Root Causes and Reduction Techniques for Rework in Global Software Development

Ritu Jain
Professor, Saraswati College of Engineering
Navi Mumbai, Maharashtra, India

Ugrasen Suman
Professor, SCSIT, Devi Ahilya University
Indore, MP, India

ABSTRACT
Global software development (GSD) is a trend adopted by numerous software companies to reduce development cost and time. However, potential cost and time savings are seldom realized due to unprecedented challenges in GSD environment. These challenges often hamper software development and project management activities. Moreover, they led to confusions, misinterpretations and conflicting perceptions regarding the software to be built, which in turn induce excessive rework. This rework significantly wastes lots of time as well as resources and may doom projects to fail. Rework in GSD setting can be reduced, if root causes of rework can be identified and thus avoided for GSD. However, there is paucity of research on this aspect. Thus, in this paper, root causes of rework in GSD are investigated. Techniques to reduce rework for GSD are also suggested. A metric for verifying the understanding score of distributed teams is also proposed. This study would aware practitioners and researchers about the causes of rework and its reduction techniques for GSD setting.

General Terms
Software Engineering, Global Software development, Distributed Software Development

Keywords
Global Software development, Rework, Causes, Reduction Techniques

1. INTRODUCTION
GSD is a software development approach in which practitioners residing in different countries work as a team to develop a software. Most of the software companies are adapting this approach to realize cost and time benefits associated with it. However, these benefits are seldom achieved due to geographical, temporal, socio-cultural, linguistic and organizational distances. These distances are collectively termed as GSD distances [1, 2]. Due to these distances, GSD projects often suffer from communication, coordination and collaboration challenges as depicted in Table1.

These communication, coordination, and collaboration challenges often led to misunderstandings, confusions, inconsistent perceptions, and defects; which in turn induce excessive rework. This rework significantly wastes lots of time, resources and thus, increases risk of project failure [3, 4]. A substantial amount of time and resources can be saved, if practitioners can reduce the rework incurred during GSD. Thus, in order to reduce rework in GSD projects, practitioners need to be aware of the root causes of rework in projects. Also, they should be well verse with the techniques which can be used for reducing rework. Thus, in this paper, root causes of reworkin GSD have been identified. Rework reduction techniques are also suggested. It would subsequently aid in improving the effectiveness of global software development process [8].

Table 1. Challenges associated with GSD Distances

Distance	Challenges
Geographical	Restricts formal as well as informal communication, project awareness, coordination and knowledge management [1].
Temporal	Increases response time, decreases synchronous communication, hampers effective coordination and collaboration [2]
Socio-cultural	Different frame of references, incompatible work ethics, inadequate cohesiveness, distrust and frustration among members belonging to different countries [3].
Linguistic	Difference in native language and linguistic accent limits communication and increases the probability of misunderstanding [3].
Organizational	Mismatch in processes, tools and work culture [1].

The paper is organized as follows. Section 2 presents a brief overview of rework in global software development and summarizes the related work. Section 3 presents the root causes of rework in GSD. Section 4 discusses rework reduction techniques. Finally, section 5 concludes the paper.

2. BACKGROUND AND RELATED WORK
All In software development setting, reimplementing or modifying a previously completed work is considered as rework. In software development, eliminating or preventing rework completely is not possible, but excessive rework signifies anomalies in development process or project management techniques [5]. However, excessively low rework

