OUTSOURCING OF SYSTEM INTEGRATION DEVELOPMENT

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

ABSTRACT:
Many companies outsource the development of system integration as well as development of any information system. There are different risks connected to the outsourced development of system integration. This essay will describe and cover only architectural risks of outsourced development of system integration. This essay will analyze these risks by using agile development methodology. The proposed solution will be based on architectural design patterns, system integration design patterns and agile development methodology.

KEY WORDS:
System integration, design patterns, system integration architecture, iterative process.

1 INTRODUCTION

Since 2003 we have been involved directly and sometimes indirectly in outsourced software development. Most of the outsourced application development had similar problems. By mentioning these problems, most of the risks and the goal of this easy will become clearer. One of the common problems of out-sourced software was that the integration with external systems was handled and solved as an integration component. Such integration component sometimes had no interface that will decouple the implementation from presentation. The integration component was part of whole software development; there were cases like missing what message and data types this component should operate on. Sometimes even the external systems objects were not transformed and directly used in the application. Using external systems data objects cause that the whole software becomes dependent on external system and any change in the external system caused a lot of work for the whole software. Of course, using external data objects cause that the software extensibility and flexibility is also affected.

Other very important common problem is that it is good to mention that the integrating component was not developed as one package. Access to the implementation of integrating component by other components of the software was in no way limited. This caused that system integration was difficult to secure and assure that the external system receives the messages. As last, all such system integration caused that system performance was also affected.

Why these problems happened and could they happen to outsourced development of system integration?

There are many reasons here, but the most important ones can be identified according to development life cycle; they are the following:

1. **Project budget (fixed price):** One of the advantages of outsourcing is finance [1]. That is why the fixed price outsourcing projects can meet this expectation. Therefore the non-fixed price outsourcing development won't be covered and analyzed in this article. The difference between outsourcing and in-house development estimate is that the in-house development budget can be updated periodically. In order to make estimate of outsourced project budget there is need for estimation of development of

system integration. Exact estimate of development before the project starts and in the beginning of the project is not possible as there are not available enough details of the project, business scope, technical requirements, and technical solution. Besides that there are unpredictable variables that effect the development too. These variables affect the final estimate as well. They could be identified based on life cycle phases of system integration [2], for example:

   a. **Systems compatibilities**: Integration of different systems needs knowledge about their compatibilities of service interfaces, technologies, protocols, data types, validation of data that the systems require. Analysis of all these compatibilities and calculation of their impacts on development takes a lot of time.
   b. **Proposed solution**: usually someone has to make an estimate of solution of the system integration. If he/she estimates development without the proposed solution there is a big risk that the project goes out of budget or he/she has to reduce the quality of the system integration.
   c. **Technology issues**: There is a number of components and frameworks - commercial as well as the open sources. That makes very difficult to predict all the problems for the selected technologies to work for the proposed solution.
   d. **Business requirements**: It is very difficult to make complete business requirement analysis to identify the scope of the project. The scope of the project is important for budget estimates. So during development of the system integration the requirements can change, which affects time for development. It has an impact on total budget.

2. **Lack of specialists**: Outsourcing companies very often have a lack of specialists or they do not want to invest in hiring of them. Both cases cause that the outsourced development of the system integration is done by non-experts. In case of in-house development, companies can hire additional resources or postpone software delivery date. Outsourcing companies are limited by budget, SLA, sometimes by a strategy or global cost saving.
3. **Lack of processes/methodologies**: Accurate and precise process for outsourcing of software development does not exist. The processes like RUP and agile development methodologies are good for in-house development. This is due to the fact that none of these methodologies cover how to divide the responsibilities and roles of software development between customer and supplier. These development methodologies do not explain also how to divide the iterations when review of software quality should happen and there is a missing relation of process to development actions. For example: what part of architecture should be built up in which iteration or phases. Some of these methodologies do not recommend the building software architecture and design. And none of them propose any lightweight process and methodology for developing software architecture iteratively. Also, none of these methodologies describe how to use design patterns in the development process.
4. **Lack of requirements**: In practice it usually means having ambiguous requirements or having vague requirements. Sometimes what happens is that there is a missing clear vision of a business or a business strategy which affects exact requirements defining. In case of in-house development during the development the requirements usually change and the project delivery date can change accordingly. The result of any unclear requirements is that the software cannot be tested and reviewed.
5. **Lack of software architecture**: Software architecture in outsourcing of development of system integration is missing for different reasons. The reasons can be: the project budget because such activity is not included in the budget, lack of specialists, lack of processes or a lack of requirements. Other very important reason is that there is a

