ISSN 0005-1144
ATKAFF 51(1), 89–98(2010)

Liguo Yu, Alok Mishra

Risk Analysis of Global Software Development and Proposed Solutions

UDK 004.413.4
IFAC 2.8.3

Review

Global software development is becoming a widely accepted practice in software industry. While benefits of global software development have been identified and publicized, potential risks have not been fully investigated and addressed. This paper analyzes the impacts of globalization on software development, especially its long term impact on software product quality and software industry competitiveness. This issue is rather critical for prominent software providers. Potential solutions are discussed to address these issues in order to reduce the risk and take advantage of the benefits of global software development.

Key words: Global Software Development, Risk, Evolution, Global Software Process, Agent

Analiza rizika globalnog razvoja programske podrške i prijedlog rješenja. Globalni razvoj programske podrške postaje naširoko prihvaćen način rada u industriji programske podrške. Dok su njegove dobre strane uočene i obznanjene, mogući rizici nisu u potpunosti istraženi i odgovarajuće adresirani. Ovaj članak analizira učinke globalnog razvoja programske podrške, posebice njegov dugoročni utjecaj na kvalitetu programa i konkurentnost industrije programske podrške. Taj je moment razmjerno kritičan uglednim proizvođačima programa. Diskutirana su moguća rješenja za adresiranje tih problema kako bi se rizik umanjio i iskoristile dobre strane globalnog razvoja programske podrške.

Ključne riječi: globalni razvoj programske podrške, rizik, globalni procesi programske podrške, agenti

1 INTRODUCTION

Over the past decades, economy has converted national markets into global markets, creating new forms of competitions and collaborations [1]. One such effect of economy globalization takes place in software development, where software products are developed by teams geographically distributed in a worldwide scale. This gave rise to the formation of distributed teams and created opportunities for restructuring traditional working ways to benefit from nations' different competitive advantages [2]. This is called global software development [3–5]. The major driving force of global software development is the easily identified benefits: capacitated professionals around the world could be hired or contracted with lower cost [6, 7] and increased development speed. It is a fact that software community appreciates an economy of merging diverse development skills and domain expertise. As communication media become more sophisticated; more companies are pushed towards a global software development as this would be cost effective and competitive than developing software product in one centralized building, company or even country [8].

Currently, the research in global software development focuses on resolving the issues of spatial distance, time zone difference, and cultural differences, which constitute the major challenges for distributed software development [9–12]. Different communication methods and coordination methods are proposed and different project management and configuration management software is developed to facilitate global software development [13–17].

Because the short term benefits of global software development are so appealing, more and more software companies begin globalizing their software development team and effort. Although the management of global software development is an arduous task and may have the risk of lowering overall productivity, its positive impact should not be overlooked. A major positive effect is innovation [18].

However, little attentions are given to the long term benefits/risks. In other words, although a moderate research has been performed [19, 20], the risk of globalization of software development, especially its long term cost, has not been fully identified in either research or practice. Ralyte et al. [21] also observed the fact that there is not sufficient research or reported practice in terms of identifying potential problems in managing, distributed project en-

vironment and evaluating the project risks. They further found that only very few solutions are proposed and most of them are based on developer's experience, interviews and case studies. Recently Mishra and Mishra [22] found that areas like quality and risk management issues could get only scant attention in distributed information system development. There is also need of further studies in non-technical areas such as team dynamics and cross-cultural risk management [23].

This paper analyzes the risk of global software development, especially the potential long term cost. The cost is related to both product quality and software companies' competitiveness. The remainder of the paper is organized as follows. Section 2 discusses the potential risks associated with global software development. Section 3 addresses each potential cost and provides some suggestions. The conclusions are in Section 4.

2 POTENTIAL RISKS IN GLOBAL SOFTWARE DEVELOPMENT

Compared to localized software development, global software development has various risks, such as those induced by the spatial and temporal distance between development teams. Prikladnicki et al. [8] identified the significance of risk management in global software development and suggested that managing global software development entails the distributation of the risk management instead of traditional risk management processes in order to be able to assess the possible impacts of the dispersion, the diverse cultures, time, attitudes and requirements elicitation. In this paper, we have focused on the risks in product quality and software company competitiveness.

- **Product quality.** Software quality is strongly dependent on the quality of the process used in its preparation and software process can be defined, managed, measured and improved [8]. Prikladnicki et al. [8] further considered this as one of the most important success factors for distributed projects. In localized software development, each development phase is enforced with quality assurance. These quality assurance principles and rules have been well established. However, for global software development, developers are distributed around the world; they might follow different software processes and use different strategies. How could we guarantee that the globally developed product has the same quality as the locally developed? This issue can become even more critical if several, temporarily-contracted distributed teams are utilized for global software development. It has been reported recently that the digital photo frame made in China contains virus [24].

  In a recent article Sangwan et al. [25] reported about a global software development project in which the technical team was forced to distribute the development of parts of the system across geographically distributed teams to achieve the compressed schedule via parallel development efforts. When the components developed by the teams were integrated together, they blew up the memory and performance budget. While individual components were carefully crafted, not enough attention had been given to the overall system goal of achieving high performance within the given resource constraints. The end result was not the desired product and the discrepancy between the desired outcome and the actual outcome cost the company hundreds of millions of dollars spent developing the system and billions of dollars in potential revenue. Pilatti and Audy [26] suggested as lesson learned from their global software development case study that quality certificates are important because they define how the process will be structured and applied to each other.

