6-13-2008

Towards an Integrative Framework for Software Architecture

Rik Maes
University of Amsterdam, maestro@uva.nl

Guido Dedene
Catholic University of Louvain, guido.dedene@econ.kuleuven.ac.be

芽|Sprouts Working Papers on Information Systems ISSN 1535-6078

Towards an Integrative Framework for Software Architecture

Rik Maes
University of Amsterdam, The Netherlands

Guido Dedene
Catholic University of Louvain, Belgium

Abstract
A new integrative framework for software architecture is proposed, based on solid notions from traditional architecture. It combines elements taken from the Unified Process Model, that has been adjusted to this end, and a generic model for information management. It is argued that this framework can contribute to the development of higher quality software.

Keywords: software architecture, software complexity, software quality, framework, Unified Process Model, Zachman

Permanent URL: http://sprouts.aisnet.org/1-6

Reference: Maes, R., Dedene, G. (2001). "Towards an Integrative Framework for Software Architecture," University of Amsterdam, Netherlands . *Sprouts: Working Papers on Information Systems*, 1(6). http://sprouts.aisnet.org/1-6

Introduction

The term 'architecture' is frequently used in the context of computers and information systems. This paper contributes to the discussion on the validity of this interpretation; it further proposes a new framework for software architecture, integrating concepts from the Unified Process Model, information systems development frameworks and information management frameworks.

Being aware that a transfer of concepts should be based on correspondences as well as differences, the paper starts by revisiting some fundamental notions in architecture; these notions are taken from classical texts in the field of traditional architecture. The switch to the information systems development point of view is made in the next paragraph, where we present and partially adapt/extend the Unified Process Model (Jacobson et al., 1999), a major standard in software development. The adaptation and extension are made in view of the current emphasis on re-use in software practices. It is concluded that the UPM, even in its adapted form, only deals with one of the three basic notions of architecture.

Next, the two remaining notions emerging from the discussion of architecture are briefly discussed in the context of software architecture. It is concluded that they are only partially covered. One of these notions can be tackled by bringing in ideas taken from a generic framework for information management as developed by Maes (Maes, 1999). The resulting temptative framework for software architecture is proposed in paragraph 5 and evaluated in the concluding paragraph.

1. The view of architects on architecture

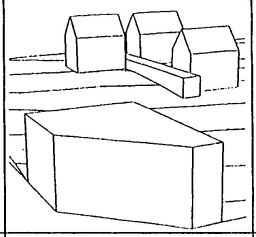
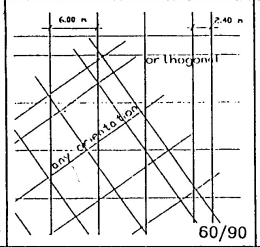
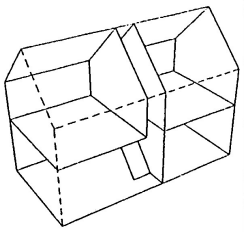
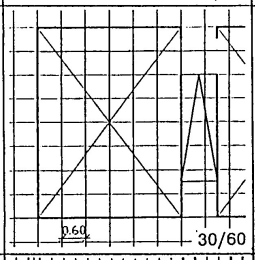
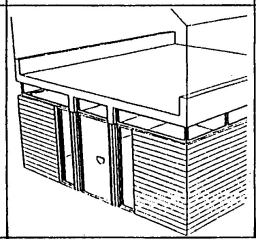
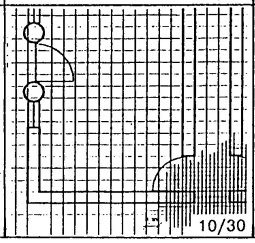
In many organizations, information systems have a questionnable reputation. Often, information systems are the critical, if not lacking factor in transforming business processes. They evolve into legacy software, which is increasingly difficult to maintain. To ensure that future systems avoid this disastrous evolution pattern, is an ongoing challenge for methodologies for information systems development.

Various authors argue that the growing *complexity* is a major factor in explaining this behaviour of information systems. Systems development should learn how to handle complexity, and – whenever possible – reduce complexity. Since the complexity of a system is lower bound by the complexity of the reality that it is supposed to handle, the emphasis is clearly on handling complexity of real world objects in the first place. This is, historically, precisely where *architecture* is entering the discussion.

