mathematics

MDPI

Article

Dynamic Agile Distributed Development Method

Muhammad Asgher Nadeem and Scott Uk-Jin Lee *

Department of Computer Science and Engineering, Hanyang University, Ansan, Gyeonggi-do 15588, Korea; engr.nadeem18@gmail.com

* Correspondence: scottlee@hanyang.ac.kr

Received: 23 July 2019; Accepted: 24 September 2019; Published: 13 October 2019

Abstract: "Agile" is an effective software engineering model with a high trust and acceptance rate among its users. The term agility comes from the concept of rapid development and working in a team for better results and a faster competition rate when compared with any other software engineering model. In this study, an assessment of the different patterns, frameworks, and application program interfaces available for distributed development in an agile model is given. After analyzing the state-of-the-art distributed models, a novel framework of a dynamic agile distributed development method (DADDM) is introduced in this paper. Many researchers have worked on global software development using the agile approach; however, we are presenting the idea of incorporating the agile benefits with dynamic distributed software development. The applicability of the proposed model is checked via two selected parameters: a feasibility study and a business study. The complete DADDM development life cycle is presented in the methodology section. The techniques used in DADDM and team members' roles and responsibilities in DADDM are defined in this study. This study reflects all pillars of planning, controlling, organizing, and management of leadership. The use of DADDM in distributed agile development encourages future researchers to use this proposed framework for comparison and testing of their models and to check the effectiveness through a comparison with DADDM.

Keywords: distributed development; agile development; application program interfaces in agile distributed projects; framework for agile distributed development

1. Introduction

In distributed agile development, the agility rate is much higher in a team working with 40 people. The main focus of distributed agile development is the ability to respond to changing requirements and environments [1]. The ability to meet changing project deadlines with a satisfied customer base is the core concept of agile development.

There are different kinds of agile methods that exist. Agile-distributed development procedures consist of various frameworks that focus on a wide range of project management methods. Major types of agile development are dynamic systems development, crystal methods, agile, and extreme programming [2]. The highly recommended and most effective methods are extreme programming and agile. Agile works on the principle of an empirical process control model. This difference makes agile more effective and a better choice for system development projects. The agile method for distributed development involves agile teams consisting of the product owner and agile master, and the definitions of events, rules, and artifacts. Agile events are used to complete the processes on time, create regularity in tasks, and reduce the time required for unnecessary meetings [3].

Currently, the value of agile development is increasing because of its intuitive functionality. Projects are being completed on time and according to the specific needs of customers. The salient features of the dynamic agile-distributed method are innovation, flexibility, and effective management.

Project development is central to the responsive and disciplinary approach, as it has the program's baseline constraints as control objectives.

Isolation exists in both customary prerequisite project building and dynamic agile distributed development method (DADDM) procedures. It works through absolute requirements by appropriately assembling, arranging, and archiving all prerequisites while avoiding any conflicts. In DADDM, reliable creation is due to the correspondence issues among clients. However, it gives the developer a chance to create prerequisites, for example, the use of cases. These utilization cases are written in common language and are given useful prerequisites. Numerous distinctions exist and are of prime significance, particularly for programming improvement ventures.

1.1. DADDM Methodologies to Client

In the past, prerequisite issues were less significant. This was because there were not many prerequisites or changes required by the customer. Many organizations are trying to improve their products and are working towards complex requirements to improve customer satisfaction. These include the prerequisites of over-burden, no client commitment, the absence of compelling correspondences, the fulfilment of time constraints, spending issues, and substantially more inconveniences. To conquer these issues, computer science specialists presented the agile manifesto in distributed development in agile modeling. The DADDM methodology essentially focuses on programming advancement rather than documentation; it effectively works when clients' requirements change to ensure clients' satisfaction, provide fast advancement, and much more.

Many techniques exist, including:

1. XP, generally known as Extreme Programming;
2. Approach of Agile Modeling;
3. Nimble AGILE Methodology;
4. Nimble Feature Driven Development;
5. Nimble Automated Software Engineering;
6. Deft Lean Systems Engineering, Lean Software Development, Adaptive Software Development of Agile.

There is typically an organized structure for the improvement of enormous and complex frameworks. The three useful systems being embraced are theory, cooperation, and learning. Theory relates to arranging the board requirements, which allows experts to prepare for future prospects. Cooperation is similar to client association, as clients and colleagues work together among themselves to discover fundamental clashes and issues with important understandings. Learning deals with preparing or change of prerequisites during the improvement process, which implies that designers or clients work together and collaborate in a team for better results. The main objective of this is to create fundamental models applicable to programming improvements that utilize tests and conceptualization.

1.2. Agile Artifact

Agile operates on a pattern that is designed with a single purpose that maximizes transparency and provides adaptation and opportunity for interoperability. A product background is updated regularly as the product gets larger. The development phase requires more improvements and involves irregularities in preferences. That is, generating a breakthrough requires certain changes be made in the product backup.

