ACSIJ Advances in Computer Science: an International Journal, Vol. 2, Issue 5, No.6 , November 2013
ISSN : 2322-5157
www.ACSIJ.org

Software Architecture Viewpoint Models: A Short Survey

Valiallah Omrani[1], Seyyed Ali Razavi Ebrahimi[2]

[1] **Computer Science & Information Technology, Payam Noor University (PNU), Tehran, Iran**
omrani@inio.ac.ir

[2] **Computer Science & Information Technology, Payam Noor University (PNU), Tehran, Iran**
ali_razavi@pnu.ac.ir

Abstract

A software architecture is a complex entity that cannot be described in a simple one-dimensional fashion. The architecture views used to describe software provide the architect with a means of explaining the architecture to stakeholders. Each view presents different aspects of the system that fulfill functional and non-functional requirements. A view of a system is a representation of the system from the perspective of a viewpoint. Architecture viewpoints in software products provide guidelines to describe uniformly the total system and its subsystems. It defines the stakeholders whose concerns are reflected in the viewpoint and the guidelines, principles, and template models for constructing its views. The results of this study may serve as a roadmap to the software developers and architects in helping them select the appropriate viewpoint model based on the stakeholders and concerns that need to be covered by views.

Keywords: *software architecture, view, viewpoint, architectural description, stakeholder, viewpoint model.*

1. Introduction

With the growing complexity and size of software-intensive systems, software architecture has become increasingly important [1]. Understanding all aspects of complex systems (people, building constructions, IT systems, etc.) completely at all times is not possible at least for human perception. It would also be impractical to attempt to do this, because not all aspects of a system are relevant all of the time. It therefore makes sense to be able to look at only those aspects of a system that are of interest at a given time. For IT systems, the concept of architecture views and viewpoints exist for this purpose [2].

Multiple software architecture views are essential because of the diverse set of stakeholders (users, acquirers, developers, testers, maintainers, inter-operators, and others) needing to understand and use the architecture from their viewpoint. Achieving consistency among such views is one of the most challenging and difficult problems in the software architecture field [3].

A number of case studies and theories based on practical experience have been published, suggesting the need for multiple architectural views to capture different aspects of a software architecture [4]. The effectiveness of having multiple architectural views is that the multiple views help developers manage complexity of software systems by separating their different aspects into separate views [5].

As part of researches on the evolvability of large software intensive systems [6], we observed that suitable architectural views are indispensable assets to improve and sustain the evolvability of systems. Such views help practitioners to understand the existing system, to plan and evaluate intended changes, and to communicate them to others efficiently [7].

The architecture views used to describe software provide the architect with a means of explaining the architecture to stakeholders [8]. Each view presents different aspects of the system that fulfill functional and non-functional requirements [9].

Architecture viewpoints in software products provide guidelines to describe uniformly the total system and its subsystems [5]. A viewpoint is a collection of patterns, templates, and conventions for constructing one type of view. It defines the stakeholders whose concerns are reflected in the viewpoint and the guidelines, principles, and template models for constructing its views. In other words, a viewpoint defines the aims, intended audience, and content of a class of views and defines the concerns that views of this class will address [10].

In this paper, we discuss the various existing architectural viewpoint models. The remainder of the article is structured as follows: Section 2 describes the key concepts used in the context of the architectural viewpoint models. Section 3 discusses various existing viewpoint models. Finally the paper is concluded in section 4.

2. Key Concepts

One of the problems encountered when we talk about architecture for software systems is that the terminology has been loosely borrowed from other disciplines and is widely used, inconsistently, in a variety of situations. This section defines and reviews some of the key concepts that underpin the discussion in the remainder of the paper.

- *Software Architecture:* Conscious architectural thinking in software development has only been around for a few decades. This is why there are still contradictory opinions on what exactly architecture means. There are numerous definitions of the term “architecture” in IT [3]. This shows that it is a challenge to find one definition that is recognized universally. But it’s always worth getting the latest perspective from some of the leading thinkers in the field. It is the definition of software architecture according to Bass et al. [11]: “The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both”.

