university of groningen / faculty of mathematics and natural sciences / computing science

Software Architecture

Specification of the architecting project

Paris Avgeriou (paris@cs.rug.nl)

Contents

1 Background

The architecting project is an essential part of the SA course and substantial time is reserved for it. It links the various subjects that are discussed and gives the participants an ideal opportunity to apply the theory covered during the lectures. The practical assignment is a realistic and non-trivial system architecting project. The starting point is a short system description (section 2) with the major requirements.

During the course the students will gradually convert this vague and maybe incomplete description into a real system architecture with a well defined business context, reasonably complete requirements and finally a software architecture. The result of the architecting project will be a system architecture document. The proposed structure of this document is given in section 5.

The time budget for the architecting project is determined by the course schedule. Although the students may spend time on the architecting project outside course hours the available time in the course should be enough to come up with acceptable results. The fact that the time budget is limited corresponds to industrial practice. This has the obvious advantage that it is rather easy to withstand the temptation to spend too much time on irrelevant details.

During the course, a number of reviews will be organized in order to provide the students with early feedback. For each review, an improved and extended version of the system architecture document has to be presented. The final document is thus built up in an iterative and incremental way.

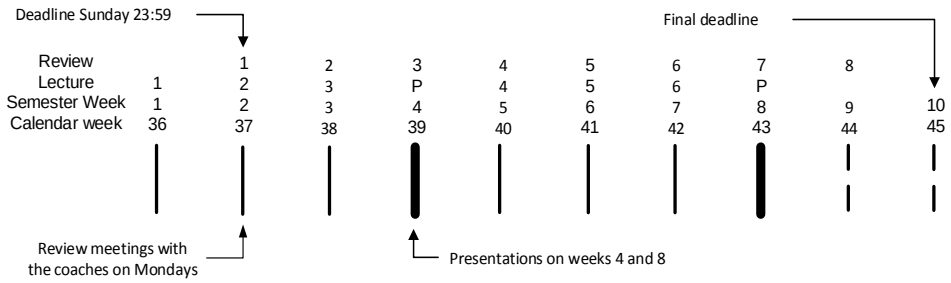
The coaches act not only as expert reviewers, but also as stakeholders. As expert reviewers, they will verify the architecture and its documentation. As stakeholders, they will also validate the requirements and the final results from the point of view of the business, the market and the users, the manufacturing department and the maintenance department. In their role as stakeholders, the coaches can also be consulted by email about critical requirements and design decisions. Individual groups are still **free to designate some of their own members to act as stakeholders**, if they consider it more suitable.

It should be understood that the stakeholders are leading in the sense that they represent the requirements of the market (the users), the customers (company commissioning the system), the service organization, the software engineering organization (the implementers), etc. On the other hand, the realization of the wishes of the stakeholders depends on the technical possibilities developed by the software architects. In case of students acting as stakeholders, they may check their decisions with the coaches.

This set-up of the assignment makes the architecting process more challenging, but also more realistic. The consequences of new requirements and change requests must be carefully negotiated and agreed between stakeholders and architects. Close cooperation is also necessary to avoid unnecessary waiting times and to complete the assignment within the timeframe of the course.

The cooperation between stakeholders and architects suggested by the table below, requires a flexible planning and does not work if the groups proceed according to a waterfall model. Quick iterations over the various aspects of the architectures (like quality attributes, views, design and verification) are required. The members of each group must thus work simultaneously on different aspects of their task and cooperate closely with the other group members.

The schedule of the course is illustrated in the next figure.



In terms of the document structure and contents, the deliverables for Semester Weeks 3, 5, 7, and 9 are defined in the following table. The deliverables for Semester Weeks 2, 4, 6, and 8 are not strictly defined and can be negotiated between coaches and groups.

Deliverable	**Group**	**Name**	**Chapters**
1 Week 3	SHs	Requirements	Draft chapter 1,2, 3
	SWAs	Extensive analysis	Draft chapter 4
2 Week 5		Initial architecture	Reworked and improved version of chapter 1 to 4 Draft chapter 5, 6 and 7
3 Week 7		Elaborated architecture Draft evaluation of the SWA Draft system evolution	Reworked and improved version of chapter 1 to 7 Draft chapter 8 and 9
4 Week 9		Final architecture Final evaluation of the SWA Final system evolution	Reworked and improved version of all chapters
Final Week 11		Consolidated architecture description	Reworked and improved version of all chapters

Rules:

