IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4931

Extreme Outsourcing

Jakob H. Iversen
College of Business Administration, University of Wisconsin Oshkosh, 800 Algoma Blvd., Oshkosh, WI 54901,
iversen@uwosh.edu

ABSTRACT

As companies are looking at reducing their software development cost, they are increasingly turning to offshore outsourcing to take advantage of lower wages overseas. This paper looks at this and another major trend in software development, agile development methodologies, to determine whether the practices of the most common agile methodology, extreme programming, can support offshore outsourcing. The paper concludes that while some practices are easily implemented, others require a great deal of consideration and may not be feasibly implemented at all. Offshore outsourcing can benefit from extreme programming practices by achieving higher visibility into the development process.

INTRODUCTION

Even as the IT world finds itself in a severe crisis with high unemployment rates, the second half of this decade may very well hold just the opposite: a severe lack of qualified software developers, database administrators, network administrators, and IT support staff. It has been estimated that by 2010, there will be a gap of 5.1 million skilled workers (Kaihla, 2003). This demographic trend is likely to hit software development companies particularly hard, as enrollments in computer science, engineering, and MIS programs have declined over the past years at the same time as eight of the ten fastest growing job occupations in the US are predicted to be information technology occupations, and three of these are in software development (Table 1).

The last decade has been one of continuous and unrelenting global competition in the software industry. A growing lower wage and a highly skilled software engineering labor force in India and other Asian countries has precipitated shifts in software production from Europe and North America to Asia. In response to these challenges senior managers in European and North American software industries are adopting a variety of strategies to stay competitive. For example, some firms are outsourcing software development to India, the Philippines, Russia, and other countries with a skilled work force demanding lower wages (Ebert & Neve, 2001; Thibodeau, 2003).

Simultaneously, software companies are faced with more rapidly changing business conditions and requirements to the software they are developing. To deal with this challenge, many software companies are investigating several new ways of organizing and conducting software development, commonly referred to as *agile methodologies* (Fowler, 2003). One of the most popular of these methods is eXtreme Programming (XP) (Beck, 2000), in which a small team of co-located developers and customer representatives work intensively on rapidly developing a solution to a particular situation in the customer organization.

The next section describes the history and reasons for offshore development, including some of the problems and opportunities companies experience when outsourcing to a foreign entity. Section 3 gives an overview of extreme programming, and section 4 shows how to combine the two trends to support agile development between on-shore and off-shore entities. Finally, section 5 provides a summary and conclusion as well as avenues for further research.

Table 1: Ten fastest growing job occupations in the US. Employment numbers are in thousands (BLS, 2001).

Occupation	*2000*	*2010*	*Absolute change*	*Percent change*
1. Computer software engineers, applications	**380**	**760**	**380**	**100**
2. Computer support specialists	506	996	490	97
3. Computer software engineers, systems software	**317**	**601**	**284**	**90**
4. Network and computer systems administrators	229	416	187	82
5. Network systems and data communications analysts	119	211	92	77
6. Desktop publishers	38	63	25	67
7. Database administrators	106	176	70	66
8. Personal and home care aides	414	672	258	62
9. Computer systems analysts	**431**	**689**	**258**	**60**
10. Medical assistants	329	516	187	57

OFFSHORE DEVELOPMENT

When companies decide to move a portion of their development activities overseas, there are two main underlying reasons: cost savings and access to a larger labor pool (Carmel & Agarwal, 2001). In the current weak economy, the former appears to be the most prevalent reason for outsourcing work (Perez, 2003). It is estimated that companies can save between 20 and 70% of the cost of developing software in the United States by outsourcing to India (Kling, 2003; Vijayan, 2003).

So far, most outsourced work has gone to companies in India, where developers have a reputation for high quality work. Indian companies are increasingly competitive with American and European firms, but wages are starting to increase too, causing work to be shifted to former Eastern European countries such as Russia and Romania, as well as less developed Asian nations such as Philippines and China. A recent survey of 252 US IT managers showed that 38% are currently outsourcing IT work to India. The second most popular destination, China, was only used by 6% of respondents (Vijayan, 2003).

