A Formalized Architecture-Centric Evolution Process For Component-based Software System

Huaxi (Yulin) Zhang, Lei Zhang, Quan Xu, Christelle Urtado, Sylvain Vauttier, Marianne Huchard

A Formalized Architecture-Centric Evolution Process For Component-based Software System

Huaxi (Yulin) Zhang*, Lei Zhang† , Quan Xu†, Christelle Urtado‡, Sylvain Vauttier‡, and Marianne Huchard§
*Ecole Normal Supérieure de Lyon - INRIA, Lyon, France
Email: huaxi.zhang@ens-lyon.fr
†SKLSPAI, Research Center of Automation, Northeastern University, Shenyang, China
Email: zl.org.cn@gmail.com, quanxu@mail.neu.edu.cn
‡LGI2P / Ecole des Mines d'Alès, Nîmes, France
Email: Christelle.Urtado, Sylvain.Vauttier@mines-ales.fr
§LIRMM, UMR 5506, CNRS and Univ. Montpellier 2, Montpellier, France
Email: huchard@lirmm.fr

***Abstract*—System quality is key part of software system in industry. It not only directly affects the customers/users' satisfaction, but also influences the entire lifecycle of system products from requirement to maintenance. Many quality assurance development methodologies and standards are proposed. However, software evolution as an another important part in software system lifecycle is less studied from the viewpoint of software quality assurance. Architectures, as the most basic and important factor in modern software engineering, are key to guarantee software system quality by replacing codes. Thus, in this paper, we propose a controlled evolution process based on ADLs and formalized by SPEM standard.**

I. INTRODUCTION

Software quality is a major challenge in software industry, as it directly affects customer satisfaction and system acceptance. Software quality is the degree to which software possesses a desired combination of attributes [1]. To control software quality, many different quality assurance developments are proposed, such as Safety lifecycle [2] and RUP [3].

Software evolution as a poster-part of software lifecycle, spends at least 60% of software lifecycle cost. Furthermore, software changes often decline software quality, owing to uncontrolled and unverified software evolution. From architecture viewpoint, the most common phenomena of this quality declination is architecture mismatches [4]. Architecture mismatches are error-prone and especially affect software maintainability for component-based software. Thus how to guarantee software quality during software evolution is an issue raised.

Software quality can be assured from two viewpoints: the viewpoint of software quality itself and the viewpoint of quality management. From the viewpoint of software itself, software architectures are commonly used to control software quality. Software architectures play as the logical underlying to guide software evolution [5]. A well-defined software architecture can better guarantee software quality like maintainability, performance and reliability. The mostly used formal way to define and model software architectures is architecture description languages (ADLs).

From the viewpoint of quality management, quality is managed or controlled by well-defined development processes. Many different standards formalize software processes to ameliorate software quality, such as ISO/IEC9126 [6], ISO/IEC25010 [7], CMMI [8], SPEM2.0 [9], BPMN [10], etc. A lot of software development processes such as RUP and Safety lifecycle [11] are formalized using these standards to manage software processes and thus to control software quality.

However, existing ADLs that support architecture-centric evolution mostly focus their attention on how to support a dynamic software evolution. But merely an ADL orients to study a complete and formalized architecture-centric evolution process which can be used to manage and control quality of software evolution process and prevent the declination of software quality after software evolution.

Thus, in this paper, we propose a controlled architecture-centric evolution process formalized using SPEM standard, which is based on Dedal ADL. Our proposed process has twofold. Firstly, an architecture-centric evolution can be better preserve software quality after evolution. Secondly, a SPEM-formalized and well-defined evolution process also meets the Maturity Level 3 of CMMI-DEV.

The remaining of this paper is organized as follows. Section II presents the context of Architecture-centric software process. Section III discusses existing ADLs how they cover architecture-centric evolution process and the objectives of this paper. Section IV presents our proposed architecture-centric evolution process which is formalized using SPEM. Section V concludes with future work directions.

II. CONTEXT AND BACKGROUND

In this section, we firstly introduce some basic concepts and their definitions related to our works.

