IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON SOFTWARE STANDARDS AND THEIR IMPACT IN REDUCING SOFTWARE FAILURES

Received October 30, 2017, accepted November 30, 2017, date of publication December 28, 2017, date of current version May 9, 2018.

Digital Object Identifier 10.1109/ACCESS.2017.2782843

Taxonomy of Factors Causing Integration Failure during Global Software Development

ATIQUE AHMAD ZAFAR[1], SHAHELA SAIF[1], MUZAFAR KHAN[1], JAVED IQBAL[1], ADNAN AKHUNZADA[1], ABDUL WADOOD[2], AHMAD AL-MOGREN[2], AND ATIF ALAMRI[2]

[1]Department of Computer Science, COMSATS Institute of Information Technology, Islamabad 45550, Pakistan
[2]Research Chair of Pervasive and Mobile Computing, College of Computer and Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia

Corresponding author: Atique Ahmad Zafar (atiquezafar@comsats.edu.pk)

This work was supported by King Saud University, through the Vice Deanship of Research Chairs.

ABSTRACT Controlling integration failure is one of the major challenges in global software development (GSD) that remains hidden during the development phase and surfaces during the system integration. The integration failures occur as a result of incompatibilities and integration complexities that subsequently lead to delays, extra cost, affect the overall quality, and can even throw the entire GSD project into chaos. A very good understanding of integration failures may help to overcome the integration challenges. The objective of this paper is to explore comprehensively the integration failure factors. This paper thoroughly reviews the available literature. Moreover, the authors have conducted an industrial survey to more closely explore the integration failure factors. This paper largely contributes by devising a detailed taxonomy of 40 integration failure factors. The classification allows to better understand the relationships between the various factors and helps in creating a holistic solution to deal with integration problems in the context of GSD.

INDEX TERMS Global software development, integration challenges, software integration, distributed development.

I. INTRODUCTION

Global Software Development (GSD) involves participation of different teams from multiple geographical locations. For last two decades, many organizations have moved their development activities to their offshore stations, but their offshore infrastructure is still not mature for it. One of the basic reasons behind that is the inability of remote teams to understand the working of the whole system as a single unit [1]. Project management team finds it very hard to manage large and complex GSD projects across the boundaries [2].

On the other hand, there is a lack of empirical studies which can provide solutions for different problems faced by the GSD management and team members [3]. The existing studies are specific in nature. The current research is also limited in terms of socio-technical aspects of these projects. According to [4], in GSD, project management and team members at remote locations follow non-standard or not agreed upon ways of implementing the project in terms of selected locations, collaborative approach, and work items. In addition to it, team members at various locations follow different development standards, processes and quality models which lead to the failure to provide the problem solution [4]–[6].

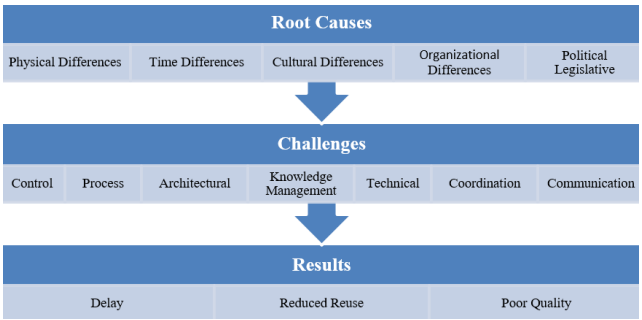

FIGURE 1. Multi-layer model to address GSD challenges [2].

According to a survey [7], 40% offshore projects resulted into failure. Several studies [2], [8]–[16] highlighted GSD challenges to address the project failure. Another study [2] provided a multi-layer model, as shown in Fig. 1, to understand GSD challenges effectively.

In addition to the above presented challenges, module integration in GSD projects is considered the most problem driven phase [5]. According to [4], more than 50% of the GSD projects face integration phase as the most problematic one. Moreover, the alarming aspect of integration problems is the late detection of the problems during GSD lifecycle that results in high cost to fix the problems [19]. Some studies [5], [18], [20], [21] emphasis on integration problems and the results demonstrate that integration problems generally follow a pattern when considering the socio-technical aspects.

In the existing literature, system integration in GSD has attained surprisingly little attention [4]. According to a study [2], a few studies has focused on the system integration. Moreover, the existing studies related to system integration are limited to specific project context or application and are unable to produce general solutions to cope with system integration issues [1].

This study contributes by exploring possible system integration failure factors experienced during system integration in GSD projects. The taxonomy of these factors is proposed based on the literature review and the industrial survey. The proposed taxonomy classifies the integration failure factors into eight classes based on the underlying causes which lead to the factors

The advantage of the proposed taxonomy is that it may help GSD project management teams to stay organized, keep track of processes and identify the problems earlier. Each factor has been assigned to a class based on the main underlying cause. The factors placed together in the same class can help project teams to identify similar failure factors related to the single cause. It simplifies the job to identify the integration failure factors and to devise viable and quick mitigation strategies.

II. RELATED WORK

Ammerlaan [4] described the pitfalls of system integration and testing practices in administrative and technical perspectives. The study highlighted system integration issues which include project delays, lower product quality and increase in project cost. This study also presented the integration strategies along with their characteristics. It was concluded that system integration as other software development phases also required a thorough risk analysis which might help to control system integration problems.

Goltel *et al.* [11] performed an experiment through a student project by maintaining global software development environment to identify system integration issues. The study argued that there was a need to plan concrete integration strategy at early stages of the project. This study recommended high level dos and don'ts for integration planning and practices.

On the basis of a case study, Herbsleb and Grinter [5] presented integration issues along with communication, coordination and unpredictable problems. According to the study, integration problems can be controlled by establishing an effective communication channel between cross site developers. It is also reported that the task of system integration becomes very difficult due to unrealistic assumptions, dependencies in overall development strategy, unclear assumptions about interface and informal specifications refinement in integration plan. Moreover, the selection of inconsistent technology and different processes across sites make the system integration highly problematic.

Ramaoorthy [1] performed the literature analysis to discover system integration issues. The reported studies are limited to specific applications which do not provide general solutions to resolve integration problems. It is also mentioned that integration issues can be avoided by facilitating integration activities in early stages of a project.

Wolf *et al.* [13] proposed a prediction model to predict integration build results by using communication structure measures. According to the study, communication among developers play a very important role in order to achieve quality results related to system integration. The study also highlights that it is very hard to determine the failure and success of integration build through individual measures. The results of this study are also limited to single project analysis.

Herbsleb *et al.* [45] analyzed nine different projects through interview to determine the challenges and benefits of distributed development. The teams which participated in the projects were dispersed in three different geographical locations. Communication and collaboration were identified as one of the major problems during development. Face to face meeting is highlighted as the important and fast means of communication during distributed development. Minimal interaction between cross sites causes an increased frustration between cross site developers and causes hurdles in system integration.

III. RESEARCH METHODOLOGY

The research methodology of this study mainly involves extensive literature review, industrial survey, and concept mapping. The following subsections provide the detailed description.

