An Adaptive Agile Process Model for Global Software Development

Ritu Jain
Department of Computer Applications
S. R.G.P. Gujarati Professional Institute, Indore, M.P., India
ritujain_doc@rediffmail.com

Ugrasen Suman
School of Computer Science& IT
Devi Ahilya University, Indore, M.P., India
ugrasen123@yahoo.com

***Abstract*— Global software development (GSD) has established its dominance in software industry whereas agile and lean practices are also believed to be the most promising software development approaches. Fierce competition and dynamic business environment has tempted software industry to experiment agile as well as lean approaches in GSD projects. However, as lean and agile methods are primarily developed for collocated development, these methods do not consider the impact of GSD distances and this amalgamation creates some discrepancies. These methods need to be modified for GSD. In this paper, we have proposed a GSD process model based on adaptive agile and kanban approach. The proposed approach can act as a guideline to GSD practitioners for effective software development.**

Keywords- global software development; agile; kanban; scrumban; scrum; extreme programming

I. Introduction

Software companies usually distribute their development activities globally to achieve several benefits. These benefits comprise cheaper professionals, shorter cycle time, and expansion of international market. However, these benefits could only be partly attained due to geographical, temporal, socio-cultural, and organizational distances across different countries and organizations [1, 2]. Geographical distance leads to inadequate communication, decreased project visibility, reduced shared understanding, and knowledge management difficulties. Temporal distance results into delayed feedback loop between distributed members and thus slow down resolution of impediments. Socio-cultural distance causes linguistic differences which lead to insufficient communication and misunderstandings. It also causes differences in work ethics, lack of trust, and reduced team cohesiveness. Inconsistent tools, standards, work practices, and processes are some of the challenges due to organizational distances. Thus, these distances cause various communication, coordination and collaboration problems [3]. Owing to these challenges, more than half of the GSD projects do not attain the intended goals and are terminated before deadline [4].

Agile practices provide software projects with flexibility, productivity, customer satisfaction, and reduced cycle time. Enhanced communication and collaboration, decreased defect rates, improved team performance, and capability to handle changing requirements are few of the benefits that agile methods provide over traditional methods [5, 6]. The success rate of agile projects is 42% as compared to 14% of waterfall projects [6]. Lean principles are also known to continuously optimize the process by removing all sorts of waste, increase the workflow and can further reduce lead time. Kanban is a lean approach focusing on continuous development and improvement. It improves project visualization and identifies bottlenecks by limiting “Work in progress” (WIP) [1, 6, 7].

In order to solve GSD challenges and gain the competitive edge provided by agile methods, practitioners have started applying agile methods and more recently lean principles based kanban approach in GSD [8, 9, 10]. Agile methods alleviate some of the challenges of GSD by improving communication, coordination and collaboration [5]. However, as these methods were primarily intended for collocated development, adaptation is required for their application in GSD. Although, numerous suggestions and best practices regarding agile in GSD are present in literature [4], but few research have been performed related to agile GSD process models [11]. Moreover, application of kanban in GSD is an emerging research domain. Thus, a development process utilizing the virtues of agile as well as kanban in GSD can help practitioners in reducing GSD challenges and achieving potential benefits [11].

In order to design a software process model for GSD, we have first extracted various challenges faced in GSD life cycle through systematic literature review [3]. Subsequently, effectiveness of various agile practices in GSD setting has been investigated. Supporting practices that can supplement the application of agile practices in distributed projects are also explored [5]. Lean principles and associated kanban software development approach which can be helpful in GSD are also mined through literature [12, 13, 14, 15, 16]. Therefore, a process model for GSD is devised by mapping benefits of agile and kanban to challenges faced by distributed practitioners during software development lifecycle (SDLC). This process model can direct practitioners to effectively develop software with project team distributed globally.

The paper is structured as follows: Section II presents a literature survey on distributed agile software development (DASD) process models. Section III introduces proposed process model for GSD and section IV maps GSD challenges to proposed process model, and finally section V concludes the paper.

II. Background and Related Work

A software process is an umbrella of activities, actions, and tasks that are carried out to create software. It is a road map, which guides project team to build high quality software within budget and time. It plays an important role in deciding the quality of final software product and ensuring customer satisfaction. Distributed software development is considered more complicated than collocated projects as it is enfolded by many challenges during entire development cycle. Agile methods alleviate GSD challenges related to communication, coordination, and collaboration to some extent. Several problems related to project visibility, progress monitoring, task awareness, shared understanding, issues and misunderstandings observed in distributed projects can be reduced by organizing recurrent scrum meetings such as daily scrum, scrum of scrums, sprint planning, and retrospectives. Various risks associated with GSD can be diminished by continuous feedback and regular demonstrations facilitated through agile practices. Configuration management issues in GSD projects can be reduced by continuous integration. Also, test driven development (TDD) act as a lightweight and unambiguous communication technique and supports cross-site communication among distributed teams. Scrum practices such as daily scrum, sprint planning, sprints, scrum of scrums, and retrospective meetings facilitate collaboration, trust, and team spirit [5].