Architecture was defined by Marcus Vitruvius Pollio as a way to deal with the complexity of the process of building, and more in particular building in such a way that value is added to the space-time in which the building is conceptualized and ultimately constructed. Vitruvius defines architecture by discussing three dimensions: “firmitas”, “utilitas” and “venustas” (taken from Vitruve, 1979).

Firmitas refers to the notion of structural construction aspects, expressed in an architectural ‘form’. Rendering structure in a building process is indeed an important instrument in dealing with complexity.

Utilitas indicates the functionality that must be realised. Modern architectural practices try to handle functionality by using abstraction levels. Abstractions should not be confused with levels of detail (Alexander, 1965). Appropriate abstraction levels can result in rendering intelligent zooming in architectural drawings. Instead of presenting more pixels while zooming, a next level of abstraction shows different concepts, which cannot be represented at a higher level of abstraction (Neuckermans, 1992). The following schema shows a fragment of a framework for architecture, that became the basis for a CAAD-tool that incorporates both abstraction levels and structural aspects (Neuckermans, 1992).

building program	design entities		grids	tests
	name	geometry & attributes	topology & attributes	
level 1 MASTERPLAN	- basic building types - masterplan blocks - circulation axes - site contingencies			. cost /m² or cost/m³ . surface / block . compactness . energy requirements . traffic . morphology . views and sights . shade and shadowing
level 2 BLOCK or TYPE	- rooms or singular spaces			. cost /m² based on ratios . surface and volume / space . temperature fluctuations . level of insulation . morphology
level 3 ROOM or SPACE	building elements: - wall - column - beam - arch - opening - door - window - stair - chimney			. gross-nett surface / space . cost based on elements method . comfort prediction . daylighting . sunshining . morphology . elementary stability

A leading principle behind this framework is the harmony between function and form in architecture. Mismatch between constructional aspects and functional aspects is avoided by aligning them.

Venustas is the third dimension in architecture, which encapsulates the esthetic notions of beauty and satisfaction. It seems needless to emphasize that Venustas and its relationship to both other aspects have extensively been subject of discussion in the literature on architecture; this concern was, e.g., eloquently expressed by the famous architect Eiffel, in his response in the newspaper Le Temps to a petition by members of the artistic establishment in Paris, protesting his project in 1887:

"Must it be assumed that because we are engineers, beauty is not our concern, and that while we make our constructions robust and durable we do not also strive to make them elegant?

Is it not true that the genuine conditions of strength always comply with the secret conditions of harmony ?

The first principle of architectural esthetics is that the essential lines of a monument must be determined by a perfect adaptation to its purpose"

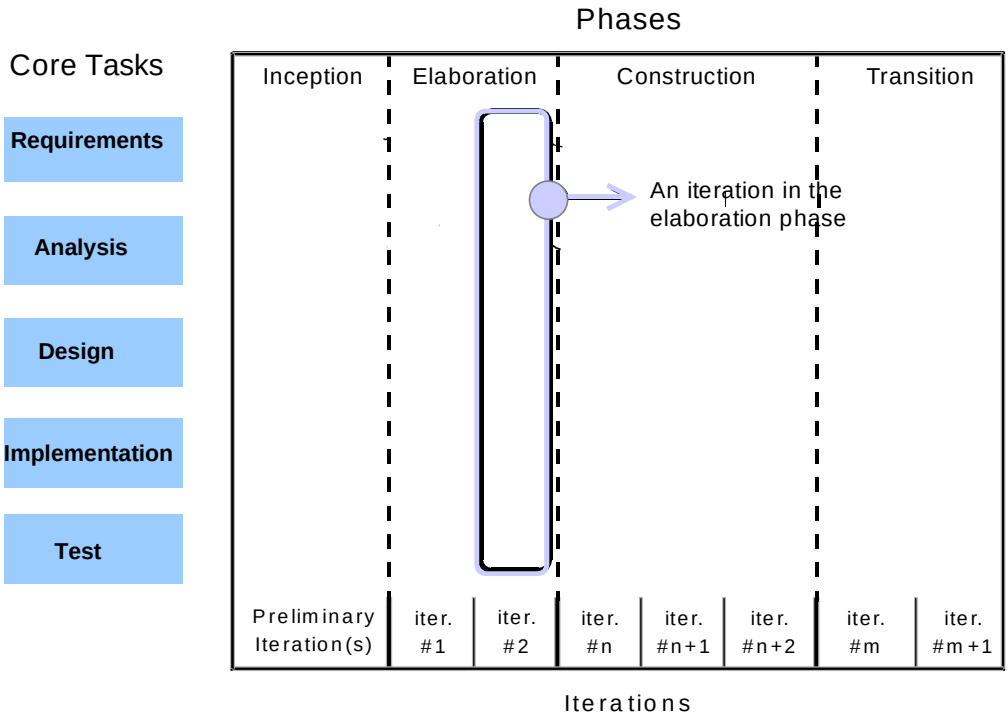
From this discussion, it becomes clear that architecture is the art as well as the science of ordering spaces, in particular building spaces, while applying and combining these three fundamental concepts. Some additional statements from architects illustrate and compliment this point of view very well:

