SindhUniv. Res. Jour. (Sci. Ser.) Vol.51 (01) 31-36 (2019)

http://doi.org/10.26692/sujo/2019.01.07

SINDH UNIVERSITY RESEARCH JOURNAL (SCIENCE SERIES)

Identification of Dependencies in Task Allocation during Distributed Agile Software Development

F. IJAZ, W. ASLAM[++]

Department of Computer Science and IT, The Islamia University of Bahawalpur, Bahawalpur, Pakistan

Received 03rd January 2018 and Revised 10th February 2019

Abstract: There is a growing interest for Distributed Agile Software Development (DASD) in software industry due to multiple benefits such as availability of resources, low development cost, changeability and customer satisfaction. Carrying out DASD poses a challenge of effective task allocation, which requires active coordination between dispersed teams. Effective coordination can be achieved by identifying and managing dependencies in DASD environment. To understand the task allocation complexity due to dispersed teams at different sites, we identify different types of dependencies in DASD. These dependencies must be recognized as they have a profound influence on software product achievement using DASD. Identification and understanding them support management to timely recognize different types of issues that can cause delay, interruptions or even cancellation of Agile Sprints. DASD being an emerging practice, their understanding also leverages methodical task allocation that can optimize on quality concerns such as cost of development and quality of product.

Keywords: Agile Software Development, Distributed Software Development, Distributed Agile Software Development, Task Allocation and Dependencies.

1. INTRODUCTION

Agile methods concentrate on individuals cooperation, respond to changes, produce functional software rather than strictly following recommendations and emphasize on customer satisfaction (Cohn, and Ford, 2003). Distributed Software Development (DSD) supports availability of experts, minimum development cost and time. These advantages have a positive impact on task allocation (Imtiaz and Ikram, 2017).

DSD teams have repeatedly described issues about task coordination. These contain undetected modifications of code and communication breaks, causing cost as well as time overruns that may reduce the promising productivity benefits of DSD work structure. Due to task coordination issues, DSD teams may generally take 2.5 times extra to succeed as compared to collocated teams (Sutanto, *et al.,* 2015). Empirical research demonstrates that proper coordination is required for software project success. The major principle of coordination is based on identification of dependencies and relevant approaches to resolve them. In this regard, Malone and Crowston's theory states 'Coordination is the supervision of all types of dependencies between activities' (Strode, 2016) ".Coordination breakdown is the critical problem. Identifying dependencies that have emerged in Agile Software Development (ASD) can support experts to take applicable coordinative practices

It is necessary to concentrate on initial detailed analysis for the identification of dependencies, assessing the impact of changes and minimizing dependencies among different locations during risk management in Distributed Agile Software Development (DASD) (Sundararajan, *et al.,* 2014). Task dependencies may exist among tasks during a Sprint or across Sprints. Insufficient identification of dependencies during the initial planning is a task dependency challenge at the task level, so is the inability to keenly monitor dependencies with an intention to resolve them. Dependencies extending beyond teams to third party vendors cause many issues such as Sprint interruptions or terminations. They impact the effectiveness of project scheduling as well as estimation, which triggers resizing and rework practices. In software development industry, dependencies on external vendors are a common reality, while these issues need further empirical investigation. It is necessary to minimize task dependencies to facilitate uncomplicated and earlier development. Dependencies can be tackled through Agile practices, for example daily Scrum, planning and review meetings related to each Sprint. Estimation as well as Sprint planning formed by long and short term visions can reduce dependencies contained in features (Hoda, and Murugesan, 2016). With such problems in focus, we aim to bring to limelight the intricacies of dependencies arising due to a DASD setup. The research question addressed in this paper is: 'Which dependencies affect task allocation in DASD?' According to the importance of dependencies in DASD, we have identified different types of dependencies and made their taxonomy. This paper is arranged in number of sections as follows.

[++]Corresponding author's email: waqar.aslam@iub.edu.pk

Related Work is explored in Section 2, while dependencies are identified and their taxonomy built in Section 3. Objective Optimization based dependencies are given in Section 4 and conclusions in Section 5.