Kanban is a software development approach based on lean principles. It visualize the process, amplify the workflow, identify bottlenecks by plotting development lifecycle phases in different columns on a kanban board and restricting WIP in each column [6, 12]. It handles changing requirement, enhances communication, cooperation and coordination between stakeholders [10, 17]. It clarifies process policies, reduces defects, enhances quality, improves team motivation as well as performance, reduces lead time, and thus improves customer satisfaction and trust [10, 18]. A hybrid approach based on scrum and kanban, namely Scrumban focuses on optimal utilization of resources, visualization of workflow by limiting WIP, and customer satisfaction. It promotes cross-functional, self-organizing and collaborative teamwork [17, 19].

Agile methods offers iterative and incremental software process management framework with periodic feedback. They coordinate the team through time-boxed iterations and several meetings such as daily scrum, sprint planning, sprint review, and retrospectives. It is expected that all team members should actively participate and communicate with others in the meetings, which however is less feasible in distributed teams. Awkward timings of these distributed meetings due to time zone difference also proved them painful. Sprints executed at distributed sites suffer from synchronization problems due to various holiday patterns [5]. If sprint are quite long then meetings will be conducted less frequently as compared to short sprints and there will be a possibility of producing twice wrong functionality. On the other hand, very short underestimated time-boxed iterations force the team to rush to complete each and every planned user story which can lead to mediocre quality product, ad-hoc bug fixing, and decreased productivity at the end of sprint [18]. Although agile methods improve communication, coordination, and collaboration in GSD projects to some extent, but scaling agile methods with the help of kanban can facilitate higher performance in GSD teams. Kanban provides greater visualization of workflow by limiting WIP and thus identifies bottlenecks quickly, continuously improves process, and reduce the lead time [6]. It avoids time-boxed iterations, decouples planning activities (prioritization, estimation) from release, and emphasizes continuous improvement (kaizen) through regular delivery [19]. Scrumban accrue favorable features of scrum and removes superfluous constraints, for instance sprints and several scrum meetings. Thus, we have proposed a process model based on a hybrid approach adapting features from XP, scrum, and kanban. This approach, if applied properly can balance agility and discipline needed for GSD.

Several researchers have proposed agile process model for GSD. Nuevo et al. proposed a hybrid methodology based on scrum and RUP approaches. They have discussed techniques for distributed team structure and its formation; strategies adopted for coordination and distributed meetings. The proposed process model had employed RUP for software development and scrum for project management [20]. Alqahtani et al. had discussed the challenges of distributed agile software development and techniques to mitigate these challenges. They have also proposed development framework for distributed agile development. However, only brief overview of the framework is given without elaborating the details [21]. Akbar et al. have proposed a distributed agile development (DAD) model especially designed for global web based projects. They have considered only four out of twelve agile principles for designing the model. Merely two contradicting characteristics of agile and distributed development are focused in DAD model. Thus, the model simply provides a general approach for web based DASD [22]. The process model proposed in this paper adapts the virtues of agile as well as Kanban for GSD to alleviate aforementioned challenges. Agile practices and kanban are meticulously combined and modified to accrue their good features and avoid unsuitable features for GSD.

III. Adaptive Agile Process Model

The proposed process model shown in Fig. 1 is composed of five phases; namely, requirement exploration, collocated planning and high level design, construction, release, and maintenance. The cross-supporting repository is also used in this process model. The detailed description of phases and practices are discussed in subsequent sub-sections. The process is based on continuous development approach and mainly suited for software development. Kanban board used in the process model will aid in effective project management by improving project visibility and progress monitoring. This process can be implemented in various types of software projects such as web based, business analytics and intelligence, mobile applications, systems development, and embedded systems.