- *Stakeholder:* The people affected by a software system are not limited to those who use it. Software systems are not just used: They have to be built and tested, they have to be operated, they may have to be repaired, they are usually enhanced, and of course they have to be paid for. Each of these activities involves a number—possibly a significant number—of people in addition to the users. We refer collectively to these people as stakeholders. A commonly used definition of stakeholder is: “A stakeholder in a software architecture is a person, group, or entity with an interest in or concerns about the realization of the architecture” [12].

- *Architectural View and Viewpoint:* A software architecture is a complex entity that cannot be described in a simple one-dimensional fashion [3]. A view of a system is a representation of the system from the perspective of a viewpoint. Formally, a view is a representation of a whole system from the perspective of a set of concerns [12]. This definition of a view clearly show the most important property of architecture views: they are motivated by stakeholders of a system (“…a set of concerns… ”). An architectural view is a way to portray those aspects or elements of the architecture that are relevant to the concerns the view intends to address—and, by implication, the stakeholders for whom those concerns are important. Viewpoint is a systems engineering concept that describes a partitioning of concerns in system restricted to a particular set of concerns. A viewpoint is a collection of patterns, templates, and conventions for constructing one type of view. It defines the stakeholders whose concerns are reflected in the viewpoint and the guidelines, principles, and template models for constructing its views. Architectural viewpoints provide a framework for capturing reusable architectural knowledge that can be used to guide the creation of a particular type of (partial) architectural descriptions [10].

- *Architectural Description:* ISO/IEC/IEEE has defined a standard for the architectural description (AD) of software-intensive systems. It includes a conceptual framework to support the description of architectures, and the required content of an architectural description: “Software architecture description is a set of practices for expressing, communicating and analyzing software architectures (also called architectural rendering); AD is the result of applying such practices: a work product expressing a software architecture” [13]. In addition to above definition, Rozanski and Woods [10] present the following definition of AD: “An architectural description (AD) is a set of products that documents an architecture in a way its stakeholders can understand and demonstrates that the architecture has met their concerns”. “Products” in this context consists of a range of things—particularly architectural models, but also scope definition, constraints, and principles.

2.1 Interrelationships between the Key Concepts

The important relationships between key concepts are illustrated in the UML diagram in fig. 1. The diagram brings out the following relationships between the concepts we have discussed so far:

- A system is built to address the needs, concerns, goals, and objectives of its stakeholders.
- The architecture of a system is comprised of a number of architectural elements and their interelement relationships.

- The architecture of a system can potentially be documented by an AD (fully, partly, or not at all).
- An AD documents an architecture for its stakeholders and demonstrates to them that it has met their needs.
- A viewpoint defines the aims, intended audience, and content of a class of views and defines the concerns that views of this class will address.
- A view conforms to a viewpoint and so communicates the resolution of a number of concerns.
- An AD comprises a number of views.

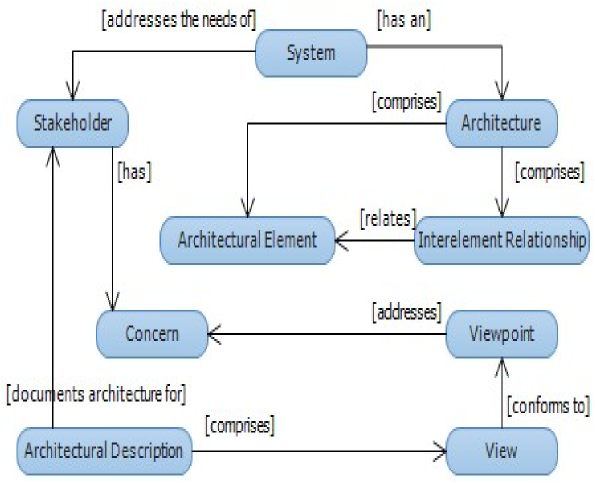

Fig. 1. Interrelationships between key concepts

2.2 The Benefits of Using Views and Viewpoints

Using views and viewpoints to describe the architecture of a system benefits the architecture definition process in a number of ways [2]:

