Вычислительные технологии, 2023, том 28, № 1, с. 92–103. © ФИЦ ИВТ, 2023
Computational Technologies, 2023, vol. 28, no. 1, pp. 92–103. © FRC ICT, 2023

ISSN 1560-7534
eISSN 2313-691X

DOI:10.25743/ICT.2023.28.1.010

Maintainability of data-driven software systems: review and experiences

Z. Stojanov
University of Novi Sad, Technical Faculty "Mihajlo Pupin", 23000, Zrenjanin, Serbia
Corresponding author: Stojanov Zeljko, e-mail: zeljko.stojanov@uns.ac.rs
Received April 30, 2022, accepted May 16, 2022.

Software maintenance as the most demanding and the most expensive phase in software life cycle should be considered and planned from the early stages of software development. Maintainability is an essential property of software systems which relates to easy and efficient modification of software system during its use. Therefore, maintainability should be considered during software design. This article presents the author's experience related to improving maintainability of data-driven software systems by integrating software change request service into software. The service is available in visual forms and assists users in specifying software change requests. In addition, the service creates an initial impact set, which contains software elements affected by the proposed software change request. Design, implementation and evaluation of the service are presented, followed by the discussion of identified advantages and disadvantages.

Keywords: maintainability, software maintenance, software development, data-driven architecture, software architecture, software change request.

Citation: Stojanov Z. Maintainability of data-driven software systems: review and experiences. Computational Technologies. 2023; 28(1):92–103. DOI:10.25743/ICT.2023.28.1.010.

Introduction

The costs and complexity of software maintenance activities has been recognized as crucial for successful software engineering practice, which indicates that software maintenance is the most demanding and difficult phase in software life cycle [1–4]. Successful maintenance should align with organizational and business objectives and strategies of software organization that provides maintenance services, but it also should satisfy clients requirements and increase their satisfaction [5].

Software maintenance practice is significantly influenced by design decisions and selection of methods during software development [6]. Minimization or even avoidance of problems in maintenance phase of software life cycle can be achieved by adopting and consistently implementing suitable development methodology [7]. Selection and design of the most suitable software architecture plays the most important role in enabling easy modification of software during maintenance phase [2]. In order to achieve the highest maintainability level, software architecture should be designed by using fine-grained and self-contained components that can be easily modified with minimal and controlled effect on other components. Ideal situation is when software engineers set some maintainability objectives early in the development phase, which enables measurement of software quality through the whole life cycle and efficiency of both development and maintenance activities. This leads to improvement of development and maintenance practice [1].

During software maintenance phase, all changes implemented in software should be controlled and handled in order to avoid inconsistencies in software architecture [8], or even in enterprise architecture [9], which points out that there is a need to develop methods that can improve maintainability as one of the most important software quality attribute. In addition, Parhizkar and Comuzzi [10] argued that careful planning and implementation of post-implementation modifications is essential for success of business software systems, which require development of methods and tools that support handling of changes and change impact analysis. Martinez et al. [11] suggested that there is a need for more empirical studies reporting the evidence on maintainability of model-driven software system.

Based on the above statements, an approach for improving maintainability of data-driven software systems is proposed and evaluated in a qualitative empirical study. This approach is based on design and integration of a software change request (SCR) service in data-driven software system that is developed for managing network scenarios within virtual learning laboratory. Design, implementation and evaluation of the SCR service are presented in the following sections, followed by the discussion of advantages and disadvantages of the SCR service and concluding remarks.

1. Maintainability of data-driven software systems

Changes or modifications of software are inevitable for software that strive to remain usable in changing business environment. Impact and costs of each individual change must be analyzed before implementing the change, because the change may propagate to the entire software system with many side effects [12]. Maintainability has been recognized as an essential attribute of good software since it refers to how easy it is to modify software during its evolution. However, maintainability is usually assessed and predicted by using process metrics, rather than product metrics (complexity, number of lines of code, number of classes, etc.). For predicting maintainability, the following process metrics can be used [2]:

- *Number of requests for corrective maintenance.* Maintainability declines if the number of bugs and failures increase over time, which indicates that new errors are introduced in software during maintenance phase.
- *Average time required for impact analysis.* Maintainability decreases if the time for detecting software components affected by modification request increases, which reflects more complex software structure. As software structure becomes more complex, it takes more time for impact analysis of new modification requests, leading to increased costs of software maintenance.
- *Average time taken to implement a modification request.* If the time for modification of software components affected by modification request increases (after impact analysis), it is indication that maintainability declines.
- *Number of outstanding modification requests.* If the number of remaining or unresolved requests increases over time, it indicates a decline in maintainability, leading to more demanding maintenance tasks.

The above stated maintainability metrics suggest that there is a need for developing software systems whose modification can be easily implemented. This means that modification requests (change requests) should be more precisely specified by users, and after that efficiently handled and solved by maintainers. Since the impact on user habits is unpredictable, the most suitable way for increasing maintainability of software systems is to design architecture of software systems in a way that enables easy and fast modifications. Several studies dealing with maintainability prediction are focused on discovering which parts of software systems are the most expensive for maintaining [13–15].