A. LITERATURE REVIEW

The literature review is conducted based on the guidelines available in [22]. The attempt is to find out all the possible evidences related to integration failure factors observed through either empirical or non-empirical research methods. The relevant literature is searched through the search string. The Boolean 'AND' is used for joining major terms and 'OR' is used for joining alternative terms. The resulting search string is as follows:

((global OR distributed OR multi-site OR remote) AND (software) AND (integration))

The five electronic databases are selected, as suggested by [23], to acquire quality material from authentic resources in software engineering domain. The electronic resources include IEEE explore, ACM Digital library, Science Direct, Springer Link, and Engineering Village (EiCompendex). The selection criteria include that the selected study must provide

the clear evidence of integration failure factor in GSD and every included study must be peer reviewed.

Data extraction strategy is concentrated on three points:

i The context of a factor
ii The observation source of a factor
iii The validation of a factor i.e. validated empirically or by expert opinion or with theoretical reason.

The content of factors should also be summarized and reported as it is originally stated in the study. In addition to it, the selection should not be based on the authors' understanding and assumptions. In total, 58 studies relevant to the integration topic are found. Fig. 2 shows the number of studies selected from various databases.

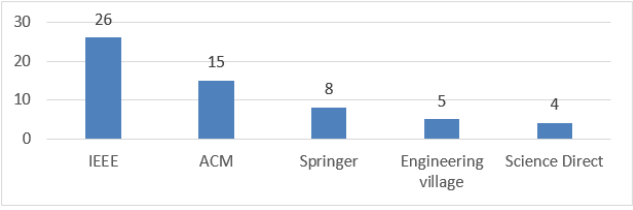

FIGURE 2. No. of studies selected from various databases.

According to Fig. 3, most of the studies (42) are based on empirical research and the remaining (16) are non-empirical studies.

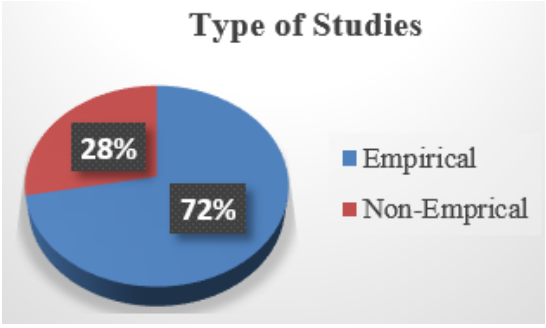

FIGURE 3. Types of selected studies in terms of empirical/non-empirical evidence.

Out of 58 selected studies, 52% are case studies, 23% are theoretical reports, 9% are interview-based, 8% are student projects, 6% are experience reports, and 2% are field studies. Fig. 4 shows the types of the selected studies.

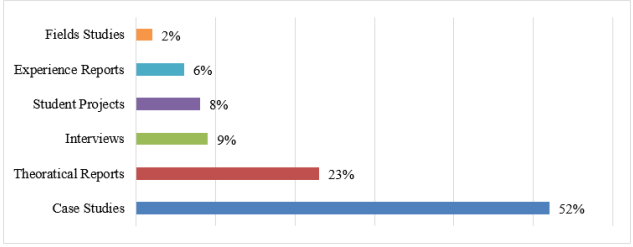

FIGURE 4. Types of the selected studies (in percentage).

TABLE 1. Profile of industrial experts.

Expert ID	Project Role	GSD Industrial Experience (in years)
A	Project manager	15
B		8
C		8
D		4
E		2
F		11
G	Team lead	10
H		6
I		5
J		1
K		5

In total, 96 integration failure factors have been found through the literature review.

B. INDUSTRIAL SURVEY

A total of 11 industrial experts having experience in GSD projects participated in this survey as shown in Table 1. Out of 11 participants, 6 were project managers and the remaining 5 were team leads. All participants had a Master degree in Computer Science with an average experience of seven years in GSD industry.

The survey was conducted through an open-ended questionnaire in which participants were asked to list the factors that cause failure during integration of functional modules in globally distributed software development projects. In total, 49 failure factors were identified through this survey.

C. FACTOR CONSOLIDATION PROCESS

In total, 145 (96 + 49) integration failure factors were identified through the literature review and industrial survey. The consolidated list of 40 unique integration failure factors was prepared based on the comparative data analysis [24] in two steps, as shown in Fig. 5. In the first step, the keywords or concepts of the failure factors were compared. If the multiple factors had the same keywords or concepts, they were selected for the consolidation. In the second step, the context of the factors was compared. The factors which contained the same context were consolidated. The reason behind two steps comparative analysis at keywords and context level was that sometime authors used different keywords for the similar factors e.g. ad-hoc communication and informal communication. In addition to this, sometimes authors used almost same keywords for different factors e.g. lack of informal communication due to different time zone and lack of informal communication due to language barrier.

D. CONCEPT MAPPING

The concept mapping technique was followed to define classes and associate each factor to the relevant class. Concept mapping is a technique that explores the relationships between concepts and presents them in a hierarchical or

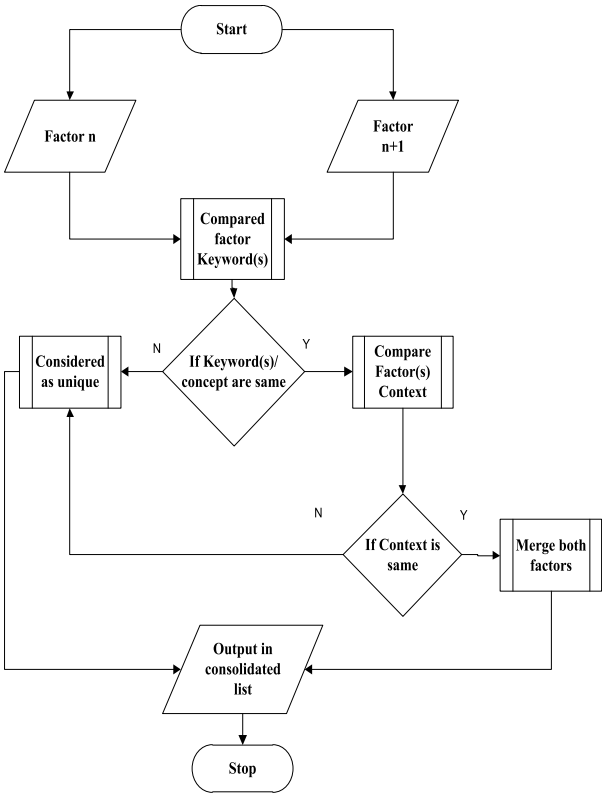

FIGURE 5. Factors consolidation process.

tree structure [52]. The map begins with a word or concept that is the primary question and the sub-levels are built by using connective terms such as ''leads-to'', ''results-from'', ''is-part-of'' etc.

The definition of classes was based on the key terms of the factors, and the definition of sub-levels (containing factors) was based on ''leads-to'' and ''results-from'' relationships. For example, Class 2 is ''inadequate process planning'' that *leads-to* ''big bang integration'' and/or ''lack of integration documents and manuals''. In total, eight classes were defined, and the factors were associated with the relevant classes after critically analyzing the context of factors. The classes and the associated factors are presented in the next section.

IV. THE PROPOSED TAXONOMY

All identified factors are not exclusive in terms of the concept areas that they cover. The factors are classified into eight primary classes based on the central focus of each factor. The classes are listed in Table 2. Fig. 6 represents the proposed taxonomy. The proposed classes and the relevant integration failure factors are presented in the following subsections.

