Preprint

QuASoQ 2021

9th International Workshop on
Quantitative Approaches to Software Quality

co-located with APSEC 2021
Taipei, Taiwan (virtual), December 6th , 2021

Editors:

Horst Lichter, RWTH Aachen University, Germany
Selin Aydin, RWTH Aachen University, Germany
Thanwadee Sunetnanta, Mahidol University, Thailand
Toni Anwar, University Petronas, Malaysia

Table of Content

The Impacts of Requirements Relationships Knowledge on Requirements Quality and Software Development Project Success

Ruhaya Ab. Aziz[1,2], Bernard Wong[3]

[1]Faculty of Computer Science and IT, UTHM), Locked Bag 101, 86400 Parit Raja, Johor, Malaysia
[2]School of Software,University Of Technology, Sydney, 15 Broadway, Ultimo NSW 2007, Australia
[3]Po Box 020 Wahroonga NSW 2076 Australia

Abstract

Requirements quality is one of the factors that may determine the success or failure of a software project. Thus, maintaining requirements quality is important but also a challenge as an individual requirement does not stand alone and they are related to one another in several ways. The problem may become more challenging as the requirements and their interrelationships are not static and will continually change. However, current research largely focusing on the assessment of the impact of requirements quality on success. There is lack of research assessing the impact of the interrelationships between requirements on success. Therefore, this research aims to investigate how the interrelationships between requirements impact requirements quality as well as the success of software development project. An empirical study to examine further the impacts was conducted from the perspective of business analyst. Using Structural Equation Modelling (SEM) and especially Partial Least Square (PLS), we found that there are significant impacts of requirements relationships towards requirements quality as well as success. The outcome from this research can be used as a guide to working with requirements relationships, knowledge useful for business analysts and research community.

Keywords

Requirements Relationships Knowledge, Requirements Quality, Software Development Project Success, Partial Least Square (PLS), CEUR-WS

1. Introduction

Requirements quality is defined as a set of requirements or software requirements specification (SRS) that having all the good characteristics that listed as proposed by IEEE-830-1998 recommended practices for SRS [1]. According to the practice; attribute of requirements quality are including correct, unambiguous, complete, consistent, ranked for importance and/ or stability, verifiable, modifiable and traceable. Thus, an SRS developed for a particular software project should fulfill all the characteristics listed to ensure the requirements quality. Consequently, to produce quality requirements specification; the comprehensive understanding of requirements is needed.

Moreover, to fulfil the necessity of comprehensively understanding requirements, it is important to acknowledge how each requirement is related to one another. Knowledge on how each requirement is related to one another may assist stakeholders to make informed decision in accomplishing many things that involve in managing requirements [2]. In this paper, the information of the relationships between requirements is defined as requirements relationships knowledge (RRK). RRK is concerns on how requirements are related to one another and other artefacts during the software development project. Accordingly, requirements relationships knowledge may provide guide in organising and structuring the requirements documentation and specification. Karlson et al. [3] indicated that one of the main contributions of requirements relationships knowledge is in the bundling structure of requirements. A good structure and organised requirements specification can facilitate better management of requirements, whether it is done manually or by any automatic tool. It will also provide a good basis for any manipulation and maintenance activities for the later phases. This will increase the possibility of achieving project success. Diev [4] ascertained that requirements structuring is an essential activities in requirements engineering as requirements structure and representation will directly impact requirements development process and the requirements quality. The importance of requirements quality is also advocated by agile practitioners [5, 6].

In relation to this, previous researchers asserted that requirements quality especially Software Requirements Specification quality has strong impact on the success or failure of a software development project [7, 8]. They have made thorough investigation into how requirements quality impacts project success. There are also some re-

QuASoQ '21: 9th International Workshop on Quantitative Approaches to Software Quality, Dec 06, 2021, Taipei, Taiwan
ruhaya@uthm.edu.my (R. Ab. Aziz);
bernad.wong@enterprise-strategy.org (B. Wong)
0000-0002-0877-7063 (R. Ab. Aziz)

searchers that suggest the contribution of RRK [3, 4, 9]. However, there are lacks of study that examine how RRK impacts requirements quality as well as software development project success. Therefore, this research will extend the previous works to examine further the three main issues: 1) the impacts of RRK on requirements quality; 2) the impacts of requirements quality on project success; 3) the impacts of RRK on project success. This research has been conducted empirically using survey method and the analysis of the data has been performed using Structural Equation Modelling (SEM) and specifically Partial Least Square (PLS).

The rest of this paper will be organized into 4 sections. First, the research context and research model is discussed in Section 2. Second, the research method, which mainly concerns on the development and the validation of the requirements relationships instrumentation design, will be discussed in Section 3. Next, section 4 of this paper will present the discussion of the result. Finally, section 5 will present the concluding remark, including the future work in both research and practice.

2. Research Context

The success of a software development project (SDP) is a concern for any related stakeholders. Success in SDP is described based on several criteria including: 1) quality of product [10, 11, 12], 2) Timeline of the delivery (schedule) [11, 12, 13, 14], 3) Cost [11,12] , 4) Satisfaction of stakeholder [11], 5) met requirements [11], 6) met business objective [11] 7) met scope [12] and 8) learning [14]. Other than that, study in software project management will also involve the factors that may impact the success.

The success of a software development project (SDP) is a concern for any related stakeholders. Success in SDP is described based on several criteria including: 1) quality of product [10, 11, 12], 2) Timeline of the delivery (schedule) [11, 12, 13, 14], 3) Cost [11,12] , 4) Satisfaction of stakeholder [11], 5) met requirements [11], 6) met business objective [11] 7) met scope [12] and 8) learning [14]. Other than that, study in software project management will also involve the factors that may impact the success.The results of both studies have shown the significant impacts of requirements quality on success. The later researchers ranked clear requirements and specification as the top factor among 26 critical success factors of software development project [15]. The ranking supports both previously bodies of knowledge in the assertion of requirements quality as the significant factor for software project success.

Accordingly, quality in requirements specification will always depends on how requirements are determined in the process of requirements determination, which is known as Requirements Engineering (RE). Not enough RE or badly performed RE activities may result in incorrect and incomplete requirements, besides the possibility of high rate of changing in the requirements, which could be the reason for the software project to be challenged. Badly performed RE process has been claimed as positively associated with software failure [16, 17, 18]. Therefore, by improving RE practices, there would be an economic as well as software quality payoff [16, 17]. Clearly, literature indicates the importance of requirements quality and requirements engineering as the critical success factor of a particular software development project. In relation to this, Verner et al. [19] argue that the most important correlation in achieving project success is to have good requirements and to manage those requirements effectively. However, to get the correlations of both factors, we need to know how requirements are related to one another which a concern of RRK.

Although requirements relationships in any software development project are not problematic, they would affect other aspects of software development project and the project as a whole. RRK is asserted as essential when making decisions in the subsequence phases of any software development project including designing [20], requirements prioritisation [21] and testing [22]. In addition, failure to consider RRK during requirements activities is argued could lead to costly mistakes [23]. Hence, RRK needs to be carefully identified, analysed, and managed to avoid any ripple effects.

Moreover, the success of requirements engineering in producing requirements quality as one of the success factors of software project has been discussed in many studies (e.g. [16, 17, 23]) but the studies that particularly discuss how RRK impacts project success are limited. Hence, the questions to be asked are, 1) is this knowledge (RRK) really significant in software development project? 2) If yes, how can RRK impact requirements quality as well as the success of software development project? 3) Other than that, in what way can this knowledge be fully utilised for that purpose? In order to answer these questions, this paper aims to discuss these issues further and extend the literature on the interrelationships between RRK, requirements quality and the related issues that have impacts on project success. The related research model are proposed and illustrated in Figure 1. The model was developed based on software project success factors that are related to RRK as discussed in literature. However, this paper will be focusing only on a part of the model in which consists of the three constructs: 1) RRK, 2) Requirements Quality, 3) success. Thus, the related hypotheses are as follows:

H1: RRK has significant impact on requirements quality.

H2: Requirements quality has significant impact on the success of software development project.

H3: Requirements relationships knowledge (RRK) has

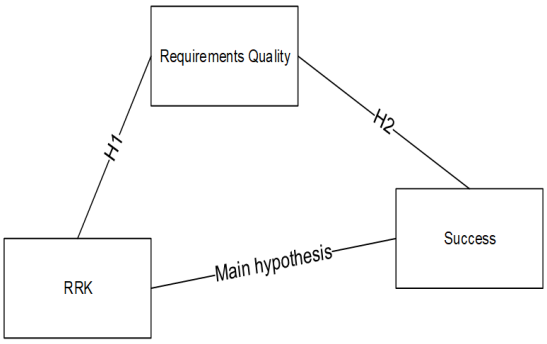

Figure 1: Research Model

The overview of the initial indicators for every construct in the model is shown in figure 2.

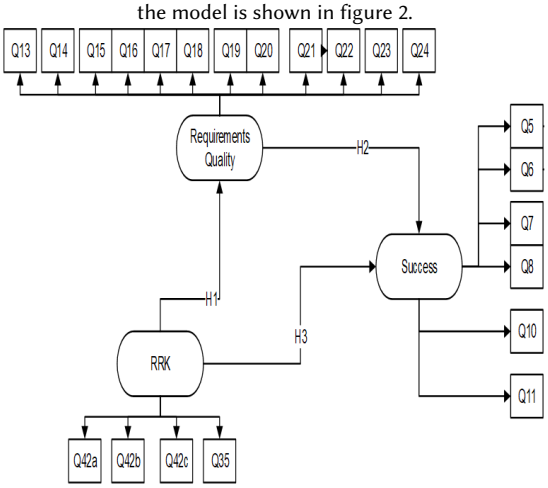

Figure 2: Initial Indicators of the Model

significant impact on requirements quality as well as on the success of a software development project.

Accordingly, to validate the hypotheses, empirical research analysis using Structural Equation Modelling (SEM) was performed. This paper will continue to discuss about the research method used to examine the interrelationships between RRK, requirements quality and success of Software development project (SDP) in the next section.

3. Methodology

3.1. Participant

In this study, the sample was chosen using non-probability sampling specifically purposive sampling. In this regard, any stakeholders involved in managing requirements in their software development project were chosen whereas those not were excluded from the sample. About 173 business analyst and related stakeholders were recruited. The inclusion criteria were including the respondents were stakeholders involved in the management of requirements in their software development project. Approximately, 60 percent of the participants were business analysts and system analysts. Most of the respondents are practitioners in Malaysia Industry and only 10-20 percent of the respondents are from Australia. The findings show that most of the respondents are from medium and large organization (refer to Table 1) and Australian Bureau of statistics classification of business framework (office of small business, 1999). In addition, in Table 2, the findings show that the industry domain of most of the organization is from Information Technology and Telecommunication (41 percent) and, infrastructure and Government (30 percent). Moreover, the respondents largely have about 2-5 years (37 percent), and about 6-9 years (25 percent) experience in requirements writing which represent approximately 62 percent of all the respondents (Table 4).

Table 1
Employment Information of the respondents

Item	Frequency	Percentage
Less Than 10	15	9
11-100	84	49
101-500	32	18
More than 500	41	24

Table 2
Industry domain of the respondents

Item	Frequency	Percentage
Factory automation	3	2
Financial	10	6
Infrastructure and Government	52	30
Medical	4	2
IT and Communication	72	41
Transportation	2	1
Others	32	18

Table 3
Work Designation of the respondents

Item	Frequency	Percentage
Business Analyst	18	11
Business Analyst Manager	6	4
System Analyst	80	47
System Designer	20	12
Tester	4	2
Others	43	25

Table 4
Experience in Requirements Writing

Item	Frequency	Percentage
Within one Year	32	19
2-5 Years	63	37
6-9 Years	43	25
10-15 Years	23	13
More than 15 years	72	41

3.2. Data Collection

There were approximately 380 self-administered questionnaires used for collecting the data from the respondents. Several methods of questionnaire distribution were employed: 1) a number of questionnaires were mailed to the respondents; 2) a number of questionnaires were emailed (on-line survey); 3) a number of questionnaires were completed using drop-off survey method. A total of 210 questionnaires were received but only about 173 questionnaires were usable for analysis. This translates to about 55.3 'percent response rate and only 45.5 percent were considered effective response rate.

Accordingly, missing values or data are asserted as part of almost all research [24]. One of the ways is to omit the subjects that have missing data. If there are missing data at about more than 20 percent from the items in a questionnaire, the subjects related are advised to be deleted from the analysis [25]. Hence, in this study we used the usable questionnaire only in the analysis. The technique used is also known as Listwise. Listwise is the technique where subjects are discarded from the analysis because of there are some questions unanswered in the survey. Even though, this technique will decrease the subjects for the analysis but this technique is used to ensure that the analysis will be done with complete data for all the subjects.

3.3. Structural Equation Modeling (SEM) and Partial Least Square (PLS)

This study is a part of a research that investigated the impacts of requirements relationships on the other factors and tasks in any software development project that possibly will also impact success. The factors and tasks may have direct and indirect relationship; they might impact one another and thus the success or failure of a particular software development project as a whole. Hence, Structural Equation Modelling (SEM) was used to validate and examine the interrelationships and the impacts that they have to one another. SEM is a statistical technique for the validation and estimation of causal relationships using a mix of qualitative causal assumption and statistical data. This method is usually used more for confirmatory rather than exploratory. Thus, SEM is more appropriate for theory testing than theory development. SEM is a generic and powerful multivariate analysis technique that includes specialised versions of several other analysis approaches as special cases. SEM is not intended for a single statistical technique but it is a family of related procedures [26]. Other related terms used are Covariance Structure Analysis, Covariance Structure Modelling, and Analysis of Covariance Structures.

Moreover, SEM can be categorized into two approaches, which are: 1) covariance-based approach, which is related to tools such as EQS and AMOS; and 2) variance-based approach, which is related to PLS. Therefore, PLS was chosen to be used in this research. Partial Least Square (PLS) was chosen because of the following reasons: 1) research on requirements relationships is relatively new; and 2) there is no measurement model that is already available. PLS is asserted as a suitable technique to be used when the phenomenon to be examined is relatively new [27, 28]. Hence, the assessment of the goodness of measure of these constructs in terms of their validity and reliability within the research framework will be discussed in the next section.

3.4. Measure and Goodness of measure

A questionnaire using five-point Likert scale was designed to collect data for each construct of the research model. Some of the instruments in the questionnaire were newly developed whereas most of the questions were designed based on the theory from literature and other empirical studies. Additionally, some parts of the instruments were adapted from previous literature. The final constructs of the model are illustrated in Figure 3:

3.5. Goodness of Measure

In this study, two main criteria have been utilised for evaluating goodness of measures, which are validity and reliability. The combination of both is essential to assure the quality of a research [29]. Validity is about how well a developed instrument measures the particular concept that is intended to be measured [30]. On the other hand, Trochim and Donnelly [29] also indicated that reliability refers to repeatability or consistency. A measure is considered reliable if it gives the same result over and over again. The validity and reliability measures of this research model are discussed in the next section.

3.6. Construct Validity

Construct validity can be described as the degree to which interferences can legitimately be made from the operational constructs in a particular study to the theoretical

Construct	Item	Description
Success	Q5 (SC1)	The outcome of the project meets the business goal.[11]
	Q6 (SC2)	The outcome of the project meets all the specified requirements.[11]
	Q7 (SC3)	The overall quality of the developed application / product is high.[10,11]
	Q10 (SC4)	The project is completed within scope.[12]
	Q11 (SC5)	The requirements-related tasks (e.g. requirements specification, requirements management) have been completed successfully in the project.[11,12,13]
Requirements Relationships Knowledge	Q42a (RRK1)	The relationships between requirements that exist between the components are considered when deciding to implement the solution. [2]
	Q42b (RRK2)	The relationships between requirements that exist between the components are considered when planning the schedule for the design/development team to complete the task. [20, 23]
	Q35 (RRK3)	Before implementing a change to a particular requirement, any possible impact it will cause to other requirements will be considered. [39]
Requirements Quality	Q14 (RQ2)	Requirements are typically grouped according to similar functionality / business area. [1, 3, 4, 7]
	Q24 (RQ3)	The requirements specified in the requirements document are easy to be located whenever needed. [1]
	Q13 (RQ1)	There is a specific structure/arrangement to follow when specifying requirements in the requirements document. [1,3, 4, 7]

Figure 3: Construct in the Modell

constructs on which those operational constructs are based on [31]. Sekaran and Bougie [30] ascertained that construct validity can be used as a confirmation on how well the results obtained from the use of the measure fit the theories around which the test is developed. Thus, convergent and discriminant validity were used to examine how the instrument fits the concept as theorised. Initially, the respective value of loadings and cross loadings in Table 6 were examined to assess whether there were any problems with any particular items. A cut-off value for loadings at 0.5 was considered as significant [25]. If there were any items with a loading of higher than 0.5 on two or more factors, then they were deemed to be having significant cross loadings. Table 5 shows that all the items that measured a particular construct would load highly on the construct and would have lower loadings values on other constructs therefore confirming construct validity.

3.7. Convergent Validity

Accordingly, the test for validity was continued with the convergent validity. This validity test is concerned about the degree to which multiple items are in agreement to measure the same concept. Factor loadings, composite reliability, and average variance extracted (AVE) were

Table 5
Loading and Cross Loading

Item	RRK	RQ	SC
RR1	**0.926**	0.234	0.246
RR2	**0.930**	0.267	0.247
RQ1	0.076	**0.715**	0.329
RQ2	0.303	**0.807**	0.323
RQ3	0.196	**0.722**	0.320
SC1	0.249	0.323	**0.716**
SC2	0.181	0.300	**0.802**
SC3	0.279	0.373	**0.782**
SC4	0.159	0.306	**0.690**
SC5	0.116	0.297	**0.733**

used to measure the convergent validity. This practice was proposed by Hair et al. [245. The convergent validity test findings showed that the factor loadings for all items exceeded the recommended value of 0.5 [25]. Next, composite reliability values illustrated in Table 6 present the degree to which the construct indicators indicated the latent, ranged from 0.793 to 0.925. The value is apparently exceeded the recommended value of 0.7 [25]. Finally, the average variance extracted assessed the variance captured by the indicators relative to measurement error. The value should be greater than 0.5 to justify the use of the construct [30]. As illustrated in Table 6, the AVE was in the range of 0.556 to 0.861.