- *Architecture is the thoughtfull making of spaces (Louis Kahn, mentioned in Van de Ven, 1978)*
- *L'architecture est le jeu savant et magnifique des formes rassemblées sous la lumière (Le Corbusier, 1958)*
- *Architecture is a system with elements which are mutually related and determined by their context (Alexander, 1963)*

2. The Unified Process Model: a critical discussion

Building software is the process resulting in a particular version/release of workable software for a business organisation. Over the past decades, many software process models have been proposed, with the "waterfall model" as probably the best known yet most controversial one.

The Unified Process Model, UPM (Jacobson et al., 1999) has been developed in conjunction with the Unified Modeling Language, UML (Booch et al., 1999), a notation standard for systems development schemas and drawings. The following shows the components originally proposed in the Unified Process Model.



The phases correspond to the major milestones in the development of a software system. The Unified Process Model stresses that for each new version/release of the software, the process must be re-iterated. The phases coincide remarkably well with the information/knowledge handling process as proposed by Choo (Choo, 1998). The phases aim at the following milestones:

- ***Inception***: the identification of the needs for the information system, also including the business case for the system and the partitioning of the system in clusters.
- ***Elaboration***: the identification and choice of the required systems components.
- ***Realisation***: the building and testing of the individual systems components, as well as the overall relationships between the system components.
- ***Transition***: handing over the system in the hands of the business professionals that will use it.

In each phase, iterations represent time boxes which should result in concrete deliverables in the software process. A typical time box in the Unified Process Model, e.g., could correspond to one, or a few "Use Cases" in the Unified Modeling Language.

The core tasks, on the other hand, represent the major types of activities that, to a larger or smaller extent, must be executed in each phase. The tasks are the following:

- ***Requirements***: the identification of the requirements for the system.
- ***Analysis***: the exploration of the problem domain for the system.