2. RELATED WORK

Previous research work has pointed out that client stated software faults are frequently due to dishonored dependencies these are not identified by software development team. Results show that logical and work dependencies are significant, impacting the possibility of defects exposure in source files (Cataldo, *et al.,* 2009) For recognizing the possible implications of changes or the estimation of which requirements must be adapted to realize changes, it is impossible to understand these changes in software systems without understanding dependencies. Dependencies between software objects often causes further modifications in software (Souza, and Redmiles, 2008).

According to the results of a systematic literature review, absence of obvious component or project ownership and components interdependencies are strong challenges for component based distributed software development. A components dependency is the second highest challenge as different components are generally developed independently at different locations, while integration of these components requires coordination to complete system requirements (Mahmood, *et al.,* 2015). The development team responsible for developing dependent components should engage in more communication than developing independent components. This is due to the fact that there is a solid relationship between components interdependencies and the frequency of required communication between developers implementing those components. (Fonseca, *et al.,* 2006). According to the literature analysis, various coordination issues are identified in DSD. Like other issues, dependencies related negatively impact the software quality attributes (Suali, *et al.,* 2017).

During software development, coordination interruptions increase the number of defects and costs. According to a study, interpersonal coordination activities can effectively manage highly interdependent tasks; however, in DSD these types of informal coordination activities are expensive. Identification of dependencies and the coordination requirements, shared by reasonable activities can decrease software defects.

In reality, of all enterprises surveyed, 81% have reported that they had coordination problems due to DSD, and many of them had to re-architect their software to minimize dependencies between DSD team members (Sutanto, *et al.,* 2015).. Identification of factors and dependencies in DASD is increasingly a focus of research community (Strode, 2016)

3. DEPENDENCIES

Task dependencies indicate the condition that achievement of specific task is essential to initiate next task, or the expansion of particular action depends on the existence of a particular item, where the item can be an artifact, a person or any material(Strode, 2016). Identification and supervision of dependencies in DSD are critical activities affecting productivity and software quality (Cataldo, and Herbsleb, 2013). There are diverse categories of coordination issues in DSD, and these issues are influenced by the nature of concerned dependencies. Technical, temporal, and process are coordination issues in DSD. Technical coordination problems appear when technical dependencies between chunks of software are not efficiently addressed. For example, redundant code, incompatible interfaces and integration problems appear due to lack of technical coordination. Temporal coordination problems occurdue to lack of effective management of time dependencies. For instance, when software activities or software pieces are not finalized according to project schedule, they affect completion of others work activities. For example, testing cannot start prior to completion of coding stage. Process coordination problems happen due to lack of monitoring the dependencies in the software development process. For example, when software process is not followed, development work is initiated before the designing activity certified or conflicts about priorities fixed. (Espinosa, *et al.,* 2007). Taxonomy is defined as ‘Categorization of systems that classify phenomena in absolute or complete groups with a sequence of distinct decisions instructions. Quality taxonomy should be concise, robust, comprehensive, extendible, explanatory and useful. **(Fig. 1)** presents dependency taxonomy, which represents basic dependencies, task interdependencies, software dependencies, Agile dependencies and distributed environment dependencies.

3.1 Basic Dependencies

Here three basic types of dependencies are defined: flow, fit and sharing.

Flow dependency: A condition in which specific activity provides output that is required for other activity. For example, design specifications are produced by designer that is required by developer.