Table 6
Loading and Cross Loading

Construct	Measurement Item	Loading	CR	AVE
RRK	RR1	0.926	0.925	0.861
	RR2	0.930		
RQ	RQ1	0.715	0.793	0.561
	RQ2	0.807		
	RQ3	0.722		
SC	SC1	0.716	0.862	0.556
	SC2	0.802		
	SC3	0.782		
	SC4	0.690		
	SC5	0.733		

In addition, the results for the measurement model are summarized in Table 7. According to the results, we can conclude that all of the three (3) constructs: Requirements Quality, Requirements Relationships Knowledge and Success were all valid measures of their respective constructs based on their parameter estimates and statistical significance.

3.8. Discriminant Validity

Then, the study was continued to validate the discriminant validity. Discriminant validity is concerned about

Table 7
Loading and Cross Loading

Construct	Measurement Item	Standardized Estimate	t value
RRK	RR1	0.926	38.286
	RR2	0.930	42.992
RQ	RQ1	0.715	7.767
	RQ2	0.807	13.406
	RQ3	0.722	11.478
SC	SC1	0.716	13.616
	SC2	0.802	22.313
	SC3	0.782	19.591
	SC4	0.690	11.708
	SC5	0.733	14.106

Table 8
Discriminant Validity Results

Item	1	2	3
1. RRK	**0.861**		
2. RQuality	0.073	**0.561**	
3. Success	0.070	0.185	**0.556**

the degree to which items differentiate among constructs where they illustrate the measures that theoretically should not be related are in reality not related. This validity test was assessed by exploring the correlations between measures of potentially overlapping constructs. The items should have the highest loading value on their own constructs in the model, and the average variance shared between every construct and its measures should be more than the variance shared between the construct and other constructs [31]. Table 8 illustrates that the squared correlation for each construct is less than the average variance extracted by the indicators measuring the construct to indicate the adequate discriminant validity. As a consequence, the measurement model has demonstrated adequate convergent validity and discriminant validity.

3.9. Reliability Analysis

Reliability is concerned about the quality of measurement. Reliability in a research is the degree to which a measurement procedure produces the same answer each time the measurement procedure is carried out [33]. One of the general classes of reliability is the internal consistency reliability that is utilised to measure the consistency of result across items within a test [29]. In relation to this, Cronbach's alpha coefficient was used to examine the reliability of the inter item consistency of the measurement items. The summarisation of loadings and alpha values are illustrated in Table 9. Based on the findings in Table 9, all the alpha values are above 0.6, which are conforming to what have been suggested by Nunnaly and Berstein [34]. Consequently, the composite reliability values also ranged from 0.793-0.925 (refer table 6). Composite reliability values are another method similar to Cronbach's alpha for internal consistency reliability estimate where a composite reliability value of 0.7 or more is considered acceptable [35]. Therefore, it can be concluded that the measurements used in this study were reliable.

Table 9
Loading and Cross Loading

Construct	Measurement Item	Cronbach Alpha	Loading Range	Num of Item
RRK	RR1- RR2	0.839	0.926-0.930	2(3)
RQ	RQ1-RQ3	0.611	0.802-0.910	3(4)
SC	SC1-SC5	0.799	0.690-0.802	5(7)

Another issue in the area of survey research is common method variance. Considering the self-reported nature of the data used, there was a possibility for this issue to happen. Hence, Harman one factor test was performed to determine the extent of this issue. Accordingly, Podsakoff and Organ [36] indicated that common method bias is problematic if a single latent factor would account for the majority of the explained variance. The unrotated factor analysis illustrated that the first factor accounted for only 22.5 percent of the total variance, consequently ascertained that the common method bias was not a serious issue in this study.

Finally, the analysis is continued with the path analysis to test all the hypotheses generated in this study. Table 10 presents the results. The result of the analysis shows that the three hypotheses: H1, H2 and H3 were supported. The results implied that there are significant interrelationships between requirements relationships knowledge, requirements quality and success of software development project. In the analysis, the path coefficient value for RRK->RQ is 0.270 whereas the path coefficient for RQ-> Success is 0.306. Both coefficient values are in the ranges of (0.20-0.30) that have been asserted as acceptable [27]. Hence, it can be concluded that there are significant relationships exist between the three constructs.

Table 10
Path Coefficient and Hypothesis Testing

Hypothesis	Relationship	Coeficient	t value	Support
H1	RRK->RQ	0.270	2.869	Yes
H2	RQ-> Success	0.306	3.398	Yes
H3	RRK->Success		3.465	Yes

Moreover, mediation effect analysis has also been

conducted. The finding reports that, there is exists mediator relationships between the three constructs. Figure 4 illustrates the analysis which represents the initial coefficient for the three constructs. There are several criteria that need to be fulfilled before any mediation effect analysis can be performed. First, the predictor (RRK) has significant impact on the mediator requirements quality (RQ) (later noted as a); second, the mediator (RQ) has significant impact on the criterion variable Success (b); and third, the predictor (RRK) has significant impact on the criterion variable in the absence of the mediators' impact (c). Therefore, to establish the mediating effect, the indirect effect of a x b (see figure 4) has to be significant. In this regard, the z statistic is applied [3], specifically the value is significant at p <0.05. If the z value exceeds 1.96 (p <0.05), then the hypothesis H3 can be accepted where there is an indirect impact of RRK through requirements quality on the success of software development project. The z value is defined as the following:

$$z = \frac{a \times b}{\sqrt{b^2 \times S_a^2 + a^2 \times S_b^2 + S_a^2 \times S_b^2}} \quad (1)$$

As illustrated in figure 4, there is a significant impact of RRK on requirements quality (0.271, p<0.05) as well as requirements quality on success (0. 387, p<0.05). Consequently, there is also a significant direct impact of RRK on the success of software development project (0.169, P<0.05); thus, requirements quality is established as a partial mediator. This mediating effect of requirements quality in this study is confirmed by z statistic [38]:

$$z = \frac{0.27 \times 0.39}{\sqrt{0.39^2 \times 0.09^2 + 0.27^2 \times 0.09^2 + 0.09^2 \times 0.09^2}}$$

z=2.502 (2)

The result demonstrates that, requirements quality has mediating effects where it implies that there is indirect impact on success; variance accounted (VAF) value then is used to represents the ratio of the indirect effect to the total effect. The VAF value indicates that 38.2 percent of the total effect of RRK on success of software development project is explained by indirect effect (requirements quality):

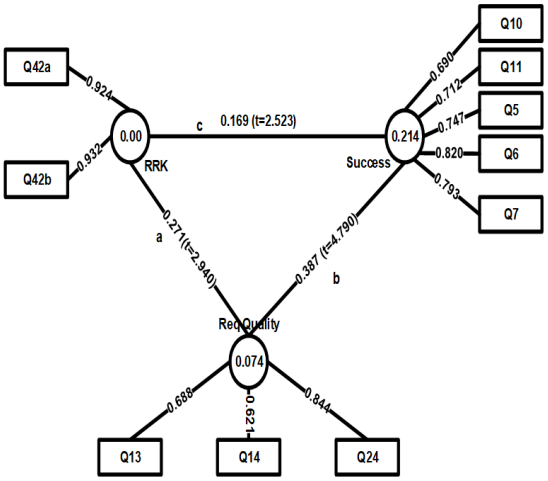

Figure 4: Mediating effect of RRK, RQ and Success

VAF=(a $\times b$) $\div$ $(a \times b + c)$
VAF=(0.271 $\times$0.387) $\div$ $(0.271 \times 0.387 + 0.169)$
$VAF = 0.382$ (3)

Therefore, it can be concluded that the relationships between the three constructs is significant and the three hypotheses are also confirmed by the mediation effects that exist among them.

4. Result and Discussion

This study focusing on the impacts of the RRK as an independent variables on requirements quality as well as the success of software development project using the PLS technique. It is a part of a research in which also investigates the impacts of RRK on other related factors of success including requirements change [39]. In the first part of this study, the goodness of measure is examined by looking at the validity and reliability of the measures using the PLS approach. The findings illustrated that the measures used demonstrate both convergent validity and discriminant validity. In addition, the reliability of the measures was also examined by observing the Cronbach Alpha values and Composite Reliability values. The findings show that both the Chronbach Alpha values and composite reliability values have fulfilled the criteria set up by other established researchers. Thus, the results have confirmed that the measures in the model

were reliable. Accordingly, the findings of this paper confirmed and supported the direct significant impacts of RRK on requirements quality (H1); the direct significant impact of requirements quality on the success of software development project (H2); and consequently supported hypothesis (H3) that proposed the indirect significant impacts of RRK on the success of software development project.

Firstly, RRK has significant impact on requirements quality in which inline with what have been ascertained in the literature (e.g. [23]) in which supporting H1. The requirements relationships knowledge provides guide on how a set of requirements can be structured and organized in requirements specification document. The requirements documentation that is properly organized and well structured can contribute to the good quality of requirements [4, 23]. According to the analysis of the result, the main characteristics of requirements quality that related to RRK are: 1) Requirements are typically grouped according to similar functionality/business area; 2) The requirements specified in the requirements document are able to be located whenever needed; and 3) There is a specific structure / arrangement to follow when specifying requirements in the requirements document. Characteristics of items 1 and 3 confirmed the important of RRK in structuring requirements in an SRS. Both items then may support the characteristics of items 2. When the requirements can be located whenever needed, an SRS can be indicated as having one of the good characteristics listed in the IEEE-830 recommended practices which is traceable as well as may help in fulfilling other requirements quality characteristics such as modifiable and verifiable. Thus, it is clear that the findings were supporting H1.

Secondly, the findings from the analysis ascertained that a software development project is considered successful when the project has fulfilled several criteria. The criteria are listed as the following: 1) the outcome of the project meets all the specified requirements; 2) the overall quality of the product is high; 3) The requirements-related tasks (e.g. requirements specification, requirements management) are successfully completed in the project; 4) The outcome of the project meets the business goal; and 5) The project is completed within scope. Indeed, all of the criteria listed are in fact quite similar to the criteria proposed by previous researchers (e.g. [10, 12]). The finding asserted that as long as the outcome meets all the specified requirements and business goal, has good quality, completed within scope, and all the requirements-related activities are completed successfully, the project will be considered successful although the project is not completed within time and budget. Thus, the findings are apparently shown how the success of requirements activities in which including maintaining requirements quality will impact success in which supporting H2.

Finally, the phenomena of the interrelationships between RRK->Requirements quality ->Success (H3) has been proved by the findings. The interrelationships between them are also found to be the strongest link that existed in this study.

5. Conclusion

Therefore, the findings have confirmed the three hypotheses listed in this study. As requirements relationships knowledge has significant impact on requirements quality (H1); and requirements quality has direct significant impacts on success (H2); it can be concluded that requirements relationships knowledge is another significant factor that will impact requirements quality as well as project success (H3). Accordingly, the findings also confirmed the significant impacts of RRK on the software project success. In the future, this quantitatively finding of this study will be continued with a qualitative study in investigating further how RRK impacts requirements quality and other related factors on the software project success from the business analyst perspectives.

Acknowledgments

The authors fully acknowledged Ministry of Higher Education (MOHE) and Universiti Tun Hussein Onn Malaysia for the approved funds which makes this important research viable and effective.

References

[1] IEEE Recommended Standard Practices for software Requirement Specification,*8*30-std-1989

[2] I. Ozkaya, 2006, 'Representing Requirements Relationship', *F*irst International Workshop on Requirements Engineering Visualisation, IEEE Computer Society, pp. 3-3.

[3] L.Karlsson, A. Dahlstedt. B.Regnell. J.N.O. Dag A. Perrson 2007, 'Requirements Engineering challenges in market driven software development-An interview study with practitioners', Elsevier Science Direct - Information and Software Technology, vol. 49, no. 2007, pp. 588-604.

[4] S. Diev, 2007, 'Structuring complex requirements', ACM SIGSOFT Software Engineering Notes Pages 1, vol. 32, no. 2, pp. 1-5.

[5] G. Lucassen, F. Dalpiaz, 2016,'Improving agile requirements:The quality user stories and tools', in J.M.E.M. van der Werf, et al. Requirements Eng (2016) 21: 383. doi:10.1007/s00766-016-0250-x

[6] P. Heck A. Zaidman, 2016,' A Systematic Literature Review on Quality Criteria for Agile Require-

ments Specification' in Software Quality Journal, vol. 24(93), Springer, United State.

[7] M.I. Kamata T. Tamai, 2007,'How Does Requirements Quality Relate to Project Success or Failure', 15th IEEE International Requirements Engineering Conference, RE 2007, October 15-19th, New Delhi, India, pp. 69-78

[8] E. Knauss C.E. Boustani 2008, 'Assessing the Quality of Software Requirements Specifications', 16th IEEE International Requirements Engineering Conference, RE 2008, 8-12 September, Barcelona, Catalunya, Spain. IEEE Computer Society, pp. 341-342

[9] R. Berntsson Svensson, T. Olsson B. Regnell (2013). An investigation of how quality requirements specified in industrial practice. Information and Software Technology, 55(7), 1224-1236. DOI: 10.1016/j.infsof.2013.01.006

[10] C. Wohlin, A.V. Mayrhauser, M. Höst B. Regnell 'Subjective evaluation as a tool for learning from software project success', Information Software Technology Vol. 42, No. 14, 2000, pp. 983-992

[11] G.Thomas, W. Fernandez, 2008 'Success in IT Projects: A matter of definition', International Journal of Project Management 26(7), October, 2008

[12] N. Agarwal U. Rathod 2006, 'Defining success for software projects: An exploratory revelation', International Journal of project management vol. 24

[13] M. Ibraigheeth S. A. Fadzli, 2019, 'Core factor for Software Project Success', International Journal on Information visualisation,Vol. 3, No. 1

[14] V. Boola 2015, Impact of project success factors in managing software projects in India : an empirical analysis, Sage Journal, Vol (3), issue 2, pp:109-125.

[15] M.H. D. Nasir S. Sahibuddin, 2011,'Critical success factors for software project: A comparative study',Scientific Research and Essays Vol. 6(10), pp. 2174-2186, 18 May, 2011

[16] K.E. Emam N.H. Madhavji, 1995, 'Measuring the success of requirements engineering processes', IEEE, pp. 204-211.

[17] L. Radlinski, 2012, 'Empirical Analysis of the impact of Requirements Engineering on Software Quality' in Lecture Note on Computer Sciences, Bjorn, R. Daniela, D.(Eds), pp. 232-238, Springer-Verlag Berlin, Heidelberg.

[18] A. Hussain, E.O. C., Mkpojiogu F.M. Kamal, 2016, 'The role of Requirements in the success or failure of software project', International review of management and marketing, Vol.6, special issue (7).

[19] J. Verner, K.Cox, S. Bleistein N. Cerpa 2005, 'Requirements Engineering and Software Project Success: An Industrial Survey in Australia and the U.S.' Australasian Journal of information systems, vol. 13, no. 1, pp. 225-238.

[20] I. Ozkaya O. Akin 2005, 'Requirement-Driven design: assistance for information traceability in design computing', Elsevier Science Direct - Design studies, vol. 27, no. 2006, pp. 381-398.

[21] A.M. Davis, 2005, Just Enough Requirements Management: Where software development meets Marketing, Dorset House Publishing, New York

[22] S. Jungmayr, 2002, 'Identfying Test-Critical Dependencies', Proceedings of the International Conference on Software Maintenance (ICSM'02), IEEE Computer Society.

[23] A.G. Dahlstedt A. Persson 2005, 'Requirements Interdependencies: State of The Art and Future Challenges', in A. Aurum C. Wohlin (eds), Engineering and Managing Software Requirements, Springer-Verlag Berlin Heidelberg, Germany, pp. 95-116.

[24] D.C. Howel, 2008, The Analysis of Missing Data in Outhwaite, W. Turner, S. Handbook of Social Science Methodology, London, Sage.

[25] J.F. Hair, W.C.Black, B.J.Babin R.E. Anderson 2010 Multivariate Data Analysis. Upper Saddle River, NJ: Prentice-Hall.

[26] R.B. Kline 1998, Principles and Practice of Structural Equation Modeling, The Guilford Press, New York.

[27] W.W. Chin 1998. Issues and opinion on structural equation modeling. MIS Quarterly, 22(1), 7-16.

[28] J.F. Hair, M. Sarstedt, C.M. Ringle J.A. Mena 2012: An assessment of the use of partial least squares structural equation modeling in marketing research, in: Journal of the Academy of Marketing Science (JAMS), forthcoming (online available). http://www.springerlink.com/content/t502155t60nv8005/

[29] , W.M.K. Trochim J.P. Donnelly 2008, The Research Methods knowledge Base, Third edn, Atomic Dog a part of CENGAGE Learning, USA.

[30] U. Sekaran and R. Bougie 2010, Research Methods for Business: A Skill Building Approach. UK: John Wiley and Sons.

[31] D.W. Barclay, R. Thompson C.A. Higgins 1995 The Partial Least Squares (PLS) Approach to Causal Modeling: Personal Computer Adoption and Use an Illustration. Tech Studies 2(2): 285-309.

[32] D.R. Compeau, C.A. Higgins, S. Huff 1999. Social Cognitive Theory and individual Reactions to Computing Technology - A Longitudinal-Study. MIS Q 23(2): 145-158.

[33] J. Kirk M.L. Miller 1986, Reliability and Validity in qualitative research, vol. 1, Sage Publications, USA.

[34] J. Nunnally L. Berstein 1994. Psychometric Theory.

