New Trends in Software Methodologies, Tools and Techniques
H. Fujita and R. Revetria (Eds.)
IOS Press, 2012

doi:10.3233/978-1-61499-125-0-73

Generic Process Framework for Developing High-Integrity Software

Binazir BIGLARI[1] and Raman RAMSIN
Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

Abstract. In high-integrity systems, certain quality requirements have gained utmost significance in such a way that failing to satisfy them at a particular level may result in the loss of the entire system, endangerment of human life, peril to the organization's existence, or serious damage to the environment. High-integrity computer systems should incorporate top-quality software in order to adequately address their stringent quality requirements. The methodologies used for developing high-integrity software must possess special characteristics in order to ensure successful realization of the requirements.
Software Process patterns represent empirically proven methods of software development that can be exploited as reusable chunks to produce bespoke methodologies, tailored to fit specific project situations and requirements. The authors provide a set of process patterns extracted from methodologies and standards which are specifically intended for developing high-integrity systems. The methodologies and standards which were used as resources for extracting these patterns were selected based on their history of successful application. The patterns have been organized into a generic High Integrity Software Development Process (HISDP); this process framework can be instantiated by method engineers to produce tailored-to-fit methodologies for developing high-integrity software.

Keywords. high-integrity system, process pattern, software development methodology, process framework

Introduction

Software is used for governing a wide variety of systems, including medical equipment, air traffic control systems, and nuclear plants; the failure of these systems may have different outcomes, thus indicating their level of criticality. High-integrity computer systems are critical systems in which addressing certain quality requirements is of utmost importance, as their failure may have dire consequences. Therefore, the development of high-integrity systems requires the application of stringent quality-assurance measures. Software is an integral part of high-integrity computer systems, and is closely linked to other parts; it should therefore be adaptable, so that it maintains its integrity at a desirable level when other parts have to be changed due to deterioration or the emergence of new technologies. The identification of system and software requirements is an important factor in quality assurance; other factors, such as the need for new technologies, the importance of the system's mission, and project size, can also affect the required level of quality assurance, albeit to a lesser degree [1].

[1] Corresponding Author: Binazir Biglari, Department of Computer Engineering, Sharif University of Technology, Azadi Ave., Tehran, Iran; E-mail: biglari@ce.sharif.edu.

High integrity systems are divided into three main categories, according to the classification proposed in [2]: safety-critical, mission-critical, and business-critical. The highest level of criticality is attributed to cases where human life is at stake, thus requiring the strictest of standards and activities. Other levels of severity are taken into account depending on the degree of seriousness of the errors made when implementing system functions, and the consequences of breakdown in terms of the damage inflicted.

There are numerous methodologies for developing high-integrity software, and they have been in use for a long time. Because of the importance of nonfunctional requirements in these systems, the dependency of software on non-software components, and the severe consequences of software failures, the software development processes used for constructing these systems must possess certain characteristics such as reliability, traceability to requirements, consistency, and production of special intermediate products. In most cases, compliance with specific standards (military standards such as MIL-STD-498, or domain-specific standards such as DO 278B) is required [1, 3].

Due to the diversity of the methods involved, there exists no general methodology for developing high-integrity systems. However, there are frameworks for this purpose, such as those presented in [4], which deal with the essentials and can be used as standards. Although the development of these systems is possible with conventional methodologies, it is highly preferable to use specialized software development processes which ensure that the final product is of acceptable quality.

Successful *solutions* to recurring *problems* in a given *context* have long been captured as "patterns". A software pattern is an abstraction of a proven solution for a common problem in the context of software development. Software *process* patterns have emerged through the abstraction of recurring software development process solutions. They were first introduced by Coplien [5], who focused on organizational and managerial processes. The notion was later refined by Ambler, who provided a more precise definition aiming at software development processes [6]. A process pattern can be employed in all aspects of software development: From a high-level viewpoint which accentuates the general approach to software development and the lifecycle employed, to a specific view of a particular part of the software development process. In addition, different levels of granularity have been defined for process patterns; according to Ambler, process patterns are of three types, in descending order of granularity: phase, stage, and task [6, 7]. Tasks are the key components, whereas phases and stages are important for organizing and using the tasks effectively. Phase process patterns refer to the high-level activities in a software development project, usually executed sequentially. Stage process patterns include the tasks related to a particular stage of the software development process, and can in turn consist of finer-grained stages. Stages are usually performed iteratively. Task process patterns refer to the details of the steps that should be performed in a fine-grained stage.

Process patterns can constitute software development methodologies, especially in the context of Situational Method Engineering (SME), where a methodology is produced from scratch in accordance with situational requirements, or a preexisting methodology is extended by adding process chunks based on past defects and new requirements [8, 9]. Process patterns can thus be used as method chunks. An example is the OPEN Process Framework (OPF), which is based on a library of reusable components, many of which are process patterns [10, 11]. Two other examples are the sets of process patterns suggested in [12] and [13] for Web engineering and component-based software development, respectively.

In this paper, the authors propose a set of process patterns for developing high-integrity software systems, extracted from the methodologies and standards related to this domain. The patterns have been organized into a generic process framework for producing high-integrity software systems (called herein as: High-Integrity Software Development Process—HISDP). This framework and its constituent patterns can be employed to produce and evaluate processes for developing high-integrity software.

The rest of this paper is organized as follows: Section 1 provides a brief review of ten methodologies and two standards which have been used as sources for extracting process patterns; Section 2 contains the general framework proposed (HISDP); the proposed process patterns are briefly described in Section 3; Section 4 deals with the validation of the patterns by showing how they are mapped to the source processes; and Section 5 contains the conclusions as well as recommendations for future research.

1. Pattern Sources: High-integrity Software Development Methodologies and Standards

From the multitude of methodologies and standards that were studied, ten methodologies and two standards (NIST and MIL-STD-498) were used for extracting the relevant process patterns. The selected methodologies include: ASPECS, Extended MaSE, HOOD, PBSE, XFun, AOM, MASTER, BUCS, AUP, and Agile+. These methodologies were selected from among those studied based on the following criteria:

- Availability of adequate resources on the specifications of the methodology;
- Existence of reports on the practical usage of the methodology;
- Support for different paradigms, including agile, agent-oriented, object-oriented, aspect-oriented, and model-driven development;
- Adequate coverage of the generic software development lifecycle; and
- Relevance to the high-integrity domain and adequate support for its three criticality levels (safety, mission, and business).

A brief overview of these methodologies and standards will be provided throughout the rest of this section.

ASPECS is an agent-oriented methodology which relies on *holonic* organizational metamodels and which is based on the PASSI methodology [14]. All stages are carried out seamlessly and smoothly by focusing on agents. The process includes four phases: system requirements analysis, agent society design, implementation, and deployment.