A. Related concepts

Lei ZHANG to whom all correspondence should be addressed.

This work was supported by the National Natural Science Foundation of China under Grant No. 61300020, the Scientific Research Funds for Introduced Talents of Northeastern University under Grant No. 28720524, the Fundamental Research Funds for the Central Universities under Grant No. N120408002 and Natural Science & Technology Pillar Program Grant #2012BAF19G00.

1) Software Architectures: A software systems architecture is the set of principal design decisions about the system [12]. Precisely, it describes the constituent elements of this system, their relationships and furthermore, some important software design decisions including functional behaviors, interactions and so on.

2) ADL: Architecture description languages: ADL is a modeling notation of software architecture, which is widely used in software architecture modeling. Using ADLs, an architecture can be formally described and thus its different consistency and consistency can be checked to ensure software quality.

3) Component-based software development (CBSD): Component-based software engineering (CBSE) is an approach to software development that relies on software reuse [13]. Component-based software development is characterized by its implementation of the "reuse in the large" principle. Reusing existing (off-the-shelf) software components therefore becomes the central concern during development. Furthermore, the "reuse in the large" principle implies that component-based software development is *architecture-centric*, as software architectures capture the "principle" design decisions of software systems based on components [14].

B. Software Evolution in CBSE

The lifecycle of software's systems is not finished after the development. The systems are always required to evolve themselves after the release, in order to correct the bugs, improve their performance, etc. Software evolution is also tightly linked with software architectures. The evolution that concerns a collection of software architectural activities to change a software from its older version to the new version and is activated by architecture changes, is defined as *architecture-centric evolution*. These architectural activities can be the modifications of software architecture models or their runtime software counterparts [4].

During the architecture-centric evolution, the architectures of a system should be modified at the same time and synchronized. Unfortunately, this does not always happen in practice. Instead, one architecture (often runtime system – assembly architecture) is often directly modified without accounting for the impact relative to the other levels of the architectures. This resulting discrepancy between a system's architectures is referred to as *architecture mismatches*. Architecture mismatches covers three related phenomena: *architecture drift* [15] [12], *architecture erosion* [15] [12] and *architecture pendency* [4].

- *Architectural drift* is introduction of principal design decisions into a system's lower level of software architecture that (a) are not included in, encompassed by, or implied by the higher level of architecture.
- *Architectural erosion* is the introduction of architecture design decisions into a system's lower level of architecture that violate its higher level of architecture.
- *Architecture pendency* is the introduction of new design decisions into a higher level of architecture that are not implemented by its lower level architecture.

All these three architecture mismatches are often dangerous and expensive, that means the quality of the architecture is much reduced. The quality of the architecture is easily affected by an architecture change. The newer architecture cannot automatically be considered to have the same quality as the older architecture.

Thus, the claim of this paper is: *A planned and controlled evolution process based on the ADL is necessary and a re-verification or deduction of the newer architecture should be executed during this evolution process.*

III. The State of the Art and Objectives

A. The State of the Art

We use a taxonomy [4] as shown in Table I how existing ADLs support evolution process[1] We select six representative ADLs from many existing ADLs, as they all support architecture evolution and are widely used or studied.

TABLE I. THE CHARACTERISTICS AND ACTIVITIES OF ARCHITECTURE CHANGE WITH THEIR POSSIBLE VALUES

Activities of architecture change	**Value**
Consistency checking	−behavior, −interaction, −refinement consistency checking
Impact analysis	−vertical impact, −horizontal impact
Evolution test	−yes, −no
Change propagation	−vertical propagation, −horizontal propagation
Versioning	−state-based versioning, −change-based versioning (extensional, or intentional)