New-York, NY: McGraw-Hill
[35] C. Fornell D.F. Larcker, 1981. Evaluating structural equation models with unobservable variables and measurement error. J Mark Res 18(1): 39-50.
[36] P.M. Podsakoff, D.W. Organ 1986, 'Self-reports in organizational research: Problems and prospects'. Journal of Management, 12(4),531–544.
[37] M. Wetzels, G.A. Schroder V.C. Oppen 2009. Using PLS path modeling for assessing hierarchical construct models: Guidelines and empirical illustration. MIS Quarterly, 33(1), 177–195.
[38] K, J.Preacher G.J. Leonardelli 2013 at http://www.quantopsy.org/sobel/sobel.htm access on 13 February, 2013
[39] A.A. Ruhaya W. Bernard 2015, 'The Interplay between Requirements Change and Requirements Relationships knowledge towards software project success: An Assessment Using Partial Least Square (PLS)', Procedia of Computer Science Vol.46, No.2015, pp. 732-741

Predicting Requirements Volatility: An Industry Case Study

Anıl Holat[1,2], Ayse Tosun[2]

[1]Aselsan Inc., Ankara, Turkey
[2]Faculty of Computer and Informatics Engineerings, Istanbul Technical University, Turkey

Abstract

Software requirements are exposed to many changes during their software development life-cycle. These changes namely additions, modifications or deletions are defined as requirements volatility. Prior requirement volatility prediction studies utilize different requirement volatility measures. In this study we predict number of changes per software requirement as requirement volatility for a large scale safety-critical avionics project in ASELSAN. We employ a comprehensive metric set to explain requirements volatility: requirement quality measures, project specific factors and requirement interdependencies. Predictive models are created through combining input metric sets with machine learners. Success of models in predicting requirement changes, the best performing input metric combinations, the best performing machine learners and success of models in predicting highly-volatile requirements are evaluated in this study. The best prediction results are obtained with the model employing quality metrics, project specific metrics, network metrics altogether with k-nearest neighbour machine learner (MMRE=0.366). Also the best model correctly identifies 63.2% of highly volatile requirements which are exposed to 80% of the total requirement changes. Our study results are encouraging in terms of creating requirement change prediction tools to prevent requirement volatility risks prior to the requirement review process.

Keywords

Predicting Requirements Volatility, Quality Metrics, Network Metrics, Requirements Quality, Requirements Change

1. Introduction

Although software engineering has experienced significant advancements in the last decades, majority of the large-scale software projects still try to cope with requirement changes during their software development life cycle due to dynamic nature of software development activities [1]. Changes for requirements namely additions, deletions or modifications are defined as requirements volatility [2]. Continual requirement changes during software development have tremendous impact on the cost, the schedule and the quality of the final product. Unfortunately, significant number of software projects cannot be completed successfully or completed partially because of requirements' high volatility [2].

According to a survey conducted by Thakurta [3], project managers use various requirement volatility measures: number of changes to the identified use cases, number of changing requirements identified within the issued change requests, realized requirements out of total requirements, and amount of budget the project had to spent on the changing requirements. Alsalemi et al. [4] also report a literature review on requirements volatility prediction. Accordingly, ten studies have employed machine learning methods to predict requirements volatility until 2017. These studies utilize different requirement volatility measures such as number of requirement changes [5], requirement stability index of the project [6], requirements that will be changed in next iteration [7], requirement change impact [8], the impact of requirements changes on project distribution and cost factor [9], software schedule [10]. Related studies also propose requirements complexity metrics [6], requirement dependency metrics [8], requirement size metrics [5] and requirements evolution metrics [7] to predict their own definition of requirement volatility measure.

In our study we aim to predict number of changes per software requirement by using requirement quality measures, project specific factors and requirement interdependencies. We define requirement volatility as the number of change requests reported for a software requirement. This change request could be either for adding a new requirement or modifying an existing requirement. We chose a safety-critical avionics software project in ASELSAN with more than 20,000 requirements for our study. Loconsole et al. [5] conducted a similar study to predict number of requirement changes using size measures on projects with less than 50 requirements. Our study complements the prior work by mining a larger dataset with thousands of requirements and a more comprehensive metric set considering quality and interdependency aspects of requirements as well as project specific factors. It should be noted that the change requests that we study in this work occured in any phase of software development after Software Requirements Specification (SRS) document has been reviewed and confirmed. Thus we investigate post-SRS requirements volatility for the avionics project under study.

The rest of paper is organized as follows. Section 2

QuASoQ'21: 9th International Workshop on Quantitative Approaches to Software Quality, December 06, 2021, Taipei, Taiwan
aholat@aselsan.com.tr (A. Holat); tosunay@itu.edu.tr (A. Tosun)
0000-0002-8727-6768 (A. Holat); 0000-0003-1859-7872 (A. Tosun)

presents related empirical studies carried on for requirement volatility prediction. Section 3 explains study design model in detail. Results and Threats to Validity of our work discussed in Section 4. Section 5 presents conclusion and points out possible directions for future work.

2. Related Work

In this section we present previous studies that aim to predict volatility for requirements, and we focus on the input metrics they employed. We report details of five relevant studies [11, 6, 7, 8, 5] from the literature review conducted by Alsalemi et al. [4]. We also discuss the approaches of other recently published, related studies [12, 13] in this section.

Nakatani et al. [11] propose a method to predict requirement volatility using social relations between executives, competitors, cooperative organizations, and the natural environment. Those measures can be applied to customer requirements easily but it would take some effort to associate them with software requirements. Christopher et al. [6] present requirements complexity metrics to define volatility. Functional requirement complexity, non-functional requirement complexity, input-output complexity, interface and file complexity measures are used to calculate whole project's stability, whereas we seek to predict requirement volatility for each software requirement. Shi et al. [7] present a model to predict future requirement changes by using previous requirement change metrics. They generated six history metrics for requirements that contain information about volatility of topic, frequency of changes and time duration between changes. History metrics can be used to predict requirements that will be changed in next iteration, but has little use in predicting requirements volatility for new projects. Pedrycz et al. [13] also employ the following change logs as input metrics: created version of requirement, last developer, number of modifications, requirement lifetime duration. Change logs are created on later phases of software development thus they are again not very useful to predict requirements volatility for projects in earlier development phases. Goknil et al. [8] and Hein et al. [12] use requirements interrelations for volatility prediction. Goknil et al. [8] utilize formal semantics of requirement relations as input features, whereas Hein et al. [12] create network metrics by using syntactical natural language data. We have combined both measures and created network metrics by using links between system and software requirements instead of lingual relations between requirement texts. Regarding network metrics we employ degree centrality, eigenvector centrality, closeness centrality and betweenness centrality metrics, whereas Hein et al. [12] used 40 network metrics.

According to the literature review, only one study conducted by Loconsole et al. [5] present an empirical study to predict number of changes per requirement, so this study is the most relevant to our work. Following size measures are used to predict number of requirement changes: number of actors interacting with use cases, number of words in each file, number of revisions of files, number of lines per file. In our study, size measures are also used to represent requirement quality but we also enriched our metric set with project specific metrics and network metrics. It should be noted that we do not take deletion requests and deleted requirements into consideration while defining requirements volatility, because in our industrial context we rarely encounter such requests for the safety-critical software. Finally, we applied our model on more than 20,000 requirements that help us assess the generalizability of our findings on predicting volatility on every software requirement using different metrics sets.

3. Study Design

In this section we explain our empirical study design in detail. In Section 3.1 research questions are explained. The analyzed project for which a model would be proposed is described in Section 3.2. In Section 3.3 selected input metrics for requirement volatility prediction are described. Section 3.4 describes the output measure of the prediction model. The used tools are explained in Section 3.5. Machine learning techniques employed in this study are presented in Section 3.6. Finally in 3.7, performance evaluation measures are defined for our model.

3.1. Research Questions

Our main goal is to predict requirements volatility at earlier stages of development lifecycle, and accordingly two research questions are defined.

Research Question(RQ) 1: To what extent do requirement quality metrics, project specific metrics and network metrics predict the volatility of a software requirement?

Previous studies used different metric sets to predict requirements volatility. In this study we aim to use a comprehensive set of input metrics, and observe their individual effects on requirement volatility prediction. While predicting the volatility, we use the number of change (addition and modification) requests on a software requirement. Being inspired by the metric sets used in the literature, we form a group of requirement quality metrics and network metrics. Additionally, project specific metrics for this particular safety-critical avionics project are defined and utilized throughout this study. During model assessment, the performance of each in-

Table 1
Release based AVPRJ statistics

Release	Number of REQs	Mean CR per REQ	Median CR Per REQ
Release 1	8,640	0.7457	1
Release 2	11,401	1.1165	1
Release 3	2,730	0.5267	0
Total	22,771	0.9051	1

put metric set and the combination of those are reported. Detailed sub-questions related to RQ 1 are also listed below:

RQ 1.1: Which metric group is a better indicator of the number of requirement changes?

RQ 1.2: Which machine learning algorithm is better at predicting the number of requirement changes?

RQ 2: How successful are the proposed models in predicting highly volatile software requirements?

Software requirements have a history of varying number of changes during software development life cycle. Some requirements do not change at all; however, some requirements expose to multiple changes and pose risks to a software project. Practically, our model should predict highly volatile requirements, so those requirements will be reviewed by experienced reviewers in detail. For this research question (RQ 2) we measure the success of our models on highly volatile requirements based on a technique in [14].

3.2. Analyzed Project

We chose a safety-critical avionics software project to perform our analysis. We will refer to this project with AVPRJ in the rest of this paper. AVPRJ has many releases from which three releases are selected. Software requirements for those releases are related since they all belong to the same project; however they are partially distinct since each release consists of implementation of different software components developed by many software developers. AVPRJ has a total of 22,771 software requirements. Some release based descriptive statistic for AVPRJ are given in Table 1. CR is used as an abbreviation for change request, REQ is used as an abbreviation for a single requirement. Most of the employed requirements belong to second release and this release has highest mean change request per software requirement value. Third release has relatively fewer software requirements and less addition or modification is performed on requirements belong to this release. More than half of the requirements are modified at least once for this project; 9,848 out of 22,771 requirements are not changed which complies with Standish Group's survey results over more than 8,000 software projects [15].

3.3. Input Metrics

We have employed several metrics to predict the volatility of each software requirement in AVPRJ. The metrics represent three dimensions: requirement quality metrics, project specific metrics and requirement network metrics. Requirement quality metrics extracted by NASA Automated Requirements Measurement(ARM) tool have been used to predict faulty modules previously [16]. Initially, we believed the way requirements are documented will affect requirements volatility besides fault proneness of modules. Some requirement quality size metrics are already utilized in predicting requirements volatility [5], therefore we decided to include requirement quality metric set in our study. Network metrics are employed to predict requirement change volatility in a recent study [12]. This sparked the idea of utilizing network metrics for requirements volatility prediction. Initial observation of various change request notes confirmed that software requirements that are changed within a particular change request have a tendency to be linked to similar system requirements. Accordingly, we employed network metrics created by traceability information. In order to enrich input metric set with a new metric group we focused on safety-critical avionics project characteristics under this study. Features are evaluated separately and the ones would provide information on requirements volatility are selected as project specific metrics. Rationales of project specific metric selection are given in detail in subsection 3.3.2. Detailed explanations for each group are given in the following subsections.

3.3.1. Quality Metrics

While selecting requirement quality metrics to predict volatility, we have inspired by two studies. The first study propose requirement metrics in the context of NASA Metrics Data Program(MDP) to predict software faults [16]. These metrics are calculated by automatically going through requirements documents to highlight vague, ambiguous, long, complex requirements. The second study also reports requirement quality metrics [17] to find out which requirement quality analyze tool is more successful regarding measurement of those metrics.

Combining both studies' list and customizing that to the requirements document templates in our industrial context, we present 20 quality metrics in Table 2. All of these metrics take numeric values, e.g. number of flow sentences in a requirement, number of directives in a requirement.

During the preprocessing, stage, we had to remove three metrics from our analysis since they gave little to no information for AVPRJ: Conditional, Rationale and Subjective. Only one requirement contains conditional expressions, three requirements contain rationale expres-

Table 2
Requirement Quality Metrics

Acronyms	The number of abbreviations in a software requirement. For AVPRJ permitted acronym list is used to extract this metric.
Actions	The number of actions to be performed if conditions of a software requirement are satisfied.
Ambiguity	The number of ambiguous expressions in a software requirement, e.g. adequate, sufficiently, optimal, slow.
Chars between punctuation	Average character count between punctuation marks. Long sentences without punctuation marks decrease readability.
Conditions	The number of conditions need to be satisfied to perform a software requirement.
Conditional	The number of phrases that give the developers freedom to whether or not to implement a software requirement, e.g. maybe, can't, would.
Connectors	The number of connectors that are employed to link multiple sentences or group of words, e.g. and, or, as well as.
Directives	The number of directive expressions to refer a table, a note, a figure or an example.
Flow sentences	The number of expressions that semantically bond a sentence to another one, e.g. although, but, else.
Imperatives	The number of phrases that command to perform particular actions in a software requirement, e.g. shall, must, will.
Implicitness	The number of pronouns that make the software requirement difficult to understand, e.g. this, that, it. A software requirement should be defined explicitly.
Incompleteness	The number of expressions that indicate a software requirement is yet incomplete, e.g. and so on, tbd, etc.
In links	The number of incoming links to a software requirement from other documents. For AVPRJ test cases are linked to software requirements, so the number of in links refer to the number of linked test cases.
Negative Sentences	The number of phrases that give negative meaning, e.g. doesn't, none, can't.
Nested levels	For AVPRJ nested level metric value is the greatest level in hierarchical nesting structure of a software requirement.
Out links	The number of out links of a software requirement. In AVPRJ software requirements are linked to system requirements. Therefore the number of out links is the total number of linked system requirements by a software requirement.
Rationale	The number of expressions that give justification in a software requirement, e.g. thus, in order to.
Speculative Sentences	The number of speculative phrases which lead to question necessity of a software requirement, e.g. normally, eventually, almost.
Subjectivity	The number of subjective expressions presenting personal opinion rather than objectivity e.g. I think, in my opinion.
Text length	The total number of characters in a software requirement.

sions and none of the requirements have subjective expressions. Thus we ended up having 17 metrics representing the quality aspect of requirements for predicting their volatility.

3.3.2. Project Specific Metrics

Project specific metrics may differ regarding the scope of a software project, but the metrics we chose to use are not so specific to the development environment, programming language, or domain in which the software is developed. We believe project specific metrics would provide information about development characteristics in an organization, and hence the factors affecting the change proneness of requirements. Table 3 list these project specific metrics employed in this study. If the project follows an inspection activity on requirements, it is more likely that the team would find the ambiguities and inconsistencies on the requirements. Since derived requirements are not part of customer needs, they cannot be validated through user acceptance tests. If a requirement has a safety aspect, more comprehensive software tests will be performed, thus exposure of a potential change is highly probable. Number of related components is a measure of impact of a software requirement on general product, thus more feedback will be given to requirements affecting many components by development team. Each software release has different dynamics that affect requirements maturity e.g. release schedule, experience of developers, complexity of system. For example if sched-

Table 3
Project Specific Metrics

Inspection	Indicates if a software requirement is evaluated through an inspection activity. This procedure might be preferred to complement functional tests.
Derived	Software requirements that are not explicitly stated in system requirements but derived based on design decisions [18].
Safety	Shows if a software requirement is safety critical.
No. of Related Components	Number of isolated software components that a requirement is related.
Release Number	Release number that the software requirement belongs to.

Table 4
Network Metrics

Degree centrality	Gives score to requirements based on the number of links.
Betweenness centrality	Measures how many times a requirement is on the shortest path in the graph.
Closeness centrality	Indicates how close a requirement to other requirements considering the whole graph.
Eigenvector centrality	Measures how a node influences other nodes in network through connections.

ule is too tight to complete SRS document, requirements could be immature and more requirements changes could be performed in the future for this release.

3.3.3. Network Metrics

Hein et al. [12] earlier utilized 40 network metrics to predict requirements change volatility. On the other hand, Valente et al. [19] present correlations between degree, betweenness, closeness, eigenvector centrality measures, and indicate that those measures are distinct but notionally related. Thus in this work instead of employing 40 metrics, we chose the metrics suggested in [19] to predict requirements volatility for AVPRJ. These centrality metrics give each software requirement a value regarding their position in network. Brief explanations of the employed network metrics are given in Table 4.

Hein et al. [12] used language processing to create network for requirements. In this study instead we used traceability information to create network graph for software requirements. Traceability links from software requirements to system requirements are used for this purpose. We assigned weights between software requirements regarding system requirement traceability links. Software requirements which are derived from similar system requirements are tend to be closer in our model. Weight assignment formula is given below (Equation 1). W is weight between software requirements, NCLINK is the number of common system requirement links between two software requirements and NTOTLINK is the total number of system requirements linked from those two software requirements. After weight assignment, a symmetrical $n \times n$ matrix is created where n denotes the number of software requirements. Then the network metrics are computed over this matrix.

$$W_{ij} = \frac{NCLINK_{i,j}}{NTOTLINK_{i,j}} \tag{1}$$

3.4. Model Output

Our proposed model output is the number of change requests per software requirement. After the SRS document is reviewed and completed for AVPRJ, change requests linked to each software requirement are reported in the issue management system, and the document is modified accordingly by the analysis team. Thus we define requirements volatility in our industrial context with respect to number of change requests that have been applied to add a new requirement or to modify an existing requirement in the associated SRS document. Please note that our model outputs decimal values, but number of change request per requirement in practice can only get integer values. Therefore we round fractional parts to the nearest integer.

3.5. Tools

We wrote scripts to extract requirement quality and project specific metrics from SRS documents. Later, UCINET tool [20] is used to create network metrics from the matrix that we extracted based on software and system requirements. Regression models with different machine learners are trained using WEKA tool [21]. Prediction results are further post-processed in MATLAB to obtain the performance measures regarding all RQs.

3.6. Machine Learning Techniques

We train models using linear regression, random forest regression, support vector regression and k-nearest neighbor regression methods. Linear regression was utilized in [5], whereas classifier version of the other three techniques were used in [12].

For k-nearest neighbor regression, inversely proportional weighting option is selected. Higher weights are assigned to closer training samples which resulted in better prediction results for our model. For support vector

regression commonly used radial basis function kernel is selected. Increasing gamma parameter too much may result in over-fitting [22] and we also experienced a great computational cost with little to no prediction success gain for large gamma. Thus C and gamma parameters are assigned as 1.

