Theory and Applications of Mathematics & Computer Science

(ISSN 2067-2764, EISSN 2247-6202)
http://www.uav.ro/applications/se/journal/index.php/tamcs

Theory and Applications of Mathematics & Computer Science 8 (1) (2018) 24 – 38

Domain Complexity in Corrective Maintenance Tasks' Complexity: An Empirical Study in a Micro Software Company

Zeljko Stojanov[a,*], Jelena Stojanov[a], Dalibor Dobrilovic[a]

[a]*University of Novi Sad, Technical faculty "Mihajlo Pupin", Zrenjanin, Serbia.*

Abstract

Corrective maintenance is very important in software engineering practice since it enables correction of problems identified in operational use of software applications. Therefore, modeling complexity of maintenance tasks is essential for estimation and planning activities in software organizations that spend majority of resources on maintenance tasks. The article presents a study aimed at developing a model for maintenance task complexity by considering specific parameters of domain complexity associated to each software application. The study was conducted in a micro software company. The model enables analysis of trends for maintenance task complexity and correlation between task complexity and time spent for completing tasks. Implication and benefits of the presented research for the selected software company, for managers in software industry and researchers are discussed. The article concludes with challenging research directions.

Keywords: Task complexity, Domain complexity, Mathematical model, Corrective maintenance.
2010 MSC: 68N30 Mathematical aspects of software engineering (specification, verification, metrics, requirements, etc.), 68Q25 Analysis of algorithms and problem complexity.

1. Introduction

Software maintenance relates to post delivery activities aimed at ensuring efficient use of software systems without significant changes in software design. Software maintenance includes planned activities, such as bug fixing or enhancing functionality, and unplanned activities, such as adapting a system to new business conditions (Tripathy & Naik, 2015). Maintenance activities involve tight cooperation of software engineers engaged in maintaining software systems and clients that use software systems, both of them with different views of software maintenance, which strongly emphasizes managerial issues as the biggest problem in software maintenance

*Corresponding author
Email addresses: zeljko.stojanov@uns.ac.rs (Zeljko Stojanov), jelena.stojanov@uns.ac.rs (Jelena Stojanov), dalibor.dobrilovic@uns.ac.rs (Dalibor Dobrilovic)

(April & Abran, 2008). Recognized technical and organizational complexity of maintenance activities resulted with high costs, which are usually between 50% and 90% of total costs in software life cycle (Grubb & Takang, 2003; Junio *et al.*, 2011; Pino *et al.*, 2012; Bourque & Fairley, 2014). Despite recognized complexity and high costs of software maintenance activities, it is much less investigated compared to software development (Banker & Slaughter, 1997; Jones, 2010). In addition, software maintenance activities are mainly short term tasks, in many cases completed on day-to-day basis in order to keep software operational (Tan & Mookerjee, 2005; Junio *et al.*, 2011). Software maintenance activities are more difficult and complex comparing to software development activities (Jones, 2010).

Software maintenance tasks are performed in order to sustain software systems useful and operable for users. Maintenance tasks are performed on already developed software, and in many cases require not only technical skills but also organizational skills related to estimation of costs and risks, and evaluation of necessary tasks to be performed. The complexity of software maintenance is reflected in existence of 23 types of work that might be performed on software systems, which are systematized by (Jones, 2010) in a book Software Engineering Best Practices. In many cases, intensive maintenance of software systems results with degradation of their structure and characteristics, making them more difficult to maintain (April & Abran, 2008).

Software maintenance tasks have been researched for over 40 years, which resulted with several classifications and typologies of maintenance types. The first and the most influential typology of maintenance types was proposed by (Swanson, 1976), in which corrective, adaptive and perfective maintenance can be distinguished. This typology was later used and interpreted in variety of ways by many researchers, resulting with several typologies and definitions of maintenance types. (Chapin *et al.*, 2001) proposed a refined classification of software evolution and maintenance types aligned to clusters related to software systems, suggesting that different software organization can use different types of software maintenance. In this refined typology with 12 types of software maintenance activities, corrective maintenance occurs in the cluster related to business rules. In standard ISO 14764:2006 Software Engineering - Software Life Cycle Processes - Maintenance(ISO, 2006), there are four maintenance categories: corrective, adaptive, perfective and preventive. Corrective maintenance is in ISO 14764:2006 defined as "*the reactive modification of a software product performed after delivery to correct discovered problems*". According to (April & Abran, 2008), corrective maintenance is reactive since it is performed after a problem or a failure is identified in a software, requiring work to solve the problem and bring the software into a usable state. In *Guide to the Software Engineering Body of Knowledge (SWEBOK)* (Bourque & Fairley, 2014) is stated that emergency maintenance is a special type of corrective maintenance aimed at mitigating identified problems without scheduling maintenance request in a classical maintenance process defined in software organizations. (Tripathy & Naik, 2015) distinguished intention-based classification of software maintenance activities which is aligned with ISO 14764:2006 (ISO, 2006), but also proposed activity-based classification with corrections and enhancements as the main types of activities. In addition, maintenance types usually overlap in industrial practice, and it is common case that adaptive and perfective work hide corrective work in software maintenance (Hatton, 2007). In any of proposed classification of maintenance types, corrective maintenance attracted significant attention since it intends to fix discovered problems and bring software to operational state for end users, and usually has priority over other types of

work (April & Abran, 2009)

