Mastering Dual-Shore Development – The Tools and Materials Approach Adapted to Agile Offshoring

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. Software development in offshoring settings with distributed teams presents particular challenges for all participants. Process models that work well for conventional projects may have to be adapted. In this paper we present case-study-reinforced advice on how to extend the Tools & Materials approach – a well established communication-centered agile design and development approach – to the field of dual-shore development in offshoring projects. We show how communication challenges can be tackled with common guiding and design metaphors, architecture-centric development, task assignments with component tasks and extensive quality assurance measures.

Keywords: Offshoring, Tools & Materials approach, dual-shore, architecture-centric development, agile practices.

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. Offshoring projects are at the top of the complexity scale with diverse issues in areas of organization, management, communication and teamwork. This is especially true for projects that feature geographically separated onshore and offshore project teams [2]. When over 5000 executives across North America and Europe were asked about the success of their offshore strategy in 2004, 36% considered their offshore strategy failed and over one in three had to move work back from their offshore to their onshore team [3].

Given these figures, it is easy to see why looking at measures that help preventing project failure is worth the effort. In this paper we show how process models can be extended and adapted to the complex challenges of offshore projects. We have explored these issues based on the Tools & Materials approach (T&M) [4] which has been used successfully in many single-site agile development projects. The extensions have been validated in a first substantial case study with onshore and offshore teams.

After giving an overview of different offshoring approaches and limiting this paper's scope to the dual-shore approach we go on to demonstrate that communication is a core challenge in offshoring projects. We then present the T&M approach and

B. Meyer and M. Joseph (Eds.): SEAFOOD 2007, LNCS 4716, pp. 83–95, 2007.

show how it has helped us meeting communication challenges in single-site projects. We describe how this approach has been adapted to dual-shore development by augmenting it by four new elements. Then we present findings from the case study. Finally, in the concluding section, we will sum up the essence of the extended approach and give an outlook on future research.

2 Collaboration Models for Offshoring

Offshoring comes in many flavors, but not all of them are pertinent to solutions in the area of software architecture. The only – although highly relevant – two complications that classic offshoring projects introduce are those of cultural differences and split locations. While these problems are not to be underestimated, they can be seen as just an exacerbation of the classic problems between the business-side and the engineering-side of a "normal" single-site software project because the line between on-site and offshore is identical with the line between specification and implementation. These difficulties, however, aren't new and have been thoroughly dealt with in software engineering literature [4].

Experience reports have shown [5] that this classic offshoring setting works best with stable specifications and a minimal need for communication during implementation. To put it in an oversimplified way: The specification is sent to the offshore location and after a while the binaries are shipped back for testing. Many software development projects are too complex to be dealt with in such a fashion. They require frequent interaction between the business-side and the engineering-side due to complex and rapidly changing requirements on the business-side. The dual-shore model for offshoring caters to these needs: As trying to discuss these changes in requirement over huge distances with people from different cultural backgrounds appears to be too difficult, development is carried out on-site as well as offshore. The on-site team is staffed with local developers who deal with the business-side. As both sides are from the same cultural group and located at the same site, classic offshoring problems between business-side and engineering-side can be avoided completely. The divide between shores now runs right through the development team. But this location of the rift is still advantageous to the classic setting because now communication partners on both sides are engineers.

It is this dual-shore setting that we have in mind when dealing with offshoring in this paper. Before describing the specifics of our dual-shore approach in section 5, we will first establish the necessary basis by taking a closer look at the challenges that these projects are faced with (section 3) and by introducing the T&M approach which encompasses many helpful concepts in overcoming them (section 4).

3 Offshoring Benefits and Offshoring Challenges

Clearly, the dominant expectation of corporations that outsource (parts of) their IT is cost saving [6]. 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 [2, 5, 7].

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. We describe these levels here as introduced in [8].

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 cannot change inherent cultural characteristics. They can help only indirectly in establishing a better understanding between people from different

cultural backgrounds. 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.

These challenges have to be dealt with in development processes. We chose the T&M approach that already incorporates measures to improve communication in development projects to give an example of how to evaluate and further enhance established development processes for dual-shore development. We will present its relevant basic concepts in the next section before we continue to describe necessary enhancements in the following section.

4 The Tools and Materials Approach

The Tools & Materials approach (T&M) facilitates application software development by providing guidance in matters of software architecture and the software development process. It is based upon object-oriented design and development and an evolutionary, agile proceeding.