In this study 10-fold cross validation technique is used to split training and test sets. Firstly, the dataset containing all software requirements is shuffled randomly and split into 10 groups of approximately equal size. One group is labeled as a test set and other groups are used to train machine learning models. This procedure is repeated 10 times until each unique group is used as test set once.

3.7. Performance Evaluation

For RQ 1, the following measures are used for performance evaluation: Mean Magnitude of Relative Error (MMRE), Median Magnitude of Relative Error (MdMRE), Pred(0.5) and Pred(0.25) [23]. Relative error is calculated according to Equation 2. $Err_{relative}$ is relative error, Val_{act} is the actual value, whereas Val_{pred} is the predicted value.

$$Err_{relative} = \frac{|Val_{act} - Val_{pred}|}{|Val_{act}|} \quad (2)$$

There are requirements with zero change requests. Thus division by zero problem arises while calculating relative error. We made an assumption for unchanged requirements as presented in Equation 3.

$$\text{If } Val_{act} = 0 \qquad Err_{relative} = \frac{|Val_{act} - Val_{pred}|}{1} \quad (3)$$

Pred(k) is a measure of variance of the error distribution. This measure is based on relative error and it shows the percentage of predictions whose errors are less than or equal to k.

For RQ2, we aim to predict highly volatile requirements, and thus, we first employ a method to identify those among the set of requirements:

- Step 1: Rank requirements by their actual number of change requests in descending order and record their rank as R_{actual}.
- Step 2: Obtain regression prediction results for each software requirement.
- Step 3: Rank requirements by their predicted number of change requests in descending order and record their rank as $R_{predicted}$.
- Step 4: Evaluate results according to the listing in Table 5. P denotes percentage of requirements which are perceived as highly volatile, and N_{req} denotes total number of requirements in validation set.

Table 5
Requirements volatility rank results evaluation

Condition	Evaluation
R_{actual},$R_{predicted} \leq N_{req} \times P$	True Positive
R_{actual},$R_{predicted} > N_{req} \times P$	True Negative
$R_{actual} \leq N_{req} \times P$, $R_{predicted} > N_{req} \times P$	False Negative
$R_{actual} > N_{req} \times P$, $R_{predicted} \leq N_{req} \times P$	False Positive

- Step 5: Calculate recall, accuracy and false alarm rate.

Table 5 can be interpreted as follows: True Positive instances are requirements that are actually highly volatile and the model also categorizes those as highly volatile. In the case of True Negatives, a requirement is actually less volatile, so is its prediction. False Negatives occur when highly volatile requirements are regarded as less volatile by the predictor. Finally, False Positives indicate less volatile requirements predicted as highly volatile.

To answer RQ2, recall, accuracy and false alarm rate measures are computed. Recall result shows how successful model in predicting highly volatile requirements. According to us this measure is the most important one regarding RQ2. Accuracy measure shows the prediction success for both highly volatile and less volatile requirements. False alarm rate presents how much effort has put in vain by mis-evaluating less volatile requirements.

4. Results and Discussion

We present and discuss the performance of the models with respect to two RQs in this section. We also compare the performance of the prediction models proposed in this study with the prior work [5] .

4.1. RQ 1

After obtaining processed data, machine learning regression methods are applied to answer the question if requirement quality metrics, network metrics, project specific metrics can be used to predict the number of changes on each software requirement by employing machine learning methods. Model performance results are gathered for all input metric and machine learning method combinations separately.

Results for RQ 1 is given in Table 6. The following abbreviations are used: ML for machine learning, Q for requirement quality metrics, P for project specific metrics, N for network metrics, KNN for k-nearest neighbor regression, LR for linear regression, RF for random forest regression and SVR for support vector regression.

In terms of input metric combinations, the best MMRE results are achieved with Q&P&N(0.366), Q&N(0.381) and

Table 6
Performance evaluation results for RQ 1

Metrics+ML method	MMRE	MdMRE	Pred(0.5)	Pred(0.25)
Q&P&N+KNN	0.366	0	0.681	0.57
Q&P&N+LR	0.53	0.5	0.524	0.411
Q&P&N+RF	0.392	0	0.663	0.545
Q&P&N+SVR	0.392	0	0.662	0.554
Q&P+KNN	0.402	0	0.641	0.529
Q&P+LR	0.513	0.5	0.541	0.428
Q&P+RF	0.45	0.333	0.6	0.486
Q&P+SVR	0.459	0.5	0.584	0.479
P&N+KNN	0.422	0	0.632	0.52
P&N+LR	0.55	0.667	0.484	0.372
P&N+RF	0.454	0.5	0.595	0.48
P&N+SVR	0.469	0.5	0.561	0.475
Q&N+KNN	0.381	0	0.665	0.553
Q&N+LR	0.534	0.5	0.52	0.407
Q&N+RF	0.394	0	0.662	0.545
Q&N+SVR	0.426	0	0.621	0.515
Q+KNN	0.443	0.333	0.598	0.488
Q+LR	0.512	0.5	0.542	0.43
Q+RF	0.455	0.5	0.594	0.483
Q+SVR	0.483	0.5	0.556	0.446
P+KNN	0.555	0.667	0.483	0.37
P+LR	0.548	0.667	0.485	0.373
P+RF	0.556	0.667	0.483	0.371
P+SVR	0.516	0.5	0.512	0.417
N+KNN	0.448	0.5	0.596	0.482
N+LR	0.549	0.667	0.484	0.372
N+RF	0.485	0.5	0.561	0.446
N+SVR	0.53	0.5	0.5	0.392

Table 7
Comparison of our performance (RQ 1) against [5]

	MMRE	MdMRE	Pred(0.25)	Pred(0.5)
Q+LR	0.51	0.5	0.43	0.54
Best model	0.36	0	0.57	0.68
NLines+LR [5]	0.58	0.27	0.5	0.63

Q&P(0.402). We may interpret that requirement quality metrics (Q) are successful at predicting number of change requests per software requirement, and its combinations with the other metrics also give good results. With respect to the machine learning algorithm, the three best performing metric combinations give the highest prediction performance when k-nearest neighbor algorithm is utilized.

MdMRE is zero for the following metric and machine learner combinations: Q&P&N+KNN, Q&P&N+RF, Q&P&N+SVR, Q&P+KNN, P&N+KNN, Q&N+KNN, Q&N+RF and Q&N+SVR. Number of change requests for more than half of the software requirements are predicted correctly with these models. Since many prediction models give the same best result, we do not rank the best performing models with regard to MdMRE.

The best performance with respect to Pred(0.25) and Pred(0.5) are obtained with Q&P&N+KNN. Q&N+KNN and Q&P+KNN report the second and third best results. Those results indicate that requirement quality metrics and k-nearest neighbor algorithm are also successful with respect to Pred measures.

To sum up, best performance results are achieved by using requirement quality metrics, project specific metrics and network metrics altogether. Accordingly, K-nearest neighbor algorithm gives best performance results for all measures. In all best performing models, quality metrics are utilized either as a pair with project or network metrics or as combination of all three. It seems the way requirements are documented has a high effect on the volatility rates.

We compared our findings against the study conducted by Loconsole et al. [5]. Table 7 reports the performance of linear regression model with the best metric set in our study, our best performing model and the best performing model of [5]. If we compare the findings only on LR, we observe that using number of lines predicts volatility better on their commercial setting, while in our context using quality metrics only does not give the best result. Other algorithms like KNN in combination with all metrics significantly improve the prediction performance by reducing MMRE down to 0.36 and MdMRE down to 0, and increasing Pred(0.25) up to 57%.

4.2. RQ 2

RQ 2 aims to measure the success of our model in predicting highly volatile requirements. We present our technique to identify highly volatile requirements in Section 3.7. We first need to determine change request coverage to categorize highly-volatile requirements, and later calculate recall, accuracy and false alarm rates. Rates for various change request coverage by most volatile requirements are given in Table 8. As change request coverage grows more requirements are labeled as highly volatile. We chose 80% change request coverage since approximately 40% percent of reviewers are considered as well-experienced in AVPRJ. Therefore by applying this model we could assign review task of 38.6% of total requirements, which are possibly highly volatile, to experienced developers in early phase of development. We did not present other coverage results in this study due to page limitation.

In Table 9 the best recall results are achieved with Q&P&N+KNN(0.632), Q&N+RF(0.616) and Q&P+KNN(0.604). Again, all best performing models have requirement quality metrics in common, whereas the best combination consists of all metrics. The best accuracy results are obtained from Q&P&N+KNN(0.716),

Table 8
Change request and requirement coverage relation for AVPRJ

CR Coverage Percent	REQ Coverage
60%	20.5%
70%	29.6%
80%	38.6%
90%	47.7%

Table 9
Performance results of RQ 2 model for 80% CR coverage

Metrics+ML method	Recall	Accuracy	False Alarm Rate
Q&P&N+KNN	0.632	0.716	0.232
Q&P&N+LR	0.531	0.638	0.295
Q&P&N+RF	0.624	0.71	0.237
Q&P&N+SVR	0.591	0.684	0.258
Q&P+KNN	0.604	0.694	0.249
Q&P+LR	0.508	0.62	0.31
Q&P+RF	0.602	0.692	0.251
Q&P+SVR	0.558	0.658	0.278
P&N+KNN	0.574	0.671	0.268
P&N+LR	0.418	0.55	0.366
P&N+RF	0.557	0.658	0.279
P&N+SVR	0.496	0.61	0.318
Q&N+KNN	0.614	0.702	0.243
Q&N+LR	0.53	0.637	0.296
Q&N+RF	0.616	0.703	0.242
Q&N+SVR	0.569	0.667	0.271
Q+KNN	0.586	0.68	0.261
Q+LR	0.508	0.62	0.31
Q+RF	0.582	0.677	0.263
Q+SVR	0.529	0.636	0.296
P+KNN	0.503	0.616	0.313
P+LR	0.423	0.554	0.363
P+RF	0.496	0.611	0.317
P+SVR	0.462	0.584	0.339
N+KNN	0.557	0.657	0.279
N+LR	0.404	0.539	0.376
N+RF	0.565	0.664	0.274
N+SVR	0.498	0.612	0.316

Q&N+RF(0.703) and Q&P+KNN(0.694). The lowest false alarm rate results are achieved by Q&P&N+KNN(0.232), Q&N+RF(0.242) and Q&P+KNN(0.249). K-nearest neighbor and random forest regression methods are successful in predicting highly volatile requirements for 80% change request coverage.

The most important measure for RQ 2 is recall since the purpose of this question is to measure success on predicting highly volatile requirements. We correctly identify 63.2% of highly volatile requirements which are exposed to 80% of the total requirement changes.

4.3. Threats to Validity

Internal validity: In this study we present requirements volatility in a software project can be predicted to some extent utilizing requirement quality metrics, project specific factors and network metrics altogether. However, this does not imply causal relationship between input and output metrics since we did not conduct a controlled experiment.

External validity: We have conducted the case study on one project, so results have local validity. However, the dataset is quite large with more than 20,000 requirements from three distinct releases developed by many software developers. Nonetheless, applying the predictive models on different projects in the future would be better in terms of generalization of results.

Construct validity: Developers did not use their native language in software requirements. Thus there could be some typos which may affect textual requirement quality metrics. Also there could be some expressions used by developers in software requirements, e.g. subjective expressions that should have taken into consideration while creating requirement quality metrics but we missed. Due to the size of dataset we couldn't manually check these kinds of typos and grammatical errors, but we know that reviewers are responsible for correcting those. We create network graphs based on traceability links between software and system requirements as indicated in the SRS. We could have use linguistic data to connect software requirements as previous study [12] and it may reflect relationship between requirements in a better way. We plan to do it as a future work.

Conclusion validity: For RQ 2 only the results of 80% change request coverage are presented due to page limitation. Regarding the results of other CR coverage, we observe higher recall and accuracy whereas false alarm rate grows undesirably as the coverage grows. Therefore RQ 2 results would differ in that way if we had chosen other CR coverage rate.

5. Conclusion and Future Work

In this paper, we have carried out an empirical study to predict number of changes per software requirement by using requirement quality measures, project specific factors and requirement interdependencies. 22,771 software requirements from a safety-critical software project in ASELSAN are utilized to build 28 prediction models and assess the best performing metric suite and algorithm. We conclude that we can predict volatility of requirements with an average MMRE of 36% by observing metrics of similar requirements through KNN. We also observe that measuring requirements from different aspects like quality, project and network dependencies gives a much better performance. We plan to integrate

such a predictor model into requirement management tools like DOORS to be used prior to the SRS review activity so that highly-volatile requirements could be automatically and accurately identified. This way, software development leads could take precautions beforehand to reduce requirements volatility related risks. Since there is not enough empirical studies conducted in related area, more empirical research should be carried out to validate the best performing models.

References

[1] N. Nurmuliani, D. Zowghi, S. Powell, Analysis of requirements volatility during software development life cycle, in: 2004 Australian Software Engineering Conference, ASWEC '04, IEEE, 2004, pp. 28–37.

[2] G. Swathi, A. Jagan, C. Prasad, Writing software requirements specification quality requirements: An approach to manage requirements volatility, Int. J. Comp. Tech. Appl. 2 (2011) 631–638.

[3] R. Thakurta, A mixed mode analysis of the impact of requirement volatility on software project success, Journal of International Technology and Information Management 20 (2011).

[4] A. M. Alsalemi, E.-T. Yeoh, A systematic literature review of requirements volatility prediction, in: 2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication, ICCTCEEC-2017, IEEE, 2017, pp. 55–64.

[5] A. Loconsole, J. Börstler, Construction and Validation of Prediction Models for Number of Changes to Requirements, Umeå University Technical Report UMINF-07.03, Umeå University Department of Computing Science, UMEÅ, SWEDEN, 2007.

[6] D. F. X. Christopher, E. Chandra, Prediction of software requirements stability based on complexity point measurement using multi-criteria fuzzy approach, International Journal of Software Engineering & Applications 3 (2012) 101–115.

[7] L. Shi, Q. Wang, M. Li, Learning from evolution history to predict future requirement changes, in: 2013 21st IEEE International Requirements Engineering Conference, RE-2013, IEEE, 2013, pp. 135–144.

[8] A. Goknil, R. van Domburg, I. Kurtev, K. van den Berg, F. Wijnhoven, Experimental evaluation of a tool for change impact prediction in requirements models: Design, results, and lessons learned, in: 2014 IEEE 4th International Model-Driven Requirements Engineering Workshop (MoDRE), MoDRE 2014, IEEE, Karlskrona, Sweden, 2014, pp. 57–66.

[9] B. Morkos, J. Mathieson, J. D. Summers, Comparative analysis of requirements change prediction models: manual, linguistic, and neural network, Research in Engineering Design 25 (2014) 139–156.

[10] X. Wang, C. Wu, L. Ma, Software project schedule variance prediction using bayesian network, in: 2010 IEEE International Conference on Advanced Management Science, volume 2 of *ICAMS 2010*, IEEE, Chengdu, China, 2010, pp. 26–30.

[11] T. Nakatani, T. Tsumaki, Predicting requirements changes by focusing on the social relations, in: Proceedings of the Tenth Asia-Pacific Conference on Conceptual Modelling, volume 154 of *APCCM 2014*, Auckland, New Zealand, 2014, pp. 65–70.

[12] P. H. Hein, E. Kames, C. Chen, B. Morkos, Employing machine learning techniques to assess requirement change volatility, Research in Engineering Design 32 (2021) 245–269.

[13] W. Pedrycz, J. Iljazi, A. Sillitti, G. Succi, Prediction of the successful completion of requirements in software development—an initial study, in: Agent and Multi-Agent Systems: Technology and Applications, Springer, 2016, pp. 261–269.

[14] T. J. Ostrand, E. J. Weyuker, How to measure success of fault prediction models, in: Fourth international workshop on Software quality assurance: in conjunction with the 6th ESEC/FSE joint meeting, SOQUA'07, Dubrovnik, Croatia, 2007, pp. 25–30.

[15] T. Clancy, The standish group report, Chaos report (1995).

[16] Y. Jiang, B. Cukic, T. Menzies, Fault prediction using early lifecycle data, in: The 18th IEEE International Symposium on Software Reliability, ISSRE'07, IEEE, 2007, pp. 237–246.

[17] G. Génova, J. M. Fuentes, J. Llorens, O. Hurtado, V. Moreno, A framework to measure and improve the quality of textual requirements, Requirements engineering 18 (2013) 25–41.

[18] A. Faisandier, Systems architecture and design, Sinergy'Com Belberaud, France, 2013.

[19] T. W. Valente, K. Coronges, C. Lakon, E. Costenbader, How correlated are network centrality measures?, Connections 28 (2008) 16–26.

[20] S. P. Borgatti, M. G. Everett, L. C. Freeman, Ucinet for windows: Software for social network analysis, Harvard, MA: analytic technologies 6 (2002).

[21] F. Eibe, M. A. Hall, I. H. Witten, The weka workbench. online appendix for data mining: practical machine learning tools and techniques, in: Morgan Kaufmann, 2016.

[22] A. Ben-Hur, J. Weston, A user's guide to support vector machines, in: Data mining techniques for the life sciences, Springer, 2010, pp. 223–239.

[23] D. Zhang, J. J. Tsai, Machine learning applications in software engineering, volume 16, World Scientific, 2005.

Towards a Catalog of Refactoring Solutions for Enterprise Architecture Smells

Lukas Liss[1], Henrik Kämmerling[1], Peter Alexander[1] and Horst Lichter[1]

[1]RWTH Aachen University, Research Group Software Construction, Ahornstrasse 55, Aachen, Germany

Abstract

The model of enterprise architecture (EA) is often a primary means of steering the business-IT development. It helps to ensure EA qualities in many ways, including identification of weaknesses in an EA or the signs thereof. Such weaknesses have been addressed through the study of EA smell, which focuses on common bad habits in EA practices. Although EA smell problems have been described in various contexts, current solutions lack the basis of evidence and practical details that are necessary for their adoption. Therefore, this study seeks to gain new insights into the solution domain of EA smells by exploring current knowledge about refactoring solutions. We present our findings in a catalog of EA refactoring solutions intended to serve as food for thought for future research directions.