- Separation of concerns: Describing many aspects of the system via a single representation can cloud communication and, more seriously, can result in independent aspects of the system becoming intertwined in the model. Separating different models of a system into distinct (but related) descriptions helps the design, analysis, and communication processes by allowing you to focus on each aspect separately.
- Communication with stakeholder groups: Different stakeholder groups can be guided quickly to different parts of the AD based on their particular concerns, and each view can be presented using language and notation appropriate to the knowledge, expertise, and concerns of the intended readership.

Management of complexity: By treating each significant aspect of a system separately, the architect can focus on each in turn and so help conquer the complexity resulting from their combination.

3. Viewpoint Models

Architecture can take place at different levels. It is therefore important to always be clear about the level we are dealing with. This is the only way of applying useful means and disciplines for the architecture level in question. The levels possible range from organizations to systems all the way down to individual building blocks. At each level, we can take different architecture views of a system. In their entirety, the views give a complementary image of the architecture to be implemented. Architecture view models enable us to look at architectures systematically and in a way that reduces their complexity for this purpose. They group relevant views from which architectures are to be considered into one model, thus enabling them to be shown in their entirety [2]. In this section we briefly describe a number of useful viewpoints models.

3.1 Zachman Framework

The Zachman Framework [14, 15] is an architecture framework whose architecture view model can be seen as the father of the common architecture view models today. The Zachman Framework first describes an organization abstractly to then show the “implementation” of the organization step by step. As a result of its generic structure, the Zachman Framework has also proven itself to be suitable for describing IT architectures across organizations. In its current structural level, the Zachman Framework recognizes consists of a two dimensional classification matrix based on the intersection of six communication questions (What, Where, When, Why, Who and How) with six levels of reification, successively transforming the abstract ideas on the Scope level into concrete instantiations of those ideas at the Operations level. In the form of a matrix, architecture views and view aspects are the core of the architecture view model.

The Zachman Framework, as a domain-independent and technology-independent architecture framework, can be used as the basis for an architecture for any type of system. Due to its orientation on aspects that apply across an entire organization, this framework is ideal for enterprise architectures [2].

Before we look more closely at the individual architecture views of the Zachman Framework, we should first explain the view aspects orthogonal to the architecture views:

- What: Describes the data;
- How: Describes the functionality;
- Where: Describes the Network;
- Who: Describes the persons with reference to an organization;
- When: Describes performance-relevant time or event dependencies between the resources of an organization;
- Why: Describes the organizational objectives and their subjects;

Table 1 shows the six architecture views of the Zachman Framework.

Table 1. Architecture views in the Zachman framework

Views	**Definition**
Context	This architecture view is concerned with the basic requirements and is the basis for estimations with regard to the cost, scope, and functionality of a system.
Business	This architecture view shows all of the business entities and processes.
System	This view determines the data and functions that realize the business model.
Technology	This architecture view is concerned with the technological implementation of a system.
Integration	This architecture view looks at deployment aspects and the configuration management of a system.
Runtime	This architecture view covers the operation of a system within an organization.

3.2 Kruchten "4+1"

In 1995, Philippe Kruchten [16, 17] published a very influential paper in which he described the concept of architecture comprising separate structures and advised concentrating on four. To validate that the structures were not in conflict with each other and together did in fact describe a system meeting its requirements, Kruchten advised using key use cases as a check. This so-called "Four Plus One" approach became popular and has now been institutionalized as the conceptual basis of the Rational Unified Process [11]. Table 2 outlines the "4+1" viewpoints.

Table 2. Architecture views in the Kruchten viewpoint catalog

Views	**Definition**
Logical	The object model of the design when an object-oriented design method is used.
Process	Captures the concurrency and synchronization aspects of the design.
Physical	Describes the mapping(s) of the software onto the hardware and reflects its distributed aspect.
Development	Describes the static organization of the software in its development environment.