Task is a central concept in studying human behavior in various situations and includes a set of activities performed by humans in a given context in order to achieve proposed goals. Studying tasks is necessary part of organizational research lading towards improvement of practice. The main directions of researching task complexity relates to complexity of information processing, decision making and goal setting (Wood, 1986; Campbell, 1988), in which the complexity has been treated as a psychological experience, as an interaction between task and person characteristics, and as a function of tack properties. The most commonly used definition of task complexity defines a task as set of products, required acts and information ques (Wood, 1986). In organizational research of human performed tasks, objective task complexity depends only on task intrinsic characteristics, while subjective task complexity depends on task solver's experience and knowledge (Maynard & Hakel, 1997; Braarud, 2001; Parkes, 2017). Since software engineering has been considered as a complex discipline that includes technical, organizational and human factors, investigation of human performed tasks is essential for understanding and improving everyday practice (Dybå *et al.*, 2011; Capretz, 2014).

Complexity is very important issue to consider in software evolution and maintenance since it influences quality of software products and all activities in software life cycle (Keshavarz *et al.*, 2011). Complexity assessment assumes the use of carefully selected metrics for measuring trends and relations in historical data about software evolution and maintenance (Suh & Neamtiu, 2010). (Sun *et al.*, 2015) stated that selecting relevant information from maintenance repositories is essential for improving maintenance tasks. Complexity has been in software engineering mostly associated to structural complexity of software systems (Lilienthal, 2009; Lu *et al.*, 2016), but (Li & Delugach, 1997) suggested that traditional software complexity metrics cannot be effectively implemented for measuring complexity of application domains. In addition, (Shaft & Vessey, 1998) indicated the importance of domain knowledge (knowledge of the problem area) on program comprehension activities, which are essential in software maintenance practice. However, due to the evolving nature of domain knowledge (evolving nature of business), (Mendes-Moreira & Davies, 1993) suggested regular update of domain knowledge for efficient maintenance of software systems.

Based on the stated observations and the authors experience in researching software maintenance processes in small software companies the proposed objective of this study is to examine the influence of software domain complexity on corrective maintenance tasks. The empirical study was organized in a local micro company with majority of resources dedicated to maintenance activities. The rest of the article is structured as follows. The second section provides a literature review of studies dealing with corrective maintenance tasks. The third section presents the study conducted in a micro software company, followed with the sections in which limitations and validity, as well as implications of the presented research are discussed. The last section contains concluding remarks and outlines future research directions.

2. Related work

It has been recognized in software industry, and reported in empirical studies, that software maintenance tasks are complex and require skilled maintainers (Jones, 2010; Bourque & Fairley,

2014). The complexity of maintenance tasks is the consequence of the following facts (Podnar & Mikac, 2001): (1) they are implemented on complex software systems, (2), they involve people who have different roles in the maintenance process, and (3) they contain several feedback loops that ensure the flow of information between participants in the process. (Ko *et al.*, 2006) emphasized the importance of collecting and tracking information relevant for each specific maintenance task, while (Vasilev, 2012) pointed out the importance of information related to processes for reduction of costs and enterprise practice improvement. Proposed or required maintenance task is usually defined in a textual field, which contains maintenance request description that is essential for efficient performance of a maintenance task (Mockus & Votta, 2000). Understanding of maintenance requests' descriptions requires both technical knowledge specific for software systems and domain knowledge specific for the domain of software use. According to (Boehm & Basili, 2001) understanding of context dependent-factors (e.g. the level of data coupling and cohesion, data size and complexity) can positively contribute to corrective maintenance tasks

(Vans *et al.*, 1999) conducted a field study with four professional software maintainers engaged in maintaining large-scale software systems, aimed at investigating program understanding behavior in corrective maintenance tasks. During the study, the authors observed maintainers while they were solving corrective maintenance tasks. Data analysis revealed that maintainers work at three levels of abstractions: code, algorithm and application domain. In addition, maintainers regularly switch between these abstraction levels based on the current problem they solve. Maintainers need information about domain concepts and connect this information to software being maintained during corrective tasks.

(De Lucia *et al.*, 2005) presented an empirical study aimed at assessing and improving the effort estimation models for corrective maintenance in an international software enterprise. The study contained two phases. In the first phase was constructed a multiple linear regression models that were validated against real data from five corrective maintenance project. In the second phase the authors replicated the assessment of the constructed models from the first phase on a new corrective maintenance project. The results enable prediction of trends for corrective maintenance tasks, while early estimates of the average number of corrective tasks contribute to practice improvement in the selected company.