4.1 Enhancing Communication

T&M focuses on two aspects of communication:

- *precise communication* between all stakeholders (customers-developers, developers-developers, customers-customers), and
- *frequent communication* between all stakeholders

Both aspects aim at reducing to a minimum the impact of unavoidable miscommunication – the core problem of software development in general and especially of offshore outsourcing.

Precise Communication
Based on the realization that communication works best on the basis of a common frame of reference, T&M provides several means of providing this very frame. To do this, it does not introduce new concepts, but recurs to culturally established concepts: metaphors, leitmotifs, and patterns:

Metaphors are at the core of the approach. They 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 T&M are Tool, Material, Automaton, Container and Working Environment (see Fig. 1). These metaphors have the benefit that they are so basic that every customer and every developer has a precise of what a tool is like and – equally important – what a tool is not. By recurring to these five metaphors, there is a level playing field on which all stakeholders can move freely without one of them gaining the upper

hand due to an advantage in communication. Developer organizations often unintentionally tend to have these advantages over customers by using UML diagrams that customers do not fully understand. While customers agree to what can be seen in the diagram out of insecurity about its precise semantics, they later complain about the software that has been developed based on this miscommunication.

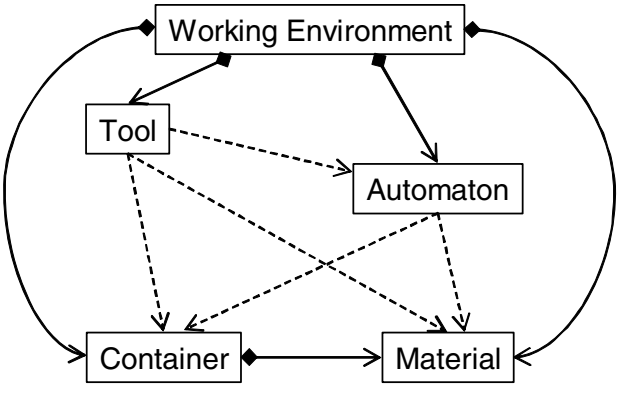

Fig. 1. Main Metaphors of the T&M approach and dependencies between them

As individual metaphors are not necessarily perfect fits, T&M makes use of *guiding metaphors* which establish a common framework into which individual metaphors fit. For generic office applications, the guiding metaphor "Expert Workplace" is a good fit: It is easy to envision Tools, Materials, Automatons (such as a calculator), Containers (such as folders) and a Working Environment (such as a desk with in and out boxes) at an Expert Workplace. Depending on the project in question, the individual set of (guiding) metaphors has to be determined. In many cases, however, only a few metaphors have to be exchanged.

Metaphors of any kind are great for communication between customers and developers (and customers and customers as well) but they are too imprecise when making the transition to executable code [9]. T&M uses two kinds of patterns to smooth that transition:

Conceptual patterns are based on one design metaphor and delineate what a software artifact based on that metaphor behaves like and what it does not behave like. For example, conceptual patterns for materials include "materials never change their state except when handled by a tool or an automaton" and "materials do not hold display code – it is the sole responsibility of tools to display the materials they let the users work on".

Design patterns describe the static and dynamic interaction of individual classes / objects. While some conceptual patterns can be broken down to at least some of the patterns introduced in [10] (Tools are *Observers* of Material), most T&M design patterns, are custom patterns that stem directly from T&M.

(Guiding) Metaphors and conceptual as well as design patterns are excellent means of establishing communications between all stakeholders and have been tested time and again since the 1990's in numerous projects of radically different application

domains such as insurances, public utilities, oncology, logistics and oncology. Nevertheless, these means can only provide the elements that are discussed during business process analysis. As conventional UML diagrams have the inherent problems mentioned in the preceding paragraph on metaphors, T&M makes use of *exemplary business process modeling* (EBPM [11]). In contrast to UML diagrams, EBPM diagrams tell the story of a certain process in pictures complete with actors, materials, tools, automatons and containers as well as a different kinds of communication and a explicit thread (indicated by ordinals) along which the story unfolds. See Fig. 2 for an exemplary diagram.