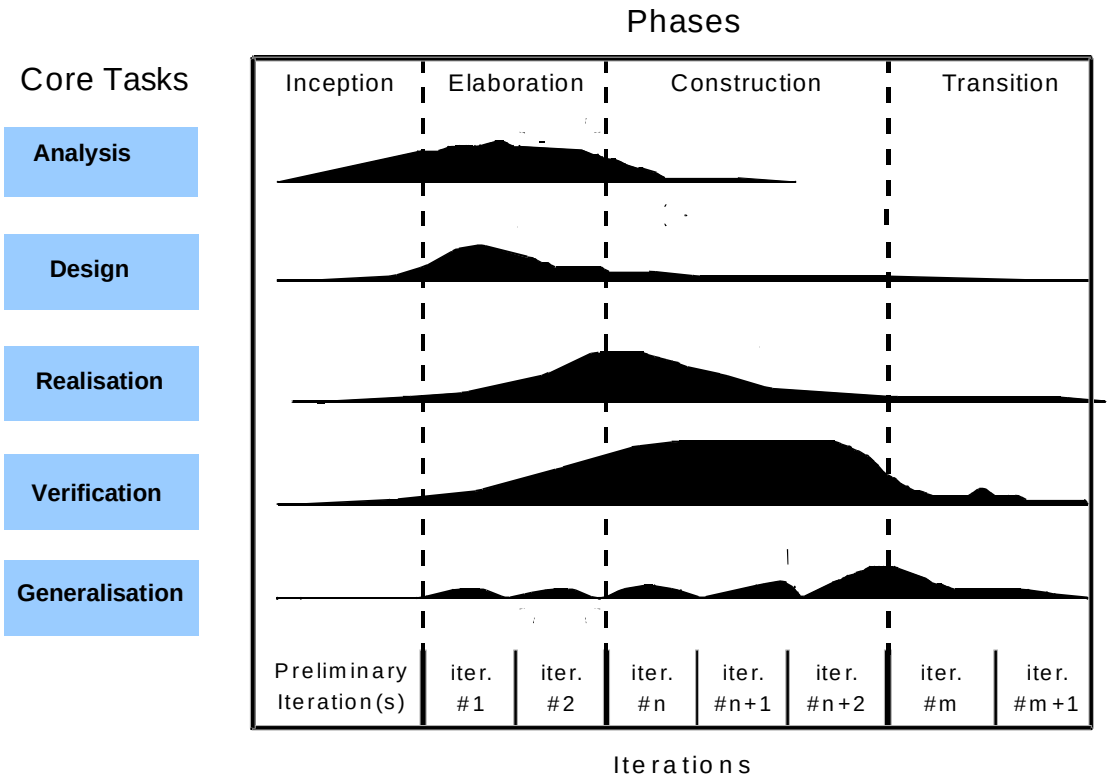
- ***Design***: the exploration of the solution domain, involving the development of alternative solutions for the problems identified during the analysis core task.
- ***Implementation***: the building of system components, including their documentation.
- ***Test***: the testing of the components and the overall system.

These generic phases and tasks of the Unified Process Model raise a number of questions. First of all, the difference between inception and requirements is unclear, even in recent publications on the Unified Process Model. Another element that is completely missing is the activity that tries to make software components (more) re-usable. Re-using software components is economically very interesting in software development, and has been proposed explicitly by various authors (e.g. Meyer, 1997) as an additional task in systems development. Finally, testing is only one way to verify the integrity and consistency of system components. Modern software engineering disciplines, including strong typing in object-orientation, allow the verification of software components very early in the definition of these components. Consequently, the following enhancements are proposed to the Unified Process Model:

- Drop the requirements task (all requirement aspects are covered in the activities of the inception phase)
- Expand the test task into a verification task
- Add a generalisation task (dealing with software re-use)

These recommendations will be taken into account when we integrate, in paragraph 5, basic ideas from the adapted Unified Process Model in an architecture-based framework for software architecture. The reason why we fall back to this model is rather straightforward: the UPM phases and generic tasks give a precise definition of the constructional aspects of software. Hence they incorporate guidelines for defining the *Firmitas* dimension of software architecture.

Remark further that the degree of people involvement in the combination of the generic tasks and the phases of this adapted Unified Process Model is not equally distributed; a global graphical representation of it is as follows:



This picture illustrates, among other things, that modern software engineering processes use more energy for the verification than for the realisation of systems components, in particular when they reuse existing software components.

The UPM is not the first (and will not be the last!) systems development model presented in the literature. A very famous and influential one was the one developed and further elaborated on by John Zachman (Zachman, 1987). It should be clear that all the explicit questions taken into account by Zachman (such as “why ?”, “what ?”, “how ?”, “where ?”, and so on) can be addressed by the constructional aspects of the enhanced Unified Process Model and hence by the Firmitas dimension of architecture.

3. Dealing with Venustas and Utilitas

A full framework for software architecture should equally well address the other two dimensions of architecture: Venustas and Utilitas. *Venustas* by and large corresponds with quality criteria for software systems, a shortlist of which can be found in the *MATURE* concept (Maes & Dedene, 1996):

- ***M***aintainability — A software system is maintainable if the opportunity costs for running the system can be kept minimal. This occurs, for example, if error corrections and implementation changes can be executed without introducing further errors in the system.
- ***A***daptability — It should be easy to add functionality to a system.
- ***T***ransparency — A system can be transferred smoothly between developers.
- ***U***ser-Friendliness — Users should recognise the systems functionality in a spontaneous fashion.
- ***R***eliability — A system should continuously perform as expected.
- ***E***fficiency — Minimal critical resources should be used for the systems performance.

