Tackling Offshore Communication Challenges with Agile Architecture-Centric Development

Andreas Kornstädt and Joachim Sauer
Software Engineering Group, Department of Informatics, University of Hamburg and C1 WPS GmbH, Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
{ak,js}@c1-wps.de

Abstract

Offshoring is not as popular as it seems. According to a recent German survey, only 1.5% of all outsourcing activities target offshore locations. This is a remarkably small figure taking into account the widely published purported benefits of offshoring. In this paper we demonstrate that communication problems are at the core of offshoring woes. This does not come as a surprise as they also play a major role in onshore projects. Based on our experience in tackling these challenges with our well established communication-centered agile design and development approach, we present case-study-reinforced advice for successful offshore projects. We show that a common view of the underlying architecture is of paramount importance for these projects.

1. Motivation

Offshoring is a dominant trend in software development with annual growth rates of about 33% in markets such as India [1]. It promises benefits in the areas of costs, flexibility, and concentration on core competencies. Empirical studies have shown, however, that offshoring also entails a considerable number of challenges. These are language and cultural barriers, skill mismatches, and quality discrepancies.

Against this backdrop it is not surprising that only 1.5% of all outsourcing activities target offshore locations, according to a recent German survey [2].

We assume that offshore development projects can benefit substantially from the integration of architecture-centric development by compensating for their common shortfalls. In this paper we will present a case study with onshore and offshore teams that we conducted to substantiate our approach.

In the following section of this paper we will demonstrate that communication is a core challenge in offshoring. We will continue by examining how communication problems can be solved by means of architecture. Then we present findings from our case study. Finally, we will sum up the essence in the concluding section.

2. Offshoring benefits and offshoring challenges

Clearly, the dominant expectation of corporations that outsource (parts of) their IT is cost saving [3]. While there are other factors such as increased flexibility, none of these factors comes close to the 90% mark that is reached by cost benefits.

While past studies used to focus on benefits, recent studies have also examined challenges that offshoring entails. These include unexpectedly high costs for infrastructure, communications, travel and cultural training; lower productivity due to high staff turnover at the offshore site and low morale at the onshore site; management problems due to cultural differences and a poor spread of information; problems when communicating with customers; and technical mismatches of all sorts [4, 5, 6].

When faced with these problems in an unsorted and condensed form as above, they appear to be very hard to tackle. It helps, however, to examine how these problems interrelate. This leads to a distinction between problems on different levels where the problems at the higher levels are direct consequences of problems at the lower levels.

Primary or *root challenges* stem directly from the decision to outsource to an offshore location:

- Morale at the onshore site is low.
- It is difficult to develop a team spirit that spans two sites. Sharing the goals of the project, expectations, and domain-specific as well as technical knowledge is not easy.
- Onshore and offshore staff comes from different cultural backgrounds. This entails various kinds of misunderstandings. Different views about how to deal with the role of authority make management an especially hard challenge. Direct communication between the

customer and the offshore site can make these problems stand out in a very pronounced way.
- Transferring data to and exchanging data with an offshore site usually reveals technical incompatibilities of some sort.
- Serving as an offshore development center for many different distant corporations, there is often a high staff turnover at the offshore site which exacerbates all other primary challenges above.

When the following measures are taken, they constitute *secondary challenges* in their own right:

- travel to establish as much face-to-face contact as possible
- cultural training for onshore and offshore teams
- additional planning to accommodate the lack of direct communication
- technical harmonization

All of these measures eventually lead to *tertiary challenges* which directly affect balance sheets:

- unexpectedly high costs
- lower than expected productivity

With this distinction between primary, secondary and tertiary levels in place, it is obvious that it is advantageous to start tackling the five challenges at the root level before proceeding to derived ones.

Software related technologies cannot do anything to ameliorate problems in the area of morale and they certainly cannot bridge cultural differences.

Of the remaining three challenges, technical incompatibilities pertain to infrastructure software exclusively and not to the software under development proper, so they are out of the scope of this paper.

Both of the final two challenges (sharing knowledge of any kind, facilitating staff changes at the offshore site) are about communication about the software under development – in the first case between onshore and offshore locations, in the second case between staff members at the offshore site.

Based on our experience with software architecture and modern development approaches we will suggest a way how to considerably improve communication with the help of these two means. To do this, we first present our view of software architecture.

3. Architecture as a means of overcoming communication challenges

3.1. Architectural terms and concepts

The ANSI/IEEE Standard 1471-2000 on the Recommended Practice for Architectural Description of Software-Intensive Systems defines *software architecture* as "the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution."