- **Design quality.** In general, the source code quality is easy to assess through measuring the fault density [27]. However, it is more difficult to evaluate the design quality of a software product being developed at various sites by adopting different software process improvement methodologies and mechanisms. For instance the modularization approach is a very useful tool for dividing the development of a complex software system into manageable units, still some technical dependencies will remain, creating task dependencies that could be difficult to identify and manage [28]. Moreover, geographically distributed development teams are at disadvantage because of the negative impact of distance on the engineers' ability to communicate and coordinate their work [11]. Hence there is need to complement modularization with appropriate mechanisms to identify relevant work dependencies and, consequently, maintain suitable levels of communication and coordination among teams developing interdependent modules [28]. Cataldo et al. [28] further observed that global software development organizations would benefit from mechanisms that allow the identification of the changes in dependencies such that manager and developers are notified of those changes and can react accordingly. Ramnath [29] also observed that early establishment of architecture, design and a process standard was critical. Bass and Paulish [30] mention that technological factors can limit *design choices* to hardware, software, architecture, platform and standards that are currently available. Further, due to the shortage of standardized tools and metrics on design quality, it makes the

monitoring of the design quality of globally developed software even more challenging.

- **Maintenance and evolution.** Maintenance and evolution are important phases in software life cycle. Software products must be continually updated to satisfy new requirements and new environments. Pilatti and Audy [26] supported this as lesson learned from their global software development case study that software maintenance was main concern for global software development projects. On the one hand, to make a software product easy to maintain, localized software development requires a consistent and complete documentation in the development process. How could this in principle be easily enforced on global software development? Prikladnicki et al. [8] also observed the importance of analyzing the level of documentation available for the offshore team. Cataldo et al. [28] mentioned the importance of documentation in global software developments as "the central team used the architectural documentation to identify dependencies among components early in the development process and represented these as a design structure matrix (DSM) which was used to identify the set of tasks to be assigned to each remote team that would minimize the dependencies and consequently, minimize the need for coordination between remote teams". On the other hand, software maintenance and evolution process is so different from development processes, that, different strategies should be used to manage the global software development and global software maintenance. For instance, when a problem appears on a customer's location it is sometimes difficult to send the right developer or consultant to resolve it because this person is located on a geographically remote site and thus it is difficult for the customer and the development team to maintain a good level of support [21]. Yan [31] found maintenance a very important step in global software development and insists on the necessity to establish agreements with the customer in order to find a support solution in case of problems requiring a quick answer and proposes the formation and employment of *technical liaison engineers* who should be qualified enough to handle most of the urgent maintenance problems and to reassure the customers. Ralyte et al. [21] concluded that development and maintenance of distributed software development projects should be taken into account during distributed software development project management.

- **Continuous evolution and restructuring.** There are basically two kinds of software evolution: continuous evolution and restructuring. Continuous evolution refers to small changes that are continually made to a software product, which results in multi versions of the product. Restructuring refers to major changes made to a software product to improve its quality and/or performance. In other words, continuous evolution involves fewer maintenance activities and less effort while restructuring involves more maintenance activities and more effort. Architectural definition or high-level design activities could take place in various points in time in the lifecycle of the project and analyses showed a significant amount of the communication in discussion forum had to do with design and architecture [28]. Cataldo et al. [28] further suggested the need for mechanisms to identify communications that involve architectural definitions or high-level redesign and facilitate organizational awareness of such situations in order to appropriately address the implications and impact of such changes.

Meadows [32] highlighted some progresses in software engineering field that makes global software development viable. These progresses include (i) increased application of component-based software development, (ii) increased standardization of modeling languages and programming languages, and (iii) increased use of standardized project management tools. These advances in software engineering could partially help to avoid the risks identified above. However, they cannot fully resolve these risks. For example, component-based software development and standardized programming language can only be helpful in software implementation, they are little helpful in analysis and design. In fact, a lot of time can be lost during for example the phases of implementation and test in global software development [33]. Figure 1 illustrates the development and maintenance of a software product in localized software development.

Figure 2 shows two icons we use to represent localized development and global development. Usually, with respect to leading US software companies, localized development has high cost but low potential risks, while globalized development has low cost but high potential risks. Trust, which is a significant factor that glues all success factors in such projects, decreases when there is a high number of contractors in a project [26] and this may also trigger potential risks [34].

Figure 3 is a simple conversion of localized software development to global software development. Because every phase is under global environment, we refer to schemes in Fig. 3 as fully global software development and maintenance process. In the next section, we will discuss whether the fully global software development and maintenance process can avoid the risks identified above.