The original version of MaSE was a general purpose methodology for developing homogenous multiagent systems [15]. In order to adapt this methodology to embedded and real-time contexts, an extended version was provided which supports requirements engineering, system environment analysis, and time-dimensional modeling of the agents' behavior [16]. This method covers three phases: requirements engineering, analysis, and design.

The HOOD methodology was developed by the European Space Agency to support architectural and detailed design of high-integrity, real-time systems [8,3]. It is an iterative top-down design method [17], suitable for developing large systems with a long lifespan in which reusability, reliability, and maintainability are essential [18].

The PBSE methodology aims to help develop embedded systems, or their building blocks, by using formal methods [19]. PBSE was employed in project ASSERT, also conducted by the European Space Agency [20]. By giving precedence to requirements

elicitation, PBSE provides a formal and appropriate description of the problem which can be used as a reliable basis for quality assurance.

Due to the need for higher levels of accuracy and reliability on the one hand, and the mutability of user requirements on the other, formal variants of Agile development methodologies have emerged. XFun is a prominent example: XFun is the result of adapting UP and combining it with the X-Machine formal method [21].

Based on aspect-oriented modeling (AOM) techniques, Georg et al. have proposed an aspect-oriented design method for developing high-integrity applications with strict security requirements [22].

The MASTER methodology is a model-driven approach developed as part of a European information project of the same name. This methodology includes a process and a set of systems engineering methods to adapt the process to customer requirements [23, 24]. The process consists of eight phases, spanning requirements capture to deployment, and provides prescribed model transformation methods.

BUCS was a research project initiated by the Norwegian Research Council and Norwegian University of Science and Technology in order to study the methods of component-based development and also the development, support, and maintenance of business-critical software. The important characteristics and cases specific to this field were extracted in order to extend existing methodologies to make them suitable for developing business-critical software [25]. A special variant of RUP was produced for this purpose which incorporates specialized hazard analysis methods [26].

The AUP methodology was introduced as a simplified agile version of RUP. AUP has been successfully used for developing high-integrity software, such as banking systems [27] and online reservation systems [28].

Agile+ is an agile methodology inspired by XP which has been used by various companies for developing large high-integrity systems [29, 30]. The Agile+ process provides support for dynamic and variable requirements [31].

The American National Institute of Standards and Technology (NIST) has provided certain quality assurance guidelines for the safety systems used in nuclear plants [1]. It has also proposed a framework for developing and assuring the quality of critical software [4], in which the requirements and characteristics of a high-integrity system are defined and guidelines are provided for developers, testers, and researchers.

The military standard MIL-STD-498 is an American military standard which outlines the prerequisites to software development and documentation for high-integrity systems. This standard provides general and detailed requirements for the processes utilized and the documents produced [32].

2. Proposed Process Framework for High Integrity Software Development

On the basis of the processes and standards studied, a generic process framework has been proposed for developing high-integrity software, a detailed description of which will be provided in this section. This framework, which we have chosen to call High-Integrity Software Development Process (HISDP), provides a high-level organization for finer-grained process patterns and highlights the position of *software* development in the overall *systems* development process (as shown in Figure 1). HISDP helps method engineers choose from among the process patterns and combine them based on the project requirements. HISDP consists of seven phase process patterns: initiation, requirements, design, coding and integration, installation, maintenance, and death. The

seven phases are performed sequentially, but their constituent stages are typically performed iteratively.

The process begins with the Initiation phase which provides the necessary infrastructure for developing software successfully by performing feasibility study, determining preliminary estimates of time and cost, and producing an overall plan. In the Requirements phase, software requirements are identified and documented with special attention to traceability. In the Design phase, the design of the software is produced at different levels of detail on the basis of the requirements. The Coding and Integration phase incorporates development activities such as coding, testing, and integrating software increments. In the Installation phase, the software produced is deployed in the user environment and integrated with non-software components of the system; software tuning is typically necessary at this stage. In the Maintenance phase, previous phases are iterated to make corrections, or to address the changes occurred in user requirements or in non-software components. In the Death phase, reusable items are extracted and the lessons learned from the project are documented for use in future projects.

Management activities are extremely important in the development of high-integrity systems; hence, the umbrella activities emphasized in methodologies and standards have been explicitly considered in HISDP. Vital management activities are typically captured in stage- or task process patterns of the framework; nevertheless, the arrow below Figure 1 lists them individually.

3. Proposed HISDP Process Patterns

This section provides detailed descriptions for HISDP's constituent process patterns (as shown in Figure 2). Only the stages that are necessary in high-integrity software development (shown as dark boxes in Figure 2) have been described in detail, each in a separate subsection.

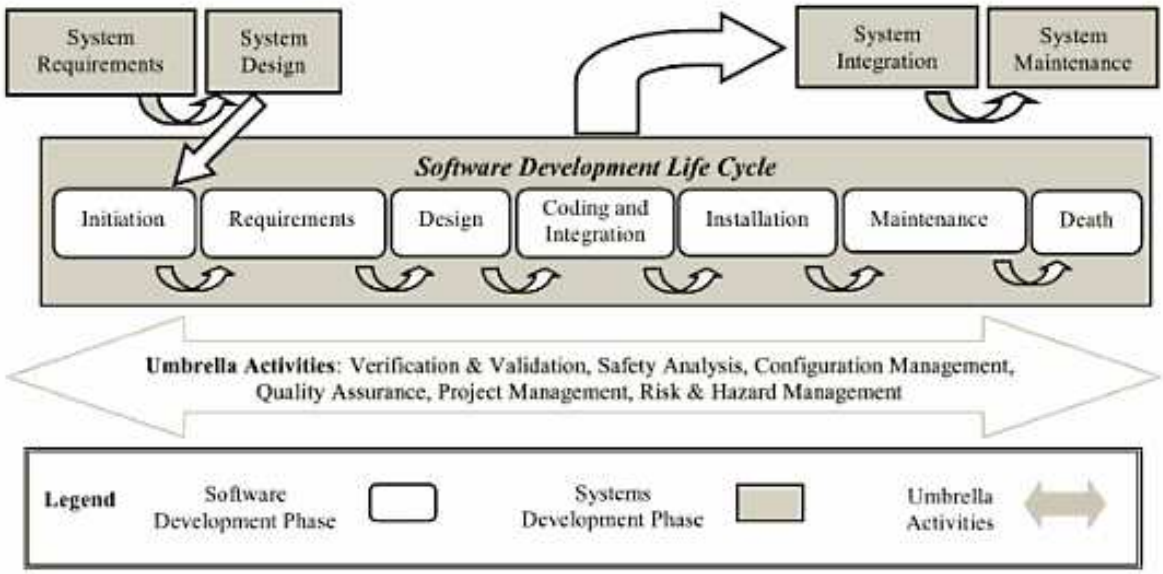

Figure 1. High-Integrity Software Development Process (HISDP) and its position in the systems development process

