ECONSTOR

WWW.ECONSTOR.EU

Der Open-Access-Publikationsserver der ZBW – Leibniz-Informationszentrum Wirtschaft
The Open Access Publication Server of the ZBW – Leibniz Information Centre for Economics

Kurbel, Karl

Working Paper

Process Models and Distribution of Work in Offshoring Application Software Development

Diskussionspapiere / Europa-Universität Viadrina Frankfurt (Oder), Wirtschaftswissenschaftliche Fakultät, No. 257

Provided in Cooperation with:
European University Viadrina Frankfurt (Oder), Department of Business Administration and Economics

Suggested Citation: Kurbel, Karl (2007) : Process Models and Distribution of Work in Offshoring Application Software Development, Diskussionspapiere / Europa-Universität Viadrina Frankfurt (Oder), Wirtschaftswissenschaftliche Fakultät, No. 257

This Version is available at:
http://hdl.handle.net/10419/23825

Nutzungsbedingungen:
Die ZBW räumt Ihnen als Nutzerin/Nutzer das unentgeltliche, räumlich unbeschränkte und zeitlich auf die Dauer des Schutzrechts beschränkte einfache Recht ein, das ausgewählte Werk im Rahmen der unter
→ http://www.econstor.eu/dspace/Nutzungsbedingungen
nachzulesenden vollständigen Nutzungsbedingungen zu vervielfältigen, mit denen die Nutzerin/der Nutzer sich durch die erste Nutzung einverstanden erklärt.

Terms of use:
The ZBW grants you, the user, the non-exclusive right to use the selected work free of charge, territorially unrestricted and within the time limit of the term of the property rights according to the terms specified at
→ http://www.econstor.eu/dspace/Nutzungsbedingungen
By the first use of the selected work the user agrees and declares to comply with these terms of use.

ZBW Leibniz-Informationszentrum Wirtschaft
Leibniz Information Centre for Economics

Mitglied der Leibniz-Gemeinschaft

Process Models and Distribution of Work in Offshoring Application Software Development

Prof. Dr. Karl Kurbel

European University Viadrina Frankfurt (Oder)
Department of Business Administration and Economics
Discussion Paper No. 257
2007

ISSN 1860 0921

Process Models and Distribution of Work in Offshoring Application Software Development*)

Abstract

Common process models for the development of application software (AS) are examined as to how well they are suited for offshoring projects. The need for communication and interaction among onsite and offshore project stakeholders is identified as a critical success factor. Process models used by organizations providing offshoring services are discussed, and a generalized offshoring life cycle model is developed. A specific focus is set on the distribution of work between the organization that outsources AS development and the offshore organization that carries out the major share of the development work. Problems and challenges that have to be faced, making offshoring a difficult task, are discussed.

1 Introduction

The context of this paper is development of application software (AS), or more specifically business information systems (IS), by or for enterprises and other organizations in Western European countries and the US. We take an approved project proposal as the starting point for a development project; i.e. the management decisions to launch the project and to build the system have been made. Various constraints may limit the degrees of freedom for the project. For example, the existing information systems landscape has to be considered. Most likely the new system will need to be interfaced with the company's ERP (enterprise resource planning) system and with other information systems. The most binding constraint is often a tight budget and scarce staffing, limiting the possibilities of what can be done.

1.1 Process Models

Any development project needs a *specification* of the problems to be tackled. Assuming that an operational problem specification has been created, development of the information system may start. The development effort can be conducted in many ways. Templates arranging development activities into a specified order are called *software process models.* (Note that the term "process" refers here to software development activities and not to business processes.) They can be defined as follows:

> A *software process model* is an ordered set of activities with associated results that are conducted in the production and evolution of software. It is an abstract representation of a type of software process.

In a formal view, a software process model can be regarded as a description of a software process at the type level. A particular process is an instantiation of the process model. However, a process model is usually normative ("how things should be done") whereas process instances are what happens in reality.

---

*) This paper was submitted to GDW 2007 – Second International Conference on Management of Globally Distributed Work, Bangalore, India, July 25-27, 2007.

A large number of software process models have been proposed since the beginning of software engineering, categorized in many ways, and described by attributes like

linear vs. iterative development,
sequential vs. incremental development,
plan-driven vs. agile development,
model-driven vs. evolutionary development.

Decades of discussion about the best approach to information systems development have gone by, and method wars have been fought over what might be the best methodology. Most approaches have survived the wars, so the variety still exists today, and new models are coming into existence as new organizational forms in the IT industry are emerging.

1.2 Offshoring Development Projects

Organizations that develop software – user organizations and software companies as well – have several choices. They can choose between inhouse and external development. If external development is the preferred option, should the external partner be a domestic or a foreign one? In the case of a multinational company, another choice is between developing at home vs. developing at a location abroad where that company has a branch and where software development is more cost-effective. It is the management's task to define a strategy and to decide which of the various options to take.

Since software development costs are significantly lower in Asian, Latin American, and Eastern European countries than in the United States and in Western Europe, many software orders have gone to vendors in such countries. While the price has been the driving factor for many years, other reasons like quality and process maturity have emerged in the recent past. For example, the know-how and maturity levels of professional software companies in India are on average higher than in the US and Western Europe.

With respect to software development, three related terms are onsite, onshore, and offshore development. While *onsite* means development at the organization's location, *onshore* stands for development at a different place in the same country (e.g. by a domestic contractor), and *offshore* stands for the same but in a different country. Offshoring in AS development describes the matter that: a) either a third party – usually a software firm in a lower-wage country – is contracted by the customer for the development of one or more information systems, or for certain steps of the development process; or b) a local branch of a multinational organization is employed for software development (captive center); or c) an outsourcing provider and the customer enter into a joint venture regarding IT services. The two parties set up a new firm that will carry out development projects.

2 Which Process Model is Suited for Offshoring?

IS development projects that lend themselves easily to offshoring are projects that automate well-documented business functions or processes where little day-to-day interaction is required. The ideal process would be one that is completely specified in terms of process steps, inputs, and outputs of those steps. In such a case, a specification could be "thrown over the wall" (i.e. handed over to the outsourcing firm) and an information system will come back as the result of the project.

Unfortunately most projects are not of that nature, requiring a lot of interaction. Therefore project teams have to be set up in a way that onsite and offshore personnel communicate intensely. Another important source of communications needs is transmitting customer or end-user requirements. Before proceeding to offshore-specific issues, we take a brief look at the spectrum of process models from this perspective.

2.1 Sequential Process Model (Waterfall Model)

The oldest and best-understood process model is the sequential process model. It is based on the idea that the development process can be divided into distinct stages with specified inputs and outputs and well-defined results. The next stage starts when the previous one is completed. Results cascade from one stage downwards to the next stage, just like a waterfall. The flow of work is sequential and basically unidirectional as illustrated by the arrows on the right-hand side of figure 1.