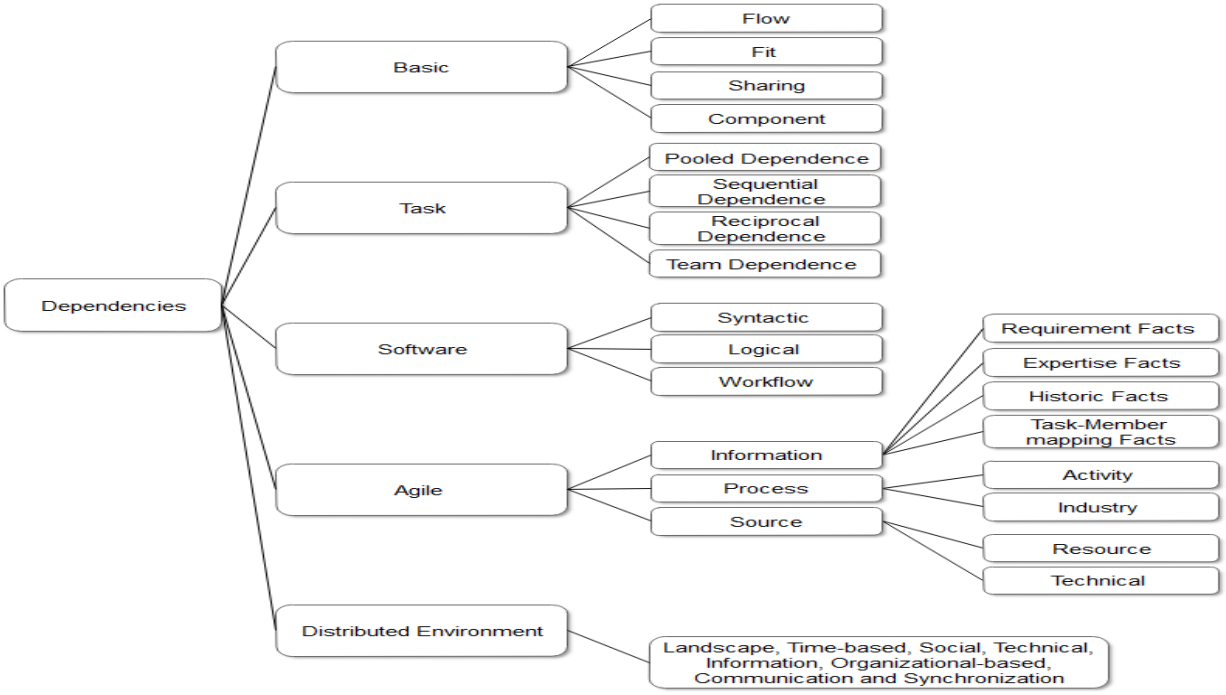

Fig. 1: A taxonomy of dependencies in DASD.

Fit dependency: When outputs are provided by numerous activities and these have to fit together, for instance, during the integration phase, when various components are fit together.

Sharing dependency: When a specific resource is required by compound activities, such as, time of an expert technical architect.

Component dependency: Software components having extreme level of dependencies should be managed wisely due to their stronger impact on other components in the architecture. Dependencies of software components influence development team activities.

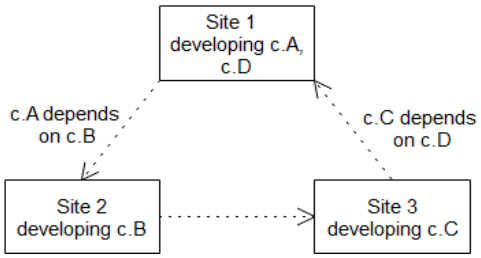

Fig. 2: Component dependency.

Component dependency is presented in **(Fig 2),** wherein components A (c.A) and D (c.D) are developed at site 1, component B (c.B) at site 2 and component C (c.C) at site 3. Component A is dependent on component B, which is itself dependent on component C, whereas component C is dependent on component D. These type of dependencies between components causes communication overhead due to the involvement of distributed sites.

3.2 Task Interdependencies

Four types of task interdependencies are recognized in this category.

Pooled dependence task: Each development team member finalizes work on his own before tasks are aggregated. For instance, individual team members code the software components independently prior tothe integration of each software component.

Sequential dependence task: Individually team members have to complete their own work prior to delivering it to the next member. For instance, during the development, team members design and create test cases relevant to their specific responsibilities earlier to delivering these test cases to the succeeding development team members who do actual tests.

Reciprocal dependence task: The work passes from side to side between development team members. For instance, throughout the debugging process, work flows between Programmers and Testers back and forth, thus creating this type of dependency.

Team dependence task: Normally all team members work simultaneously to identify the issues and offer solutions. For example, during software requirement analysis and making set of software requirements, all team members have to understand and suggest requirements of users.

3.3 Software Dependencies

Three types of dependencies are recognized due to software artifacts.