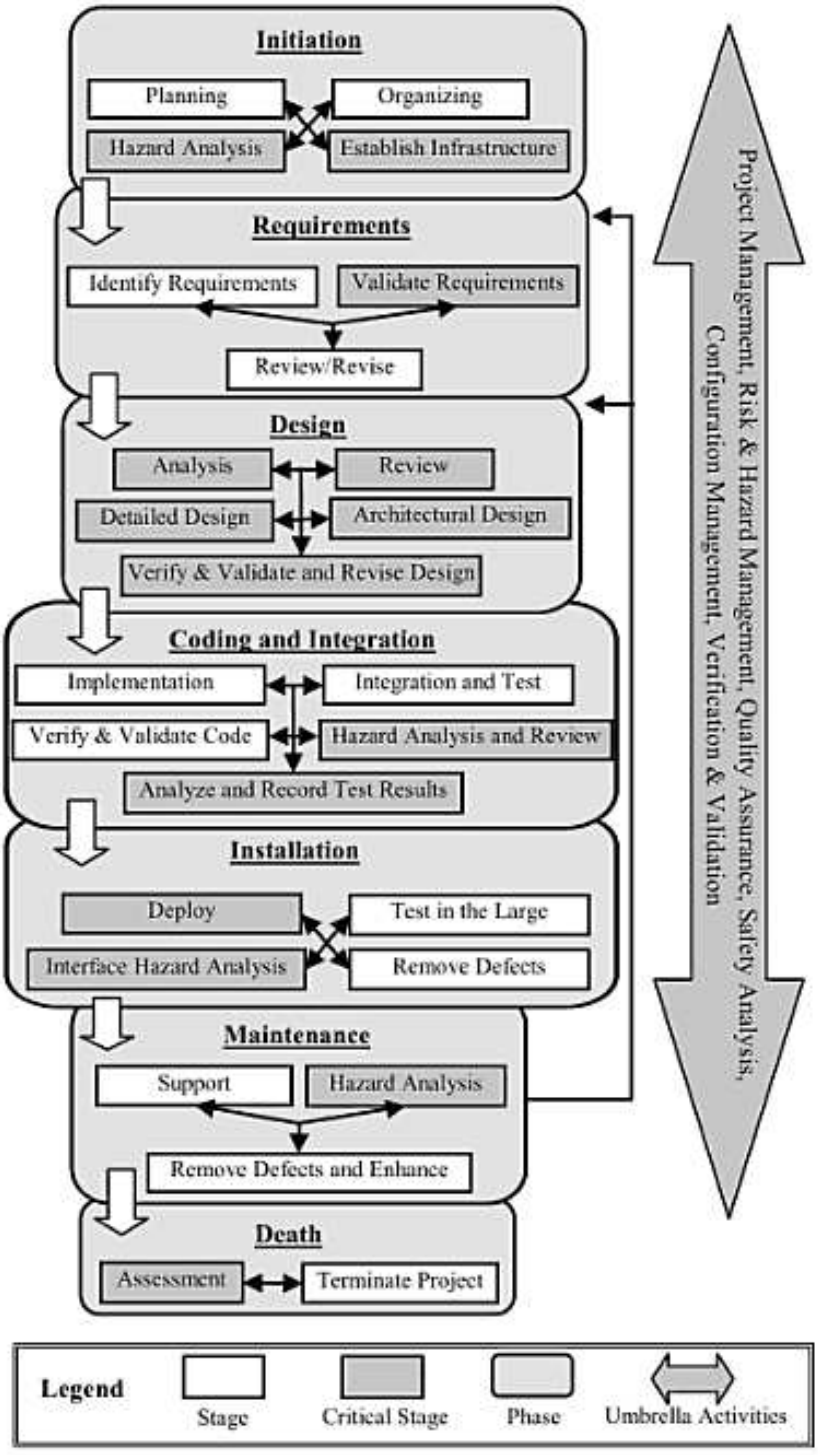

Figure 2. HISDP and its constituent Stage process patterns

3.1. Establish Infrastructure (Initiation Phase)

In this stage, individuals are recruited according to organizational positions, and project teams are formed (Figure 3). Decisions on employment, including staffing plans and training policies, are documented. Feasibility study is also performed, along with analyzing and providing different approaches to guiding the project, predicting project scope and possible undesirable incidents, ascertaining assumptions, making plans, and foreseeing the results of possible solutions for each action. A software development

process is selected and the relevant standards are determined. Process selection includes the selection of methods, tools, and techniques for developing and testing the output products, as well as the provision of project management support. Decisions on how to reuse preexisting infrastructures are also made in this stage.

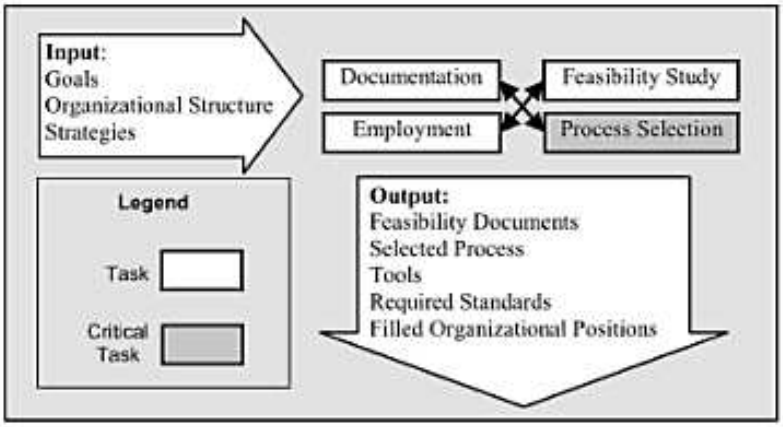

Figure 3. Establish Infrastructure (stage process pattern)

3.2. Hazard Analysis (Initiation Phase)

In this stage (Figure 4), software criticality requirements (such as safety) are identified and the critical components – as well as failure outcomes – are determined, based on which appropriate plans and methods are determined in the direction of eliminating or acceptably reducing the identified hazards. This stage delivers the software safety plan, test plans, and a system description in which hazards are taken into account.

