Learning from the Case Studies, How Global Software Development Process is executed in an Agile Method Environment

Ridi Ferdiana[1], Lukito Edi Nugroho[2], Paulus Insap Santoso[3], Ahmad Ashari[4]
Department of Electrical Engineering and Information Technology
Gadjah Mada University
Jl. Grafika No. 2 Kampus UGM, Yogyakarta 55281, Indonesia
Email: ridi@te.gadjahmada.edu[1], lukito@te.gadjahmada.edu[2],
insap@te.gadjahmada.edu[3], ashari@ugm.ac.id[4]

Abstrak. Belajar dari Studi Kasus, Bagaimana Proses Pengembangan Perangkat Lunak Global Dieksekusi Pada Lingkungan Metode Agile. *Tantangan terbesar dalam Software Development Global (GSD) adalah efisiensi waktu untuk mengembangkan. GSD menyediakan panduan untuk menggunakan proses bersama dengan muka seperti proses metode analisis terpadu atau metode air terjun. Meskipun, itu memberikan manfaat melalui dokumentasi yang komprehensif dan kejelasan, ia memberikan menghambat organisasi yang ingin menggunakan GSD tetapi dalam terburu-buru. Metode Agile mengklaim efisien dan pendekatan yang efektif untuk pengembangan perangkat lunak. Makalah ini laporan tentang bagaimana organisasi menggabungkan proses GSD dengan metode tangkas seperti eXtreme Programming (XP), Scrum, Agile Unified Process (UP Agile), Pengembangan Fitur Driven (FDD), dan Microsoft Solusi Kerangka Agile (MSF Agile). Makalah ini menggunakan studi kasus untuk mendapatkan pengalaman organisasi dan menjelaskan praktek yang berguna untuk organisasi yang ingin menerapkan GSD dengan metode tangkas.*

Kata Kunci: *Siklus Hidup Pengembangan Perangkat Lunak, Agile, GSD*

Abstract. The biggest challenge in Global Software Development (GSD) is the efficiency of time to develop. GSD provides a guidance to use the process along with up-front analysis method like unified process or waterfall method. Although, it gives a benefit through comprehensive documentation and its clearness, it gives inhibits the organization which wants use GSD but in a rush. Agile methods claim an efficient and the effective approach to software development. This paper reports on how organizations combine the GSD process with agile methods like eXtreme Programming (XP), Scrum, Agile Unified Process (Agile UP), Feature Driven Development (FDD), and Microsoft Solution Framework Agile (MSF Agile). The paper uses case study to get organization experiences and describe useful practices for the organization that want to implement GSD with an agile method.

Keywords: Software Development Lifecycle, Agile, GSD

1. Introduction

Global Software Development (GSD) is terms that is interchange-ably used to describe a software engineering process solution to overcome software engineering inhibitors in the distributed development. GSD is a contemporary form of software development undertaken in globally distributed locations and facilitated by advanced information and communication technology (ICT), with the predominant aim of rationalizing the development process (Sangwan, 2007). GSD offers a theoretical process to handle distributed software development.

As a software engineering process, GSD offers planning strategy, organization structure, and progress control and monitoring. Rather than others approaches or terms, GSD provides more sufficient process and workflow in the software engineering framework.

While GSD promises economic benefits through its software engineering process model, GSD still has consequences. Mockus and Herbsleb (Mockus, 2001) define the pitfalls like differences in infrastructure in different development locations, interdependencies among work items, and difficulties of coordination because of the differences (culture, language, time). Some research has addressed the GSD weakness through specific approaches like Rational Unified Process (RUP) for GSD (Sangwan, 2007), software configuration management in GSD (Pilatti, 2006), or a shared mental model in GSD (Bass, 2006). This research provides temporal solution and addresses a specific part of the GSD process, since GSD is an enormous topic and need a different approach to handle the pitfall.

Relating the Agile and GSD is somewhat contradictive. Agile needs intensive communication both within the customer and the team, but GSD make a distribution of the team that also contributes to the team dysfunction. Communication between peers becomes important to the team's collective understanding. Remote team members miss this, and consequently, their understanding suffers. Miller (Miller, 2008) shows that when the direct communication does not exist in their practices. It will weaken the adopted method. Taylor, et al (Taylor, 2006) shows that the Agile adoption in GSD is just like reinventing the wheel, since many of the Agile GSD experience reports do not provide any additional value in the existing GSD guidance. They also recommend the researcher to create a value or a framework to integrate Agile in GSD context.
Although it contradictive in the terms of condition, several re-searchers also provide field reports about successful agile adoption in GSD. Miller (Miller, 2008) confidently states that his team at Microsoft delivered sufficient software by integrating Scrum, XP, and GSD. Hazzan and Dubinsky (Hazzan, 2006) states the diversity that exists in GSD is naturally supported by agile software development. Paasivaara et al (Paasivaara, 2009) captures the Scrum practices that successfully adopted in three GSD projects.