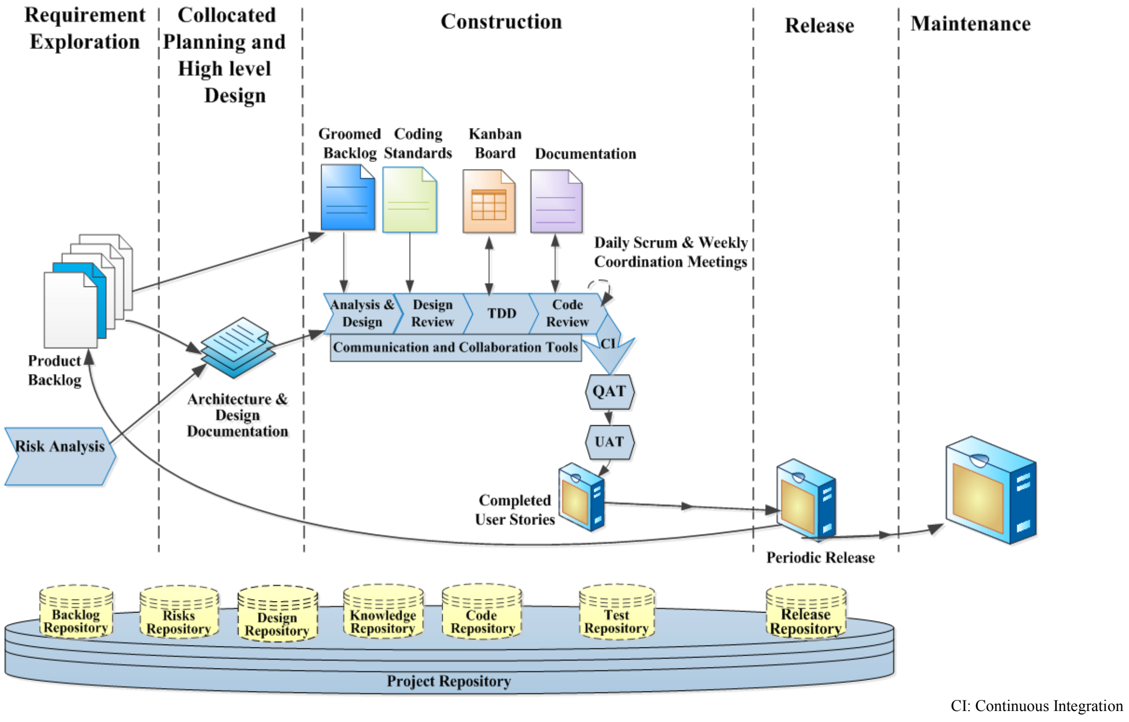


Figure 1. Adaptive Agile Process Model for GSD

A. *Requirement Exploration*

It is the most crucial phase of GSD life cycle. It encompasses overall scope and vision of the project followed by preliminary requirement collection and analysis. At the inception of the project, activities such as risk analysis, feasibility study, and estimation related to size, duration, budget, and resources are performed. Subsequently, business representatives (business owner and business analyst) communicate their project requirements to chief product owner (CPO), team product owner (TPO), and architects through several meetings. These requirements can be represented in the form of user stories. CPO prioritizes user stories in accordance with business representatives and stores them in product backlog for further discussion. An offshore senior developer or architect can act as a TPO and resides at onshore location for the project duration. He will guide and resolves confusions of the offshore team, thus work as a social and technical liaison between onshore and offshore site.

B. *Collocated Planning and High Level Design*

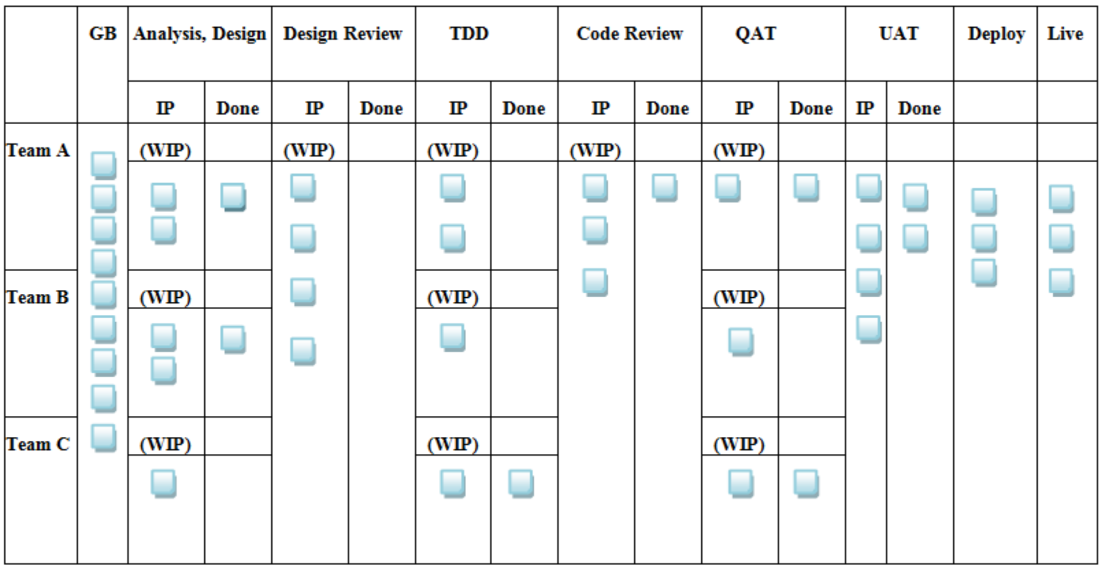
In this phase; high level design, architecture, expected time to completion, number of releases and the objectives of these releases are decided. Dependencies should be identified as early as possible to facilitate effective subsystems and task allocation, so that distributed teams can work independently but collaboratively. Design and coding standards, process policies, vision of the project are discussed and stored in globally accessible knowledge repository. These standards and policies should be understood and followed by all team members throughout the project.

