## Chapter 4: Case 1 Analysis and Findings

This chapter presents the findings and analysis of case 1. The purpose is to analyse the coordination dependencies and mechanisms that were used to coordinate the software development process using a distributed agile development approach. .

This chapter presents the analysis of Case 1 using the following structure: background and context of the project (section 4.1), contextual analysis of the project (section 4.2) (e.g. team structure, team organisation to signpost the distributed segments of the project), overview of the software development process which is the baseline of the coordination analysis (4.3), coordination analysis of the software development process Social Network Analysis (Ref) and delay analysis (Ref) (4.4 & 4.5.), and analysis of the coordination mechanisms (section 4.6). The coordination complexities and challenges are presented in section 4.7 which is followed by the analysis of the collaborative technologies in section 4.8.

The outcomes presented in this chapter, based on the analysis of Case 1, reports the understanding of the coordination process of the researcher and also provides a basis for the cross-case analysis provided in later section.

### Project Context

The first case code-named as ‘Pigeon’ is one of the key projects for a US-based multinational company which focuses on developing software systems for managing health informatics. At the time of data collection, the mother company had just been merged with another big healthcare service provider for better market acquisition. As part of their projects, the organisation collects and stores complex healthcare related data into the cloud database, analyses those data using their key product ‘Pigeon’, and delivers unique and valuable insights into diseases, treatments, costs and outcomes to their clients. The organisation is globally spread around 100 countries blending 15,000 employees to help clients run their operations more efficiently.

The case Pigeon is a matured product that has been used by the customers for a long time. Currently, the development process includes new feature enhancements, fixing bugs along with customer support and maintenance related issues. The product has a front-end service that utilises the strong backend infrastructure to provide critical insights regarding healthcare related services. This product can be titled as highly critical from both the organisational and client’s perspective. As this is the key software that drives the organisation’s business, enables it to provide end-to-end service to their clients. On the other hand, the clients depend on this solution for their day-to-day services to offer critical insights from accurate data and predictive analysis.

The product Pigeon mainly has two main streams of work, Product enhancements and Product Bug Fixing and support ~~known as~~ *~~Firefighting~~*.. Teams are distributed in NZ, USA and Europe and each team is specialised in special parts of the overall product. There are total 8 teams involved in the product’s work streams and teams are distributed in NZ, USA, Europe and Asia. Four teams are located in NZ, code-named as Team-R, Team-FT, Team-RT, and Team-M. Two teams are USA based, code-named as Team-FW and Team-C. Quality Assurance (QA) and Client Support teams are distributed in different parts of Europe and Asia. But we have collected data about most of the teams involved and the data contains information about their team structure and work specialisations. These data help us to get the clear understanding about the teams and their contributions in the project overall.

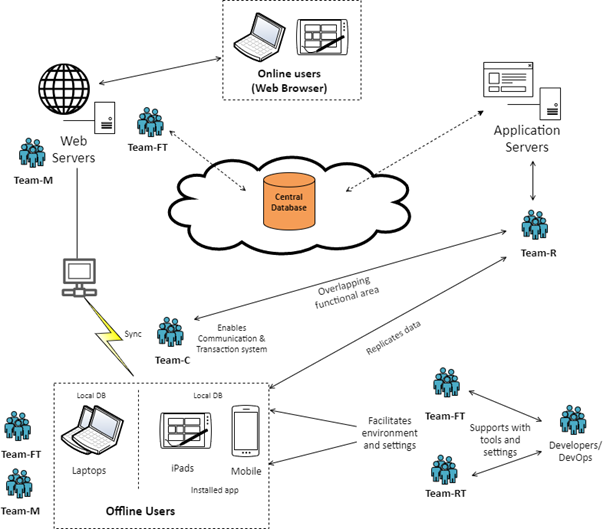


Figure 4.1: System Architecture of the Product

All the interviews of this research are from Team-R that is specialised in replication services. Replication service is one of the backend services of the product that figures out which bit of data needs to go to which remote machines like tablets or iPads. This service ensures the optimised data replication facility so that only the necessary amount of data is replicated or copied to the user’s machine to avoid copying the whole database at a time to save device space and bandwidth. Team-R is located in NZ and has 7 members: 1 Development Manager, 4 developers, 1 tester and 1 Scrum Master (SM). The tester is a regular team member who performs the acceptance tests for all the development works before submitted to the Product or integration tester. The Product/Integration testers are referred as QA by all the participants, so we will use the same term to be consistent. All the QA members are *global* and located in India and Japan. Initially all the teams shared the QA members, but recently, they have assigned specific QA member for each team for better performance and QA experience.