When selecting a region for outsourcing, companies typically consider the following factors:

- Language. The ability to communicate with workers at the outsourcing site is greatly increased with a common language. For US companies, the ability to communicate in English is a great asset when considering where to place outsourcing. Canada and Ireland naturally have an advantage on this point, but India has also gained a reputation for good English language skills among its developers.
- Distance. The further separated the on-shore and off-shore sites are, the more difficult it is to visit physically. In some third-world countries, a poor infrastructure may further add to the travel time required to reach the site. Electronic communication may to some extent alleviate these distances, but time zone difference is also a barrier; when developers in India meet for work, their American counterparts will just be ready to leave, giving very little ability to communicate in real time by either phone or video conference. On the other hand, companies can also use time differences around the world in a "follow-the-sun" approach, where teams around the world package their day's work and pass it on to the team eight time zones to the west that is just getting ready to meet for work. Having development centers in the US, India, and Europe can attain such a scheme (Carmel & Agarwal, 2001).

- Technical quality. One of the main reasons for going overseas is to have access to a larger pool of qualified labor. However, in order for development to be successful, it is important that the technical quality of the developers is sufficient that they be able to perform the required tasks. In India, Ireland, and Canada this is generally true whereas in China, Romania, and Russia, it is more difficult to find the required technical talent.
- Cost. Lower wages is the major reason for sending work off-shore. But there are also great differences between different regions around the world. But the wages generally correspond to the technical quality and language skills of the developers. Whereas developers in Canada are cheaper than US workers, the difference is not as great as that of Indian developers and especially Chinese and Romanian developers. However, the technical quality and language skills of Canadian developers exceed those of the off-shore developers, making Canada a good choice for many outsourcing companies (Horowitz, 2003).

Companies who have outsourced some of their software development have encountered a number of problems over the years. One of the most difficult problems is that the visibility into the off-shore process is low. This and other problems increase the risk of the software project. In the past, the solution to this has typically been to develop more comprehensive documentation and conduct more up-front analysis and design. However, such approaches also slow down progress on the project.

AGILE DEVELOPMENT

Agile Development methodologies have been used in software development since the 1950s under a number of different names, such as iterative, incremental, and evolutionary development (Larman & Basili, 2003). Agile development is mostly seen as a contrast to plan-driven methodologies such as the waterfall model. However, there has recently been an effort to determine how agile and plan-driven approaches can inform and benefit from each other (Boehm & Turner, 2003).

Extreme Programming (XP) is one of the most popular forms of agile development. Developed in the late 1990's by Kent Beck and Ward Cunningham it focuses on empowering small teams to determine the best process for quickly developing a solution. The process is usually characterized by the 12 practices described below (Jeffries, 2001):