(Wang & Arisholm, 2009) investigated the difficulty level of maintenance tasks based on the number and complexity of classes that would be affected by the change. The study was based on two controlled experiments with 3rd to 5th year software engineering students without prior knowledge of the software being maintained. Results revealed that solving easier tasks (less complex) before harder (more complex) is more appropriate for inexperienced programmers, and that task order influence correctness of performed maintenance tasks.

(Li *et al.*, 2010) presented an empirical study aimed at analyzing around 1400 corrective maintenance activities associated to defect reports in two large software companies in Norway. The most important cost drivers for corrective maintenance tasks identified in the first company are: size of the system to be maintained, complexity of the system to be maintained and maintainers experience. In the second company the most important cost driver is domain knowledge. These results indicate that: (1) models resulted from empirical research studies should be customized for each company based on its specific characteristics, and (2) maintainers experience and domain knowledge significantly influence corrective task performance.

(Nguyen *et al.*, 2011) conducted a controlled experiment to assess the productivity and effort distribution of three different maintenance types: enhancive, corrective, and reductive. As the metrics were used three independent LOC (lines of code) metrics (added, modified, and deleted). Results revealed that: (1) the productivity of corrective maintenance is significantly lower than that of the other types of maintenance, and (2) task comprehension activity is the most complex task in maintenance. Based on these results it is evident that corrective maintenance tasks are more complex than other maintenance tasks, which requires highly skilled maintainers.

(Lee *et al.*, 2015) organized a qualitative study aimed at identifying factors that impact effort in corrective maintenance tasks. By using causal mapping methodology, the authors identified and ranked a set of 17 factors that contribute to corrective maintenance tasks implementation. Among all identified factors *High code complexity* (structural complexity) was ranked as the most critical with the weighted score of 0.8027, while *High version/deployment complexity* (management of multiple versions of software systems) was ranked at the 12th place with the weighted score of 0.6508. Regarding the influence of maintenance request description, the important identified factor is *Low clarity or availability of defect documentation*, which is ranked at the 10th place with weighted score of 0.6729. The identified factors suggest that complexity factors related to software structure and domain of problem should be taken into account in corrective maintenance tasks.

(Lenarduzzi *et al.*, 2018) presented an industrial case study aimed at prioritizing corrective maintenance tasks caused by crash reports and the exceptions for android applications for the period of four years. The applications were developed by an Italian public transportation company, while crash reports were collected directly from the Google Play Store. The study results indicate that six exceptions caused over 70% corrective tasks, and that most of the exceptions were generated by bad development practices. Results are useful for the selected company for improving its corrective maintenance efforts.

To summarize, there has been a large number of empirical studies addressing corrective maintenance tasks. Some studies were organized at universities (Wang & Arisholm, 2009; Nguyen *et al.*, 2011), while some of them were organized in industrial settings (De Lucia *et al.*, 2005; Li *et al.*, 2010; Lee *et al.*, 2015; Lenarduzzi *et al.*, 2018). Although these studies address different aspects of corrective maintenance in different settings, there is significant space for researching this important segment of industrial practice. Our study differs from the outlined studies because it deals with the subjective assessment of domain complexity influence on corrective maintenance tasks, which has not been addressed in previous research.

3. Case study

The study was conducted in an indigenous software company, which can be classified as a micro enterprise according to European Commission for Enterprise and industry publications (Commission, 2015). The company has 7 employees: 3 senior programmers, 3 junior programmers and 1 administrative worker. The company develops business software applications for local clients in Serbia. Totally 48 software applications are used by over 100 client companies in Serbia.

Data analysis is based on historical data extracted from the company internal repository of tasks, which is common practice in empirical software engineering studies aimed at investigating

Table 1. Distribution of software maintenance tasks according to the typology proposed in (Stojanov *et al.*, 2017)

Maintenance task type	Number of tasks	Share [%]
Adaptation	22	1.08
Correction	489	24.02
Enhancement	1050	51.57
Preventive	8	0.39
Support	467	22.94
TOTAL	**2036**	**100.00**

what happens in everyday practice (Dit *et al.*, 2013; Stojanov *et al.*, 2013*b*). The study is a continuation of the research on maintenance trends in the selected company (Stojanov *et al.*, 2013*a*), but with the improved typology of software maintenance tasks introduced in (Stojanov *et al.*, 2017). The data set consists of totally 2293 tasks solved in 2013 and 2014 years, where 2036 tasks were categorized as maintenance tasks (88.79% of all tasks). The classification of software maintenance tasks according to the typology presented in (Stojanov *et al.*, 2017) is presented in table 1.

Maintenance tasks are created in order to solve maintenance requests (MR) submitted by software users. Submission of a MR assumes that a user should provide a textual description of a request and indicate a software application to which a MR relates to. Each request is assigned to one of the programmers who is responsible for maintaining a target software application. Maintenance task record contains the fields that enable tracking of all relevant data for processing associated MR and calculating costs of implemented work.

Corrective maintenance tasks account for almost one quarter of all maintenance tasks (24.02%), and since these tasks relates to direct solving of client problems with software, they deserve attention to be analyzed. The aim of the study is to analyze the influence of domain complexity on corrective maintenance tasks complexity, where domain complexity reflects the complexity of a business domain in which software is used.