These four views are combined by using another view called use case view that illustrates the four views using use cases, or scenarios. The use case view helps developers to understand the other views and provides a means of reasoning about architectural decisions.

This viewpoint set appears to be the oldest viewpoint set and is widely known, discussed and supported (partially due to its inclusion in the RUP "architectural profile"). The set is simple, logical and easy to explain. We found that colleagues, clients and stakeholders understood the set with very little explanation. But the viewpoint set does not explicitly address data or operational concerns. Both of these aspects of a large information system are important enough to warrant their own view (and more importantly guidance relating to these aspects of developing an architecture needs to be captured somewhere).

3.3 SEI Viewpoints (Views and Beyond)

Views and Beyond (V & B) is a collection of techniques that carry out an underlying philosophy. The philosophy is that an architecture document should be helpful to the people who depend on it to do their work (far from least of which is the architect). The techniques can be bundled into a few categories:

1. Finding out what stakeholders need.

2. Providing the information to satisfy those needs by recording design decisions according to a variety of views, plus the beyond-view information.

3. Checking the resulting documentation to see if it satisfied the needs.

4. Packaging the information in a useful form to its stakeholders.

While items 3 and 4 denote document-centric activities, items 1 and 2 denote activities that should be carried out in conjunction with performing the architecture design. V & B comprises three views, which are then specialized by a set of associated architectural styles for each one, as shown in Table 3. The SEI Viewpoints are defined in Clements et al. [3].

Table 3. Architecture views in the SEI viewpoint catalog

Views	Definition
Module	Enumerates the principal implementation units, or modules, of a system, together with the relations among these units. Following styles are defined for the Module view type: - Uses: for capturing inter-module usage dependencies; - Generalization: for capturing commonality and variation (inheritance) relationships between modules - Decomposition: for specifying how modules are composed from simpler elements - Layered: for specifying how modules are arranged in layers according to their level of abstraction
Component and Connector	A Component and Connector view shows elements that have some runtime presence. The following styles defined for this view type all relate to commonly occurring runtime system organizations: - Pipe-and-Filter - Shared-Data - Publish-Subscribe - Client-Server - Peer-to-Peer - Communicating-Processes
Allocation	Presents a mapping between software elements (from either a module view or a component-and-connector view) and non-software elements in the software's environment. The following styles are defined for this view type: - Deployment: for specifying how software elements are mapped to elements of the deployment environment - Implementation: for specifying how software modules are mapped to the development environment - Work Assignment: for mapping software modules to those responsible for creating, testing, and deploying them

3.4 Garland and Anthony

This viewpoint set is much larger than the others; each viewpoint has a narrower scope. The advantage of this is that each view is clearly focused, has a manageable size, and plays an obvious role. The disadvantage is that it is harder to manage the problems of fragmentation in the AD and cross view consistency. Table 4 shows these viewpoints. These viewpoints are defined in Garland and Anthony [18].

Table 4. Architecture views in the Garland and Anthony viewpoint catalog

Views	Definition
Analysis Focused	Illustrates how the elements of the system work together in response to a functional usage scenario
Analysis Interaction	Presents the interaction diagram used during problem analysis
Analysis Overall	Consolidates the contents of the Analysis Focused view into a single model
Component	Defines the system's architecturally significant components and their connections
Component Interaction	Illustrates how the components interact in order to make the system work
Component State	Presents the state model(s) for a component or set of closely related components
Context	Defines the context within which the system exists, in terms of external actors and their interactions with the system
Deployment	Shows how software components are mapped to hardware entities in order to be executed
Layered Subsystem	Illustrates the subsystems to be implemented and the layers in the software design structure
Logical Data	Presents the logical view of the architecturally significant data structure
Physical Data	Presents the physical view of the architecturally significant data structure
Process	Defines the runtime concurrency structure
Process State	Presents the state transition model for the system's processes
Subsystem Interface Dependency	Defines the dependencies that exist between subsystems and the interfaces of other subsystems