missing lightweight methodology that would help even an inexperienced architect to propose a solution and prevent or limit the above problems. However, it can happen that in the in-house development is a missing software architecture. Nevertheless, this is usually covered by a specialist to watch the implementation of a system integration development.

6. **Lack of design**: According to some development methodologies the system architecture is a high level design of software and the design is a detailed solution of the software architecture. Sometimes design is understood also as a meta-model (for example: UML class diagram) of implementation of the software. But this last definition will not be described in this article. Many developers still do not understand how to utilize the design patterns or they still do not understand the advantages of them.

This article will further analyze and cover topics such as a system integration software architecture, design and lightweight process for system integration software architecture building iteratively.

Why lightweight process is better than the existing one?

The existing processes and methodologies are very general and open. It is because one has to be able to apply them on any type of software development. For system integration development outsourcing it is needed to have more precise processes and recommendations that limit developers from selecting from many options. This will help to quickly propose the architecture for outsourced system integration software and design. What is a good base for a budget estimate and the architecture will not be a discipline for senior specialists anymore.

In this article, a solution for the system integration based on view driven architecture will be proposed. Many authors try to solve system integration design patterns instead of system integration architecture and how and where in system integration architecture to use these patterns. We would like to propose a different way which will show how to build system integration architecture and where to use these design patterns. The details of the design patterns themselves will not be described, rather the links will be provided in the reference list at the end of this study.

2 SYSTEM INTEGRATION ARCHITECTURE BUILDING METHOD

System integration architecture can be seen as system architecture of any other multilayer application. According to multidimensional principles of MMDIS [3] every problem needs to be analyzed from different views. System architecture 4+1 view [4] describes views that are not enough for system integration architecture. That is why the important views of system integration architecture will be identified and it will be described what types of design patterns could be used in each view.

One can assume here that the business requirements are gathered and analyzed. It is important to have identified goals that the system integration should achieve. Some technical requirements can be defined or assumed as quality models and quality criteria in order to meet them in the final system. The ISO product quality model (Systems and software Quality Requirements and Evaluation (SQuaRE)) could be an example of such quality model [5]. Of course, according to IT governance (COBIT) [6], every IT goal should be mapped into the business goals; so also the SQuaRE model should be mapped into the business quality model.

2.1 Logical view

This view describes whether the integration is peer-to-peer, one-to-n or n-to-m and also the style of integration.

The goal of this view is also to describe the layers and tiers of the integration. Because layering is an architectural pattern, therefore there should be also other design patterns used like SOA and ESB or EAI [7]. Logical view describes big picture of the whole integration system and can be applied also in IT strategy of the company. As one of the important aspects of IT strategy is the resource management and applications. Existing services are resources, therefore this view describes what services and systems will be reused. Based on this view one can also make a rough estimate of the project and can also show how the implementation team will be created.

Sometimes this view can also contain main conceptual business components. Business components are understood as functionality that needs to be implemented and they will be converted to technical components by component view.

2.2 Component view

Component view describes all the components in each layer and the tier of the integration architecture. Example of components can represent such design patterns like:

Pipes and Filters, Message Router , Message Translator, Point to Point Channel, Publish Subscribe Channel, Dead Letter Channel, Message Bus, Content Based Router, Message Filter, Recipient List, Aggregator, Resequencer, Routing Slip, Throttler, Delayer, Content Enricher, Content Filter, Messaging Mapper and so on. Because the design pattern is a proven solution of specific and repeatable case and the implementation of each design pattern can be some component, the best place for identification of components are integration design patterns [8].