A. CLASS C1: LACK OF COORDINATION BETWEEN REMOTE SITES

The success of any software process depends on the coordination between the team members from the highest order to the lowest. A study [5] suggests that a collocated software team spends an average of 75 minutes daily in informal communication. But in distributed teams, this communication frequency becomes remarkably low. This lack of coordination can take many forms such as uncooperative behavior or lack of awareness about activities on remote sites. Importance of frequent and fruitful communication cannot be ignored in globally distributed teams. A list of relevant factors is given in Table 3.

TABLE 2. The proposed classes.

Class ID	Class Name
C1	Lack of coordination between Remote Sites
C2	Inadequate process planning
C3	Inadequate process implementation
C4	Communication barriers
C5	Poor resource management
C6	Improper requirements change management
C7	Inadequate team expertise
C8	Improper tool selection

TABLE 3. Factors associated with lack of coordination between remote sites.

Factor ID	Integration Failure Factors
C1-F1	Long term lack of awareness activities at remote site
C1-F2	Uncooperative behaviour among remote team members
C1-F3	Lack of coordination between remote teams

1) FACTOR C1-F1: LONG TERM LACK OF AWARENESS ACTIVITIES IN REMOTE SITE

In general, the division of work among various sites is such that any component being developed at one site directly affects the component being developed at another. The continuous awareness regarding products, between sites becomes utmost important. Multiple teams must be aware of updates about the processes, changes in the products, collaborative tool integrations, information regarding potential users of the code, and the integration teams [25]–[28]. According to [27], communication problems turn into integration problems which lead to defected products.

2) FACTOR C1-F2: UNCOOPERATIVE BEHAVIOR AMONG REMOTE TEAM MEMBERS

According to [12], when two teams work on a dependent piece of software, one of the teams may sabotage the work of the other by producing a fix that may work only in a specific environment. This causes an integration failure and provokes the other team to rework. The study [32] claims that hiding crucial information results in inconsistencies during the project evolution and failure at the time of integration.

3) FACTORC1-F3: LACK OF COORDINATION BETWEEN REMOTE TEAMS

Ovaska *et al.* [29] suggests that integration problems are common when remote teams fail to share information about

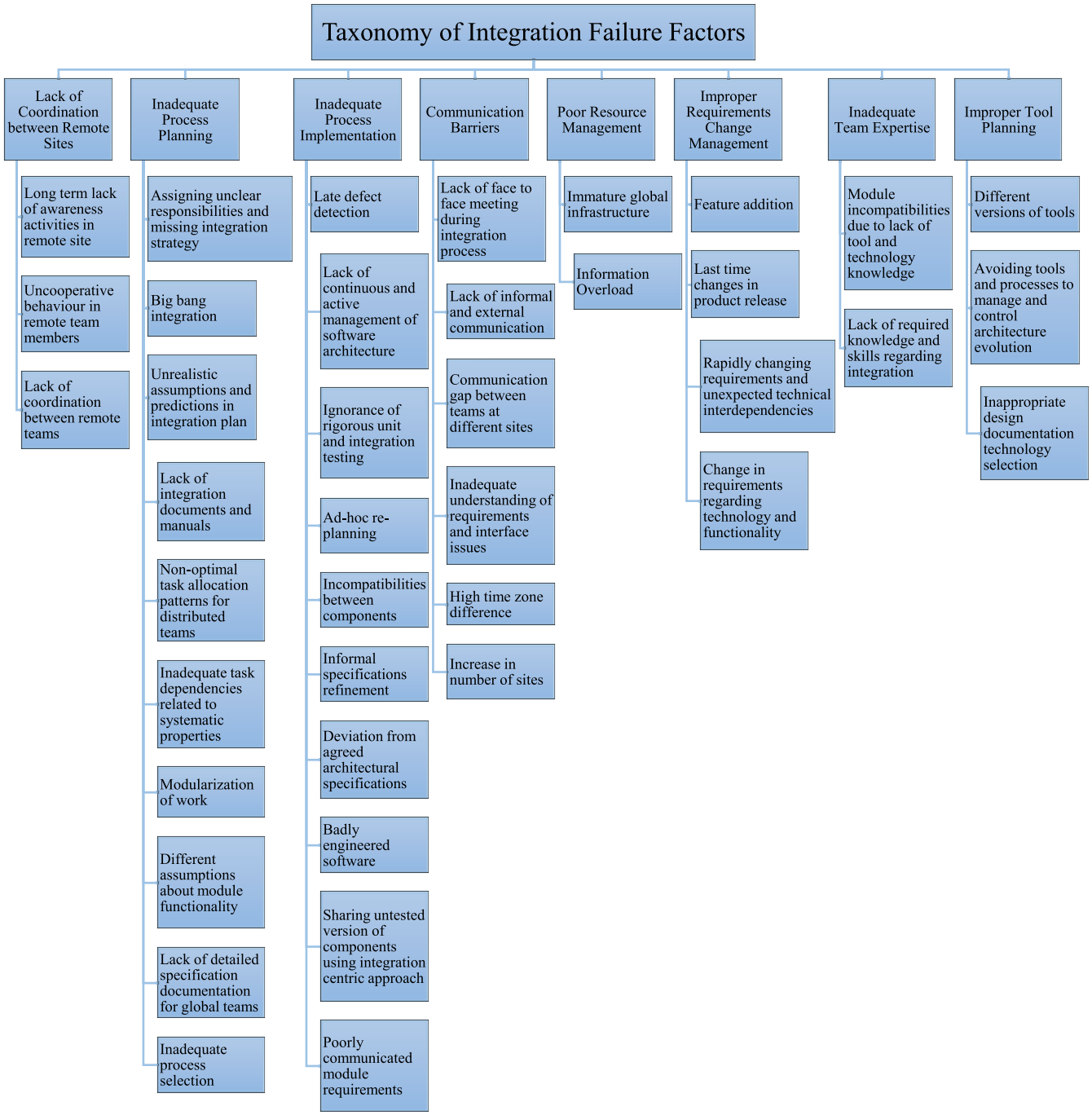

FIGURE 6. Taxonomy of integration failure factors.

the changes on time. This can lead to delayed product release. It is noted that coordination is required in three key areas: ''technical, process and scheduled events'' [15]. Any shortcoming in terms of coordination between teams can cause technical failures that become apparent at integration phase.

The study [30] highlights that global teams often suffer from ill-formed coordination mechanisms which are essential for facilitation of processes. This creates a social and psychological distance between the team members. Delays in release are frequently caused due to non-communicating teams that are working on the dependent modules. A team or at least one member should be assigned to look over the development status and confirmation to standards to enable seamless integration.

B. CLASS C2: INADEQUATE PROCESS PLANNING

The process planning involves the planning about requirements gathering, analysis techniques and the software

TABLE 4. Factors associated with inadequate process planning.

Factor ID	Integration Failure Factors
C2-F1	Assigning unclear responsibilities and missing integration strategy
C2-F2	Big bang integration
C2-F3	Unrealistic assumptions and predictions in integration plan
C2-F4	Lack of integration documents and manuals
C2-F5	Non-optimal task allocation patterns for distributed teams
C2-F6	Inadequate task dependencies related to systematic properties
C2-F7	Modularization of work
C2-F8	Different assumptions about module functionality
C2-F9	Lack of detailed specification documentation for global teams
C2-F10	Inadequate process selection

model selection. The planning also focuses on the integration methodologies, tools selection, and communication tools and technologies. Prikladnicki *et al.* [54] discuss the issues with GSD and highlight on a broad level that knowledge management and technical issues need to be dealt with for successful project completion. Inadequate process planning leads to improper management of these issues and ultimately causes the project failure. The factors associated with this class are given in Table 4.