3.1. Maintenance tasks

Maintenance tasks were recorded in a local repository of tasks in the company. For each task, the following parameters recorded in the repository are interesting for the data analysis:

- *Worker ID*. The identification number of a programmer engaged in solving a task.
- *Application*. The name of the application to which a maintenance task relates to.
- *Maintenance request description*. The description of a maintenance request to which the task is associated to.
- *Working Hours Company [WHC]*. Working hours spend in the company on solving the task.

Table 2. Distribution of corrective maintenance tasks on software applications

Software application	Number of tasks	Share [%]
Application 1	89	18.20
Application 2	48	9.82
Application 3	97	19.84
Application 4	28	5.73
Application 5	97	19.84
Other 29 applications	130	26.58
TOTAL	**489**	**100.00**

- *Working Hours Internet [WHI].* Working hours spend at Internet on solving the task. Programmer works in the company and uses Internet to access software application at client side.
- *Working Hours Client Side [WHCS].* Working hours spend at the client side (in the client's organization) on solving the task.
- *Working Hours TOTAL [WHT]* Total number of working hours spent on solving the task

Total number of working hours is the cum of three types of working hours, which is expressed in the following way

$$WHT = WHC + WHI + WHCS. \tag{3.1}$$

The values for WHC, WHI, and WHCS enters a programmer assigned to the task after completing it. The value for WHT is calculated and stored in the repository.

3.2. Corrective maintenance trends in the company

Corrective maintenance tasks were performed on 34 software applications. The data about corrective maintenance tasks were extracted from the local repository for tracking all tasks in the company. Each maintenance task is associated to a maintenance request (MR) received from a user, and assigned to a programmer in the company.

Initial data analysis based on descriptive statistical methods revealed that only 5 software applications have more than 5% of the total number of tasks. Other 29 software applications together consume 26.58% of all tasks, which is approximately less than 1% per software application, which can be treated as insignificant for further statistical data analysis. Based on this fact, further data analysis is focused on the selected 5 software applications. Distribution of corrective maintenance tasks for all software applications is presented in table 2.

For the selected five software applications, average number of working hours spent on corrective tasks is presented in table 3.

Average values of working hours spent on corrective maintenance tasks presented in table 3 revealed the following interesting trends: (1) Average time spent on tasks is usually between 1 and 1.5 hours, (2) tasks associated to applications 2 and 3 last longer than tasks for applications 1,4

Table 3. Average number of working hours for 5 selected software applications

Software application	Average WHC	Average WHI	Average WHCS	Average WHT
Application 1	0.26	0.49	0.45	1.21
Application 2	0.48	0.58	0.35	1.42
Application 3	0.41	0.45	0.66	1.53
Application 4	0.25	0.52	0.54	1.30
Application 5	0.14	0.61	0.49	1.25

Table 4. Scale for subjective rating of domain complexity

Level	Abbreviation	Value
Very Low	VL	1
Low 2	L	2
Medium	M	3
High	H	4
Very High	VH	5

and 5, and (3) programmers usually access clients information system via Internet (WHI) or work at client side (WHCS) since the average values for WHC are the lowest for all applications.

3.3. Domain complexity model

Software is used to solve problems in specific domains of business or living, which influences all requirements and activities related to software. Maintenance activities are triggered by maintenance requests submitted by software users, who describe requests by using unstructured text with domain terminology. These requests should be understandable for programmers who are engaged to solve reported problems by correcting identified faults. Therefore, it is important to describe domain complexity, and develop a model that can be used for modeling complexity of corrective maintenance tasks. Domain complexity reflects intellectual effort required for understanding a domain of software use, and how the domain influences complexity of maintenance tasks performed on that software.

In this study, domain complexity for all software applications was rated by the company manager by using predefined scale with the values presented in table 4.

Since the majority of time and effort in maintenance consume activities related to comprehension of a maintenance request and software to be modified (Von Mayrhauser & Vans, 1995; O'Brien *et al.*, 2004), complexity in this study relates to understanding a maintenance task for the given domain of a software application based on the description available in the maintenance request. For that purpose, a set of subjective measures (parameters) for domain complexity for all software applications is defined:

- *Terminology Complexity (TC)*. Complexity of terminology used for defining and describing entities, relations and processes in a domain.

Table 5. Subjective ratings of domain complexity for selected software applications

Software application	TC	RC	BPC	HFC
Application 1	4	5	5	5
Application 2	4	5	4	4
Application 3	5	5	4	5
Application 4	4	3	3	3
Application 5	5	5	4	5

- *Relations complexity (RC)*. Complexity of relations between entities and processes in a domain.
- *Business processes complexity (BPC)*. Complexity of business processes in a domain (process flow, sub-processes, constraints, inputs and outputs).
- *Human Factor Complexity (HFC)*. Complexity of humans who perform business processes in a domain, including the number and roles of people engaged in business processes.