2.3 Process view

This view is very important to describe how each tier, layer, and component interacts with each other. Also, here will be described: when the message is going to be transformed, when the data types are mapped and how they are mapped, if a message is persisted in order to service crash and how message delivery is assured. Every subscriber gets only one copy of the message. In this view it is also made clear whether the communication is synchronous or asynchronous or integration is batch-wised or real-time.

2.4 Data flow view

This view is about to find out what types of information are taken from what system and used for what purposes by which consumer. Also, here will be described the direction of the data streaming through components. It is important to describe the data format and provide a mapping list of transformation and mapping between systems data and their types.

2.5 Deployment view

This view will describe how each logical component, layer and tier will be deployed into infrastructure and network. It is to identify the relation of logical components described in components view to the infrastructure and network. This way we can see the runtime behavior of the whole system integration. This view also makes clear the high availability, failover, load-balancing, and security of the system integration.

2.6 Technology view

This view should solve the technology selection problem of the system integration. Of course, it is important to know all information about the logical view, physical view, and components view due to a fact that this view supports the system integration implementation.

The identification of all technologies that reflects the logical view layers, tiers, protocols and frameworks belongs here too. The difference between this and the deployment view is that hardware, network, operation systems of low level layers used by application systems for communication with each other are not part of technology view but the deployment one. This division is important because every view will be managed by different IT specialist. The use of both views by system integration is also different; technology view directly affects system integration while deployment one rather indirectly.

2.7 Implementation view

This view is going to describe how to package or group all the components and identifies how each component is constructed. It means identifying parts of each component and their associations, aggregation, inheritance and other object oriented features. The object oriented design pattern for identification of the relation between components can also be used in this view. An example of these design patterns is the gang of four [9].

3 PROPOSED PROCESS

Developing system integration architectural view can be achieved in many ways. However, the optimal one is to iterate on all the steps of integration method - see chapter 2.

Having the process completed we need inputs; assume the inputs are prepared in other phases of the system integration project before starting the elaboration and construction phase. Assume the project has four phases: inception, elaboration, construction and transition [10]. The main inputs for this process are:

1. Business strategy
2. Business goal of the system integration
3. Business analysis (use cases).

The output is quite clear and it is not necessary to explain that - it is a proper architecture document. Due to responsibility of the identification the actor is an architect.

Loop 1{

1. Logical view: To describe this view there is a need to follow the following process:
    a. Take the business goals and convert them to the system goals.
    b. Identify what services the system integration should provide.
    c. Identify all the systems that provide services that are important for building new service.
    d. Identify tiers and layers of the system.
    e. Identify how each layer and tiers communicate with each other.
2. Component view: As described above, this view describes technical components of the system integration. Here we need to find out how to identify these components.
    a. In order to identify the components there is a need to analyze the business provided by the current service. The result of such analysis is that it sometimes describes the ad use case based on RUP methodology [10].
    b. How to reuse current services, that means what data the services provide and which data are needed to extract and how these extracted data together are merged and provided to the new service.
    c. How to interact with the current system.
    d. Based on this information, reuse the proven solutions like design patterns.
3. Process view: In order to identify how the components will interact with each other and their dependencies we should follow the following steps:
    a. Identify all the use cases based on business requirements.

    b. For each use case there are already identified components in the component view and identify interaction between the components for each used case.
    c. Skip use cases that have similar scenario of interaction, in order to have effective documentation and architecture.

4. Data flow view: to identify this view we need:
    a. To identify all the data needed for the new service.
    b. All the systems that provide the needed data.
    c. Identify format of the data that are needed in order to identify the interaction message format between the services.
    d. Identify the data flow direction.
5. Deployment view: For this view the important steps for describing are:
    a. Identify whether the system should be in cluster.
    b. Identify whether the system use load-balancer.
    c. Identify whether the system should use database.
    d. Identify the system requirements on hardware.
    e. Identify hardware size.
    f. Identify the location of the hardware and under which domain they will be placed.
    g. Identify how the whole system will be secured; whether the system will use external authentication system like LDAP, internal authentication components, access to the system from different networks.
6. Technology view: This view is about mapping of layers, tiers and sometimes components to technologies, so to describe them we need minimally the following steps:
    a. Identify list of technologies for each tier and layer.
    b. Identify the list of technologies, frameworks or open a source component that matches the components view.
    c. Analyze all alternatives or combination of technologies, components and evaluate the best alternative. A brain storming method could be used here or any other quantity-based method.
    d. Make proof of concept of the best or optimal alternative.