- *Consistency checking.* All existing ADLs support consistency checking, as it's the most important effect of architectures in software evolution. However, the support of these ADLs is incomplete, because none of them checks all kinds of architecture consistencies: name, behavior, interface, interaction and refinement. They more or less particularly concentrate one or two kinds of consistencies, like C2 [17], [18] focuses on refinement consistency, Wright [19] focuses on deadlock, Darwin [20], [21] on State, and SOFA2.0 [22], [23] on behavior.
- *Impact analysis.* Impact analysis in existing ADLs is rarely supported, except Darwin. In Darwin, the impact analysis is held in the same time with consistency checking, as the changes in configuration level will be propagated to runtime level to check the state consistency before enabling the changes.
- *Evolution test.* Most of them do not support evolution test except MAE [24], [25]. This is an ignored point of ADLs as they often focus on checking changes in architecture logical point of view. MAE tests the component substitution at runtime system, however the other operations of changes are not tested like component addition and removal.

[1]The details of taxonomy can be found in Zhang *et al.* 2010 [4] and a taxonomy of Jamshidi *et al.* 2013 [16] inspired from the previous taxonomy.

TABLE II. THE COMPARISON OF CHARACTERS AND ACTIVITIES OF CHANGE IN EXISTING ADLS

Activities of architecture change	C2	Darwin	Dynamic Wright	SOFA2.0	xADL2.0	MAE
Consistency checking	Refinement consistency checking	State consistency checking	Name, interaction and deadlock consistency checking	Behavior consistency checking	—	Sub-type consistency checking
Impact analysis	—	Horizontal impact analysis	—	—	—	—
Evolution test	—	–	—	—	—	Perfective test for component substitution
Change propagation	Horizontal propagation (top-down)	Horizontal propagation (to-down)	—	—	Horizontal propagation (top-donw)	—
Versioning	—	—	—	State-based versioning	—	Change-based versioning

- *Change propagation.* As ADLs are used to model the configuration of system, thus most existing ADLs support top-down propagation from configuration level to runtime system, like xADL2.0 [26], [27], Darwin, C2. However, none of them support a bottom-up propagation.
- *Versioning.* Few ADLs enable to version architectures, except MAE supporting change-based versionning and SOFA2.0 [22], [23] supporting state-based versioning.

From the process management viewpoint, architecture-centric evolution processes in these ADLs are not explicitly defined. None of these ADLs has a formalized architecture-centric evolution process which can be used to follow and control the quality of software evolution.

In conclusion, the above studied ADLs supply a support to software architecture-centric evolution to some extent, however there are some common weak points in these works: (1) lacking a complete evolution process covering from evolution planning, test, implementation, propagation to its re-versioning and (2) missing a well defined and formalized architecture-centric evolution process.

B. Objectives

The objectives of this paper is to propose a formalized architecture-centric evolution process based on Dedal[2].

- Fully support the architecture-centric evolution activities identified in our proposed taxonomy.
- Formalize the process with SPEM2.0 standard to make it enable to follow and control the quality of software evolution. We choose SPEM2.0 to formalize our proposed process, because SPEM2.0 has a tool support Eclipse Process Framework (EPF) tool support. The formalized process can be easily used as a template for software maintainers.

IV. EVOLUTION PROCESS

In this section, we present an architecture-centric evolution process based on Dedal for component-based software (as shown in Fig. 1). Architecture evolution can be triggered from any representation level in Dedal. Moreover, both top-down (re-factoring) and bottom-up (re-engineering) evolution processes are supported. In a classical top-down evolution process, evolution operations at a representation level are controlled in order to enforce its conformance with upper (more abstract) representation levels. Conversely, changes are propagated to lower representation levels in order to update them and maintain their consistency to upper representation levels. Our approach allows bottom-up evolution too, in which transitional non-conform architectures can be created to experiment new solutions [28]. After a successful test, changes are committed and propagated to the other representation dimensions in order to restore consistency. Combined top-down and bottom-up evolution aims to address the issues of architecture mismatches [15].