The model of domain complexity assumes subjective rating of each specific parameter of domain complexity obtained from the company manager (the most experienced programmer). Subjective ratings of domain complexity TC, RC, BPC and HFC for selected 5 software applications are presented in table 5.

3.4. Task complexity model

Task is the basic unit of work in software maintenance in the company, aimed at solving a maintenance request. Each task is performed by one programmer, and always is associated to a specific software application. The task is defined with the description of a maintenance request, which is in the form of unstructured text submitted by a user. The description is implemented as a text field in each task record stored in a local repository of tasks. For each task description complexity measures are defined as a subjective ratings of TC, RC, BPC and HFC parameters expressed with values from the table 4. These values present subjective ratings of a task complexity provided by a programmer engaged in solving the task. Maintenance task complexity is expressed as

$$TaskCompl = TC * mtTC + RC * mtRC + BPC * mtBPC + HFC * mtHFC, \qquad (3.2)$$

where coefficients TC, RC, BPC and HFC presents specific domain complexities for the selected application, while mtTC, mtRC, mtBPC and mtHFC presents specific complexities for a selected maintenance task.

For each of the five selected applications, overall maintenance task complexity is calculated for all maintenance tasks by using formula 3.2.

For the extracted data and defined subjective specific domain complexity measures for all software applications, the arithmetic mean (MEAN), standard deviation (STDEV) and coefficient

Table 6. Measures of spread for corrective maintenance task complexity affected by domain complexity parameters for selected software applications

Software application	MEAN	STDEV	CV [%]
Application 1	50.79	11.00	21.66
Application 2	40.63	8.46	20.82
Application 3	39.43	10.63	26.95
Application 4	31.50	6.96	22.10
Application 5	29.99	6.47	21.56

Table 7. Correlation coefficients between calculated corrective maintenance task complexity and working hours spent on solving task

	WHC	WHI	WHCS	WHT
Task complexity of Application 1	0.10	0.19	0.05	0.35
Task complexity of Application 2	0.74	0.18	0.02	0.85
Task complexity of Application 3	0.61	0.17	0.48	0.81
Task complexity of Application 4	-0.08	-0.06	0.63	0.71
Task complexity of Application 5	0.05	-0.13	0.59	0.61

of variance (CV) for all tasks are calculated (Buglear, 2001). Calculated values are presented in table 6.

Data presented in table 6 revealed that Application 1 has the highest complexity of maintenance tasks (50.79 in average), followed with Application 2 with maintenance task complexity of 40.63 in average, while the simplest tasks are tasks related to Application 5 (29.99 in average). Tasks are almost twice as complex for Application 1 than for Application 5.

Variance coefficient analysis for selected applications revealed that the spread of task complexity for each application is acceptable (between 20.82 for Application 2 and 26.95 for application 3), which indicates small variances of task complexity. This enables more reliable predictions of task complexity for further maintenance activities. Based on data presented in table 6, the most reliable prediction of maintenance task complexity can be given for Application 2, while the most unreliable predictions of maintenance task complexity are for Application 3.

3.5. The task complexity and working hours correlation

Table 7 presents correlation coefficients between corrective maintenance tasks complexity and working hours spent on solving tasks. Correlation coefficients are calculated between task complexity and each type of working hours: in the company (WHC), at Internet (WHI), in the client company (WHCS) and total working hours WHT calculated by using formula 3.1.

Data presented in table 7 revealed that the correlation between task complexity and total working hours (WHT) vary from 0.35 for the Application 1 to 0.85 for the Application 2. Calculated values indicate strong correlations for Application 2, Application 3 and Application 4, which means that based on subjective evaluation of task complexity provided by a programmer, reliable

assessment of total working hours can be given. For Application 1 (r=0.35) and Application 5 (r=0.61) it is not possible to reliably estimate working hours.

Although estimates of total working hours based on task complexity are reliable for Application 2, Application 3 and Application 4, correlation between task complexity and specific types of working hours (WHC, WHI and WHCS) are weak, which means that it is not possible to estimate these specific working hours. The only exception is correlation between task complexity and WHC for the Application 2 with value of 0.74, which means that only working hours in the company can be estimated for the Application 2.

4. Limitations and validity

Despite the clear and useful results obtained through empirical data analysis, this study certainly has some limitations that influence the results and conclusions. The first limitation is quite simple mathematical model for calculating task complexity. The model resulted with results that enable reliable estimates in some segments of maintenance practice, but it will be incrementally improved in order to increase reliability of decisions based on obtained results.

The next limitation relates to initial examination and proper preprocessing of data that deviate from typical values in empirical data set (outliers) (Chatfield, 1985; Cousineau & Chartier, 2010). Data analysis with appropriate treatments of outliers could provide more reliable results and estimates, which will be used for assessment and improvement of task complexity models, and finally better planning and decision making in the company. This limitation will be addressed in further research, and results will be compared with results obtained in this study.