The waterfall model was the first process model in software engineering. Its goes back to a systems engineering model that was adopted to software development by Winston Royce [Royce 1970]. Since it was "the" process model for a long time, it has also been called *software life cycle model* (SLC model). Until today the term software life cycle model is used to refer to the waterfall model, although many different types of software life cycles have come into existence in the meantime.

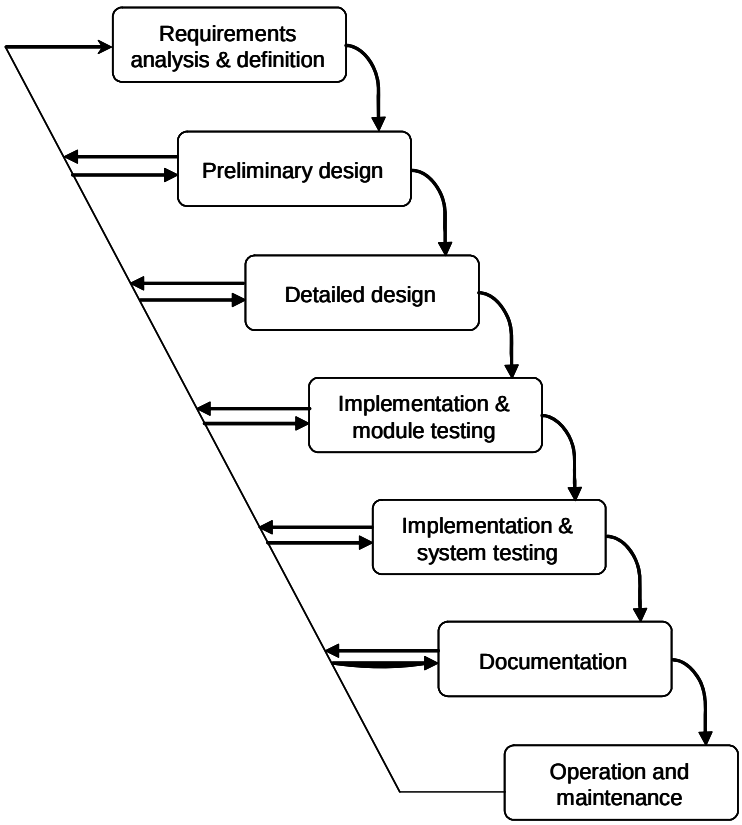

Fig. 1: Waterfall model

Drawbacks and advantages of the waterfall model have been extensively discussed for many years. Most authors agree that the assumption of distinct phases performed in strict sequential order does

not conform to what happens in practical projects. Mostly it is unrealistic to expect that one phase can be definitely completed with a correct result before the next phase starts.

An empirical observation, for example, are changing requirements. When a system is built for a customer it happens quite often that requirements are modified and/or new requirements are formulated by the customer later in the project. In such a case, the design has to be reworked and perhaps the implementation changed. There are many examples of situations where work in the next stage has an impact on results of one or more previous stages. To cope with these real-world circumstances, modifications of the waterfall model that include revisiting earlier stages were proposed. The underlying ideas are illustrated by the arrows on the left-hand side of figure 1. One immediate variant is that information from the next stage flows back to the previous stage, causing earlier results to be revised. Larger iterations are induced when the need to return to a stage upfront arises.

The fundamental disadvantage of the waterfall model and its extensions is the sequential flow of information and results from one stage to the next. Even in its iterative variants, the main process is a sequential one. Iterations are basically done to correct flaws and to improve specification and design features that were done badly before, be it because of lack of knowledge at that time or just because of mistakes.

2.2 Evolutionary Process Models

To overcome the drawbacks of the waterfall model, two guiding principles that are fundamentally different from the sequential approach were established:

1. Making software development an evolutionary process
2. Building prototypes

While most systems undergo an evolution after they have been installed, i.e. they grow and change, *evolutionary development* means that system growth and changes are already embedded as integral parts in the development process. Iterations in the software lifecycle model serve this purpose to some extent, but they are rather considered a necessary evil than a welcome process feature. In a truly evolutionary process model, an information system comes into existence through evolution: Starting with incomplete and perhaps insufficient knowledge about what the final information system has to be like, a limited subsystem is created in the beginning. Continuing from this subsystem, an enhanced, extended, and/or better subsystem is created. This subsystem may include new or better components than the previous subsystem. The process continues until a satisfactory and complete information system has evolved.

Process models based on explicit *development cycles* supporting evolutionary and incremental processes were developed in the 1980's and 1990's both in practice and in academia [e.g. Kurbel, Pietsch 1990]. Many organizations started to adapt their processes to the new ideas. The ultimate outcome of the R&D efforts is the Rational unified process discussed next.

2.2.1 Rational Unified Process (RUP)

The *Rational unified process (RUP)* process is a model that was created in a joint effort by three well-known experts in object-oriented analysis and design, Ivar Jacobson, Grady Booch and James Rumbaugh. The major outcome of this effort is RUP and UML (unified modeling language).

RUP is a process model, a framework to create process models, and a methodology to develop software systems. As a *process model*, it supports incremental development, dividing large projects into smaller sub-projects. Characteristics are iterations and increments, strong involvement of all stakeholders (developers, architects, end users, managers, customers, etc.) at all stages, and built-in quality assurance.

The process model is two-dimensional. The dimensions are phases and disciplines (originally called workflows). Phases extend in time, and disciplines expand into activities. The *phases* are called:

- inception,
- elaboration,
- construction, and
- transition

The *disciplines* are: business modeling, requirements, analysis & design, implementation, test, deployment, configuration & change management, project management, and environment.

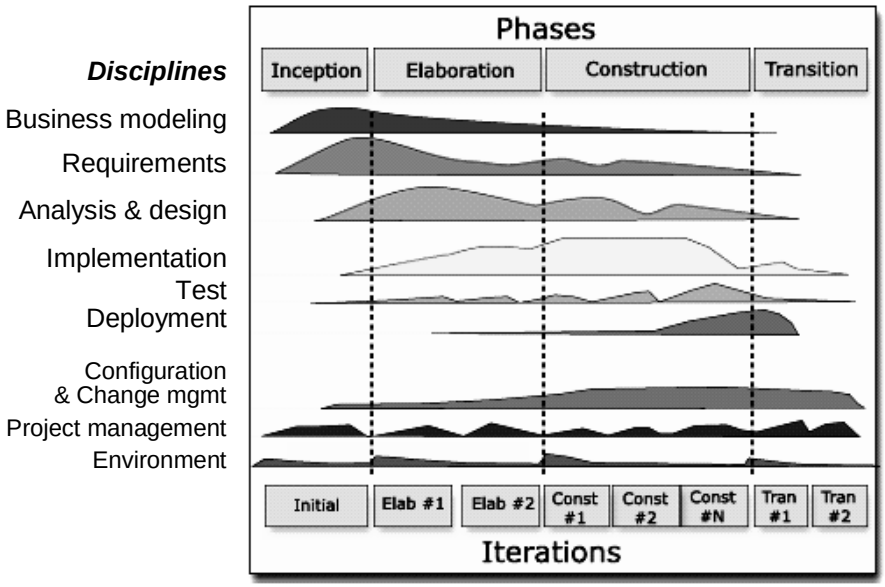

Fig. 2: RUP lifecycle – phases and disciplines [Ambler 2005, p. 2]

The hump-chart diagram of figure 2 is one of the landmarks of RUP. It expresses several things quite clearly:

1. Disciplines extend across phases. This means that typical activities like modeling, analysis, design, implementation and test are not confined to one phase but are ongoing activities during the entire lifecycle.