This entails that every software system has an architecture – at least implicitly. Making this architecture explicit and taking special care of its definition, usage and evolution leads to a development style that is called either architecture-based or architecture-centric.

According to Bass and Kazman [7], *architecture-based development* "differs from traditional development in that it concentrates on driving design and maintenance from the perspective of a software architecture. The motivation for this change of focus is that a software architecture is the placeholder for system qualities such as performance, modifiability, security, and reliability. The architecture not only allows designers to maintain intellectual control over a large, complex system but also affects the development process itself, suggesting (even dictating) the assignment of work to teams, integration plans, testing plans, configuration management, and documentation. In short, the architecture is a blueprint for all activities in the software development life-cycle."

Architecture-based development thus facilitates communication by improving comprehension through one common object of work that all participants use and understand. The architecture description introduces terms and concepts that serve as a common language for all stakeholders. Hence it enables precise discussions and arrangements.

Further benefits for communication through architecture-based development arise in diverse areas:

Task assignment

The architecture helps in identifying components that can be developed largely independent by teams at distant sites. So the organization can be split along the product structure [8], reducing the need for inter-site coordination.

Construction

Because architecture serves as a blueprint for developers, it constitutes an abstract description of the application and thus the basis for verifiable architecture rules. Automatic rule checking improves implementation consistency and reduces the number of errors.

Record of experience

The architecture reflects technology choices as well as the concepts and artifacts of the application domain. In doing so, it establishes a firm basis on which matters pertaining to these issues can be discussed.

Architecture can also be seen as a means to record design rationale. By using similar architectures in similar projects, much of this knowledge may be reused, minimizing the demand for other forms of documentation or direct inquiries to the original developers.

Evolution

Good architecture anticipates future changes to the application. It establishes variability at certain locations of the code which are sometimes called hot spots. By clearly identifying these hot spots, architecture greatly improves the chances that developers will find the right point for extending the code without picking a "wrong" location. Once explained, the hot spot concept greatly minimizes the need for additional communication because developers already know that they can extend the system at these points without asking for permission or detailed instructions.

3.2. Underlying model architecture

Sharing an architectural vision is facilitated significantly by having one's architecture foot on a common model architecture:

"A *model architecture* abstracts from the characteristic features of a set of similar software architectures. It defines the kind of elements used, their connection and the rules for their combination. In addition, a model architecture includes criteria for the composition of elements into modeling and construction units, and guidelines for the design and quality of a specific architecture." [9]

Model architectures serve as broad templates which can be refined into more specific, project-specific architectures. They can be perceived as reference architectures which need to be simple to remain comprehensible. Model architectures are valid within a certain domain-specific or technological parameters only. If these parameters are predominantly of a domain-specific nature, they can be seen as domain-specific software architectures.

3.3. Metaphors and the Tools & Materials approach

Although having (model) architectures significantly improves communication between project stakeholders, it still leaves some potential untapped. This becomes clear when one compares different architectures. When establishing our Tools & Materials approach (T&M) [9], we found out that model architectures are more comprehensible when they are based on metaphors. Metaphors provide a very high level of abstraction which is ideally suited for a field that is governed by a high degree of complexity. Without reducing complexity to meaningless statements, metaphors are very compact ways of throwing light on specific aspects of an issue.

The main metaphors of the T&M approach are Tool, Material, Automaton, Container and Working Environment. These metaphors have the benefit that they are so basic that every customer and every developer has a precise idea of what a tool is like and – equally important – what a tool is not. So everyone can communicate based on this common basis. Misconceptions are reduced.

While the difference between a tool and a material appears to be obvious on an everyday level, it isn't when it comes to implementing them. To establish a clear guideline for developers, there is a small set of behavioral and implementation rules for each metaphor.

For example, the question whether a tool or material is responsible for providing material-related display information is settled in such a way that the material merely provides methods to query it for its state which the tool in turn uses to produce a (graphical) rendition of the material the user is currently working on. Limiting the almost endless possibilities for implementing a system is greatly facilitated by using a (model) architecture. Reducing options in such a way does not unduly limit the stakeholders' flexibility as the can still choose what kinds of tools, materials, etc. they want to populate their system with and what their features should be. However, if they decide to have a certain set of tools, then they have to be implemented according to the given guidelines.

The beneficial effects on communication can be further improved by casting model architectures into frameworks. Based on our experience with the T&M-based JWAM-framework we have already demonstrated that we could significantly expedite the model architectures' reuse [9].

4. Case study

During four months two teams developed a prototype for an order entry and customer information system. The teams consisted of up to six onshore developers at Hamburg, Germany, and six offshore developers at Pune, India.

