An Architecture Process for Repeatable Design

Stef Joosten[1,2], Rieks Joosten[3]

[1]*Ordina* stef.joosten@ordina.nl

[2]*Open University of the Netherlands* stef.joosten@ou.nl

[3] *TNO Informatie- en Communicatie Technologie* H.J.M.Joosten@telecom.tno.nl

Abstract

Architects face the challenge to make their work more concrete in the eyes of their clients, their users and other stakeholders. Their mission is to create a coherent and consistent structure of applications, systems, and business processes that satisfies the rules and requirements of the business. However, members of an architecture team sometimes get caught in the middle of complex terminology discussions, deadlines and extremely large amounts of design artifacts. So how are they going to deliver useful results for the business? This paper presents a software tool that signals inconsistencies and incompleteness in business, application and infrastructure architecture. An architecture team can monitor its collective work in real time, allowing architects to remove the last inconsistency. Besides, the software provides project managers with an objective instrument to monitor architecture projects.

1. Launching Business Initiatives

After putting men on the moon, NASA wanted to intensify space travel. She was in dire need for more repeatable, safer, more reliable and cost efficient means to make frequent trips to space. That is why the space shuttle program was developed. Information Technology is in a similar situation. In order to make further progress, IT must become more repeatable, safer, more reliable and cost efficient. Rather than managing every IT project individually (comparable to Saturn missions), organizations conduct IT programs (comparable to the space shuttle program) to make IT more manageable, less costly, and more predictable. Organizations that depend on continuous innovation must launch business initiatives at ever shorter time intervals. This justifies a substantial effort to turn innovation a repeatable process. That is why organizations are willing to invest in architecture. Scientific developments in this area focus on patterns [e.g. 1, 2], enterprise frameworks [e.g. 3], or methodology [e.g. 4]. Our approach is to study the *architecture process*, referring to the work of architects that is concerned with developing satisfactory and feasible system concepts, maintaining the integrity of those system concepts through development, certifying built systems for use, and assuring those system concepts through operational and evolutionary phases [5]. An architecture process serves as a booster rocket, fuelling the innovation process to perform reliably when launching new business initiatives.

This article focuses on the architecture process, which we view as a process of rule making and monitoring. We will show how these rules can be used to monitor the architecture process in real time. This brings the idea of IT architecture from a description mechanism towards a control mechanism. Only then architecture might do for IT what the electricity plug has done for home appliances: more freedom to bring new ideas to larger markets with a better chance of success.

Essential ingredients of architecture are business rules, i.e. rules that are verifiably true or false, universally valid in a particular context, and provide relevant information to the business when violated. Throughout this paper, we use the word *rule* or *business rule* in this particular meaning. Otherwise we use the word *principle* or *guideline*.

This paper starts with a discussion on the architecture process. We then discuss the role of an architecture monitoring tool. The tool we have built uses the ArchiMate[1] language [6] in the role of "standardized electricity plug". Then we discuss an experiment conducted with that tool. The results show that the rules, which govern architecture, can be used to build architecture checkers in a generative way. That provides IT-governance with a concrete instrument for checking architecture compliance.

[1] The ArchiMate project (http://ArchiMate.telin.nl) was partially funded by the Department of Economic Affairs and delivered on December 31st, 2004.

This paper adheres to the architecture definitions from the IEEE Recommended practice 1471-2000 [5].

2. Designing Repeatably

The quality of a large design depends largely on the level of coordination an architecture team can achieve. Members of an architecture team spend much of their time trying to match design decision with business requirements, trying to fit solutions in the infrastructure, trying to solve difficulties with legacy applications, trying to avoid inconsistencies, communicating with stakeholders, trying to keep users involved, and so on. Making a large design consistent and complete often requires many meetings, peer reviews, and lots of interviews and workshops. Coordination is the name of that game.
In our analysis we have identified the following (groups of) stakeholders:

- architect: wants clarity, less discussion and more results;
- Architecture team: wants a concrete result in a consistent, buildable way, with support of all stakeholders;
- Project leader: wants to manage a team of architects;
- Acquirer (e.g. an executive who has assigned an architect to a project): wants assurance, low cost, control;
- Customer: wants fast, flexible and fine services.

The purpose of architecture is to accelerate and improve the innovation process such that new business initiatives can be launched routinely and reliably. The research focuses on the question how an architecture process fosters repeatability.

3. Managing Architecture

Architects face the challenge of structuring complex situations. They must bring clarity and reduce perceived chaos by providing simple icons and metaphors that inspire stakeholders. They are put to the challenge of curbing that complexity. This broader challenge must be understood before turning to solutions. To that end, we have studied the architecture process.

