artifacts can be performed based on this traceability link.

In these two techniques, there is a tendency of missing some actual impacted class due to inconsideration of actual source code analysis. This is based on the precept that some of the effect of a change from a class to other classes may only be visible through dynamic or behavior analysis of the changed class [19, 20].

2.2 Dynamic Analysis

Two most related dynamic analysis works are identified in our research which are the Influence Mechanism technique [11, 13] and the Path Impact technique [12]. These two techniques analyzing the actual source code in order to predict the impact set which consists of classes or methods.

The Influence Mechanism technique [11, 13] introduces the Influence Graph (IG) as a model to identify the impacted classes. However, this technique only analyze the class artifacts as the only source of analysis with the condition that the source code are completely developed.

Next, the Path Impact technique [12] uses the Whole Path DAG (Directed Acyclic Graph) model as a model to identify the impacted classes. The technique is almost similar to the Influence Mechanism technique as this technique also uses the class artifacts as a source of analysis and assumes that the class artifacts are completely developed. In addition to that, this technique also performs a preliminary analysis prior to performing a detail analysis. The main limitation of this technique is the implementation is time consuming as the technique opens to a huge number of data when the analysis goes to a large application.

The main similarity of both techniques in terms of its limitation is there is no traceability process or formal mapping from requirements artifacts or design models to the class artifacts. This process is crucial in impact analysis process as changes not only come from class artifacts but it may also come from design and/or requirements

artifacts. Since design and requirements artifacts do interact among them vertically (between two different artifacts of a same type) and horizontally (between requirement and design artifacts), changes that happen to them could contribute to different affected class artifacts. In some circumstances, focusing on the source code analysis may not able to detect those affected classes.

3 The CIAT

There are two main modules in the Change Impact Analysis Tool (will called "CIAT" henceforth). The modules are Class Interaction Prediction (CIP) Module and Impact Analysis Module as in Fig. 1.

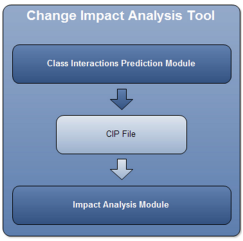

Fig. 1. System overview

The CIP model is a traceability model that shows interactions of all the software artifacts. This model will be used for static change impact analysis implementation. This model can be developed in any format, but for the purpose of this project we developed it for a consistent xml format. It consists of two sections, first section contains the project information and the second section contains the artifacts information.

For the impact analysis module, the process begins with performing static impact analysis to identify direct and indirect impacted classes. The process starts with the static impact analysis identify direct impacted classes which are the first layer of classes affected by a particular changes requirement. To note, this layer has yet included vertical traceability analysis. Later, the identification of indirect impacted classes are executed by performing the vertical traceability analysis.

Due to space limitation, this paper concentrates on impact analysis module only. Basically, the interaction between these two modules is done through an interface file named CIP. This file will be exported from Class Interactions Prediction Module and imported by Impact Analysis Module. In other words the output of the first CIP Module acts as an input for impact analysis module.

Overall functionalities of this tool are: (1) To import the CIP file to the system database; (2) To acquire the change request information; and (3) To perform static and dynamic impact analysis. See Fig. 2:

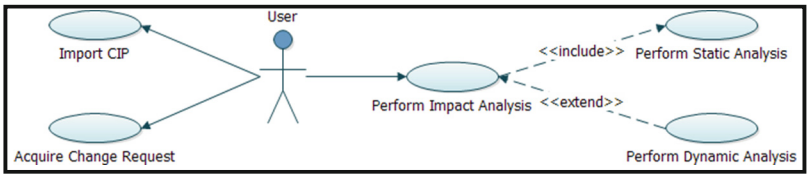

Fig. 2. Overall functionalities of CIAT

The following sub-sections explain briefly the implementation steps of this tool as shown in Fig. 3:

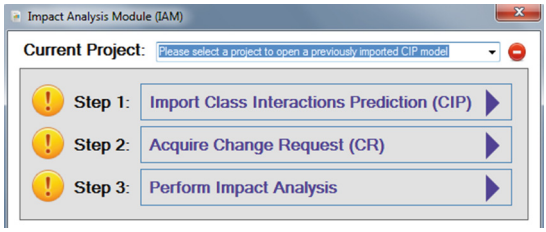

Fig. 3. Main page form

3.1 Step 1: Import CIP

The purpose of this procedure is to import the generated CIP model file. The CIP model file can be an XML (.xml) file or a CIP (.cip) file. Both should have been developed according to the CIP descriptions in XML format. The tool will extracts the software artifacts: (1) requirement, (2) design and (3) class information in the CIP model file as in Fig. 4 below.