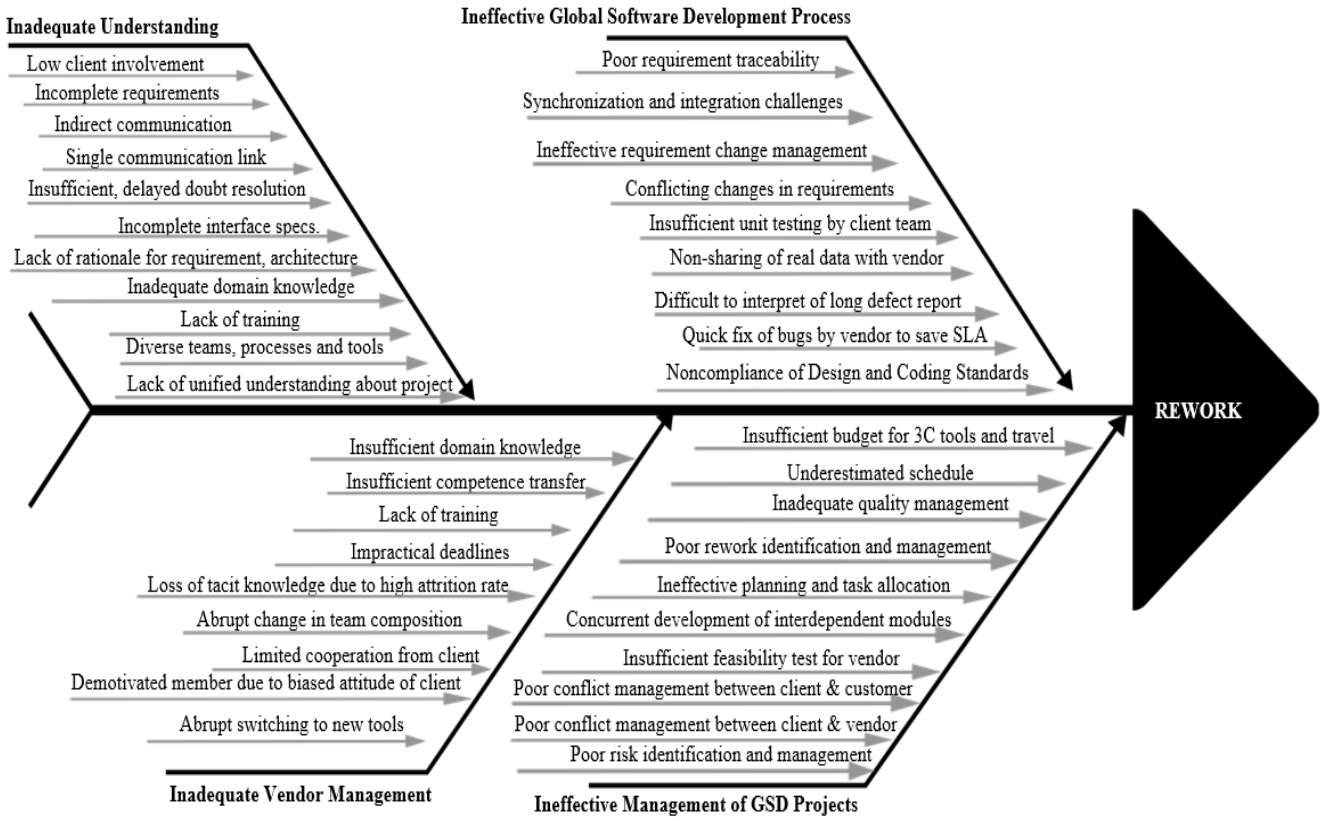

Fig 1: Root Causes of Rework in GSD

could be due to insufficient quality assurance or defect detection activities [5]. Excessive rework increases the cost of development, deteriorates product quality and demotivates team members, which in turn increases the risk of project failure [6]. Boehm et al has advocated avoidance of rework as one of the major approaches for enhancing productivity, decreasing development time and effort of a software project [8].

It has also been reported that approximately 40-50% of software development effort is wasted in rework. However, little research has been performed on investigating the root causes of rework in GSD [9]. A software development process can be effectively improved, if rework can be reduced or avoided [8]. Thus, strategies to reduce rework in GSD setting are also explored.

Few researchers have investigated rework in software development. Ramdoo et al. has identified root causes of rework in software development. According to their study, ambiguous requirements, misunderstanding among stakeholders, insufficient testing, inadequate maintenance of history and versioning are root causes of rework in software development [6]. However, they have not explicitly investigated rework causes as well as solution strategies for GSD projects.

Gopal et al. have investigated the impact of key process areas (requirements engineering, training, project planning, product engineering, software configuration management, peer reviews, and defect prevention) on project performance measured in terms of effort, elapsed time, and software rework in GSD projects. They found that requirement instability, rework stage, prior experience, and quality processes strongly affect rework [10]. However, the study basically investigated performance of projects measured in terms of effort, elapsed time and software rework and have not dedicatedly investigated causes of rework specifically in GSD setting.

Alahyari et al. has performed exploratory research to identify different types of wastes that are confronted in agile/lean software development organizations. They have discerned that rework is one of the software development wastes that need to be eliminated. According to this study, poorly specified requirements and ineffective architectural design and lack of common design pattern are considered as major sources of rework [11]. However, the study investigated all the sources of waste in agile/lean organizations and have not pinpointedly explored causes of rework in GSD. In GSD, additional rework is generated due to GSD distances. However, none of the research study has explicitly investigated rework for GSD setting. Thus, this paper attempts to explore this aspect for GSD. This study would aware practitioners about the probable causes of rework in GSD in order to avoid them. It would also draw attention of researchers to further investigate this software waste to improve the effectiveness of GSD process.