- In order to give the coaches sufficient time to review your documents and perform the review on Monday afternoon, the deliverables must be sent to coaches no later than 23:59 hours on Sunday.
- Documents delivered after the deadline will, in principle, not be reviewed and you will miss important feedback and help. It is the courtesy of the coach in question if he, nevertheless, reviews the document. Most probably this review will be more superficial because of lack of time.
- All 8 reviews should be scheduled by Semester Week 2.
- The architecture description must be either a Word document or a pdf document, growing per deliverable. If exported from an online version (e.g. Google Docs) make sure the formatting is appropriate.
- The first page of the architecture description must show the title of the project, the group number and members, the release date and the document version.
- An appendix must show the person responsible for a particular chapter or section together with the contributors.
- The document structure suggested in section 5 is suggested, but not mandatory.
- After each review, the reviewed parts of the document must be updated too.
- Incremental changes must be indicated in the change history and in the document (**tracked changes**), independently of the format used (Word/LaTeX).
- For questions, you can contact your coach by email or by trying to see him.

2 Description of the system to be developed: Smart Flood Monitoring

2.1 Introduction

You are the architecture team in a company that specializes in designing and developing Smart Systems for Environmental Monitoring: innovative ICT-based products and services that aim at monitoring natural environments like rivers, lakes, forests, farms, natural parks etc. With the emergence of new sensor technologies and novel aerial and terrestrial vehicles, your company plans to extend their offered systems with a new product line of systems that provide monitoring of potential floods in natural environments. Floods are very problematic in many parts of Europe and the problem is aggravated every year through climate change and extreme weather phenomena. Floods can destroy farmlands and crops and cause massive problems in residential areas, even loss of human lives. The first product of your company in this endeavor will be a system providing flood monitoring; this first product will serve as a pilot product for your company for the next generation of environment monitoring systems, paving the way for systems with more complicated sensors and vehicles and integration with emergency services.

2.2 Basic principle

Your company does not include hardware manufacturing in its product development. Instead it reuses off-the-shelf hardware such as sensors, vehicles, control units and other platforms and builds upon them to offer its products. In the case of your system, your company plans to use hardware that is already available on the market and may offer standard functionality that can be extended to build Flood Monitoring solutions in rivers, dykes, waterways etc. The hardware will be comprised of reconfigurable hardware components that may be inter-connected in different configurations. The development project for your system will, on the one hand, reuse the off-the-shelf hardware platform, and on the other hand develop the software system that controls the hardware in order to implement the Flood Monitoring functionalities.

Your system should offer at least the following functionalities:

- **Monitoring activities**: the system should monitor activities and properties of rivers, waterways, dykes, such as the water level and pressure or the consistency of the dyke. Monitoring can be performed through different devices, e.g. analog and digital sensors, Unmanned Aerial Vehicles (UAVs) and Vehicular Ad-hoc Networks (VANETs) etc.
- **Warning**: in case of an imminent flood, the system should issue warnings to the authorities and emergency services, but also directly to citizens who are subscribed for such messages (e.g. through SMS or mobile apps).
- **Guidance**: the system should provide runtime information to guide its constituents or third parties, .e.g. the UAVs may provide information to VANETs embedded in vehicles crossing the area, thereby guiding vehicles driving towards a flood area to avoid certain routes. Meanwhile, the VANETs can also provide an alternative route to emergency services or citizens to avoid the flood area.

Your system should support at least the following non-functional requirements:

- The system has an inherent emergent behavior and should thus be **resilient**, i.e. faults occurring during run-time such as sensors or vehicles failing, should be dealt with, without affecting the functionality or QoS of the system.
- The system is in fact a system-of-systems comprised of a number of different constituent systems that need to be seamlessly **integrated**, for example Wireless Sensor Network (WSN), UAV, robots, and network routers.
- The **performance** of the system is of paramount importance for its success: an early flood warning resulting from sudden heavy rainfall or extreme weather can determine the success of the system. However the system is comprised of constituent systems of limited memory, battery life and processing power that exchange large amount of data across unstable environments, thus potentially compromising performance.
- The system should **interoperate** with third parties: for example, data should be exchanged between your system and weather forecasting services in order to predict e.g. the development of water rising. Data should also be exchanged with authorities and emergency services that need to deal with floods.
- Your system should provide **correct** flood warnings: it is especially important to confirm an imminent flood warning through different sources (e.g. by sensors in the river and a UAV) in order to avoid false positives (which could seriously undermine the credibility of your system).
- Finally the domain of environmental monitoring is expected to grow rapidly, so your system must be **future-proof**, thereby seamlessly allowing the integration of new functionalities, such as monitoring water pollution or detecting extreme weather phenomena.