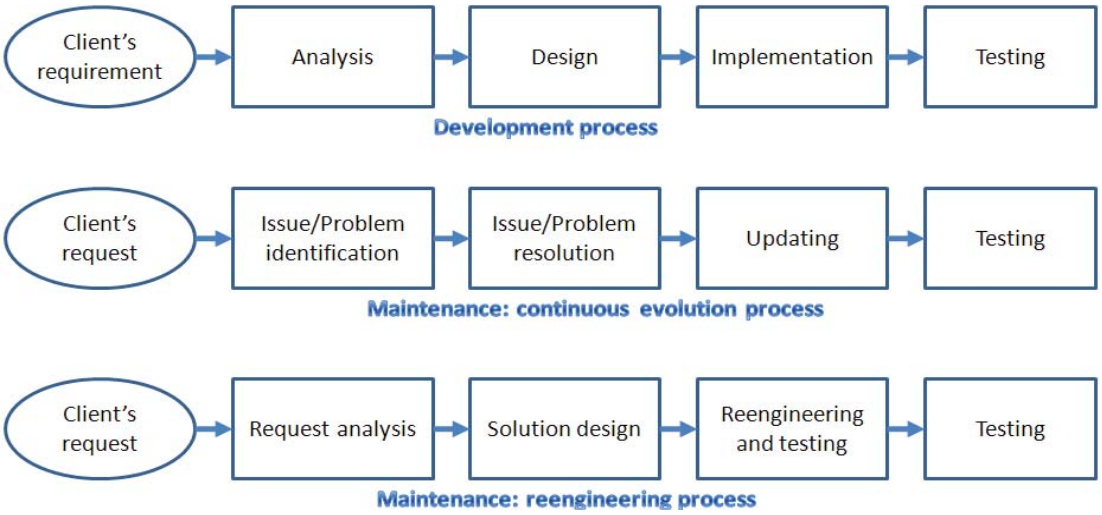

Fig. 1. The general (localized) software development and maintenance process

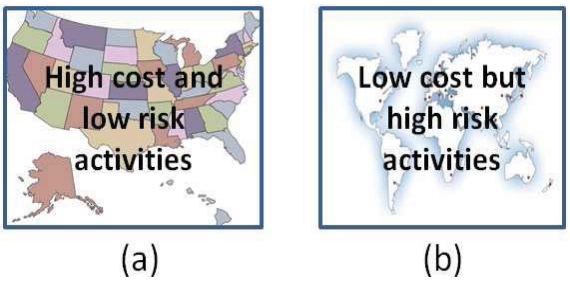

Fig. 2. (a) Localized software development; and (b) global software development

3 PROPOSED SOLUTIONS

In this paper, we identify three stakeholders in global software development. They are *client*, *primary agent*, and *secondary agent*. Client is the individual who wants the product to be developed or evolved. Primary agent is the software company that the client directly contracts with. Primary agent delivers the final product to the client. Secondary agent is the software company that the primary agent subcontracts with. The secondary agent develops parts or whole of the product as required by the primary agent. Figure 4 shows the relations between client, primary agent, and secondary agent, in which we assume that both the client and primary agent are located nationally while secondary agents are globally distributed.

3.1 Software development

Figure 3 shows the fully global software development process. This process has the largest potential risk: from analysis to testing, every artifact of software product, such as Software Requirement Specification (SRS), Software Design Specification (SDS), is produced globally, which increase the risks of software quality degradation and project failure. Ramnath [29] observed that understanding domain requirements as well as the complete architecture of the product lines being developed needed to be a constant focus for both on-shore product group and as well as the off shore one.

As analysis and design processes are less standardized and their qualities are difficult to monitor, we propose that these phases should be developed locally. In contrast, implementation and unit testing are recommended to be performed globally to take the advantages of the standardization in programming language and integrated development environment. The corresponding process is illustrated in Fig. 5, in which, each distributed team works on one component. The integration and testing activities are centralized. Note in Fig. 5, Quality Assurance (QA) and Standardization such as Capability Maturity Model Integration – (CMMI) must be enforced in distributed sites.

We restate here that, first the development process proposed in Fig. 5 should support the component-based software development. Because each software component has clearly defined interface, it is easy to test the correctness and performance of software components, which allows the primary agent (the company that contracts with the client and outsources that project) to directly monitor its quality. Second, the global development activities must be standardized. Here, it is important to note the observations of Ralyte et al. [33] that during the test phase, a number of problems emerged because of the lack of standardization,