Keywords

enterprise architecture, enterprise architecture smell, refactoring solution

1. Introduction

The model of enterprise architecture (EA) greatly supports sustainable management of complex IT landscapes. It provides insights into the current implementations and future orientations of the EA developed, thereby providing a reasoning basis for important decisions. Nevertheless, the maintenance of the EA model is often pushed down the priority list due to, e.g., lack of resources or supporting methods. Flaws and deficiencies in the EA may remain ignored and create barriers to EA evolution known as EA debt [1]. To prevent such tendencies, practical methods for supporting the continuous evaluation and improvement of EA models are needed.

Using EA models to guide the improvement of EA qualities has been proposed by some studies. Salentin and Hacks coined the concept of EA smell to address common bad habits in EA practices and derived EA smells [2] by transferring known code smells into the context of EA. Lehmann et al. made a similar attempt to derive process modelling anti-patterns for EA analyses [3] by transferring known workflow anti-patterns into the context of EA. Both of these studies suggest, among others, some solutions to refactor the problems they identify. However, existing descriptions of these solutions lack the basis of evidence and practical details that are necessary for their application. This study therefore focuses on exploring current knowledge about refactoring solutions to find further evidence about EA refactoring solutions and new ways of approaching them.

Considering that current EA smells were derived from code smells, we argue that exploring known code refactoring solutions can result in meaningful insights into the refactoring solutions for EA smells. Therefore, based on the code smells used in EA smell studies, we first searched for relevant code refactoring solutions in major code smell and refactoring catalogs. The code refactoring solutions collected were then used to answer the following research questions (RQ):

RQ 1 What EA refactoring solutions can be derived from the existing code refactoring solutions?

RQ 1.1 How to derive insights about EA refactoring from analyzing code refactoring solutions?

RQ 1.2 What attributes can describe an EA refactoring solution?

RQ 2 How can EA practitioners and researchers benefit from knowing about EA refactoring solutions?

The remainder of this paper is structured as follows: Section 2 gives an overview of studies related to refactoring solutions. Section 3 describes our methodology for deriving and documenting refactoring solutions for EA smells. Section 4 presents the resulting EA refactoring solutions mapped to the corresponding EA smells. Section 5 demonstrates some small practical examples of using EA refactoring solution for mitigating EA smells. Section 6 discusses our finding, its implications, and the threats to its validity. Section 7 motivates future research directions and concludes this paper.

QuASoQ 2021: 9th International Workshop on Quantitative Approaches to Software Quality, December 06, 2021, Taipei, Taiwan
lukas.liss@rwth-aachen.de (L. Liss);
henrik.kaemmerling@rwth-aachen.de (H. Kämmerling);
alexander@swc.rwth-aachen.de (P. Alexander);
lichter@swc.rwth-aachen.de (H. Lichter)
0000-0001-6534-278X (P. Alexander); 0000-0002-3440-1238 (H. Lichter)

2. Related Work

The concept of refactoring has been developed to support software design and evolution, specifically to rectify design flaws and improve software quality through restructuring plans while preserving the observable behavior. The term 'refactoring' can be used in different ways, namely 'refactoring' (noun) to mention the restructuring applied to a software [4], 'to refactor' (verb) to mention the activity of restructuring a software [4], or 'refactoring solution' to mention a possible technique for restructuring a software (e.g. [5]). For the sake of clarity, these ways of usage are applied in this paper.

After being extensively used in programming domains, such as the procedural programming (e.g. [6]), object-oriented programming (e.g. [5]), and functional programming (e.g. [7]), the concept of refactoring has been explored in a wider spectrum of software engineering. In the domain of software modelling, the concept of model refactoring was introduced to enable restructuring plans on the high-level description of software [8]. The aim of model refactoring can be twofold: to allow for an earlier (i.e. at the design stage) and easier (i.e. reduced complexity) refactoring of software; or to improve semantic and syntactic quality measures of the model. Whichever aim is set, model refactoring should preserve both the behavior of the modelled software and the semantic of the model.

Studies of model refactoring have varied in their focus, whether it be on a specific modelling language or particular architectural aspect. Modelling languages such as the object constraint language (OCL), unified modelling language (UML), and web ontology language have been studied and supported with designated refactoring solutions [9] [10] [11]. Architectural aspects such as process aspect, data aspect, and style aspect have also been analyzed in this context to support a comprehensive architecture restructuring [12] [13] [14]. Furthermore, the so-called large refactorings have been proposed to deal with refactoring in complex projects, specifically when changing significant parts of a system, which often takes longer than a day [15]. Still, there are growing possibilities to explore new refactoring solutions for new design anomalies, especially with the growing interest in bad smell research—which focuses on detecting concrete indication of the need for refactoring [4].

The concept of bad smell has been adopted in the domain of EA and referred to as EA smell, which represents negative examples and bad habits that, when ignored, may harm the performance of EA activities or even the organization as a whole [2]. The concept was proposed together with a catalog of 45 EA smells, which describes (among others) the problem contexts, applicable detection methods, and possible mitigation solutions. Based on the scope of occurrence, an EA smell can be of the business, application, and/or technology architecture. This scope was recently expanded by the exploration into EA process anti-pattern, which resulted in 18 EA smells for process-related issues [3]. As the interest in EA smell research is growing, more EA smells will continue to be identified through new explorations into other aspects of EA, such as management.

3. Methodology

This study uses the design science research (DSR) methodology according to the guidelines proposed by Peffers et al. [21] and Hevner et al. [22]. A DSR aims to devise an artifact that addresses a "heretofore unsolved and important business problem" by drawing on the existing knowledge, and the resulting artifact must be rigorously evaluated in terms of its "utility, quality, and efficacy" and effectively communicated to relevant audiences. With respect to the DSR guidelines above, this study follows the six main steps of DSR as listed below.

1. **Problem identification and motivation.** Define the specific research problem and justify the value of the solution proposed
2. **Solution objective definition** Infer the objectives of the solution proposed from the problem definition and knowledge of what is possible and feasible
3. **Design and development.** Create the artifact
4. **Demonstration.** Demonstrate the use of the artifact to solve one or more instances of the problem
5. **Evaluation.** Observe and measure how well the artifact supports a solution to the problem
6. **Communication.** Communicate the problem and its importance, the artifact, its utility and novelty, the rigor of its design, and its effectiveness to researchers and other relevant audiences

The following subsections describe the process and methods employed in this study with respect to the steps above.

3.1. Problem identification and motivation

Despite current results in EA smell research (as presented in section 2), the identification of EA smells is never an end in itself. The identified EA smells should, e.g., inform the decisions for imposing suitable EA improvement measures with regards to the current circumstances and interests. To reason such decisions, enterprise architects need to first understand the applicability of each solution alternative through evidence and practical details,

UML → ArchiMate	Source	UML → ArchiMate	Source
Activity → Interaction, Function, Process	[16, 17, 18]	**Interface** → Interface, Service	[19, 16, 17, 20, 18]
Actor → Actor, Role	[19, 16, 17, 20]	**Method** → Application/Infrastructure Function, Infrastructure Service	[16]
Artifact → Artifact, Contract, Deliverable, Gap, Product, Representation	[19, 16, 17, 20]	**Note/Comment** → Meaning, Representation	[18]
Association → Association, Aggregation, Composition	[17, 20, 18]	**Node** → Node, Communication Path, Device, Infrastructure Interface, Location, Network, System Software	[19, 16, 17, 20]
Class → Actor, Collaboration, Component, Data Object, Meaning, Motivational Concepts, Object, Plateau, Representation, Role, Stakeholder, Value	[19, 16, 20, 18]	**Object** → Object, Contract, Data Object, Meaning, Product, Representation, Value	[16]
Collaboration → Collaboration, Function, Interface, Location	[19, 16, 17, 20, 18]	**Opaque Behavior** → Application Interaction, Business Event, Business Interaction, Business Process, Junction, Work Package	[20]
Component → Component, Grouping	[19, 16, 17, 20]	**Swimlane** → Actor, Business Service, Role	[16]
Communication Path → Communication Path, Network	[17, 20, 18]	**Usage** → Access, Serving/Used By	[20]
Dependency → Assignment, Derived, Influence, Serving/Used by	[17, 20]	**Use Case** → Business Function, Business Interaction, Requirement, Service	[19, 16, 17, 20]
Device, Event, Execution Environment → Device, Event, System Software	[19, 16, 17, 20]	**Realization, Composition, Aggregation** → Realization, Composition, Aggregation	[19, 17, 20, 18]
Generalization → Specialization	[19, 20, 18]		
Information Flow → Flow, Triggering	[20]		
Interaction → Interaction, Application Service, Event, Function, Process	[16, 18]		

Table 1
A mapping from UML to ArchiMate

which are still lacking in the hitherto solution suggestions for EA smell mitigation. Furthermore, current knowledge about the refactoring of architectural smells focuses on rather technical aspects (i.e. software architecture), which are hardly applicable for evaluating and evolving an EA. Therefore, we argue that EA research should pursue the identification of possible EA refactoring solutions to guide enterprise architects in finding possible and feasible solutions for the EA smell problems at hand.

3.2. Solution objective definition

Considering the problem described above, this study aims to devise the concept of EA refactoring solution by drawing on the existing knowledge about refactoring. To guide the solution design and development, this study sets the following solution objectives:

1. Support enterprise architects in finding appropriate solutions to a certain EA smell by identifying possible EA refactoring solutions and providing clear descriptions about their context and implementation

2. Support enterprise architects in communicating relevant EA refactoring solutions in a consistent manner by identifying relevant attributes and providing a standard documentation template

3. Allow the EA community to obtain an up-to-date overview of results in EA refactoring research and contribute to the its development

3.3. Design and development

Since the first EA smells were derived from code smells [2], we assume that refactoring solutions for code smells may hold some clue to the refactoring solutions for EA smells. This assumption leads to the design of our methodology, which includes three main steps: Concept analysis, concept mapping, and concept transformation.

Concept Analysis. Our first step is to collect relevant code refactoring solutions for the code smells used by existing studies on EA smells [2]. Our analysis covered some existing code smells or anti-patterns catalogs (i.e., [4], [23], and [24]). While we are aware that "the presence of code smells does not imply the presence of architectural smells and vice versa," [25], we argue that some problematic design patterns addressed by code smells or anti-patterns (e.g. cyclic dependency or duplication) may also occur in EA context. Furthermore, the practice of transferring existing concepts into a different domain has been performed to generate first ideas within a new problem domain and inspire further research (e.g. [2], [3]).

Attribute	Meaning
Name	Gives the refactoring solution a meaningful designator
Connected EA Smells	Links the refactoring solution to the targeted EA smells
Derived from	Names the code refactoring solution from which this refactoring solution is derived
Summary	Gives a brief overview of this refactoring solution
Intent	Describes the main goal of this refactoring solution
Motivation	Describes the reasons of applying this refactoring solution
Prerequisites	Describes the conditions prior to applying this refactoring solution
Impact	Describes the influence of this refactoring solution to EA qualities
Mechanics	Describes the practical steps to apply this refactoring solution
Discussion	Describes the situations when this refactoring solution can be useful
Graphical example	Shows a graphical representation of the refactoring solution to reduce misinterpretations

Table 2
Attributes of EA refactoring solutions (adapted from [4], [26], and [27]).

Concept Mapping. The second step in our methodology is to establish a mapping between the concepts in code and EA modelling, which should serve as a basis for conceptually transforming the code refactoring solutions collected into EA refactoring solutions.

Although the code and the EA lie on two opposite sides in the spectrum of abstraction, the languages used for modelling them share a good deal of similar constructs in that both support the description of architectural aspects. Previous studies have proposed some mappings between the notations of UML and ArchiMate, which are the most used languages for code and EA modelling, respectively. Some of these mappings are described in the ArchiMate specification [17]—as some notations thereof are indeed derived from UML [19]—, while the rest have been exclusively suggested by the studies. Since most code refactoring solutions have been described and exemplified using UML, we strongly argue that such mappings can guide the transformation of code refactoring solutions into EA refactoring solutions.

We believe that the first mapping between the concepts in UML and ArchiMate was proposed by Wiering et al. [18]. Their study focuses on identifying matching concepts and relationships by comparing the notations in the two languages by the properties thereof. Another attempt was made in a study by Gill, which focuses on finding concepts in other modelling languages beside ArchiMate which are applicable for EA modelling [16]. The resulting contribution includes a mapping of some concepts in UML to ArchiMate. Furthermore, Lankhorst also advocate the use of ArchiMate in conjunction with other modelling languages (including UML) to create a more holistic EA description [19]. To support this in practice, this contribution highlights not only the similarities but also important differences in ArchiMate and UML, e.g. the absence of separate concepts for *service* and *interface* as well as the reverse interpretation of the ArchiMate relationship *serving* in UML. Lastly, another mapping of UML and ArchiMate is proposed by Gericke, which also considers ArchiMate concepts under the motivation layer [20]. Table 1 summarizes all these mappings which we use as a basis to conduct the concept transformation.

Concept Transformation. To finally derive EA refactoring solutions, the code refactoring solutions collected are processed as follows:

- We analyzed the descriptions and examples of code refactoring solutions to identify instances of UML concepts and relationships. Based on the mapping of UML and ArchiMate notations shown in table 1, the identified UML constructs are translated into ArchiMate constructs. Since each UML notation does not necessarily map exclusively with one ArchiMate notation (e.g. both Collaboration and Interface in UML can be translated into Interface in ArchiMate), the translation has to be inferred rationally from our understanding of what solution is possible and feasible for the EA smell addressed.
- In some cases, the translation may not directly result in an ArchiMate construct that conveys a valid solution in the context of EA. Such ArchiMate construct is either modified for a reasonable EA refactoring solution or excluded from consideration.

Because of the manual concept transformation in this process, the outcome is highly influenced by the skill, knowledge, and experience of the researchers. Also, since the mapping used gives more than one matching ArchiMate notations for every UML notation, the resulting translation may vary depending on the researcher's own understanding and interpretation of the two modelling languages (UML and ArchiMate).

EA smell → EA refactoring solutions	EA smell → EA refactoring solutions
Ambiguous Viewpoint → 5 Viewpoints	Jumble → Architecture Partitioning
Architecture by Implication → Goal Question Architecture	Lazy Component → Ghostbusting or Inline Service
Big Bang → Process Manager	Message Chain → Process Manager or Extract & Move Data Object
Bloated Service → Merge input	Missing Abstraction → Add Abstraction
Business Process Forever → Process Manager	Multifaceted Abstraction → Split Phase
Chatty Service → Front End Gate	Nanoservices → Front End Gate
Combinatorial Explosion → Extract Shared Functionality	No Legacy → Process Manager
Connector Envy → Extract Component	Nothing New → Process Manager
Cyclic Dependency → Cyclic Dependency Removement	Overgeneralization → Extract Shared Functionality
Data-Driven Migration → Functionality First, Data Last	Sand Pile → Grouping
Data Service → Encapsulate Data Service	Scattered Parasitic Function → Merge Components
Dead Component → Remove Dead Component	Shared Persistency → Encapsulate Business Objects
Deficient Encapsulation → Break Up Component	Shotgun Surgery → Grouping
Deficient Names → Rename Component	Stovepipe System → Architecture Framework or EA Planning
Dense Structure → Complexity Reduction	Strict Layers Violation → Move Service to Different Layer
Documentation → Rename Component	Temporary Solution → Extract Temporary Solution
Duplication → Extract Shared Functionality	The God Object → God Object Decomposition
Feature Envy → Move Component or Front End Gate	The Shiny Nickel → Plan Ahead
Golden Hammer → Boundaries	Vendor Lock-In → Isolation Layer
Hub-like Modularization → Break Up Component	Warm Bodies → Small Project
Incomplete Abstraction → Grouping	Weakened Modularity → Split Modularity
Incomplete Node or Collaboration → Introduce Local Extension	Wrong Cuts → Reorganization
Inconsistent Versioning → Semantic Versioning	

Table 3
Catalog of EA Refactoring Solutions

3.4. Demonstration

In this DSR step, the use of the artifact is demonstrated to solve one or more instances of the problem [21]. The fundamental questions to be answered are, "What utility does the new artifact provide?" and "What demonstrates that utility?" [22] In the context of this study, the EA refactoring solutions proposed should provide solution alternatives for mitigating EA smells and the practical details needed to understand their feasibility as well as relevance according to the current conditions. Therefore, to demonstrate the applicability of our result for solving the problems described in section 3.1, we illustrate some small scenarios of mitigating EA smells using the EA refactoring solutions proposed. For this purpose, we use a small ArchiMate model which contains some EA smells. Then, we identify candidates of EA refactoring solutions and select one that we deem the most appropriate for the EA smell given. It is worth noting that the systematic approach to prioritizing EA refactoring solution candidates is still a subject to future work; hence, the selection of EA refactoring solution demonstrated in this paper relies on the authors' perspectives. Finally, we elaborate the architectural changes suggested by the EA refactoring solution selected, show the resulting ArchiMate model after applying these architectural changes, and discuss the impact on the ArchiMate model's overall quality.

3.5. Evaluation

The evaluation step in DSR aims to review the effectiveness of the artifact proposed in supporting a solution to the problem [21]. In this paper, this step is embodied in a discussion, in which we explain how the main RQs of this study have been answered based on our findings and their demonstration. Since the discussion presented in this paper is focused only on the qualitative performance of EA refactoring solution within our example scenario, we recommend future research to extend the evaluation by including more (real-world) examples and quantifiable measures (e.g. using EA model quality framework [28]).

3.6. Communication

Finally, the communication step focuses on presenting the artifact proposed, its utility, and its effectiveness to researchers and other relevant audiences. [21] To communicate our results in an intuitive and consistent manner, we document the EA refactoring solutions in a standard template (see table 2) which we adapted from some existing templates of refactoring solution used in [4] [26] [27]. Furthermore, the EA refactoring solutions are categorized based on the domains of the corresponding EA smells (i.e. business, technology, and application domains) [2] and made publicly available on a web page [29].