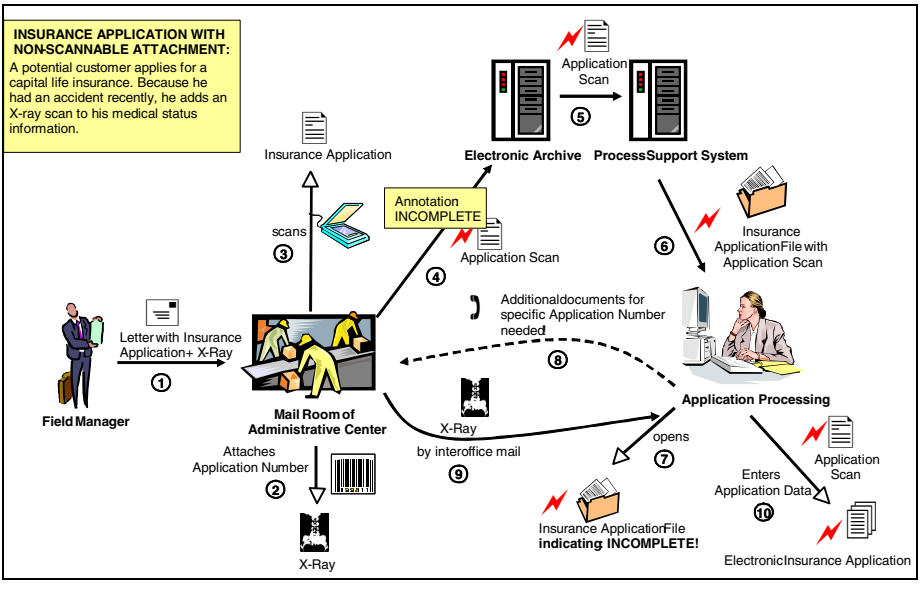

Fig. 2. Example of an EBPM cooperation scenario in the insurance sector

Frequent Communication

As has been shown in the beginning, large communication gaps will eventually lead to costly miscommunications. This is especially true in projects with complex application domains and / or complex team structures. To avoid this source of miscommunications, T&M employs an agile development process with numerous feedback loops ranging from months to seconds in length. For a full list see [9]. Important loops include:

Releases aim at developing new application functionality. The scope is negotiated by customers and developers in planning games about every 6 weeks. This allows for maximum flexibility and avoids the typical problems of formal "complete" specifications which are usually outdated the moment they have been completed (see [9] for benefits of agile development).

Daily *stand-up meetings* during which developers tell each other what they did since the last stand-up and what they intend to do until the next one. These meetings help to evenly spread knowledge about what goes on in every corner of the project.

Programming pair negotiations take place twice a day. By sharing a single computer, developers derive a common understanding about almost every part of the source code. During pair programming, developers are exposed to each others constructive criticism every second so that the software's architecture is constantly a matter of discussion.

Developers can request the presence of an *On-Demand-Customer* any time in case they have questions that cannot be answered by looking at the specification made during the planning game. The customer is obliged to help them within one day. [12]

4.2 Architecture-Based Development

For the implementation part, T&M encourages architecture-based development. According to Bass and Kazman [13], *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 project 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. It also constitutes the basis for verifiable architecture rules. Automatic rule checking improves implementation consistency and reduces the number of errors.

4.3 Summary

Fig. 3 brings together the most pertinent features of the T&M approach: (Guiding) Metaphors form the basis for communication between all stakeholders.

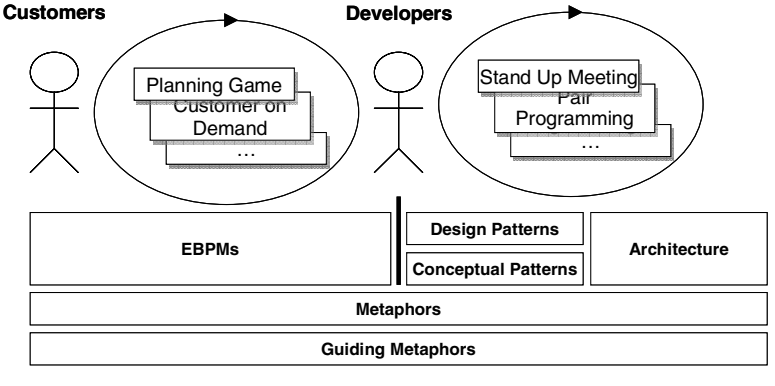

Fig. 3. Overview of the main features of the T&M approach

Customer-Customer and Customer-Developer communication also draws upon EBPMs while Developer-Developer communication uses Conceptual Patterns, Design Patterns and architecture on top of the Metaphors.