The research contribution presented in this paper is continuing the former research by doing self-evaluation. The self-evaluation is done by presenting case studies from four sample organizations in Asia pacific and others. The purposes of these case study are to learn how the real implementation of an agile method in GSD process. The research has identified several patterns and practices that happen in these case studies. The patterns and practices are described in several categories; project planning, requirements, architecture and design, and product development.

The pattern and practices that are described in this paper can lead to benefits for an organization in the following areas: (1) Planning the GSD in Agile environment more precisely especially in release and team planning. (2) Identifying what challenges in the requirements and how to solve it. (3) Identifying the GSD best practices that happen in the real project.

2. Observed Case Studies

The research chooses five case studies samples. Each case study delivers its own unique value regarding with others. Although the projects have unique values, they work in same population, which is distributed development population and executing using GSD approach. The samples come from several commercial organizations and non-profit organization with the purpose of capturing the execution behavior of each organization. In order to create a full picture of the agile GSD execution, the research makes an effort to captures several different projects that have unique attributes. There are three main unique attributes in the samples that are team scale, project type, and project licensing.

Team scale describes the team structure of the project. It can be a small (less than dozen), medium (not more than a hundred), and large (more than a hundred). Team scale is intended to expose the scalability of GSD in the executed project. Team scale will help the research to obtain what values that works and does not work in the different team scale.

Project type describes the result of the project. It gives an abstract technical complexity of the software. This attribute will guide the research to acknowledge how effective the process in the variety of technical complexity. This attribute also shows how an organization addressing the different problem with same GSD process does.

Project licensing tells the project initiative and its related with the process behavior. Commercial product means that the software can be acquired of all people by buying the product, open source means that the software can be acquired freely, and tailor-made product is custom software, which dedicated for the clients who order the software. Those varieties lead the research to gain the information which is related in organization policy regarding the project licensing and execution structure. Table 1 shows the quick preview about selected case studies with theirs own attributes.

Table 1. Research Case Studies

Project alias	Team scale	Project type	Project licensing
Alpha	Large	Operating system	Commercial product
Beta	Large	Operating system	Open source
Charlie	Medium	Information and automation system	Commercial product
Delta	Small	Community information system	Tailor-made product
Epsilon	Small	Knowledge management system	Tailor-made product

Alpha project is a commercial operating system project that developed by multinational independent software vendor organization. The organization itself has worldwide contact for more than 50 countries and has several offices in six continents. The organization has been prior experiences to build operation system from the scratch, and have more than 3000 developers with equals than 3300 unique modules that developed for this project.

Beta project is the open source project that initiative from an organization that worked for UNIX based operating system. Rather than just a kernel, Beta project is a complete operating system. It shows sophisticated modern architectures with more than 19 million of codes and documentation. Beta project is developed and maintained as open-source software by a team of more than 350 individuals located throughout the world. The work foundation can be roughly to code for system kernel, operating system utilities, porting third party program, and building documentation.

Charlie project is commercial application that developed by six academic organization and one commercial organization. The project is to develop a unified management station for building automation systems such as heating ventilation and air conditioning, access control, and lightning. This project is leaded by a commercial organization that separated in six countries and has 76 software engineers. Charlie project is executed in multi-year and divided its worked in United States, India, England, Germany, Brazil, and France.

Delta project is community web site application that developed by independent software vendor organization in South East Asia. The software itself is developed for a client that operated worldwide and has investment relation by the organization (parent and branch organization). The client is separated at eight countries within the same continents. Delta project vision is to build community information system that related with specific technology adoption from the client requests. The solution is worked as a rich internet application with more than 17 modules and worked by five people that separated in a region but still in the same country.

Epsilon project is intranet web application that outsourced to an independent software vendor organization. The client of this project is a mining and oil company. The outsource vendor is also come from Indonesia that have specialization to build web application product. Epsilon project is a mining monitoring system, which is displayed in their intranet portal and their executive information system (EIS). The project itself has more than 23 modules and worked by four people remotely and one people onsite.