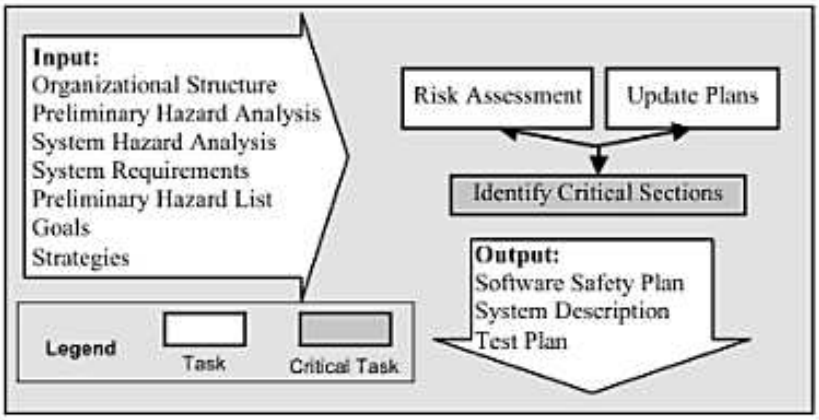

Figure 4. Hazard Analysis – Initiation (stage process pattern)

3.3. Validate Requirements (Requirements Phase)

It is essential to specify explicit measures in order to assess the degree to which software requirements and system goals can be relied upon. This is performed by the Validate Requirements stage (Figure 5), which determines the system requirements that are delegated to software. Software requirements are then checked for understandability, accuracy, testability, consistency, completeness, and all other qualitative properties that have been defined according to requirements standards and testing strategies. It is essential to make sure that critical requirements are well-defined.

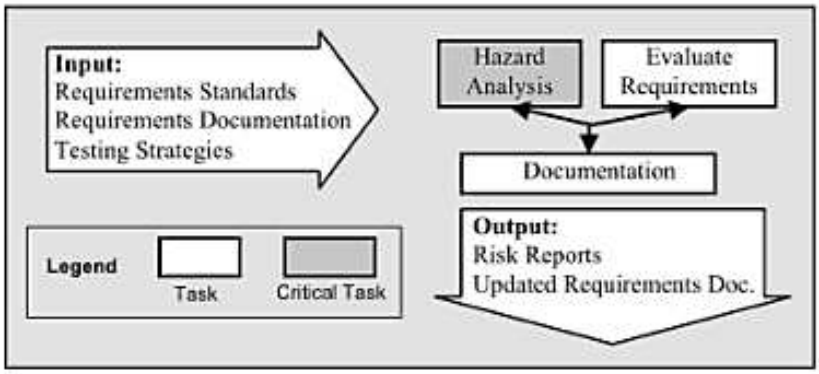

Figure 5. Validate Requirements (stage process pattern)

3.4. Analysis (Design Phase)

The aim of this stage is to analyze, understand, and model the problem domain based on the requirements (Figure 6). The boundaries of the software part are determined, and its interfaces with non-software parts are evaluated for precision, completeness, consistency, and accuracy. Software requirements are assigned to the interfaces, including interfaces to other systems, other software, and human users. Requirements are also assigned to architectural components. Software requirements undergo hazard analysis as well; if hazards are not at an acceptable level, the above activities are repeated. The user manual is updated on the basis of detailed requirements.

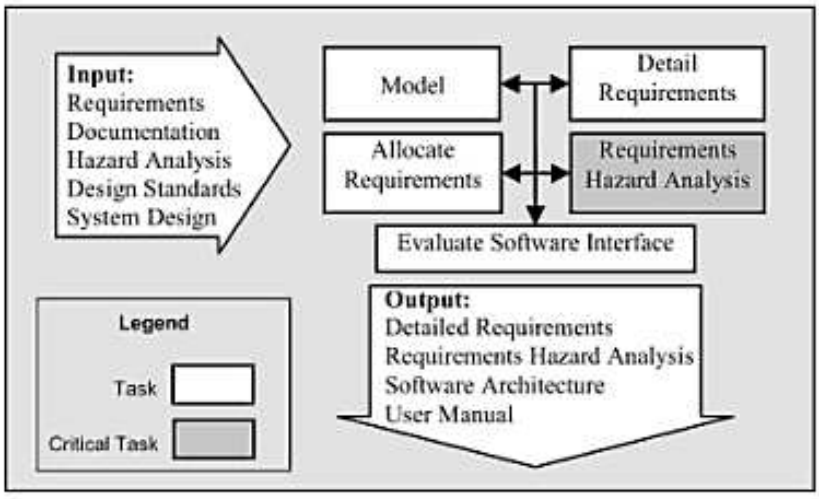

Figure 6. Analysis (stage process pattern)

3.5. Architectural Design (Design Phase)

This stage includes architectural design activities in which non-functional requirements and constraints are applied on the design while preserving traceability (Figure 7). Since high-integrity software is usually a part of a larger system, a logical architecture is first defined, which is then followed by determining the physical architecture considering the constraints, preexisting components, and relationships with the system design. Therefore, one important activity in this stage is the selection and use of preexisting architectural components.

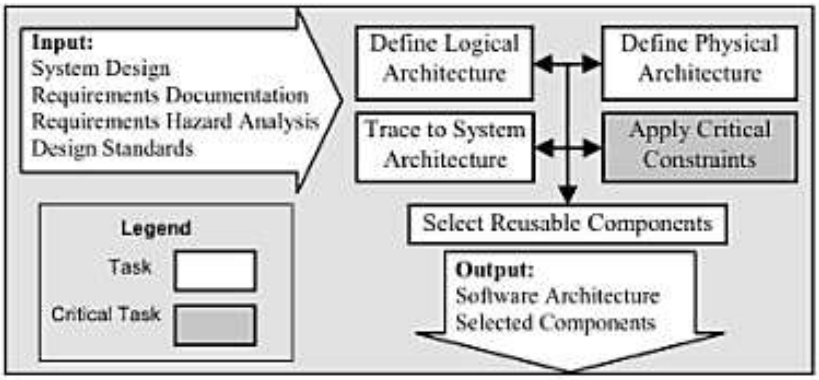

Figure 7. Architectural Design (stage process pattern)

3.6. Detailed Design (Design Phase)

This stage uses the analysis models and architectural design to develop the detailed design (Figure 8). Each component is decomposed into finer-grained constituents in order to develop the detailed design of the code components. For each component, external and internal interfaces are described and modeled, and general and domain-specific design patterns are applied. Components related to safety, security, or other criticality requirements are determined with special attention to software-hardware co-design. Formal methods are used for determining how accurately the design meets the requirements, with the results carefully recorded. Tracking mechanisms are implemented by establishing the relationships among requirements, design, and documentation. An integration test plan is produced according to the standards and goals, and test cases, test procedures, and test data are prepared.