Team-C, a USA based team, is responsible for ensuring the connection establishment and transactions of data within the product features. There are 3 members in this team consists of a team lead, 1 developer and 1 integration tester and all of them are working from 3 different places in USA. This team shares the Scrum Master (SM) with the Team-R because of their overlapping functional area. Team-R and Team-C needs to collaborate with each other for establishing the connection before starting the data replication and properly transferring the data from database to user’s device. Due to this overlapping functional area, intense coordination activities are required to successfully complete their tasks.

Team-FT, is a NZ based team, responsible for managing the admin tools, provides all environmental settings (i.e. services, clusters, the database), and also works as a representative for the client teams to setup for their tenants, basically the configuration settings they might have for the apps, e.g. adding users, configure how the user are able to interact and security settings etc. They also develop the UI for the front-end services to initiate the back-end jobs.

Team-RT, is also a NZ based team, works with all sorts of build tools, development and DevOps tools, configuration management (i.e. how does install, how to authenticate etc.). Team-M is specialised in integrating the web servers for offline and online services which is based in NZ. For understanding the relationships between teams and their functional areas in the product, a diagram of the system architecture is presented in Figure 4.1.

A close up of a map

Description automatically generated

Figure 4.2: Team Organisation including their locations

All Product Owners (PO) for this product are located in different parts of Europe, and there is a liaison person in NZ who acts as Proxy PO. All the team’s organisations and their structure, including the product owners**,** are presented in Figure 4.2. In terms of stakeholders, Global POs, clients and Proxy PO (local) are primary stakeholders for product requirements. and other teams (e.g. replication, deployment, client support) and QAs are the stakeholders for feature development and bug fixing. This project context indicates several global touchpoints among different stakeholders in their day-to-day work and to better understand their relationships, we have performed contextual analysis that is discussed in the next section.

### Contextual Analysis

(Discuss about the distribution and agile approach implemented in the organisation)

### Overview of the Software Development Process

Pigeon development is subdivided into two main streams, those are Product Feature Enhancement and Bug fixing and support. All software development works are focused in these two streams. All the participants of this study are from Team-R and majority of their software development activities are concentrated in the backend service design and development. While performing their development activities, team members need to exchange information, acquire knowledges, ask for decisions, negotiate their work items, consult and communicate to have a shared understanding of the requirements and so on. In order to fulfil these needs, members of Team-R have to coordinate with their peer team members and, at the same time, external team members which involves globally distributed members of other teams. All the participants mentioned that due to the extensive overlapping of product functional features, they have to rely on Team-C and as a result, a lot of communication and synchronisation efforts are performed to coordinate their activities. In the next paragraphs, I will discuss ~~about~~ the software development processes followed in each of the work streams. This ~~discussion~~ will be supportive to point out the interdependencies and explanation of the analysis findings in late sections. A diagram showing all the top-level activities in the two work streams are given in Figure 4.3.

#### Product Enhancement Process

In the product or feature enhancement process, the primary source of new ideas are the product owners. The POs are the globally distributed in parts of Europe and the principal one is from France. There is a liaison person working as Product Manager (PM) located in NZ branch who acts as Proxy Product Owner. The proxy PO is also one of the important sources of product enhancement ideas. Besides, any other teams involved in the Pigeon development, can come up with any enhancement idea and after approval can be included in the release backlog. All the ideas are recorded in the requirements tracking software, i.e. ‘*JIRA*’ for further decision making. It is noteworthy to mention that new feature or enhancement ideas regarded as requirement can also be resulting product of the Bug categorisation.

A screenshot of a cell phone

Description automatically generated

Figure 4.3: Overview of the Software Development Process

All the requirements listed in JIRA goes through the analysis and decision-making process where all the POs (both local and global) and development teams participate and finalise whether to accept of discard any idea. During this analysis, all the required expertise and high-level dependencies are identified to distribute the tasks to the teams of matching expertise and involve them in the requirement prioritisation process. Accepted requirements are then prioritised in a session termed as ‘*Backlog grooming*’ by the PM to be included in the current release or queued for the future releases. The Development Managers (DM) from each team are involved throughout these processes to have a mutual understanding of the release requirements. The DMs are now responsible for continuing the further development process along with the Scrum Master (SM) and other team members to meet the release deadlines. The product enhancement releases are scheduled around every 4 months and it includes all the sprint releases completed within that time. The new enhancement requirements are assigned to a particular team, after that the team performs impact analysis of enhancement. This analysis may result in involvement of other specialised teams impacted, for example, Team-R would be involved in any new enhancement that has impact on what needs to be replicated.