3. Data Gathering Procedures

The purpose of the data gathering procedure is providing enough information to observe the case studies through research variables. The research will collect the data by exploring the case study related artifacts in a GSD formal workflow. GSD formal workflow describes four main phases, which are project planning, requirements engineering, architecture designs, and product developments (Sangwan, 2007). Each phase will be captured and discussed in scope of research variables, this following steps in the research called as data gathering phases. Based on the GSD formal workflow there will be four data-gathering phases, which are data gathering in planning phase, data gathering in requirements phase, data gathering in architecture phase, and data gathering in product phase. Figure 1 show the case studies process.

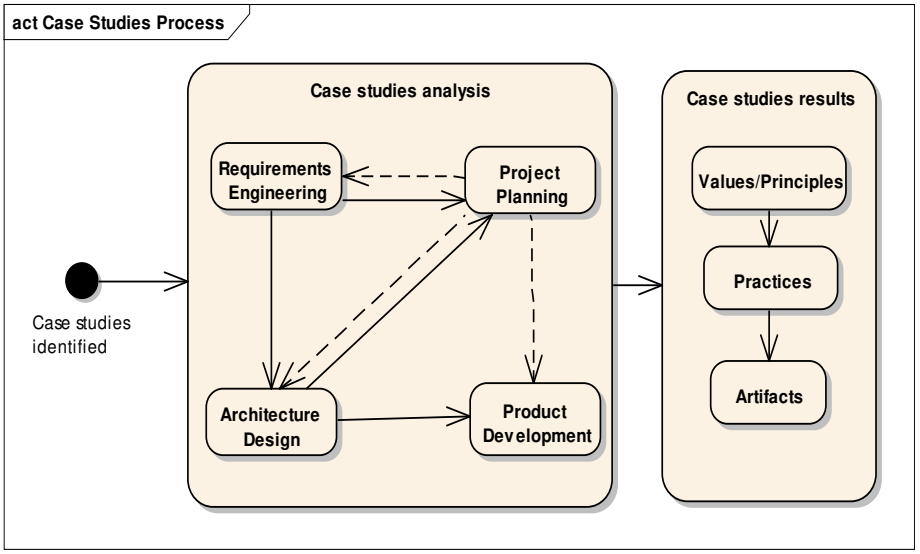

Figure 1. Case Studies Process

The collected data is stored in several variables There are two main categories in the research which are tangible and intangible variables. Tangible variables are sets of variables that quantitatively available such as team numbers, software line of codes, and anything that can quantitatively described. Intangible variables are sets of variables which qualitatively variables such as organizational wisdom, strategic planning, software complexity, etc. Tangible assets are concrete and codified, whereas intangible ones are implicit. Both variables will become significant inputs to discover several result attributes in the designed framework. In order to align the research with the software engineering research, variables is categorized in three main variables in software engineering that are people, process, and technology. Table 2 provides the tangible and intangible variables that exist in the research.

Table 2. Research Variables

Factor	Variable	Category
People	Team members	Tangible
	Team members experience	Tangible
	Same project experience	Tangible
	Same team experience	Tangible
	Organization structure	Tangible
Product	Product platform	Tangible
	Product type	Tangible
	Product complexity	Tangible
	Product size	Tangible

	Product Artifacts	Tangible
	Integration with the existing product	Tangible
Process	Adopted process	Intangible
	Communication pattern	Intangible
	Adopted Tools	Tangible
	Development Practices	Intangible
	Adopted methods	Intangible

Research variables are gathered from several constituents that can be categorized as follows. (1) Set of information that already published in whitepapers, communication log, and live artifacts that exist in the web or offline session. (2) Product result and case studies that displayed as a best practices in application lifecycle management document. (3) Supporting surveys that works as secondary survey that can support the research need. (4) Several case studies also provide the researcher a non-disclosure agreement (NDA). Therefore, several authors, live artifacts, and whitepapers publication is not cited because the request of the NDA.

In the modest form, the data gathering procedures are done through several steps that are (1) Identifying available case studies artifacts or output. (2) Identifying the research variables. (3) Categorizing the research variables in several categories. The categorization purposes are to make the variables is easy to understand and measure. (4) Identifying the phases in the GSD process. (5) Obtaining the research variables from the phases. (6) Analyzing the variables into the valuable information. (7) Extracting the information into general pattern and practices.

4. Case Studies Analysis

This section is divided into five sub main sections according to the GSD process which are software method analysis, project planning analysis, requirement engineering analysis, architecture design analysis, and product development analysis.