In case of first time collaboration, onshore team should support and help offshore members to learn their culture, processes and technology. The whole project team could be collocated (either onshore or offshore) for discussing architecture. Short demo cum training session can be organized to brief team members about the process, tools, techniques, and terminologies to be used in the project to avoid hesitation, confusion, and misunderstandings. This meeting can be recorded and saved in knowledge repository to be able to refer afterwards. This collocation will build collective understanding and a common vision towards project. It enables architectural knowledge dissemination and offshore members would be able to understand the rationale of the architecture. Informal team building activities during the collocation would increase team acquaintance and cultural awareness, thus aid in communication, coordination and collaboration. If collocation of entire team is not possible due to budget constraints, all TPOs can travel onsite with offshore teams collaborating virtually via tools. After this collocated design phase, architecture team should participate in daily scrum meetings whenever needed to resolve confusions or convey architectural modifications to distributed members. The architecture team should also observe the fulfillment of architecture rules.

C. Construction

In this phase of GSD, development team starts implementing user stories as per customer demand followed by testing activities. A groomed backlog is prepared from product backlog in weekly coordination meetings. It contains high priority user stories to be developed next. A TPO can assign user stories to his team from groomed backlog. Developers can complete the design and analysis of a user story and publish it for TPO and other members to review. If some unresolved dependencies are found in the proposed design then CPO, TPO and associated teams can address it. Technical design documentation can be finalized after design review. Subsequently, in test driven development (TDD), developer creates unit tests for a user story, allow the test to fail, write the coding for the story and unit test it. After unit testing, he can apply refactoring to the code. He integrates his code regularly. TDD is followed by code review. A developer can ask TPO or any senior developer to review his code. These reviews can uncover any misunderstood functionalities early as well as keep the quality of code and design up-to-date. Code review provides immediate feedback to developers about their code and enhances knowledge sharing. After code review, development team handover the code to quality assurance and testing (QAT) team which further test it. Subsequently, it is handed to user acceptance testing (UAT) team and finally to deployment team. If some issues are found during QAT or UAT, it is fixed by the responsible developer and is promoted back to QAT environment. While test team is testing the code, developers can update associated documentation to resolve confusions, reduce rework and support maintenance. They also starts interacting with TPO for initial design and analysis of next user story. Distributed developers can occasionally use remote pair programming to code complicated and dependent functionalities. For simple tasks, code review can serve the purpose of pair programming in distributed environment. In case of complex projects, development can be initiated with well understood functionalities to pace and boost the development team.

The distributed development teams can use kanban board to improve visualization of work performed at different locations and identify bottlenecks during development. An electronic kanban board can be maintained for GSD projects. Fig. 2 represents the kanban board for three distributed teams. Each team would have one swim lane. A phase of a development process has two columns, viz. “In progress” and “Done“. The “In Progress” column contains operational user stories of the phase. The “Done” column contains completed user stories for the particular phase. A WIP limit is set for each phase of a swim lane which specifies maximum number of operational user stories a team can handle at a particular time in the phase. Several phases such as design, analysis, TDD, and QAT are performed at different sites thus WIP limit can be set according to number of members in the team, skill set, average experience of team members, domain knowledge, and number of projects each member is engaged. It could be adapted according to the team configuration, size as well as complexity of user stories. Design review, code review, UAT, deploy, and live phases are conducted centrally. User stories should be created such that they represent the concept of “minimum marketable feature” and their size should be almost same to facilitate the concept of WIP and to avoid large user stories to hinder the way of smaller user stories. Kanban board should be updated regularly to visualize project progress and find bottlenecks. Bottlenecks should be resolved as soon as possible to reduce lead time.



GB: Groomed Backlog, IP: In Progress

Figure 2. Format of Kanban Board for 3 Distributed Teams

On kanban board, a user story can be represented by a kanban card. A kanban card contains story ID, title, order date, start date (date at which it's development is started), and associated developer's contact details. In case of any change in user story, it should be highlighted at kanban board through different colour. Modifications can be described through changes in user story description, use cases, test cases, and dependencies and concerned persons can be immediately informed through daily scrum. This board can improve cross-site visibility issues and solve most of the coordination problems. The proposed kanban board will show the status of other teams work. It is also possible to measure the amount of work done by various teams through the kanban board. The kanban board will globally show the user stories waiting for code reviews and design reviews and thus can be pulled immediately by responsible persons, thereby reducing the waiting time of this activity.