2.3 Business information

Your company wants to offer a pioneering product that will revolutionalize the market of environmental monitoring. To this end, it plans to offer your system as a product that will allow early and correct detection of floods. The mission statement of your company is to limit the social and financial consequences of floods and most importantly avoid the loss of human lives.

Your system should be installed in a variety of natural environments. Your system is a visionary system and aims at being a core part of the future generation of Environmental Monitoring systems.

The revenues from your system will be derived not only from the selling price of the product, but even more from the maintenance costs and the services (e.g. future upgrades) that your company will provide to its customers.

The market around environmental monitoring is expected to be extremely competitive, thus your system needs to take into account the existing and the expected competition.

3 Assignment

Develop a software architecture.
In addition to your system, the development includes the interfaces with third-party systems. The architecture has to be developed in an incremental and iterative manner. The details of the architecture are added step by step.

Since an architecture cannot be developed without knowing its environment, the context of the system (chapter 1), and the architectural relevant business information (chapter 2) need to be specified, briefly but effectively. The requirements for the system (chapter 3) must also be negotiated with the stakeholders and decided upon. The hardware architecture (chapter 6) should not be conceived from scratch, but it should be a selection of already existing hardware parts that are able to support the system's functionality and quality attributes. In this way, the project can focus on the software architecture of your system and thus the amount of work is kept manageable within the course.

First, an initial model of the software architecture is developed.
The initial model is a model at a high level of abstraction, demonstrating the *feasibility*, the *affordability* and the potential *risks* for the system to be built. It has to show the essentials of the operation, functional, structural as well as behavioral. The latter implies that **quantitative** analysis is an important part of the rationale behind the initial model and the final software architecture.

The initial model describes the implementation of the architectural significant use-cases in terms of the major sub-systems, their interfaces and interactions.

The initial model should be understandable by stakeholders and comprise also a feasibility study and a risk analysis. For the purpose of this course, the other students and the coaches should understand the way the required functionality and non-functional constraints should be realized. Based on this understanding, it should be possible to refine the design.

Choices that influence functionality and quality of the system should be presented to the stakeholders together with possible alternatives, and with as much quantitative information as possible. If necessary, experiments, measurements, and other information sources should be described.

Based on the initial model, the software architecture will be refined into an elaborated model.
This implies a more thorough description of components, interfaces, component interactions/protocols, quality attributes, and so on. Important design decisions must be documented together with a rationale.

As the end result of the practical assignment, an architecture description must be available that covers the software design including the interfaces of your system with all involved third-party systems, such that a software engineering organization is capable of building the system, without being confronted with "surprises". As this is not always possible, this design should identify the major risks and describe the measures to defend against them.

These issues and the functionality listed in section 2 can be altered by the groups, as long as they justify their changes and subsequent design choices.

4 Points of attention

4.1 Document

- Note that the architecture description is a linear piece of text and does not reflect the way the architect works. Architecting is done in a highly incremental and iterative way and not according to a waterfall model. This implies that an architect works usually on several issues (document sections) in parallel (top-

down as well as bottom-up) and does not elaborate the different chapters one after the other.

- The system architecture document(s) must provide a good overview over the various solutions. Although some details may be very important, never forget that architecting means that you concentrate on the essential issues on a level of abstraction that is as high as possible.
- You may decide to split up the system architecture document in a number of smaller documents each covering a particular aspect of the system. In that case however, it is needed to add an additional document that describes the documentation structure.
- A single system architecture document should have the structure as described in section 5. In case you decide to provide another set of documents they must cover all subjects listed in section 5.
- The documentation structure must be defined explicitly, including overview, and glossary (definition of abbreviations and terms).
- Make a clear distinction between things that are *architecturally relevant* and things that are not. The document should only describe architectural relevant things.
- Important *design decisions* must be explained in terms of *alternatives*, their advantages/disadvantages and a rationale for the final decision. A good way to evaluate the different alternatives is to weigh them with respect to the requirements.
- Usually, it is sufficient to consider no more than 3 design alternatives.
- Design decisions must be made explicit in the text (different font, box, etc.).
- The document must be written in such a way that a reviewer can understand it, i.e. also assumptions, concepts, notations (syntax and semantics), etc. must be documented.
- Use figures and tables to illustrate the essential points.
- Avoid scattering of related information because this makes the document difficult to read.
- Avoid unnecessary redundancy because this makes the document difficult to maintain.
- The final version of document(s) (without appendices and overhead) must not be longer than 70 pages. An indication for the number of pages per chapter, the *page budget*, is given in section 5. Necessary details should be described in appendices. Note that more pages is not necessarily better: it is important to be able to convey the same information in fewer pages. In practice, stakeholders do not have the time to read lengthy documents.
- Change bars must indicate changes with respect to the previous version.
- For all calculations, the **dimension** of the numbers must be mentioned. Also, a clear distinction between bits (e.g. b) and Bytes (e.g. B) must be made.
- The precision of the calculation must be in accordance with the accuracy of the estimations, i.e. don't use irrelevant decimals.
- The *syntax & semantics* of the notation you use (e.g. for diagrams and formal languages) must be clear.