Recognized complexity and cost of software maintenance activities have influenced development of approaches to data-driven design aimed at improving maintainability of software systems. Lin et al. [16] proposed an approach for change request management in model-driven engineering of software in industrial automation systems. The approach enables easy analysis of impact and feasibility of change, which is tested in a laboratory scale production line. Dam et al. [9] presented an approach that supports change propagation in the maintenance of Service Oriented Architecture (SOA) models. The authors proposed ChangeAwareHierarchicalEA language that supports change analysis and propagation at business and technical levels. Ricca et al. [17] conducted controlled experiments aimed at comparing the influence of model-driven and code-centric development to software maintainability. Study results indicate that model-driven development reduce maintenance time, but not the correctness of maintenance tasks.

Literature review indicates that there is a need for more research on model-driven data-intensive software systems, while Hutchinson et el. [18] suggested that there is a lack of empirical studies reporting industrial experience, especially in the field of software maintenance. Software maintainability, as an essential software architecture quality attribute, should be considered when selecting or developing model-driven platform and approach [19], and especially in cases when hand-written code is integrated into model-driven generated code, which may decrease maintainability [20].

All above considerations revealed that there is a significant space for developing methods and tools that support maintainability of model-driven data-intensive software systems.

2. Experience with data-driven software application

This section presents the author's experience related to improving software maintainability of data-driven software systems by extending software architecture with software change request service. The objective is to propose an approach for assisting in software maintenance through detection of software elements affected by proposed change requests. This approach can be applied to software systems with graphical user interface (GUI) based on intensive use of data, management and control, and computation [21].

Literature review of critical steps in the process of software modification and the author's experience in researching software maintenance practices in very small software companies [22] formed the basis for developing an approach to software design that provides support for efficient processing of software change requests (modification requests). Experience in development, implementation and evaluation of the proposed data-driven development approach focused on improving maintainability of software systems is presented in the following subsections.

2.1. Data-driven development approach

Development of data-driven software application with integrated change request service assumes the following technical aspects: (1) modelling software architecture, with the focus on data-driven organization of software elements, (2) designing application user interface based on data model, and (3) modelling software change request service, with the focus on software change request and initial change impact set.

Software architecture. The main idea of this approach is to propose software architecture based on the data-driven organization of logical packages dealing with entities, which is reflected in a user interface design. Data-driven organization of a layered software system is presented in Fig. 1.

In this approach, design of visual forms in the user interface, as well as navigation among visual forms, reflect data-driven model. With this approach in the user interface design it becomes easy to track user session and to detect elements of software affected by the proposed change (initial impact set [23, 24]) specified by the user.

The service packages contains software elements that provide specific services to other element or packages in the software application. The services have manifestation in the user interface layer, have processing elements, and some services have elements for storing data. The most important service enables specification of software change requests in the context of running application. This service is named software change request service (SCR service) and it can be called in the same way as any other service available within the user interface.

The main form package contains elements for initializing and starting the software application, together with the main application visual form. Access to other visual forms and services is available within the main visual form.

The main part of the proposed data-driven software architecture are entity packages. These packages are associated with entities in the database and contains software elements from all three layers (see Fig. 1). The elements in each entity package are entity abstract class, visual forms located in the user interface layer (graphical user interface (GUI)), background processing elements located in the middle layer, and elements in the database layer (each table in the database is associated with strictly one entity). The main element is abstract class `Entity`, which defines entity for which the logical package is created. Entity may have table model `EntityTableModel`, which is used for correct manipulating and presenting tabular data for the entity in the form `FormMultipleRecords`. Visual form `FormSingleRecord` is