Kanban Board visualizes the flow of user stories through various sub-phases of construction phase. All the user stories are initially stored in product backlog. High priority user stories from product backlog are selected and transferred to groomed backlog in weekly coordination meetings. A TPO can pull user stories on behalf of its team, after discussing it with CPO and assign it to its team according to their vacant WIP. User stories are broken down into tasks and low level design is prepared in design and analysis phase. User stories for which analysis and design are completed are transferred to "Done" column of analysis and design phase and subsequently pulled for design review. After review, if the capacity permits, user stories can be pulled into TDD phase where they are coded, unit tested, refactored and finally moved to "Done" column of TDD. The code of developed user stories can be reviewed by a senior developer or TPO. Once the code for a user story is reviewed and WIP of QAT column permits, it can be pulled by QAT team. If mild bugs are detected during QAT, it can be fixed by the developer while the story remains in "In Progress" state of QAT. Otherwise, the coded story has to be transferred to "In Progress" state of TDD phase of the particular team and some operational user story might be halted for some time. If these stories pass QAT, they are pulled by UAT team. After UAT testing, they are ready to be deployed and are shifted to Deploy column. Stories that are made available for use are moved to live column.

Meetings play an important role in coordinating and monitoring distributed development work. These meetings help to identify hidden issues, resolve confusions and improve project transparency. As distributed meetings are always painful for global teams, some of the scrum meetings can be avoided due to better visibility and coordination provided by electronic kanban board. In distributed projects, weekly coordination meeting can be organized in three phases. Firstly, local coordination meeting can be organized by TPO before weekly coordination meeting in which team members reviews flow of their user stories on kanban board, discusses shortcomings and suggest improvements regarding the process. The team members can present demo of work they have performed during the week. TPO can prepare minutes of meeting. Later, a distributed coordination meeting can be organized where CPO, TPOs, architects, along with business representatives prepare groomed backlog from product backlog by prioritizing, selecting, and analyzing user stories that need to be developed next. They reviews the flow of all operational user stories present on the kanban board, search bottlenecks and retrospect about what is going well, what needs improvements, and adapt the process accordingly. In this meeting, TPO presents demo of the functionality his team have completed in the week. CPO can act as proxy customer, verifies and reviews functionalities developed at various sites. Finally after distributed coordination meeting, TPO can explain the user stories lying in groomed backlog to their teams and assign them.

Daily scrum is a short meeting conducted every day for 15 minutes in which team members have to answer three questions about yesterday's accomplishment, today's goal, and impediments, if any. It plays a vital role in project monitoring and visibility. In this meeting, team reviews the progress of user stories on kanban board. A distributed daily scrum meeting can be conducted for project teams having less than 15 members. Otherwise, local daily scrum meetings can be conducted whereas CPO can attend meetings of all locations. The answers to daily scrum questions can be mailed to shorten the meeting and alleviate linguistic distance. It will further allow CPO and TPO to think about solutions to the developer's impediment. These mails, if stored in repository, can act as record of communication and improves awareness about other developer's work. These short meetings can be used to schedule longer meetings. A demo meeting can be organized in which CPO and TPOs can present demo of completed functionalities to customers as and when needed, on behalf of various distributed teams and indirectly give feedback to developers.

D. Release

The software product can be released on a pre-decided release date or according to the customers demand. The release frequency depends upon several factors such as cost of deployment, type of software product, and market competition. The completed user stories are stored in release repository. Deployment team can test and deploy it into production and inform the users about the new release.

E. Maintenance

Once all the customer requirements are fulfilled, the product moves to maintenance phase. If customer detect some bugs or need some new functionality, he can discuss it with CPO, who will in turn add it to product backlog.

F. Project Repository

A globally accessible repository can be used to store all documents related to the project. The repository can be classified into many sub-repositories such as backlog, risks, design, knowledge, code, test, and release repository.

Backlog repository contains product backlog, groomed backlog, and archive backlog. Product backlog contains all user stories requested by the customer. Groomed backlog contains high priority user stories to be developed next. These user stories can be associated with story ID, title, brief explanation (audio/video/textual) of required feature, design as well as interface specification, business driver context, short use case diagram, acceptance criteria through test cases, definition of done, order date (date at which it is chosen for development), status (halted, ready, waiting, running), and dependencies with other user stories. These detailed user stories can be coined as descriptive user story (DUS). DUS can solve problems related to insufficiently detailed, ambiguous, and misinterpreted requirements. Archive backlog contains all user stories completed by the team. The user story in archive backlog follow the syntax and semantics of DUS along with the name of its development team and its transit statistics during development phase.