4.1. Software Method Analysis

Project Alpha adopts a framework called Microsoft Solution Framework Agile (MSF Agile). MSF Agile provides guidance, values, and discipline to provide technical how-to to develop the system. MSF Agile expands the generic process into several project phases that are envisioning, planning, development, stabilization and deployment. Envisioning covers the information gatherings, requirements development, and capturing business process. Planning phase focuses in creating project planning and project schedule. Development phases focuses in constructing the high-level architecture and develop the implementation based on proposed architecture. Stabilization phase covers the testing model that done by the internal team and the customer. The end of the project is by deploying the system and enters the result to the maintenance model.

Project Beta adopts a feature driven development (FDD) methods. FDD is an agile method and is a Lean method that incorporates many aspects of Lean thinking (Anderson, 2003). FDD assumes five phases that are shape-modeling, feature list, plan by subject area, design by feature set, and build work package. Shape modeling promotes the design through a diagram to represent requirements engineering and architectural modeling. Feature list improves the existing result from shape modeling into streamlined package based on the feature list. The features lists assigned in a subject area plan and feature plan. The features list is detailed in design models. This step involves detailed, in-depth UML modeling, including enhancement of the Class Diagram and development of a UML Sequence Diagram for each Feature in the activity. Product development is started in the last phase by developing the code and unit tests for the code.

Project Charlie adopts Agile Unified Process. Agile Unified Process is simplified development methodology for RUP in an agile context (Ambler, 2008). The main difference between Agile UP and existing RUP is the limitation of up-front analysis and design modeling. Agile UP is captured into four phases that are inception, elaboration, constructions, and transition. Inception is a phase where the requirement and planning is happen. Elaboration is a phase where the team creates the architecture model. A construction is a phase where the architecture is coded into software. The transition process is a process where the software is tested and deployed into client.

Project Delta adopts Scrum software method. Scrum is a software methodology that is focused in project management and agile software development (Bates, 2008). Scrum divides the project execution into four phases that are product backlog, sprint backlog, sprint, and deliverable increment software. Product backlog is a requirements engineering process in project Delta. Sprint backlog is a planning model that exists in Scrum. The sprint backlog result is a plan that executed in coding and testing activities in a Sprint phase. The result of the Sprint phase is software that is deployed incrementally.

The Epsilon project adopts Ad-hoc methods that categorized as an eXtreme Programming (XP). XP in Epsilon project is adopted seamlessly in four main phases that are exploration, planning, iteration, production and maintenance.

4.2. Project Planning Analysis

Project planning is started by defining team structures for the project. Team structures are come up from many different situations. There are several backgrounds why the team is structured in several circumstances. For example, Charlie and Epsilon projects are outsourcing project that are developed by different organization, therefore there is a consensus before the project is executed. The consensus itself is suitable when agreed in budget and time circumstances. In the large projects like Alpha and Beta, the team structuring is happen before the software process executed. Large product development plan and prior experiences is the two main reasons why the team structure is structured before the software process phases. Another example is come from Delta projects, since delta project is software solution that developed for internal need of an organization, therefore the team structuring exist "just in time" when the software need to be developed. Delta projects do not need a specific consensus since the project itself is created for them.

Based on case studies team structures are developed under three circumstances that are. (1) Projects already agreed in the term of budget and time. This circumstance happens for small and medium project scales. In our case studies, this happens in Charlie and Epsilon projects. The research describes this circumstance as planning phase team structuring. (2) Projects are in initiation and market intense gatherings. This circumstance happens for large project scales that need more time and more fixed plan. In case studies, this happens in Alpha and Beta projects. The research describes this circumstance as pre-requirements team structuring. (3) Projects modules are already analyzed and designed. This circumstance happens for internal small project scales like Delta projects. The research describes this circumstance as development team structuring.

As mentioned, GSD team composition is varied from each case study. The research identified that team composition is influenced by some characteristics such as. (1) Team characteristics such as size, geographic dispersion and members shared work experience. (2) Organization characteristics like past experiences with other forms of virtual arrangements, organizational inertia, politics, and culture. (3) Tasks characteristics like its difficulty, ambiguity, and priority.

The research assumes that the geographical distribution is a major reason the differentiation between team models. The case studies show that the team model is equal with the Sangwan, et al research (2006). Sangwan, et al categorized the team model in three models as follows. (1) Remote model (hub-to-spoke model), this model is introduced in Epsilon project

and it is identified by geographically separation of client and development team. Since the team still in same place, there is no problem in technical communication, however the main interaction problem is happen between team and client. (2) Virtual model (hub-to-hub model), this model is introduced in Delta project. This model adjusted a separation between client and distributed team. The main idea of virtual model is the team works as a big team although there are separated in distant. The team is still in one management and work in same stream with the others member that separated in distant. (3) Mesh model (fully distributed), this model is introduced in Alpha, Beta, and Charlie projects. As is stated by its name, this model is the most distributed than the others model. Team and client are separated by distant, and team is distributed in different management that exists in the every team site.