Syntactic dependency: It represents explicit associations among source files that are expressed by data and functional syntactic dependencies. Their strength can be evaluated by the number of data and function/method references that span from a source file towards another source file ($x$to y), denoted asS_{xy}.

Logical dependency: It represents semantic or indirect associations among source files and obvious relationships. These dependencies create an association between source files that are reformed together according to a modification requirement(s) by one or more development team members.

Work flow dependency: It represents definite relationships between development team members influenced by workflows and processes.

3.4 Agile Dependencies

To support ASD, previous research presents information, process and source dependencies, with information dependencies having stronger impact than others.

Information dependency: It includes facts about requirement, expertise, history and task-member mapping dependencies.

Requirements facts dependency: Correct information about requirements is significant, while lack of this information has negative impact on project progress.
Expertise facts dependency: A condition in which technical or task related information is acknowledged by a specific team member or a group of team members. In software development, there are three main categories of expertise: technical, design, and domain.

Historic facts dependency: A condition in which information related to previous decisions is required and absence of this type of information influences project success.

Task-Member mapping facts dependency: A situation in which tasks assigned to team members must be well known. It has crucial role in project progress.
Process dependency: It includes activity and industry process dependencies.
Activity dependency: It is a dependency relationship in which progress of an activity is not possible unless other activity is finished.

Industry process dependency: It is a dependency relationship in which current business process becomes a reason to complete tasks in a specific order.

Source dependency: It includes object and technical dependencies.

Resource dependency: A state in which a resource (individual, location, or thing) when required, must be accessible, otherwise resource temporary postponement or even deadlock may occur. An individual may be a developer, a programmer or a tester.

Technical dependency: A state when a technical characteristic of software development influences the progress, for example, when some component needs to cooperate with other component. Its existence or deficiency has an effect on project progress..

3.5 Distributed Environment Dependencies

Distributed locations give rise to landscape, time, social, technical, information, organizational, communication and synchronization based dependencies. These dependencies when considered with other related factors, complexity during task allocation is increased (Lamersdorf, *et al.*, 2009). **(Fig. 3)** presents distributed environment dependencies. It shows four sites: 1, 2, 3 and 4, which depend on one another in some sense, reflecting Distributed Environment Dependencies (DEDs).

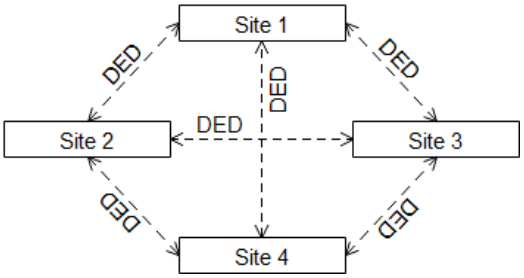

Fig. 3: Distributed Environment Dependencies.

These identified dependencies should be considered in task allocation method especially in DSD, because neglecting these dependencies negatively affects the schedule of project.

Recognizing dependencies in software development process support task allocation, especially when an unambiguous task to team member mapping is sought. **(Fig. 4)** presents different types of dependencies, such as sequential dependency (SD), activity dependency (AD), fit dependency (FT), resource dependency (RD), team dependence tasks (TDT) and reciprocal dependence tasks (RDT). According to Agile paradigm, software development process has Product Backlog containing user story (US) list and Sprint Backlog containing task list. After selection of USs for Sprint, software development phases such as analysis, designing, coding, testing and integration have SD and AD. Analysis phase has TDT depending on all DT for maintaining Sprint Backlog. Integration phase, including SD and AD, has FT. RDT affects coding and

testing phases with two way forward and backward dependencies. All tasks for completion also depend on entity and knowledge according to their requirements, which represent ED and KD. **(Fig. 4)** also shows the influence of dependencies on various phases of software development process and relationships of team members as affected by these dependencies. During task allocation in DASD, these dependencies should be considered to decrease complexity due to neglecting their deep effect.