Risks repository contains list of presumed risks in GSD and their mitigation actions. Project manager should perform risk analysis to find the list of risks applicable to the particular project. He can also find new risks pertaining to the project, its mitigations and add it to the repository. Design repository can contain design and architecture of the product. Architecture team can store concise and prioritized archnotes, design decisions and its rationale in this repository. Knowledge repository store design and coding standards, process policies, vision of the project, terminologies to be used in the project. It can also store successful practices, and contact information with photographs of all team members engaged in the project. The retrospection in weekly coordination meeting unveils successful practices which can contribute to knowledge repository whereas unsuccessful practices can add up to risk repository. Code repository contains code of completed user stories. Test repository holds test cases associated with user stories whereas release repository comprises code of completely tested user stories along with documentation ready to be deployed. All repositories should be updated regularly. Artefacts related to a particular user story should be linked in various sub-repositories to facilitate traceability of requirements.

G. Supporting Practices

Professional accreditation must be encouraged for offshore team members to establish trust amongst team. Language and cultural trainings can be provided to reduce the effect of various distances. Distributed teams should respect each other's culture. Team performance should be acknowledged as compared to individual performance to boost team building and cohesion. Technical as well as soft skills of distributed team members should be enhanced. Some onshore roles can have mirroring counterparts at distributed sites to push decisions making to lower levels. Frequent visits should be encouraged. Time overlap of at least two hours could facilitate synchronous communication and handover between teams. Project team should effectively use various communication and collaboration tools to diminish the effect of various distances. Synchronous audio-visual communication tools can reduce the dearth of face to face communication. Communication frequency and mechanism should be proactively monitored and managed [3, 5]. Work in progress limit should also be set for team members. None of the team member should be actively involved in more than 2 projects at a time. Sudden growing/ shrinking of team during the project should be avoided unless it is strictly necessary.

IV. DISCUSSION

GSD is a fashion followed by almost entire software industry but it enfolds numerous challenges. Software practitioners encounter these challenges throughout global software development life cycle (GSDLC). We had explored the challenges faced during GSDLC in the previous work [4]. The proposed process model can alleviate most of these challenges. The mapping of GSDLC challenges to solutions proposed in process model is shown in Table I.

TABLE I. MAPPING OF CHALLENGES TO PRACTICES PROPOSED IN PROCESS MODEL

Phase	Challenges	Solutions
RE, Design, Coding, Integration	Diverse processes	• Globally accessible process policies, design, and coding standards • Collocated high level design phase • Better process visualization through Kanban board • Frequent feedback due to continuous development, testing, and integration of user stories
RE, Design, Coding, Testing	Uncomfortable distributed meetings	• Less distributed meetings due to improved visualization and monitoring through kanban board
	Meager and Outdated documentation	• DUS • Updating of documentation related to user story after code review • Globally accessible repositories
RE, Coding, Testing, Integration	Insufficiently detailed and misinterpreted requirements	• DUS • Collocated high level design phase • Design and code review • Daily scrum and frequent visits • Communication and collaboration tools • Globally accessible repositories and linking artefacts for traceability • Regular feedback through continuous development, testing, and integration of user stories
	Lack of domain knowledge and common understanding about project	• Collocated high level design phase • Design and code review • Globally accessible repositories • Kanban board • Daily Scrum and weekly coordination meetings • Communication and collaboration tools
	Lack of team cohesiveness and distrust among distributed team members	• Cultural and language training • Collocated high level design phase • Kanban board • Frequent visits • Professional accreditation • Communication and collaboration tools • Respect each other culture • Team performance is encouraged
	Inadequate cultural awareness	• Cultural and language training • Collocated high level design phase • Frequent visits • Communication and collaboration tools
RE, Coding, Testing	Different terminologies leads to misinterpretation	• Knowledge repository • Collocated high level design phase • Frequent visits • TPO works as social and technical liaison
	Unawareness related to tasks and	• Kanban board

	responsible persons	• Globally accessible repositories • Linking artefacts to facilitate requirement traceability • Daily Scrum and weekly coordination meetings
	Slow clarification of issues	• Kanban board • Linking artefacts to facilitate requirement traceability • Globally accessible repository • TPO works as social and technical liason • Daily scrum and weekly coordination meetings • Communication and collaboration tools
Design, Coding, Integration	Inappropriate handling of dependencies increases waiting time and frustration.	• Identification of dependencies as soon as possible • DUS • Linking artefacts to facilitate requirement traceability
Design, Testing, Integration	Inadequate interface description	• DUS
RE, Integration	Inexperience related to GSD projects	• Cultural and language training • Collocated high level design phase • Globally accessible repositories • Frequent visits • Daily scrum and weekly coordination meetings • Frequent feedback • Communication and collaboration tools • TPO works as social and technical liason
RE, Design	Slow and inefficient requirement change management	• DUS • TDD • Kanban board • Daily Scrum and weekly coordination meetings
	Insufficient knowledge of rationale behind decisions	• Collocated high level design phase • Daily Scrum and weekly coordination meetings • Globally accessible repositories
RE, Coding	Linguistic difference causes decreased requirement comprehension	• Language training and emphasis on soft skills • Format of DUS • Frequent visits • Written answers to daily scrum questions • TPO works as social and technical liaison
	Differences in work ethics	• Cultural and language training • Collocated high level design phase • Globally accessible repositories • Frequent feedback
RE	Abruptly adding or removing members from project team	• Restricted shrinking and growing of teams • Globally accessible repositories • Daily Scrum and weekly coordination meetings
	Requirement traceability problems	• Linking artefacts to facilitate requirement traceability
	Reduced process transparency	• Better process visualization through kanban board • Daily Scrum and weekly coordination meetings
	Apathy related to distributed review	• Kanban board highlights pending reviews
Coding	Distrust related to vendor developer's proficiency	• Professional accreditation
	Offshore members work on assumptions due to unresolved doubts	• Daily Scrum and weekly coordination meetings • DUS • Collocated high level design phase • Globally accessible repositories • Frequent visits