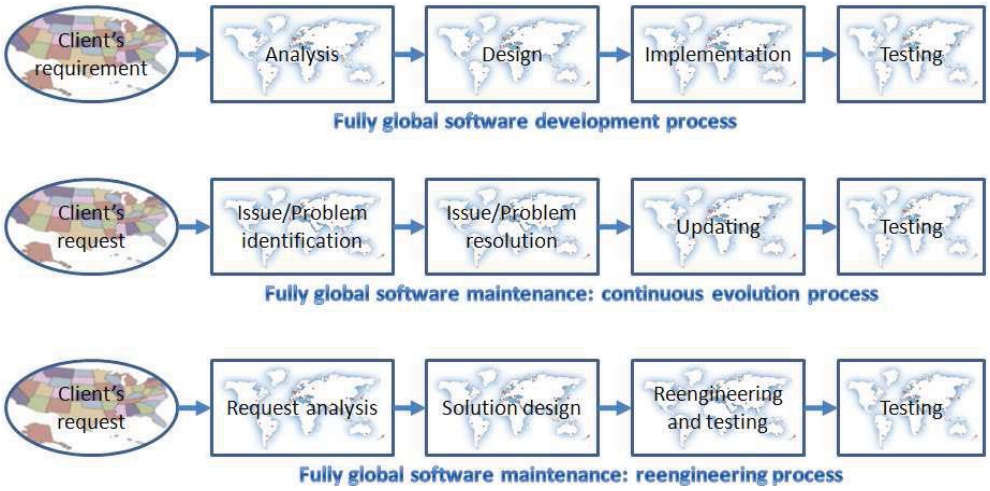

Fig. 3. Fully global software development and maintenance process

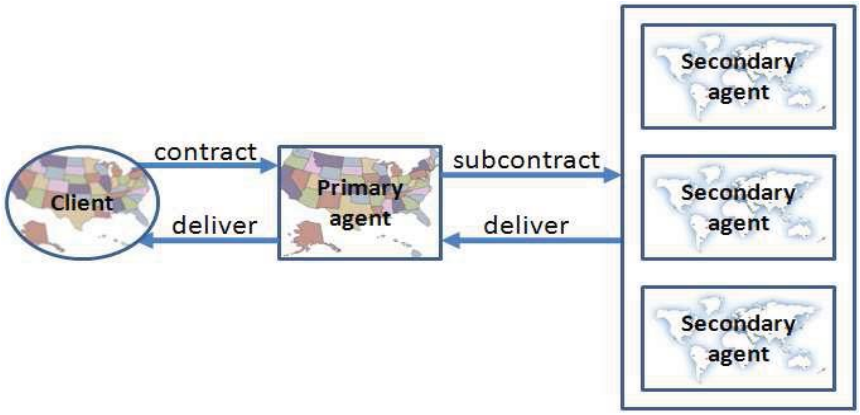

Fig. 4. The relationship between client, primary agent, and secondary agent

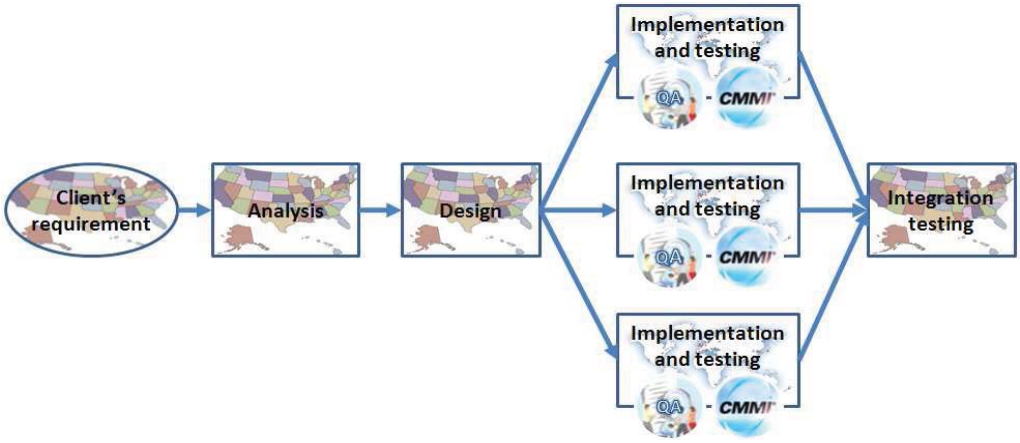

Fig. 5. Partially globalized software development process

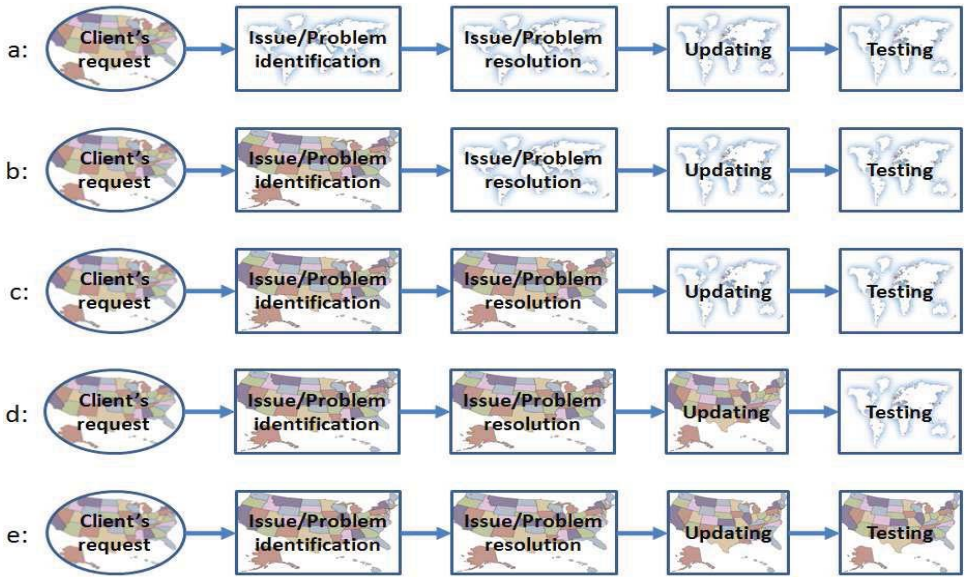

Fig. 6. Different scenarios of maintenance: continuous evolution process

different test platforms and databases and difficulty to reproduce errors. The primary client company should promote its standard of product and process to the secondary agent companies globally. This means that all distributed sites should follow the same standard. This standardization will not only make the quality control easier, but also reduce the complexity of integration.