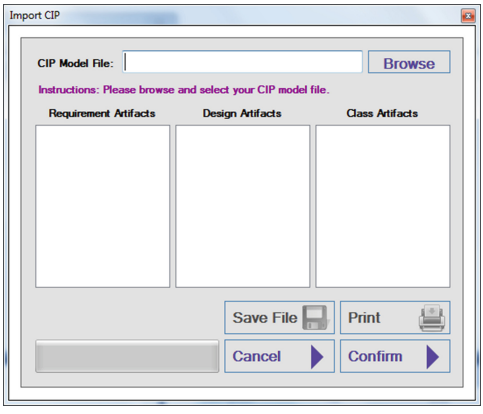

Fig. 4. Import CIP form

3.2 Step 2: Acquire Change Requests

The purpose of this step is to get the required change request information, see Fig. 5. Acquiring change request will be performed in Acquire Change Request Form (see Fig. 7). There are seven inputs in Acquire Change Request Form as follow: (1) Identification Number, which is filled automatically; (2) Change Requester; (3) Requested Change; (4) Change Priority; (5) Affected Requirements; and (6) Comments, which is optional.

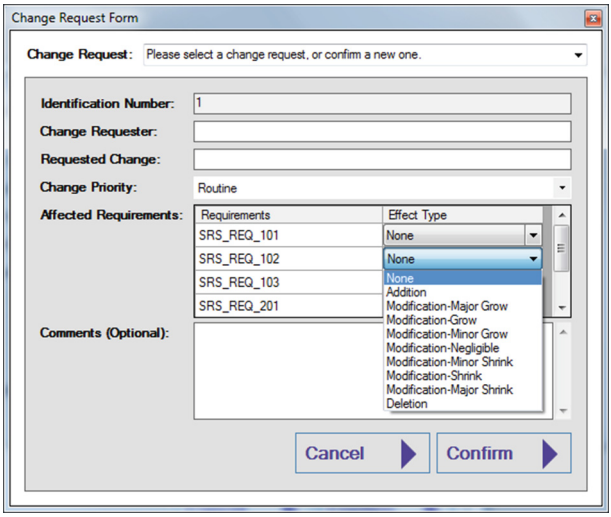

Fig. 5. Change request form

3.3 Step 3: Impact Analysis Implementation

This step analyses change request document to identify a set of potential impacted classes. The result of this step is the initial set of impacted classes. After identifying the initial set of impacted classes, the tool performs impact analysis using two filtration levels and they are Class Dependency Filtration (CDF) and the Method Dependency Filtration (MDF) levels.

In brief, the CDF analyzes the initial set of potential impacted classes using static analysis technique. This analysis is implemented in order to identify the existence of any interaction links that has no change impact value in the initial set of potential impacted classes. We define this interaction link as if there is no change happens to one side of two interacting classes, the other class will not be affected because the opposite class does not require the changed class for its implementation. For the MDF level, the following section gives explanation on it.

Static Analysis – Class Dependency Filtration Level. In this level, the class dependency filtration (CDF) will be performed on the static impact analysis results as in Fig. 6. The process begins with performing static impact analysis to identify direct and indirect affected classes; the static impact analysis will firstly find the direct impacted classes which are the first layer of classes affected by a particular changes requirement without vertical traceability relations consideration. Then indirect impacted classes will be identified by complete traceability search through the CIP interactions to find all related classes to the changed requirement.