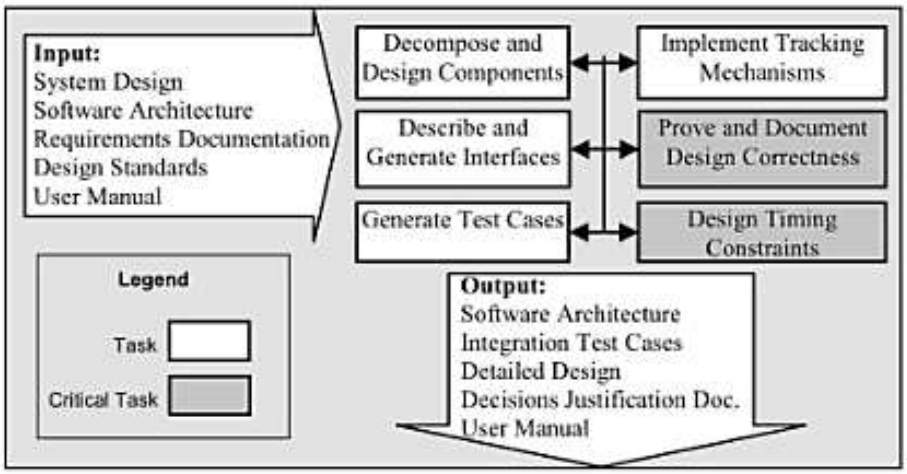

Figure 8. Detailed Design (stage process pattern)

3.7. Verify & Validate and Revise Design (Design Phase)

In this stage, design documents are verified and validated (Figure 9). If the identified hazards and risks are not at a desirable level, the design must be changed along with the integration test plan. Software design is evaluated and analyzed for understandability, accuracy, testability, consistency, completeness, and other properties defined in the requirements process. In addition, solution accuracy is verified by considering feasibility conditions, analysis results, and decision justifications. The traceability of

the software design to software requirements is verified. Certain measures are applied to assess whether the requirements and qualitative properties have been realized in the software design. Risks are identified and mitigated by performing static analysis and hazard analysis. The critical software components and the test program are also scrutinized for problems. Interfaces are analyzed for precision, completeness, consistency, and accuracy of design. Changes are made to the design in order to resolve the problems and address the hazards that have not been properly mitigated or controlled. Changes are accordingly made to the software integration test plan.

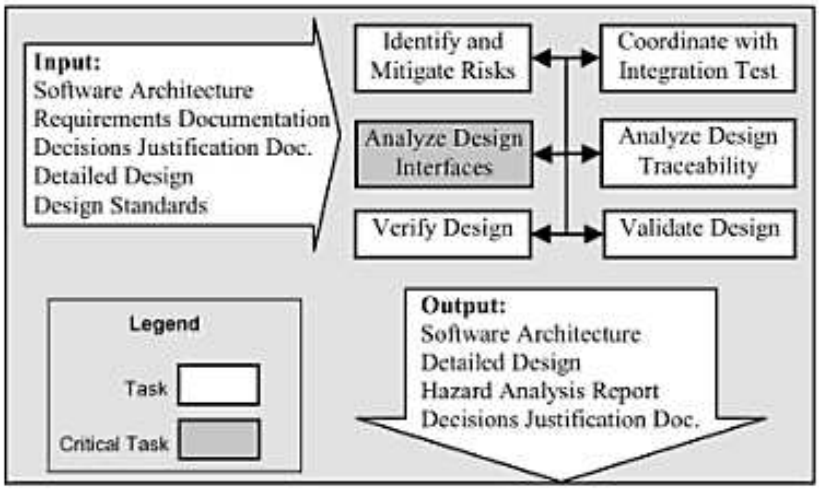

Figure 9. Verify & Validate, and Revise Design (stage process pattern)

3.8. Review (Design Phase)

Considering the results of the previous stage, project plans are reviewed in this stage; also, methods and standards are improved on the basis of the problems reported (Figure 10). Based on the results of quality assurance, software verification and validation, and software hazard analysis, the necessary changes are applied to the software development process and its outputs. Unit tests are also planned. The standards of coding and design are selected or improved, as well as the activities, methods, and tools. Major problems are reported and fed back to the design process. If necessary, the requirements descriptions, user manual, and software development plan are modified.

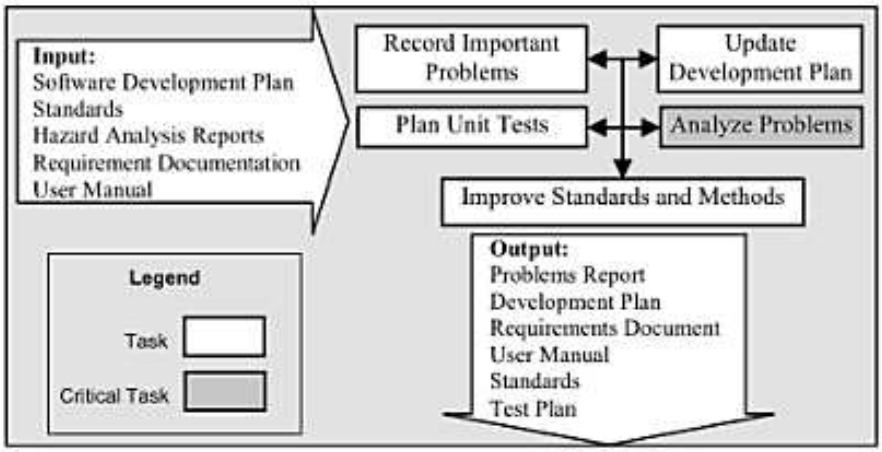

Figure 10. Review (stage process pattern)

3.9. Analyze and Record Test Results (Coding and Integration Phase)

In this stage, analysis is performed on the basis of the results of the Verify and Validate Code stage, based on which a part of the Verify and Validate Code stage may be re-executed (Figure 11). The aim of this stage is to raise the code to the quality level indicated in the quality assurance plan. Code is evaluated based on qualitative properties, and the relevant documentation (such as the user manual and the comments added to the code) is evaluated to assess completeness, consistency, and correctness. Code interface analysis (including evaluation based on hardware-, user-, and software interfaces) is conducted to ensure the precision, completeness, consistency, and correctness of the code produced. Coverage of unit tests is assessed as part of the analysis of test results. Completed test cases are reviewed, and if necessary, re-executed; this trend continues until quality reaches the level indicated in the software quality assurance plan and test plan. The results obtained are rigorously recorded.