3.2 Continuous evolution process

Similar to the development process, fully global continuous evolution process illustrated in Figure 3 might not be the best strategy. To evaluate this strategy, Fig. 6 shows the different scenarios of maintenance: continuous evolution process.

In the following, we discuss each scenario in Fig. 6 in detail.

- Scenario *a* is known as the fully globalized maintenance and evolution. This scenario has two drawbacks: (i) all activities are performed by the secondary agent, the primary agent has less information about the maintenance objective, cost, etc.; (ii) the quality control is difficult to achieve due to fact that the primary agent does not control this process directly.
- Scenario *b* separates the issue/problem identification and resolution process. Issue identification and issue resolution are clearly correlated. This separation might increase unnecessary maintenance efforts.
- Scenario *c* separates the issue/problem resolution process through updating process. Same as Scenario *b*, this separation might increase unnecessary maintenance efforts.
- Scenario *d* is the worst choice. The regression testing should always be performed locally by the primary agent. An offshore testing without the participation of the primary agent could be a nightmare for product quality.
- Although Scenario *e* does not take the advantage of the benefits of global software development, it could avoid the risks brought by global development.

Summarizing the discussions, for continuous evolution process, we propose to use fully localized process. The fundamental idea is that the gains of globalization could be much smaller than the risks of global process with respect to small maintenance/evolution activities.

3.3 Reengineering evolution process

As shown in Fig. 1, reengineering process is more similar to development process than to continuous evolution process. Figure 7 shows the different scenarios of reengineering a software product globally.

1. Scenario *a* could take the advantage of the benefits of globalization. However, the primary agent has no information about the cost of the project, because he/she gives full control of the project to the secondary agent.

