CAUSE-BASED MODEL OF SOFTWARE EVOLUTION

Miroslav SABO
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice,
Letná 9, 042 00 Košice, Slovak Republic, tel.: +421 55 602 4179, e-mail: miroslav.sabo@tuke.sk

ABSTRACT

Domain-specific languages are used to develop highly specialized software. From the implementation perspective, evolution of such software can not utilize evolutionary methods used for software developed in a traditional way. This paper presents the model of evolution, considering the software system as a composition of two separate parts – domain-specific language reflecting the application environment and system by itself reflecting the actual solution to a specified problem. The process of evolution is driven in accordance to the nature of evolutionary change. The paper also discusses the categorization of changes by cause which induced them.

Keywords: *cause of change, complexity of software system, domain-specific language, language evolution*

1. INTRODUCTION

The laws of software evolution were written by Lehman in early 1970s [10] but despite the long time period, tools for effective solving of the problems identified by these laws have not yet been developed. The model of software evolution proposed in this paper is targeted towards first two of the laws - *law of continuing change* and *law of increasing complexity* and should serve as a common basis for development of tool support for software evolution.

Law of continuing change states that effectivity of the system will be progressively deteriorated until it is continually adapted to changes in the application environment. Many solutions have been proposed to address this law [1, 6, 16], but success was always achieved by increase of complexity of the system. The negative side effect of the first law is also the main concern of the second Lehman's law. It states that as system evolves, its complexity will continually increase until progressive or anti-regressive effort is invested into maintaining or reducing it. This means that with changes implemented to the system successively one upon each other, interactions and dependencies between system elements increase in an unstructured pattern and lead to an increase in system's entropy. The best results in addressing this issue have been achieved with the generative methods of software development. In this approach changes are applied to the model of a system on higher level of abstraction and final implementation is generated from this model automatically.

Model of software evolution proposed in this paper introduces the differentiation between two parts of a system which represent the application environment and actual solution to a specified problem. That way evolutionary changes can be applied directly to the subject they concern, without increasing the overall complexity of a system. Categorization of evolutionary changes by the cause which induced them is defined in Sec. 3. Changes of each category are targeted towards specific part of the system, modification of which does not influence other parts, therefore overall complexity of a system is well preserved during the whole process of evolution.

2. INEVITABILITY OF CHANGE

Change is the main characteristic of software evolution as software systems have to react on constantly evolving requirements and underlying platforms and other impulses from environment which they operate in. Changes are inevitable from different reasons:

- **New requirements on system** – requirements on system can change early in the process of software development but this phase may not always be convenient for their implementation from different reasons (e.g. firmly determined deadlines do not allow for unforeseen activities). On the other hand, it is the pressure from satisfied customers which are creating new requirements for functional extensions of a system.
- **Modelling of reality** – as the environment of a system dynamically evolves and changes, system must be continually adapted else it becomes progressively less satisfactory [10]. In extreme cases when systems are interconnected with application environment too tightly, environment is influenced by the system right after the deployment which in turn results in immediate need for adaptation of the system on these changes.
- **Bug fixing** – these requirements arise mainly in the testing phase.
- **Architectural changes** – significant changes in the structure of a system (e.g. system working with business processes evolves and increasing complexity requires integration of the rule engine which will interact with many modules within the system).
- **Enhancing the performance and reliability of the system**.

3. TYPOLOGY OF SOFTWARE EVOLUTION

In the 1970s, Swanson proposed the typology of software maintenance [11] which was distinguishing between maintenance activities accordingly to the purpose which they were executed for:

Unauthenticated | 194.138.39.60
Download Date | 11/14/13 3:19 PM
VERSITA EMERGING SCIENCE PUBLISHERS

1. **adaptive** – ensure the usability of a system after changes in environment or technical infrastructure of a system happen.

2. **corrective** – remove bugs from implementation, usually cover the solving of problems caused by discrepancies between requirements and implementation.

3. **perfective** – any enhancements which increase the quality of a system (e.g. adding new features, increasing performance or system documentation).

Some taxonomies [5] added another category:

4. **preventive** – this last category is often the subject of discussions, considered by some as part of perfective maintenance [2]. IEEE software engineering terminology standard [5] defines preventive maintenance as "maintenance executed with intention to prevent the problems before they even occur".