Internal and external validity are commonly used for judging quality and reliability of empirical studies in software engineering (Kitchenham *et al.*, 2002; Shull *et al.*, 2008). Internal validity relates to selection and definition of used parameters (variables) and proper use of selected data analysis methods that leads to reliable results. The main threat to internal validity relates to data set used for modeling domain complexity of maintenance tasks, which is collection of subjective measures provided by programmers for each maintenance tasks. The improvement of the presented model will include more objective measures based on data extracted directly from maintenance request descriptions and data related to technical details of maintained software applications (e.g. number of lines and modules affected by maintenance request). This improvement of task complexity model requires more accurate data in the company repository of tasks, and will be addressed in future research after improving recording of maintenance tasks in the company.

(Briand *et al.*, 2017) discussed context-driven aspect of empirical research in software engineering and suggested that there is no need to force external validity issue related to generalizability of study results. However, generalizability can be viewed from the aspect of used research methods that may produce specific, but different, findings in other settings. Therefore, it is possible to use the presented methods for analyzing complexity of tasks in other software (or engineering) organizations and get context-specific results that can be of benefit for these organizations.

5. Implications and benefits

Despite limitations stated in the previous section, this research has significant benefits and implications for practice and research in the field of software engineering. Study design and results

can be of benefits to the selected company, software industry in general, and software engineering research community.

The benefits for the selected company are: (1) The presented model of task complexity enables calculation of complexity for corrective maintenance tasks by considering subjective evaluation of maintainers that solve these tasks, which further enables identification of trends for task complexity for each software application, (2) Based on the calculated task complexity, the company staff can estimate required time for solving maintenance tasks based on the interpretation of the correlation between task complexity and spent working hours, and (3) Based on calculated task complexity and estimated time for solving the task, the company management can design more reliable and effective organization of maintenance activities in the company (e.g. scheduling of maintenance tasks among programmers in order to accelerate processing of maintenance requests).

Managers and experts from software industry can find useful directions how to collect and use field data in their organizations for developing models for task complexity by considering some specific characteristics of the practice in their organizations. In addition, they can find some directions for correlating task complexity with elements of planning in their organizations

Researchers can find lessons how to organize an empirical study aimed at assessing complexity of tasks in real industrial settings by considering subjective evaluations of specific parameters that affects specific types of tasks. Presented study shows how to identify attributes that influence task complexity, how to define a scale for evaluating specific attributes of task complexity, and how to identify the correlation between the complexity of tasks and organizational parameters that are essential for planning activities in a selected organization.

6. Concluding remarks

As the task complexity is significant factor that affects software maintenance, presented study contributes to software maintenance practice and research. The study presents the model for calculating the complexity of corrective maintenance tasks, which is based on subjective evaluations of domain specific factors provided by programmers engaged in handling maintenance tasks. The model enables calculation of corrective maintenance task complexity, which can be further used for estimating the time needed to solve the tasks. The results can be useful for planning in everyday maintenance practice in the selected software company, but the study design can be implemented in other software companies by considering their specificity.

Several further work directions can be distinguished. The first direction relates to including other factors that influence software maintenance tasks in the analysis of task complexity. These factors might be characteristics of human factor involved in maintenance tasks (experience, knowledge of specific software technologies, domain knowledge, communication skills) and objective (quantitative) attributes of software applications such as structural complexity of maintained software systems. The second direction relates to developing more accurate model for task complexity by including preliminary analysis of empirical data and excluding from the analysis all data that significantly variate from a typical set of values. The third direction relates to implementation of the proposed model on other types of maintenance tasks in the selected company (enhancement and support tasks) which will enable more reliable planning and scheduling of all maintenance

activities. And finally, adaptation of the presented model to other software organizations by considering their specificity is also challenging research direction that will provide further evaluation of the model usefulness.

Acknowledgments

Ministry of Education, Science and Technological Development, Republic of Serbia, supports this research under the project "*The development of software tools for business process analysis and improvement*", project number TR32044, 2011-2018.

References

April, Alain and Alain Abran (2008). *Software Maintenance Management: Evaluation and Continuous Improvement*. John Wiley & Sons. Hoboken, NJ, USA.

April, Alain and Alain Abran (2009). A software maintenance maturity model (S3M): Measurement practices at maturity levels 3 and 4. *Electronic Notes in Theoretical Computer Science* **233**, 73–87. Proceedings of the International Workshop on Software Quality and Maintainability (SQM 2008).

Banker, Rajiv D. and Sandra A. Slaughter (1997). A field study of scale economies in software maintenance. *Management Science* **43**(12), 1709–1725.

Boehm, Barry and Victor R. Basili (2001). Software defect reduction top 10 list. *Computer* **34**(1), 135–137.

Bourque, Pierre and Fairley, Richard E. (Dick), Eds. (2014). *Guide to the Software Engineering Body of Knowledge (SWEBOK), Guide V3.0*. IEEE Press. Piscataway, NJ, USA.

Braarud, Per Øivind (2001). Subjective task complexity and subjective workload: Criterion validity for complex team tasks. *International Journal of Cognitive Ergonomics* **5**(3), 261–273.