2. The humps in the curves indicate how much effort will be needed at what times. For example, most of the analysis & design effort occurs in the elaboration phase, whereas the implementation curve has its highest point in construction.

3. Iterations occur within phases. Examples of iterations are given at the bottom of figure 2: Inception has only one iteration in this example; the elaboration, construction, and transition phases consist of two, three, and two iterations, respectively.

Iterations are intended for all phases, yet only within a phase. The result of an iteration, especially in the construction stage, is an *increment* or a subsystem which could conceivably be deployed to users as a release. The phases as such are sequential. For example, the construction phase starts when elaboration is over. However, activities within a phase, e.g. analysis, design, implementation, and test, may be performed repeatedly until a satisfactory outcome is obtained. In this way, RUP combines sequential and iterative process aspects in one process model ("serial in the large, iterative in the small" [Ambler 2005, p. 1]).

Disciplines as the steps of the iterations are not performed in a strictly sequential manner. In fact, activities often overlap. While the "natural" sequence: requirements → analysis → design → code → test, etc. still exists, one activity does not need to be finalized before the next one can start. A more common approach is to take a subset of the requirements, do some analysis, go back to re-work some of the requirements, proceed to analysis & design, re-work requirements again, start coding, go back to design, etc.

The RUP life cycle incorporates many characteristics of *evolutionary development* and *prototyping* as mentioned earlier. For example, work products – models, plans, source code, documents – evolve throughout the life cycle. Work products are not finished until the system is released into production [Ambler 2005, p. 5] . The project is planned in a rolling wave. This means that planning is detailed for things closer to today and less detailed for things further away. As the project progresses and tasks get closer, detailed planning for these tasks is done.

2.2.2 Agile Development and Extreme Programming

Even more incremental and iterative than RUP are the *agile development (AD)* and *extreme programming (XP)* approaches. AD was born in "... a rebellion in the industry against a growing tide of poor performances, long lead times, poor quality, disappointed customers, and frustrated developers." [Anderson 2004, p. xxviii] After three decades of software engineering, the state of the art is still that many development projects exceed budgets and delivery dates – provided that they don't fail completely – and maintenance costs are high due to low software quality. "Senior executives, perplexed by the spiraling costs of software development and depressed by poor results, poor quality, poor service, and lack of transparency are simply shrugging their shoulders and saying, 'if the only way this can be done is badly, then let me do it badly at a fraction of the cost'. The result is a switch to offshore development and layoffs." [Anderson 2004, p. xxv]

The most famous AD document is the *agile manifesto* ("manifesto for agile software development"). This is a position statement indicating what the agile alliance considers fundamental ideas for better software development.

While the agile manifesto and the agile principles express a certain way of thinking about and attacking software development, they provide neither an operational approach nor a process model. Several methodologies have been developed to fill this gap. Among them are:

- Extreme programming (XP),
- Scrum (http://www.controlchaos.com/about/)
- Feature driven development (FDD – http://www.featuredrivendevelopment.com/),
- Crystal Clear [Cockburn 2004],
- Adaptive software development (ADP – http://www.adaptivesd.com/)

Extreme programming (XP) is the best-known of the agile methodologies. It became popular through Kent Beck's book in 1999 (first edition of [Beck, Andres 2004]). XP stresses customer satisfaction. The main goal of XP is to reduce the cost of change. In traditional system development methods, requirements are determined in the beginning and often fixed from that point on. As pointed out earlier, the cost of changing the requirements at a later stage can be very high. XP sets out to lower the cost of change by introducing basic values and principles, making the process more flexible with respect to changes. Based on these values and principles, extreme programming is a set of simple and concrete practices that combines into an agile development process [Martin 2003, p. 17].

The values and principles XP is based on are, among others:

- *Communication* – frequent and extensive, both within the project and with the customer ("the customer is a team member and always available").
- *Simplicity* – always starting with the simplest possible solution and turning it into a better one later ("short cycles, small releases").
- *Feedback* – frequent and rapid feedback from the system (by unit testing), from the customer (by frequent acceptance tests), and from the team (by involving the team into requirements changes immediately).

A legitimate question to ask is: Does extreme programming have a process model at all? In the agile community, "process" is often used with a negative undertone, implying that a process has attributes like "disciplined" or "structured" which are disliked by agile developers. These attributes are associated with the waterfall model and with certain iterative processes that contain planning in detail. XP instead is described in terms of values and practices.

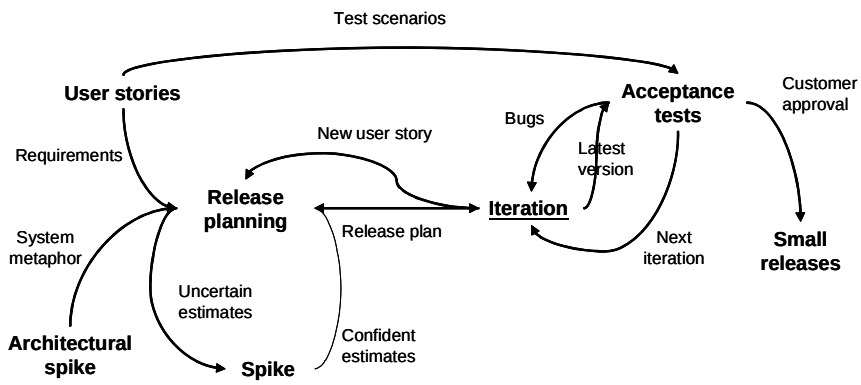

Fig. 3: XP process model [Wells 2006]

Nevertheless, the description of relationships between iterations, releases, stories, and tests indicates that there is a certain process paradigm underlying extreme programming. Figure 3 illustrating the flow of work can be interpreted as a generic model of XP development processes. A concrete project is characterized by a particular path through the graph. Spikes are simple prototypes – solutions created to figure out answers to tough technical or design problems. Most spikes are not good enough to keep, so they are usually thrown away.