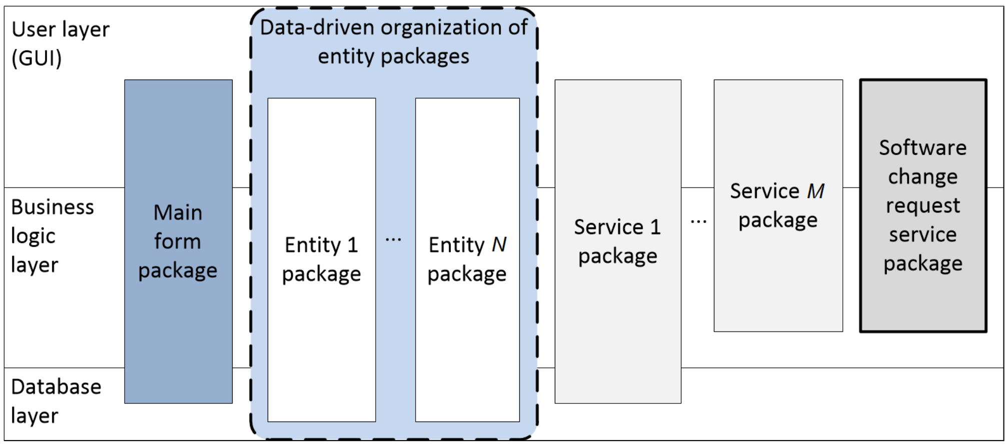


Fig. 1. Layered data-driven software organization

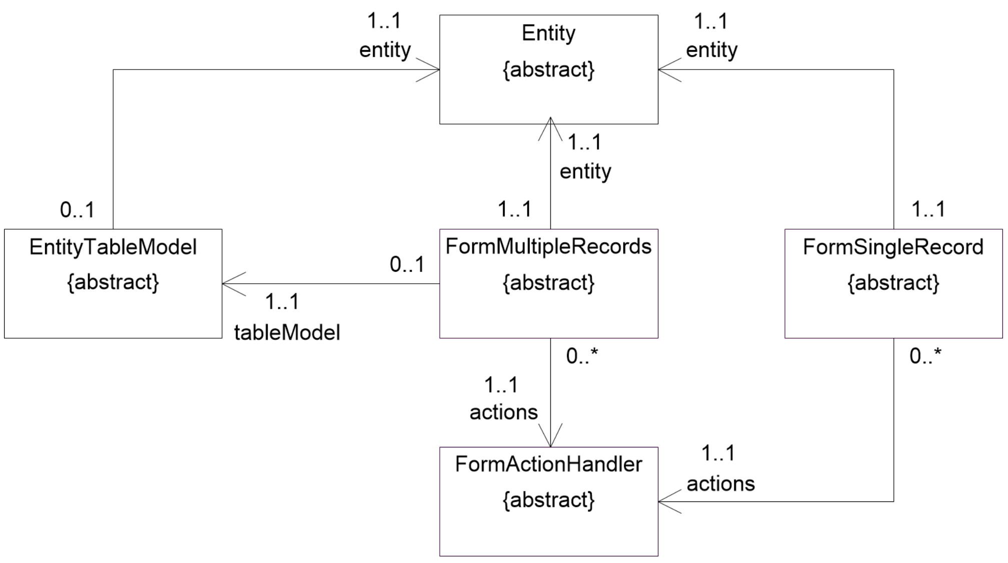


Fig. 2. Entity package structure

used for manipulation of a single entity instance. Actions for all visual forms are implemented within abstract class `FormActionHandler`, which implements interface `IFormActionHandler` with the defined set of methods. The structure of the entity package is presented in Fig. 2.

Graphical user interface. Proposed user interface design with standard visual forms is based on the model of generic form [25]. The aim is that user interface provides a distinctive and uniform look and feel for users [26], while preserving efficiency and intuitiveness [27]. User interface reflects data-driven architecture of software, which means that each entity in the data model has a corresponding visual form for manipulating one record of the entity (`FormSingleRecord`), and a visual form for tabular representation and manipulation of multiple records of the entity (`FormMultipleRecords`).

In addition to standard functionalities, both types of visual forms have the ability to call a service for specifying a software change request, which is called in the same way as other functionalities. A layout of standard visual forms for working with data entities is presented in Fig. 3.

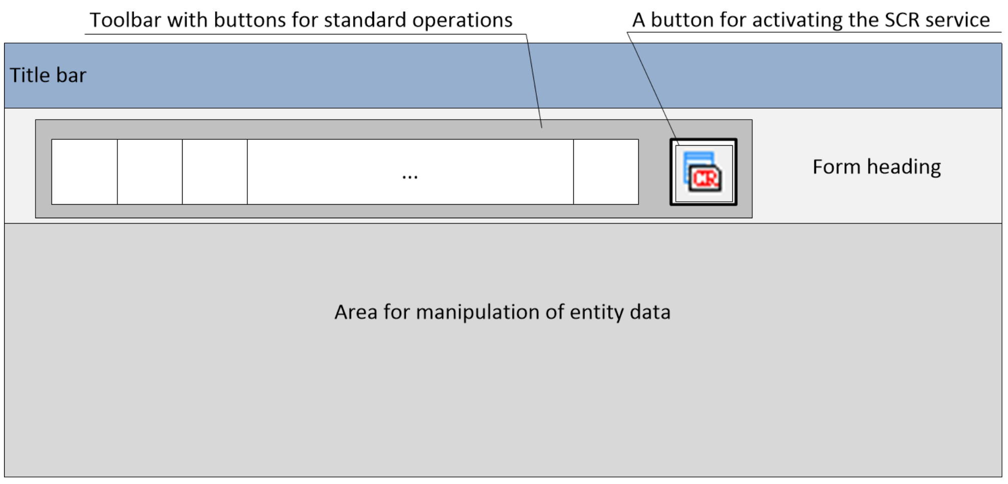

Toolbar with buttons for standard operations
A button for activating the SCR service
Title bar
...
CR
Form heading
Area for manipulation of entity data

Fig. 3. Layout of standard visual forms for manipulating entities

Software change request service. `SCRSpecifier` is the key software component in SCR service that enables specification of software change requests in the context of the current active visual form. Information about the current form context is collected in the object of type `SCREvent` (software change request event) when the SCR service is invoked. `SCRSpecifier` receives `SCREvent` object and starts a visual form `FormSCRSpecifier` that provides visual context for completing specification of the request, which is recorded in an object of type `SoftwareChangeRequest`. The user can see the state of the current request by invoking a visual form of a type `FormSCRPreview`. Model of the SCR service is presented in Fig. 4.