This typology had been rafined over time and based on the work experience the classification of 12 types of software evolution and software maintenance [3] was defined later (Tab. 1).

Table 1 Typology of software evolution (E) and software maintenance (M)

Object of change	Type of change	E/M
Business rules	Enhancive Corrective Reductive	E/M
Software properties	Adaptive Performance Preventive Groomative	E/M M
Documentation	Updative Reformative	M
Support interface	Evaluative Consultive Training	M

Complementary view on this topic presents Mens in his work [12] which is focused towards technical aspects of the software change. He proposes the taxonomy of software evolution based on characteristic mechanisms of change and factors which influence these mechanisms.

Even though precise fine-grained typology of software evolution is well documented, the model of software evolution proposed in this paper distinguishes only four fundamental types of evolution - adaptive, corrective, perfective and preventive.

4. CAUSE-DRIVEN SOFTWARE EVOLUTION MODEL

Model of software evolution proposed in this paper is focused on elimination of the negative side effect of adaptation to continually evolving environment - increased complexity of a system. The main idea is targeting the application of changes strictly to those parts of the system implementation which represent the evolved objects in real world. For systems developed in general purpose languages this constitutes a complicated problem because development requires implementation of the concepts of application environment at first and just after that the new solution by itself may be implemented. Both implementations are tangled together and therefore adaptation of the system to environmental changes requires identification of the parts of a system to be adapted before adaptation can be executed. Even after that, change of the adapted code may be delegated further into system because of tangled code. The result is increased complexity of the structure of system. Proposed model of software evolution separates the implementation of application environment from the implementation of the solution to a problem therefore evolution can be targeted directly to the actual subject of change. Domain-specific languages, as technology following the principles of generative approach to software development, are utilized as tool for representation of application environment of the evolving system.

4.1. Software evolution in domain-specific languages

For the implementation of the change in software systems developed in general purpose languages (GPLs), its type is not relevant because all changes are applied on the same level - source code of the application. It does not matter whether changes relate directly to the change in specification of the system or are induced by the change of environment thus do not relate to the specification at all.

From the perspective of evolution, development of software systems in domain-specific languages offers some benefits. Changes are always executed on the level which they directly relate to. Domain-specific languages (DSLs) are by definition [17] languages which directly reflect some specific domain. Therefore they can be considered as a model of application domain [4]. The implication of this is that any changes arisen in the application domain should be reflected in appropriate DSL which models this domain. Contrary to software systems developed in GPLs, the impact of such changes is on models/specifications of systems developed in DSL minimal or none [8]. Considering the evolution induced by change of environment and not by the change of definition of a problem, DSL approach follows these events precisely:

- changes in environment ⇒ changes in DSL and generator/interpreter

- (no) changes in definition of the problem ⇒ (no) changes in model/specification of the application

Unauthenticated | 194.138.39.60
Download Date | 11/14/13 5:19 AM
VERSITA EMERGING SCIENCE PUBLISHERS

4.2. Causes of change

Changes which occur during software evolution can emerge from 3 different sources:

1. application domain
2. solution specification (problem definition)
3. solution implementation

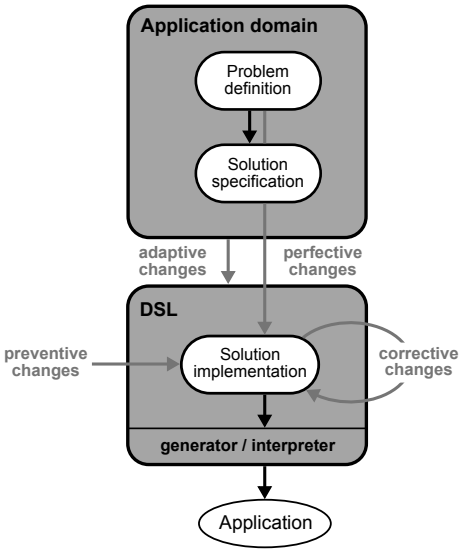

Fig. 1 Causal model of the software evolution

Evolutionary changes of application domain are reflected in the modification of a DSL in which the application is modeled/implemented. The modifications include addition, removal or adjustment of the language constructs, rearrangement of relations between the constructs and adjustment of the generator or interpreter. The system as a composition of two separate parts – language and actual solution to a problem – brings the advantage of minimal to none impact on a solution while performing the modifications to the language. The changes are introduced to the new version of a system either by its regeneration using an evolved generator or by using an evolved interpreter [7, 9]. Changes caused by evolution of application domain are called *adaptive changes*.