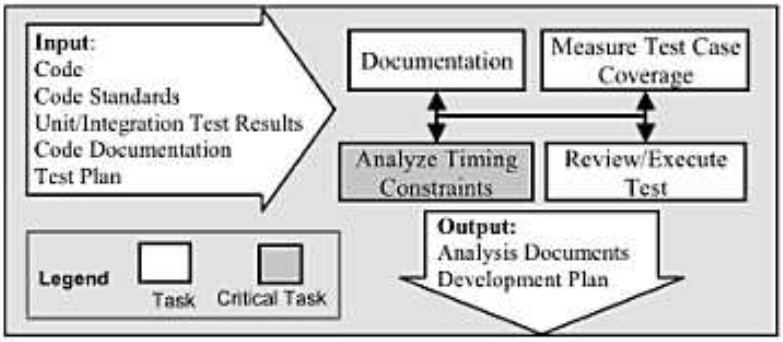

Figure 11. Analyze and Record Test Results (stage process pattern)

3.10. Hazard Analysis and Review (Coding and Integration Phase)

In this stage, hazard analysis is carried out on the basis of the results of unit and integration tests, resulting in modifications to the process or products. This analysis includes code hazard analysis and software safety tests (Figure 12). To perform code-level software hazard analysis, the code, system interfaces, and software documentations are analyzed in order to ensure that they meet the requirements; also, recommendations are offered to make changes to the design, code, and tests. For software safety testing, components which are critical in terms of safety are tested under normal and abnormal conditions of inputs and environment. Testing will be iterated under the same conditions after applying corrective measures.

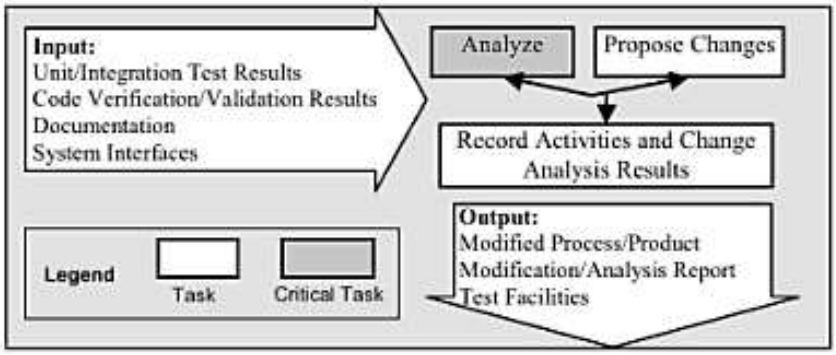

Figure 12. Hazard Analysis and Review (stage process pattern)

Based on the results of quality assurance, software verification and validation and software hazard analysis, necessary changes are applied to the software development process and its products. Results are fed back to the design process.

3.11. Deploy (Installation Phase)

In this stage, the integrated software is deployed after the required infrastructures have been prepared (Figure 13). To install the software, it is necessary to first prepare the destination environment. The deployment method should be specified and described in a diagram. If a deployment plan does not already exist, it will also be developed, and the feasibility of software installation is evaluated. The deployment plan will be reviewed and revised to ensure correctness and completeness. A plan regarding the maintenance activities is developed after the system has been installed. Manuals are produced/updated for all users and operators (in particular, the end users and the maintenance and support teams), and. The software support and maintenance manuals are completed, with special attention to maintenance standards. Training of users and maintenance/support personnel is also conducted at this stage.

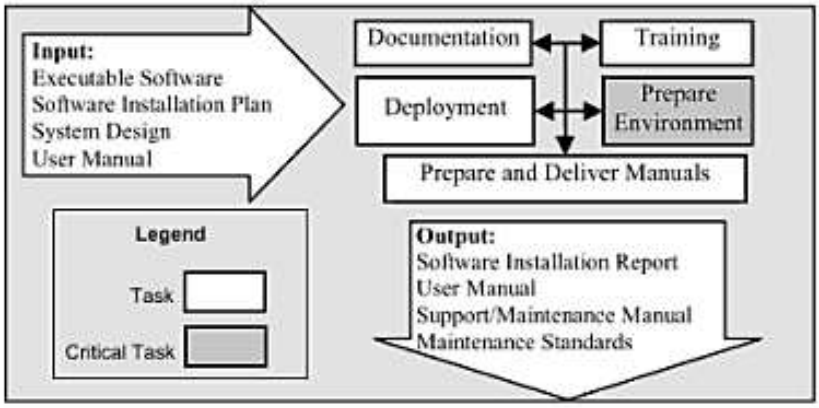

Figure 13. Deploy (stage process pattern)

3.12. Interface Hazard Analysis (Installation Phase)

In this stage, software hazards are reevaluated and mitigated based on the overall interface of the software (Figure 14). The software interface is tested along with non-software parts, in line with the general testing strategies. Interface hazard analysis manages hazards that have not been eliminated or controlled in the design phase.

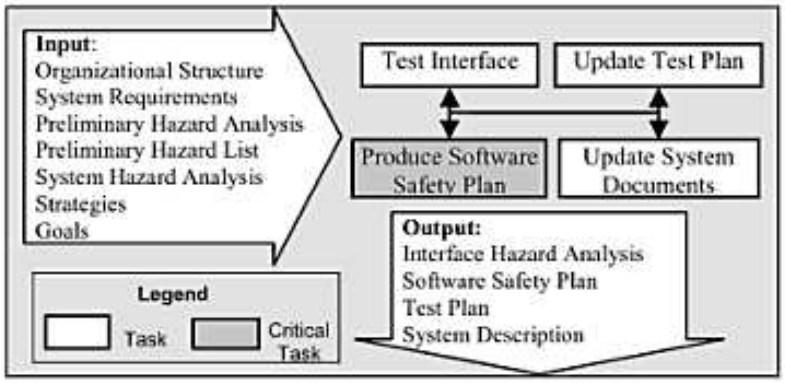

Figure 14. Interface Hazard Analysis (stage process pattern)

Testing activities encompass developing and recording test cases, test procedures, and test data, as well as test execution and analysis of test results. By applying modifications to the design, which help identify hazards, methods are suggested for recovering from the situations caused by hazards. The software safety plan is adapted, test plans are updated, and system documentation and design are modified based on the results of interface evaluations.

3.13. Hazard Analysis (Maintenance Phase)

This stage evaluates the modifications made during maintenance along with their effects, and analyzes and manages the hazards caused by these modifications (Figure 15); in addition, quality assurance processes are specified, and new quality assurance plans are produced accordingly. All the changes applied to the software should be analyzed in order to determine their effects on safety and other critical properties. For each change, hazards and test results are analyzed to ensure that modification have created no new hazards and have had no exacerbating effect on existing hazards. Changes to software requirements are also analyzed. If hazard management is required, quality assurance activities are planned and executed accordingly.