4.3. Requirement Analysis

When looking in requirements gathering, the research did not find any different approaches that happen in requirements gathering. However, as with others aspects of software development, the requirement engineering related issues that GSD projects experience can also found in the collocated software development. However, some of the issues are different in GSD for example the difficulties in change management, quality assurance control, and impact on related process. Change management is an area that can be most difficult under the best circumstances. The problematic problem is the ripple effect of a requirement changes including impact analysis, updating the associated artifacts, re-planning of system. Quality assurance activities are also quite significant for GSD projects. It is important to get precise feedback or progress of the various teams. One of the things can be problematic for many projects are to have test plan that have adequate coverage and executed parallel. Impact on related process occurs when the requirement changes, the GSD difficulties is to maintain a clear traceability, to update all the sites about the changes, and to create a log about the changes. All of the difficulties can be addressed through requirements artifacts that should have sufficient details to provide dependencies between requirements, to re-adjust as needed when the changes happen, and to provide backlog mechanism when the changes happen.

Alpha project develops their own technique in requirement engineering through several artifacts such as user profiles document to identify the users, vision scope document to identify the functional and non-functional requirements, and several usage scenarios to identify the business process detail. Since the Alpha project is developed as commercial operating system (OS), the requirements are gathered from the feedback legacy operating system, anonymous report usage, and technology adoption request. The gathering process is done by selecting the feature into three categories which are critical feature (like performance issues), business feature (like integration and compatibility issues), and nice to have feature (like 3D animation). Those features is identified and documented in artifacts. The requirements itself are quite stable because is driven by product and market initiative not directly by the client. The issue in this project is managing the requirements artifacts by keeping it up to date between sites. It solved by providing the team a collaboration workspace.

Beta project has similarity with Alpha project that also develop operating system. The differentiation between of them is their execution model. Beta project is executed as open source software (OSS) development; therefore, the development firm is driven by communities and users. The requirements is written and published in online wiki. In order to modify wiki (such as adding the information, request feature, etc.), the member should get approval from release engineering team. Release engineering teamwork as central team in beta project and they should make a sustainable plan for the OS, approving proposal for new developer, and resolving changes. The requirements itself quite stable because its nature as a software product. It is developed through forum-based requirement gathering process. After forum agreement, the community creates several wiki sites to discuss specific feature or application. A member who has an idea can contribute through the idea and update the wiki but not for the features that already freezes. Some of the member uses a diagram to make communication more effective.

Charlie project maintains its requirements through the requirements engineer that exist in central and remote team. Requirements engineer in central team creates high-level functional requirements, use-case specification, user interface specification, team verifies and validates requirement and maintain traceability between all these artifacts. Requirements engineer in site team facilitates team's understanding of all requirements, and ensure that all requirements specification work assigned to the team is being delivered per schedule. The wiki is the primary medium for specifying requirements specifications. It stores all the artifacts that tie together, i.e. use case specifications and sequence diagram is drawn by the Borland Together™, and it is linked with user interface (UI) document that is created by Microsoft Visio™. The artifacts is composed as Rational Unified process template and uploaded to the Wiki to provide easy traceability.

Delta projects started their requirements gathering by seeing the equal system, which exists in others region that already have community information system (North America region). The list of features is extracted by the architects and composed in a document. The document is sent through mail system to the project manager. The project manager discuss the document with others stakeholder that separated geographically by using email distribution list. Once the feature list is approved by the stakeholder, the architect start to build technical requirements and send it to the team members including project manager.

Epsilon project is another interesting project that used a conventional collocated model. Although the development is done by the remote team, the requirement engineering is done by the people (in this case project manager) who travelled to the client. The requirement process is done through ASR (Architecturally Significant Requirements) workshops that held every week during envisioning phases. The result of ASR workshops is a flowchart that describes how the system will work, and what to need to make it work. The remote team use the ASR result to build the proof of concept (POC).

All of these projects lead several inferences about requirements gathering in GSD that are. (1) The artifacts are the key of requirement engineering in the GSD projects. (2) The requirement artifacts are developed based on adopted software method. (3) The artifacts are stored and sent through tools or electronic communication such as email, wiki, web forum, and collaboration workspaces.

