COVERING ARCHITECTURAL PROBLEMS OF SOFTWARE DEVELOPMENT OUTSOURCING

Aziz Ahmad Rais
University of Economics, Prague, Faculty of Informatics and Statistics,
Department of Information Technologies
W. Churchill Sq. 4, 130 67 Prague 3
arais@seznam.cz

ABSTRACT:

In the modern world, developing information systems has become more difficult than ever before, and without properly proposed and agreed architecture it is impossible to develop and outsource software development projects. To outsource software development it is necessary to have the tools and language to communicate the goals and business needs. Software architecture is the right tool to prevent some types of failures and to provide the right means for communicating the shape of the end product. This paper will cover how to develop and proceed with successful design of architecture using an agile methodology.

KEY WORDS:

Outsourcing, architecture, agile development

1 INTRODUCTION

It is important to understand why architecture is crucial for outsourcing software development. A successful project needs teamwork to prevent and reduce the risks of failures. It is very important that teams communicate with each other in order that the components and the interfaces developed by different members of the team interact with each other without bugs and errors. To prevent errors and bugs it is necessary to describe and model the system.

However, there are other considerations to take into account as well. Applications should be modular, flexible and should have layers for better separation and organization of components. During the post-outsourcing life cycle there can appear other types of problems with applications, like maintainability and extensibility, among others.

All these problems need to be discovered during the outsourcing phase of software development and solved or eliminated by using software architecture.

2 ARCHITECTURE OF INFORMATION SYSTEMS

Architecture of information systems is a concept that many people either do not understand or expect a lot from. Some people do not even understand its importance. It is worth describing what software architecture is, why it is good to define the architecture of a system before developing the system itself and what would happen if it were not defined.

Information system architecture is about collecting different views from different perspectives like business, infrastructure, technology, logical view, data flow, data domain and processing or interaction. Each of these views describes how the system looks from the corresponding viewpoint. Each view then defines the set of components that makes the system serve the request. Each view provides a specific collection of logical components organized through the layers and tiers, and describes specific interactions between the components.

All the information given by these views are useful for many aims, e.g. for better troubleshooting, support, implementation responsibilities, outsourcing and deployment.

Defining the architecture is important in order to meet non-functional requirements that are important for the life cycle of the system. Some of them include maintainability, extensibility, security, performance, portability, platform independent and scalability.

Some of the non-functional requirements are difficult to measure. It is necessary to write a criterion that will help us to accept the system being developed by a third party. A criterion that is not defined correctly can sometimes limit developers during development. Having unclear criterion needs negotiation and allows exceptions causing a delay in software delivery.

Although information system architecture can be specified from different viewpoints, it is still important to identify the level of the details in information system architecture. Today's information systems consist not only of one component or service but of many more. Examples could be the construction of a data warehouse or the distribution of content for IPTV.

For designing information systems it is useful to distinguish three levels of abstraction and granularity to make agreement and expectations from the proposed architecture easier for both client and vendor.

1. **Enterprise architecture** describes the complete IT system of a company along with the full picture of infrastructure, systems and business services.
2. **Information system architecture** describes the system components and their interaction and the services that the system provides. These components are organized in different tiers and layers.
3. **Information system design** concerns applying design patters to the components and their interfaces described in the higher level.

Outsourcing software development is about developing an information system by a third party. This article describes the second level of information system architecture and identifies how to achieve and design such architecture.

The first step of developing the architecture of an information system is gathering all the necessary requirements. This first step can be divided into two types of requirements.

Functional requirements should describe what the system should do in order to provide a service to the end-user. Usually these requirements can be gathered in the form of use cases, but there are also other methods and standards like BPMN or using sequence or activity diagrams. These requirements can be designed by means of tools like *Enterprise Architect* or *Magic Draw*. Describing the scenarios by means of text documents is also possible.

Non-Functional requirements are usually considered as technical requirements, but they can stem from functional requirements as well.