Software change request is specified in an XML document by using marshalling technique available in JAXB (Java architecture for XML binding) technology [28]. JAXB is selected because it enables easy mapping of complex structure of objects from Java code to an XML document that reflects that structure. XML document with specified SCR is sent to a web service for collecting maintenance requests [29].

Support for maintainability is implemented through support for identification of initial impact sets from specified SCRs. Initial impact set is a set of software elements affected by the proposed SCR. Automated identification of an initial impact set is enabled with code instrumentation technique [30], which means that code for tracking and recording user session at visual forms level is inserted into software application before compilation. This trace of user session is included into created SCR, which is later used for identifying software elements affected by the SCR. This approach for creating session trace by tracking user movement through user interface (transition from one visual form to another) relies directly on data-driven architecture in which user interface reflects data structure.

2.2. Implementation of data-driven development approach

Data-driven model of software with integrated service for specifying SCRs is implemented in software application ScenarioBuilder which is used for managing network scenarios for teaching computer networks concepts within a virtual network laboratory at the faculty in Zrenjanin, Serbia. Computer network scenarios are specified by using network node description language (NNDL), which is an XML based language developed for this laboratory [31].

Specifications of network scenarios in NNDL language are used as a starting point for developing a model for data-driven software application [32]. Data-driven design process of ScenarioBuilder is presented in Fig. 5.

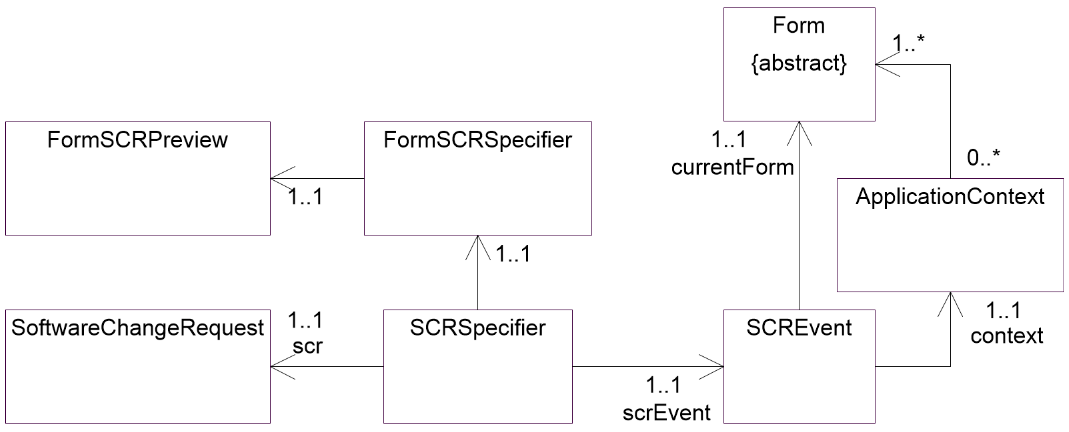


Fig. 4. Model of SCR service

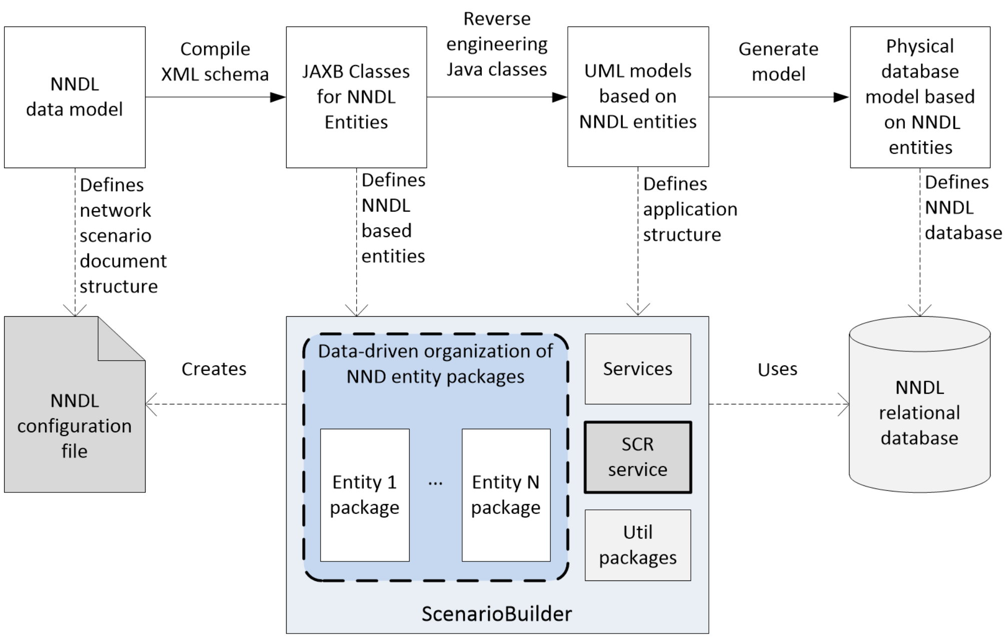
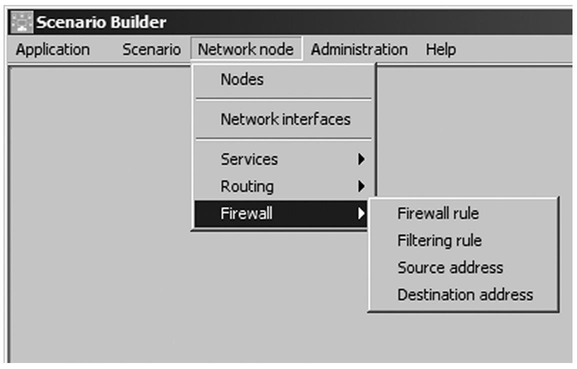