3. ROOT CAUSES OF REWORK IN GSD

Rework can hamper effective software development and project management [8]. To reduce the software development rework, underlying causes of rework need to be investigated. Thus, in this research work, root causes of rework in GSD setting have been explored as illustrated in Fig. 1. Major causes of rework in GSD are inadequate understanding of remote team members about project, ineffective global software development process, inadequate vendor management and ineffective GSD project management.

3.1 Inadequate Understanding

In GSD projects, low involvement of remotely located client, incomplete or poorly specified requirements and indirect communication between offshore developers and onsite

customers usually result into misinterpreted requirements which often lead to confusion and rework [1, 12]. Single communication link between onshore and offshore teams sometimes causes communication bottleneck [1]. Geographical, temporal and socio-cultural distances often cause insufficient or delayed resolution of doubts of offshore members [1]. Incomplete interface specifications also induce misinterpretations about other stories being implemented remotely [1]. Offshore developers have inadequate understanding about requirements, architecture and their rationale. In addition to this, inadequate domain knowledge often leads to conflicting modules [13]. Diverse teams, tools, processes, terminologies, different communication styles and lack of training often hamper understanding of remote developers. Lack of common understanding about project's vision, process policies, coding standards, design standards as well as cultural and organizational unawareness may also induce rework [13, 14]. Unawareness in offshore team about expected quality, process, and schedule adherence can further induce misunderstandings and rework [13]. Due to these incompatibilities, remote team members lack unified understanding about project.

3.2 Ineffective GSD Process

There is no silver bullet for GSD projects due to its varied characteristics. Different tools, processes and teams often restricts requirement traceability and could lead to synchronization and integration problems [3]. Requirement change management is difficult aspect of software development. Thus, in GSD when requirements keep on changing, remote members sometimes have to work on the basis of assumptions as information about changes in requirements is often not communicated timely and effectively to distant team members [1]. Also, conflicting changes in requirements are difficult to manage when team is dispersed [1]. Insufficient unit testing performed by client developers, before transferring code to vendor for testing and unavailability of real-life data at vendor site increases effort of testing [1]. Remote developers usually are unable to interpret long and unreadable defect report [1]. Test engineers belonging to different organizations sometimes quick fix the bugs, in order to save service level agreement (SLA) [1]. Due to lack of training and awareness, offshore developers do not comply to the decided design and coding standards which can induce misinterpretation [13].

3.3 Inadequate Vendor Management

Vendor team have insufficient domain knowledge due to lack of communication and cooperation. Insufficient competence transfer, process, culture, and language training for vendor team can induce rework [1]. Impractical deadlines force offshore members to finish work without focusing on quality [12]. It results in burnt out of offshore team members, high attrition rate of offshore members, reduced productivity and loss of tacit knowledge [1, 13]. Abrupt changes in team composition (shrinking and expansion) during project execution also induces rework [8]. Vendor team sometimes get demotivated due to biased attitude of client and inadequate senior management support [13]. Unfamiliarity with client's tools and switching to new tools and technology during project life cycle may create confusions and misinterpretation at vendor site [14].

3.4 Ineffective GSD Project Management

GSD projects encounter several challenges due to hampered communication, coordination and collaboration. Insufficient budget for collaboration establishment factors, such as travel expenses, 3C tools. Underestimated schedule due to inconsideration of GSD distances and inadequate time, resource allocation for quality management processes may force the team to finish work without focusing on quality Additional rework due to not allocating time for rework during schedule planning and management [17]. Inadequate planning and ineffective task allocation can lead to incompatible components [13]. Dependent or interdependent activities if accomplished concurrently at different locations as well as high dispersion of team members working on a single module can induce rework [15]. Insufficient feasibility study for GSD appropriateness, technically incompetent developers or inexperienced/ incompetent project manager can induce rework [6,13]. Inadequate conflict management between client and vendor and inadequate conflict management between customer and developers may create confusions [13]. Poor risk identification and management can also induce rework in GSD [16].

4. GSD REWORK REDUCTION TECHNIQUES

Rework is a major risk encountered during execution of GSD projects. It can be reduced with the help of the proposed techniques:

4.1 Architecture and Dependencies Need to be Effectively Managed

It is difficult to handle dependencies between user stories being developed at different locations. Thus, dependencies between user stories need to be identified as soon as possible to avoid rework. Stories having high dependencies need to be developed at same location and same iteration. Any change in dependencies need to be monitored continuously [14]. Software architecture need to be planned and designed cautiously, because it would influence quality, communication, and coordination requirements of distributed team; and can lead to high rework, if not handled cautiously. Architect and offshore project manager can continuously monitor the compliance of defined architecture and process used in development [18].