2.1 Architecture development methodology

This article will not cover business requirements gathering and analysis methodologies. Nevertheless it should be carefully considered what methodology to use to develop the architecture as part of the project. However, before developing the system architecture we should first develop the guiding methodology. So far, the agile methodologies have proven themselves as sufficiently effective and robust approach based on iterative software development.

According to the Rational Unified Process (see [2]), software development consists of four phases: inception, elaboration, construction and transition. From the outsourcing point of view the transition phase is not of particular interest. The construction phase includes also quality assurance activities since they are tightly coupled with the implementation process. Moreover, splitting the elaboration phase into the architecture and the design phases has been proven advantageous, for instance, it facilitates an early planning of iterations. Because term

inception, coined by RUP, has not become widely used, I decided to call this phase analysis, since it better describes the main activities performed within it.

In principle, there are four possible start points from which to begin planning iterations.

1. **Beginning to iterate in the requirements phase:** The business analyst, project manager and client together should prioritize the requirements. Based on this prioritization, the requirements should be divided into iterations.
2. **Beginning to iterate in the architecture phase:** This approach assumes the business requirements have already been gathered and analyzed. Then, the software architect and the project manager plan the iterations based on business requirements, component dependencies, technologies used, layers and tiers. The architects deliver their architecture to the designers in iterations so that they can design the components.
3. **Beginning to iterate in the design phase:** This approach assumes the business requirements have already been gathered and analyzed, and the architecture has been developed. The designers deliver their design per component or per set of components and layers to the developers. The software architect, designer and project manager plan the iterations in this approach. Of course, other parties can be involved too, but they are optional and need to have technical knowledge.
4. **Beginning to iterate in the implementation phase:** All the previous phases of the development life-cycle have been already completed. The iterations in this approach are based on the delivered design, architecture and business requirements. The implementation of the whole system is built upon the results of the previous phases.

The later we start with the iterations the farther we are from the real agile methodology and the closer we get to the waterfall one. The earlier we start the bigger is the chance that we identify all types of failures sufficiently soon.

Therefore my suggestion is to start with the iterations in the analysis phase. There is often a need for different insights, because describing all the information in one view and diagram would make the architecture unclear and impossible to describe.

One of the most important rules for the architecture is to be sure that our proposed architecture meets all the requirements. Therefore you need to know the scope of the system and all the requirements. If you are missing some requirements, you should take the role of the client into account and make appropriate assumptions. Sometimes you can reveal new requirements. It is also very important to keep in mind that the development of the architecture requires a consistent terminology, naming conventions and views. Let’s define the type of information the views can provide to the viewer. The 4+1 view (see [1]) is limited and doesn’t cover how to solve problems connected with some of the views. It is therefore difficult to say where and how these problems can be solved, and it is not clear whether the 4+1 view allows solving them. The sole usage of the 4+1 views makes solving all the architectural problems difficult.

Let’s outline briefly all views through which the architecture can be described.

1. **Business view**: This view describes the business requirements in general. The analysis of the requirements is out of the scope of this view. This view contains a list of business entities and the relationships between them (business entity model). This model is not meant as a physical model describing how the data are saved to a persistent storage, instead, it focuses on identifying the business; in other words, how the business data should be put together. All the entities and relationships between the entities are parts of the entity model.