Changes of the solution model/implementation are either caused by issues discovered in the previous version (*corrective changes*) or result from new requirements on performance, usability, maintainability or other attributes of a system (*perfective* and *preventive changes*). These changes are performed directly to the model/implementation of a solution or they might also require some minor modifications of a language.

5. SOFTWARE EVOLUTION USING CAUSAL MODEL

Considering the target of a change implementation, as defined in Tab. 2, evolution of a software system can be divided into:

1. evolution of a language
2. evolution of a solution model/implementation

Table 2 Sources and targets of evolutionary changes

Change		Change type			
Source	Target	A	Pe	Pr	C
Application domain	DSL	*			
Solution specification	Solution		*	*	
Solution implementation					*

5.1. Evolution of a language

Evolution of a language is closely related to the manner of a language design. Domain-specific languages are generally designed in two ways - *internal* and *external*. Considering internal DSLs as higher level abstraction of GPLs, the evolution of such languages gets down to the common evolution of a code written in general purpose language.

On the other hand, external DSLs are usually designed using metamodelling approaches of language workbenches specialized for language development [14]. After the model of a language is created, the complete development environment for the new language, including tools such as editors, browsers, generators and interpreters, is generated automatically from the model. The evolution of such DSLs therefore boils down to the modification of the model of a language [17].

Similar approach to the evolution of external textual DSLs is provided by the language development tool YAJCo [13,15] which is based on the definition of abstract syntax. Model of the language consists of Java classes which represent abstract syntax. Concrete syntax is defined upon abstract syntax classes through annotations. The change of the language, in the same manner as language workbenches, requires only modifications on the abstract and concrete syntax level and new generator for the language is created automatically.

5.2. Evolution of a solution model/implementation

Evolution of a solution implementation, which might be considered as a model, is similar to the evolution of the program written in GPL. The usage of DSL, however, brings some advantages specific for this approach such as implementation directly in the concepts of the domain, domain-specific control checking and domain-specific optimization.

The biggest advantage of using domain-specific languages, however, is that changes targeting solution model/implementation can be performed in a straightforward manner because implementation language (DSL) in which the changes will be applied is on the same level of abstraction as language in which the requirements are specified (language of domain experts). All in all, evolution of the software systems developed in domain-specific languages is simple, easy to execute and less error-prone.

Unauthenticated | 194.138.39.60
Download Date | 11/4/15 5:19 AM
VERSITA EMERGING SCIENCE PUBLISHERS

6. CONCLUSION

In this paper I have presented the cause-based model of software evolution. This model satisfies both of the first two Lehman's laws of software evolution - law of continuing change and law of increasing complexity. Preservation of the complexity of software system during the process of evolution, as a major problem identified by these laws, is achieved by application of evolutionary changes strictly to those elements of the system which represent the subject of change in a real world. The part of a system to accommodate the evolutionary change is determined with respect to the type of a change. The categorization of types of changes, based on the cause which induced them, is also presented in the paper.

ACKNOWLEDGEMENT

This work is the result of the project implementation: Centre of Information and Communication Technologies for Knowledge Systems (ITMS project code: 26220120020) supported by the Research & Development Operational Programme funded by the ERDF.

REFERENCES

[1] CAZOLLA, W. – PINI, S. – ANCONA, M.: Evolving Pointcut Definition to Get Software Evolution. ECOOP '04: Proceedings of the Workshop on Reflection, AOP and Meta-Data for Software Evolution, pp. 83-88, 2004.

[2] CHAPIN, N.: Do We Know What Preventive Maintenance Is? ICSM '00: Proceedings of the International Conference on Software Maintenance (ICSM'00), ISBN 0-7695-0753-0, IEEE Computer Society, Washington, DC, USA, pp. 15–17, 2000.