4.4. Architecture and Design Analysis

Architecture design in GSD and collocate projects is almost the same. The difference between of them majorly focus in how the team communicate the architecture through indirect media, split the work to separated team, and integrate the work when it's ready. Software product is defined its quality through better architecture, and architecture itself is defined by the organization structure behind of it. Since the tasks related with the architecture, components are tightly coupled and require more communication. The team needs to eliminate the coordination and social issue in global software development. Case studies in this research provide some alternatives how the team tackles those issues.

Alpha project defines the architecture by joining all the architects in online conferences. They use collaboration software to accelerate the communication between separated team. Video conferences, presentation sharing, and instant messaging are three keys to communicate the idea and solve the cultural issues through English conversations. Head architect headed a presentation that consist of structure template, team composition, and work unit for each architect. Through presentation sharing, an architect from central team described the overall solution in separated daily session. The architecture that described by central team architect is common sense high-level architecture; the detail about the architecture itself is done by the separated site team who handle it. Every architect has at least knowledge the overall architecture through published book and journal, internal documentation, and prior system architecture.

Beta project started the architecture development through communication between peers in IRC, Forums, and Wiki. The core architecture on beta project usually related with the

core operating system function such as managing and knowing the hardware resources. The initiator uploads the architecture artifacts through wiki and subversions. Diagram is the main component that displayed in the artifact. Diagram can be displayed as ASCII, picture, or common free case tools to create a diagram like DIA or Launchpad. The artifact mostly provides high-level structure of the system where the detail one usually is reversed engineering from the code. The artifact will be discussed and refined by the others member through comment system and the initiator or volunteer will update the artifact when it necessary. Since the project is available to the public, versioning control is necessary to control the changes that happen in architecture.

Charlie project started the architecture development by dividing the architecture in two main views that are module view and components-connectors view. Module views provide high-level software modules and their dependencies. Components-connectors view shows the dynamic interaction between the modules. The structure of the modules can be further refined to yield sub-modules and their dependencies. These modules and sub-modules can then be assigned to the individual teams for development when enough detail is worked out so each team understands what modules it relies on for its piece of the project and what other teams rely on its work. In order to provide the detail view, the component-connector will help the development team to simulate the modules behavior at the runtime process. Central team initiated overall architecture development, site team will add the additional modules when it necessary. The hub-to-spoke model is adopted in this project.

Delta project is started the architecture development indirectly through rapid development approaches. Development team creates the architecture when they built the proof of concept (POC). POC is a prototype based modeling that focus in several essential feature of the system. Development team creates staging servers in order to communicate the architecture, for further view in architecture communication, the architect in development team sent the architecture presentation through email.

Epsilon project is started the requirements through direct communication between development team and project manager. Project manager who is already get in touch with stakeholder communicate the result to the development team. The development team creates the high-level architecture in one-day workshop. For a week, the detail of the architecture is exposed to the client. It contains high-level architecture, technology recommendation, and some screenshot of system mock that should be approved by the client.

Architecture development in our case studies provides us broad view how the design is created in different organization. Several organization projects like Alpha and Charlie has a formal and forward structure to create the architecture. Beta project has unique architectures development through diagramming and initiated by the community. Delta and Epsilon create architectures through agile modeling, some of them also using reverse engineering technique to create the architecture.

Every project has unique approaches to overcome the architecture design. Based on this section, the research concludes some generic patterns as follows. (1) Synchronous or direct communication is needed in architecture design discussions. Therefore, CSCW tools like instant messaging, presentation sharing, or video conferences often needed to increase the limitation of direct interaction. (2) Diagram is the most valuable in term of asynchronous architecture artifacts. When the team could not meet each other's (e.g. because time differences), the diagram will help the team communicate their idea. (3) Prototyping like POC will give better feasibility to communicate the overall architecture when the cultural and language become obstacles for the team to communicate effectively.

4.5. Product Development Analysis

Products development in GSD is a topic that really depend on the product that being developed. The development model is depended on the product, organization experiences, method adoption, technology adoption, and organizational policy. Others than the product,

product development also depend on the software methodology that adopted by the organization.

Alpha project chooses MSF Agile as a framework that can support agile software developments. The codes are developed with short lifecycles and delivery-oriented teams based on architecture development. In order to integrate the result between the separated team, teams are supported by the configuration management tools called Team Foundation Server (TFS). TFS is provides configuration management of both code and documentation artifacts. It provides testing automation services, quality control workflow and serves as a central repository and collaboration portal for a software development team. The development task in Alpha project is driven architecture and time to market when each module is finished. TFS will store the module and integration team will integrate the module by referencing with the architecture.