Release requirements are put into Jira as product enhancement user stories under the Product backlog items and the DM is responsible for writing the User stories and breaking them into tasks. The development team members then participate in the ‘*planning poker*’ session, sometimes part of the ‘*backlog refinement*’ process. Team members play for each story to estimate the tasks and further decides that tasks that will be included in the current sprint in the ‘*sprint planning*’ session. Sprint planning sessions are separate meetings, but occasionally they are in the same session. The SM coordinates with the team during the sprint planning session and finalises the points to be included and then members self-assign tasks and do break down. The DM participates in the task break down session and answers any questions to clarify the stories as much detail as possible. The DM has got significant amount of technical and domain knowledge, so he also discusses about probable solution ideas to the developers that helps them to pick it up. At the same time, the SM figures out the duration of the sprint based on the items in the backlog, usually it is either 2- or 3-weeks sprint. SM is the person to coordinate the tasks in the sprint backlog items and the task assignment during the sprint planning.

All the tasks and their allocations are inserted into the ‘*Squad board*’ in Jira and through this system and it has Pending, In progress, Code review and Ready for test columns. Through this squad board, the task allocations and sprint progress are visible to all members. The team also used ‘*Slack*’ tool to report their daily work updates at the day end and during the daily standup, each member just reads off what they logged in slack. This approach helps team members to just join the meeting without any prior preparation and to avoid missing any items to be discussed. If there is any item requires a bit of discussion, they just take that offline after the stand up. Most of the time the DM continues the longer discussion with other team members to talk over the issues, examine their issues, respond to their queries and if necessary, help them out to solve the issues. Whenever any task is finished development, it is put in a queue for the local tester to run acceptance tests.

The team has got a local tester to perform acceptance tests and participates in all the software development activities. Tester is a part of the planning and he attends all the meetings, such as sprint planning, backlog refinement, daily standup, sprint review and retrospectives, code reviews. He can see what is being discussed in those meetings and develop understanding about what is the requirement, what is the solution. Based on this understanding, he can start thinking about the possible test cases, prepare a test plan. Typically, the test plans are creating using graphics as it is easy to understand, and test coverage items are reported as bulletpoints rather than wordy ones. In the Team-R, the tester is a crucial part of the team and acts similar to any other developers. The DM mentions that “*the tester works much like a developer, he will go and construct the test cases, and he would do some kind of review, and discuss to get feedback from developers and manager*.”

The team conducts regular ‘*test case review*’ sessions to give feedback about the test cases that indicates whether the tester is on the right track, or there is any misunderstanding about the requirements. Whenever, the tester identifies any issue, he reports to the DM and gets confirmation whether it is a real bug or not. The tester also attends the developers ‘*code review*’ sessions and he is also part of that session. One of the benefits of attending this session is that the tester can see what has been tested in the developer’s end, so that he can skip those cases to save times. The tester maintains separate ‘*Squad board*’ for testing activities which shows the progress of testing activities and their status. Despite of working together, the testing activities does not exactly follow the 2- or 3-weeks sprint. All the completed stories of the sprint come into the testing squad board and the tester starts testing them.

The tester’s squad board also has Pending, Test in progress, Test completed columns. At the end of the testing process, if the testing failed, the tester consults with the DM and if he decides that is a bug, then it is logged as Bug in the Jira. The tester also discusses with the developer to have a consensus about the bug and the bug is listed for the next sprint to be fixed. If the test is passed, the tester marks the ticket as complete and forwarded to the QA team with comments and related documents. Recently, the Team-R has started doing ‘*Demo*’ for the QA team and the purpose of this demo is to show how the solution works in the system. The tester mentioned that, "*We just started to do a demo for the QA team. Typically one of the functionalities how this piece of ticket works. But we don't really show the QA what we tested, and we don't show the code base or anything. Just how it works.*"

The QA team members perform integration testing to check the solution is working in the final product before it goes to production team. If the QA team has any query regarding the ticket or they face any problem, they communicate with the tester in general, but if required, they directly communicate with the DM or developers. The QA members can access the tester’s squad board and see the ticket details and comments specific for QA. To synchronize with the QA teams, they conduct a scheduled fortnightly ‘*QA meeting*’ with the QA leads where all the testers and all the team members from the Team-R, C, RT and FT attends. After the approval from the QA teams, stories are queued for the release and production team members are involved to integrate the features in the live system.