- ***Whole Team.*** The whole team is co-located and has a customer representative on the team (Baheti et al., 2002).
- ***Planning game.*** One of the characteristics of XP is constant planning of small increments of the system. This involves two practices: 1. *Release planning*, which specifies which features will be included in which releases. The release plan is only accurate for next release, but is updated regularly. 2. *Iteration planning*, which is the plan for a two-week iteration, at the end of which a running version of the software is available. The customer supplies the desired functionality for the iteration, but the developers ulti mately decide which features will be included based on historical performance data. These two planning steps allow for high visibility into the process at any given time.
- ***Customer Tests.*** Every time a new feature is proposed by the customer, the customer also specifies what tests will determine if the feature is implemented correctly. These tests are imple mented and run as automatic regression tests that the system will be tested against in any future tests. This type of test-first development is now supported by many IDEs through unit testing, examples include Eclipse (www.eclipse.org), BlueJ (www.bluej.org), and JBuilder (www.borland.com/jbuilder).
- ***Small Releases.*** Each iteration results in a running system that is released to the customer, either to be used for evaluation or for actual deployment to end users.
- ***Simple Design.*** The focus in XP teams is to constantly work on simplifying the design of the system. The goal is to find the simplest solution that could possibly work (Beck, 2000).
- ***Pair Programming.*** One of the most controversial features of XP is its reliance on pair programming. In XP, production code is always developed by two programmers working in concert at a single computer. This extends the research that has shown that one of the most effective cost and quality control techniques is peer review (Fagan, 1986). In pair programming, peer review is constant, as every line of code is immediately reviewed by a second programmer (Williams & Kessler, 2000). Pair program ming in XP is considered one of the primary vehicles for communication and knowledge exchange among members, and the pairs are therefore shuffled frequently.
- ***Test-Driven Development.*** Developers who use XP, believe that the best way to ensure continuous quality of the system, is to create a set of test cases that are run automatically and regularly to determine any problems in the system. As described under Customer Tests, the test cases are written before the code that they will test, and only when the resulting code passes the test, is a given feature complete. Any time the system fails a test, all further development stops until the system passes all tests. This provides immediate and rich feedback to the developers on the state of the system, and ensures that there is always a running version of the system.
- ***Design Improvement / Refactoring.*** A striking characteristic of XP, is that developers are constantly seeking ways to remove duplication, lower coupling, and increase cohesion. The philoso phy is that time spent on improving the design, will be regained later as new features need to be added. Developers are expected to identify and correct any signs of poor design, primarily by making individual classes well designed, but also by improving connections between classes.
- ***Continuous Integration.*** In the past, many projects have expe rienced problems when they attempted to integrate many mod ules written by disparate teams or individuals. Integration was often one of the last activities to be carried out. In XP, integration is an ongoing activity that is carried out at least daily. This practice supports test-driven development in ensuring that the system is always operational.
- ***Collective Code Ownership.*** Traditionally, individual program mers or teams would be responsible for each their own part of the code. If a developer needed changes made in someone else's code, this may have taken a long time, causing the change to be made as duplication in the developer's own code base. In XP, every developer has access to every part of the system, and can modify every line of code. To ensure that this does not degenerate the code base, the test-driven approach ensures that any errors are caught early, and pair programming can help developers under stand parts of the code by teaming with someone who is an expert on that part of the code.
- ***Coding Standard.*** Adherence to a strict coding standard also helps counter the ill effects of unchecked collective ownership as well as ensure future maintainability.
- ***Metaphor.*** To ensure that everyone works in the same direction, XP teams develop a common vision, or "metaphor" for the system. The metaphor is "a simple evocative description of how the program works".
- ***Sustainable Pace.*** The XP methodology realizes that program mers have lives outside of the workplace, and as such prescribe that teams recognize the need to work at a pace that doesn't burn out the individuals. Overtime may be needed in certain periods, but it should be the exception, not the norm.

AGILE OFFSHORE DEVELOPMENT

As offshore development becomes increasingly widespread, the problems that will be tackled by dispersed teams in offshore settings are likely to become increasingly complex, as the simpler problems have already been solved (Rothman, 2003).

In addition, it is likely that as the pace of business increases, the uncertainty of most projects will also increase, making it difficult to make detailed up-front design that can subsequently be passed on to a

low-level outsourcer for programming. Instead, projects will increasingly employ developers who are capable of capturing and understanding changes to user requirements and implement them rapidly.

As companies may be forced to outsource high level work, XP is one methodology that could help successfully complete highly complex, uncertain, and geographically dispersed projects. XP was designed to handle projects executed under uncertain conditions, and there is good evidence that XP teams are able to successfully complete such projects (Murru *et al.*, 2003; Rasmusson, 2003; Schuh, 2003).