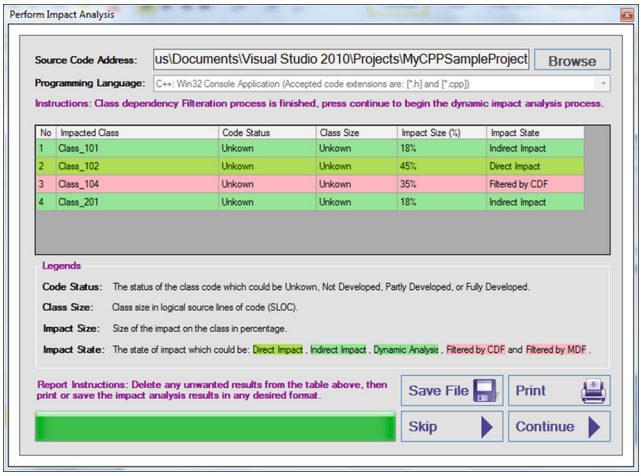

Fig. 6. Sample of CDF filtration on impact analysis

Dynamic Analysis – Method Dependency Filtration (MDF). The MDF conducts another layer of filtration on the CDF level outcomes or results. All method execution paths from the CDF level results will be extracted and analyzed to remove any false detected impacted classes. We have selected the backward and forward analysis technique [13].

There is one main challenge of the current dynamic impact analysis approaches from the software development phase perspective which is they do not consider partially developed class in their process. This is happening due to the nature of classes in the software maintenance phase have been fully developed.

We claim that the inclusion of partially developed class analysis plays significant role in impact analysis from software development perspective. A situation might exist in the software development phase where of some classes are still under development. Figure 7 shows the dynamic impact analysis form.

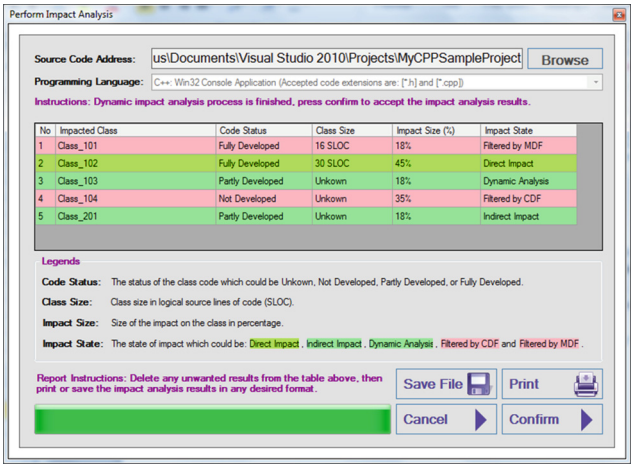

Fig. 7. Sample of dynamic impact analysis

4 Evaluation Strategy

The ultimate aim of the prototype development is to answer a question of "does the developed prototype tool gives an acceptable accuracy of impact analysis results than the selected current impact analysis techniques? To answer this question, we have compared the accuracy of the prototype results with current impact analysis approaches: (1) Class Interactions Prediction with Impact Prediction Filters (CIP-IPF) approach [17, 18]; and (2) the Path Impact approach [11].

We constructed four evaluation attributes: (1) case study; (2) development process; (3) evaluation metrics; and (4) hypothesis.

4.1 Case Study

We have selected three groups of masters student that is currently undertaking master of software engineering course. There are five to seven members of each group that performing various roles in software development activity. We issued several change request to each group and asked them to perform impact analysis at several specific phases. Three impact analysis approaches are used: CIP-IPF approach [17, 18], the Path-Impact approach [11] and the newly developed prototype tool.

4.2 Development Process

Several change requests were issued during the development of a software i.e., requirement phase, design phase, coding phase. Waterfall kinds of development

structure were selected [21]. For future evaluation, Agile model kinds of development structure will be taken into consideration.

4.3 Evaluation Metrics

We used the evaluation metrics as described in [19]. Each prediction results on the impacted classes were grouped according to:

- Not Predicting and Not Changing (NP-NC): number of pairs of classes correctly predicted to not be changing;
- Predicting and Not Changing (P-NC): number of pairs incorrectly predicted to be changing;
- Not Predicting and Changing (NP-C): number of classes incorrectly predicted to not be changing; and
- Predicting and Changing (P-C): number of classes correctly predicted to be changing.