[3] CHAPIN, N. – HALE, J. E. – KHAM, K. M. – RAMIL, J. F. – TAN, W.: *Types of software evolution and software maintenance*, Journal of Software Maintenance **13**, No. 1 (2001) 3–30

[4] CZARNECKI, K.: Overview of Generative Software Development. pp. 326-341, 2004.

[5] Electrical, Institute O. and (ieee), Electronics E.: IEEE 90: IEEE Standard Glossary of Software Engineering Terminology, 1990.

[6] GREEVY, O. – DUCASSE, S. – GÎRBA, T.: *Analyzing software evolution through feature views: Research Articles*, Journal of Software Maintenance and Evolution: Research and Practise **18**, No. 6 (2006) 425–456

[7] KOLLÁR, J. – PORUBÄN, J. – VÁCLAVÍK, P. – BANDÁKOVÁ, J. – FORGÁČ, M.: Adaptive Compiler Infrastructure. Komunikačné a informačné technológie, ISBN 978-80-8040-324-9, Tatranské Zruby, pp. 4–5, 2007.

[8] KOLLÁR, J. – PORUBÄN, J. – VÁCLAVÍK, P. – FORGÁČ, M. – BANDÁKOVÁ, J.: How to Adapt Programming Languages instead of Software Systems, Computer Science and Technology Research Survey, Košice, Elfa, 2007, 2, pp. 69–79, ISBN 978-80-8086-071-4.

[9] KOLLÁR, J. – PORUBÄN, J.: Building Adaptive Language Systems, INFOCOMP - Journal of Computer Science, 7, 1, 2008, pp. 1–10, 1807-4545.

[10] LEHMAN, M. M.: Laws of Software Evolution Revisited. EWSPT '96: Proceedings of the 5th European Workshop on Software Process Technology, Springer-Verlag, London, UK, pp. 108–124, 1996.

[11] LIENTZ, B. P. – SWANSON, E. B.: Software Maintenance Management: A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations. Addison-Wesley Publications, 1980.

[12] MENS, T. – BUCKLEY, J. – RASHID, A. – ZENGER, M.: Towards a taxonomy of software evolution. ECOOP '02: Proceedings of the Workshop on Unanticipated Software Evolution, Vrije Universiteit Brussel, 2002.

[13] MERNIK, M. – PORUBÄN, J.: Language Design with Concrete/Abstract Syntax: LISA vs. YAJCo Compiler Generators Approaches. Informatics'09: Proceedings of the 10th International Conference on Informatics, Vol. 10, ISBN 978-880-8086-126-1, Elfa, Košice, 2009.

[14] METACASE: MetaEdit+, http://www.metacase.com, 2009.

[15] PORUBÄN, J. – FORGÁČ, M. – SABO, M.: Annotation Based Parser Generator. WAPL '09: Proceedings of the International Multiconference on Computer Science and Information Technology, Vol. 4, ISBN 978-83-60810-22-4, Mragowo, Poland, pp. 707–714, 2009.

[16] REISS, S. P.: Constraining Software Evolution. ICSM '02: Proceedings of the International Conference on Software Maintenance (ICSM'02), ISBN 0-7695-1819-2, IEEE Computer Society, Washington, DC, USA, pp. 162–171, 2002.

[17] SPRINKLE, J. – GRAY, J. – MERNIK, M.: Fundamental Limitations in Domain-Specific Language Evolution. IEEE Transactions on Software Engineering, Vol. 35, No. 3, 2009.

Received September 9, 2010, accepted January 5, 2011

BIOGRAPHY

Miroslav Sabo was born on 16. 11. 1984. In 2008 he graduated (MSc) with distinction at Department of Computers and Informatics of Faculty of Electrical Engineering and Informatics at Technical University in Košice. Currently he is a doctoral student at Department of Computers and Informatics, Technical University of Koice, Slovakia. The subject of his research is the utilization of generative methods in development and evolution of software systems in permanently changing environment.

Unauthenticated | 194.138.39.60
Download Date | 11/14/13 5:19 AM
VERSITA EMERGING SCIENCE PUBLISHERS