Methodologies for systems development can indeed be evaluated against these criteria to determine how much *Venustas* they establish for both the developers as well as the professional users of the system.

Utilitas, the functionality of a software system, is traditionally treated in systems development methodologies without making any distinction in terms of abstraction levels. A typical example is the handling of requirements: methodologies tend to treat requirements as a global set of systems requirements (as also emphasized in the original Unified Process Model). The discussion of architecture given in paragraph 1 indicates that functionality can best be tackled by the use of levels of abstraction.

The next problem is the determination of the appropriate abstraction levels for software architecture. The influential Zachman framework for Information Systems Architecture (Zachman, 1987), e.g., identifies multiple levels of abstraction: Scope – Business – Information – Technology – Details. However, several questions can be raised about these levels of abstraction (Maes & Dedene, 1996), the most important of which are the following:

- Does ‘scope’ really belong to the abstraction levels of architecture, or is it a mere indication of the boundary conditions for information systems. Is ‘scope’, in other words, not primarily about making strategic choices concerning the context of (in particular: multiple) information systems?
- Is ‘details’ a valid abstraction level, or just one of the constructional aspects ?
- Zachman presents the levels of abstraction as levels of detail, which may not be appropriate in a contemporary view on architecture, as explained in paragraph 1.

We conclude that little well-thought attention has been paid to Venustas and Utilitas in developing architectural frameworks and hence that there is plenty of room for frameworks for software architecture that take the full spectrum of architectural notions into consideration. In the next paragraphs, we go looking for a temptative approach to this challenge, where we first look for an appropriate solution for the levels of abstraction (paragraph 4) and then combine Firmitas and Utilitas into a more integrative framework (paragraph 5).

4. Looking for the appropriate levels of abstraction

Abstractions, core concepts in Utilitas, are treated in the software literature as unclear and complex concepts. Therefore, we depart from regular approaches to software development and revert to models dealing with the ICT-business relationship in general. A reference model there is the one of Strategic Alignment, as proposed by Henderson and Venkatraman (Henderson & Venkatraman, 1993). An interesting viewpoint on abstraction is presented in a recent elaboration of this model, the generic framework for information management as developed by Maes (Maes, 1999). Unlike the Strategic Alignment and many other frameworks from the nineties, which consider merely "Business" and "IT – Information Technology" as abstractions, this management framework introduces three fundamental abstraction levels:

- ***Business***: the viewpoint of the business professionals, responsible for that part of the business that must be addressed and supported by the software system.
- ***Information/Communication***: the viewpoint of the professional users of the systems, in terms of the inherent information and communication processes and hence of the input/output functionality of the software system.
- ***Technology***: the viewpoint of the software engineers, responsible for the implementation and scheduling alternatives for the operational software system.

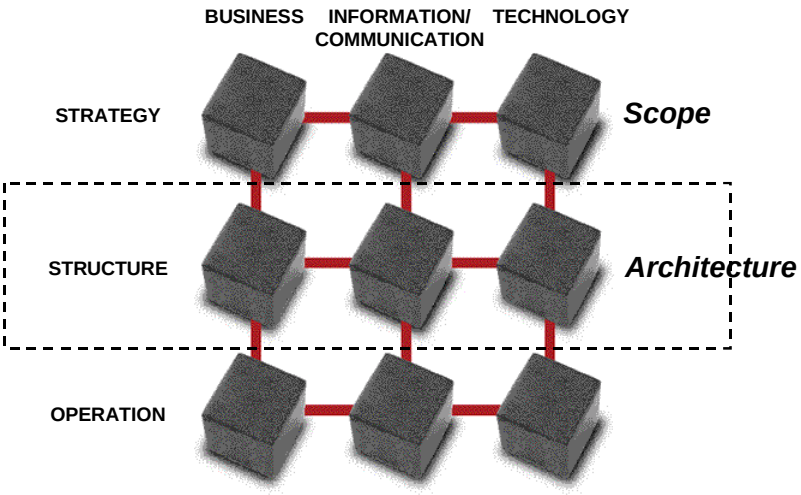
When using these abstraction levels, the systems requirements can be refined in business requirements, information/communication requirements and technology requirements. The expertises involved correspond with the ones introduced by Choo (Choo, 1998): domain expertise, information expertise and technology expertise. Complexity, hence, can be broken down in three relatively separate areas, each treated by the appropriate expert/architect; this reduction of complexity is precisely what architecture is all about!