2.3 Evaluation of Process Models from an Offshoring Perspective

Process models on the spectrum from one extreme – strictly sequential – to the other one – completely incremental and discussion-based like – exhibit different requirements as to communication and interaction in the project. Figure 4 illustrates this matter by plotting evolution intensity against the need for interaction among project stakeholders.

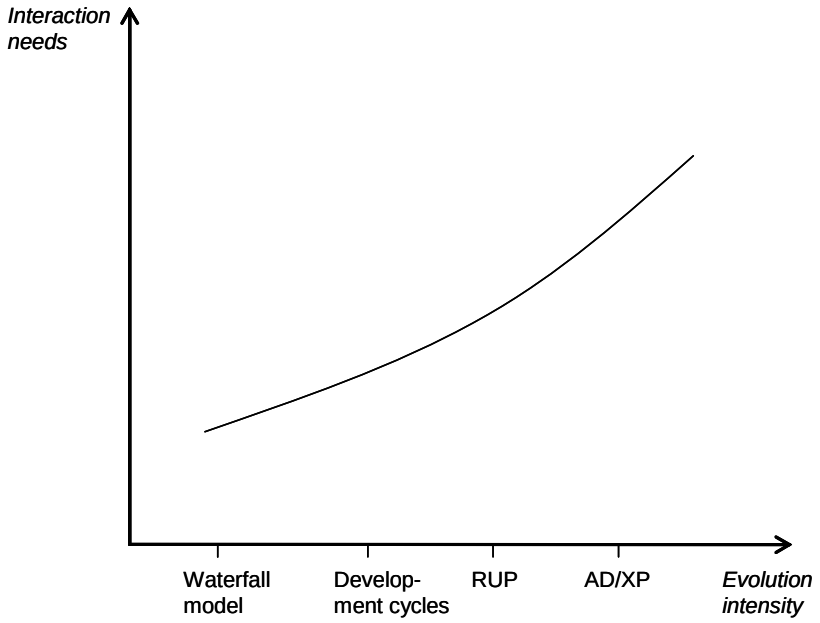

Fig. 4: Spectrum of process models

In a strictly sequential process model like the waterfall model, communication between different stakeholders is limited: System analysts talk to the customer and/or end-users in the requirements analysis & definition stage. At the end of this stage, they hand over the requirements specification to the designers. Designers or architects develop software models and give those models to the software developers at the end of the design stage, etc.

In contrast, at the other end of the spectrum, communication and interaction among the stakeholders are going on permanently. In XP, for example, programmers, architects, quality staff, customers, etc. meet almost continually face-to-face to discuss problem, process, and solution issues.

Considering the situation in offshoring projects where the stakeholders are geographically dispersed over the world, it is obviously not possible to have ongoing face-to-face discussions every other day. That is one reason why the more to the right-hand side of figure 4 process models are located, the less they are used in the offshoring practice. Agile approaches, for example, rely on informal processes to facilitate coordination whereas distributed software development typically relies on formal mechanisms [Ramesh 2006, p. 41].

Process models employed by offshoring providers today tend towards the left-hand side of the spectrum. An obvious advantage of a sequential model is that it has well-defined milestones and documents which can be used for interfacing onsite and offshore tasks. Communication between the onsite and offshore teams is largely based on documents in this process model.

On the other hand, modern approaches like AD and XP have shown beneficial in many projects. Therefore organizations are trying to transfer their benefits to offshoring projects as well. A discus-

sion of how agile methods can be used in globally distributed software development can be found in the October 2006 issue of the Communications of the ACM (in particular [Lee et al. 2006, Ramesh et al. 2006]). Although the findings, based on real-world projects, seem to be encouraging, we must state that in the offshoring practice sequential-type process models are dominating.

3 Offshoring-specific Process Models and Work Distribution

Organizations outsourcing information systems development to an offshoring provider are typically either user organizations with their own development groups, attempting to reduce cost, or software organizations developing custom information systems, with the user organization involved in the development. Rather atypical outsourcers would be user organizations that neither have their own development group nor a commissioned software firm for the development. Although conceivable, organizations without software development know-how are unlikely to turn to remote offshoring providers directly but will rather look for help from a domestic firm (who in turn may outsource some of the development). Underlying the process models outlined in section 2 is the implicit assumption that either the user organization itself or a software company commissioned for the job is developing the system.

With regard to communication, the case of a domestic software company building a custom information system for the customer is not much different from an internal software-development group doing the job, because the contractor is likely to work in a similar way as the internal group, being more or less in daily contact with the customer.

In offshoring projects, the situation is different. The organization carrying most of the development work is at a remote location, far away (offshoring) or at least not close (nearshoring) so that daily face-to-face communication is not a typical characteristic. This means that communication and interaction need to be planned and ensured in a different way. Transition of work results from one organization to the other one has to be prepared based on well-defined documents, software, and quality assurance.

At first sight, process models followed by organizations that offshore AS development are not much different from models for onsite development. Most process models in practice are based on one of the previously discussed models, yet with specific extensions to capture offshoring needs and reality. In particular, organizations that developed software themselves before they started offshoring are often continuing to use the same process model as before. The obvious reason is that a sound foundation of project-management experience and know-how with that model is available.

Offshoring providers, on the other hand, are somewhat limited in the choice of process model for their part, because their model has to match the offshorer's model. The looser the connections between the customer and the contractor are, the more freedom has the contractor to proceed according to his own needs and preferences. Obviously the offshore provider cannot follow an iterative approach like RUP if the customer's process model is strictly sequential.

Since many organizations apply a sequential process model or a variant of such a model in their development projects, it is not surprising that sequential approaches are dominating the literature. Notwithstanding its many disadvantages, the waterfall model provides clear milestones, deliverables, and points of management control that are particularly helpful when communication and feedback are by nature not as close as in an onsite project.

High-level offshore process models often exhibit the same sequential phases as a conventional process model for onsite development, i.e. business problem definition, requirements, analysis & design, implementation, testing, deployment, operations & support. Differences lie in the responsibilities for the activities of these phases (who does what?), including additional activities not present in conventional process models. Responsibilities for activities depend on the chosen scope of outsourcing – i.e. does the organization wish to outsource:

1. coding and testing only,
2. module design, coding, and testing,
3. system design, module design, coding, and testing,
4. "the problem"?

In the first and second cases, most life cycle tasks remain with the customer's project team. In the third case, the cut-off stage is the system design. Often the onsite and offshore teams collaborate for the design as illustrated in figure 5. The figure reflects the high-level process model of a large, globally acting German software company. The underlying division of labor leaves project management, architecture, high-level design, risk and quality management, and the final testing responsibilities with the outsourcer while the detailed design, coding, and much of the testing are in charge of the offshoring provider.