#### Bug fixing (Firefighting) Process

The bug fixing or firefighting activities adds a significant amount of workload throughout the year and it is kind of a continuous process that involves multiple teams. Thought there is no particular metrics to measure the amount of firefighting, but the DM of the Team-R mentions that it requires approximately 50% of their work efforts. Each team working in the product has to perform firefighting activities, but the firefighting process, i.e. how they handle all bug fixing activities is different across teams. For example, Team-C integrates all the firefighting request with the new enhancement workflow to some extend with a few good troubleshooters handling them. On the other hand, in case of Team-R (i.e. our case team), the DM is doing most of the firefighting work so as not to interrupt development team’s flow of work. Because he has good product knowledge and development experience, he handles all those requests along with one or two developers who are assigned intermittently. The Bug fixing/support process goes through several key phases, those are: *Bug identification or creation*, *reporting, reviewing, Categorising, Requirement processing, task allocation, development, local testing, QA* and finally included in the new release (see figure 4.4)

The bug fixing process starts by *identifying* any potential bug by any of the teams involved. In most cases, the bugs are identified by internal tester or the QA team who is working offsite. Other parties, such as the support team or customers can also report any bug through different channels. Once the bug is identified, they need to be *reported* to the development team using proper channel. If the bug is identified by the tester or the QA team, they need to report by creating a JIRA ticket with details of the problem they are facing. In case of the support team or clients, then they have to use a ‘*SMART tool*’ developed in-house by the organisation. Another communication channel that is popular in this case is email and any of the parties can directly report any bug by sending email to any predefined or known point of contact, such as DM or any specific developer.

A screenshot of a cell phone

Description automatically generated

Figure 4.4: Bug Fixing activities flow

Whenever any bug is reported through any of the channels, usually the DM of the Team-R *reviews* the bugs before it goes to the development team. As mentioned earlier, DM takes care of the firefighting activities, he investigates the reported bugs to confirm the authenticity of the reported bug. Afterwards, the request goes through the categorising process to measure the importance of the issue based on the user impacts. Based on the impact analysis, *Categorising* process might involve POs (both local and Global), PO Proxy (local), SM and development teams if the bug may lead to product enhancement. In such cases, the rest of the process follows the steps mentioned earlier in terms of product enhancement.

Alternatively, the adhoc support cases are handled by the firefighting team lead by DM and he will either himself will work for the solution or assign any fellow developers to solve that. The *development* process starts straight after the task allocation as the sprint progress. As soon as the developers complete any ticket, the tester in the team starts the *testing process.* Whenever both the tester and the developers are confident about the solution, subsequently, the bug/support ticket goes to the external QA team for further *verification*. If the QA accepts the solution, it is included in the *next release*, otherwise goes through further development and testing process.

The overall bug fixing process also goes through certain process management activities which is particularly performed by both the Scrum Masters and the Development Manager. Process management activities includes decision making, impact analysis based on development team dependencies, task synchronization between teams (both local and global), progress tracking and sharing and other management activities. These activities play a vital role in the case of coordinating the entire bug fixing process effectively.

Taken together, both the work streams consist a number of activities that involves people working from local and distributed sites. While performing these activities, they need to synchronise their works because of the input-output dependency (Malone & Crowston, 1994), exchange knowledge (Razzak & Ahmed, 2014) and expertise (Faraj & Sproull, 2000) which requires appropriate coordination mechanisms to achieve effective coordinated state (Espinosa et al., 2002; Strode et al., 2011). In the next section, I will report about the analysis findings of the above mentioned work process to understand their interdependencies, applied mechanisms and their effectiveness.

### Analysis of Coordination process using SNA

### Impact analysis using Delay diagram

The shared message may not be received as it was intended due to communication challenges in the GDAD environment (P. J. Hinds and M. Mortensen, "Understanding conflict in geographically distributed teams: The moderating effects of shared identity shared context and spontaneous communication", Org. Sci., vol. 16, no. 3, pp. 290-307, Jun. 2005.).