		• Design review and code review
	Difficult to evaluate performance of distant team members	• Improved task visualization through kanban board
	Over and ad-hoc communication	• Communication medium, frequency and mechanism should be proactively monitored and managed
	Difficulties related to tracking information	• Kanban board • Linking artefacts to facilitate requirement traceability • Daily scrum and weekly coordination meetings • Communication and collaboration tools
Testing	Insufficient unit testing	• TDD
	Difficult to find the developer of the code.	• DUS • Archive Backlog
	Insufficient allocation of time and resources	• Better visualization of bottlenecks through Kanban board • Continuous development and testing of user stories
Integration	Multiple code branches	• Single code branch for all sites
	Teams do not comply to the defined process and architecture	• Continuous monitoring by architects, CPO, and TPOs • Globally accessible repositories • Better process visualization through kanban board
	Incompatible components, insufficient coupling between business strategy and processes, underestimated time and effort.	• Most of the problems related to integration resolve due to continuous integration • Regular feedback through continuous integration • Design review and code review

RE: Requirement Engineering

V. CONCLUSION

Global software development is a paradigm followed by most of the companies whereas agile methods and kanban are prime need of the hour. The proposed process model provides an overview of how combination of XP, scrum and kanban approach can be adapted to distributed projects. The process model follows inspect and adapt cycle through several practices such as design review, TDD, refactoring, code review, updating documentation after code review, kanban board, daily scrum, and weekly coordination meetings. There are various challenges associated with distributed software development which need to be carefully considered throughout SDLC. The proposed process model can mitigate aforesaid challenges associated with GSD. Distributed scrum challenges such as time boxed iterations, painful distributed meetings, and inaccurate estimations are also eliminated due to continuous development approach of kanban.

REFERENCES

[1] J.D. Herbsleb and D. Moitra, “Global software development,” IEEE Software, vol. 18, no. 2, pp. 16-20, March-April, 2001.

[2] E. Ó. Conchúir, P. J. Ågerfalk, H. H. Olsson, and B. Fitzgerald, “Global software development: where are the benefits?,” Commun. ACM, vol. 52, no. 8, pp. 127-131, August 2009.

[3] J.D. Herbsleb and D. Moitra, “Global software development,” IEEE Software, vol. 18, no. 2, pp. 16-20, March-April, 2001.

[4] E. Ó. Conchúir, P. J. Ågerfalk, H. H. Olsson, and B. Fitzgerald, “Global software development: where are the benefits?,” Commun. ACM, vol. 52, no. 8, pp. 127-131, August 2009.

[5] R. Jain and U. Suman,“A systematic literature review on Global software development life cycle,” ACM SIGSOFT Softw. Eng. Notes, vol. 40, no. 2, pp. 1-14, April 2015.

[6] C. Ebert, M. Kuhrmann, and R. Prikladnicki, “Global software engineering: evolution and trends,” in ICGSE 2016: Proceedings of 11th International Conference on Global Software Engineering, Orange County, California, USA, pp. 144-153, 2016.

[7] R. Jain and U. Suman, “Effectiveness of Agile practices in Global software development,” International Journal of Grid and Distributed Computing, vol. 9, no. 10, pp. 231-248, October 2016.

[8] V. Mahnic “Improving software development through combination of Scrum and Kanban” Recent Advances in Computer Engineering, Communications and Information Technology, Espanha, pp. 281-288, 2014.

[9] V. Mandić, M. Oivo, P. Rodríguez, P. Kuvaja, H. Kaikkonen, and B. Turhan, “What is flowing in Lean software development?,” in Abrahamsson P., Oza N. (eds) Lean Enterprise Software and Systems, Lecture Notes in Business Information Processing, vol. 65. Springer, Berlin, Heidelberg, pp. 72-84, 2010.