The generic framework for information management (Maes, 1999) further considers a second dimension, adding up to an enneahedron. This second dimension in its way presents a solution for the

classical discussion on "scope" versus architecture. The elements of this vertical dimension of the enneahedron correspond with the following (information) management subdisciplines:

- ***Strategy***: the discipline of making strategic choices, and aligning them across business, information/communication and technology.
- ***Structure***: the discipline of translating the strategic decisions into mutually adjusted business, information/communication and technology (infra)structures. This discipline represents the organizational component of management.
- ***Operations***: the discipline of executing and operating the proposed structures in such a way that operational excellence is attained. At this level, e.g., all typical ITIL (IT Infrastructure Library) procedures come into consideration.

Considering this managerial dimension, it is evident that scope is positioned at the strategic level. Architecture, on the contrary, is precisely the discipline dealing with the structuring of strategic decisions and translating them ultimately into operations. The following summarizes this discussion, showing the position of the concepts of architecture and scope in the generic framework for information management:

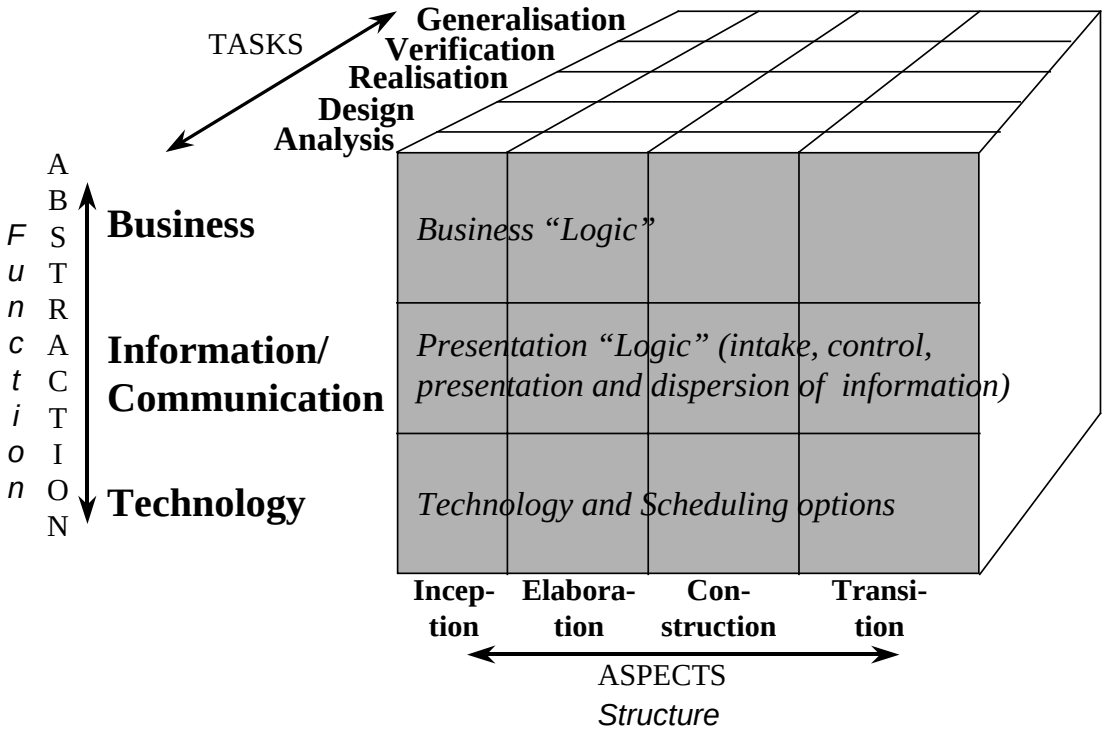


5. Outline and application of an integrative framework for software architecture

The framework for software architecture we propose in this paper, is based on the adapted Unified Process Model (paragraph 2) and on the generic framework for information management (paragraph 4):