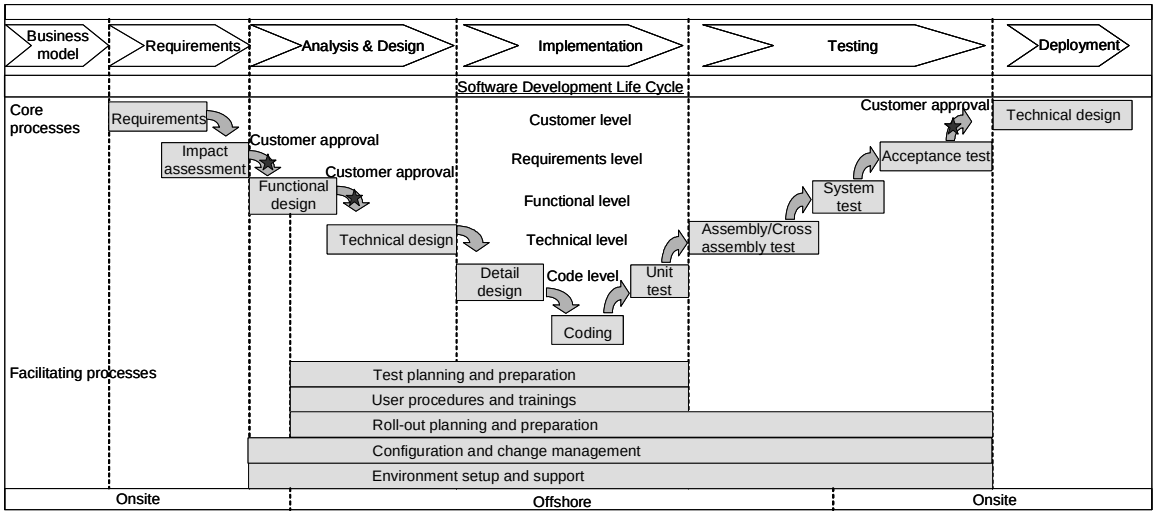

Fig. 5: Onsite and offshore responsibilities in a high-level process model [T-Systems 2005]

The fourth case, outsourcing the entire business problem, is again different in that most of the responsibilities are with the offshoring provider, often implying a different business model of the provider than the one in cases 1. to 3.

The assignment of work packages to onsite and offshore locations is reflected in Infosys's GDM (global delivery model). Containing all major stages of the software life cycle, this model can be regarded as a process model as well. As figure 6 shows, four major phases are identified: discovery, design & build, deploy, and post-implementation support. While the "discovery" (of requirements and scope) and deployment of the system happen onsite, the early design and building stages – high-level design and user interface design – involve activities on both the client's and Infosys's sides. The rest of the development stages and activities as well as bug fixing and maintenance are done in India.

Examining offshore process models in more detail exhibits a number of additional tasks different from conventional process models. They are largely due to additional communication, collaboration, and control requirements between onsite and offshore personnel which are not present in onsite projects.

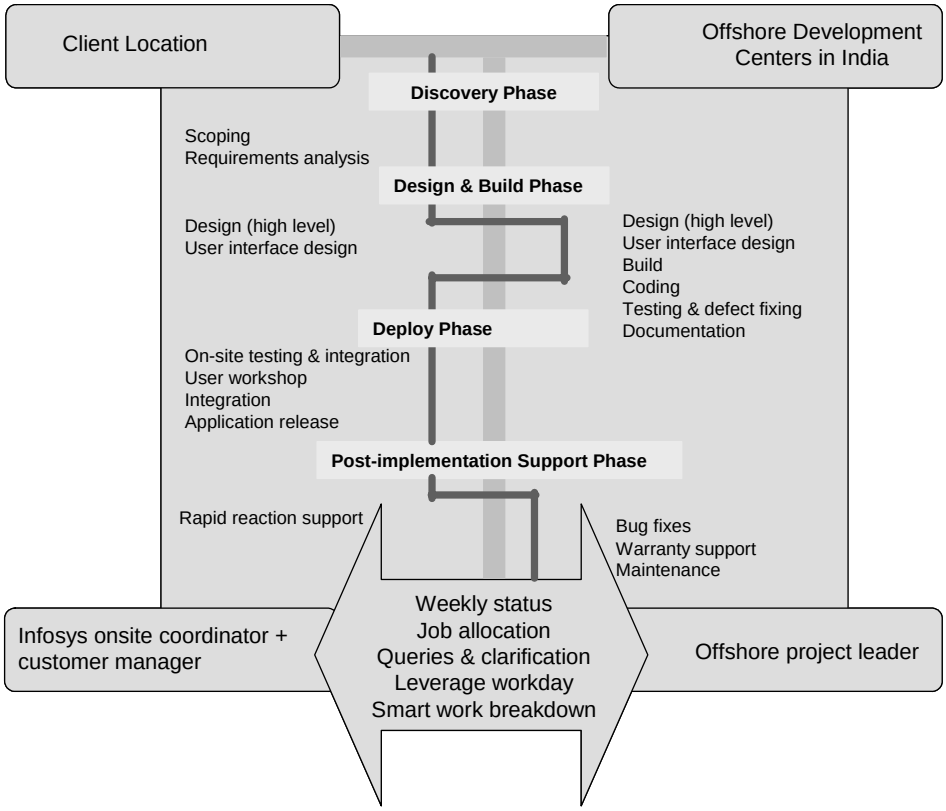

Fig. 6: Infosys GDM for application software development [Chaudhuri 2004]

One such task is examining if the project is suited to offshoring at all. Some criteria are listed in figure 7. For example, the outsourcer's management should commit to offshore the development. If the outsourcer is a software company developing the information system for a customer, the customer might object to offshoring. Small projects are usually not suited to offshoring because of the additional overhead as compared to onshore development. If the system cannot be divided into separate work units, then it is difficult to outsource portions of the project and reliably control deliverables. Usage of standard technologies makes offshoring easier than usage of proprietary technologies. An information system with tight connections and many interfaces with other systems in the customer's IS environment raises more difficulties than a system that is only loosely coupled. If the offshore provider needs to know and understand the customer's business process in detail, it might be too hard to transfer all that knowledge from the customer to the provider.