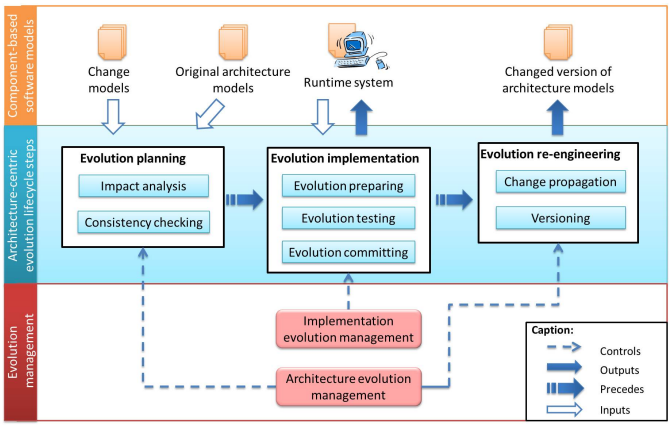

Fig. 1. Architecture centric evolution process for component-based software

The above evolution process is formalized in SPEM 2.0 [9][3]. The evolution process contains three phases:

1) *Evolution planning phase* to analyze the change impact and check its consistency in each abstraction level of software,
2) *Evolution implementation phase* to prepare, test the change and implement it in its implementation environment,

[2]The evolution process is based on component-based software development supported by Dedal. The details can be found in [5].

[3]The process template is also created as an evolution process library in EPF, which can find in http://www.irit.fr/ Yulin.Zhang/Dedal.html

3) *Evolution re-engineering phase* to propagate the change to *unchanged* levels and version software architectures if necessary.

This evolution process is controlled by evolution management which contains architecture evolution management module and implementation evolution management module to govern architecture models and implementation respectively.

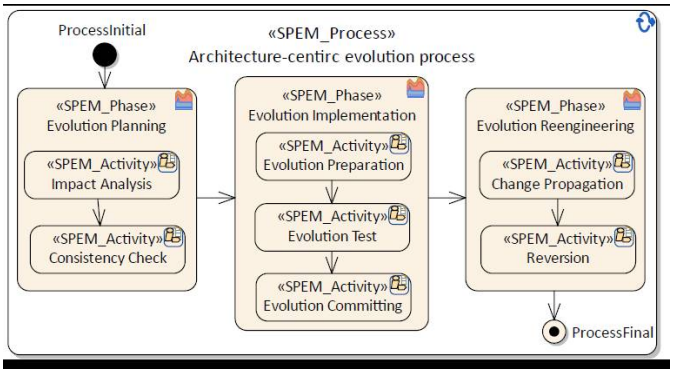

Fig. 2. Architecture centric evolution process based on SPEM 2.0

A. Evolution Planning Phase

Evolution planning is the first phase of software evolution, to decide how to apply the change. It is composed by two activities: *impact analysis* to analyze the change impact and *consistency check* to check the completeness and consistency of the target architecture as shown in Fig. 3.

1) Impact Analysis Activity: During the impact analysis, the architecture evolution management module produces the change lists from its input change request. A *change list* is the list of changes which should be performed by the evolution manager to modify the architectures. A change request can produce three change lists for architecture specification, architecture configuration and component assembly separately.

- *Vertical Analysis Task Definition.* Firstly, the request change is analyzed vertically and changes are generated for the architecture level in which the change request is performed.
- *Horizontal Analysis Task Definition.* Secondly, this change list is analysed horizontally to produce two propagated change lists for the other two levels. To propagate changes, we restraint the propagation only authorizes between the successive levels (see Fig. 4).

2) Consistency Checking Activity: Consistency is an internal property of an architecture model, which intends to ensure that different elements of that model do not contradict with one another [12]. The aim of consistency checking is to predict whether changes induce inconsistencies inside and among the three dimensions of a given architecture. We talk about intra-dimension consistency checks and inter-dimension consistency checks. If changes preserve consistency, the thought evolution will be permitted. If not, it will either be forbidden or trigger the derivation of a new architecture version for which consistency will be ensured.