NNDL data model
Compile XML schema
JAXB Classes for NNDL Entities
Reverse engineering Java classes
UML models based on NNDL entities
Generate model
Physical database model based on NNDL entities
Defines network scenario document structure
Defines NNDL based entities
Defines application structure
Defines NNDL database
NNDL configuration file
Creates
Data-driven organization of NND entity packages
Entity 1 package
...
Entity N package
Services
SCR service
Util packages
ScenarioBuilder
Uses
NNDL relational database

Fig. 5. Engineering process of data-driven design based on NNDL



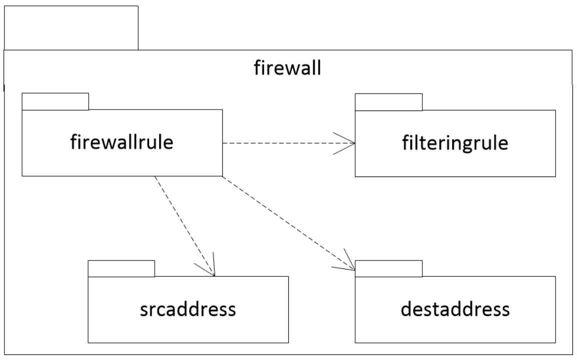
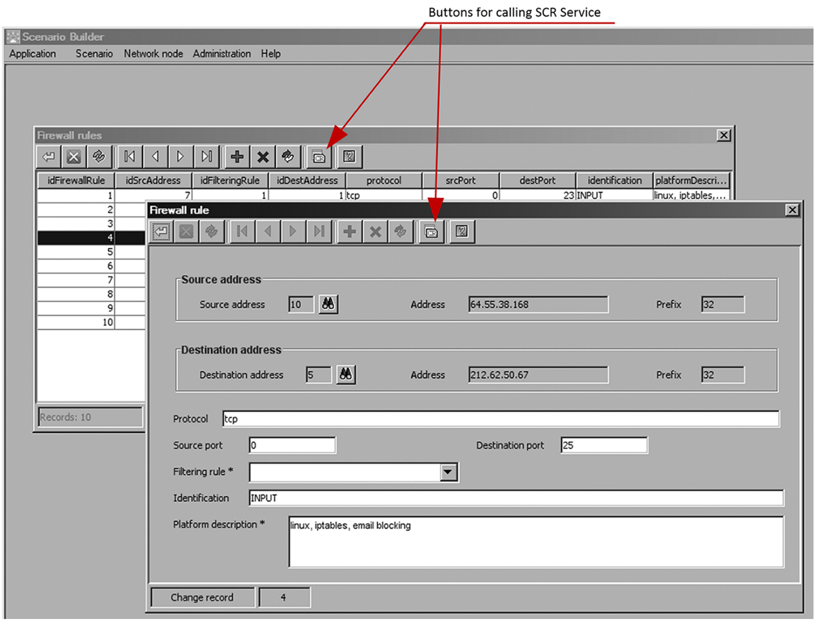


Fig. 6. Submenu associated with firewall NNDL entity package

Fig. 7. Structure of firewall NNDL entity package



idFirewallRule	idSrcAddress	idFilteringRule	idDestAddress	protocol	srcPort	destPort	identification	platformDescri...
1	7	1	1	tcp	0	23	INPUT	linux, iptables,...
2								
3								
4								
5								
6								
7								
8								
9								
10								

Records: 10

Source address
Source address: 10   Address: 64.55.38.168   Prefix: 32

Destination address
Destination address: 5   Address: 212.62.50.67   Prefix: 32

Protocol: tcp
Source port: 0   Destination port: 25
Filtering rule *:
Identification: INPUT
Platform description *: linux, iptables, email blocking

Change record   4

Fig. 8. Calling SCR service from visual forms for Firewall Rule NNDL entity

As an example of data-driven package, submenu in *ScenarioBuilder* that is associated to Firewall NNDL entity package is presented in Fig. 6. The `firewall` entity package contains the following basic packages that correspond to the basic concepts on which the firewall is based in NNDL: firewall rule (`firewallrule`), filtering rule (`filteringrule`), source address (`srcaddress`), and destination address (`destaddress`). The structure of `firewall` NNDL entity package is presented in Fig. 7.