Beta project run their project in individual module development. Individual module development is worked by a team that has separated members. Each member will do code development either in small regional team or single person code development. Beta project development is also known as "Round Clock Development". Round clock development is started by creating a workspace in a source-code versioning site like Sourceforge.com or Codeplex.com. The architecture plan is uploaded and the assignment is assigned to the team member. The codes are developed by an individual and then committed through collaboration site with FDD method. Committed code can be worked by different team member. Round clock development utilizes the 24-work shift for a member that has different time zone.

Charlie project started the development process through identification work unit's dependencies. Before the development, significant amount of communication and negotiation has to occur between the developers and the users of the software modules to arrive at a good understanding of the final product. It is become best practices to co-locate development of software units highly dependent on each other with relatively unstable descriptions. If co-locate is not supported, the central team chose the nearest teams to work for the software units and rely on asynchronous communication. Charlie projects using standardized the tools and the Agile UP artifact format for every site team.

Product developments in Delta project is solved by one central development site. Therefore the experience between collocate product development is existed in Delta project. Development team in Delta project is using Scrum agile development (Schwaber, 2004). The heart of Scrum lies in the iteration. The team looks at the requirements, considers the available technology, creates the product backlog, and evaluates its own skills and capabilities. When the initial features are approved by all stakeholders, the team is doing iteration for the features. The issue happens when the client request features changes in the middle of sprint. That kind issue is agreed by the development team and the stakeholder through a consensus. The consensus explicitly states that the feature changes will be discussed through Scrum meeting after or before iteration not in the middle iteration.

Epsilon project starts the product development through improvement of a mock object that is developed in architecture design. The mock object itself contain a layout of the system, design screenshot, and user interface interaction, Project manager visit the development team with a client feedback. By seeing the client feedback, development team creates a development planning through a planning game session. Planning game is a part of an agile eXtreme Programming (XP) method (Beck, 1999). Planning game provides release and iteration planning for the development. Modules is identified and structured as user stories, which will be developed in a pair programming technique.

Every project has its own specific technique to do product development. Product development itself is not stated explicitly in GSD process. GSD only provides the workflow and generic process to do distributed software development. The rest of the development is done by adopting specific methodology or framework like MSF Agile, FDD, Agile UP, Scrum, and XP. Some of the patterns are described as follows. (1) Unit decomposition is the main awareness to implement distributed development in order to limit the dependencies. (2) The development unit is worked in small iteration in order to make the progress is tracked and synchronized. (3)

Source code versioning and configuration management is deployed to track the progress of the codes in real time.

5. Understanding Success In Agile GSD

Some classics definitions about software success level that is successful, challenged, and impaired (Shore, 2008). Successful is delivery on time, on budget, with all features and functions. Challenged is delivery but over budget, over the time estimate, with fewer features and functions. Impaired is canceled at some point during development lifecycle. Successful has further definition in software project. It is defined as intersection between personal success, technical success, and organizational success.

The research shows that the GSD success related directly with organizational success. GSD or not GSD is a justification that usually comes from organization, therefore when the GSD is adopted; organization has some interest with the benefits. Therefore when adopting the GSD especially in agile method, the patterns and practices are readdressed for the organization wisdom to do several things which are. (1) Creating team model and composition. (2) Creating standard artifacts and knowledge based system. (3) Preparing communication standards and tools. (4) Proposing coding standard and modularization rules. (5) Tracking the progress through several tools and engagement process.

Based on those facts the successful of the software development in GSD is organizational success. Organization should understand the benefits being agile, working in distributed environment, and collaborating with different culture. Several assumed benefits that exposed implicit in these case studies are. (1) Bargain development costs. Alpha, Delta and Epsilon enjoy this benefit by moving parts of the development work to low wage countries, the same work can be done for a fraction of the cost. (2) Leveraging time different for additional work-shift. Alpha and Beta project utilizes the time zone to make 24 hours of work shift that can decrease cycle time. (3) Cross-Site modularization of development work based on their technical specialty. Alpha, Beta, and Charlie enjoy this benefit by developing modules in parallel which can reduces cycle time. (4) Admission to large skill labor pool. Five case studies show that GSD has potential to facilitate access to large pool of highly skilled workers. (5) Invention and shared best practices. Beta and Charlie projects lead to increased innovation and shared best practices amongst team member. (6) Closer proximity to market and customer. Alpha, Beta, Delta and Epsilon projects gain a more direct interaction with the customers become possible. Close culture and neighboring position will have better knowledge of local business conditions.