4.2 Architecture

- For estimating dependability features (timeliness, performance, reliability, availability, safety and security), the contributions of all elements (e.g. access

times, bandwidths and execution times) that are involved in the relevant end-to-end transactions must be considered.
- With respect to reliability, special attention should be paid to single points of failure. With respect to availability, the repair times are crucial.
- Note that the form of resource-constraint architectures is mainly determined by the non-functional requirements.
- Start with a *typical-case calculation*. Based on the understanding of the system that you have obtained (and only if it is necessary), you can refine your calculations by considering worst-case situations, using statistical approaches, rate-monotonic analysis, etc.
- The description of the architecture must also comprise the possible *evolution* of the system, i.e. the quantitative and qualitative changes that are envisaged for the future.
- The description of the architecture must be accompanied by a *risk analysis*. Risks are things that can happen outside the sphere of influence of the project (the architect) but may hamper it seriously.
- Make sure that the relation (mapping) between the different architectures (system, hardware and software) is clear.

4.3 Requirements

- It is wise to start with a simple description of the business in terms of product flow, money flow, etc. Such a business description defines also what belongs to the system and what not. It provides guidance for making sensible architectural design decisions. Note, however, that a business description is not a business- or marketing model.
- The *stakeholders* and their *objectives* (also called stakeholder concerns or key-drivers) must be described. Note that the stakeholder objectives (together with the business model) define the goals of the system and give important hints about the essential quality attributes.
- The architectural relevant *use-cases* must be described. From these use-cases, the *functional requirements* can be derived. It is often sufficient to have one high-level use-case diagram that shows all actors. Note that a use case is a description of a piece of externally visible functionality from the point of view of one or more actors (stakeholders or other systems).
- Structure the requirements in form of a *key-driver graph*.
- *Requirements tracing* must be achieved by extending the key-driver graph with the essential design decisions
- Think also about *non-functional requirements* (many of them are related to quality factors). Define them in a quantitative and verifiable way, as much as meaningful.
- The requirements should not have an implementation bias.
- The requirements should be logically structured, e.g. into functional and non-functional requirements, or into user, business, product, operational and general requirements.
- All requirements must be SMART (Specific, Measurable, Agreed, Realistic and Time-bound).
- Add a *rationale* for all requirements that are not directly derived from in the problem statement.
- Requirements should have a *priority*, e.g. must, optional and future (release).

- Also the relevant user interface-, operational- (database size, database actuality, etc.), installation- (including data conversion, the filling of databases, business statistics, etc.), maintenance-, etc. requirements of the system must be considered.
- The requirements should be completed with an assessment of the architectural relevant risks, e.g. in terms of business, technical and operational risks. For every risk, it should be clear what the priority is (dependent on probability and severity); who the owner is; and what the preventive and corrective actions are. Note that the risks drive the architectural process, i.e. what must be done first.

4.4 Analysis

- In the analysis section, important design alternatives should be discussed in terms of their qualitative and quantitative features, their advantages/disadvantages and their feasibility. For this purpose, a graph of the relevant alternatives is a good starting point.
- *Assumptions* must be made explicit.
- All the architectural *design decisions* should be discussed here.

4.5 System architecture and Software Architecture

- The *initial model* is a first draft of the system architecture. It should be described in terms of high-level components/subsystems and their interaction (i.e. no detailed class diagrams). The level of detail must be sufficient to perform an analysis of the feasibility of the chosen alternative. It is useful to start with the simplest system and to refine it until the necessary functionality is covered.
- Several *architectural alternatives* should be considered. For each alternative, an initial model must be worked out, including the most essential design decisions and the advantages/disadvantages of this alternative. The latter is necessary in order to choose the most attractive alternative that will be worked-out in more detail.
- Remember that the architecture can be derived from the design alternatives identified during the analysis and is driven by the quality attributes, like performance, reliability, availability, security, maintainability, usability, testability, etc.
- The *feasibility* of the system architecture must be verified by a quantitative analysis. Think especially about the verification of important architectural relevant requirements (e.g. end-to-end requirement for timeliness and reliability) and business assumptions (e.g. costs and time-to-market). Note that the verification of the feasibility starts with the initial model in order to exclude infeasible solutions as early as possible.
- The description of the system architecture must also comprise the total investments in hardware and software respectively the cost price of the product/system, the installation costs and the TOC (Total cost Of Ownership) of the product/system.
- It might be convenient to document also the system architecture in terms of the 4+1 views model and UML.