Loop 2 {

7. Implementation view:
    a. Identify how each component will be packaged.
    b. Identify the physical interfaces of each component.
    c. Identify how implementation of each interface will be invoked.

}//END of loop 2

}**//END of loop 1**

Every loop means that we can iterate over all views and the sequence diagrams describe that when we can start and with what type of view. This means that we can work parallelly on some views and as you can see in the diagram, we can identify the components in iterations and every change in components view can affect all other views except the logical one.

In order to achieve optimal process there is a need for correct business requirements engineering process, and change management process [11]. Both of these processes are needed in order to better identify the functional defect and implement the change in new iteration and release of the system.

4 SUMMARY

The paper was assumed that there are different problems with system integration when outsourcing the software development projects. The main goal of this paper was to discuss the architectural risks by identifying architectural and design problems.

Many ways how to prevent the problems could be identified; the article proposes a solution through identifying the big picture of the system integration. The big picture means describing the architecture of the system; use the appropriate design patterns to describe what should be integrated with what and how. Describing and using architecture and design patterns are the right media for interpreting business requirements into technical and model requirements in diagrams. Software architecture has some similarity with architecture of buildings and machines. The similarity is that in order to better imagine the end product we need to see a building or machine from different insights. That is why for better understanding and for reducing problems and misunderstandings there was proposed an integration architecture method with different views. In order to better understand each view there was a need for providing overview of architectural and integration design patterns that are used in each view.

Finally this paper provided a process that makes the building of the architecture and design easy and cost effective.

Based on this paper the recommendation is that every system integration project has to separate integration layer. It does not matter whether it will be deployed on middleware or together with business logic of system consumer. The reason for this is that this layer can be easily decoupled from the system and deployed on middleware in case of the need for extending the system integration to n-m integration type.

LITERATURE

[1] VOŘÍŠEK Jiří, BRUCKNER Tomáš: Outsourcing IS/IT z hlediska zadavatelského podniku. červen 1998

[2] VOŘÍŠEK Jiří: Systémová integrace na prahu nového tisíciletí - čtyři základní koncepty systémové integrace. září 1999

[3] VOŘÍŠEK Jiří: MMDIS - principy a konceptuální modely, 2001 září.

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

[5] International Standards for Business, Government and Society (ISO): http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=35733

[6] VOŘÍŠEK, J. a kol.: Principy a modely řízení podnikové informatiky, Oeconomica Praha 2008. ISBN 978-80-245-1440-6

[7] CHAPPELL Dave: Enterprise service bus, O'Reilly June 2004, ISBN 0-596-00675-6.

[8] HOHPE, G. -- WOOLF, B.: Enterprise integration patterns: designing, building, and deploying mes-saging solutions, Addison Wesley October 10, 2003 ISBN 0-321-20068-3

[9] FREEMAN Eric, ROBSON Elisabeth, BATES Bert, SIERRA Kathy: Head First Design Patterns. O'Reilly Media October 2004

[10] KRUCHTEN Philippe: Rational Unified Process, The: An Introduction, Third Edition, Addison-Wesley Professional December 10, 2003 ISBN-10: 0-321-19770-4, ISBN-13: 978-0-321-19770-2

[11] HATZIKOU Melpomeni, AGIOVLASITIS Iraklis-Panagiotis: Leveraging ICT Deployment and Integration in a Public Organization Aged 176 Years A Greek Case Study. http://www.jucs.org/jucs_16_8/leveraging_ICT_deployment_and/jucs_16_08_1102_1116_hatzikou.pdf