Consistency check checks at specification and configuration (interface and behavior consistency checking), assembly

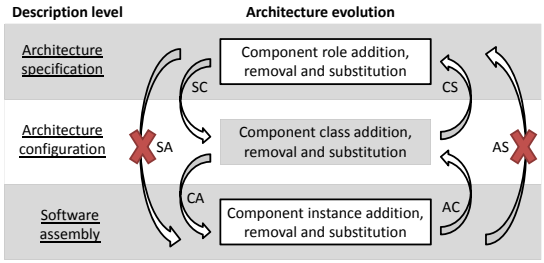

Fig. 4. The propagation relationship between the three architectural levels

(attribute consistency checking), among three levels (map consistency checking). Name and interaction inconsistencies reuse the definitions of Taylor [12].

- *Interface Inconsistency Task Definition.* Interface inconsistencies are detected using the component specialization rule that suits new component connection or component substitution. Interfaces consistency calculus can be automated as previously studied in Arévalo *et al.* [29], for example. These inconsistencies are searched for during intra-dimension consistency checks and inter-dimension consistency checks.
- *Behavior Inconsistency Task Definition.* Behavior inconsistency detection reuses the work of Plasil *et al.* [30] on various behavior protocol comparisons. These inconsistencies are searched for during intra-dimension consistency checks and inter-dimension consistency checks.
- *Attributes Inconsistency Task Definition.* Attributes inconsistencies are detected automatically using introspection capabilities on component classes and instances. These inconsistencies are searched for solely during inter-dimension consistency checks.
- *Mapping Inconsistency Task Definition.* Mapping inconsistencies occur between two successive levels of the description of level software architectures[4]. This is an inter-level consistency checking.

B. Evolution Implementation Phase

Evolution implementation phase aims to test evolution at runtime to assure the feasibility of changes for the running system. It can be considered as a sub evolution process based on runtime systems, called gradual evolution process [32], [28]. It is controlled by the implementation evolution manager. The main idea is to get the original system evolve into a target system through a transitional step. During the transitional step, a transitional assembly is produced to test the proposed changes in the real execution environment and either validate the evolution or invalidate it to return to the original state. There are three activities (shown in Fig. 5): evolution preparation, evolution test and evolution committing.

1) Evolution Preparation Activity: The objective of the evolution preparing step is to change the original system into the transitional system. The change description list in the assembly level is transformed into change transactions. *Change transactions* are the executive programs that operate on a

[4]The rules can be found in [31].