1) FACTORC2-F1: ASSIGNING UNCLEAR RESPONSIBILITIES AND MISSING INTEGRATION STRATEGY

Quick fixes to patch the problems that arise at the time of integration cause new defects [31]. Sometimes the original developers may not be associated with the project till integration phase which forces the other developers to improvise. In such scenarios, it is helpful to have a plan or strategy for integration that was previously devised. Most often the plan that delegated responsibilities is not available causing serious integration issues [20].

A surprising cause of failure is the project managers' inability to provide ample time and resources to the planning and execution of integration. This is the result of the mindset that integration is of lesser significance than other software lifecycle phases [4].

2) FACTORC2-F2: BIG BANG INTEGRATION

Ramamoorthy [1] in his work has studied the effect of large number of teams on integration. The study confirms that highly interdependent or large number of teams are difficult to manage and coordinate. This is a perfect recipe for creating integration problems. Integration is usually done on multiple levels based on module usage such as subsystem and cluster. Without the plan, the integration on every level gives rise to numerous problems – the phenomenon termed as 'big-bang integration problem' [1].

3) FACTOR C2-F3: UNREALISTIC ASSUMPTIONS AND PREDICTIONS IN INTEGRATION PLAN

Herbsleb and Grinter [5] observed that the initial integration plan based on many unrealistic assumptions about requirements, schedules, and staff issues led to integration failure. Such integration plans could not be followed with the passage of time.

4) FACTORC2-F4: LACK OF INTEGRATION DOCUMENTS AND MANUALS

A thorough and correct process document sets the groundwork for an effective system development and integration, but it is often neglected in terms of maintenance after requirement or design changes. The off-sites are left unaware of the changes and thus are unable to produce valid integration strategy [4].

5) FACTORC2-F5: NON-OPTIMAL TASK ALLOCATION PATTERNS FOR DISTRIBUTED TEAMS

In GSD, there are often many known and unknown dependencies that exist between modules being developed at distributed sites. The allocation of tasks to inexperienced or incompetent person on ad-hoc basis can lead to mismanagement of resources and problems at time of integration [53].

6) FACTORC2-F6: INADEQUATE TASK DEPENDENCIES RELATED TO SYSTEMIC PROPERTIES

Systemic properties such as memory footprint and performance budget are inadequately addressed for their dependencies across components being built at remote sites. Not addressing the issue results in duplicated efforts and even conflicting solutions from different teams leading to integration problems [33].

7) FACTORC2-F7: MODULARIZATION OF WORK

The proper modularization of work can ease the distribution of work across many sites. If the remote teams are unaware of the dependencies between the modules or changes that may have been incorporated into requirements or design at remote sites then it leads to integration problems [34], [35]. MacGregor *et al.* [36] report integration problems that occurred due to inefficient distribution of architectural modules to teams distributed across technical and cultural boundaries.

8) FACTORC2-F8: DIFFERENT ASSUMPTIONS ABOUT MODULE FUNCTIONALITY

Designers at different sites may independently design the modules that may lead to problems. As the designers do not understand the complete software, therefore, the interpretation of requirements may be different at various sites that leads to incompatible modules or incorrect functionality [1], [37].

9) FACTORC2-F9: LACK OF DETAILED SPECIFICATION DOCUMENTATION FOR GLOBAL TEAMS

Globally distributed teams often have different native languages and use one language, such as English, for communication. According to expert I, speakers of different languages may interpret same terms differently which emphasizes the significance of a detailed documentation. Expert H has emphasized the importance of writing down the specifications to the utmost detail. He argues that for non-native speakers it is not possible to comprehend what the text might mean if all the details are not included.

10) FACTORC2-F10: INADEQUATE PROCESS SELECTION

If processes are selected without considering the system or organization, they may lead to integration failure even though they are being followed properly [26]. A common case, that is the distribution of tasks that require frequent informal communication between remote teams, would cause inefficient work progress in large global organizations [26]. According to expert J, another cause of failure is disagreement between teams on engineering or development methodology.

C. CLASS C3: INADEQUATE PROCESS IMPLEMENTATION

The processes are executed successfully when the required resources are readily available. Among these, the most difficult to manage is the human resource. It has been observed that sometimes the process plans are made but are not strictly followed leading to software defects and problems in later stages of the software. Table 5 shows the factors associated with inadequate process implementation.

TABLE 5. Factors associated with inadequate process implementation.

Factor ID	Integration Failure Factors
C3-F1	Late defect detection
C3-F2	Lack of continuous and active management of software architecture
C3-F3	Ignorance of rigorous unit and integration testing
C3-F4	Ad-hoc re-planning
C3-F5	Incompatibilities between components
C3-F6	Informal specifications refinement
C3-F7	Deviation from agreed architectural specifications
C3-F8	Badly engineered software
C3-F9	Sharing untested version of components using integration centric approach
C3-F10	Poorly communicated module requirements

1) FACTORC3-F1: LATE DEFECT DETECTION

Defects that may occur due to changes in requirements do not surface until the integration stage. Mostly defects are highlighted when the locally modified components are to be integrated with the components developed at other sites. Any changes done at this stage further delay the project and result in projects full of bugs which are extremely costly to fix [38].

2) FACTOR C3-F2: LACK OF CONTINUOUS AND ACTIVE MANAGEMENT OF SOFTWARE ARCHITECTURE

Architectural change influences the software on multi-levels and across all modules. It is thus crucial to continuously and actively manage and control any change to the architecture, to have an elaborate representation of it and to communicate the same with all concerned parties [20].

3) FACTORC3-F3: IGNORANCE OF RIGOROUS UNIT AND INTEGRATION TESTING

The success of integration depends on many previous development phases and activities; testing being one of them. According to expert J, teams should ensure thorough unit testing and integration testing to allow for fault-free integration.

4) FACTORC3-F4: AD-HOC RE-PLANNING

In GSD projects, the practice is to create project and integration plans from the requirements that are initially provided. Problem arises when teams at different sites change the plan or the requirements without fully understanding the effect it may have on dependent objects that are being developed at other sites. Most often these changes are not even communicated to all concerned parties. This can be a disaster at the time of integration [39].

5) FACTORC3-F5: INCOMPATIBILITIES BETWEEN COMPONENTS

In a study by Bosch and Bosch-Sijtsema [40], it is noted that component incompatibilities are identified during integration stage. The conflicting attributes of independently developed modules make it difficult to test the end-to-end scenarios. This makes the integration process more time consuming, costly and unpredictable causing major difficulties for all the parties involved.

6) FACTORC3-F6: INFORMAL SPECIFICATIONS REFINEMENT

Any changes brought into the original plan always result in complications later even if these updates are done to refine or improve the original specifications. Most often the updates are not documented thoroughly. Incomplete documentation causes problems for software management and testing. Handling the changes during the integration is a major problem especially if the people who originally worked on the project are no longer available [5].

7) FACTORC3-F7: DEVIATION FROM AGREED ARCHITECTURAL SPECIFICATIONS

Cataldo *et al.* [41] discuss an interesting phenomenon called ''architectural drift''. The teams at remote locations deviate from the architecture that is agreed upon and communicated to all teams. Resultantly individual components or subcomponents are not developed based on a single architecture and therefore are not easy to integrate.