However, XP was also developed for small colocated teams working on fairly short term schedules. Williams and Cockburn (2003) observe that most experts have agreed that agile methods *"best suit colocated teams of about 50 people or fewer who have easy access to user and business experts and are developing projects that are not life-critical."* Others have, however, argued that XP may be amended for complex, long-term projects. Crocker (2001), for instance argues that adding a team coordination layer will allow XP to be used with loosely coupled collaborating teams. This coordination layer would replace the metaphor with a more structured "Up-front Architecture Light", add a Liaison role to each team, and amend the planning game with a team-version where representatives from each team plan the work for the overall project. In addition, one team acted as a link between the real customers and the developers.

In the following, I will examine the XP practices and assess to which extent they are able to support geographically dispersed teams, and to which extent they need updating or replacing. The discussion is divided into three parts according to how easy each practice is to apply in a global outsourced project.

Practices to Follow on Any Project

Extreme Programming builds on sound software engineering principles taken to the extreme. Some of these principles are fairly simple to apply in any project, outsourced or not and include *Design Improvements/Refactoring*, *Coding Standard*, and *Sustainable Pace*.

Practices to Adopt With Little Effort

Some of the practices require more foresight before applying in an offshore outsourced setting. These generally require subscription to an agile development philosophy and can therefore typically not be adopted without making significant concessions to the agile methodology.

Small Releases should work with no problems. Elssamadisy (2001) recommends staying with two-week iterations on a 50-person colocated team. Such a short cycle may be more difficult to attain on a dispersed project, but the releases should be kept small enough that iterations take no longer than four weeks.

Test-Driven Development could be used without problems, provided that the development environment supports unit testing, and that the necessary tool support for *Continuous Integration* is in place for all teams. If all teams aren't able to continuously integrate, the unit tests may be of little value as the regression tests will not run on the entire system.

Simple Design should be a goal; although it may become a problem as the system itself becomes larger in scope. However, simple design may in itself help the team manage the complexity of the system scope.

Practices Requiring Considerable Consideration

The very different nature of globally distributed projects and typical XP project requires considerable effort in implementing several of the XP practices. Since the purpose for global outsourcing is to have more than one team, the *Whole Team* cannot stay together. However, to the extent that teams can work autonomously, each subteam may use XP and thus use this practice locally. For many projects, the major problem will be having a customer working on the team. This may be solved by using proxy customers in the form of analysts that are located close to the actual customer (Crocker, 2001). This also means that *Customer Tests* become more difficult to carry out. In addition to having proxy customers, tool support may also help to allow the customers to define test cases.

Crocker (2001) suggests that the *Planning Game* be adapted to work across teams, but this requires more formalism than what is traditionally done in XP. This could be done by letting the release plan remain global, but the iteration plan local.

As a vehicle for communication throughout the team, *Pair Programming* is not suitable, because the communication links would be difficult to manage across distances temporal and geographical distances. However, as a way to exchange technical knowledge and increase quality, this practice is still valuable (Baheti et al., 2002). In addition, tool support is becoming more readily available (Mezick, 2003).

Collective Code Ownership might be problematic as the project scales. There may be a need for quality control to ensure that conflicting changes aren't checked in. However, the continuous integration and test-driven development may be a simpler and cheaper solution on most projects, as any problems would be detected immediately.

The *Metaphor* has been rejected as both too weak (Crocker, 2001) and too complex (Elssamadisy, 2001) for large projects. The metaphor is likely valuable on any project, but may need to be more formalized to be communicated effectively and precisely to distributed teams. However, there is some evidence that the effectiveness of the metaphor, even in smaller settings, isn't as good as it should be (Herbsleb et al., 2003).

CONCLUSION

So far, only a few projects have attempted using offshore and agile development (Simons, 2002). The results from those efforts indicate that some of the key values of agile methods (close customer involvement and communication) are more difficult to attain. On the other hand, some of the XP practices such as continuous integration and frequent deliverables solve significant problems in traditional outsourcing projects by providing a high degree of transparency in the processes, as well as increasing the project velocity.