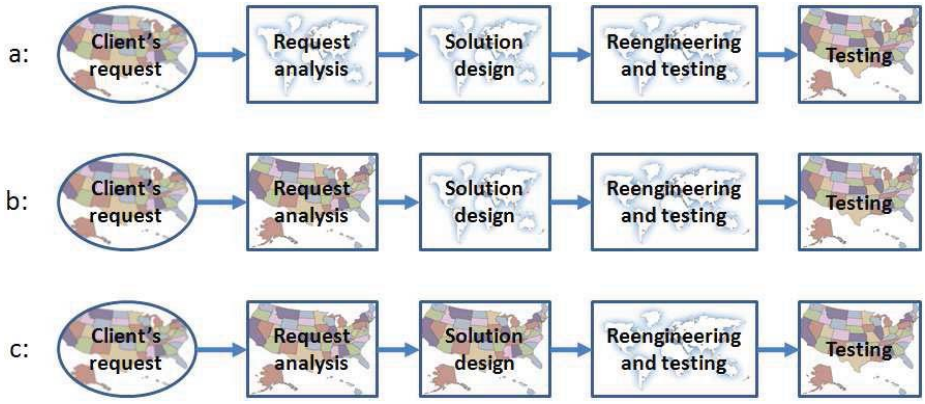

Fig. 7. Different scenarios of maintenance: reengineering process

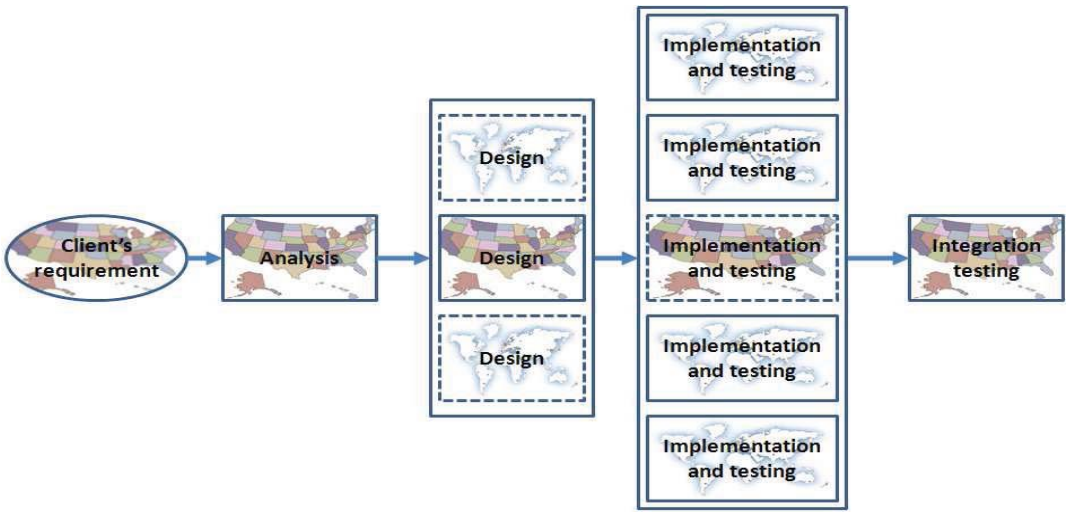

Fig. 8. Revised partially globalized software development process

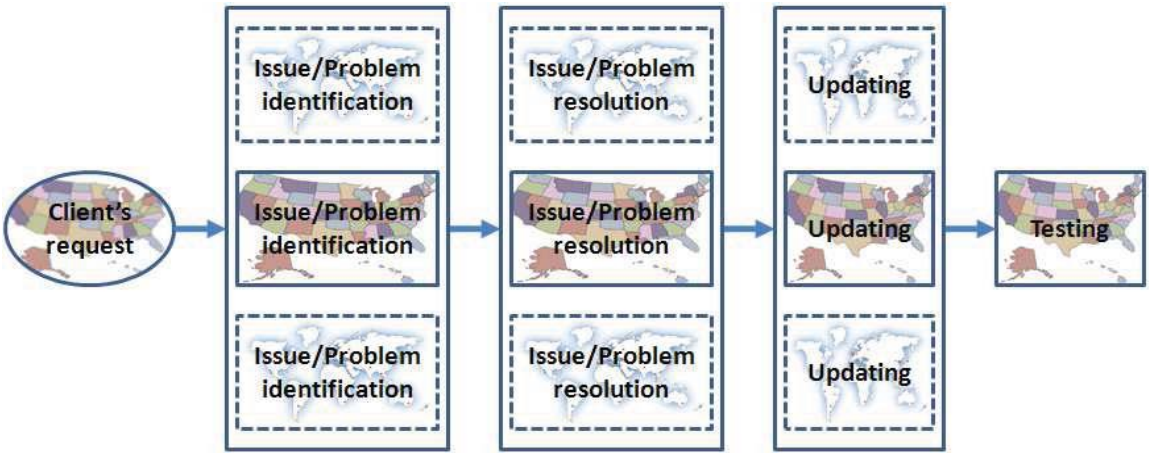

Fig. 9. Revised partially globalized maintenance: continuous evolution process

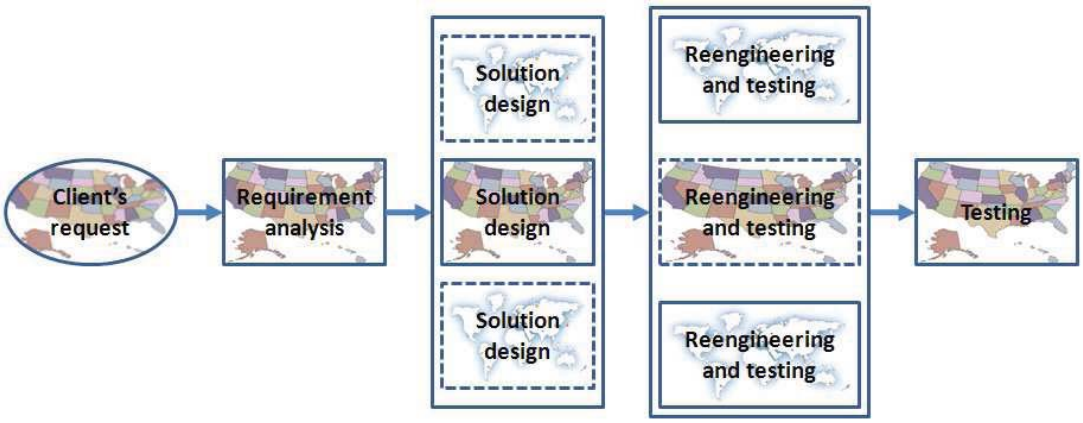

Fig. 10. Revised partially globalized maintenance: reengineering process

2. Scenario *b* separates the analysis and design. The primary agent could have more information about the project and its cost. However, the design activity is performed globally, which makes it difficult for the primary agent to control.

3. Scenario *c* is similar to the partially globalized software development process shown in Figure 5. Because the reengineering and unit testing process can be standardized, it makes the primary agent easy to monitor the implementation quality. Paasivaara and Lassenius [35] claim that iterative and incremental development process seems to be viable approach in global software development.

Summarizing the previous discussions, we propose to use scenario *c* in Fig. 7 as the global reengineering process.

3.4 Discussions

The proposed global software development, continuous evolution, and reengineering processes could reduce various risks in fully global software process. Some of these benefits are discussed below.

- For a long-lived software product, documentation is important for continuous maintenance, evolution, and restructuring. However, a major issue in global software development is the difficulty of enforcing documentation standard. In our proposed partially-global software processes, this risk could be largely reduced, because we only promote the globalization of implementation phase, which only involves the documentation of source code. Nowadays, the integrated development environment supports the automatic documentation, such as Javadoc [34] and VSdocman [35], which make the standardization easier to enforce.