The higher communication effectiveness comes at the price of a considerably longer time of communication. The short message may be insufficient to deliver clear information. Low communication efficiency can negatively impact communication effectiveness by increasing delays that impede the rapid development of shared understanding. Moreover, if the message is complex, with large amounts of information or a high diversity of information, a team member will require more time to assess and deliberate on the information [48], [90]. Communication efficiency may impair the development of understanding because members will not have the time required to fully process the information, which may cause a greater cognitive load on the individual [48], [90] and encourage premature action or speculated decision or innovation in decision making [48]. (Alzoubi & Gill, 2020)

**48.**Dennis, Fuller and Valacich, "Media tasks and communication processes: A theory of media synchronicity", MIS Quart., vol. 32, no. 3, pp. 575, 2008.

**90.**D. Te’eni, "Review: A cognitive-affective model of organizational communication for designing IT", MIS Quart., vol. 25, no. 2, pp. 251, Jun. 2001.

### Analysis of Coordination Mechanisms (what went well and what not)

### Coordination Challenges in the project

### Analysis of Collaborative Technologies

### Chapter Summary

## Discussion (Work in Progress)

(Discussion about coordination strategy to ‘coordinating’ process, i.e. instead of pre-defined mechanisms, appropriate mechanisms are adopted)

In Case 01, initially the RS team used to initiate and engage with CS team whenever they are blocked in their development work. This mechanism requires a certain amount of time to coordinate as either side does not have any idea what other team is doing. They have to first develop a shared understanding what other teams are doing and what is the blocking point and then they can start taking the actions to solve it. This type of mechanisms came out to be time consuming and adding delays in the process which sometimes blocked the ongoing release.

The team has identified that these mechanisms are not effective, and they have started to be pro-active for the last 2 releases to make if effective. In the starting of any release, the team started listing down the tasks that either needs other teams’ involvement or impact other teams work. They start communicating with those teams proactively to discuss the possible inter-dependencies, and create a plan by setting up point of contact, scheduling regular meetings to update each other’s progress etc. The SM mentioned that, “*We list a thing that okay that might affect us, so we contact them proactively and ask there is problem. Before we don't know what other team were doing, until they have been blocked by us. it takes a while to figure out the problem, to then introduced the effects. A list can tell for ourselves we think we need to talk to these three teams because they may affect us. So we send emails right now and find out. and then we plan..*”

The mechanisms seem to be effective as all the dependent teams know what other teams are doing, how they may impact them, whom to contact and thus develop a shared understanding. This is how the team changed their mechanisms to make it effective and thus the process of ‘coordinating’ works.

### Coordination Mechanisms effectiveness (Work in Progress)

We have observed that regular scheduled meetings are proved to be effective while coordination between local and remote teams. Emails

### Coordination Challenges (Work in Progress)

The key challenges, identified in the case, are the time difference and proper division of responsibility/ownership of user stories so that one team can take the responsibility to solve any bug or delivering a story. As multiple teams are working in the same story or bug, it requires proper coordination among teams to complete the story or fix the bug. Additionally, the process includes globally distributed QA, and integration testers on both sides which adds complexity in effective coordinating the issues to avoid any blocker or delay.

***References***

Alzoubi, Y. I., & Gill, A. Q. (2020). An Empirical Investigation of Geographically Distributed Agile Development: The Agile Enterprise Architecture is a Communication Enabler. *IEEE Access*, *8*, 80269–80289. https://doi.org/10.1109/ACCESS.2020.2990389

Espinosa, J. A., Lerch, J. F., & Kraut, R. E. (2002). Explicit vs. Implicit Coordination Mechanisms and Task Dependencies: One Size Does Not Fit All. *Process and Performance at the Inter- and Intra-Individual Level*, *January*, 39. https://doi.org/10.1.1.59.8342

Faraj, S., & Sproull, L. (2000). Coordinating Expertise in Software Development Teams. *Management Science*, *46*(12), 1554–1568. https://doi.org/10.1287/mnsc.46.12.1554.12072

Malone, T. W., & Crowston, K. (1994). The Interdisciplinary Study of Coordination. *ACM Computing Surveys*, *26*(1), 87–119. https://doi.org/http://doi.acm.org/10.1145/174666.174668

Razzak, M. A., & Ahmed, R. (2014). Knowledge sharing in distributed agile projects: Techniques, strategies and challenges. *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference On*, 1431–1440. https://doi.org/10.15439/2014F280

Strode, D. E., Hope, B. G., Huff, S. L., & Link, S. (2011). Coordination Effectiveness In An Agile Software Development Context. *Proceedings of the Pacific Asia Conference on Information Systems (PACIS), Brisbane, Australia, 7-11 July, 2011*, Paper 183. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.231.9750&rep=rep1&type=pdf