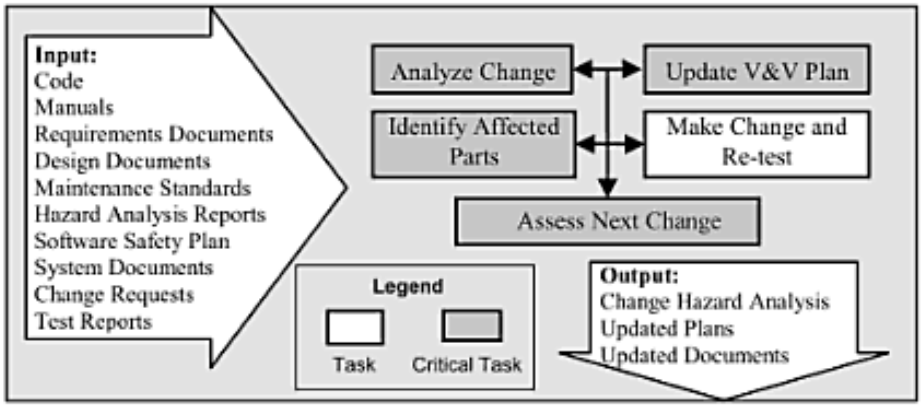

Figure 15. Hazard Analysis – Maintenance (stage process pattern)

3.14. Assessment (Death Phase)

This stage reviews the project plans and management documentation on project progress and measurement of the quality of processes and products, in order to collect and record the significant cases as the lessons learned, to be used in future projects (Figure 16). All project entities are studied, including test reports, hazard analysis reports, standards, change documents, manuals, software, methods, tools, and the personnel/roles involved. Management documents such as the project management plan, configuration management plan, quality assurance plan, and organizational documents are also important resources. Successful/unsuccessful experiences and any reusable assets are extracted and documented. The following can be mentioned as examples of significant quality assurance cases that should be considered for documentation: project control deviations, significant user feedback, reports on the capabilities of software vendors, and reports on the compliance of the process and products with standards and plans. It should be noted that inefficient methods and techniques must be recorded as unsuccessful experiences that should be avoided in the future. The management methods applied and the strategic decisions made are also

significant in this context; examples include communication/coordination mechanisms, and decisions on recruitment, including employment plans and training policies.

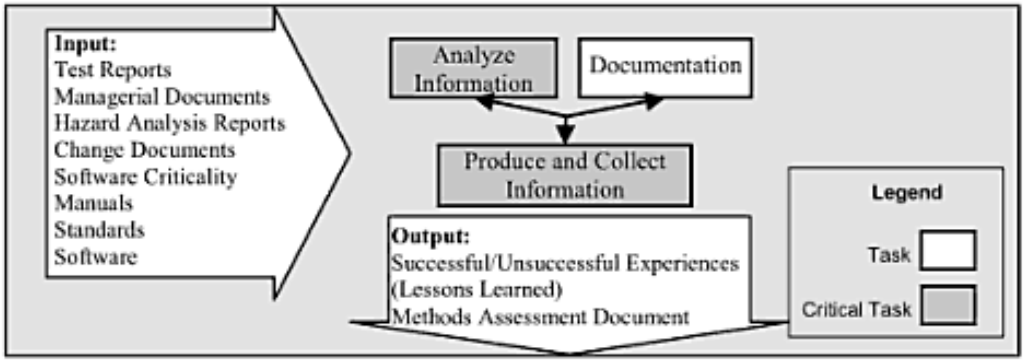

Figure 16. Assessment (stage process pattern)

4. Mapping of Proposed Process Patterns to Source Methodologies

Completeness and proper coverage of the proposed process patterns needs to be evaluated in order to show that these process patterns adequately cover the phases of the source methodologies. Correspondence of the proposed process patterns to the main methodologies used as pattern sources is shown in Table 1. Comparison suggests that the proposed framework and patterns do indeed cover the activities of high-integrity software development; in other words, it demonstrates that these methodologies can be engineered by using the proposed process patterns and framework. Moreover, it shows how the phases of these methodologies have been used as sources for eliciting the proposed process patterns. Thus, it can be deduced that the proposed framework is valid, although new patterns can be added to further enrich it.

5. Conclusions and Future Work

We propose a generic process framework along with a set of process patterns to develop high integrity software systems. To produce these patterns, we have selected prominent methodologies from the most commonly used processes of this domain, and have extracted their common sub-processes as process patterns. These patterns have been organized into a generic process framework which can be instantiated to yield specialized processes for developing high-integrity software.

This research can be further specialized for each of the three types of high-integrity systems. Task process patterns can be defined so that the use of the framework in SME projects is facilitated. Also, essential patterns can be mapped to critical contexts so that the selection of method chunks is further enhanced. Research can also focus on providing an expansion framework to tailor existing patterns for use in critical contexts.

ACKNOWLEDGMENT

We wish to thank the Iranian Research Institute for Information and Communication Technology for sponsoring this research.

Table 1. Mapping of Proposed Process Patterns to Source Methodologies

Methodology	Methodology Phase	Corresponding Process Patterns	
		Phase	*Stages*
ASPECS	System Requirements Analysis	2	Requirements Identification, Requirements Validation, Review/Revise
		3	Analysis
	Agent Society Design	3	Architectural Design, Detailed Design, Verify & Validate and Revise Design, Review
	Implementation	4	Implementation, Verify and Validate Code, Integration and Test, Analyze and Record Test Results
	Deployment	4	Integration and Test, Analyze and Record Test Results
		5	Deploy, Test in the Large, Interface Hazard Analysis
MaSE	Requirements Engineering	2	Requirements Identification, Requirements Validation, Review/Revise
	Analysis	3	Analysis
	Design	3	Architectural Design, Detailed Design, Verify & Validate and Revise Design, Review
		5	Deploy
HOOD	Requirements Analysis	3	Analysis
	Design	3	Architectural Design, Detailed Design, Verify & Validate and Revise Design, Review
	Implementation	4	Implementation, Hazard Analysis and Review, Integration and Test
	Test	4	Verify and Validate Code, Integration and Test
PBSE	Requirements Capture	2	Requirements Identification, Requirements Validation
		3	Analysis
	System Design and Validation	3	Architectural Design, Detailed Design, Verify & Validate and Revise Design
	Feasibility and Dimensioning	3	Verify & Validate and Revise Design
		4	Hazard Analysis and Review
	Integration Testing	4	Integration and Test, Analyze and Record Test Results
XFUN	Planning	1	Planning, Hazard Analysis
	Requirements	2	Requirements Identification, Requirements Validation, Review/Revise
	Design	3	Architectural Design, Detailed Design, Verify&Validate and Revise Design
	Implementation & Test	4	Implementation, Verify and Validate Code, Hazard Analysis and Review, Integration and Test, Analyze and Record Test Results
	Deployment	5	Deploy
		6	Remove Defects and Enhance
AOM	Design	3	Architectural Design, Analysis, Detailed Design, Verify & Validate and Revise Design, Review
MASTER	Capture User Requirements	2	Requirements Identification
	PIM Context Definition	2	Requirements Validation, Review/Revise
	PIM Requirements Specification	3	Analysis
	PIM Analysis	3	Architectural Design, Analysis, Detailed Design
	Design	3	Architectural Design, Detailed Design, Verify&Validate and Revise Design
	Coding & Integration	4	Implementation, Verify and Validate Code, Hazard Analysis and Review, Integration and Test
	Testing	3	Verify & Validate and Revise Design
		4	Verify and Validate Code, Hazard Analysis and Review, Integration and Test, Analyze and Record Test Results
		5	Test in Large, Remove Defects
	Deployment	5	Deploy, Test in the Large
		6	Remove Defects and Enhance
AUP	Inception	1	Planning, Organizing, Establish Infrastructure
		2	Requirements Identification, Requirements Validation, Review/Revise
		3	Architectural Design
	Elaboration	3	Architectural Design, Verify & Validate and Revise Design, Review
	Construction	3	Detailed Design, Verify & Validate and Revise Design, Review
		4	Implementation, Verify and Validate Code, Hazard Analysis and Review, Integration and Test, Analyze and Record Test Results
		5	Deploy, Test in the Large
	Transition	5	Deploy, Test in the Large, Remove Defects
		6	Support, Remove Defects and Enhance
		7	Assessment