6. Discussion and Further Improvement

One of the goals of this research is discovering “real picture” of GSD project execution in agile environment. Case studies offer some of execution practices, recommended team structures, and the success criteria of the GSD project. Those result work as raw material to enhance the agile process into distributed software development. Case studies show several key patterns that already described before. (1) GSD process in agile environment is used by adjoining the existing method that is chosen by the organization wisdom. (2) The team structure is based on the organizational wisdom in three different phase, which are planning phase, pre-requirements, and development phase. (3) The requirements in agile GSD are treated as casual requirements that needed to handle changes and collaboration through artifacts and better communication tools. (4) The agile architecture and design phase improves the GSD performance through diagramming, prototyping, and synchronous communication. (5) The codes development in agile GSD show that decomposition, small iteration, and source code become patterns in code and production phase.

Those patterns provide valuables information for organizations who want implement GSD in agile environment. Furthermore, the paper results can become a substance for the

researcher to create a formalized GSD framework in agile method.

Acknowledgment

The authors wish to thank Microsoft Innovation Center UGM, Gadjah Mada University Yogyakarta Indonesia, and all the local reviewer of this paper.

References

Ambler, S. W. 2008. *Agile Software Development at Scale. In Balancing Agility and Formalism in Software Engineering: Second IFIP TC 2 Central and East European Conference on Software Engineering Techniques,* CEE-SET 2007, Poznan, Poland, October 10-12, 2007, Revised Selected Papers, B. Meyer, J. R. Nawrocki, and B. Walter, Eds. Lecture Notes In Computer Science, vol. 5082. Springer-Verlag, Berlin, Heidelberg, 1-12.

Anderson, J.D. and Schragenheim E., 2003. *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results.* Prentice Hall.

Bass, M. 2006. *Monitoring GSD projects via shared mental models: a suggested approach.* In Proceedings of the 2006 international Workshop on Global Software Development for the Practitioner (Shanghai, China, May 23 - 23, 2006). GSD '06. ACM, New York, NY, 34-37.

Bates, C. D. and Yates, S. 2008. *Scrum down: a software engineer and a sociologist explore the implementation of an agile method.* In Proceedings of the 2008 international Workshop on Cooperative and Human Aspects of Software Engineering (Leipzig, Germany, May 13 - 13, 2008). CHASE '08. ACM, New York, NY, 13-16.

Beck, K. 1999. *Extreme Programming Explained.* Addison-Wesley.

Hazzan, O. and Dubinsky, Y. 2006. *Can diversity in global software development be enhanced by agile software development?* In Proceedings of the 2006 international Workshop on Global Software Development for the Practitioner (Shanghai, China, May 23 - 23, 2006). GSD '06. ACM, New York, NY, 58-61.

Miller, A. 2008. *Distributed Agile Development* at Microsoft Patterns and Practices. Microsoft.

Mockus, A. and Herbsleb, J. 2001. *Challenges of Global Software Development*. In Proceedings of the 7th international Symposium on Software Metrics (April 04 - 06, 2001). METRICS. IEEE Computer Society, Washington, DC, 182.

Paasivaara, M., Durasiewicz, S., and Lassenius, C. 2009. *Using Scrum in Distributed Agile Development: A Multiple Case Study.* In Proceedings of the 2009 Fourth IEEE international Conference on Global Software Engineering (July 13 - 16, 2009). ICGSE. IEEE Computer Society, Washington, DC, 195-204.

Pilatti, L., Audy, J. L., and Prikladnicki, R. 2006. *Software Configuration Management Over A Global Software Development Environment: Lessons Learned From A Case Study.* In Proceedings of the 2006 international Workshop on Global Software Development for the Practitioner (Shanghai, China, May 23 - 23, 2006). GSD '06. ACM, New York, NY, 45-50.

Sangwan, R., Bass, M., Mullick, N., Paulish, D. J., and Kazmeier, J. 2007. *Global Software Development Handbook* (Auerbach Series on Applied Software Engineering Series). Auerbach Publications.

Schwaber, K. 2004. *Agile Project Management with Scrum*. Microsoft Press.

Shore, J., and Warden, S. 2008. *The Art of Agile Development*. O'Reilly Media, Inc.

Taylor, P. S., Greer, D., Sage, P., Coleman, G., McDaid, K., and Keenan, F. 2006. *Do Agile GSD Experience Reports Help The Practitioner?* In Proceedings of the 2006 international Workshop on Global Software Development for the Practitioner (Shanghai, China, May 23 - 23, 2006). GSD '06.