2. **Logical view**: This view describes the layers, tiers and important components. All the business and technical components (and possibly others constituting the information system) should be identified in this view.
3. **Technology view**: This view covers technologies, frameworks and third party components as well as servers used for building the system under development. This view can be described in the form of a diagram that exactly maps each component and layer in logical view to the corresponding technology.
4. **Process view**: This view describes the interaction between layers and components and it is used to describe all processes. In order to avoid a duplicate description of processes it is recommended to describe only one representing the group of similarly described processes.
5. **Integration view**: This view describes how to integrate all external systems. In this view we concentrate on identifying design patterns, technologies and ways of integration (e.g. batch-like, synchronous or asynchronous). Here, it is also necessary to consider message formats, data types mapping and support of the system after deploying it to the production.
6. **Domain view**: This view identifies the object domain model based on the entity model. The domain object model describes a representation of the business entity model as an object model. (The physical data model is created in the data view.)
7. **Data flow view**: This view is useful when building a complex system composed of the services provided by other systems. Here, the data formats are described along with specifying sources and destinations for the identified data.
8. **Physical view**: This view shows how the system will be deployed, and shows the runtime environments of the system. This view has to provide the sizing of physical environments, how to integrate the whole runtime environment into the network topology. This view also includes a description of how the system is secured from the network point of view, how disaster recovery is solved, how the scalability is achieved, how those data is archived and how the data back up is performed by the information system. This view can also describe how to monitor the network and the hardware of each system. Furthermore, this view determines security solutions for low-level components (the operating system, the hardware and the network), denial of service, etc.
9. **Data view**: This view describes how each business entity will be persisted into a persistent storage. The physical view can be done conceptually only and the details can be added in the detailed design phase later on.
10. **Implementation view**: Here, all the business components are identified based on the functional requirements. The flow of interactions between business components follows the process view. The implementation of the business components is done iteratively. New business components can be added in each iteration and the interaction between the components is depicted by sequence diagrams in order to show how the functional request will be processed.

2.2 Agile software architecture development

Another principle of the agile approach recommends doing some activities of one phase in parallel with activities from other phases. In other words, we start the next phase of the development life cycle before the other phase is finished.

After the business requirements are gathered and before the analysis of the requirements is finished, you can do the following on each view:

1. **Business view**: In this view we can describe high-level use cases. The exact number of use cases depends on the business requirement analysis.
2. **Logical view**: In this view we can identify the components that support the business, the supportive components, e.g. for logging, caching, monitoring, configuring, authenticating, access control, auditing, filtering, tracing and validating. You can also identify the layers and the tiers.
3. **Technology view**: In this view a discussion on technologies can take place even before the requirements gathering completes. Also, based on the logical view, the technology used in each layer and tier should be identified. For each technical component, an adequate framework or third party components should be identified.
4. **Process view**: In this view the process between layers and tiers can be specified. The process is actually performed by many components inside the layers, but it is very important to underpin here only the complex processes, and to show the entry and exit points of each layer and tier for the others.
5. **Integration view**: In this view the analysis of external systems, their data and message types as well as availability of their interfaces should be done.
6. **Domain view**: In this view we propose the first draft of the domain object model.
7. **Data flow view**: In this view we describe the data formats along with the sources and destinations for the identified data.
8. **Physical view**: In this view we can be identify available hardware resources and the network bandwidth as well as how the cluster and disaster recovery will be solved.
9. **Data view**: In this view, we identify the possible data model and the conceptual tables and relationships.
10. **Implementation view**: This view cannot be started before the business requirements analysis is finished. Here, as mentioned above, we need to describe the business components.

What to do after the business requirements analysis is ready? After the analysis review is done the check whether all requirements have been met should be performed. Then we finalize the architecture and specify which business components should be implemented per iteration.

3 SUMMARY

First, the paper distinguished the three levels of abstraction and granularity to make the agreement and expectations from the proposed architecture easier for both client and vendor. From the four phases defined in RUP, i.e. the inception, elaboration, construction and transition, the elaboration phase was split to the architecture and design phase for facilitating early planning of iterations. The four possible start points from which to begin planning iterations were identified and analyzed with the recommendation to start the iterations as soon as possible.

To design the architecture better it was suggested to describe the system under development from ten viewpoints. Each individual view was briefly described along with the corresponding processes. Thus, when outsourcing software development, this architecture development methodology provides the guidance for negotiating the solution, which the vendor should deliver. Analogously, the customer can check the software deliverables more efficiently and eventually reject or accept the deliverables.

REFERENCES

[1] Philippe KRUCHTEN: *Architectural Blueprints—The "4+1" View Model of Software Architecture*. IEEE Software, 12(6), Nov. 1995, pp. 13-16.

[2] Philippe KRUCHTEN: *The Rational Unified Process: An Introduction (3rd Edition)*. Addison-Wesley Professional, 2003. ISBN 0-321-19770-4.