8) FACTORC3-F8: BADLY ENGINEERED SOFTWARE

Expert A emphasizes the significance of re-factoring the code by arguing that complex code can be difficult to modify and maintain. Any ill-hearted attempt at it leads to creation of new defects causing delays and resource wastage. Code analysis and architectural information should guide the process of refactoring.

9) FACTORC3-F9: SHARING UNTESTED VERSION OF COMPONENTS USING INTEGRATION CENTRIC APPROACH

Organizations using an integration centric approach rely on integration phase to test various components of the software. By this time, defect detection and resolution has become an expensive problem [40].

10) FACTORC3-F10: POORLY COMMUNICATED MODULE REQUIREMENTS

According to expert K, ''for any developer to fully understand the requirements, it is necessary that he sees how any particular component fits in the overall picture. The dependencies and constraints can be implemented only when the need is clearly understood. This must be communicated timely to remove the inaccuracy threat.''

D. CLASS C4: COMMUNICATION BARRIERS

Distributed teams' effective communication is crucial for timely product delivery and feature completion. Geographical, cultural and temporal distances affect the trust building among the remote team members. Communication is hindered due to various reasons, most of which are prominent in distributed development as given in Table 6.

TABLE 6. Factors associated with communication barriers.

Factor ID	Integration Failure Factors
C4-F1	Lack of face to face meeting during integration process
C4-F2	Lack of informal and external communication
C4-F3	Communication gap between teams at different sites
C4-F4	Inadequate understanding of requirements and interface issues
C4-F5	High time zone difference
C4-F6	Increase in number of sites

1) FACTORC4-F1: LACK OF FACE TO FACE MEETING DURING INTEGRATION PROCESS

Lack of face to face meeting during integration process between remote development teams and the team that performs system integration produces a lot of problems during integration process. One of the study [42] states that it is very hard to create harmony in remote teams. To achieve this, project management teams often travel to meet the remote teams [42]. Another study [43] describes that the system integration in many GSD projects is performed either by offshore or outsource team. One of the industrial experts of the study also highlighted the issue that without face to face communication between remote teams, it is very hard to avoid integration problems.

2) FACTORC4-F2: LACK OF INFORMAL AND EXTERNAL COMMUNICATION

Spontaneous, external and informal communication often lack in GSD projects due to teams' remote locations [11], [27], [44], [45]. Because of this problem, team members are often unaware of each other activities. In addition, it is very hard to predict each other's progress especially in terms of task completion. The delayed development at any site can create frustration for other project members [45].

The team member at remote location often requires informal communication to understand design decision [29]. The formal communication with the help of internet technology cannot make up for the lack of informal communication problems [11].

3) FACTORC4-F3: COMMUNICATION GAP BETWEEN TEAMS AT DIFFERENT SITES

According to experts D, E, G, H, I and K, team members of different locations often belong to different ethnic backgrounds and cultures. These members usually have different native languages and they communicate with each other in English. Having this situation, they interpret the clearly explained requirements in different ways leading to different implementations. These 'differences' become visible at integration time and may lead to failures.

4) FACTORC4-F4: INADEQUATE UNDERSTANDING OF REQUIREMENTS AND INTERFACE ISSUES

Inadequate understanding of requirements and interface issues is the common reason for the integration failure. Sometimes requirements and interface agreements are interpreted incorrectly. One of the main reasons for this, is that the project members at remote sites have different assumptions about other subsystems. These different assumptions are also believed during unit tests and appear at the time of integration [8]. According to other studies [29], [32], the essential details are often missing in interface specification. These details include return values, return type, message type, assumption about performance constraint and input parameters.

Another study [45] concludes that the integration problems which arise due to lack of common understanding in requirement specification remain hidden throughout the development cycle. Expert C also mentions this problem as developers at remote locations often do not have common understanding of requirements and interface specification which leads to serious integration problems.

5) FACTORC4-F5: HIGH TIME ZONE DIFFERENCE

According to studies [27], [4] and many industrial experts [A, E, G, H, I, K], high time zone difference is one of the main reasons behind the defects that are detected during system integration phase. The experts collectively describe

that normally, emails are used to discuss the problems with remote teams and clients in case of offshore development with high time zone difference. During critical stages, the feedback or the problem to be discussed between remote teams often takes more than one-day time.

6) FACTORC4-F6: INCREASE IN NUMBER OF SITES

A study [19] suggests that increase in the number of sites belonging to different countries, increases the likelihood of defects during the system integration. In addition, the increased number of sites leads to more dependencies among the sites that complicates the coordination efforts. The increase in dependencies might stem from multiple sources such as the distribution of roles and responsibilities or technical properties of the system.

E. CLASS C5: POOR RESOURCE MANAGEMENT

Distribution of development teams is challenging in terms of task management with respect to time, tools and technology unifications and integration of various software components. Factors related to poor resource management are presented in Table 7

TABLE 7. Factors associated with poor resource management.

Factor ID	Integration Failure Factors
C5-F1	Immature global infrastructure
C5-F2	Information Overload

1) FACTORC5-F1: IMMATURE GLOBAL INFRASTRUCTURE

Immature global infrastructure also affects the integration process in GSD negatively. It includes limited network bandwidth among remote sites, hardware and firmware connectivity, and incomplete hardware configuration in distributed environment. Some studies [45], [47] highlight this problem by analyzing GSD projects in case of which global infrastructure is not good enough to address distributed environment issues. Sometimes the network bandwidth between remote sites is not capable of handling large amount of data traffic across the network. As a result, the developers at remote locations working on same project face long delays [47]. The experts C and G also point out that different hardware environments invoke integration problems.

2) FACTORC5-F2: INFORMATION OVERLOAD

The information overload can cause integration failure [14]. Due to information overload, people at remote locations often are unable to receive important messages. For instance, in case study [22], the person responsible for integration sent a notification to the developers at remote locations to keep related files intact. At a remote location, a developer was unable to read that notification due to information overload and made changes to the files that resulted in integration failure.

F. CLASS C6: IMPROPER REQUIREMENTS CHANGE MANAGEMENT

Requirements engineering is an important phase of software development lifecycle. In globally distributed development effort, any changes to requirements need to be communicated to all concerned parties at different sites. Requirements change management is often not given the importance that leads to problems at later stages of software development. In globally distributed teams, the decision power often gets split between the onsite team which most often has a direct communication with the client and an offsite/offshore team. This setting requires more rigorous requirements engineering process and change management process [55]. Table 8 presents the factors associated with improper requirements change management.

TABLE 8. Factors associated with improper requirements change management.

Factor ID	Integration Failure Factors
C6-F1	Feature addition
C6-F2	Last time changes in product release
C6-F3	Rapidly changing requirements and unexpected technical interdependencies
C6-F4	Change in requirements regarding technology and functionality

1) FACTORC6-F1: FEATURE ADDITION

During GSD, when new features are added to the system without formal communication to all teams, the features are prone to defects that become apparent during the system integration and testing phase. The added features are not properly documented, communicated, and checked for dependencies and constraints [17].

2) FACTORC6-F2: LAST TIME CHANGES IN PRODUCT RELEASE

According to expert C, any changes made to a product version that is ready for release often lead to problems due to shortage of time. The effect of changes to other modules is neither carefully studied nor documented or communicated.