All stakeholders communicate on the basis of guiding metaphors and metaphors. On top of that, EBPMs are used between customers and developers. Among developers, architecture descriptions as well as conceptual and design patterns are employed.

5 Extending T and M for Offshore Projects

In section 3 we have discussed the basic problems affecting offshoring projects, leading to the conclusion that communication is of paramount importance. In section 4, we then continued to describe the (single-site) T&M approach which already puts a strong emphasis on communication by introducing metaphors and assigning much importance to architecture. In this section, we draw conclusions from our extensive experience with the application of the T&M approach in single-site projects and challenges and solutions we found in case studies with offshoring projects.

We will first present how single-site T&M should be extended to a dual-shore T&M which can facilitate dealing with offshore communication problems. After introducing our dual-shore model, we will discuss the importance of having a strong focus on architecture and assigning offshore-development tasks component-wise before validating our approach in section 6.

5.1 Dual-Shore Development with Adjusted Agile Practices

The geographical separation of teams in the dual-shore model prevents offshore developers from having an on-site customer at their disposal (*Customer-On-Demand*) and from participating in iteration *Planning Games*. To accommodate for these changed settings, roles are unequally distributed across the teams. The onshore team is made up of software architects and developers. Software architects are responsible for designing and maintaining the application's architecture and carrying out quality assurance. They also serve as business analysts that directly interact with the customer, elucidate the requirements, plan iterations and releases and design the application. The onshore developers train their offshore counterparts at the beginning of projects, perform the main implementation work during the first iterations and tackle difficult implementation work in later iterations. They may also directly interact with the customer to resolve questions.

The offshore team consists entirely of developers. They receive work assignments in the form of component tasks (see 5.3) which they implement in a largely independent fashion, possibly clarifying questions with onshore software architects or in exceptional circumstances with onshore developers.

If possible, the first iterations should be tackled in mixed teams so that the developers get to know each other and develop a common understanding of the domain and the development process. This phase of common development establishes a sound communication basis which can be drawn upon after the offshore team has moved to its offshore location.

5.2 Architecture-Centric Development in Offshoring Projects

While we use architecture-centric development in conventional projects following the T&M approach, it becomes even more valuable in offshoring settings. Communication between the teams benefits greatly from a uniform language and a common technical basis [8].

Architecture also helps in assigning tasks that are decoupled from each other and thus can be developed largely independent by teams at distant sites. So the organization can be split along the product structure [14], reducing the need for inter-site coordination. Additional communication can also be avoided when developers know how they can introduce new features to an application without asking for permission or detailed instructions, e.g. by providing hot spots for enhancements or an explicit plugin-concept.

Architecture rules are defined and regularly checked with automated tools. Onshore software architects design and maintain the application's architecture. The architecture description is regularly communicated to all developers. Changes to the architecture by the developers have to be arranged with the software architects. This way the architecture also evolves from the basis, not only top-down from a software architect's specification. It would be impractical if developers always had to consult a software architect regarding these changes. They should on the other hand be guided in their actions to ensure a reasonable evolution of the architecture.

In the extended T&M approach, the architecture is maintained by an onshore software architect. He verifies that changes and enhancements by the developers are valid and compliant with his architectural vision against the background of the overall application architecture and planned future requirements. He also maintains a master description of the project-specific architecture that is made available to all developers, e.g. through the common version control system. Controversial or comprehensive architectural changes should be discussed with the development team to ensure a common understanding.

This division of labor guarantees that developers can work without bottlenecks and that the evolution of the architecture is guided by an experienced architect. Our experience shows few cases where architectural changes by the developers had to be corrected by the software architect. With the guidance of a common architecture, explicit metaphors and good examples in the existing implementation, developers have a good basis for their design decisions.

With the importance of architecture validation and the complexity of today's applications, a software architect has to rely on software tools for quality assurance. Their help permits an automated comparison of the planned and the really implemented architecture. They also provide metrics and queries for an in-depth review of the implementation [15].

5.3 Component Tasks

The story cards of the widespread agile process model of Extreme Programming, that are also used in most current projects based on the T&M approach, capture only the essence of requirements in the form of informal stories. The details need to be discussed and clarified with the customer and the team. This is difficult in offshoring

settings and increases the demand for communication. Therefore, we use component tasks in the adapted T&M approach.