[10] J. Prochazka, M. Kokott, M. Chmelar, and J. Krchnak, “Keeping the spin -- from idea to cash in 6 weeks: success story of Agile/Lean transformation,” In Proceedings of the 2011 IEEE Sixth International Conference on Global Software Engineering (ICGSE '11), IEEE Computer Society, Washington, DC, USA, pp. 124-130, 2011.

[11] R. Noordeloos, C. Manteli and H. V. Vliet, "From RUP to Scrum in Global software development: a case study," in ICGSE 2012: Proceedings of the IEEE Seventh International Conference on Global Software Engineering, Porto Alegre, Brazil, pp. 31-40, 2012.

[12] M.O. Ahmad, J. Markkula, and M. Oivo, “Kanban in software development: a systematic literature review,” in SEAA 2013: Proceedings of the 39th EUROMICRO Conference on Software Engineering and Advanced Applications, IEEE, Santander, Spain, pp. 9-16, 2013.

[13] I. Richter, F. Raith, and M. Weber, “Problems in Agile Global software engineering projects especially within traditionally organised corporations: [an exploratory semi-structured interview study],” in C3S2E '16: Proceedings of the Ninth International Conference on Computer Science and Software Engineering, ACM, New York, USA, pp. 33-43, 2016.

[14] D.J. Anderson, Kanban: Successful Evolutionary Change for Your Technology Business, Blue Hole Press: Sequim, WA, 2010.

[15] M. Poppendieck and T. Poppendieck, Lean software development: An agile toolkit, Boston, Massachusetts, USA: Addison Wesley, 2003.

[16] N. Tripathi, P. Rodríguez, M.O. Ahmad, and M. Oivo "Scaling Kanban for software development in a multisite organization: challenges and potential solutions,", in Lassenius, C., Dingsøyr, T., Paasivaara, M. (eds) Agile Processes in Software Engineering and Extreme Programming, Lecture Notes in Business Information: Proceedings of the Sixteenth International Conference on XP 2015, Springer, Helsinki, Finland, vol. 212, pp. 178-190, 2015.

[17] M. Ikonen, P. Kettunen, N. Oza, and P. Abrahamsson, "Exploring the sources of waste in Kanban software development projects," in Proceedings of the 2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA '10). IEEE Computer Society, Washington, DC, USA, pp. 376-381, 2010.

[18] M. Korkala and F. Maurer, "Waste identification as the means for improving communication in globally distributed agile software development," J. Syst. Softw., vol. 95, pp. 122-140, 2014.

[19] A. Banijamali, R. Dawadi, M. Ahmad, J. Similä, M. Oivo, and K. Liukkunen, "An empirical study on the impact of scrumban on geographically distributed software development," in MODELSWARD 2016: Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development, pp. 567-577, 2016.

[20] D. I. K. Sjøberg, A. Johnsen and J. Solberg, "Quantifying the effect of using Kanban versus Scrum: a case study," in IEEE Software, vol. 29, no. 5, pp. 47-53, September-October 2012.

[21] C. Ladas, Scrumban - Essays on Kanban Systems for Lean Software Development, Modus Cooperandi Press, 2009.

[22] E. d. Nuevo, M. Piattini and F. J. Pino, "Scrum-based methodology for distributed software development," in Proceedings of the IEEE Sixth International Conference on Global Software Engineering, Helsinki, pp.66-74, 2011.

[23] A. S. Alqahtani, J. D. Moore, D. K. Harrison, and B. M. Wood, "Development framework for distributed Agile software development," International Journal on Advances in Software, vol. 7 no. 1 and 2, 2014.

[24] R. Akbar, M. Haris, and M. Naeem, "Agile framework for globally distributed development environment (the DAD model)," in AIC'08: Proceedings of the 8th conference on Applied informatics and communications, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, pp. 423-428, 2008.

AUTHORS PROFILE

Ritu Jain is a Ph.D. research scholar at School of Computer Science and IT, Devi Ahilya University, Indore, India. She has completed her Master of Computer Application in 2005 and has qualified UGC-NET for lecturership in 2012. Her research interests include Global Software Development and Agile Methods. She has published several research papers in international journals. She is currently working as an associate professor and has 11 years of experience of teaching undergraduate and postgraduate students.

Dr. Ugrasen Suman received his PhD degree in Computer Science from Devi Ahilya University Indore, India. His areas of research are Software Engineering, Information System, Software Reuse, Software Maintenance, & Reengineering, Agile Methods, Software Architectures, Service-Oriented Computing, Knowledge Management and Mining. He has published three book chapters and 80 research papers in National & International Journals/ Conferences. He has authored a book, Software Engineering: Concepts & Practices (Cengage Learing, 2013). Seven PhD Scholars and 43 M.Tech dissertations and several other PG Projects have been awarded under him. He is currently working as a professor at School of Computer Science & Information Technology, Devi Ahilya University, Indore, MP, India.