3) FACTORC6-F3: RAPIDLY CHANGING REQUIREMENTS AND UNEXPECTED TECHNICAL INTERDEPENDENCIES

A successful integration is achieved when all aspects of software development are considered. Cusumano [48] discusses that integration failure sometimes can be attributed to delays and erroneous shift from design to implementation. Technical interdependencies between distributed modules should be handled carefully. Rapidly changing requirements badly affect the integration process. Underestimating the time and resources needed for integration, in complex systems where many parties are involved, is a classical reason for integration failure [4].

4) FACTORC6-F4: CHANGE IN REQUIREMENTS REGARDING TECHNOLOGY AND FUNCTIONALITY

Changes to a plan are always costly. Particularly, changing the technology or functionality at later development stages

can cause chaos for the integration plan [1]. The system may require major additions in terms of functionalities and even technologies. A change management procedure should be in place to address such scenarios [1].

G. CLASS C7: INADEQUATE TEAM EXPERTISE

Small teams with little or no experience in GSD can jeopardize the project. The inexperience may be in terms of tool and technology management, or people management. Herbsleb and Grinter [5] recommend that educational trainings for software developers should include team-oriented training with global development perspective. Table 9 highlights the factors related to inadequate team expertise.

TABLE 9. Factors associated with inadequate team expertise.

Factor ID	Integration Failure Factors
C7-F1	Module incompatibilities due to lack of tool and technology knowledge
C7-F2	Lack of required knowledge and skills regarding integration

1) FACTORC7-F1: MODULE INCOMPATIBILITIES DUE TO LACK OF TOOL AND TECHNOLOGY KNOWLEDGE

Module incompatibilities occur when we try to integrate modules developed with different technologies by independent teams. Avram *et al.* [49] document the effects of using two different configuration management tools by teams working on the same software. According to the study, due to unfamiliarity with tools, the teams are required to perform extra work during integration and release. The same has been backed by the experts A, B, E and F. Expert A proposes that by performing dynamic and static code analysis, training developers in different technologies, and conducting peer reviews can mitigate this problem.

2) FACTOR C7-F2: LACK OF REQUIRED KNOWLEDGE AND SKILLS REGARDING INTEGRATION

A successful integration requires the presence of a domain expert at the time of integration. Often integration phase is not given the required importance and a site expert is left out. The integration teams cannot conduct a successful integration in the absence of necessary system and domain expertise [20]. Developers leading the project in the start may also not be available by the time of integration. In some instances, the required expertise towards the end of project are unknown in the beginning of project [4].

H. CLASS C8: IMPROPER TOOL SELECTION

By essence, distributed development occurs on unconnected locations, which at times leads to presence of different development setup at different sites. The processes being followed at each site can also be different due to various social, political or economic factors. This leads to inconsistencies even in terms of tools being used at different sites. The factors about improper tools selection are given in Table 10.

TABLE 10. Factors associated with improper tool planning.

Factor ID	Integration Failure Factors
C8-F1	Different versions of tools
C8-F2	Avoiding tools and processes to manage and control architecture evolution
C8-F3	Inappropriate design documentation technology selection

1) FACTORC8-F1: DIFFERENT VERSIONS OF TOOLS

Organizations establish their own work environment and have a set of tools and technologies they work with. In cases of mergers and unions, different working practices are inherited. Software's developed in such environments lead to artifacts that cannot be integrated [50]. As expert F describes that the synchronization between numerous components becomes an acute challenge when development goes global.

2) FACTOR C8-F2: AVOIDING TOOLS AND PROCESSES TO MANAGE AND CONTROL ARCHITECTURE EVOLUTION

Some GSD projects ignore the use of tools for management of processes and changes in requirements and architecture. Rigorous evolutions need a lot of re-work that often leads to teams' unwillingness to control architecture evolution. Following the defined processes, in such circumstances, becomes challenging. All this leads to an inadequate and inappropriate integration strategy [51].

3) FACTORC8-F3: INAPPROPRIATE DESIGN DOCUMENTATION TECHNOLOGY SELECTION

Developers at multiple sites rely on design documentation for accurate information about what to build. Any document that cannot incorporate the complexities of the requirements or does not reflect later changes can lead to development of incorrect functionalities. Most often this problem arises because of inappropriate technology or tool selection for document management [5].

V. CONCLUSION

Successful integration is one of the biggest challenges in GSD that is compromised by many factors. The aim of this study is to explore the factors that may cause integration failure during GSD. The study identifies40 unique factors through extensive literature review and survey with GSD industrial experts. The factors cover various aspects of global software development from the beginning of the process to its very end, including factors related to tools and technologies, project management, immature processes and deviations from defined processes.

The proposed taxonomy classifies the integration failure factors into eight classes based on the underlying causes which lead to the failure. The taxonomy includes the classes: lack of coordination between remote sites (C1), inadequate process planning (C2), inadequate process implementation (C3), communication barriers (C4), poor resource management (C5), improper requirements change management (C6), inadequate team expertise (C7), and improper tool selection (C8). This taxonomy allows us to focus more on problem domain rather than the individual factors. It can be

very effective to identify integration problems that may help in devising their solution. It may also help to formulate a fruitful integration strategy at early stages of GSD projects.