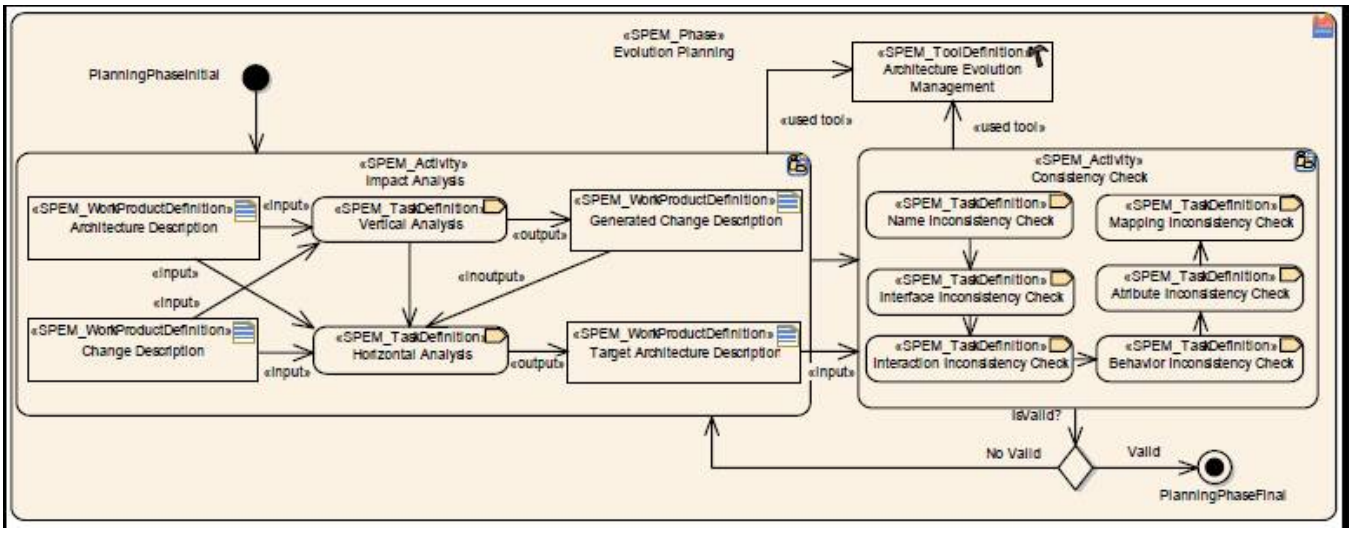

Fig. 3. The evolution planning phase

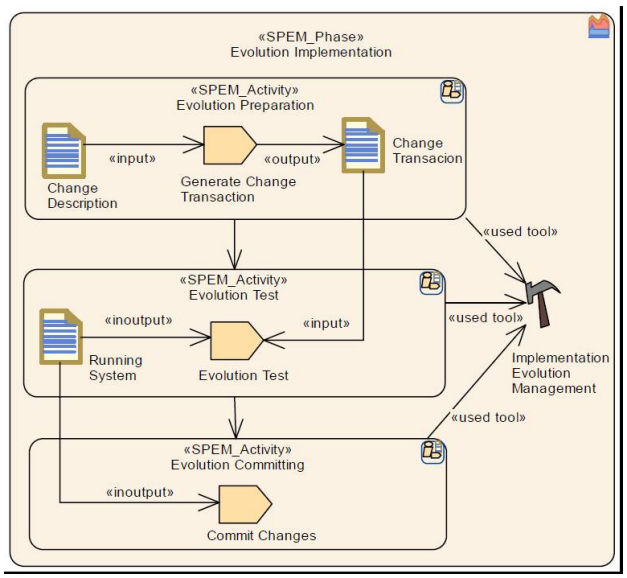

Fig. 5. The evolution implementation phase

system to make changes. It is composed of basic operations: deploy (deploy the component into runtime environment), mutate (mutate the component instances according to the assembly rules from the assembly level or directly from old existing component instance), add, delete, connect, disconnect.

2) Evolution Test Activity: Evolution test uses the connector-driven gradual assembly evolution process [32], [28]. One of the main ideas exposed in gradual evolution process is to have the original assembly of the system evolved into an objective assembly through a transitional assembly. The transitional assembly is transformed from the original assembly by merging changes into the assembly without deleting any system elements. For example, to replace one component with its new version, the transitional assembly makes two versions of the component co-existing and connecting in the system at the same time. The transitional assembly aims to test new component versions and either validate the evolution (commit changes) or invalidate it to rollback to the original state.

3) Evolution Committing Activity: If the changes are validated by the implementation evolution management module after evolution has been tested, changes will be committed in the running software system.

C. Evolution Re-engineering Phase

The evolution re-engineering involves modifying architecture descriptions, versioning the modified architectures, as shown in Fig. 6.

1) Change Propagation Activity: Architecture evolution manager module treats the thought evolution as being part of a new architecture version. This is solved using change propagation techniques [33]. The changes imply in each level of software architectures according impact analysis results. The new architecture version is derived and its content inferred so as to be consistent with the thought change. The information necessary to derive new versions on each of its levels is extracted from lower to higher levels.

2) Reversion Activity: In this activity, the modified architecture models are reversioned with new versionID and the changes applied to these models. These changes preserved in new versioned architecture models comprise the given, generated and propagated changes that represent the delta between two versions.

V. Conclusion

In this paper, we firstly define a taxonomy of architecture-centric evolution activities which can be used to evaluates different ADLs on how they support evolution process. Then based on this taxonomy, we select six representative ADLs to analyze their architecture-centric evolution process. However, there is merely an ADL that fully cover all the necessary architecture-centric evolution activities and has a formalized evolution process to ensure the quality of software evolution.