Criteria for Offshoring	
Is the management committed to offshore the project?	opposed ⟷ strongly in favor
If the system under consideration is developed for a customer, does the customer agree to offshoring?	yes – doesn't care – no
What is the project size (in €, $, or person month)?	very small ⟷ very large
Can the system easily be divided into separate modules, tasks and/or activities?	very easy ⟷ very difficult
What is the share of standard technologies required in the project?	very low ⟷ very high
Is the system tightly connected with other systems at the customer's site (i.e. how many interfaces with other systems)?	very loose ⟷ very tight
How much knowledge of the underlying business process must be offshore provider have?	very little ⟷ very much
How well is the system documented (e.g. completeness, correctness, and understandability of specifications)?	very well ⟷ insufficient
If there is a language barrier between the offshore and the onsite teams, can appropriate communication be ensured?	very easy ⟷ very difficult

Fig. 7: Criteria to examine for offshore projects

An example of additional communication requirements is a sequence of approval steps as illustrated in figure 8. The process summarized in this figure is practiced by DC&M, a Brazilian offshoring provider [DCM 2006]. Here it is assumed that the offshore company enters the process at a very early stage, when the business problem is investigated and business requirements are being specified. The specification created by the offshoring provider's consultant specifies the functional requirements from the business perspective. This specification is created in several steps and iterations, and finally approved by the customer's stakeholders.

Once formally approved, the technical specification is created. Technical staff offshore convert the functional specification into a technical solution (system design). This specification is also created in several steps and iterations with onsite and offshore technical staff involved. Finally it has to be approved by the key players from the technical perspective (e.g. technical project leaders offshore and onsite, developers).

Developing the technical solution comprises implementation and testing. An approval step has to be passed again, this time internally by the offshore provider's quality assurance staff. Afterwards the approved solution is delivered to the offshore provider's business consultant who wrote the functional specification. If the solution passes this review step also, the system is handed over to the customer for reviewing the final result with respect to the business problem and requirements that initiated the project.

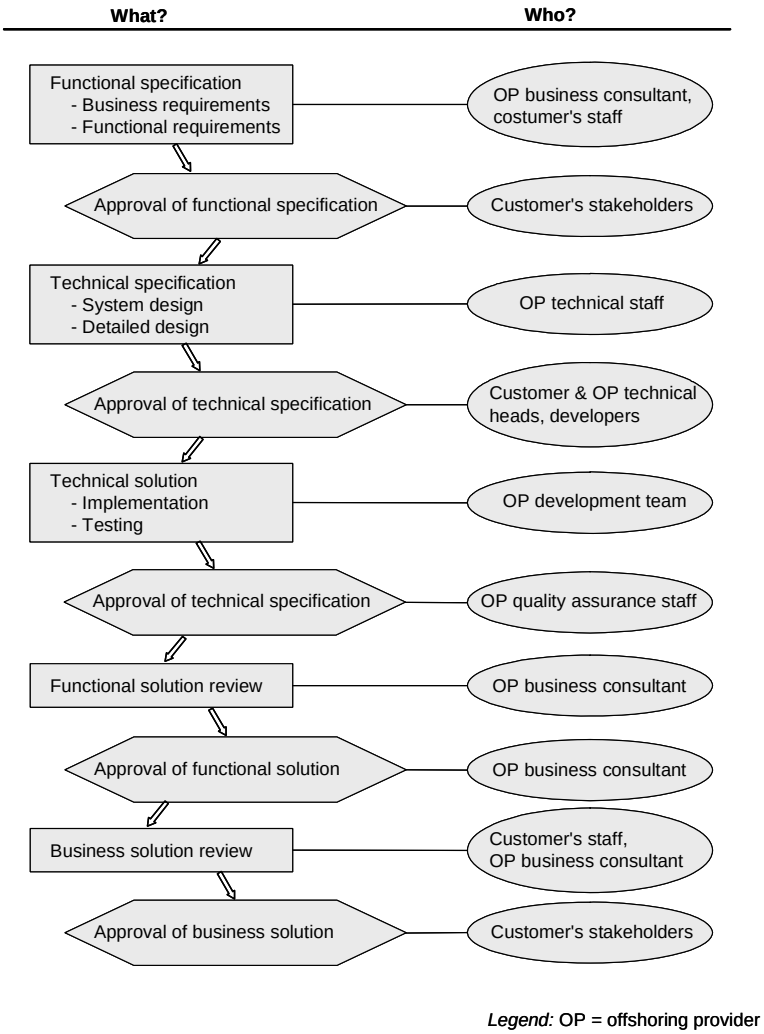

Fig. 8: Offshoring project approval steps [DCM 2006]

4 A Generalized Offshoring Process Model

Since offshoring projects differ in what tasks are outsourced to the offshore company, it is hardly possible to formulate a universal process model for all offshore ISD projects. Figure 9 attempts to capture the major stages that most offshore development projects go through. Depending on the development stage where the outsourcing starts, offshoring-specific tasks and stages are entered earlier or later in the project – following the business-problem specification, requirements analysis & definition, high-level design, or detailed-design stage, respectively. Offshoring-specific stages are the following:

- *Examination of offshoring feasibility:* The first offshoring-specific task, provided that offshoring is considered a serious option, is to investigate in more detail if the project is suited to offshoring. Criteria like the ones listed in figure 7 will be applied.

- *Making the project ready for offshoring* includes the following tasks: Determining what has to be done before a work order can be placed with the provider; setting up a project organization that takes offshoring-specific requirements into account; establishing a rough overall project schedule and a more detailed transition plan; management commitment to outsource project stages offshore; and preparing onsite project members and other stakeholders to deal with the offshoring

situation.