- Another threat to the primary agent is that global software process (outsourcing) could result in the transferring of expertise to secondary agent companies, which eliminates the first agent's unique ability to develop competitive products. In other words, globalization might be beneficial in the short term, but it may be harmful in the long term [36]. Our proposed partially-global software process can reduce this risk through only outsourcing regular software components while keeping the development of key components, which involve proprietary data and algorithms, locally by the primary agent.

- Fully globalized software process might cause the primary agent's loss of control over future programs [19], if the major decisions about the software product are completely made by the secondary agent. Our partially globalized process can reduce this risk by allowing the primary agent to maintain the control of the management and design of the product.

4 NOT UNIQUE SOLUTIONS

In Section 3, we have proposed partially globalized processes for software development, continuous evolution, and restructuring. We restate here that the proposals are not unique solutions to global software development. In practice, the risks and benefits should be balanced in applying these processes. It is absolutely acceptable to alter these processes. For example, if the design could be componentized and the quality could be controlled, it makes sense to outsource these activities. Figure 8 through Fig. 10 illustrate flexible versions of partially global software processes, in which a dashed rectangle represents an optional composition. For example, in Fig. 8, a localized implementation and testing of a component is optional; it is only compulsory for key components or core assets.

5 CONCLUSIONS

To be successful in the global market, a company should manage the risks of global software development, and use the positive aspects as input to shape the development process in detail and the culture in general [18]. This paper observed and discussed the risks/benefits of global software development. We proposed partially globalized processes for software development, continuous evolution, and reengineering. These processes can be followed to balance the risks and benefits of global software development. Our principles of partially globalized software process are summarized below.

- We encourage outsourcing of implementation and discourage outsourcing of design;
- We encourage outsourcing of software components and discourage outsourcing of the entire product;
- We encourage global software development and discourage global software contract;
- We encourage long-term employment relation and discourage short-term contract;
- We encourage standardization of global software development process and discourage using different standards;
- We encourage maintaining control and visibility to the whole project and discourage giving full control to the secondary agent.

ACKNOWLEDGEMENT

We would like to thank Editor-in-Chief and referees for their valuable comments to improve the quality of this paper. We would also like to thank Dr. Ceylan Ertung of Academic Writing and Advisory Center (AWAC) – Atilim University for nicely editing the manuscript.

REFERENCES

[1] P. Abrahamsson, J. Warsta, M. T. Siponen, and J. Ronkainen, "New directions on agile methods: a comparative analysis," in *Proceedings of the 25th International Conference on Software Engineering*, pp. 244–254, 2003.

[2] A. Barcus and G. Montibeller, "Supporting the allocation of software development work in distributed teams with multi-criteria decision analysis," *Omega*, vol. 36, pp. 464–475, 2008.

[3] J. D. Herbsleb and D. Moitra, "Guest editors' introduction: global software development," *IEEE Software*, vol. 18, no. 2, pp. 16–20, 2001.

[4] M. Bittner, "Global product development seen as a boon for product lifecycle management vendors," *Technology Evaluation*, 2005.

[5] R. Prikladnicki, J. L. N. Audy, and R. Evaristo, "An empirical study on global software development: offshore insourcing of IT projects," in *Proceedings of the 3rd International Workshop on Global Software Development*, (Edinburgh, Scotland), pp. 53–58, May 2004.

[6] B. Boehm and R. Turner, *Balancing Agility and Discipline – A Guide for the Perplexed*. Addison-Wesley Professional, 2004.

[7] K. Braithwaite and T. Joyce, "XP expanded: distributed extreme programming," in *Proceedings of the 6th International Conference on eXtreme programming and agile processes in Software Engineering*, pp. 180–188, 2005.

[8] R. Prikladnicki, J. L. N. Audy, and R. Evaristo, "A reference model for global software development: Findings from a case study," in *Proceedings of the 2006 IEEE International Conference on Global Software Engineering*, 2006.

[9] S. Cherry and P. N. Robillard, "Communication problems in global software development: spotlight on a new field of investigation," in *Proceedings of the 3rd International Workshop on Global Software Development*, (Edinburgh, Scotland), pp. 48–52, May 2004.

[10] D. Damian, "Global software development: growing opportunities, ongoing challenges," *Software Process: Improvement and Practice*, vol. 8, no. 4, pp. 179–182, 2003.

[11] J. D. Herbsleb and A. Mockus, "An empirical study of speed and communication in globally-distributed software development," *IEEE Transactions on Software Engineering*, vol. 29, no. 6, pp. 481–494, 2003.

[12] F. Lanubile, D. Damian, and H. L. Oppenheimer, "Global software development: technical, organizational, and social challenges," *ACM SIGSOFT Software Engineering Notes*, vol. 28, no. 6, pp. 2–2, 2003.

[13] K. R. Al-asmari and L. Yu, "Experiences in distributed software development with Wiki," in *Proceedings of 2006 International Conference on Software Engineering Research and Practice*, (Las Vegas, Nevada), pp. 389–393, June 2006.

[14] K. R. Al-asmari, R. P. Batzinger, and L. Yu, "Experience distributed and centralized software development in ipdns project," in *Proceedings of 2007 International Conference on Software Engineering Research and Practice*, (Las Vegas, Nevada), pp. 46–51, June 2007.