REFERENCES

[1] C. V. Ramamoorthy, "Distributed techniques in software systems integration," in *Proc. 5th IEEE Comput. Soc. Workshop Future Trends Distrib. Comput. Syst.*, Aug. 1995, pp. 252–256.
[2] O. Tufekci, S. Cetin, and A. Arifoglu, "Proposing a federated approach to global software development," in *Proc. IEEE 4th Int. Conf. Digit. Soc. (ICDS)*, Feb. 2010, pp. 150–157.
[3] D. Šmite, C. Wohlin, T. Gorschek, and R. Feldt, "Empirical evidence in global software engineering: A systematic review," *Empirical Softw. Eng.*, vol. 15, no. 1, pp. 91–118, 2010.
[4] J. Ammerlaan, "Identifying pitfalls of system integration—An exploratory study," in *Proc. IEEE Int. Conf. Softw. Test. Verification Validation Workshop (ICSTW)*, Apr. 2008, pp. 331–338.
[5] J. D. Herbsleb and R. E. Grinter, "Splitting the organization and integrating the code: Conway's law revisited," in *Proc. ACM 21st Int. Conf. Softw. Eng.*, May 1999, pp. 85–95.
[6] D. Smite, C. Wohlin, R. Feldt, and T. Gorschek, "Reporting empirical research in global software engineering: A classification scheme," in *Proc. IEEE Int. Conf. Global Softw. Eng. (ICGSE)*, Aug. 2008, pp. 173–181.
[7] S. Betz, J. Makio, and R. Stephan, "Offshoring of software development—Methods and tools for risk management," in *Proc. 2nd IEEE Int. Conf. Global Softw. Eng. (ICGSE)*, Aug. 2007, pp. 280–281.
[8] B. Sengupta, S. Chandra, and V. Sinha, " A research agenda for distributed software development," in *Proc. ACM 28th Int. Conf. Softw. Eng.*, May 2006, pp. 731–740.
[9] T. Poikolainen and J. Paananen, "Performance criteria in inter-organizational global software development projects," in *Proc. 2nd IEEE Int. Conf. Global Softw. Eng. (ICGSE)*, Aug. 2007, pp. 60–70.
[10] S. Abufardeh and K. Magel, "The impact of global software cultural and linguistic aspects on global software development process (GSD): Issues and challenges," in *Proc. IEEE 4th Int. Conf. New Trends Inf. Sci. Service Sci. (NISS)*, May 2010, pp. 133–138.
[11] O. Gotel, V. Kulkarni, C. Scharff, and L. Neak, "Integration starts on day one in global software development projects," in *Proc. IEEE Int. Conf. Global Softw. Eng. (ICGSE)*, Aug. 2008, pp. 244–248.
[12] L. D. Panjer, D. Damian, and M.-A. Storey, "Cooperation and coordination concerns in a distributed software development project," in *Proc. ACM Int. Workshop Cooperat. Hum. Aspects Softw. Eng.*, May 2008, pp. 77–80.
[13] T. Wolf, A. Schroter, D. Damian, and T. Nguyen, "Predicting build failures using social network analysis on developer communication," in *Proc. 31st Int. Conf. Softw. Eng.*, May 2009, pp. 1–11.
[14] D. Damian, L. Izquierdo, J. Singer, and I. Kwan, "Awareness in the wild: Why communication breakdowns occur," in *Proc. 2nd IEEE Int. Conf. Global Softw. Eng. (ICGSE)*, Aug. 2007, pp. 81–90.
[15] A. Taweel, B. Delaney, T. N. Arvanitis, and L. Zhao, "Communication, knowledge and co-ordination management in globally distributed software development: Informed by a scientific software engineering case study," in *Proc. 4th IEEE Int. Conf. Global Softw. Eng. (ICGSE)*, Jul. 2009, pp. 370–375.
[16] P. S. Brockmann and T. Thaumuller, "Cultural aspects of global requirements engineering: An empirical Chinese-German case study," in *Proc. 4th IEEE Int. Conf. Global Softw. Eng. (ICGSE)*, Jul. 2009, pp. 353–357.
[17] M. Cataldo and S. Nambiar, "The impact of geographic distribution and the nature of technical coupling on the quality of global software development projects," *J. Softw., Evol. Process*, vol. 24, no. 2, pp. 153–168, 2012.
[18] D. Smite, "Project outcome predictions: Risk barometer based on historical data," in *Proc. 2nd IEEE Int. Conf. Global Softw. Eng. (ICGSE)*, Aug. 2007, pp. 103–112.
[19] R. T. Nakatsu and C. L. Iacovou, "A comparative study of important risk factors involved in offshore and domestic outsourcing of software development projects: A two-panel Delphi study," *Inf. Manag.*, vol. 46, no. 1, pp. 57–68, 2009.
[20] R. Kommeren and P. Parviainen, "Philips experiences in global distributed software development," *Empirical Softw. Eng.*, vol. 12, no. 6, pp. 647–660, 2007.
[21] R. Prikladnicki, J. L. N. Audy, D. Damian, and T. C. de Oliveira, "Distributed Software Development: Practices and challenges in different business strategies of offshoring and onshoring," in *Proc. 2nd IEEE Int. Conf. Global Softw. Eng. (ICGSE)*, Aug. 2007, pp. 262–274.
[22] S. Keele, "Guidelines for performing systematic literature reviews in software engineering," Dept. Comput. Sci., Univ. Durham, Durham, U.K., Tech. Rep. EBSE Version 2.3, 2007.
[23] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *J. Syst. Softw.*, vol. 80, no. 4, pp. 571–583, 2007.
[24] C. Willig, *Introducing Qualitative Research In Psychology*. New York, NY, USA: McGraw-Hill, 2013.
[25] B. A. Farshcian, "Integrating geographically distributed development teams through increased product awareness," *Inf. Syst.*, vol. 26, no. 3, pp. 123–141, 2001.
[26] M. Jiménez, M. Piattini, and A. Vizcaíno, "Challenges and improvements in distributed software development: A systematic review," *Adv. Softw. Eng.*, vol. 2009, Mar. 2009, Art. no. 710971, doi: 10.1155/2009/710971.
[27] M. Cataldo and S. Nambiar, "On the relationship between process maturity and geographic distribution: An empirical analysis of their impact on software quality," in *Proc. 7th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng.*, Aug. 2009, pp. 101–110.
[28] C. R. de Souza, S. Quirk, E. Trainer, and D. F. Redmiles, "Supporting collaborative software development through the visualization of socio-technical dependencies," in *Proc. Int. ACM Conf. Supporting Group Work*, Nov. 2007, pp. 147–156.
[29] P. Ovaska, M. Rossi, and P. Marttiin, "Architecture as a coordination tool in multi-site software development," *Softw. Process, Improvement Pract.*, vol. 8, no. 4, pp. 233–247, 2003.
[30] J. Kotlarsky, P. C. van Fenema, and L. P. Willcocks, "Developing a knowledge-based perspective on coordination: The case of global software projects," *Inf. Manag.*, vol. 45, no. 2, pp. 96–108, 2008.
[31] J. C. Jacobs, J. H. van Moll, P. J. Krause, R. J. Kusters, and J. J. M. Trienekens, "Effects of virtual development on product quality: Exploring defect causes," in *Proc. IEEE 7th Annu. Int. Workshop Softw. Technol. Eng. Pract.*, Sep. 2003, pp. 6–15.
[32] M. Cataldo, J. D. Herbsleb, and K. M. Carley, "Socio-technical congruence: A framework for assessing the impact of technical and work dependencies on software development productivity," in *Proc. 2nd ACM-IEEE Int. Symp. Empirical Softw. Eng. Meas.*, Oct. 2008, pp. 2–11.
[33] R. Sangwan, C. Neill, M. Bass, and Z. El Houda, "Integrating a software architecture-centric method into object-oriented analysis and design," *J. Syst. Softw.*, vol. 81, no. 5, pp. 727–746, 2008.
[34] M. T. Lane and P. J. Ågerfalk, "On the suitability of particular software development roles to global software development," in *Proc. IEEE Int. Conf. Global Softw. Eng. (ICGSE)*, Aug. 2008, pp. 3–12.
[35] E. Ó. Conchúir, H. Holmstrom, P. J. Ågerfalk, and B. Fitzgerald, "Exploring the assumed benefits of global software development," in *Proc. IEEE Int. Conf. Global Softw. Eng. (ICGSE)*, Oct. 2006, pp. 159–168.
[36] E. MacGregor, Y. Hsieh, and P. Kruchten, "The impact of intercultural factors on global software development," in *Proc. IEEE Can. Conf. Elect. Comput. Eng.*, May 2005, pp. 920–926.
[37] P. Wongthongtham, E. Chang, T. S. Dillon, and I. Sommerville, "Ontology-based multi-site software development methodology and tools," *J. Syst. Archit.*, vol. 52, no. 11, pp. 640–653, 2006.
[38] J. Jacobs, J. van Moll, P. Krause, R. Kusters, J. Trienekens, and A. Brombacher, "Exploring defect causes in products developed by virtual teams," *Inf. Softw. Technol.*, vol. 47, no. 6, pp. 399–410, 2005.
[39] N. Mullick *et al.*, "Siemens global studio project: Experiences adopting an integrated GSD infrastructure," in *Proc. IEEE Int. Conf. Global Softw. Eng. (ICGSE)*, Oct. 2006, pp. 203–212.
[40] J. Bosch and P. Bosch-Sijtsema, "From integration to composition: On the impact of software product lines, global development and ecosystems," *J. Syst. Softw.*, vol. 83, no. 1, pp. 67–76, 2010.
[41] M. Cataldo *et al.*, "CAMEL: A tool for collaborative distributed software design," in *Proc. 4th IEEE Int. Conf. Global Softw. Eng. (ICGSE)*, Jul. 2009, pp. 83–92.
[42] H. Holmstrom, E. Ó. Conchúir, P. J. Ågerfalk, and B. Fitzgerald, "Global software development challenges: A case study on temporal, geographical and socio-cultural distance," in *Proc. IEEE Int. Conf. Global Softw. Eng. (ICGSE)*, Oct. 2006, pp. 3–11.
[43] P. Wongthongtham, E. Chang, and T. Dillon, "Multi-site distributed software development: Issues, solutions, and challenges," in *Computational Science and Its Applications—ICCSA*. Berlin, Germany: Springer, 2007, pp. 346–359.