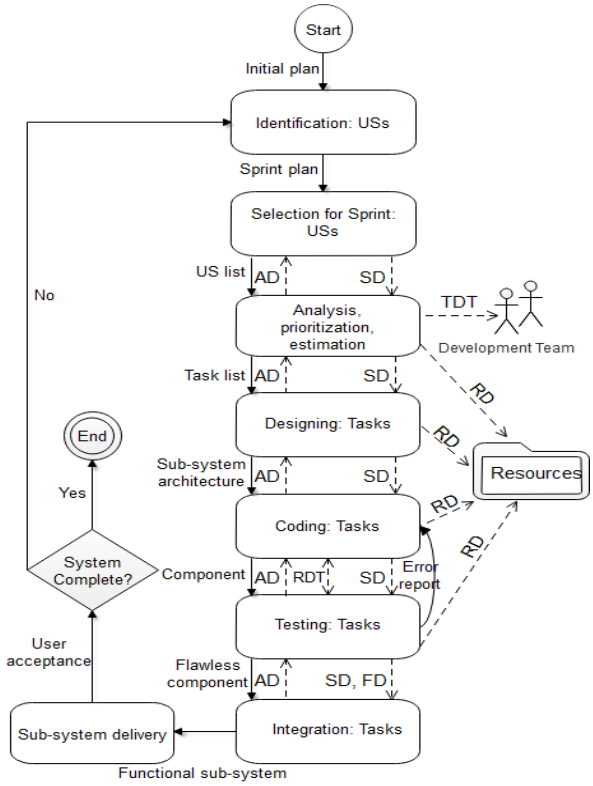

Fig. 4: Different dependencies identified during software development process.

4. OBJECTIVE OPTIMIZATION BASED DEPENDENCIES

Objectives of the development organization also affect dependencies in DSD. We picked idea from decision model for task allocation) (Lamersdorf, *et al.*, 2009)., and used this concept to define dependencies impact changing due to changes in the development organization goals. The tasks of the project are related to different activities such as requirements engineering (RE), design, implementation and integration. For example, project development may be distributed at three sites: Pakistan, Singapore and United States of America (USA). Development at USA is expensive but offers excellent skills in RE and design. The Singapore site is also expensive but less than USA, offering good implementation and integration capabilities. Pakistan site has countless differences with two other sites, especially in language and culture. Still development is quite inexpensive and offers good implementation skills, though lack in RE and design experiences. We give pointers to quality and cost concerns when considered as development objectives.

Quality Objective: Software quality is defined as 'The degree to which a software artifacten counters established requirements, although, quality determined by the degree to which those established requirements correctly show customer needs as well as expectations' (adapted from ISO/IEC/IEEE 24765:2010) ("IEEE Standard for Software Quality Assurance Processes - Redline," 2014). If an organization aims at good quality, RE and designing should be completed in USA, implementation in Pakistan and integration in Singapore. This condition is represented in **(Fig. 5),** which shows dependencies between sites and development phases.

Fig. 5: Task assignment dependencies with aim on quality.

Cost Objective: According to DSD, software cost assessment should include software development cost, software maintenance cost and re-engineering cost. There are four categories of cost factors such as product, platform, personnel and project. These factors impact DSD and are associated to different characteristics of distributed projects such as spoken, social and time dissimilarities (Bajta, *et al.*, 2015). If an organization aims at low cost, all the activities except R A, ought to be completed in Pakistan. RE preferably needs to be completed in USA. This condition is represented in **(Fig. 6).**

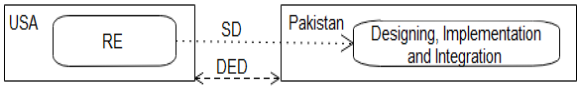

Fig. 6: Task assignment dependencies with aim on cost.

When software development objective changes from quality to cost **(compare Figs 5 and 6),** dependencies change accordingly. For instance, there is less number of locations in Figure 6 than Figure 5, due to which there is no FD in **(Fig. 6).**

Finally it is commented that if an organization aims at saving time, then all the tasks should be assigned to one location. It will reduce communication overhead due to distributed environment dependencies. When task allocation is done objectively, dependencies in task allocation strategy change accordingly.