SCR service can be called from visual forms of all NNDL based entities, including single entity forms (`FormSingleRecord`) and forms for handling multiple entities (`FormMultipleRecords`), which is presented in Fig. 8.

2.3. Evaluation of SCR service

Evaluation of the SCR service was conducted as a part of the evaluation of all software maintenance services within virtual learning environment at Technical faculty "Mihajlo Pupin" Zrenjanin, Serbia [33, 34], while details on evaluation of the SCR service integrated into ScenarioBuilder was presented in [32]. The objective of the SCR service evaluation was to determine advantages and disadvantages of the SCR service, which can serve as a basis for improving maintenance services.

Evaluation was performed by using qualitative methods, which enable collecting real experience of laboratory users (students). In evaluation participated 31 students who were included in three focus groups organized in the laboratory at the faculty. Focus groups were selected as a method for collecting data from participants that fosters communication and discussion, and can help in collecting real and original raw empirical data [35]. Data from the students were collected as answers on open-ended questions [36, 37], which ensures that the students expressed their personal views and opinions. The use of open-ended questions ensures that collected answers are not anticipated, but reflect the real users' experiences.

Data analysis of students answers was performed by using grounded theory coding techniques proposed by Charmaz [38]. Advantages of the SCR service discovered through qualitative data analysis are presented in Table 1, while discovered disadvantages are presented in Table 2.

T a b l e 1. SCR service advantages

Advantage	Description
Assistance in the use	The service gathers all application related data for the specified request in the context of the current visual form, which means that users should add only additional description and files if there are any
Intuitiveness	Since the service is fully integrated into the software and available in user interface, its activation and use can be easily perceived and mastered
Reliability	Service automates some steps in specifying requests, which minimizes possibility for user mistakes. This is very important because service identifies software elements affected by the change, and user should only provide additional description of the proposed change
Simplicity	Straightforward access and use of the service contribute to its simplicity, which means that the process of activating and using service can be easily comprehended and completed
Availability	The service is available in all visual forms in the software application, which means that users can activate it without leaving the current working context (the current visual form)

T a b l e 2. SCR service disadvantages

Disadvantage	Description
Confusing	Confusion is caused by the introduction of a new functionality in visual forms (user interface). The problem is that users acquire some routines and habits in using software, and adding new functionality should be planned
Misuse	Due to the service high availability in visual forms, it can be misused for submitting requests for non existing problems

3. Discussion

Presented software design approach based on data-driven architecture and user interface that reflects data organization is suitable for integration of SCR service that enables tracking user session, provides assistance to users in specifying change requests, and enables easy identification of software elements affected by the requested change (initial impact set). Identified service advantages revealed that service assists users in specifying change request, leading to more reliable service, while making easier maintenance tasks at the same time (increasing maintainability).

Discovered disadvantages form the basis for service improvement, which leads to increased maintainability of software. Since the disadvantages are expressed by the software users, implementation of improvements will lead to solving stated problems with the service. Based on the stated disadvantages (see Table 2) identified improvements for the service are: (1) inclusion of additional verification that created change request reflects the real user needs, and (2) integration of help for using service in the visual forms.

Conclusion

This paper presents the authors experience in designing, implementing and evaluating data-driven software systems aimed at increasing their maintainability. Maintainability is supported through integration of the SCR service into visual forms that form software user interface, while data-driven organization of software architecture supports easy identification of software elements affected by change request (initial impact set).

Discovered SCR service advantages through evaluation by the users revealed that service is useful and can make software maintenance easier. Discovered disadvantages form the basis for service improvement, leading to even more easier and reliable maintenance.

Acknowledgements. 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 No TR32044.

References

[1] **April A., Abran A.** Software maintenance management: evaluation and continuous improvement. Hoboken: Wiley-IEEE Computer Society; 2008: 314.

[2] **Sommerville I.** Software engineering. Boston: Addison Wesley; 2011: 773.

[3] **Bourque P., Fairley R.E.D.** Guide to the software engineering body of knowledge (SWEBOK). Piscataway: IEEE Press; 2014.

[4] **Tripathy P., Naik K.** Software evolution and maintenance: a practitioner' s approach. Hoboken: John Wiley & Sons; 2015: 416. DOI:10.1002/9781118964637.

[5] **Stojanov Z.** Software maintenance improvement in small software companies: reflections on experiences. CEUR-WS Proceedings. 2021; (2913):182–197. Available at: `https://ceur-ws.org/Vol-2913/paper14.pdf`.