4.1. Setting and process

The development was based on the T&M approach with adaptations for dual-shore offshoring, using the JWAM framework. The elicitation of business requirements and iteration planning was done by onshore analysts with the customer. Onshore developers built a core system during the first iteration while instructing two offshore colleagues on site. These returned to India after the first iteration and established a developer team there, consisting of about half a dozen members. In the following iterations, offshore and onshore developers worked in parallel, with the onshore developers concentrating on work that required customer interaction and deeper architectural know how. Unit tests were developed offshore in a test first manner. Quality assurance was carried out onshore before integrating components developed offshore.

A software architect was responsible for the initial design of the architecture and for quality assurance. Advancements of the architecture were done autonomously by the developers and checked weekly by the architect who also maintained the central architecture description. The architecture description documents were shared with the offshore team after updates.

4.2. Findings

There was no need to diverge from the intended communication paths. The separation of tasks between onshore and offshore-teams worked very well. Almost no architecture violations were committed by the onshore or offshore teams. The few ones that occurred could be detected within a few days although this was already longer than we had anticipated.

While it is possible to detect architectural violations manually, we have noticed that we could improve architectural validation by using a tool – in our case the Sotograph (software tomography) tool [10]. With the help of this tool, a software architect could validate all of the code within two hours. Although tools such as the Sotograph offer a wide range of features including automated regression validation, they can only deliver raw findings pointing to potential violations and problems. It appears impossible to automatically evaluate these findings and prescribe corrections without human intervention.

The learning curve for the offshore developers was quite steep. Comprehension could be significantly improved by providing good examples such as similar components implemented by experienced onshore developers. We assume that these problems could be reduced even further by extending the initial phase during which members of the future offshore team work together with the onshore team.

5. Conclusion

In this paper we described how onshore and offshore teams can successfully collaborate based on an architecture-centric approach.

Starting by defining an architecture based on a set of comprehensible metaphors, we have laid out an approach that clearly assigns tasks to either the on- or offshore team. The onshore team keeps liaises with the customer, defines the architecture and is responsible for quality assurance covering unit tests as well as checking architectural integrity. The offshore team focuses on development after an initial training at the onshore site during which a common understanding of the project is gained. The (metaphor-based) architecture forms the base to which all further communication recurs to. It is due to this common base that the amount of miscommunications can be greatly reduced.

We are confident that architecture-centric approaches can play a significant role in overcoming offshore communication woes and thus lead to an offshoring quota which is well above the current 1.5%.

6. References

[1] J. Ribeiro, India's offshore outsourcing revenue grew 33%, http://www.computerworld.com/action/article.do?command=printArticleBasic&articleId=9000877, Computerworld, June, 2006

[2] K. Friedmann, *Deutsche IT-Manager planen schlecht*, http://www.computerwoche.de/it_strategien/it_management/571685/, Computerwoche, February 2, 2006

[3] J. C. McCarthy, *Offshore Outsourcing: The Complete Guide*, Forrester Research, Cambridge, MA, 2004

[4] H. Huntley, *Five Reasons Why Off-shore Deals Fail*, Gartner, Stamford, CT, 2005

[5] R. Kalakota, M. Robinson, *Dual-shore project management: Seven techniques for coordinating onshore-offshore projects*, http://www.informit.com/articles/article.asp?p=409917, 2005

[6] J. Sauer, "Agile practices in offshore outsourcing - an analysis of published experiences.", In: *Proceedings of the 29th Information Systems Research Seminar in Scandinavia, IRIS 29,* August 12-15, Helsingoer, Denmark

[7] L. Bass, R. Kazman, *Architecture-Based Development*, Technical Report CMU/SEI-99-TR-007, ESC-TR-99-007, 1999

[8] R. E. Grinter, J. D. Herbsleb, D. E. Perry, "The Geography of Coordination: Dealing with Distance in R&D Work", In *Proceedings of the international ACM SIGGROUP Conference on Supporting Group Work* (Phoenix, November 14 - 17, 1999). GROUP '99. ACM Press, New York, pp. 306–315

[9] H. Züllighoven, *Object-Oriented Construction Handbook: Developing Application-Oriented Software with the Tools & Materials Approach,* dpunkt.verlag. Co-publication with Morgan-Kaufmann, 2004

[10] W. R. Bischofberger, J. Kühl, S. Löffler, "Sotograph – a pragmatic approach to source code architecture conformance checking", In F. Oquendo, B. Warboys, R. Morrison (eds.), *Software Architecture, First European Workshop, EWSA 2004*, St. Andrews, UK, May 21-22, 2004, Proceedings. Lecture Notes in Computer Science, Springer 2004, pp. 1–9