This research has pointed to some of the areas where XP can be adopted easily for outsourced projects, and where there might be problems. As companies continue to explore offshore outsourcing, they should find value in carefully employing XP practices and other agile methodologies to support those efforts.

References

Baheti, P., Williams, L., Gehringer, E., Stotts, D., and Smith, J. M., 2002. Distributed Pair Programming: Empirical Studies and Supporting Environments (Technical Report TR02-010), Chapel Hill, NC: Department of Computer Science, University of North Carolina.

Beck, K., 2000. Extreme Programming Explained, Addison-Wesley

BLS, 2001. BLS RELEASES 2000-2010 EMPLOYMENT PROJECTIONS (USDL 01-443), Washington, D.C.: United States Department of Labor, Bureau of Labor Statistics.

Boehm, B., and Turner, R., 2003. Using Risk to Balance Agile and Plan-Driven Methods, Computer. 36, 6, 57-66.

Carmel, E., and Agarwal, R., 2001. Tactical Approaches for Alleviating Distance in Global Software Development, IEEE Software. 18, 2, 22-29.

Crocker, R., 2001. The 5 reasons XP can't scale and what to do about them, Proceedings of the 2nd International Conference on eXtreme Programming and Flexible Processes in Software Engineering, Villasimius, Italy, 62-65.

Ebert, C., and Neve, P. D., 2001. Surviving Global Software Development, IEEE Software. 18, 2, 62-69.

Elssamadisy, A., 2001. XP On A Large Project – A Developer's View, Proceedings of the XP Universe 2001

Fagan, M. E., 1986. Advances in Software Inspections, IEEE Transactions on Software Engineering. 12, 7, 744-751.

Fowler, M., 2003. The New Methodology, http://www.martinfowler.com/articles/newMethodology.html. Retrieved: September 3, 2003.

Herbsleb, J., Root, D., and Tomayko, J., 2003. The eXtreme Programming (XP) Metaphor and Software Architecture (CMU-CS-03-167), Pittsburgh, PA, USA: School of Computer Science, Carnegie Mellon University.

Horowitz, A. S., 2003, September 15. Canada: Safe, secure and 'near-shore'. ComputerWorld.

Jeffries, R., 2001. What is Extreme Programming?, http://xprogramming.com/xpmag/whatisxp.htm. Retrieved: October 3, 2003.

Kaihla, P., 2003, September. The Coming Job Boom. Business 2.0.

Kling, J., 2003, September 15. IT's Global Itinerary: Offshore Outsourcing Is Inevitable. ComputerWorld.

Larman, C., and Basili, V. R., 2003. Iterative and Incremental Development: A Brief History, Computer. 36, 6, 47-56.

Mezick, D., 2003, August 18. Outsourcing 2.0: Collaborative Development. ComputerWorld.

Murru, O., Deias, R., and Mugheddu, G., 2003. Assessing XP at a European Internet Company, IEEE Software. 20, 3, 37-43.

Perez, J. C., 2003, January 30. Gartner: Offshore outsourcing gains steam. ComputerWorld.

Rasmusson, J., 2003. Introducing XP into Greenfield Projects: Lessons Learned, IEEE Software. 20, 3, 21-28.

Rothman, J., 2003, September 15. 11 Steps to Successful Outsourcing: A Contrarian's View. ComputerWorld.

Schuh, P., 2003. Recovery, Redemption, and Extreme Programming, IEEE Software. 18, 6, 34-41.

Simons, M., 2002. Internationally Agile, InformIT.

Thibodeau, P., 2003, August 18. Soviet Skills Draw R&D Work. ComputerWorld.

Vijayan, J., 2003, September 15. India Inc., Still Going Strong. ComputerWorld.

Williams, L., and Cockburn, A., 2003. Guest Editors' Introduction: Agile Software Development: It's about Feedback and Change, Computer. 36, 6, 39-43.

Williams, L., and Kessler, R., 2000. All I Really Need to Know About Pair Programming I Learned in Kindergarten, Communications of the ACM. 43, 5.