5 Document template

Note that this template is only a suggestion and not mandatory. You can work out your own document structure or use other templates (e.g. http://www.arc42.org/)

Change history

Table of contents

1. System context *(max. 1 page)*
   Description of the position of the envisaged system in its environment.

2. Architectural relevant business information *(max. 5 pages)*

2.1 Business vision
   Description of the envisaged business opportunities and unique selling points
   Advantages and disadvantages with respect to the current situation.
2.2 Business rationale
2.3 Product/service description and its evolution over time
2.4 Target audience
2.5 Business/Domain model
2.6 Roadmaps about market development, product introduction, etc.
2.7 Financial model
   Unit price (production & selling), turnover
   Investment, operational costs, profit, break-even-point, etc.
2.8 Competitors

3. Requirements *(max. 10 pages)*

3.1 Architectural vision
3.2 Stakeholders and their concerns
   Customer, users, implementers, configurators, maintainers, resellers, etc.
3.3 Stories and Use-cases
3.4 Functional requirements
3.5 Commercial non-functional requirements
   Product appearance, prizing, etc.
3.6 Technical non-functional requirements:
   Dependability (timeliness, performance, reliability & safety, availability, security), adaptability, scalability, maintainability, interoperability, etc.
3.7 Evolution Requirements
   Typical change-cases with respect to the environment, the features and the technology of the system
3.8 Risk assessment
   E.g. in terms of business, technical, implementation, operational and other risks.

4. Analysis *(max. 5 pages)*

4.1 Assumptions
4.2 Technology roadmaps
4.3 Design alternatives
   Important design alternatives, their advantages/disadvantages, their feasibility.
   The major design decisions must be documented together with their rationale.

5. System architecture *(max. 10 pages)*

5.1 Initial models (several important alternatives should be considered)
Design rationale and summary of the set of alternatives that will be worked out subsequently.

5.2 Elaborated model

5.3 Verification
Verification of important requirements like costs, end-to-end performance in terms of responsiveness and system resource usage, reliability, etc.

6. Hardware architecture *(max. 10 pages)*

High-level, reverse-engineered description of the hardware platform and its application interfaces.

7. Software architecture *(max. 15 pages)*

7.1 Software Architecture Design (Attribute-Driven Design)

7.2 Architectural views (4+1), including patterns and tactics

7.2 Components, including their functionality, interfaces and interactions.

7.3 Decision views

8. Architecture evaluation *(max. 5 pages)*

8.1 Requirements verification

8.2 Architecture evaluation (ATAM, SAAM etc.) & conclusions/improvements.
Use scenarios from Chapter 3 (Requirements) when applying ATAM.

9. System evolution *(max. 5 pages)*

Appendices: Project plan (planned iterations and real ones) like references, glossary and open points

One appendix must show the changes per deliverable as well as the person responsible for a particular chapter or section together with the contributors.

6 Grading

The grading of the architecting project is performed based on the final document version, the process followed and the presentations, according to the following percentages:

- **Business model (5%):** System context, Architectural relevant parameters, Financial model
- **Requirements (10%):** Stakeholders & concerns, Use case descriptions, Requirements, Risk analysis
- **Analysis (10%):** Overall, Architectural issues, Design alternatives
- **System architecture (8%):** Overview, Components and interfaces, Allocation to HW and SW
- **Hardware architecture (7%):** Reverse engineering, Fit in overall architecture
- **Software architecture (20%):** Soundness, Architectural views, Architectural patterns, Decomposition, Component responsibilities & interfaces
- **Architecture evaluation and evolution (5%):** ATAM or/and SAAM, Conclusions/Improvements
- **Evolution (5%):** Anticipated changes and their management
- **General (10%):** Comprehensibility, Balance abstraction/detail, Arch. concepts & Design decisions
- **Process (10%):** Teamwork & meeting deadlines, Motivation, Initiative and Creativity, Following feedback
- **Presentations (10%):** content and style, answering and asking questions