[6] **Zhang X., Windsor J.C.** An empirical analysis of software volatility and related factors. Industrial Management and Data Systems. 2003; (103):275–281. DOI:10.1108/02635570310470683.

[7] **Schach S.R., Tomer A.** A maintenance-oriented approach to software construction. Journal of Software Maintenance. 2000; (12):25–45.

[8] **Li B., Sun X., Leung H., Zhang S.** A survey of code-based change impact analysis techniques. Software Testing, Verification and Reliability. 2013; (23):613–646.

[9] **Dam H.K., Le L.-S., Ghose A.** Managing changes in the enterprise architecture modelling context. Enterprise Information Systems. 2016; (10):666–696. DOI:10.1080/17517575.2014.986219.

[10] **Parhizkar M., Comuzzi M.** Impact analysis of ERP post-implementation modifications: design, tool support and evaluation. Computers in Industry. 2017; (84):25–38. DOI:10.1016/j.compind.2016.11.003.

[11] **Martinez Y., Cachero C., Melia S.** Empirical study on the maintainability of web applications: model-driven engineering vs code-centric. Empirical Software Engineering. 2014; (19):1887–1920. DOI:10.1007/s10664-013-9269-5.

[12] **Elkholy M., Elfatatry A.** Change taxonomy: a fine-grained classification of software change. IT Professional. 2018; (20):28–36. DOI:10.1109/MITP.2018.043141666.

[13] **Anda B.C.D.** Assessing software system maintainability using structural measures and expert assessments. Proceedings of the 23rd International Conference on Software Maintenance (ICSM 2007). Paris, France. 2007: 204–213. DOI:10.1109/ICSM.2007.4362633.

[14] **Kozlov D., Koskinen J., Sakkinen M., Markkula V.** Assessing maintainability change over multiple software releases. Journal of Software Maintenance and Evolution: Research and Practice. 2008; (20):31–58. DOI:10.1002/smr.361.

[15] **Riaz M., Mendes E., Tempero E.** A systematic review of software maintainability prediction and metrics. Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM'09). Lake Buena Vista, FL, USA. 2009: 367–377. DOI:10.1109/ESEM.2009.5314233.

[16] **Lin H.-Y., Sierla S., Papakonstantinou N., Shalyto A., Vyatkin V.** Change request management in model-driven engineering of industrial automation software. Proceedings of the IEEE 13th International Conference on Industrial Informatics (INDIN 2015). Cambridge, UK. 2015: 1186–1191. DOI:10.1109/INDIN.2015.7281904.

[17] **Ricca F., Torchiano M., Leotta M., Tiso A., Guerrini G., Reggio G.** On the impact of state-based model-driven development on maintainability: a family of experiments using unimod. Empirical Software Engineering. 2018; (23):1743–1790. DOI:10.1007/s10664-017-9563-8.

[18] **Hutchinson J., Whittle J., Rouncefield M.** Model-driven engineering practices in industry: social, organizational and managerial factors that lead to success or failure. Science of Computer Programming, Special Issue on Success Stories in Model Driven Engineering. 2014; (89):144–161. DOI:10.1016/j.scico.2013.03.017.

[19] **Farshidi S., Jansen S., Fortuin S.** Model-driven development platform selection: four industry case studies. Software and Systems Modeling. 2021; (20):1525–1551. DOI:10.1007/s10270-020-00855-w.

[20] **Rahad K., Badreddin O., Mohsin Reza S.** The human in model-driven engineering loop: a case study on integrating handwritten code in model-driven engineering repositories. Software: Practice and Experience. 2021; (51):1308–1321. DOI:10.1002/spe.2957.

[21] **Forward A., Lethbridge T.C.** A taxonomy of software types to facilitate search and evidence-based software engineering. Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds (CASCON'08). Ontario, Canada. 2008: (14):179–191. DOI:10.1145/1463788.1463807.

[22] **Stojanov Z.** Discovering automation level of software change request process from qualitative empirical data. Proceedings of the 6th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI 2011). Timisoara, Romania. 2011: 51–56. DOI:10.1109/SACI.2011.5872972.

[23] **Arnold R., Bohner S.** Software change impact analysis. Los Alamitos, CA, USA: Wiley-IEEE Computer Society Press; 1996: 376.

[24] **Sun X., Li B., Tao C., Wen W., Zhang S.** Change impact analysis based on a taxonomy of change types. Proceedings of the IEEE 34th Annual Computer Software and Applications Conference. Seoul, Korea. 2010: 373–382. DOI:10.1109/COMPSAC.2010.45.

[25] **Milosavljevic G., Perisic B.** A method and a tool for rapid prototyping of large-scale business information systems. Computer Science and Information Systems (ComSIS). 2004: (1):57–82.

[26] **Wettasinghe M.** Branding the feel: applying standards to enable a uniform user experience. CHI'08 Extended Abstracts on Human Factors in Computing Systems (CHI EA'08). Florence, Italy. 2008: 2265–2268. DOI:10.1145/1358628.1358665.