4.2 Proactive Management of Rework

Some inevitable rework should be completed on regular basis. It would reduce generation of new rework. During project planning and estimation, ample time need to be allocated for rework activities to avoid excessive schedule pressure, slipped deadline, and further rework. Probable causes of rework need to be identified during project planning as well as iteration planning. These causes of rework should be monitored and mitigated during project execution. Weekly retrospective meetings can be organized to evaluate amount of rework which have been avoided and could have been avoided [19, 20].

4.3 Adaptive Agile Approach to Reduce Rework

In agile, higher priority is assigned to tasks having high customer value, which can lead to negligence of quality preserving tasks (managing backlogs, architecting, updating documentation, and refactoring) having zero customer value. Thus, balance of work needs to be maintained between quality oriented and high customer valued tasks which could maintain quality and reduce rework [21]. Agile practices, such as test-driven development, pair programming, refactoring, software craftsmanship, iteration management, and mindfulness about technical debt help in avoiding rework [19, 21, 22]. Customer awareness as well as involvement need to be improved to

reduce rework. It would help the team to better understand their needs, reduce requirement volatility, ambiguous and incomplete requirements [7].

4.4 Validation and Verification Activities

Greater emphasis is required on verification and validation activities in GSD projects. Rework can be reduced by emphasizing quality assurance activities, such as inspections, technical reviews, and walkthroughs in early phases of software development. Reviews which can verify requirements, design, architecture, code, and test scenario must be performed throughout software development. Documentation stored in repositories also need to be reviewed and updated periodically to avoid ambiguities. Root-cause analysis of the errors detected during these activities need to be performed and subsequently preventive actions need to be taken [14].

4.5 Disciplined Software Engineering Practices

Disciplined as well as mature software development process, robust requirement engineering mechanism, systematic software configuration management, and mindful interface design aid in avoiding rework. Adequate knowledge management, competent, trained, and experienced project manager as well as practitioners would reduce rework in GSD [5, 14].

4.6 Project Management Practices to Reduce Rework

Team having members with cross functional skills at each site would aid in reducing rework. High turnover rate swipe off tacit knowledge from the project thus, increases rework. Pair programming, updated documentation, and cross training are some of the ways to preserve tacit knowledge in teams with high attrition rate [14, 20]. Vendor's inexperience or incompetence regarding GSD projects can be pretested before commencement of project by conducting economic, technical, and behavioral feasibility. Freedom of expression, conflict management between onshore, offshore team and customer, and appropriate task allocation can reduce rework. Proper scope, cost, and time estimation as well as resource allocation need to be performed with consideration to GSD distances [13].

4.7 Team Understanding Score Metric for Improving Understanding of Remote Team Members

One of the major causes of rework in GSD is reduced understanding and awareness about requirements, architecture, project, culture, tools, and technology in remote team. In this research work, a metric that indicates understanding score of offshore members to verify sufficient understanding about these aspects has been proposed. The formulae for the proposed metric is as follows:

Team Understanding Score = Ud + Uo + Ut + Uc (1)

where,

Ud: understanding about domain, requirement, architecture, and their rationale.

Uo: understanding about organizational processes, policies, vision, and work culture of onshore site.

Ut: developer's expertise about usage of tools and technology.

Uc: understanding about cultural values, language of remote counterparts, and temporal distance.

Team members can be asked to provide score for values of Ud, Uo, Ut, Uc individually in the range of 0 to 5, according to their understanding level. A team having individual Ud, Uo, Ut, Uc score greater than 3 and average score greater than 12 has good understanding of these aspects and possesses capability to work in GSD environment.

Work overlapping hours would increase synchronous communication, facilitate timely feedback, and enable early doubt resolution. Previous working relationship, collocated design phase, regular meetings, and globally accessible repositories would improve domain knowledge, rationale behind decisions, and team's understanding about the project and organization. Descriptive user story comprises clear interface specifications which would reduce the problem of insufficiently detailed and misunderstood requirements. Short cultural and language training, frequent visits, relocation of offshore representative at onshore location, and tools related to communication, coordination and collaboration would increase cultural awareness, improve team cohesiveness, and reduce linguistic distance [13, 14].

5. CONCLUSION

Global software development is an approach which is being adopted by numerous software companies across the globe whereas, rework is a software development waste which impedes the effectiveness of software development. In this paper, root causes of rework in GSD have been investigated. Major causes of rework in GSD are inadequate understanding of remote team members about project, ineffective software development process, inadequate vendor management and ineffective GSD project management. Several techniques to reduce rework have been discussed. A team understanding metric to monitor and improve the unified understanding about the project has been proposed. This study would aware practitioners about the causes of rework in GSD. It would also draw attention of researchers to further investigate this software waste to improve the effectiveness of GSD process.