If architecture makes innovation into a repeatable process, and a repeatable innovation process is required to launch business initiatives, a strong resemblance with the space shuttle program emerges. The large fuel tank corresponds to the innovation process, where architecture and management serve as booster rockets (figure 1)[2]. The launching of the shuttle itself represents the launching of business initiatives, which is done repeatedly, reliably, and relatively cost efficient. The entire system is designed to bring large numbers of business initiatives into orbit.

In our analysis we have identified three levels of architecture: the project, the program and the corporate level. architects provide concrete form and meaning in all three levels. In IT projects they create innovations that affect both the organization and information technology. Depending on the particulars of each project, various kinds of designers are

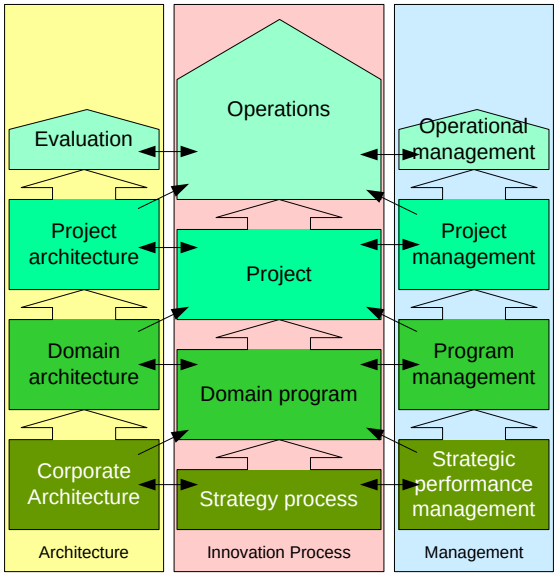

figure 1

involved, such as business designers, process designers, application designers, infrastructure designers, etcetera. One level up, at the program level, architects make rules and principles for the purpose of coordinating individual project efforts. Here, architects study commonalities of large numbers of projects, enforce standards, create reference models, collect best practices in the domain, and disclose their work to all stakeholders involved in projects within that program. On the next level, corporate architects set standards, devise rules that implement governance principles (such as IFRS,

[2] The idea of an architecture process and a management process that support the innovation process from two sides is due to Tinus de Gouw, who currently works with Rabobank.

the Sarbanes-Oxley act of 2002, safety regulations, etc.), implement corporate policies, etcetera. The project, the program and the corporate levels correspond to the operational, tactical and strategic management levels of the innovation process.

Architecture can be understood as a process of rule making and rule monitoring on all three levels. On the corporate level, architecture provides the rules and principles that are valid throughout the organization. Within each program, rules are defined that are valid throughout the program but not beyond. Each project must abide by the rules of the program and the corporate rules. Besides, every project may have its own architecture, setting particular rules within the project. In this analysis, architects require a rule base in which a rule is valid within its particular context.

Any omission and any violation of a rule made by an architect may yield problems when the design is realized. It always takes extra time, but may also cause rework or even redesign, leading to possible setbacks in the innovation process. Thus, violations of rules pose a direct threat to the repeatability and reliability of the innovation process. If designs are guaranteed to be free of architectural violations, this increases repeatability of innovation, and decreases the risk of launching new business initiatives.

In order to obtain flawless designs, we need a mechanism to signal violations. This requires to know which rules apply to a design, a mechanism to compute signals on the basis of violations, and a way to communicate those signals to a stakeholder with the authority to act upon each signal. Computer support is needed here. There are many different rules that are valid within many different contexts in an organization, so it is not reasonable to manage those rules 'by hand' and expect no mistakes. There are many different projects and a vast amount of design artifacts, so it is far too much work to take out all rule violations without the help of computers. These requirements inspired us to build an architecture checker.

As a result, an architecture process can be implemented as depicted in figure 2. If all design changes are fed into a repository, a checker can produce signals and feed them back into the architecture process. A signal confronts an architect instantaneously with design decisions of his or her peers. The mechanism is limited to a signaling function only. Enforcement is left to the individual style of each project.

Our analysis shows that designers need more than tools for drawing and software generation. Besides the available tools, a checker to monitor architecture is useful to keep team members aligned with the rules of the business.