The use of component-based development is well-suited for agile projects [16]. Components make it possible to divide along well-defined interfaces. The relationship between components has to be explicitly defined by architecture rules so that they can be developed and tested mutually independently to a large extent. The component descriptions can serve as a basis for coordination and discussion between teams.

Components can have different sizes and can be ordered hierarchically. This enables an incremental shift of more and more tasks from the onshore to the offshore team. Small initial components give offshore developers a manageable task to start with. They do not have to understand all of the domain and the business logic from the beginning. These components are assembled into more complex components and integrated into the application by an experienced onshore team. Over time, bigger and bigger components can be constructed and integrated offshore, leading to overall cost reduction.

Component tasks define not only components to be developed but also the required context of the application domain to minimize callbacks, the hot spots or extension points for this component and, if possible, tests that the component has to satisfy.
Fig. 4 shows the adapted T&M approach.

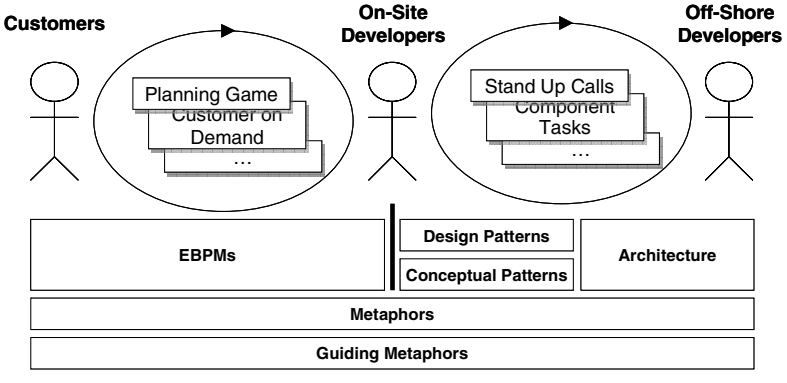

Fig. 4. The adapted T&M approach

As in the basic T&M approach, developers communicate on the basis of guiding metaphors, metaphors, conceptual patterns, design patterns and architecture descriptions. Single-site agile techniques which are incompatible with dual-shore development are replaced with suitable alternatives. The communication between customers and (on-site) developers remains unchanged.

6 Case Study

To validate the extensions for offshoring projects to the T&M approach, a case study was conducted. During four months (March to June 2005) 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.

6.1 Setting and Process

The development was carried out following the dual-shore offshoring model. The elicitation of business requirements and the 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, architectural know how such as integrating components that were built offshore. These components were aligned with the application architecture. They were specified in fair detail with the necessary domain knowledge. Unit tests were developed together with the components. Quality assurance was carried out onshore before integrating the components.

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 descriptions were shared with the offshore team after updates.

6.2 Findings

The results from the case study show that the described extensions to the T&M approach work well in practice. The following issues are worth noting:

Dual-Shore Development
The separation of tasks between onshore and offshore-teams worked very well. There was no need for direct communication between onshore and offshore developers. Coordination occurred solely between the onshore and offshore project leads. The offshore team also did not communicate directly with the customer. Tasks that demanded direct communication, e.g. set-up of the database connection, were handled onshore.

Architecture-Centric Development
Almost no architecture violations were committed by the onshore or offshore teams. The few ones that occurred could be detected and resolved very fast. Extending the architecture was solely the onshore teams' tasks. The learning curve for the offshore developers was quite steep. Comprehension could be significantly improved by providing good examples, e.g. similar components implemented by experienced onshore developers. A longer prior training and pair programming with experienced developers at the start of the project could help.

Our experience also shows that architecture violations are much easier to correct right after they are introduced rather than at later stages. This is especially true for cyclical dependencies. Small cycles are easy to comprehend and dissolve. As cycles

tend to grow rapidly, it usually does not take long before they embrace so many artifacts that it is not obvious where to cut them. The conclusion is to take architecture validation seriously and to correct mistakes right away.

Component Tasks
We found that a stronger orientation on components can improve task sharing between onshore and offshore teams with a more strongly formalized approach on the basis of a common architecture.

The concept of component tasks worked well. The structure of the task descriptions was refined throughout the project. Most of the time the tasks were defined clear enough and only minor misunderstandings occurred. At first, only small tasks were handled offshore. In later iterations of the project, bigger tasks, e.g. larger components, could be developed offshore. At the peak about a quarter of the overall work was done offshore.