In order to control the quality of software evolution and prevent the declination of software quality after evolution, we propose a full architecture-centric evolution process which covers all the necessary evolution activities, based on Dedal,

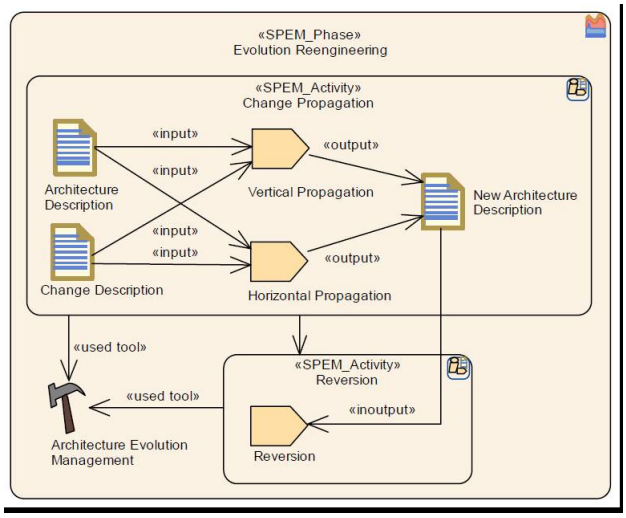

Fig. 6. The evolution Re-engineering phase

and furthermore we formalize the process using SPEM2.0 standards.

The process templates has been implemented in EPF (Eclipse) with SPEM2.0. However, the complete functional editor of Dedal is an independent editor, our objective is to immigrate Dedal as an eclipse editor, to make evolution process management and Dedal architecture evolution activities can be collaborated and synchronized automatically.

REFERENCES

[1] E. Iee, "IEEE Std 1061-1998," *IEEE Standard for a Software Quality Metrics Methodology*, 1998.

[2] I. S. . IEC 61508, *Functional safety of electrical/ electronic/programmable electronic safetyrelated systems*, International Electrotechnical Commission Std., 2000.

[3] P. Kruchten, *The rational unified process: an introduction*. Addison-Wesley Professional, 2004.

[4] H. Y. Zhang, "A multi-dimensional architecture description language for forward and reverse evolution of component-based software," Ph.D. dissertation, Montpellier Uiniversity II, 2010.

[5] H. Y. Zhang, L. Zhang, C. Urtado, S. Vauttier, and M. Huchard, "A three-level component model in component based software development," *SIGPLAN Not.*, vol. 48, no. 3, pp. 70–79, Sep. 2012. [Online]. Available: http://doi.acm.org/10.1145/2480361.2371412

[6] ISO/IEC, *ISO/IEC 9126. Software engineering – Product quality*. ISO/IEC, 2001.

[7] ISO/IEC, "ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models," Tech. Rep., 2010.

[8] M. B. Chrissis, M. Konrad, and S. Shrum, *CMMI Guidlines for Process Integration and Product Improvement*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[9] *Software & Systems Process Engineering Meta-Model Specification*, OMG, 2008.

[10] *Business Process Model and Notation (BPMN) Version 2.0*, OMG, 2011.

[11] Y. Zhang, B. Hamid, and D. Gouteux, "A metamodel for representing safety lifecycle development process," in *ICSEA 2011, The Sixth International Conference on Software Engineering Advances*, 2011, pp. 550–556.

[12] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons, January 2009.

[13] I. Sommerville, *Software Engineering*, 8th ed. Addison Wesley, 2006.

[14] I. Crnkovic and M. Larsson, Eds., *Building Reliable Component-Based Software Systems*. Artech House Publishers, 2002.

[15] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *ACM SIGSOFT Software Engineering Notes*, vol. 17, no. 4, pp. 40–52, 1992.

[16] P. Jamshidi, M. Ghafari, A. Ahmad, and C. Pahl, "A framework for classifying and comparing architecture-centric software evolution research," in *CSMR*, 2013, pp. 305–314.