- The greatly adjusted core tasks of the UPM
- The phases of the UPM, in combination with the tasks dealing with Firmitas
- The abstraction levels of the generic framework for information management, dealing with Utilitas (function)

The resulting integrative framework for software architecture that is proposed in this paper is graphically represented in the following schema:

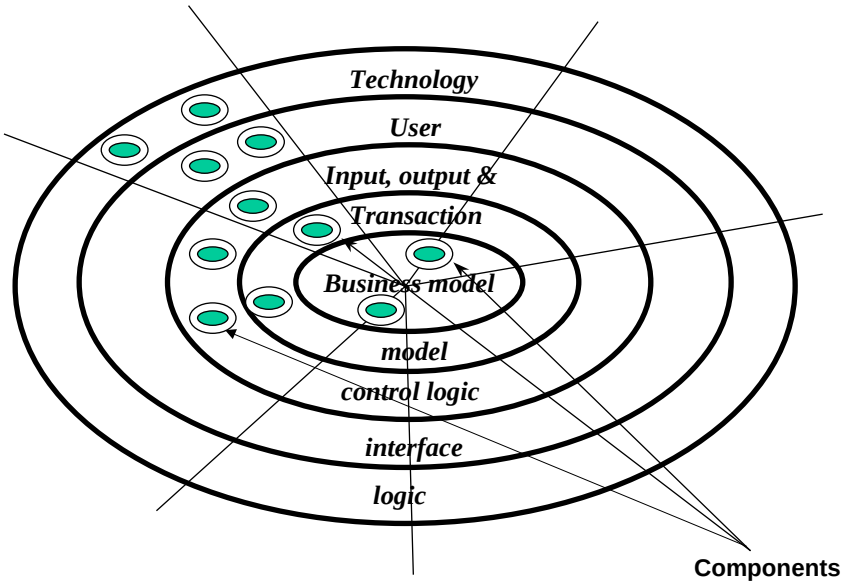


This schema incorporates an explicit checklist for the *Utilitas* and *Firmitas* dimensions of software architecture. The *Venustas* dimension is covered by the *MATURE* list of quality criteria discussed before.

As a matter of fact, the real test of any construct is in its interpretation and its application (its own *utilitas*). Various interpretations/applications of this integrative framework can be envisioned, and some of them are suggested here. First of all, an obvious question is how systems developed under architecture can be recognised and discerned from others. An important criterion is of course the realisation of the different abstraction layers of the *utilitas* dimension in the software specifications. The business functionality constitutes the external, core functionality of the software system, and the other layers are built around it.

Such an approach is not new, and was proposed earlier (Jackson, 1983). A fundamental critique at that time was the fact that this layering heavily impacts the software development process: the

information/communication layers, e.g., can only be developed when the business model is completed. Today, with component-based and object-oriented development approaches, these remarks are obsolete. An architected software system can nowadays be developed in slices, where each slice contains the components/objects with their appropriate levels of abstractions (which can in their turn be represented as stereotypes in the Unified Modeling Language notation).



Another important application of the proposed framework is the fact that it incorporates a structure for the meta-models which underlie Computer Aided Software Engineering (CASE) tools and application software frameworks. Frameworks and tools that really assist the developers in realising their tasks should incorporate the dimensions of the proposed framework for software architecture.

The framework will also stimulate the research for patterns for software re-use: business analysis patterns are different from, for example, technical design patterns. An overwhelming rich variety of patterns has been proposed in the literature today (Jézéquel et.al., 2000). The framework can help considerably in classifying these patterns, also in view of their applicability in the software development process.

Finally, the framework can help in realising a better degree of exchange between software system specifications. Software architecture can help to improve the degree of model interchanges between software development communities. Needless to emphasize that this may be very important to stimulate modern software approaches, such as Open Source Software.

Conclusions

This paper started by analyzing the very nature of architecture. Based on this, an investigation was made of the various components of an integrative framework for software architecture. A new, more integrative framework was derived from the Unified Process Model and from the generic framework for information management. The proposed framework enjoys a number of advantages:

- The structure of the framework is compliant with the way the original and seminal Zachman framework for Information Systems Architecture (Zachman, 1987) was conceived. However, it also satisfies further formalisations of this type of frameworks, as proposed by Martin and Robertson (Martin & Robertson, 1999).
- The proposed framework coincides with two of the three dimensions in architecture, in the way they have been discussed in the classical literature on architecture.
- The proposed framework is compliant with and enhances the Unified Process Model and the Unified Modeling Language. This allows a fair integration of the framework with a variety of tools that are available today. Moreover, the proposed framework induces further structure in the UML notations.
- The proposed framework fits neatly into a generic conceptual framework for information management.