[44] M. Cataldo and S. Nambiar, "Quality in global software development projects: A closer look at the role of distribution," in *Proc. 4th IEEE Int. Conf. Global Softw. Eng. (ICGSE)*, Jul. 2009, pp. 163–172.
[45] J. D. Herbsleb, D. J. Paulish, and M. Bass, "Global software development at siemens: Experience from nine projects," in *Proc. IEEE 27th Int. Conf. Softw. Eng. (ICSE)*, May 2005, pp. 524–533.
[46] M. Cataldo and J. D. Herbsleb, "Communication networks in geographically distributed software development," in *Proc. ACM Conf. Comput. Supported Cooperat. Work*, Nov. 2008, pp. 579–588.
[47] M. Yap, "Follow the sun: Distributed extreme programming development," in *Proc. IEEE Agile Conf.*, Jul. 2005, pp. 218–224.
[48] M. A. Cusumano, "Managing software development in globally distributed teams," *Commun. ACM*, vol. 51, no. 2, pp. 15–17, 2008.
[49] G. Avram, L. Bannon, J. Bowers, A. Sheehan, and D. K. Sullivan, "Bridging, patching and keeping the work flowing: Defect resolution in distributed software development," *Comput. Supported Cooperat. Work*, vol. 18, nos. 5–6, p. 477, 2009.
[50] D. Redmiles *et al.*, "Continuous coordination: A new paradigm to support globally distributed software development projects," *WirtschaftsInformatik*, vol. 49, no. 1, p. 28, 2007.
[51] F. Salger, "Software architecture evaluation in global software development projects," in *On the Move to Meaningful Internet Systems: OTM Workshops*. Berlin, Germany: Springer, 2009, pp. 391–400.
[52] M. Davies, "Concept mapping, mind mapping and argument mapping: What are the differences and do they matter?" *Higher Edu.*, vol. 62, no. 3, pp. 279–301, 2011.
[53] N. Ramasubbu, "Governing software process improvementsin globally distributed product development," *IEEE Trans. Softw. Eng.*, vol. 40, no. 3, pp. 235–250, Mar. 2014.
[54] R. Prikladnicki, J. L. N. Audy, and R. Evaristo, "Global software development in practice lessons learned," *Softw. Process, Improvement Pract.*, vol. 8, no. 4, pp. 267–281, 2003.
[55] J. M. Bhat, M. Gupta, and S. N. Murthy, "Overcoming requirements engineering challenges: Lessons from offshore outsourcing," *IEEE Softw.*, vol. 23, no. 5, pp. 38–44, Sep./Oct. 2006.

ATIQUE AHMAD ZAFAR received the M.S. degree in software engineering with specialization in technical management track from the Blekinge Institute of Information Technology, Sweden, in 2010. He has been a Lecturer of computer science with the COMSATS Institute of Information Technology (Islamabad Campus), Pakistan, since 2011. His area of research is Global software development challenges.

SHAHELA SAIF received the M.S. degree in computer software engineering from NUST in 2012. She is currently a Lecturer with the COMSATS Institute of Information Technology, Islamabad, Pakistan. Her interest areas include software development, GSD, and machine learning.

MUZAFAR KHAN received the M.Sc. degree in software engineering from the Blekinge Institute of Technology, Sweden, in 2008 the Ph.D. degree in human–computer interaction from Universiti Teknologi PETRONAS, Malaysia, in 2012. He is currently an Assistant Professor with the Department of Computer Science, COMSATS Institute of Information Technology, Islamabad, Pakistan. He has keen interest in interdisciplinary research related to human–computer interaction and software engineering.

JAVED IQBAL received the Ph.D. degree in computer science from the University of Malaya, Malaysia, in 2016. He is currently an Assistant Professor with the Department of Computer Science, COMSATS Institute of Information Technology, Islamabad, Pakistan. His areas of interest are software process improvement, requirements engineering, and software development outsourcing.

ADNAN AKHUNZADA is currently as an Assistant Professor and Incharge Programme (Software Engineering and Telecommunication Networks) with the COMSATS Institute of Information Technology, Islamabad, Pakistan. He got a great experience of teaching international modules and a renowned member of several research communities. He has authored several high impact research journals, IEEE TRANSACTIONS, highly reputable magazines, book chapter, and conference papers. His current research interests include secure and dependable software defined networks, light weight cryptography, man-at-the-end attacks, human attacker attribution and profiling, remote data auditing, and modeling secure software defined smart cities.

ABDUL WADOOD received the Ph.D. degree in signal and image processing from the University of Poitiers, France, in 2011. He is currently an Assistant Professor with the Department of Computer Engineering, CCIS, King Saud University. His research interests are focused on color image watermarking, multimedia security, steganography, fingerprinting, and biometric template protection.

AHMAD AL-MOGREN received the Ph.D. degree in computer sciences from Southern Methodist University, Dallas, TX, USA, in 2002. He was an Assistant Professor of computer science and a member of the Scientific Council with the Riyadh College of Technology. He served as the Dean of the College of Computer and Information Sciences and the Head of the Council of Academic Accreditation, Al Yamamah University. He is currently an Associate Professor and the Vice Dean for the development and quality with the College of Computer and Information Sciences, King Saud University, Saudi Arabia. His research areas of interest include mobile and pervasive computing, computer security, sensor and cognitive network, and data consistency. He has served as a guest editor for several computer journals.

ATIF ALAMRI received the B.Sc. and M.Sc. degrees in information systems from College of Computer and Information Sciences (CCIS), King Saud University (KSU), Riyadh, Saudi Arabia, in 2000 and 2004, respectively, and the Ph.D. degree in computer science from the School of Information Technology and Engineering, University of Ottawa, Canada, in 2010. He is currently an Associate Professor with the Information Systems Department, CCIS, KSU. He is one of the founding members of the Chair of Pervasive and Mobile Computing, CCIS, KSU, and successfully managing its research program, which transformed the chair as one of the best chairs of research excellence in the college. He is also acting as an Assistant Vice-Rector for Technical, Vice President's Office for Quality and Development, KSU. His research areas of interest are multimedia assisted health systems, ambient intelligence, service-oriented architecture, multimedia cloud, sensor-cloud, Internet of Things, big data, mobile cloud, social network, and recommender system.