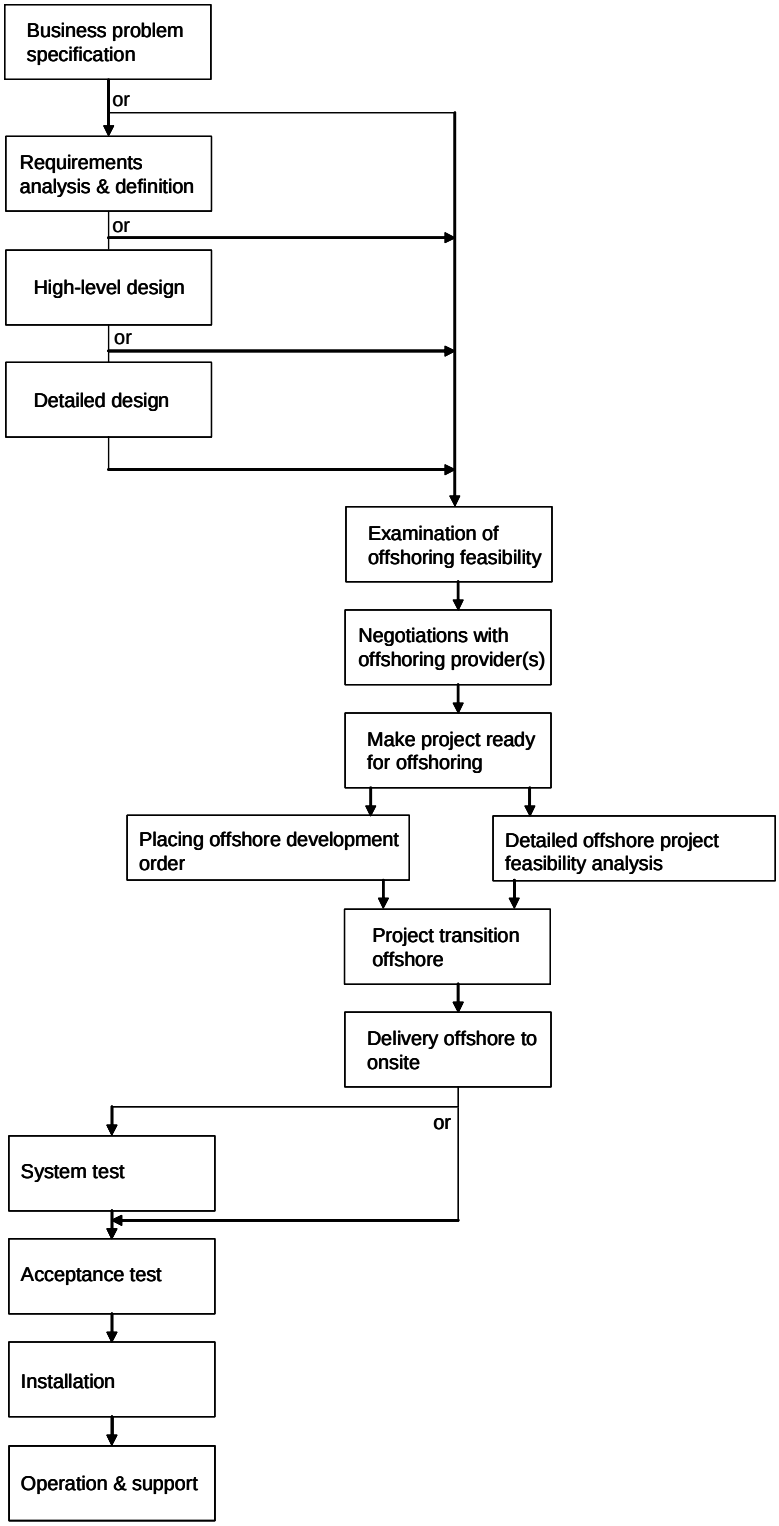

Fig. 9: A generalized offshoring life cycle model

- *Detailed offshore project feasibility:* In this stage, the customer and the offshoring provider analyze in detail if it is reasonable to assume that the outsourced tasks will be solved as expected. The offshore provider needs to collect information necessary to examine whether the required expertise, manpower, and technical infrastructure are available or can be allocated at his site in order to be able to make a definite commitment. The customer's objective is to be sure that the

partner is reliable and capable of delivering as expected. This stage includes refining the project organization, the transition plan, and the project schedule.

- *Placing the offshore development order:* Provided that the offshore partner is willing and capable of performing the outsourced tasks, both parties enter into an agreement specifying among other things the work to be done, the deliverables, and perhaps process characteristics.

- *Project transition:* The major activities in this stage are knowledge transfer and ensuring a working project-wide technical infrastructure. The objective of this stage is to make sure that the offshoring provider is able to continue the project successfully offshore. In order to acquire necessary project knowledge (e.g. domain, business-process, tool, or environment knowledge), offshore personnel may have to be trained by the customer. For this purpose, employees of the offshore provider visit the customer's organization onsite in many cases, collecting information and knowledge that they take home to disseminate among their colleagues in the project.

- *Delivery:* While a significant workload will be carried by the offshoring provider in the meantime, the next stage from the customer's point of view is when they get the result. This is supposed to be the working information system. In addition to the software, the system documentation, testing results and quality assurance reports will be handed over to the customer. Delivery usually takes place onsite, with offshore personnel available onsite to solve problems detected directly.

Depending on what tasks and stages were outsourced, the process continues at the customer's site with system testing or with acceptance testing.

Although the model in figure 9 is basically sequential, with some activities possibly going on in parallel, the offshore-specific stages need not necessarily be performed in a strict sequence nor as disjoint stages as the figure suggests. For example, RFCs and negotiations can be performed before, after, or parallel to making the project ready for offshoring. Likewise, if the offshore provider is already known, then the project feasibility study leading to a final commitment to offshore the project can be done together with the stage in which the project is made ready for offshoring.

Furthermore, there are break points in the process which are not explicitly marked in the figure. Such a point, where the project may be canceled or the process may go back to an earlier stage, is, for example, the end of the offshore project feasibility analysis. If the offshore provider cannot credibly assure that the project is in good hands, the customer may negotiate with a different provider, or the offshoring idea may be completely canceled because the problems occurring with this provider are expected to be same with other providers.

The Offshoring Provider's Perspective

Unless the offshoring provider was set up as a subsidiary or a pure captive center of a multinational enterprise, it is a normal software company in its home country, perhaps specialized in working with customers abroad. From the point of view of such a company, an offshoring project is just another project that has to be acquired on the market, bidding against competitors. As software companies, offshoring providers offer more than coding, and projects covering more stages or even the entire life cycle are more attractive to them than plain implementation or maintenance projects.
With regard to process models, offshoring providers for their part are restricted by the overall model imposed by the customer. However, the further up in the system life cycle the offshoring provider enters the process, the more freedom he has to form the process according to his needs and experi-

ence. When the entire system development process is in the hands of the offshoring provider, this organization is free to choose an iterative or evolutionary approach, for example. If the customer is ready for continuous engagement and collaboration, and if the offshoring provider has onsite personnel, even a comprehensive iterative approach like RUP may be applied.

In the beginning of the offshoring trend, maintenance as well as coding & testing projects constituted the majority of offshore projects because programmers were cheap in the offshore countries. With accumulating offshoring experience, the level of software-capabilities maturity has grown over the years. In this way, many offshoring providers were recognized as trusted partners for pre-coding stages and for outsourcing entire business functions or processes as well.

The more projects an offshoring provider successfully completes with the same customer, the closer the business relationship with that customer becomes, and the likelier are future contracts or awards in a bid. With a long-standing relationship, trust between the partners grows, and the customer may be increasingly willing to outsource more critical process stages likes requirements analysis or even the entire solution process for the business problem to the offshore partner.

Many offshoring providers offer a full spectrum of IT services to the customer. As business organizations, they naturally attempt to sell those services they can generate the most revenue with. This means that they are on a move from coding and testing to designing systems, analyzing and specifying the customer's requirements, and capturing and modeling the business problem. With business consulting services, globally acting offshoring companies have become serious competitors for the consultancy industry in Western countries.

Large organizations like Tata Consultancy Services and Infosys Technologies in India offer a full-fledged spectrum of services and products beyond what is traditionally called offshoring. They have their own proven process models applied in their projects, covering the entire software life cycle or major parts of it. On the other hand, a large number of small and medium-size offshoring providers worldwide still live largely from "traditional" offshoring projects in which the customer outsources implementation & testing or maintenance and imposes the overall process model.