Name	Process manager	Encapsulate Business Objects
Description	This refactoring increases the user orientation of service interfaces with the goal to enable user involvement in business processes. A process manager that realizes the service calls is created. Previous caller of the process call the process manager now with process control data that parameterizes the wanted process.	Route data access through dedicated data services that encapsulate the needed business objects.
Connected EA smell	Business Process Forever, Big Bang, Nothing New, No Legacy	Shared Persistency
Derived from	Process Manager	Encapsulate Variable
Intent	Create a process manager component which orchestrates the components involved in the process.	Narrow down the data visibility by routing data access through dedicated data services.
Motivation	This refactoring is meant to create a better user involvement. It also adds more flexibility and extensibility to the architecture.	Reduce the likeability that teams unwillingly depend on each other because the visibility of the data is too broad.
Prerequisites	A long, very strict process chain with many different stakeholders involved.	Multiple business services that access the same database.
Impact	This refactoring makes the application service interfaces more user oriented which enables the user involvement in business processes.	The structure created by the refactoring reduces data visibility to only those of the owned business objects, thereby preventing any unwanted interdependence between teams and services.
Mechanics	1. Generate a new service P called the Process Manager 2. Add a service call from P to all the process components of the Process manager 3. All calls to the individual process components calls now the Process Manager with process control data C that parameterize the wanted original Process. Test each call individually. 4. Remove the service calls between the services that belong now to the Process Manager P	1. Create a new data service that will contain the dedicated data services 2. For each business service, create a dedicated data service that routes the access to the database. Test each data service. 3. Move the database into the grouping data service.
Discussion	This is not only a refactoring but also an architecture pattern itself. It can be used to generate more online involvement for everyone that should be involved. In a peer-to-peer system it is usually not desired to have many centralized services, so the process manager needs to be evaluated carefully.	The data visibility is very narrow when using this refactoring which reduces unwanted dependencies between teams.

Table 4
Documentation of the Process Manager and Encapsulate Business Objects EA refactoring solutions

4. Result

As a result, this study yields a catalog of 37 EA refactoring solutions for all 45 EA smells proposed in [2] (see table 3). In the catalog, some EA refactoring solutions are suggested for multiple EA smells, such as the Process Manager EA refactoring solution which is suggested for the Big Bang, Business Process Forever, Message Chain, No Legacy, and Nothing New EA smells. Such result is obtained because some code refactoring solutions–from which we derived EA refactoring solutions–have also been suggested for mitigating various code smells. While we are aware that specific adaptations may be necessary to make one kind of solution works for different problems, the catalog currently provides only a general description of how to apply each EA refactoring solution. Describing such adaptations is a subject to future work.

Furthermore, the catalog also suggests multiple EA refactoring solution candidates for some EA smells, such as the Move Component and Front End Gate which are suggested as solution candidates for mitigating the Feature Envy EA smell. Such result is obtained because we identified different studies which suggest different code refactoring solutions for the same code smell. While we are aware that the selection of an appropriate EA refactoring solution should consider the specific circumstances surrounding the EA smell problem, such specific circumstances are beyond the scope of this study. Therefore, we suggest future studies in this context to investigate the possibility to systematically prioritize all EA refactoring solutions recommended for a certain EA smell. In such a prioritization approach, various quality requirements may be considered, such as scalability and extensibility.

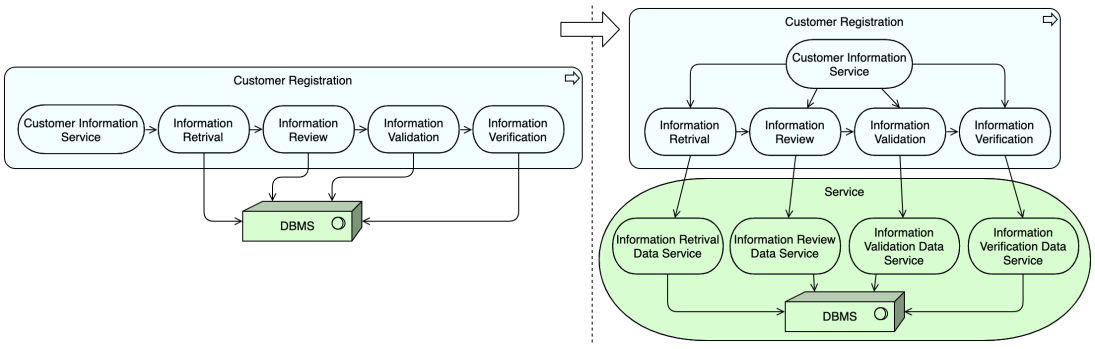

Figure 1: EA refactoring demonstration, showing before (left) and after (right) refactoring EA smells in an ArchiMate model

5. Demonstrating EA Refactoring Solutions

In order to illustrate the application of the proposed EA refactoring solutions, we performed an experiment to apply EA refactoring solutions on a small ArchiMate model (see fig. 1), which we adapted from [2]. This ArchiMate model contains two EA smells: First, a *message chain* due to the sequential calls over 5 application services to fulfill the same application process (i.e. CUSTOMER REGISTRATION). Second, a *shared persistency* because of the direct relations between 3 application services and a database management system (i.e. DBMS).

The first EA smell (i.e. *message chain*) occurs when at least 5 services sequentially interact to fulfill a process, which may harm availability and evolvability [30]. To solve this, our catalog (see table 3) proposes two possible EA refactoring solutions, i.e. the **process manager** (inspired from 'process manager' in [31]) and **extract and move data object** (adapted from 'extract and move class' in [4]). To select from several possible refactoring solutions, the architect must first evaluate whether the main idea and practical details thereof suit the context of the subjected EA smell. In the presented ArchiMate model, we assume that the exemplified *message chain* does not stem from data but process architecture issues. Therefore, the **process manager** is chosen, as presented in table 4.

The chosen refactoring solution suggests to restructure the collaboration scheme between the chained services into an orchestration scheme—in which one service acts as a 'process manager' that coordinates independent services to fulfill the requested business process. This scheme eases the maintenance and extension of the supported business process, thereby remediating the availability and evolvability issues posed by the *message chain*. In our ArchiMate model, this refactoring solution is applied by first encapsulating all involved operations across the chained application services into independent activity modules that are accessible through external interfaces. Afterwards, a process manager must be implemented (in this case, the CUSTOMER INFORMATION SERVICE) which receives requests for customer information processing and fulfills these by delegating tasks to the available services.

The second EA smell (i.e. *shared persistency*) is detected when multiple services access the same data collection or schema, thereby reducing team and service independence [30]. For this, our catalog proposes the **encapsulate business objects** refactoring solution (adapted from 'encapsulate variable' in [4]), as presented in table 4. This refactoring solution suggests to route accesses to the DBMS through dedicated data services; each of which encapsulates all business objects required by one application service. This structure reduces data visibility to only those of the owned business objects, thereby preventing unwanted interdependence between teams and services.

6. Discussion

In this section, we describe the extent to which the main RQs (see section 1) of this study have been answered through our finding and its demonstration; elaborate the implications of the EA refactoring solutions identified for researchs and practitioners; and identify the potential threats to the internal and external validity of our result.

6.1. Explanation of the Result

With regards to answering the RQ1 and its sub-questions, this study identifies the first catalog of EA refactoring solutions for all the EA smells proposed in [2]. We achieved this result by performing a conceptual transformation on relevant code refactoring solutions, which relies on a mapping between code and EA modelling concepts. To document the resulting EA refactoring solutions, we use a template of attributes which we adapted from some existing templates in code refactoring domain. Finally, we demonstrated the use of several refactoring solutions

in some small scenarios of EA smell. While the solution choices demonstrated may not be suitable to all cases in reality, we believe that the presented catalog provides a useful platform for EA researchers and practitioners to develop new or better refactoring solutions to common problems in EA.

6.2. Implication for Practitioners & Researchers

Recent study in EA has proposed the concept of EA debt [1] which represents the divergent EA evolution from the EA standards and principles. One possible source of EA debt is the implementation of sub-optimal solution design. The role of EA smells is to indicate the signs of existing sub-optimal solutions within the IT landscape, so that further investigation can be triggered in time [2].

With regards to answering the RQ2, the implication of our result can be seen from both practical and research perspectives. For EA practitioners, the proposed EA refactoring solutions may help to find possible methods or techniques for solving the EA smells identified. Also, such a solution catalog can help to establish common terminologies within an organization, thereby supporting various stakeholders to discuss possible solutions.

For the EA research community, our result gives motivation and food for thought for further investigations into the concept of EA refactoring. Future attempts to extend the catalog by transforming refactoring solutions from other domains (e.g. process smell, infrastructure smell, etc) may also adopt the conceptual transformation methodology used in this study. To allow public contributions in the development of EA refactoring solutions, the catalog has been published in a website together with a guideline on contributing [29].

6.3. Threats to Validity

The results of this study have to be seen in the light of some limitations. In this section, we present an assessment of possible threats to the internal and external validity of our result. Internal validity focuses on the reliability of the result within the given environment, whereas external validity focuses on the ability to generalize the result [32].

Internal Validity. The design of our methodology may pose certain threats to the internal validity of our result. Firstly, the code refactoring solutions analyzed were collected from a selection of relevant books on code smell and refactoring, thereby threatening the completeness of the EA refactoring solution catalog. Secondly, the mapping between UML and ArchiMate notations used in the transformation was summarized from some selected sources, thereby threatening the completeness of the mapping used in this study. Last but not least, the process of transforming code refactoring solutions into EA domain relies on subjective analysis, which potentially results in biased considerations upon deriving the EA refactoring solutions. To reduce the bias, the resulting EA refactoring solutions were reviewed and decided among the authors of this paper. Despite these weaknesses, we believe that the resulting EA refactoring solution catalog is a step in the right direction towards solutions for common design issues in EA.

External validity. As the resulting EA refactoring solutions are yet to be evaluated in industrial context, their descriptions may still rather hypothetical and theoretical, thereby posing challenges to their adoption. Furthermore, as there could be multiple solution alternatives for every EA smell, further research is still needed to identify possible factors and conditions which influence the suitability of a certain EA refactoring solution. Finally, the adoption of EA refactoring solutions greatly depends on the progress in EA smell research. At the time of writing, the existing EA smells are yet to be evaluated and their descriptions enriched with practical examples.

7. Conclusion & Future Work

Our main goal is to support EA practitioners in analyzing possible improvements with regards to the current circumstances and interests. We strongly argue that approaches to EA improvement must extend from EA modelling capabilities. Therefore, this study investigates the concept of refactoring in the context of EA modelling. Our methodology focuses on defining EA refactoring solutions for the EA smells proposed in [2] by analyzing the known associations between code refactoring solutions and code smells. The resulting contribution is a catalog of EA refactoring solutions which is publicly available on a web page [29].

In spite of this progress, there is still work to be done, especially in improving the catalog, or where further work is necessary. Some potential future works in this context are as follows: Firstly, further EA refactoring solutions can be derived for EA smells from different domains, such as the EA process smells [3]. For such purpose, we suggest to adapt the methodology from the one used in this study, e.g. by integrating a mapping of ArchiMate to another modelling language of interest as proposed in [16]. Secondly, empirical studies can be performed to further review and improve the hitherto knowledge in this area. Last but not least, tool supports can be developed for, e.g., recommending suitable EA refactoring solutions (e.g. [33]) or supporting the implementation thereof.

References

[1] S. Hacks, H. Höfert, J. Salentin, Y. C. Yeong, H. Lichter, Towards the definition of enterprise architecture debts, in: 23rd IEEE International Enterprise Distributed Object Computing Workshop, EDOC Workshops 2019, Paris, France, October 28-31, 2019, 2019, pp. 9–16. doi:10.1109/EDOCW.2019.00016.

[2] J. Salentin, S. Hacks, Towards a catalog of enterprise architecture smells, in: N. Gronau, M. Heine, H. Krasnova, K. Poustcchi (Eds.), Entwicklungen, Chancen und Herausforderungen der Digitalisierung: Proceedings der 15. Internationalen Tagung Wirtschaftsinformatik, WI 2020, Potsdam, Germany, March 9-11, 2020. Community Tracks, GITO Verlag, 2020, pp. 276–290. doi:10.30844/wi_2020_y1-salentin.

[3] B. Lehmann, P. Alexander, H. Lichter, S. Hacks, Towards the identification of process anti-patterns in enterprise architecture models, in: H. Lichter, S. Aydin, T. Sunetnanta, T. Anwar (Eds.), Proceedings of the 8th International Workshop on Quantitative Approaches to Software Quality, co-located with 27th Asia-Pacific Software Engineering Conference (APSEC 2020), Singapore (virtual), December 1, 2020., CEUR-WS.org, 2020, pp. 47–54. URL: http://ceur-ws.org/Vol-2767.

[4] M. Fowler, Refactoring - Improving the Design of Existing Code, Addison Wesley object technology series, Addison-Wesley, 1999. URL: http://martinfowler.com/books/refactoring.html.

[5] A. S. Nyamawe, H. Liu, Z. Niu, W. Wang, N. Niu, Recommending refactoring solutions based on traceability and code metrics, IEEE Access 6 (2018) 49460–49475. URL: https://doi.org/10.1109/ACCESS.2018.2868990. doi:10.1109/ACCESS.2018.2868990.

[6] W. G. Griswold, Program Restructuring as an Aid to Software Maintenance, Ph.D. thesis, Department of Computer Science and Engineering, University of Washington, USA, 1992. UMI Order No. GAX92-03258.

[7] S. J. Thompson, Refactoring functional programs, in: V. Vene, T. Uustalu (Eds.), Advanced Functional Programming, 5th International School, AFP 2004, Tartu, Estonia, August 14-21, 2004, Revised Lectures, volume 3622 of *Lecture Notes in Computer Science*, Springer, 2004, pp. 331–357. URL: https://doi.org/10.1007/11546382_9. doi:10.1007/11546382_9.

[8] A. Folli, T. Mens, Refactoring of UML models using AGG, Electron. Commun. Eur. Assoc. Softw. Sci. Technol. 8 (2007). URL: https://doi.org/10.14279/tuj.eceasst.8.112. doi:10.14279/tuj.eceasst.8.112.

[9] J. Reimann, C. Wilke, B. Demuth, M. Muck, U. Aßmann, Tool supported OCL refactoring catalogue, in: M. Balaban, J. Cabot, M. Gogolla, C. Wilke (Eds.), Proceedings of the 12th Workshop on OCL and Textual Modelling, Innsbruck, Austria, September 30, 2012, ACM, 2012, pp. 7–12. URL: https://doi.org/10.1145/2428516.2428518. doi:10.1145/2428516.2428518.

[10] M. Misbhauddin, M. Alshayeb, Uml model refactoring: A systematic literature review, Empirical Software Engineering 20 (2015) 206–251. URL: https://doi.org/10.1007/s10664-013-9283-7. doi:10.1007/s10664-013-9283-7.

[11] M. R. Hacene, S. Fennouh, R. Nkambou, P. Valtchev, Refactoring of ontologies: Improving the design of ontological models with concept analysis, in: 22nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2010, Arras, France, 27-29 October 2010 - Volume 2, IEEE Computer Society, 2010, pp. 167–172. URL: https://doi.org/10.1109/ICTAI.2010.97. doi:10.1109/ICTAI.2010.97.

[12] D. Silingas, E. Mileviciene, BPMN 2.0 Handbook, Future Strategies Inc., Lighthouse Point, FL, USA, 2012, pp. 125–134. URL: http://www.futstrat.com/.

[13] S. W. Ambler, P. J. Sadalage, Refactoring Databases: Evolutionary Database Design, Addison-Wesley Professional, 2006.

[14] J. Bruel, M. Mazzara, B. Meyer (Eds.), Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment - Second International Workshop, DEVOPS 2019, Château de Villebrumier, France, May 6-8, 2019, Revised Selected Papers, volume 12055 of *Lecture Notes in Computer Science*, Springer, 2020. URL: https://doi.org/10.1007/978-3-030-39306-9. doi:10.1007/978-3-030-39306-9.

[15] M. Lippert, S. Roock, Refactoring in large software projects: performing complex restructurings successfully, John Wiley & Sons, 2006.

[16] A. Q. Gill, Agile enterprise architecture modelling: Evaluating the applicability and integration of six modelling standards, Inf. Softw. Technol. 67 (2015) 196–206. URL: https://doi.org/10.1016/j.infsof.2015.07.002. doi:10.1016/j.infsof.2015.07.002.

[17] The Open Group, Archimate 3.1 specification, 2019. URL: https://pubs.opengroup.org/architecture/archimate3-doc/.

[18] M. J. Wiering, M. M. Bonsangue, R. van Buuren, L. Groenewegen, H. Jonkers, M. M. Lankhorst, Investigating the mapping of an enterprise description language into UML 2.0, Electronic Notes in Theoretical Computer Science 101 (2004) 155–179. URL: https://doi.org/10.1016/j.entcs.2004.02.020. doi:10.1016/j.entcs.2004.02.020.

[19] M. M. Lankhorst (Ed.), Enterprise Architecture

at Work - Modelling, Communication and Analysis, Fourth Edition, Springer, 2017. URL: https://doi.org/10.1007/978-3-662-53933-0. doi:10.1007/978-3-662-53933-0.

[20] T. Gericke, ArchiMate to UML Mapping, Technical Report, Adocus AB, Stockholm, Sweden, 2018. URL: http://www.adocus.com/media/21703/archimate-to-uml-mapping-whitepaper.pdf.

[21] K. Peffers, T. Tuunanen, M. A. Rothenberger, S. Chatterjee, A design science research methodology for information systems research, J. Manag. Inf. Syst. 24 (2008) 45–77. URL: http://www.jmis-web.org/articles/765.

[22] A. R. Hevner, S. T. March, J. Park, S. Ram, Design science in information systems research, MIS Q. 28 (2004) 75–105. URL: http://misq.org/design-science-in-information-systems-research.html.