[27] **Mandel T.** User/system interface design. Encyclopedia of Information Systems. N.Y.: Elsevier; 2003; (1):535–549. DOI:10.1016/B0-12-227240-4/00190-8.

[28] **Vajjhala S., Fialli J.** JSR-000222, Java architecture for XML binding (JAXB) 2.0. Santa Clara, USA: Sun Microsystems, Inc.; 2006. Available at: `http://www.jcp.org/en/jsr/detail?id=222`.

[29] **Stojanov Z., Dobrilovic D.** An approach to integration of maintenance services in educational web portal. Proceedings of the 8th International Symposium on Intelligent Systems and Informatics (SISY2010). Subotica, Serbia. 2010: 443–448. DOI:10.1109/SISY.2010.5647343.

[30] **Geimer M., Shende S.S., Malony A.D., Wolf F.** A generic and configurable source-code instrumentation component. Computational Science: Proceedings 9th International Conference (ICCS 2009). Part II. Berlin, Heidelberg: Springer Berlin Heidelberg; 2009: 696–705. DOI:10.1007/978-3-642-01973-9_78.

[31] **Dobrilovic D., Stojanov Z., Odadzic B., Markoski B.** Using network node description language for modeling networking scenarios. Advances in Engineering Software. 2012; (43):53–64. DOI:10.1016/j.advengsoft.2011.08.004.

[32] **Stojanov Z., Dobrilovic D., Stojanov J.** Extending data-driven model of software with software change request service. Enterprise Information Systems. 2018; (12):982–1006. DOI:10.1080/17517575.2018.1445296.

[33] **Stojanov Z., Dobrilovic D., Perisic B.** Integrating software change request services into virtual laboratory environment: empirical evaluation. Computer Applications in Engineering Education. 2014; (22):63–71. DOI:10.1002/cae.20529.

[34] **Stojanov Z., Dobrilovic D.** Qualitative evaluation of software maintenance services integrated in a virtual learning environment. International Journal of Engineering Education. 2016; (32):790–803.

[35] **Stewart D.W., Shamdasani P.N., Rook D.W.** Focus groups: theory and practice. London: SAGE Publications; 2007.

[36] **Roulston K.J.** Open-ended question. L.M. Given. The SAGE Encyclopedia of Qualitative Research Methods. Thousand Oaks, CA, US: SAGE Publications; 2008: 582.

[37] **Popping R.** Analyzing open-ended questions by means of text analysis procedures. Bulletin of Sociological Methodology. 2015; (128):23–39. DOI:10.1177/0759106315597389.

[38] **Charmaz K.** Constructing grounded theory: a practical guide through qualitative analysis. London: Sage Publications; 2006: 224.

Вычислительные технологии, 2023, том 28, № 1, с. 92–103. © ФИЦ ИВТ, 2023 ISSN 1560-7534
Computational Technologies, 2023, vol. 28, no. 1, pp. 92–103. © FRC ICT, 2023 eISSN 2313-691X

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

DOI:10.25743/ICT.2023.28.1.010

Обслуживаемость программных систем, управляемых данными: обзор и опыт

Ж. Стоянов

Технический факультет им. Михайло Пупина, Университет Нови-Сад, 23000, Зренянин, Сербия
Контактный автор: Стоянов Желько, e-mail: zeljko.stojanov@uns.ac.rs

Поступила 30 апреля 2022 г., принята в печать 16 мая 2022 г.

Аннотация

Сопровождение программного обеспечения как наиболее требовательную и самую дорогостоящую фазу жизненного цикла программного обеспечения следует рассматривать и планировать с самых ранних стадий его разработки. Ремонтопригодность является важным свойством программных систем, которое связано с легкой и эффективной модификацией программной системы во время ее использования. Следовательно, при разработке программного обеспечения следует учитывать удобство сопровождения.

В статье представлен опыт автора, связанный с повышением ремонтопригодности программных систем, управляемых данными, путем интеграции в программное обеспечение службы запросов на его изменение. Услуга доступна в визуальных формах и помогает пользователям указывать запросы на изменение программного обеспечения. Кроме того, сервис создает начальный набор воздействий, который содержит программные элементы, затронутые предлагаемым запросом на изменение программного обеспечения. Представлены дизайн, реализация и оценка услуги, после чего следует обсуждение выявленных преимуществ и недостатков.

Ключевые слова: обслуживаемость; обслуживание программного обеспечения; разработка программного обеспечения; архитектура, управляемая данными; архитектура программного обеспечения; запросы на изменение программного обеспечения.

Цитирование: Стоянов Ж. Обслуживаемость программных систем, управляемых данными: обзор и опыт. Вычислительные технологии. 2023; 28(1):92–103. DOI:10.25743/ICT.2023.28.1.010.

Благодарности. Министерство образования, науки и технологического развития Республики Сербии поддержало это исследование в рамках проекта № TR32044 “Разработка программных инструментов для анализа и улучшения бизнес-процессов”.