[15] E. Carmel and R. Agarwal, "Tactical approaches for alleviating distance in global software development," *IEEE Software*, vol. 18, no. 2, pp. 22–29, 2001.

[16] P. Karecki, "Managing global software development," *CSC World*, pp. 18–21, July–September 2007.

[17] J. Lipnack and J. Stamps, *Virtual Teams: Reaching Across Space, Time, and Organizations with Technology*. John Wiley & Sons, 2000.

[18] C. Ebert and P. D. Neve, "Surviving global software development," *IEEE Software*, pp. 62–69, March–April 2001.

[19] B. A. Aubert, S. Dussault, M. Patry, and S. Rivard, "Managing the risk of IT outsourcing," in *Proceedings of the 32nd Annual Hawaii International Conference on System Sciences*, vol. 7, 1999.

[20] R. Prikladnicki and M. H. Yamaguti, "Risk management in global software development," in *Proceedings of the 3rd International Workshop on Global Software Development*, (Edinburgh, Scotland), pp. 18–20, 2004.

[21] J. Ralyte, X. Lamielle, N. Arni-Bloch, and M. Leonard, "A framework for supporting management in distributed information systems development," in *Proceedings of the 2008 IEEE Conference on Research Challenges in Information Science*, 2008.

[22] D. Mishra and A. Mishra, "Distributed information system development: review of some management issues," *Lecture Notes in Computer Science*, vol. 5872, pp. 282–291, 2009.

[23] D. Mishra and A.Mishra, "A review of non-technical issues in global software development," *International Journal of Computer Applications in Technology*, 2010. In press.

[24] H. Mungenast, "How viruses get on digital photo frames." www.digital-photo-framemarket.info/articles/virus_on_digital_photo_frame_china_a_security_threat.htm, accessed on the 12th January, 2010.

[25] R. Sangwan, C. Neill, M. Bass, and Z. Houda, "Integrating a software architecture-centric method into object-oriented analysis and design," *The Journal of Systems and Software*, vol. 81, pp. 727–746, 2008.

[26] L. Pilatti and J. Audy, "Global software development offshore insourcing organization characteristics: Lesson learned from a case study," in *Proceedings of the 2006 IEEE International Conference on Global Software Engineering*, 2006.

[27] P. Kulik, "A practical approach to software metrics," *IT Professional*, vol. 2, no. 1, pp. 38–42, 2000.

[28] M. Cataldo, M. Bass, J. Herbsleb, and L. Bass, "On coordination mechanisms in global software development," in *Proceedings of the 2007 IEEE International Conference on Global Software Engineering*, 2007.

[29] R. Ramnath, "Global software development for the enterprise," in *Proceedings of the 30th Annual International Computer Software and Applications Conference*, 2006.

[30] M. Bass and D. Paulish, "Global software development process research at siemens," in *Proceeedings of the 3rd International Workshop on Global Software Development*, (Edinburgh, Scotland), May 2004.

[31] Z. Yan, "Efficient maintenance support in offshore software development: a case study on a global ecommerce project," in *Proceeedings of the 3rd International Workshop on Global Software Development*, (Edinburgh, Scotland), May 2004.

[32] C. J. Meadows, "Globalizing software development," *Journal of Global Information Management*, vol. 4, no. 1, pp. 5–14, 1996.

[33] J. Ralyte, X. Lamielle, N. Arni-Bloch, and M. Leonard, "Distributed information system development: A framework for understanding and managing," *International Journal of Computer Science and Applications*, vol. 5, no. 3b, pp. 1–24, 2008.

[34] http://java.sun.com/j2se/javadoc/.

[35] http://www.helixoft.com/vsdocman/overview.html.

[36] S. McConnell, *Rapid Development: Taming Wild Software Schedules*. Redmond, WA: Microsoft Press, 1996.

Liguo Yu received the Ph.D. degree in Computer Science from Vanderbilt University. He is an assistant professor of Computer and Information Sciences Department at Indiana University South Bend. Before joining IUSB, he was a visiting assistant professor at Tennessee Technological University. His research concentrates on software coupling, software maintenance, software reuse, software testing, software management, and open-source software development.

Alok Mishra is an Associate Professor of Computer and Software Engineering at Atilim University, Ankara, Turkey. His areas of interest and research are software engineering, information system, information and knowledge management and object oriented analysis and design. He has published articles, book chapters and book-reviews related to software engineering and information system in refereed journals, books and conferences. He has received excellence in online education award by U21Global Singapore. He had also served as chief examiner computer science of the International Baccalaureate (IB) organisation. He is recipient of various scholarships including national merit scholarship and department of information technology scholarship of Government of India.

AUTHORS' ADDRESSES

Asst. Prof. Liguo Yu, Ph.D.
Computer Science and Informatics,
Indiana University South Bend,
South Bend, IN, USA,
email: ligyu@iusb.edu

Assoc. Prof. Alok Mishra, Ph.D.
Department of Computer and Software Engineering,
Atilim University,
Incek, 06836, Ankara, Turkey,
email: alok@atilim.edu.tr

Received: 2009-08-25
Accepted: 2010-01-06