[23] S. Kebir, I. Borne, D. Meslati, Automatic refactoring of component-based software by detecting and eliminating bad smells - A search-based approach, in: L. A. Maciaszek, J. Filipe (Eds.), ENASE 2016 - Proceedings of the 11th International Conference on Evaluation of Novel Approaches to Software Engineering, Rome, Italy 27-28 April, 2016, SciTePress, 2016, pp. 210–215. URL: https://doi.org/10.5220/0005891602100215. doi:10.5220/0005891602100215.

[24] W. H. Brown, R. C. Malveau, H. W. S. McCormick, T. J. Mowbray, AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis, 1st ed., John Wiley & Sons, Inc., USA, 1998.

[25] F. A. Fontana, V. Lenarduzzi, R. Roveda, D. Taibi, Are architectural smells independent from code smells? an empirical study, Journal of Systems and Software 154 (2019) 139–156. URL: https://doi.org/10.1016/j.jss.2019.04.066. doi:10.1016/j.jss.2019.04.066.

[26] G. Suryanarayana, G. Samarthyam, T. Sharma, Refactoring for Software Design Smells: Managing Technical Debt, 1st ed., Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2014.

[27] J. Kerievsky, Refactoring to patterns, in: C. Zannier, H. Erdogmus, L. Lindstrom (Eds.), Extreme Programming and Agile Methods - XP/Agile Universe 2004, 4th Conference on Extreme Programming and Agile Methods, Calgary, Canada, August 15-18, 2004, Proceedings, volume 3134 of *Lecture Notes in Computer Science*, Springer, 2004, p. 232. URL: https://doi.org/10.1007/978-3-540-27777-4_54. doi:10.1007/978-3-540-27777-4_54.

[28] S. Hacks, F. Timm, Towards a quality framework for enterprise architecture models, EMISA Forum 38 (2018) 31–32.

[29] L. Liss, H. Kämmerling, P. Alexander, H. Lichter, Enterprise architecture refactorings, 20.10.2021. URL: https://swc-public.pages.rwth-aachen.de/ea-refactoring-solutions/web-catalog/.

[30] J. Salentin, S. Hacks, Enterprise architecture smells, 2020. URL: https://ba-ea-smells.pages.rwth-aachen.de/ea-smells/.

[31] J. Král, M. Zemlicka, The most important service-oriented antipatterns, in: Proceedings of the Second International Conference on Software Engineering Advances (ICSEA 2007), August 25-31, 2007, Cap Esterel, French Riviera, France, IEEE Computer Society, 2007, p. 29. URL: https://doi.org/10.1109/ICSEA.2007.74. doi:10.1109/ICSEA.2007.74.

[32] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, Experimentation in Software Engineering - An Introduction, volume 6 of *The Kluwer International Series in Software Engineering*, Kluwer, 2000. URL: https://doi.org/10.1007/978-1-4615-4625-2. doi:10.1007/978-1-4615-4625-2.

[33] H. Zhang, S. Jarzabek, A bayesian network approach to rational architectural design, Int. J. Softw. Eng. Knowl. Eng. 15 (2005) 695–718. URL: https://doi.org/10.1142/S0218194005002488. doi:10.1142/S0218194005002488.

A Condition Coverage-Based Black Hole Inspired Meta-Heuristic for Test Data Generation

Derya Yeliz Ulutaş [1,2], Ayşe Tosun [2]

[1] *ASELSAN Inc., Ankara, Turkey*
[2] *Istanbul University, Faculty of Computer and Informatics Engineering, Istanbul, Turkey*

Abstract

Combinatorial Testing (CT) strategy is one of the well-known methods to achieve high code coverage rates during testing for safety critical systems. While generating test data in CT, we often encounter the problem of test case explosion, especially for the systems with multiple parameters and values. To overcome this challenge, search-based CT (CSST) strategies are introduced. In this study, we propose a new algorithm that inspires from a binary variant of Black Hole Algorithm (BBH) in CSST and adopt BBH according to the CT challenges in an industrial context. The proposed BBH version, *BH-AllStar*, aims the following: (1) obtaining higher condition coverage, (2) avoiding being stuck in local minima and (3) handling discrete input values. We finalize the solution space of *BH-AllStar* by reassessing the previously removed stars and incorporating the useful ones into it. We evaluate our approach on a real-life software project in the safety-critical domain with respect to condition coverage, number of test cases and execution time. Compared to BBH, *BH-AllStar* generates more test cases which achieve up to 43% increase in condition coverage.

Keywords

Combinatorial Search-based Software Testing (CSST), Test Data Generation, Meta-heuristic

1. Introduction

The complexity and the criticality of the safety critical systems in the defense industry are growing [1, 2], hence, these systems require more thorough and efficient testing technologies, such as test automation and regression test selection/prioritization [3] and continuous integration. The software testing field in the defense industry needs to develop more efficient and qualified techniques to meet security-critical requirements. [1]. For instance, various test techniques such as Boundry-Value Analysis and Equivalence Partitioning are used to meet the DO-178 Standard to ensure safety-critical requirements, which has been imposed by the U.S. Federal Aviation Administration (FAA). [4, 5]. To assess the effectiveness of these methods, different coverage criteria are employed [6]. During functional testing, Combinatorial Testing (CT) methods are widely utilized especially for software systems in which there exist many parameters and conditions that take a combination of multiple input values [7, 8]. CT method such as t-way testing executes the software using test cases with the interaction of the possible input values [9]. The radar software in our industrial context, for instance, takes 12 input parameters each takes two to six values. If we would like to test this software with full input coverage, we would end up executing 63.34.24.5=1,399,680 test cases. This is an impossible goal to reach, and instead in practice, we follow t-way testing with the IPOG strategy [10]: select the t value as high as possible with the cost of increasing the number of test cases. The current limitation of the employed t-way testing in our industrial setting is the lack of relationship between the selected test cases and their code coverage, because the strategy of selecting the test inputs is not dependent on the covered branches/conditions of the software system.

QuASoQ'21: 9th International Workshop on Quantitative Approaches to Software Quality, December 06, 2021, Taipei, Taiwan
EMAIL: dyulutas@aselsan.com.tr (D.Y. Ulutas); tosunay@itu.edu.tr (A. Tosun)

To address this challenge, combinatorial search-based software testing (CSST) approaches are suitable [11]. One of the studies by Al-Sammarraie and Jawawi [9] inspired our work. The authors propose three new approaches namely Binary Black Hole (BBH), Multiple Black Hole (MBH) and Binary Multiple Black Hole (BMBH) based on the Black Hole Algorithm (BHA), which is first introduced in [12] as an alternative to Particle Swarm Optimization (PSO). Their aim was generating test inputs with respect to the number of test cases and code coverage. The authors also aim to avoid being stuck in local minima and they propose a novel energy mechanism for MBH that helps to choose useful black hole populations only. The results show that MBH outperforms PSO and BHA in terms of the number of test cases, consistency and high coverage rates.

We share the same goal with [9] of generating test data using a search-based meta-heuristic, which avoids local minima and achieves higher coverage than BBH. The approach in [9] could be well fitted in our industrial problem because of two main motivations: (1) their problem domain and data set are similar to our project, i.e., radar software, (2) they innovatively apply BHA, which is originally a clustering algorithm, to generate test data for functional testing. However, considering the scale of our radar software under test, BBH approach does not fully address our challenge. More specifically, BBH concentrates on generating minimum number of test cases rather than increasing the coverage criteria. Due to this, it lacks assessing the generated stars except the black hole in terms of their coverage. The term star here corresponds to the test case (e.g., [3, 5, 7] is an input set, also a test case and named as 'star'), as depicted in Table 2. This in turn eliminates some of the useful stars during the operations of BBH. In this study, we propose a new algorithm called BH-AllStar by modifying several strategies in BBH, such as elimination mechanism based on distance between stars, moving stars in discrete space, and selecting the best black hole population from the history. Our new approach generates relatively more test cases with higher condition coverage than BBH, although we cover only 0.14% of 1,399,680 test cases in our industrial context. We assess the applicability of BBH in [9] and its new variant BH-AllStar in an industrial context with respect to the number of test cases, coverage rate and execution time.

The remainder of this paper is organized as follows: Section 2 presents the related work. We report the details of our approach BH-AllStar and research methodology in Section 3. Section 4 shows the results and discussion; Section 5 explains the threats to validity. Finally, Section 6 concludes this study with future directions.

2. Related work

Combinatorial Search-based Software Testing (CSST) is a testing method based on Combinatorial Testing (CT). CT aims at generating input vectors combining possible input values of all input parameters of the Software Under Test (SUT) [10]. The SUT can have n input parameters: c_i (i = 1,2,3…n). Every parameter can take a set of values V_i where i can be from 1 to n. For example, V_1 is the value set of c_1 and contains m different values. (v_1,v_2,…,v_m). One test case is composed of a set of values from all V_i of all c_i (e.g. $v_x \in V_1$, $v_y \in V_2$, $v_z \in V_3$) [10]. There are different methods to build all combinations of input parameters for CT such as t-way testing or randomization-based methods in the literature [13]. CSST is one of the search-based approaches to reduce the large solution space [9]. To find the optimum solution of a problem, especially for the problems, which contain very large sets of solutions and have computational constraints, meta-heuristic search techniques are commonly implemented [14]. These meta-heuristic search techniques are based on initially generated random solution space and narrowing down the search space based on the evaluation criteria.

A recent systematic mapping study in [14] highlights 260 relevant studies in CSST, and narrows down to 42 primary studies to investigate the proposed approaches and their test case generation accuracy. The authors summarize well-known meta-heuristic search techniques as follows: Genetic Algorithm (GA), Particle Swarm Optimization Algorithm (PSO), Simulated Annealing Algorithm (SA), and Ant Colony Optimization (ACO). Their mapping strategies show that GA is the most popularly applied technique, whereas some combine multiple algorithms such as GA and SA. They conclude that the primary studies provide benefits in terms of code coverage and execution time.

In Table 1, we list a sample of studies obtained from [14] that we examine in terms of the approach, dataset on which the approaches are evaluated, and the fitness criteria. Among the

studies, we selected eight studies targeting a similar objective to our industry problem, and varying in terms of fitness functions and datasets. The studies [15-19] utilize nature inspired optimization methods for CSST, whereas [9, 12, 20] use a heuristic based on black hole phenomenon. All studies in Table 1 show that the improved versions of nature inspired optimization methods outperform original methods with respect to code coverage or execution time. All the prior studies validate the success of their proposed approach on simple functions like the source code that calculates the greatest common divisor or checking the validity of the date, etc. Unfortunately these projects are not representative of real programs, and achieving high coverage is relatively easier with few test cases. Except one study [15], all studies focus on branch coverage as one of their evaluation criteria, although in practice wıth more complex algorithms, multiple conditions need to be taken into account with a condition converage criterion. Below, we give more description on our baseline algorithm BHA [12] and its application on test case generation [9].

Table 1
Summary of studies in the literature

Study	Baseline Method	Dataset/Project	Fitness/Evaluation Criteria
[15]	ACO	triangleType, gcd, calday, isValidDate, cal	Branch coverage
[16]	SA	unknown	Condition coverage
[19]	PSO	triangleType, gcd, calday, isValidDate, cal	Branch coverage
[17]	GA	triangle classification and nextDate	Branch distance
[18]	SA	triangle classification	Branch Coverage, # of Detected Mutants/Defects
[12]	BHA	Six-benchmark dataset from ML databases	Distance between blackhole and the star
[20]	BHA	Well-known mathematical functions	
[9]	BHA	pizza ordering, smart mobile, heart disease	k number of test cases

2.1. Black hole algorithm

BHA has been first introduced in the study [12] as a new heuristic algorithm inspired by the black hole phenomenon. Based on Newton's law, scientists invented a concept about the stars with high power and strong gravitational field [21]. They name this star as "black hole". The reason of this name is that any object moving around this special star cannot escape and is absorbed into the black hole if the objects cross the Schwarzschild radius [21]. This special star is described with the adjective 'black' because light cannot be reflected and make this star invisible [21, 22]. With the inspiration of the nature of black holes, BHA is proposed for data clustering [12]. Fig.1 presents the pseudocode of BHA adopted from [9, 12].

```
Algorithm 1: Black Hole Algorithm
  Result: Solution space consists of stars
  P = initializePopulation();
  while iteration do
    P = moveStars(P);
    BH = updateFitness(P); //select the best star as black hole;
    for each star S in P do
      if star(i crosses the R)) then
        replace(P, star(i)); //destroy current star and generate new one;
      end
    end
    BH = updateFitness(P); //select the best star as black hole;
  end
```

Figure 1: BHA pseudocode adopted from [9, 12]

According to the terminology used in [9], a population with random stars is initially generated, and a fitness value is calculated for every star in this population. The best star with the best fitness value is selected as the black hole. Later, all stars in the population are moved towards the black hole as described in Eq.1. The new location of the current star x(t+1) is calculated by multiplying a random number with the difference between the BH Star and the current star (BH-currentStar), and adding the result to the previous location x(t). The current star is moved closer to the BH Star as a result of this operation. Movement of the stars towards the black hole provides a grouping process and force all the other stars to converge to the best fitness value. If a star crosses the event horizon (R) of the black hole defined in Eq.2, it is destroyed, and a new random star is generated and added into the population. The R value is calculated as the ratio of the fitness value of BH Star and sum of the fitness function outcomes of all stars in the population. This step leads to adding various stars to the population and increases the probability of convergence to the desired solution.

$$x(t+1) = x(t) + \text{rand x (BH-currentStar)} \quad (1)$$

$$R = f(BHStar) / \sum_{i=1}^{n} f(Star(i)) \quad (2)$$

The searching process stops if the stopping criterion is met. This criterion can be the maximum number of iterations or a sufficiently good fitness criterion [9, 12].

Hatamlou conducted an experiment on the six-benchmark dataset [12] and highlights two

advantages of BHA: (1) simple and easy to implement, (2) applicable to other problem domains, and (3) outperforming other clustering methods. The performance of BHA is later tested in [21] on mathematical functions to reach their minimum and maximum values. The authors in [21] compare BHA with GA and PSO, and report that BHA outperforms others and can be applied to other problem domains.

2.2. The baseline study

Based on the BHA proposed in [12], Al-Sammarraie and Jawawi [9] propose three variations of BHA for test case generation: BBH, MBH and BMBH. Terms and function names used in the study [9] are described in Table 2. For the inputs that has discrete values like binary values (0,1), *moveStars* operation (Eq. 1) does not work because it causes a shift in the continuous domain due to the multiplication of the distance with a random value. To handle this issue, a binary variant of black hole is proposed. This approach generates a random value r_d between 0 and 1. This random value is compared against the constant value, p_r (Eq. 3). If it is greater than r_d, the new location of the star is going to be equal to that of black hole. Otherwise, new location of is equal to its own old value [9].

Table 2
Terminology of BBH used in [9]

Term	Meaning
Parameter	x: can take values x_1, x_2, x_3 y: can take values $y_1, y_2, y_3, \ldots$
Star	$[x_1\ y_2\ z_3\ t_4]$
Population	$[star_1, star_2 \ldots star_n]$
updateFitness()	Calculate the coverage of every star in the population and determine the best star as the black hole.
moveStars()	Change location of stars towards the blackhole.
updateRadius()	Update the radius value of black hole.
replace()	Remove the current star and generate a new one.
R	Radius of the black hole (eq.2)

$$xi(t+1) = \begin{cases} BH(t)\ if\ r_d < pr \\ xi(t)\ otherwise \end{cases} \quad (3)$$

The authors reach consistent and high coverage by extending solution space with the help of multiple swarm approach called MBH [9]. The main idea in MBH is to generate more than one population and in turn, more than one black hole in a single iteration to avoid being stuck in local minima. At the end of the iterations, final list of populations become the solution space. The findings showed that their approach suffers from being stuck in local minima since there is no mutation operation. The authors point that the population initialization and *moveStars* operations could be improved.

Our work complements the previous study [9] in several ways: (1)We propose a condition coverage-based elimination mechanism instead of a distance-based one to keep the test cases that likely contribute to total coverage. (2) We prioritize achieving higher coverage rates than restricting the number of test cases. (3) We propose two new approaches within BHA to assess all the generated stars and black holes and to extend the selected population without being stuck in local minima. (4) We apply the BHA method to a real-life software system for test case generation.

3. Our methodology

The software under test in this study is part of a large-scale radar software developed for the defense industry. Due to its confidentiality constraints, we are not able to give the details of the algorithm or its pseudocode. We can briefly describe the complexity of our SUT: it takes 12 input parameters with different possible values, and outputs an integer value. It contains two functions with 17 if and nested if statements with up to three levels, two for loops, two switch case statements and 72 branches in total. We built a technological setup, in order to conduct our analysis on BBH and its variants including BH-AllStar. Our analysis starts by selecting the baseline algorithm as shown in Fig 2, namely BBH, to generate test cases, i.e., a black hole (BH) surrounded with stars, and our software is executed against this test case. Then the system collects the information of missed conditions using Jacoco testing tool. Iteratively, the condition coverage information is stored in a file, and used in the elimination decision mechanism. Based on the decisions, the algorithm generates new test cases (stars) and the cycle continues. The algorithm terminates after K (10-500) number of iterations.

We modify several operations in [9] and propose our approach: BH-AllStar. We use *initializePopulation()* and *updateFitness()* methods as is, but modify *moveStars()*, and created a new star elimination criterion. We also propose two new methods to select the final population both of which will be described below. Our motivation is also determining the best BH in

reaching as high coverage rates as possible, and shaping the other stars around it without being stuck in local minima. During our analysis, we noticed that the distance criteria defined in [9] removes some stars although they contribute to the coverage ratio. It also prioritizes to keep the number of test cases as few as possible. Increasing the number of test cases is preferable for our industrial setting than having low coverage rates. Fig. 3 shows the pseudocode of our approach. Our contributions are presented in the below subsections.