Briand, L., D. Bianculli, S. Nejati, F. Pastore and M. Sabetzadeh (2017). The case for context-driven software engineering research: Generalizability is overrated. *IEEE Software* **34**(5), 72–75. DOI: 10.1109/MS.2017.3571562.

Buglear, John (2001). *Stats means business: a guide to business statistics*. Buttworth-Heinemann. Oxford, UK.

Campbell, Donald J. (1988). Task complexity: A review and analysis. *The Academy of Management Review* **13**(1), 40–52.

Capretz, Luiz Fernando (2014). Bringing the human factor to software engineering. *IEEE Software* **31**(2), 104.

Chapin, Ned, Joanne E. Hale, Khaled Md. Kham, Juan F. Ramil and Wui-Gee Tan (2001). Types of software evolution and software maintenance. *Journal of Software Maintenance* **13**(1), 3–30.

Chatfield, C. (1985). The initial examination of data. *Journal of the Royal Statistical Society. Series A (General)* **148**(3), 214–253.

Commission, European (2015). *User guide to the SME Definition*. Enterprise and industry publications. Publications Office of the European Union. Luxembourg. http://ec.europa.eu/regional_policy/sources/conferences/state-aid/sme/smedefinitionguide_en.pdf [accessed 30.6.2018.].

Cousineau, Denis and Sylvain Chartier (2010). Outliers detection and treatment: a review. *International Journal of Psychological Research* **3**(1), 58–67.

De Lucia, Andrea, Eugenio Pompella and Silvio Stefanucci (2005). Assessing effort estimation models for corrective maintenance through empirical studies. *Information and Software Technology* **47**(1), 3–15.

Dit, B., A. Holtzhauer, D. Poshyvanyk and H. Kagdi (2013). A dataset from change history to support evaluation of software maintenance tasks. In: *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)*. San Francisco, CA, USA. pp. 131–134.

Dybå, Tore, Rafael Prikladnicki, Kari Rönkkö, Carolyn Seaman and Jonathan Sillito (2011). Qualitative research in software engineering. *Empirical Software Engineering* **16**(4), 425–429.

Grubb, Penny and Armstrong A. Takang (2003). *Software Maintenance: Concepts and Practice*. 2nd ed.. World Scientific Publishing Company. Singapore.

Hatton, Les (2007). How accurately do engineers predict software maintenance tasks?. *Computer* **40**(2), 64–69.

ISO (2006). *ISO/IEC 14764:2006 and IEEE Std 14764-2006. Software Engineering - Software Life Cycle Processes - Maintenance*. ISO. Geneve, Switzerland.

Jones, Capers (2010). *Software Engineering Best Practices*. McGraw-Hill. New York, NY, USA.

Junio, Gladston Aparecido, Marcelo Nassau Malta, Humberto de Almeida Mossri, Humberto T. Marques-Neto and Marco Tulio Valente (2011). On the benefits of planning and grouping software maintenance requests. In: *Proceedings of the 15th European Conference on Software Maintenance and Reengineering*. Oldenburg, Germany. pp. 55–64.

Keshavarz, Ghazal, Nasser Modiri and Mirmohsen Pedram (2011). Metric for early measurement of software complexity. *International Journal on Computer Science and Engineering* **3**(6), 2482–2490.

Kitchenham, Barbara A., Shari Lawrence Pfleeger, Lesley M. Pickard, Peter W. Jones, David C. Hoaglin, Khaled El Emam and Jarrett Rosenberg (2002). Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering* **28**(8), 721–734.

Ko, Andrew J., Brad A. Myers, Michael J. Coblenz and Htet Htet Aung (2006). An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Transactions on Software Engineering* **32**(12), 971–987.

Lee, Michael, Marcus A. Rothenberger and Ken Peffers (2015). Effort estimation factors for corrective software maintenance projects: A qualitative analysis of estimation criteria. *Journal of Information Technology Theory and Application* **16**(2), 39–56.

Lenarduzzi, Valentina, Alexandru Cristian Stan, Davide Taibi, Gustavs Venters and Markus Windegger (2018). Prioritizing corrective maintenance activities for android applications: An industrial case study on android crash reports. In: *Software Quality: Methods and Tools for Better Software and Systems* (Dietmar Winkler, Stefan Biffl and Johannes Bergsmann, Eds.). Vol. 302 of *Lecture Notes in Business Information Processing*. pp. 133–143. Springer, Cham. Vienna, Austria.

Li, Jingyue, T. Stalhane, J. M. W. Kristiansen and R. Conradi (2010). Cost drivers of software corrective maintenance: An empirical study in two companies. In: *Proceedings of 2010 IEEE International Conference on Software Maintenance*. Timisoara, Romania. pp. 1–8.

Li, Wei and H. Delugach (1997). Software metrics and application domain complexity. In: *Proceedings of Joint 4th International Computer Science Conference and 4th Asia Pacific Software Engineering Conference*. Hong Kong. pp. 513–514.

Lilienthal, Carola (2009). Architectural complexity of large-scale software systems. In: *Proceedings of the 13th European Conference on Software Maintenance and Reengineering*. Kaiserslautern, Germany. pp. 17–26.