6. REFERENCES

[1] Jain, R., Suman, U. 2015. A Systematic Literature Review on Global Software Development Life Cycle. ACM SIGSOFT Softw. Eng. Notes 40(2), (2015), 1-14.

[2] Espinosa, J.A., Carmel, E. 2003. The Impact of Time Separation on Coordination in Global Software Teams: A Conceptual Foundation. Softw. Process Improv. Pract. 8(4), (2003), 249-266.

[3] Babar, M. A., Zahedi, M.2012. Global Software Development: A Review of the State-Of The-Art (2007-2011). IT University Technical Report Series. IT University of Copenhagen.

[4] Herbsleb, J.D., Mockus, A., Finholt, T.A., Grinter, R.E. 2001. An Empirical Study of Global Software Development: Distance and Speed. In: Proceedings of the 23rd International Conference on Software Engineering (ICSE '01), pp. 81-90. IEEE Computer Society, Washington, DC, USA.

[5] Fairley, R.E., Willshire, M.J. 2005. Iterative Rework: The Good, the Bad, and the Ugly. Ieee Computer. 38(9), (Sep. 2005), 34-41.

[6] Ramdoo, V., Huzooree, G. 2015. Strategies to Reduce Rework in Software Development on an Organisation in Mauritius. International Journal of Software Engineering & Applications. 6(5), (2015), 9-20.

[7] Schwalbe, K. 2015. Information Technology Project

Management. 8th edn. Cengage Learning.

[8] Boehm, B.W., Papaccio, P.N.1988. Understanding and Controlling Software Costs. IEEE Trans. Softw. Eng. 14(10), (1988), 1462-1477.

[9] Boehm, B., Basili, V.R. 2001. Software Defect Reduction Top 10 List. Ieee Software. 34(1), (2001), 135-137.

[10] Gopal, A., Mukhopadhyay T., Krishnan, M.S. 2002. The role of software processes and communication in offshore software development. Commun. ACM. 45(4), (2002), 193-200.

[11] Alahyari, H., Gorschek, T.,Svensson, R. B. 2019. An exploratory study of waste in software development organizations using agile or lean approaches: A multiple case study at 14 organizations. Information and Software Technology, 105, (2019), 78-94.

[12] Hoda, R., Murugesan, L. K.2016. Multi-level agile project management challenges: A self-organizing team perspective. J. Syst. Softw. 117, (2016), 245-257.

[13] Jain, R., Suman, U. 2018. A Project Management Framework for Global Software Development. ACM SIGSOFT Softw. Eng. Notes 43(1), (2018), 1-10.

[14] Jain, R., Suman, U. 2017. An Adaptive Agile Process Model for Global Software Development. International Journal on Computer Science and Engineering. 9(6), (2017), 436-445.

[15] Browning, T.R., Eppinger, S.D. 2002. Modeling the Impact of Process Architecture on Cost and Schedule Risk in Product Development. IEEE Trans. Eng. Manag. 49(4), (2002), 428-442.

[16] Verner, J.M., Brereton, O.P., Kitchenham, B.A., Turner, M., and Niazi, M. 2014. Risks and Risk Mitigation in Global Software Development: A Tertiary Study. *Inf. Softw. Technol.* 56 (1), (2014), 54-78.

[17] Coram, M., Bohner, S. 2005. The Impact of Agile Methods on Software Project Management. In: Proceedings of the 12th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, IEEE Computer Society, Washington, DC, USA (2005), 363-370.

[18] Nord, R.L., Ozkaya, I., and Sangwan, R.S. 2012. Making Architecture Visible to Improve Flow Management in Lean Software Development. IEEE Softw. 29(5), (2012), 33-39.

[19] Fairley, R.E., and Willshire, M.J. 2017. Better Now Than Later: Managing Technical Debt in Systems Development. Ieee Computer, 50(5), (2017), 80-87.

[20] Nelson, R.R. 2007. IT Project Management: Infamous Failures, Classic Mistakes, and Best Practices. MIS Quart Exec. 6(2), (2007), 67-78.

[21] Nord, R.L., Ozkaya, I., Kruchten, P., and Gonzalez-Rojas, M. 2012. In Search of a Metric for Managing Architectural Technical Debt. In Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture. WICSA-ECSA '12, IEEE Computer Society, Washington, DC, USA (2012), 91-100.

[22] Brown, N., Cai, Y., Guo, Y. et al. 2010. Managing technical debt in software-reliant systems. InProceedings of the FSE/SDP workshop on Future of software engineering research FoSER '10, ACM, NY, USA, Santa Fe, New Mexico, USA (2010), 47-52.