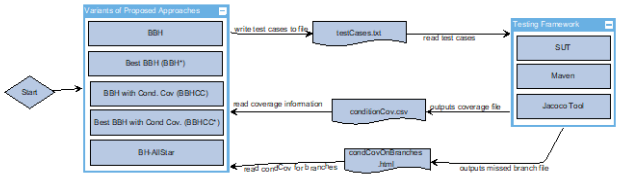

Figure 2: Our experiment setup

```
Algorithm 1: BH-AllStar
  Result: Solution space consists of stars
  P = initializePopulation();
  BH = updateFitness(P);
  while iteration do
      P = moveStars(P);
      BH = updateFitness(P);
      for each alive star S in P do
          if not isBestAtLeastOnOneBranch() then
              ratio = calculateBranchRatio();
              if ratio is less than 1 then
                  S.isAlive = false;
                  S = generateNewStar(); //generate new star different from black hole
                    P.add(S);
              end
          end
      end
      BH = updateFitness(P);
      calculateTotalCoverage();
      bestPopulation = keepPopulationWithMaxTotalCoverege();
  end
  solutionSpace = allStar(P, bestPopulation);
```

Figure 3: Pseudocode of our proposed approach

3.1. Condition coverage-based elimination mechanism

We calculate condition coverage for each branch existing in our testing code. For example, an "if" statement with one condition, e.g. x<5, contains two branches, true or false, and the condition coverage for this branch can be 0, 50% or 100%. If an "if" statement has two conditions, e.g. x<5 && y>10, then it has 4 conditions (TT, TF, FT, FF). With the input set (x=3, y=11; x=3, y=9; x=6, y=9), the condition coverage for this branch is calculated as 75%. Only traditional branch coverage is not enough for our study since we need to cover all the conditions of each branch in our testing code. Therefore, we replace the distance based mechanism, which calculates to what extent the distance between current star and black hole exceeds the R value to decide whether to destroy the star or keep it alive, with our condition coverage based mechanism, as shown in Fig.3 described as *isBestAtLeastOnOneBranch*() and *calculateBranchRatio*() methods. With this condition coverage-based mechanism for all branches of our SUT, we give chance to the stars with lower coverage ratios then the BH Star to cover uncovered branches. We have two steps in our decision mechanism:

isBestAtLeastOnOneBranch We control if the current star is the best among all stars in the population in terms of covering at least one more branch. This operation would keep an efficient star, even if it has a smaller coverage ratio compared to other stars.

calculateBranchRatio We compare current star and black hole in terms of covering all branches one by one and calculate the ratio of the number of branches that current star covers better than black hole, and the number of branches that current star covers worse than black hole. We use here a coefficient=3 to multiply the number of better covered branches in order to give a star more chance (see Eq. 4). We have made several tests with different coefficients and chosen the value 3 as the optimum value for deciding the stars to be accepted or not.

$$\frac{\#ofBetterCoveredBranches*3}{\#ofWorseCoveredBranches} \quad (4)$$

Fig. 4 illustrates a sample of stars: An eliminated star (purple), a previously eliminated but later revived star (yellow), and an always alive star (orange) with their coverage values in the timeline, when our proposed mechanism is applied. The total coverage significantly increases at the end of the iterations as we let a previously killed star (yellow) to be included into the final population. Eliminating a star (purple) also increase total coverage during 18th iteration.

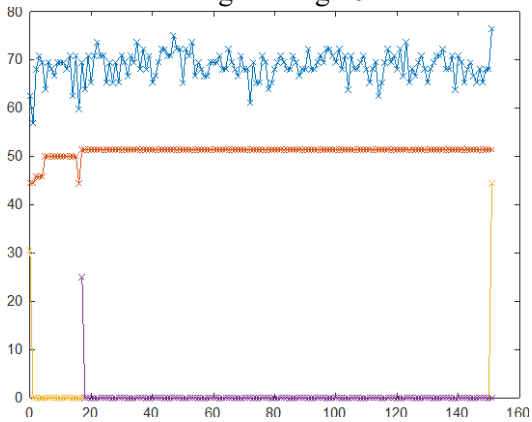

Figure 4: The x-axis is the number of iterations and the y-axis is condition coverage rate.

3.2. Moving Discretely

In *moveStars()* operation, to shift the stars towards the black hole, we propose an index-based method for discrete-valued parameters, similar to 'Binary Variants' in [9]. We store our input parameters in an array and we generate a random index value between the input value index of the current star and the BH. Then we move current star to this index and assign the value to the current star as the updated input parameter.

3.3. Choosing the best population over history

In *keepPopulationWithMaxTotalCoverage*(), we keep the population generated in all iterations of BBH including coverage rates and alive/dead states of each star. At the end of the iterations, we select the population that has the highest coverage ratio, not the last population according to BH.

3.4. All Star operation

In *allStar*() method, we compare all the stars between t_0 and tend against the BH at t_n, and add the best stars in terms of condition coverage into our final population. Although this causes an increase in the number of test cases, it also increases the coverage of the final population.

The variants of BBH are as follows: (1) Best BBH (**BBH***): We incorporate "Choosing Best Population from History" approach into BBH. (2) BBH with Condition Coverage (**BBHCC**): We use "Condition Coverage-Based Elimination Mechanism". (3) Best BBH with Condition Coverage (**BBHCC***): We add "Choosing Best Population over History" onto BBHCC. (4) **BH-AllStar**: We add "AllStar Operation" onto BBHCC*. We assess all according to condition coverage of the population, number of test cases, and execution time.

4. Results and discussion

The results over different iterations and with variants of BBH, reported in Table 3, show that higher code coverage rates are obtained compared to BBH. For instance, with 500 iterations, coverage values are 69.4% in BBH, whereas 75% in BBH*, 70.8% in BBHCC, 73.6% in BBHCC* and 76.8% in BH-AllStar. We observe that same coverage rates can be achieved with fewer test cases but higher execution time for different number of iterations. The maximum code coverage rate is approximately 76% in iterations 150 and 500 with BH-AllStar. Solution space consists of 190 test cases for 150 iterations and executes in 327 secs. However, for 500 iterations, only 19 test cases are used but execution time is 815 secs.

In addition, the baseline algorithm, BBH has a better coverage (72.2%) than our condition coverage based BBH approach (65.3%) for 300 iterations. However, our additions help to increase the coverage from 65.3% to 75% in BH-AllStar. BBH* always reaches better coverage rates than BBH. When the coverage criterion is set as condition (BBHCC), we do not always see a better convergence than BBH. But choosing the best population and adding all the "good" stars eventually converge to much higher coverage ratios. Instead of defining multiple black hole as in [9], we stick to the single black hole phenomenon but we give a certain flexibility to the star selection and achieve a similar improvement like in [9]. We can confirm the prior finding that a black hole inspired meta-heuristic could be successful at generating test data for software systems with too many parameters.

Table 3
Performance of BBH Variants and BH-Allstar

Iteration		BBH	BBH*	BBH CC	BBH CC*	BH-AllStar
10	**Stars (#)**	10	10	10	10	12
	Cov. (%)	51.4	70.8	68.1	72.2	73.6
	Time	19	19	22	22	22
150	**Stars (#)**	10	10	10	10	190
	Cov. (%)	68.1	73.5	68.7	75	76.8
	Time	320	320	372	372	372
300	**Stars (#)**	10	10	10	10	14
	Cov. (%)	72.2	75.0	65.3	73.6	75
	Time	460	460	476	476	476
500	**Stars (#)**	10	10	10	10	19
	Cov. (%)	69.4	75	70.8	73.6	76.4
	Time	782	782	815	815	815

Table 4
Coverage for different number of iterations

		BH	Alive Stars			BH-AllStar
Coverage	#iter	Star	Min	Max	Median	
	10	51.4	25	51.4	37.5	73.6
	150	51.4	23.6	51.4	43	76.8
	300	51.4	23.6	51.4	37.5	75
	500	51.4	25	51.4	38.8	76.4

Considering the number of test cases, we observe that BH-AllStar may take higher values

with respect to the positions of the stars, condition coverage ratios of each star and the black hole star. In 150 iterations, we generate 190 test cases, which is a higher value compared to the other experiment results. Since we pick the best stars in terms of coverage during *All-Star* operation, there may be many "best" stars in the population compared to the selected black hole. We did not perform any filtering on these best stars like pruning, but it might be possible to reduce those that cover similar conditions as a final step.

Table 4 presents the coverage rate of the BH Star, Alive Stars and BH-AllStar in the final solution space. In both BBH and our proposed approach, the black hole has the same coverage rate (51.4%). Alive stars in the solution space have a minimum of 23.6%, a maximum of %51.4 (BH itself) and a median of 43% coverage ratios. A star with less coverage ratio than the BH Star is acceptable in our study because it might have a higher coverage on any branch compared to the BH star.

Fig. 5 presents the coverage rates of all the branches according to 190 stars in the solution space for 150 iterations. As depicted in Fig.5, there are still some branches (e.g., 19th to 27th) which are not sufficiently covered although we increase the total coverage ratio. This is partly due to the fact that BH algorithm essentially revolves around the black star that is picked. So, some of the areas in the search space might be missed during the iterations.

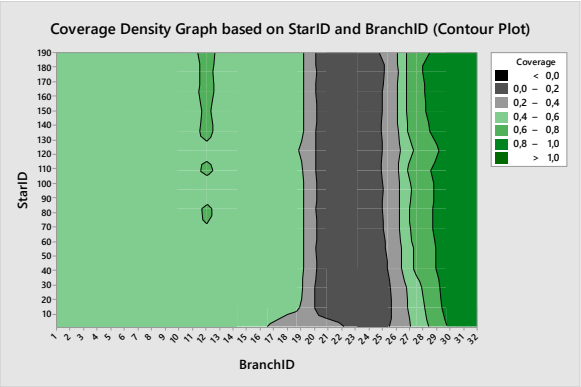

Figure 5 Coverage Density Graph based on Star ID and Branch ID

Fig. 6 illustrates the change in the total coverage ratio of BBH and BH-AllStar for 150 iterations. It is seen that one or more populations were found with BBH (around 60th and 100th iterations) that reach a higher coverage, but there were later lost. The final coverage ratio was also smaller at the end. On the other hand, in our approach, a peak can be seen just after all the iterations are executed, at t=151. The population with the best coverage has been kept and the good stars over the history are added to that population.

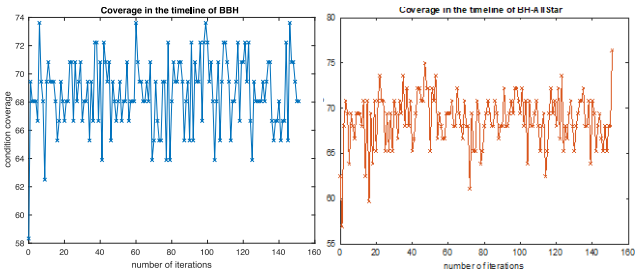

Figure 6 Coverage in the timeline of BBH

The results show that our Choosing Best Population from History approach always provides better coverage rates. Since the nature of the movement to the black hole and random generation of stars might cause a decrease in total coverage in BBH, we handle this problem by keeping the population with maximum coverage. The main reason that BH-AllStar outperforms BBH is that the latter eliminates stars because it is too close to the black hole star. However, it does not mean that these stars do not contribute to total coverage.

5. Threats to validity

We highlight several issues that might jeopardize the validity of our findings, and discuss these in this section. The first issue is related to the construct defined as the fitness criteria of our proposed algorithm. We aim to improve the final population of the algorithm by assessing the condition coverage ratio, which is calculated through Jacoco tool for each branch of the tested algorithm. Jacoco gives a condition coverage ratio for each branch, but it does not specifically say which conditions are satisfied in that branch. When two stars having 50% condition coverage for the same branch are evaluated, we consider those the same. But we do not know whether both cover different conditions and complement each other. This might have caused to falsely eliminate some stars from the population. We plan to work on this issue in our future studies by checking other unit testing tools.

The second issue is related to the binary strategy applied to turn BHA to BBH algorithm. In Section III, we describe how the movement of the stars happens on a discrete scale.

Unfortunately, the description in [9] was not detailed enough and we had to make some assumptions. This might cause a deviation between BBH in [9] and BBH we developed here. It does not affect our comparisons between BBH, its variants and BH-AllStar on the software under test in this paper.

The third issue is related to the internal validity of the experimental design. We are aware that choosing a different coefficient value in Eq.4 in BH-AllStar would impact our results, in fact, we tried multiple values and observed the convergence of the algorithm in terms of condition coverage to pick the final value as 3. We also think the randomness in the initial population, and the selection of best stars from all the iterations could affect the final results. We plan to work on these as future works as both require empirical analysis of different strategies. Finally, our results are limited to a single industrial context because we chose to solve the problem of our industrial partner through an improved CSST method. The prior works often assess their approaches on simple programming exercises, which suffer from generalization of the results on real life projects. We believe our study validates the real application of Black Hole inspired approaches adopted to condition coverage criterion.

6. Conclusion

In this study, we present a new approach to generate test data by implementing a previously proposed meta-heuristic searching technique BBH on a real-life engineering problem that we face for our safety critical systems in the industry. Our BBH variants and new algorithm BH-AllStar perform better than BBH in terms of condition coverage up to 43%. Although BH-AllStar generates more test cases compared to BBH, we achieve extending search space and obtaining higher condition coverage rates in the same execution time. As a future work, we plan new experiments to compare Multiple Black Hole Approach reported in [9] and our BH-AllStar. We also plan to decrease the number of test cases in our current approach, the selection of best stars for each branch in BH-AllStar can be limited by selection only one among the best stars. We also intend to conduct further research on every condition in a decision to be covered of the SUT, rather than having higher condition coverage, which is known as MC/DC (Modified Conditon/Decision Coverage). We think that our proposed approach can be applied to other problems, such as testing algorithms with different level of complexity, nested conditions and line of codes. Initialization of the population, which is the first step of BBH can also be optimized to start with a better population rather than a random start as in its current setting.

7. References

[1] J. Park, H. Ryu, H.-J. Choi, and D.-K. Ryu, *A survey on software test maturity in Korean defense industry*. 2008, pp. 149-150.

[2] R. Kenett, F. Ruggeri, and F. Faltin, "Analytic Methods in Systems and Software Testing," 07/01 2018, doi: 10.1002/9781119357056.

[3] V. Garousi, R. Ozkan, and A. Betin-Can, "Multi-objective regression test selection in practice: An empirical study in the defense software industry," *Information and Software Technology,* 06/01 2018, doi: 10.1016/j.infsof.2018.06.007.

[4] S. Nidhra, "Black Box and White Box Testing Techniques - A Literature Review," *International Journal of Embedded Systems and Applications,* vol. 2, pp. 29-50, 06/30 2012, doi: 10.5121/ijesa.2012.2204.

[5] W. Youn, S. Hong, K. Oh, and O. Ahn, "Software Certification of Safety-Critical Avionic Systems: DO-178C and Its Impacts," *Aerospace and Electronic Systems Magazine, IEEE,* vol. 30, pp. 4-13, 04/01 2015, doi: 10.1109/MAES.2014.140109.

[6] M.-H. Chen, M. Lyu, and W. Wong, "Effect of code coverage on software reliability measurement," *Reliability, IEEE Transactions on,* vol. 50, pp. 165-170, 07/01 2001, doi: 10.1109/24.963124.

[7] L. Hu, W. Wong, D. Kuhn, and R. Kacker, "How does combinatorial testing perform in the real world: an empirical study," *Empirical Software Engineering,* vol. 25, 07/01 2020, doi: 10.1007/s10664-019-09799-2.

[8] D. R. Kuhn, R. N. Kacker, and Y. Lei, *Introduction to Combinatorial Testing* (Chapman & Hall/CRC Innovations in Software Engineering and Software

Development Series). Taylor & Francis, 2013.

[9] H. Al-Sammarraie and D. Jawawi, "Multiple Black Hole Inspired Meta-Heuristic Searching Optimization for Combinatorial Testing," *IEEE Access,* vol. PP, pp. 1-1, 02/13 2020, doi: 10.1109/ACCESS.2020.2973696.

[10] y. Lei, R. Kacker, D. Kuhn, V. Okun, and J. Lawrence, *IPOG: A general strategy for T-way software testing*. 2007, pp. 549-556.

[11] C. Nie and H. Leung, "A Survey of Combinatorial Testing," *ACM Comput. Surv.,* vol. 43, p. 11, 01/01 2011, doi: 10.1145/1883612.1883618.

[12] A. Hatamlou, "Black hole: A new heuristic optimization approach for data clustering," *Information Sciences,* vol. 222, pp. 175–184, 02/01 2013, doi: 10.1016/j.ins.2012.08.023.

[13] K. Zamli, "T-Way Strategies and Its Applications for Combinatorial Testing," *International Journal of New Computer Architectures and their Applications (IJNCAA),* vol. 2, pp. 459-473, 01/01 2011.

[14] D. Y. Ulutas, "The Generation of Optimized Test Data: Preliminary Analysis of a Systematic Mapping Study," 2020.

[15] C. Mao, X. Yu, J. Chen, and J. Chen, *Generating Test Data for Structural Testing Based on Ant Colony Optimization*. 2012, pp. 98-101.

[16] N. Tracey, J. Clark, K. Mander, and J. McDermid, "An automated framework for structural test-data generation," in *Proceedings 13th IEEE International Conference on Automated Software Engineering (Cat. No.98EX239)*, 13-16 Oct. 1998 1998, pp. 285-288, doi: 10.1109/ASE.1998.732680.

[17] A. El-Serafy, G. El Sayed, C. Salama, and A. Wahba, *Enhanced Genetic Algorithm for MC/DC test data generation*. 2015, pp. 1-8.

[18] K. Wang, Y. Wang, and L. Zhang, *Software testing method based on improved simulated annealing algorithm*. 2014, pp. 418-421.

[19] C. Mao, X. Yu, and J. Chen, *Swarm Intelligence-Based Test Data Generation for Structural Testing*. 2012.

[20] M. Farahmandian and A. Hatamlou, "Solving optimization problem using black hole algorithm," *Journal of Advanced Computer Science & Technology,* vol. 4, 12/25 2015, doi: 10.14419/jacst.v4i1.4094.

[21] E. Curiel, "The many definitions of a black hole," *Nature Astronomy,* vol. 3, 01/08 2019, doi: 10.1038/s41550-018-0602-1.

[22] E.-G. Talbi, *Metaheuristics: From Design to Implementation*. 2009.