The viewpoints are quite thoroughly defined, with purpose, applicability, stakeholder interest, models to use, modeling scalability and advice on creating the views all presented. In most cases there is also guidance provided that often includes potential problems to be aware of. Nevertheless, there are a lot of viewpoints in the set (14) and so the set can be quite unwieldy to explain and use. Moreover, Many of the viewpoints are relevant to a large or complex system, and so there appears to be a real danger of the architectural description becoming fragmented. We take Garland and Anthony's point that you should only apply the viewpoints relevant to a particular system, but you should do this when applying any viewpoint set, and we feel that for many systems you will end up with quite a few viewpoints when using this set.

3.5 Rozanski and Woods

In 2005, Nick Rozanski and Eoin Woods [19] wrote a very useful book in which they prescribed a useful set of six viewpoints (in the ISO 42010 sense) to be used in documenting Software architectures. The seven viewpoints [10], based on an extension of the Kruchten 4+1 set, are shown in Table 5.

Table 5. Architecture Views in the Rozanski and Woods Viewpoint Catalog

Viewpoints	Definition
Functional	Documents the system's functional elements, their responsibilities, interfaces, and primary interactions
Information	Documents the way that the architecture stores, manipulates, manages, and distributes information
Concurrency	Describes the concurrency structure of the system and maps functional elements to concurrency units to clearly identify the parts of the system that can execute concurrently and how this is coordinated and controlled
Development	Describes the architecture that supports the software development process.
Deployment	Describes the environment into which the system will be deployed, including capturing the dependencies the system has on its runtime environment
Operational	Describes how the system will be operated, administered, and supported when it is running in its production environment.
Context	This describes the relationships, dependencies, and interactions between the system and its environment (the people, systems, and external entities with which it interacts).

3.6 The Open Group Architecture Framework (TOGAF)

The Open Group Architecture Framework (TOGAF) was developed by The Open Group [20] based on the Technical Architecture Framework for Information Management (TAFIM) of the United States Department of Defense. It has been available on the market since 1995. TOGAF comprises a method (Architecture Development Method: ADM), a framework for defining the structural content of architecture (Architecture Content Framework: ACF), as well as tools, reference models, and taxonomies. Numerous best practices, principles, guidelines, and technologies also play a part [2].

Through the ACF, TOGAF provides numerous recommendations, guidelines, procedures, and classifications for creating and using viewpoints and architecture views. It adapts the ISO/IEC 42010:2007 standard and also recommends this standard for creating viewpoints and architecture views. In the ACF, TOGAF defines different viewpoints for developing architecture views for enterprise architecture. It also defines architecture views for IT systems [2]. These architecture views are described in table 6.

Table 6. Architecture views in TOGAF

Views	Definition
Business Architecture	This view is concerned with aspects of the system user. The aim is to achieve a comprehensive understanding of the functional requirements.
Enterprise Security	Covers typical questions regarding security (access protection, handling of threats, etc.)
Software Engineering	This architecture view provides guidelines for developing software systems.
System Engineering	In this architecture view the focus is on the distribution Of the software building blocks to the hardware building blocks and on models for their interaction.
Communication Engineering	Supports the planning and design of networks with regard to infrastructure (e.g., LAN) and communication (e.g., OSI)
Data Flow	Covers aspects around modeling and the processing of persistent data.
Enterprise Manageability	This architecture view is concerned with the aspects operation, administration, and management of IT systems.
Acquirer	Provides requirements, guidelines, and procedures for acquiring commercial off-the-shelf (COTS) building blocks.