4. Checking the Rules

The architecture checker that was built has a simple structure (figure 3). A repository is the foundation. It contains information about business processes, roles, applications, services, nodes, communication paths, etcetera, according to a structure described in the ArchiMate project. This choice was made because the ArchiMate architecture language has a reknown status in the Netherlands and is acknowledged by Dutch professionals throughout science and industry. The ArchiMate reference manual [7] provides an accurate description of the language structure in terms of a metamodel. Semantic rules however, are described in natural language. Most of these rules describe multiplicity restrictions, i.e. omissions and ambiguities that might arise from design errors. The repository (written in MySQL) satisfies the ArchiMate structure (the metamodel) and the rules of ArchiMate have been translated into a software component (written in PHP) that checks for violations (the service layer) and presents them as signals in a browser (the presentation layer).

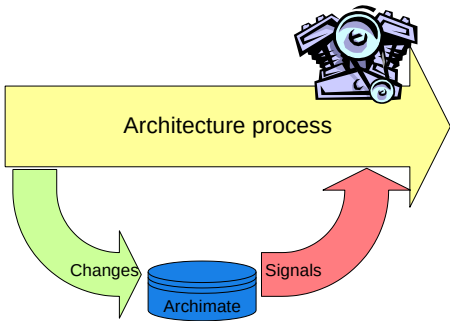

figure 2

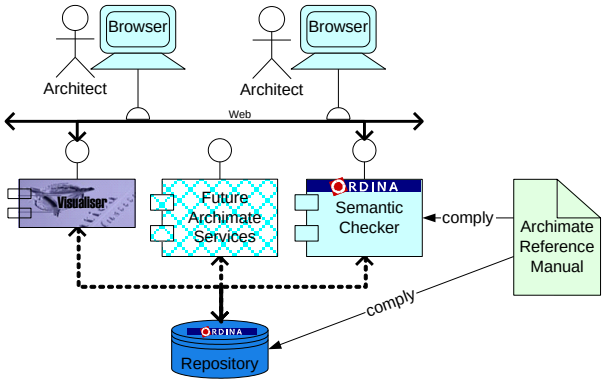

figure 3

Architects gain access to the repository and checker by means of a browser. The repository allows multiple users, so any changes made by one architect are visible for the team members. The visualization component is currently (at the time of writing) being installed at the Telematica Institute in Enschede. The repository and checker have been built at Ordina. The design is such that later extensions can be made without excessive effort.

Designers can use the checker by inspecting and analyzing signals from the checker and changing the design (as represented in the repository) accordingly. In doing so, new signals may arise from the checker. By dividing the total design space among themselves, architects can distribute the work. For instance, one might concentrate on the business architecture, another on the application architecture and a third on the infrastructure. If for example, a team member defines a new service, an omission arises in one of the tables in the repository, saying that a node is required on which to run that service. When an application is defined to use that service, a signal is risen when there is no interface to make that service available. These examples (and all others) show how an architecture checker provides architects with useful information to complete or correct their work. The repository stores concrete design choices in tables, such as the assignment of application components to processes, business roles to business interfaces, network components to software components, etcetera. Whenever a signal occurs, it is up to the designer to determine the meaning of that signal (diagnosis). The checker provides the signals only, relating them to the particular rule being violated. When a team is done and all signals are resolved, the checker guarantees that the design satisfies all of ArchiMate's rules.

5. Experimenting with the checker

The first experiment was carried out on September 12th 2005. The purpose of this experiment was to gain insight in practical questions: Can architects grasp the idea quickly enough? Does the tool impose unreasonable restrictions? What can an architecture team achieve in a limited amount of time? Is the software sufficiently robust? And most importantly: do architects feel that this type of tool is useful?

We picked three experienced architects, one of which was knowledgeable with Archimate. We confronted the subjects with the design of a (fictitious) insurance company, ArchiSurance [6]. Before the experiment, the ArchiSurance design was translated literally from Archimate documentation into the repository. The team was asked to prepare by studying an Archimate primer [6] and the Archisurance case contained in that primer. The experiment consisted of resolving all signals detected by the checker in one hour. Since the checker was new to all team members, a short oral instruction was provided just before the experiment. Each team member was given one part of the design space as his own responsibility. By keeping the preparation down to an absolute minimum, the experiment provided a good indication about the threshold of use.

During the experiment, it took the team about 15 minutes to get used to what the tool showed them and to get going. After an hour, the team had investigated twenty signals and resolved thirteen. Team members would typically trace a signal straight back to the original design, and negotiate who would make the necessary adjustments.