7 Conclusion – An Extended T and M Approach

In this paper we examined benefits and challenges of offshoring and described how process models can be adapted to offshoring projects by the example of the Tools & Material approach.

While the basic concepts, such as guiding and design metaphors, conceptual and design patterns, architecture-centric development based on an explicit model architecture and agile, iterative development remain unchanged, the process model was adapted to incorporate onshore and offshore teams with fixed assignments and responsibilities. Architecture-centric development plays an even more important role in the extended T&M approach and helps in assigning tasks to teams, directing and formalizing communication between them and thus reducing the need for direct communication. We also presented results from a case study that we conducted to evaluate the adapted approach and where we could validate the extensions for offshoring. In the future, we plan to evaluate the approach in other projects and advance it further.

We hope that our results on how a single-site approach can be extended to offshoring settings will be transferable to other development approaches and that this helps to decrease the rate of failed offshore projects in the medium term.

References

1. Ribeiro, J.: India's offshore outsourcing revenue grew 33%, Computerworld, 06/06 (2006), http://www.computerworld.com/action/artcle.do?command=printArticleBasic&articleId=9000877
2. Kalakota, R., Robinson, M.: Dual-shore project management: Seven techniques for coordinating onshore-offshore projects (2005), http://www.informit.com/articles/article.asp?p=409917
3. Hatch, P.J.: Offshore 2005 Research: Preliminary Findings and Conclusions, Vers.1.2.5 Ventoro (2005), http://www.ventoro.com/Offshore2005ResearchFindings.pdf

4. Züllighoven, H.: Object-Oriented Construction Handbook: Developing Application-Oriented Software with the Tools & Materials Approach, dpunkt.verlag. Co-publication with Morgan-Kaufmann (2004)
5. Sauer, J.: Agile practices in offshore outsourcing – an analysis of published experiences. In: Proceedings of the 29th Information Systems Research Seminar in Scandinavia, IRIS 29 - Paradigms, Politics, Paradoxes, August 12-15, pp. 12–15. Helsingoer, Denmark (2006)
6. McCarthy, J.C.: Offshore Outsourcing: The Complete Guide. Forrester Research, Cambridge, MA (2004)
7. Huntley, H.: Five Reasons Why Offshore Deals Fail, Gartner, Stamford, CT (2005)
8. Kornstädt, A., Sauer, J.: Tackling Offshore Communication Challenges with Agile Architecture-Centric Development. In: Proc. of the Sixth Working IEEE/IFIP Conference on Software Architecture (WICSA 2007), Mumbai, India, January 6-9, pp. 6–9 (to appear, 2007)
9. Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA (1999)
10. Gamma, E., et al.: Design-Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley, London, UK (1995)
11. Breitling, H., Kornstädt, A., Sauer, J.: Design Rationale in Exemplary Business Process Modeling. In: Dutoit, A.H., McCall, R., Mistrik, I., Paech, B. (eds.) Rationale Management in Software Engineering, pp. 191–208. Springer, Heidelberg (2006)
12. Lippert, M., Becker-Pechau, P., Breitling, H., Koch, J., Kornstädt, A., Roock, S., Schmolitzky, A., Wolf, H., Züllighoven, H.: Developing Complex Projects Using XP with Extensions. IEEE Computer Magazine 36, 06/03 (2003)
13. Bass, L., Kazman, R.: Architecture-Based Development, Technical Report CMU/SEI-99-TR-007, ESC-TR-99-007 (1999)
14. Grinter, R.E., Herbsleb, J.D., Perry, D.E.: The Geography of Coordination: Dealing with Distance in R&D Work. In: Proceedings of the international ACM SIGGROUP Conference on Supporting Group Work, November 14-17. GROUP '99, pp. 306–315. ACM Press, New York (1999)
15. Bischofberger, W.R., Kühl, J., Löffler, S.: Sotograph – a pragmatic approach to source code architecture conformance checking. In: Oquendo, F., Warboys, B.C., Morrison, R. (eds.) EWSA 2004. LNCS, vol. 3047, pp. 1–9. Springer, Heidelberg (2004)
16. Stojanovic, Z., Dahanayake, A.N.W., Sol, H.G.: Component-oriented agile software development. In: Marchesi, M., Succi, G. (eds.) XP 2003. LNCS, vol. 2675, pp. 315–318. Springer, Heidelberg (2003)