Lu, Yao, Xinjun Mao and Zude Li (2016). Assessing software maintainability based on class diagram design: A preliminary case study. *Lecture Notes on Software Engineering* **4**(1), 53–58.

Maynard, Douglas C. and Milton D. Hakel (1997). Effects of objective and subjective task complexity on performance. *Human Performance* **10**(4), 303–330.

Mendes-Moreira, H. M. C. L. and C. G. Davies (1993). Business domain knowledge libraries to support software maintenance activities. *Journal of Software Maintenance: Research and Practice* **5**(3), 165–179.

Mockus, Audris and Lawrence G. Votta (2000). Identifying reasons for software changes using historic databases. In: *Proceedings of 2000 International Conference on Software Maintenance*. San Jose, CA, USA. pp. 120–130.

Nguyen, Vu, Barry Boehm and Phongphan Danphitsanuphan (2011). A controlled experiment in assessing and estimating software maintenance tasks. *Information and Software Technology* **53**(6), 682 – 691.

O'Brien, Michael P., Jim Buckley and Teresa M. Shaft (2004). Expectation-based, inference-based, and bottom-up software comprehension. *Journal of Software Maintenance and Evolution: Research and Practice* **16**(6), 427–447.

Parkes, Alison (2017). The effect of individual and task characteristics on decision aid reliance. *Behaviour & Infor-*

mation Technology **36**(2), 165–177.

Pino, Francisco J., Francisco Ruiz, Félix García and Mario Piattini (2012). A software maintenance methodology for small organizations: Agile_MANTEMA. *Journal of Software: Evolution and Process* **24**(8), 851–876.

Podnar, Ivana and Branko Mikac (2001). Software maintenance process analysis using discrete-event simulation. In: *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*. CSMR '01. Lisbon, Portugal. pp. 192–195.

Shaft, Teresa M. and Iris Vessey (1998). The relevance of application domain knowledge: Characterizing the computer program comprehension process. *Journal of Management Information Systems* **15**(1), 51–78.

Shull, Forrest, Singer, Janice and Sjøberg, Dag I.K., Eds. (2008). *Guide to Advanced Empirical Software Engineering*. first ed.. Springer-Verlag London. London, UK.

Stojanov, Zeljko, Dalibor Dobrilovic and Jelena Stojanov (2013*a*). Analyzing trends for maintenance request process assessment: Empirical investigation in a very small software company. *Theory and Applications of Mathematics & Computer Science* **3**(2), 59–74.

Stojanov, Zeljko, Dalibor Dobrilovic, Jelena Stojanov and Vesna Jevtic (2013*b*). Estimating software maintenance effort by analyzing historical data in a very small software company. *Scientific Bulletin of The Politehnica University of Timioara, Transactions on Automatic Control and Computer Science* **58 (72)**(2), 131–138.

Stojanov, Zeljko, Jelena Stojanov, Dalibor Dobrilovic and Nikola Petrov (2017). Trends in software maintenance tasks distribution among programmers: A study in a micro software company. In: *IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY 2017)*. Subotica, Serbia. pp. 23–27. DOI: 10.1109/SISY.2017.8080547.

Suh, S. D. and I. Neamtiu (2010). Studying software evolution for taming software complexity. In: *Proceedings of the 21st Australian Software Engineering Conference*. ASWEC. Auckland, New Zealand. pp. 3–12.

Sun, Xiaobing, Bixin Li, Hareton Leung, Bin Li and Yun Li (2015). Msr4sm: Using topic models to effectively mining software repositories for software maintenance tasks. *Information and Software Technology* **66**, 1–12.

Swanson, E. Burton (1976). The dimensions of maintenance. In: *Proceedings of the 2nd international conference on Software engineering*. ICSE '76. San Francisco, CA, USA. pp. 492–497.

Tan, Y. and V. S. Mookerjee (2005). Comparing uniform and flexible policies for software maintenance and replacement. *IEEE Transactions on Software Engineering* **31**(3), 238–255.

Tripathy, Priyadarshi and Kshirasagar Naik (2015). *Software evolution and maintenance: a practitioner' s approach*. John Wiley & Sons. Hoboken, NJ, USA.

Vans, A. Marie, Anneliese von Mayrhauser and Gabriel Somlo (1999). Program understanding behavior during corrective maintenance of large-scale software. *International Journal of Human-Computer Studies* **51**(1), 31–70.

Vasilev, Julian (2012). Guidelines for improvement information processes in commerce by implementing the link between a web application and cash registers. *Theory and Applications of Mathematics & Computer Science* **2**(2), 55–66.

Von Mayrhauser, A. and A. M. Vans (1995). Program comprehension during software maintenance and evolution. *Computer* **28**(8), 44–55.

Wang, Alf Inge and Erik Arisholm (2009). The effect of task order on the maintainability of object-oriented software. *Information and Software Technology* **51**(2), 293–305.

Wood, Robert E. (1986). Task complexity: Definition of the construct. *Organizational Behavior and Human Decision Processes* **37**(1), 60–82.