3.7 ISO/IEC/IEEE 42010:2011

ISO/IEC 42010 is the ISO standard, Systems and software engineering—Architecture description. This standard replaces IEEE 1471:2000. ISO 42010 is centered on two key ideas: a conceptual framework for architecture description and a statement of what information must be found in any ISO 42010-compliant architecture description. ISO 42010 defines a view as a "work product representing a system from the perspective of architecture-related concerns" [13]. This standard defines viewpoint as a work product establishing the conventions for the construction, interpretation, and use of architecture views and associated architecture models [3]. Although this international standard does not require any particular viewpoints to be used, There should be a viewpoint for each view. Each view should have a viewpoint explaining the conventions being used in that view.

The template consists of a set of slots or information items. Each slot is identified by a name followed by a brief description of its intended content, guidance for developing that content, and in some cases “sub slots”. Not every slot is needed for documenting every viewpoint. This template is based on one proposed in [21]. These architecture views are described in table 7.

Table 7. Architecture views in ISO/IEC/IEEE 42010-2011

Viewpoints	Definition
Name	The name for the viewpoint, and any synonyms for the viewpoint.
Overview	An abstract or brief overview of the viewpoint and its key features.
Concerns and anti-concerns Typical stakeholders	A listing of the architecture-related concerns framed by this viewpoint. It can be useful to document the kinds of issues a viewpoint is not appropriate for. Articulating anti-concerns may be a good antidote for certain overused notations. A listing of the system stakeholders expected to be users or audiences for views prepared using this viewpoint.
Model kinds	Identify each type of model used by the viewpoint. For each type of model used, describe the language, notation, or modeling techniques to be used.
Model kind-metamodel	A metamodel presents the AD elements that comprise the vocabulary of a model kind. There are different ways of representing metamodels. The metamodel should present entities, attributes, relationships and constraints.
Model kind-templates	Provide a template or form specifying the format and/or content of models of this model kind.
Model kind-languages	Identify an existing notation or model language or define one that can be used for models of this model kind. Describe its syntax, semantics, tool support, as needed.
Model kind-operations	Define operations available on models of the kind.
Correspondence Rules	Document any correspondence rules defined by this viewpoint or its model kinds. Usually, these rules will be “cross model” or “cross view” since constraints within a model kind will have been specified as part of the conventions of that model kind.
Operations on Views	Operations define the methods to be applied to views or to their models. Operations can be divided into categories: Creation methods are the means by which views are prepared using this viewpoint. These could be in the form of process guidance (how to start, what to do next); or work product guidance (templates for views of this type); heuristics, styles, patterns, or other idioms. Interpretive methods are the means by which views are to be understood by the reader and system stakeholders. Analysis methods are used to check, reason about, transform, predict, apply and evaluate architectural results from this view. Design or implementation methods are used to realize or construct systems using information from this view.
Examples	This section provides examples for the reader.
Notes	Any additional information that users of this viewpoint might need or find helpful.
Sources	Identify the sources for this viewpoint, if any, including author, history, literature references, prior art, and more.

This viewpoint set initially appeared to be very promising, having an intuitive structure and seemingly being aimed at the kind of systems that we are interested in building. However, further investigation suggested that this viewpoint set is quite specialized and perhaps really aimed at supporting standards efforts rather than mainstream information-systems-architecture definition.

3.8 Common Architecture Viewpoint Model

In 2011, Vogel et al. present a common architecture view model to simplify the handling of view models [2]. This architecture view model abstracts from the views of the architecture view models subsequently handled and covers viewpoints that specify the name, the stakeholders and their concerns, and the important artifacts for the architecture views used. Table 8 shows the common architecture view model. This model has arisen following the architecture views from [13], [17], and [19].

Table 8. Architecture Views in Common Architecture View Model

Viewpoints	Definition
Requirements	Documentation of the architecture requirements.
Logical	Documentation of the architecture design.
Data	Documentation of aspects with regard to saving, manipulating, managing, and distributing data.
Implementation	Documentation of the implementation structure and the implementation infrastructure.
Process	Documentation of the control and

	coordination of concurrent building blocks.
Deployment	Documentation of the physical deployment of software building blocks.