Further research should reveal to what extent the proposed framework can assist in comparing and assessing software development methodologies. It is our impression that systems that are realised under architecture enjoy a higher degree of quality and a lower degree of complexity. The proposed framework incorporates modern insights taken from the field of software development and of information management, but equally well standing insights from classical literature on architecture. At least it offers a guarantee that systems built under this architecture and complying with the *MATURE* set of conditions are up-to-date elaborations of time-honoured visions on architecture.

References

Alexander Ch., 'A City is not a Tree', in: *Architectural Form*, April 1965, vol. 122, nr. 1, pp. 58–61.

Alexander Ch., 'The determination of Components for an Indian village', in: *Conference on design methods*, Pergamon Press 1963, pp. 83–114.

Booch G., Rumbaugh J. & Jacobson I., *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.

Choo C. W., *The Knowing Organization*, Oxford University Press, 1998.

Henderson J.C. and Venkatraman N., 'Strategic alignment: leveraging information technology for transforming organisations', in: *IBM Systems Journal*, 1993, vol. 32, nr.1, pp. 4-16.

Jackson M., *Systems Development I*, Prentice-Hall, 1983.

Jacobson I., Booch G. & Rumbaugh J., *Unified Software Development Process*, Addison-Wesley, 1999.

Jézéquel J.M., Train M. & Mingins Ch., *Design Patterns and Contracts*, Addison-Wesley, 2000.

Le Corbusier, *Vers une Architecture*, Fréal, Paris, 1958.

Maes R. & Dedene G. , *Reframing the Zachman Information System Architecture*, Tinbergen Institute discussion paper TI 96-32/2, Amsterdam, 1996.

Maes R. , *Reconsidering Information Management Through a Generic Framework*, PrimaVera Working Paper 99-15, Universiteit van Amsterdam, 1999.

Martin R. & Robertson E.L., *Formalization of Multi-level Zachman Frameworks*, Technical Report 522, Computer Science Department, Indiana University, 1999.

Meyer B., *Object-oriented Software Construction*, Prentice Hall, 1997.

Neuckermans H., 'A conceptual model for CAAD', in: *Automation in construction*, 1992, vol. 1, nr. 1, pp. 1–6.

Van de Ven C., *Space in architecture*, Van Gorcum, Assen (Ndl), 1978.

Vitruve, *Les dix livres d'architecture corrigés et traduits en 1684 par Cl. Perrault*, Margada, Brussels, 1979.

Zachman J.A., 'A framework for information systems architecture', in: *IBM Systems Journal*, 1987, vol. 26, nr. 3, pp. 276–292.

Towards an integrative framework for software architecture

Rik Maes [1] & Guido Dedene [1,2]

[1] Afdeling Accountacy & Informatiemanagement
Faculteit der Economische Wetenschappen en Econometrie
Universiteit van Amsterdam
Roetersstraat 11
NL 1018 WB Amsterdam
The Netherlands
maestro@fee.uva.nl

[2] Vakgroep Beleidsinformatica
Faculteit Economische en Toegepaste Economische Wetenschappen
Katholieke Universiteit Leuven
Naamsestraat 69
B-3000 Leuven
Belgium
guido.dedene@econ.kuleuven.ac.be

ABSTRACT: A new integrative framework for software architecture is proposed, based on solid notions from traditional architecture. It combines elements taken from the Unified Process Model, that has been adjusted to this end, and a generic model for information management. It is argued that this framework can contribute to the development of higher quality software.

KEY WORDS AND PHRASES: software architecture, software complexity, software quality, framework, Unified Process Model, Zachman

Contents

芽|Sprouts

Working Papers on Information Systems | ISSN 1535-6078

Office:
Sprouts
University of Amsterdam
Roetersstraat 11, Room E 2.74
1018 WB Amsterdam, Netherlands
Email: admin@sprouts.aisnet.org

芽|Sprouts