[17] N. Medvidovic, D. S. Rosenblum, and R. N. Taylor, "A language and environment for architecture-based software development and evolution," in *Proceedings of the 21st International Conference on Software Engineering (ICSE'99)*, Los Angeles, CA, May 1999, pp. 44–53.

[18] N. Medvidovic, D. S. Rosenblum, , and R. N. Taylor, "A type theory for software architectures," Department of Information and Computer Science, University of California, Irvine, Tech. Rep. UCI-ICS-98-14, 1998.

[19] R. Allen, R. Douence, and D. Garlan, "Specifying and analyzing dynamic software architectures," in *Proceedings of the 1998 Conference on Fundamental Approaches to Software Engineering (FASE'98)*, Lisbon, Portugal, March 1998, pp. 21–37.

[20] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer, "Specifying distributed software architectures," in *Proceedings of the 5th European Software Engineering Conference*, Sitges, Spain, September 1995, pp. 137–153.

[21] J. Magee and J. Kramer, "Dynamic structure in software architectures," *SIGSOFT Softw. Eng. Notes*, vol. 21, no. 6, pp. 3–14, 1996.

[22] T. Bures, P. Hnetynka, and F. Plasil, "Sofa 2.0: Balancing advanced features in a hierarchical component model," in *SERA '06: Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications*. Seattle, USA: IEEE Computer Society, 2006, pp. 40–48.

[23] P. Hnetynka, F. Plasil, T. Bures, V. Mencl, and L. Kapova, "Sofa 2.0 metamodel," Dep. of SW Engineering, Charles University, Tech. Rep., December 2005.

[24] R. Roshandel, A. V. D. Hoek, M. Mikic-Rakic, and N. Medvidovic, "Mae–a system model and environment for managing architectural evolution," *ACM Transactions on Software Engineering and Methodology*, vol. 13, no. 2, pp. 240–276, 2004.

[25] A. van der Hoek, M. Mikic-Rakic, R. Roshandel, and N. Medvidovic, "Taming architectural evolution," *SIGSOFT Softw. Eng. Notes*, vol. 26, no. 5, pp. 1–10, 2001.

[26] E. M. Dashofy, A. van der Hoek, and R. N. Taylor, "A comprehensive approach for the development of modular software architecture description languages," *ACM Trans. Softw. Eng. Methodol.*, vol. 14, no. 2, pp. 199–245, 2005.

[27] E. M. Dashofy, A. V. der Hoek, and R. N. Taylor, "A highly-extensible, xml-based architecture description language," in *WICSA '01: Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01)*, Washington, DC, USA, 2001, pp. 103–112.

[28] H. Y. Zhang, C. Urtado, and S. Vauttier, "Connector-driven process for the gradual evolution of component-based software," in *Proceedings of the 20th Australian Software Engineering Conference (ASWEC2009)*, Gold Coast, Australia, April 2009.

[29] G. Arévalo, N. Desnos, M. Huchard, C. Urtado, and S. Vauttier, "Formal concept analysis-based service classification to dynamically build efficient software component directories," *International Journal of General Systems*, vol. 38, no. 4, pp. 427–453, May 2009.

[30] F. Plasil and S. Visnovsky, "Behavior protocols for software components," *IEEE Trans. Softw. Eng.*, vol. 28, no. 11, pp. 1056–1076, 2002.

[31] H. Y. Zhang, C. Urtado, and S. Vauttier, "Architecture-centric component-based development needs a three-level ADL," in *Proceedings of the 4th European Conference on Software Architecture*, ser. LNCS, M. A. Babar and I. Gorton, Eds., vol. 6285. Copenhagen, Denmark: Springer, August 2010, pp. 295–310.

[32] ——, "Connector-driven gradual and dynamic software assembly evolution," in *Proceedings of the International Conference on Innovation in Software Engineering (ISE08)*, Vienna, Austria, December 2008.

[33] J. Rumbaugh, "Controlling propagation of operations using attributes on relations," in *Proc. of the Int. Conf. on Object-Oriented Programming Systems, Languages and Applications*, 1988, pp. 285–296.