Based on the groups, the following values were then calculated [20, 22]: (1) Completeness value: The ratio of the actual class interactions or impacted classes that were predicted; (2) Correctness value: The ratio of the predicted class interactions that were actually interacting or impacted classes that were actually impacted; and (3) Kappa value [19]: This value reflects the accuracy or the prediction (0 is no better than random chance, 0.4–0.6 is moderate agreement, 0.6–0.8 is substantial agreement, and 0.8–1 is almost perfect agreement [16, 20]).

4.4 Hypotheses

- H_0: CIAT does not give higher accuracy of impact analysis results than the selected current techniques results
- H_a: CIAT gives higher accuracy of impact analysis results than the selected current evaluation Results

To validate the hypothesis, an Independent T-Test statistical analysis was used. At the first stage, we compared Means results between the CIP-IPF approach and the CIAT whereas the second stage is between Means results of Path Impact technique and CIAT.

5 Evaluation Results

Table 1 below shows the impact analysis results produced by all selected impact analysis techniques (CIP-IPF, Path Impact and CIAT).

Table 1. Impact analysis results produced by all techniques

CR ID	CIP-IPF			Path Impact			CIAT		
	Com (%)	Corr (%)	Kappa value	Com (%)	Corr (%)	Kappa value	Com (%)	Corr (%)	Kappa value
CR1	80	100	0.785	66.7	100	0.652	86.7	100	0.876
CR2	81.3	100	0.821	78.6	100	0.789	92.9	100	0.935
CR3	76.9	100	0.768	80	92.3	0.752	100	93.8	0.944
CR4	83	94	0.795	88.7	94.1	0.85	94.4	94.4	0.903
CR5	83	91	0.767	91.7	91.7	0.852	91.7	91.7	0.852
CR6	82.4	100	0.832	76.5	92.9	0.721	94.1	94.1	0.842
CR7	81.8	90	0.734	80	94.1	0.764	95	95	0.912
CR8	80	100	0.806	78.6	100	0.787	92.9	100	0.935
CR9	75	100	0.752	87.5	100	0.884	87.5	100	0.884
CR10	76	100	0.77	88.2	100	0.892	94.1	100	0.947
CR11	85.7	100	0.863	73.7	93.3	0.695	94.7	94.7	0.908
CR12	80	100	0.773	68.8	100	0.676	87.5	100	0.884
CR13	90.9	90.9	0.83	76.5	100	0.769	94.1	100	0.947
CR14	83	100	0.843	77.8	100	0.784	94.4	100	0.95
CR15	80	92	0.749	80	100	0.804	86.7	100	0.874

Key: Com- Completeness; Corr- Correctness

Table 2 below shows the summary of T-Test results.

Table 2. T-Test results

Stage Analysis	Technique	Means Results
Stage 1	CIP-IPF	0.7927
	CIAT	0.9060
Stage 2	Path Impact	0.7773
	CIAT	0.9060

5.1 Stage 1 Analysis: The CIP-IPF Technique Vs. CIAT

Table 2 shows the CIAT means value is 0.9060 and CIP-IPF approach value is 0.7927. This indicates that CIAT value is higher than the CIP-IPF approach. Thus, the values reject the null hypothesis (H_0: CIAT does not improve on the CIP-IPF approach results) and accept the alternate hypothesis (H_a: CIAT approach gives higher accuracy of impact analysis results than the CIP-IPF approach).

5.2 Stage 2 Analysis: The Path Impact Technique Vs. CIAT

The results show CIAT value is 0.9060 and the Path Impact approach value is 0.7773. This shows that the CIAT value is higher than the Path Impact approach. Thus, the

values reject the null hypothesis (H_0: CIAT does not give higher accuracy of impact analysis results than the Path Impact approach) and accept the alternate hypothesis (H_a: CIAT gives higher accuracy of impact analysis results than the Path Impact approach).

6 Conclusion and Future Work

Our contribution on this paper is an automated prototype tool. This tool implements our previously developed change impact analysis approach. The uniqueness of the approach or the prototype tool is the introduction of Class Dependency Filtration (CDF) and Method Dependency Filtration (MDF) in impact analysis implementation, The MDF is used to handle the partially developed class analysis issues. For the future works, we plan to extend the tool implementation from agile methodology perspective instead of waterfall methodology.