5 Problems of Offshoring Application Software Development

A number of problems are posing significant challenges for offshore application software development, making offshoring projects more difficult to manage than onsite or onshore projects. Such problems are:

- *Documentation flaws:* Communication between the onsite and offshore teams is inevitably and to a significant extent based on documents. Therefore the availability and quality of the relevant documents play an important role. The more defects the requirements and design specifications contain, for example, the more misinterpretation on the developers' side will occur.

- *Different languages:* The languages on both sides may pose a problem for communication if the language is not the same and the language barrier cannot be overcome.

- *Manpower turnover:* Since manpower turnover in the booming outsourcing industry in India is high, new people may have to be integrated into the team rather frequently. Offshoring providers continuously hire new people and freelancers. Integration creates significant challenges for project management.

- *Time differences:* Different time zones may generate problems and frustration in communication. When the customer in New York City, for example, sends an urgent change request at lunchtime to the development team in Pune, India, it is almost midnight there, and the request is likely not to be processed until the next day.

- *Cultural differences:* Not only the geographical distribution but also cultural and language differences can make an offshoring project difficult to manage. Even if the customer's language is the same as the outsourcing company's, misunderstandings and problems occur because of social, religious, and behavioral differences [Brett et al. 2006]. The legal environment, civil rights, bureaucracy, and an unstable political situation of the offshoring country can be further sources of risk.

- *Resistance:* Interfacing the offshoring provider with the customer's organization, in particular with inhouse software developers, is a serious management challenge. Those who "survive" the partial outsourcing of software development might still not be supportive of the deal and resist a cooperation. The more stages of the software life cycle are outsourced, the more people in the organization are affected. When coding & testing are outsourced, plain programmers are not needed any more. When design is also commissioned to the offshoring provider, software architects are affected. When outsourcing begins with requirements analysis & definition already, systems analysts are not needed to the same extent as before.

6 Conclusions

The driving force for outsourcing AS development to offshore software companies has been and still is cost savings, yet global software companies are increasingly competing through quality achieved (or promised) with the help of mature software processes.

Project management in offshore development projects is more complex than in onsite and onshore projects. Since project management activities are bound to process steps, adequate process models reflecting both onsite and offshore tasks are needed. This paper discussed common general process models as well as offshoring-specific models. A generalized offshoring life cycle model was presented.

Taking the problems and disadvantages of offshoring AS development into account, a challenge for the outsourcer's management is to communicate the benefits of offshoring for the competitiveness of the company to the own staff and to manage the transformation process from inhouse to offshore development. In offshoring projects, there is still plenty of work left in the customer's organization. Yet this work is different, rather on the business and departmental level, e.g. preparing projects to make them ready for offshoring and identifying new opportunities for IS solutions. Personnel down the development cycle can be qualified to take on work up in the life cycle, closer to the business problem, or in the coordination of onsite and offshore activities.

A final remark refers to the risks of offshoring. Many companies fail to manage risks. A proper risk assessment and mitigation plan should be prepared in advance [Morrison, Macia 2005]. Infosys Technologies, for example, include a detailed plan for risk identification, monitoring and mitigation as part of project planning. This plan covers risk identification, prioritization and mitigation options. The status of risks is continuously tracked and reviewed using a monthly milestone mechanism [Infosys 2006]. As organizations consider the vast benefits and allure of offshore outsourcing, they must balance the risks and uncertainties with the potential for labor arbitrage [Davision 2003].

References

Ambler, S.W.: A Manager's Introduction to The Rational Unified Process (RUP), Version Dec 4, 2005; http://www.ambysoft.com/downloads/managersIntroToRUP.pdf (accessed Dec 20, 2006).

Anderson, D.J.: Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results; Prentice Hall, Upper Saddle River, NJ 2004.

Beck, K., Andres, C.: Extreme Programming Explained: Embrace Change, 2nd Edition; Addison-Wesley Professional, Boston, MA 2004.

Brett, J., Behfar, K., Kern, M.C.: Managing Multicultural Teams; Harvard Business Review 84 (2006) 11, pp. 84-91.

Chaudhuri, D.: Off-shoring Process, Learnings and Case Studies; Presentation at Euroforum Seminar "Offshore-IT-Projekte erfolgreich managen", Munich (Germany), Feb 17, 2004; online available at http://www.competence-site.de/offshore.nsf (accessed Jan 12, 2007).

Cockburn, A.: Crystal Clear: A Human-Powered Methodology for Small Teams; Addison-Wesley Professional, Boston, MA 2004.

Davison, D.: Top 10 Risks of Offshore Outsourcing; December 9, 2003; http://techupdate.zdnet.com/techupdate/stories/main/Top_10_Risks_Offshore_Outsourcing.html (accessed Dec 16, 2006).

DC&M Partners LLC: Offshore Development Process Model; http://www.dcm-partners.com/offshore-development-process-model.htm (accessed Dec 7, 2006).

Kurbel, K., Pietsch, W.: A Cooperative Work Environment for Evolutionary Software Development; in: Gibbs, S., Verrijin-Stuart, A.A. (Eds.), Multi-User Interfaces and Applications; Amsterdam et al. 1990, pp. 115-127.

Lee, O.-K., Banerjee, P., Lim, K.H. et al.: Aligning IT Components to Achieve Agility in Globally Distributed System Development; Communications of the ACM 49 (2006) 10, pp. 49-54.

Martin, R.C.: Agile Software Development – Principles, Patterns, and Practices; Prentice Hall, Upper Saddle River, NJ 2003.

Morrison, P., Macia, M.: Offshoring: All Your Questions Answered; http://www.alsbridge.com/outsourcing_leadership/dec2005_offshoring.shtml (accessed Dec 16, 2006).

Ramesh, B., Cao, L., Mohan, K., Xu, P.: Can Distributed Software Development Be Agile?; Communications of the ACM 49 (2006) 10, pp. 41-46.

Royce, W. W.: Managing the Development of Large Software Systems: Concepts and Techniques; IEEE WESCON Technical Papers, Western Electronic Show and Convention, Los Angeles, Aug. 25-28, 1970, pp. 1-9; reprinted in: Riddle, W.E. (Ed.), Proceedings of the 9th International Conference on Software Engineering, Monterey, CA; IEEE Computer Society Press, Los Alamitos, CA, March 1987, pp. 328-338.

T-Systems India: Competencies, Service Offerings & Projects – Business Solutions: Offshore Delivery Model (Application Development); Company Presentation, Nov 2005.

Wells, D.: Extreme Programming: A Gentle Introduction; http://www.extremeprogramming.org/ (accessed Dec 10, 2006).