4. Conclusions

In this paper, we have surveyed the state of art of architectural viewpoints models and frameworks. The use of viewpoints makes it easier to handle architecture views. Generic aspects in the creation of architecture views are easier to reuse and we do not have to redefine them redundantly for every system. Viewpoints provide a framework or template for creating architecture views. Architecture view models cover all relevant architecture views and thus enable us to make the architecture tangible and visible. The results of this study may serve as a roadmap to the software developers and architects in helping them select the right viewpoint model for their interests.

References

[1] U. van Heesch, P. Avgeriou, R. Hillard, “A documentation framework for architecture decisions”, Journal of System and Software, Vol, 85, No. 4, 2012, pp 795-820.

[2] O. Vogel, I. Arnold, A. Chughtai, T. Kehrer, Software architecture : a comprehensive framework and guide for practitioners, New York, Springer, 2011.

[3] P. Clements, F. Bachman, L. Bass, D Garlan, J. Ivers, R. Little, P. Merson, R. Nord, J. Stafford, Documenting software architectures : views and beyond, Upper Saddle River, NJ, Addison-Wesley, 2011.

[4] P. Clements, R. Kazman, M. Klein, Evaluating software architectures : methods and case studies, Boston, Addison-Wesley, 2002.

[5] M. A. Babar, I. Gorton, “Software Architecture”, In Proceedings of the 4th European Conference. ECSA, Denmark, 2010.

[6] P. van de Laar, P. America, J. Rutgers, S. van Loo, G. Muller, T. Punter, D. Watts, “The Darwin Project: Evolvability of Software-Intensive Systems”, Presented at Third International IEEE Workshop on Software Evolvability, 2007.

[7] B. Trosky, C. Arias, P. America, P. Avgeriou, “Defining and documenting execution viewpoints for a large and complex software-intensive system”, Journal of Systems and Software, Vol 84, No 9, 2011, pp 1447-1461.

[8] B. J. Williams, J. C. Carver, “Characterizing software architecture changes: A systematic review”, Information and Software Technology, Vol. 52, No 1, 2010, pp 31-51.

[9] J. Grundy, J. Hosking, “High-level static and dynamic visualisation of software architectures”, in Proceedings of the 2000 IEEE International Symposium on Visual Languages, Seattle WA, 2000, pp 5–12.

[10] N. Rozanski, E. Woods, Software systems architecture : working with stakeholders using viewpoints and perspectives, 2nd ed., Upper Saddle River NJ, Addison-Wesley, 2012.

[11] L. Bass, P. Clements, R Kazman, Software architecture in practice, 3rd ed., Upper Saddle River, NJ, Addison-Wesley, 2013.

[12] IEEE Computer Society. “Recommended Practice for Architectural Description”, IEEE Std-1471-2000, Available: http://standards.ieee.org/reading/ieee/std_public/description/se/1471- 2000_desc.html, 2000.

[13] ISO/IEC/IEEE, “Systems and software engineering - Architechture description”, Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=50508, 2001.

[14] J. A. Zachman, “The Zachman Framework Evolution. Zachman International”, Available: http://zachman.com/ea-articles-reference/54-the-zachman-framework-evolution, 2009.

[15] J. A. Zachman, “A framework for information systems architecture”, IBM Publication, 1987.

[16] P. Kruchten,, “Architecture Blueprints - the "4+1" View Model of Software Architecutre”, IEEE Software, Vol 12, No 6, 1995, pp 42-50.

[17] P. Kruchten, The rational unified process : an introduction, Boston, Addison-Wesley, 2004.

[18] J. Garland, R. Anthony, Large-scale software architecture : a practical guide using UML, Chichester New York, J. Wiley, 2003.

[19] N. Rozanski, E. Woods, Software systems architecture : working with stakeholders using viewpoints and perspectives, Upper Saddle River NJ, Addison-Wesley, 2005.

[20] D. Hornford, “TOGAF Version 9.1”, Zaltbommel. Van Haren Publishing, 2011.

[21] R. Hilliard, “Viewpoint modeling”, First ICSE Workshop on Describing Software Architecture with UML, 2001.