Acknowledgements. The research is financially supported by Ministry of Education Malaysia and Universiti Teknologi Malaysia under Prototype Research Grant Scheme (PRGS), Vot No: 4L617.

References

1. Pfleeger, S.L., Bohner, S.A.: A framework for software maintenance metrics. In: Proceedings of the International Conference on Software Maintenance, pp. 320–327 (1990)
2. Bennet K.H., Rajlich, V.T.: Software maintenance and evolution: a roadmap. In: Proceedings of the International Conference on the Future of Sofware Engineering, pp. 75–87 (2000)
3. Kotonya, G., Somerville, I.: Requirements Engineering: Processes and Techniques. Wiley, Chichester (1998)
4. Arnold, R.S., Bohner, S.A.: Impact analysis-towards a framework for comparison. In: CSM-93, Proceedings Conference on Software Maintenance, pp. 292–301, 27–30 September 1993 (1993)
5. Antoniol, G., Canfora, G., Casazza, G.: Information retrieval models for recovering traceability links between source code and documentation. In: Proceedings of the International Conference on Software Maintenance, pp. 40–44 (2000)
6. Kama, N.: A change impact analysis approach for the software development phase: evaluating an integration approach. Int. J. Soft. Eng. Appl. **7**(2), 293304 (2013)
7. Kama, N.: Integrated change impact analysis approach for the software development phase. Int. J. Soft. Eng. Appl. **7**(2), 293–304 (2013)
8. Basri, S., Kama, N., Ibrahim, R.: A novel estimation approach for requirement changes during software development. Int. J. Softw. Eng. Appl. **9**(1), 237–252 (2015)
9. Kama, N., Basri, S.: Considering partially developed artifacts in change impact analysis implementation. J. Softw. **9**(8), 2174–2179 (2014)
10. Breech, B., Tegtmeyer, M., Pollock, L.: Integrating influence mechanisms into impact analysis for increased precision. In: Proceedings of the 22nd International Conference on Software Maintenance, pp. 55–65 (2006)

11. Law, J., Rothermal, G.: Whole program path-based dynamic impact analysis. In: Proceedings of the 25th International Conference on Software Engineering (ICSE 2003), pp. 308–318 (2003)
12. Breech, B., Danalis, A., Shindo, S., Pollock, L.: Online impact analysis via dynamic compilation technology. In: Proceeding of the 20th IEEE International Conference on Software Maintenance, Washington, US, 11–17 September 2004
13. Law, J., Rothermel. G.: Incremental dynamic impact analysis for evolving software systems. In: Proceeding of the 14th International Symposium on Software Reliability Engineering, Washington, US, 17–20 November 2003
14. Tartler, R., Lohmann, D., Scheler, F., Spinczyk, O.: AspectC++: an integrated approach for static and dynamic adaptation of system software. Knowl.-Based Syst. **23**(7), 704–720 (2010)
15. Abaei, G., Selamat, A., Fujita, H.: An empirical study based on semi-supervised hybrid self-organizing map for software fault prediction. Knowl.-Based Syst. **74**, 28–39 (2015)
16. Hassine, J., Rilling, J., Hewitt, J., Dssouli, R.: Change impact analysis for requirement evolution using use case maps. In: Proceeding of the 8th International Workshop on Principles of Software Evolution, Washington, US, 5 September 2005
17. Kama, N., French, T., Reynolds, M.: Design patterns consideration in class interactions prediction development. Int. J. Adv. Sci. Technol. **28**, 6 (2011)
18. Kama, N., Azli, F.: Requirement level impact analysis with impact prediction filter. In: Proceeding of the 4th International Conference on Software Technology and Engineering, Phuket Thailand, 1–2 September 2012
19. Lindvall, M., Sandahl, K.: How well do experienced software developers predict software changes. J. Syst. Softw. **43**, 1 (1998)
20. Cohen, J.: A coefficient of agreement for nominal scales. J. Educ. Psychol. Measur. **20**, 1 (1960)
21. Sommerville, I.: Software Engineering, 7th edn. Pearson Education, New Jersey (2008)
22. Landis, J.R., Koch, G.G.: The measurement of observer agreement for categorical data. J. Biometrics **33**, 1 (1977)