REFERENCES

[1] D. Wallace, L. Ippolito, and D. Kuhn, *High Integrity Software Standards and Guidelines*, National Institute of Standards and Technology (NIST Special Publication 500-204), 1992.

[2] J. Rushby, Critical System Properties: Survey and Taxonomy, *Reliability Eng. and System Safety* **43** (1994), 189–219.

[3] D. Ström, *Purposes of Software Architecture Design and How They Are Supported by Software Architecture Design Methods*, Master's Thesis, Blekinge Institute of Technology, Sweden, 2005.

[4] D. Wallace, L. Ippolito, and D. Kuhn, *High Integrity Software Standards and Guidelines*, National Institute of Standards and Technology (NIST Special Publication 500-223), 1994.

[5] J. Coplien, A Generative Development Process Pattern Language, In: *Pattern Languages of Program Design*, ACM Press/Addison-Wesley, 1995, 187–196.

[6] S. Ambler, *Process Patterns: Building Large-Scale Systems Using Object Technology*, Cambridge University Press, 1998.

[7] S. Ambler, *More Process Patterns: Delivering Large-Scale Systems Using Object Technology*, Cambridge University Press, 1999.

[8] J. Ralyté, S. Brinkkemper, and B. Henderson-Sellers (Eds.), *Situational Method Engineering: Fundamentals and Experiences*, Springer, 2007.

[9] J. Ralyté, R. Deneckere, and C. Rolland, Towards a generic model for situational method engineering, In: *Proc. CAiSE'03* (2003), 95–110.

[10] B. Henderson-Sellers, Method Engineering for OO Systems Development, *CACM* **46** (2003), 73–78.

[11] D. Firesmith and B. Henderson-Sellers, *The OPEN Process Framework: An Introduction*, Addison-Wesley, 2001.

[12] R. Babanezhad and R. Ramsin, Process Patterns for Web Engineering, In: *Proc. COMPSAC'10*, 2010, 477–486.

[13] E. Kouroshfar, H. Yaghoubi Shahir, and R. Ramsin, Process Patterns for Component-Based Software Development, In: *Proc. CBSE'09*, 2009, 54–68.

[14] M. Cossentino, N. Gaud, V. Hilaire, S. Galland, and A. Koukam, ASPECS: An agent-oriented software process for engineering complex systems, *J. Autonom. Agents and Multiagent Sys.* **20** (2010), 260–304.

[15] S.A. Deloach, M.F. Wood, and C.H. Sparkman, Multiagent Systems, *Int'l J. Software Eng. and Knowledge Eng.* **11** (2001), 231–258.

[16] I. Badr, H. Mubarak, and P. Göhner, Extending the MaSE Methodology for the Development of Embedded Real-Time Systems, *Lecture Notes in Computer Science* **5118** (2008), 106–122.

[17] J. Rosen, *HOOD: An Industrial Approach to Software Design*, HOOD Technical Group, 1997.

[18] *Software Engineering and Standardisation- HOOD*, European Space Agency, 2006.

[19] F. Hesling, A. Schyn, R. Sezestre, and J.F. Tilman, *Engineering with AADL*, 2005, Available online at: http://aadl.sei.cmu.edu/aadlinfosite/LinkedDocuments/d2_1000_PBSE_with_AADL.pdf.

[20] *Software Engineering and Standardisation- Proven by design: Computer systems for aerospace applications*, European Space Agency, 2008.

[21] G. Eleftherakis and A. Cowling, An Agile Formal Development Methodology, In: *Proc. SEEFM'03*, 2003, 36–47.

[22] G. Georg, I. Ray, K. Anastasakis, B. Bordbar, M. Toahchoodee, and S. Houmb, An Aspect-Oriented Methodology for Designing Secure Applications, *Inform. and Software Technology* **51** (2009), 846–864.

[23] X. Larrucea, A. Diez, and J. Mansell, Practical Model Driven Development Process, In: *Proc. MDA'04*, 2004, 99–108.

[24] X. Larrucea, A. Diez, and A. Belen, Process Engineering and Project Management for the Model Driven Approach, In: *Proc. MDA-IA'04*, 2004, 63–69.

[25] O. Vindegg, *BUCS Implementing Safety: An Approach as to How to Implement Safety Concerns*, Master's Thesis, Norwegian University of Science and Technology, 2006.

[26] T. Hermansen, *Creating more Reliable Business Critical Enterprise Systems Using RUP and System Safety*, Depth Study, Department of Computer and Information Science, NTNU, 2005.

[27] I. Christou, S. Ponis, and E. Palaiologou, Using the Agile Unified Process in Banking, *Software* **27** (2010), 72–79.

[28] S. Finch, M. Bukowy, L. Wilder, and D. Nunn, Agile Software Development at Sabre Holdings, In: *Software Engineering: Evolution and Emerging Technologies*, IOS Press, 2005, 27–38.

[29] D. Opperthauser, Defect Management in an Agile Development Environment, *J. Defense Software Eng.* **16** (2003), 21–24.

[30] J. Dutton and R. McCabe, *Agile/Lean Development and CMMI*, SEPG, 2006.

[31] *Developing Business-Critical Software: Methodology*, AgileTek, 2011.

[32] *MIL-STD-498: Military Standard for Software Development and Documentation*, US-DoD, 1994.