In a retrospection session, both the architecture process and the tool were experienced positively. Team members focused their attention especially to the rules, questioning whether the right rules were being checked. They experienced the nature of ArchiMate's rules to be too general. Control questions showed that the subjects were very much aware of what they were doing. For example, they were able to place the checker flawlessly in the upper left area of figure 1 (without having read this paper…) The fact that the entire design was represented in a repository allowed them to get down to work straight away. None of the team members had felt the urge to address terminology of definitions underlying the architecture. The primary contribution was seen in the mutual coordination among architects in a team.

6. Results

The results of the experiment show that the checker has supported the team as intended. On the basis of these results, more experiments and more specific experiments will be conducted in the near future.

The architecture checker means different things to different stakeholders.

Designers have an instrument to coordinate their work. They can freely invent their designs, but their work may yield signals elsewhere. The discussions that arise are concrete, since they are based on concrete signals. Also, these discussions are necessary in order to resolve signals. The experiment showed that these discussions are necessary, relevant

and to the point, indicating that the checker indeed helps to avoid abstract, pointless discussions.

For the team as a whole, the checker results in a consistent result. Once all signals have been resolved, all rules are satisfied and consequently the design complies to the architecture. Only when rules are not being checked, signals might still occur. The entire result is like the team has worked as one architect. Since abstract discussions (e.g. about terminology) are avoided, the team effort as a whole is more manageable and predictable.

The project manager can benefit from the list of signals, because it measures rule violations in an objective way. This provides managers with real-time feedback on progress in the team. It reduces their dependencies on reports from team members, which may be subjectively flawed. Besides, the lists of omissions and ambiguities provide an attractive means for work distribution among team members.

An acquirer gets more assurance about the quality of designs. The absence of signals about a particular rule means that the design satisfies that requirement for 100%. Besides, more predictable design times translate directly into a reduced project risk. Finally, and most importantly, every business rule satisfied is a business requirement fulfilled. This can even be guaranteed in writing and signed off by a chief architect.

Customers have indirect benefits, albeit not less noticeable. For consistent architecture yields a flexible and maintainable system, which enables the organization to respond adequately and flexibly to the individual and continuously changing needs of their customers.

Besides results for stakeholders, there is one observation of scientific interest. The responses of subjects in the usability experiment have provided a new insight. Apparently, the set of rules coming from ArchiMate were not sufficiently relevant for the architects. They were considered too general. Architects require a more specific level, but this would make ArchiMate either impractically loaded with terms or far to specific to be of use for many architecture projects. This is subject for further research.

Our findings correspond to predicted findings in earlier work [8]. Benefits of concreteness in architecture and a speed-up of the work of an architecture team were corroborated in this experiment.

4. Conclusions

Monitoring architecture processes by means of an automated checker can bring repeatability in innovation. This has been demonstrated by building the checker and performing the usability experiment.

The Archimate reference manual has proven to be an adequate basis for building tools. Practically all of that manual could be implemented directly.

The usability experiment has shown that real-time feedback provided by the checker is definitely an improvement of the architecture process. It allows architects to act more professionally, and renders the architecture process more predictable and reliable.

Further research must be conducted to include project specific rules into the checker.

10. References

[1] M. Fowler, Analysis Patterns - Reusable Object Models, Addisson-Wesley, Menlo Park, 1997.

[2] E. Gamma and R. Helm and R. Johnson and J. Vlissides, Design Patterns - Elements of Reusable Object-Oriented Software, Addison-Wesley, London, 1995.

[3] M.E. Fayad, D.S. Hamu, and D. Brugali, Enterprise Frameworks Characteristics, Criteria, and Challenges, Communications of the ACM, October 2000/Vol. 43, No. 10, pp. 39-46.

[4] S.M. Yacoub and H.H. Ammar, UML Support for Designing Software Systems as a Composition of Design Patterns, in: M. Gogolla and C. Kobryn (Eds.): UML 2001, Springer-Verlag Berlin Heidelberg, LNCS 2185, pp. 149-165, 2001.

[5] IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, IEEE Std 1471-2000.

[6] Marc Lankhorst and the ArchiMate team, ArchiMate Language Primer: Introduction to the ArchiMate Modelling Language for Enterprise Architecture, version 1.0, Telematica Instituut, 25-08-2004. http://doc.telin.nl/dscgi/ds.py/Get/File-43840/

[7] René van Buuren, Stijn Hoppenbrouwers, Henk Jonkers, Marc Lankhorst, Gert Veldhuijzen van Zanten, Architecture Language Reference Manual, version 4.0, Telematica Instituut, Radboud Universiteit, 13-12-2004. http://doc.telin.nl/dscgi/ds.py/Get/File-31626

[8] Joosten, S.M.M., Architectuur met rendement, Database systems conferentie, RAI Amsterdam, 24 april 2002.