5. CONCLUSION

In this paper, we have identified and explored dependencies in DASD and their taxonomy proposed. This taxonomy presents basic information about different types of dependencies such as basic and those related to software, task-task mutual relations, Agile process and distributed environment. We have also presented the relationship of these dependencies during various phases of software development process. Different types of dependencies impact the progress of software development process with varying degree of impact. We have also identified objective optimization based dependencies. They describe and point out that DASD based project objectives change the intensity of dependencies among distributed team members. Identification of these dependencies in DASD contributes to better understanding of required coordination among distributed team members that must be recognized during task allocation in DASD. Future work can be towards a dependency model that reflects the level of impact of these dependencies according to their weights in DASD.

REFERENCES:

Aslam, W. and F. Ijaz (2018) "A Quantitative Framework for Task Allocation in Distributed Agile Software Development," IEEE Access, vol. 6, 15380-15390.

Bajta, M. E., A. Idri, J. L. Fernández-Alemán, J. N. Ros, and A. Toval, (2015) "Software cost estimation for global software development a systematic map and review study," in International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), 197-206.

Cataldo, M. and J. D. Herbsleb, (2013) "Coordination Breakdowns and Their Impact on Development Productivity and Software Failures," *IEEE Transactions on Software Engineering*, vol. 39, no. 3, 343-360.

Cohn, M. and D. Ford, (2003) "Introducing an agile process to an organization [software development]," *Computer*, vol. 36, no. 6, 74-78.

Cataldo, M.., A. Mockus, and J. D. Herbsleb, (2009) "Software Dependencies, Work Dependencies, and Their Impact on Failures," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, 864-878

Espinosa, J. A., S. A. Slaughter, R. E. Kraut, and J. D. Herbsleb, (2007) "Team Knowledge and Coordination in Geographically Distributed Software Development," *Journal of Management Information Systems*, vol. 24, no. 1, 135-169.

Fonseca, S. B., C. R. B. D. Souza, and D. F. Redmiles, (2006) "Exploring the Relationship between Dependencies and Coordination to Support Global Software Development Projects," in 2006 IEEE International Conference on Global Software Engineering (ICGSE'06), 243-243.

Hoda, R. and L. K. Murugesan, (2016) "Multi-level agile project management challenges: A self-organizing team perspective," *Journal of Systems and Software*, vol. 117, no. Supplement C, 245-257.

Imtiaz, S. and N. Ikram, (2017) "Dynamics of task allocation in global software development," *Journal of Software: Evolution and Process*, vol. 29, no. 1, pp. e1832.

IEEE Standard for Software Quality Assurance Processes - Redline, (2014) *IEEE Std 730-2014 (Revision of IEEE Std 730-2002) - Redline*, 1-231.

Lamersdorf, A., J. Munch, and D. Rombach, (2009) "A Survey on the State of the Practice in Distributed Software Development: Criteria for Task Allocation," in 2009 Fourth IEEE International Conference on Global Software Engineering, 41-50.

Lamersdorf, A., J. Münch, and D. Rombach, (2009) "A decision model for supporting task allocation processes in global software development," *Product-Focused Software Process Improvement*, 332-346

Mahmood, S., M. Niazi, and A. Hussain, (2015) "Identifying the challenges for managing component-based development in global software development: Preliminary results," in 2015 Science and Information Conference (SAI), 2015, 933-938.

Sutanto, J., A. Kankanhalli, and B. C. Y. Tan, (2015) "Investigating Task Coordination in Globally Dispersed Teams: A Structural Contingency Perspective," *ACM Trans. Manage. Inf. Syst.*, vol. 6, no. 2, 1-31.

Strode, D. E (2016) "A dependency taxonomy for agile software development projects," *Information Systems Frontiers*, vol. 18, no. 1, 23-46,

Sundararajan, S., M. Bhasi, and P. K. Vijayaraghavan, (2014) "Case study on risk management practice in large offshore-outsourced Agile software projects," *IET Software*, vol. 8, no. 6, 245-257.

Souza, C. R. B. d. and D. F. Redmiles, (2008) "An empirical study of software developers' management of dependencies and changes," in Proceedings of the 30th international conference on Software engineering, Leipzig, Germany, 241-250.

Suali, A. J., S. S. M. Fauzi, and M. H. N. M. Nasir, (2017) "Developers' coordination issues and its impact on software quality: A systematic review," in 2017 3rd International Conference on Science in Information Technology (ICSITech), 659-663.