1.2.1. Role of the Agile Team

Agile works on pre-determined characteristics for effective participation and the fast delivery of the product [4]. These are as follows:

Owner of product

It is the owner's job to accurately understand the metrics and needs of the market and business values.

Product Owner Responsibilities

- The development team should work in a specific order to ensure rapid work completion;
- The purpose of the product must be retained within the team by defining and updating product requirements;
- Feedback must be collected from customers, clients, and partners, and this information must be transferred to the team;
- Optimized work must be carried out in every stage;
- The quality of the product and delivery method must be considered;
- A product release plan must be constructed [5].

An overview of the team members is shown in Figure 1.

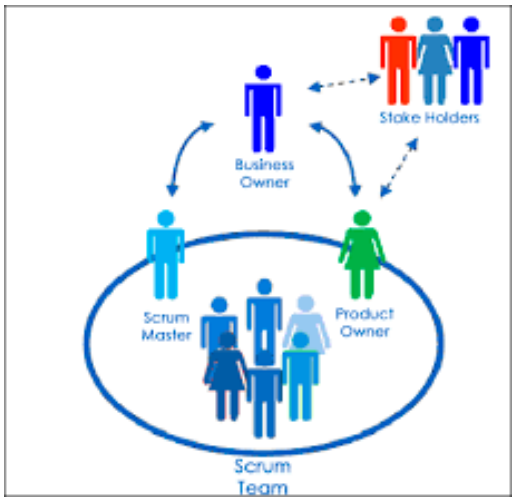

Figure 1. Overview of agile team members.

1.2.2. Agile Master

To help the team and owner, the agile master works as a trainer for understanding the approaches used in the management process of an agile project. This role helps streamline the workflow to adjust the agile project management protocols. In the project, the agile master acts like a backbone. The agile master fully understands the jobs of the team and is also aware of the requirements needed for getting a job done. In addition, the agile master knows all the details of the project and wears multiple hats in the project.

Role of the Agile Master

- Helps the team understand the values and approaches of the agile process;
- Helps the product owner update, check, and document product requirements;
- Makes sure the opportunity and goals of the product are well understood by every team member;
- Is responsible for properly conducting agile events;
- Minimizes team clashes and other weaknesses that are barriers to the on-time delivery of product;
- Manages the team in the right way to attain maximum production;
- Keeps a record of implementation of every successful sprint.

1.2.3. Development Team

The team consists of different divisions, including developers, testers, and engineers. The agile team is well organized and works in collaboration by helping each other to achieve targets. Every member of the team updates and shares their work with other members to keep everyone informed about the progress of the project [3].

Responsibilities of Agile Development Team

- Teams are independent, and the product owner or agile master do not put pressure to give any output at the end but check their work consistently with light touch;
- Groups do not separate, and each team member works according to their experience;
- Groups are cross-functional and cooperate to check the progress after each iteration.

1.3. Agile Framework

Agile frameworks work on a group of approved and synchronized procedures. Every event is time-boxed and very similar to sprints in which the entire development of the product takes place. The life cycle of every agile event is dependent on the objective success status for the stage.

1.4. Types of Agile Meetings

a. Sprint Planning Meeting

The first step is to arrange a planning meeting before the project starts. In this phase, the product owner plans the product requirements and identifies the most important things that are mandatory and must be executed quickly. In turn, the team strategizes to implement the checklist provided by the product owner. In this meeting, the team plans the deadlines and decides how work will be done efficiently. All aims and goals are planned during the sprint meeting.

b. Daily Agile Meeting

Daily meetings of teams begin once the sprint planning conference has started. These are very short meetings, not more than fifteen minutes, in which challenges, issues, progress, and feedback of the project are discussed among team members. New solutions and different strategies are also discussed on a daily basis.

c. Sprint Review Meeting

This review meeting is held at the end of the sprint to discuss what the team has achieved at the end of the iteration. Stakeholders and clients participate in this meeting. The team discusses the next plan that will be executed in the next phase. This review meeting emphases cost, timeline, and available resources.

1.5. Extreme Programming

This procedure of agile project management emphases changes in response to client and customer preferences. It was created by Kent Back in 1999 as a new perspective for software development with a basic emphasis on how the code is written effectively, rapidly, and error free. The main idea is that a unified method of XP involves the development of a system based on teamwork and elements being developed including coding, testing, and design screening [6].

1.5.1. Extreme Programming Value

Extreme programming is not just a multi-functioning method; there are many positive things to help and guide the team's behavior and to promote extremely high programming for better project management.

There are five key values for high-level project management:

- Simplicity—Workload is divided into small components to eliminate the complexity. The team only focuses on part of the work at a specific time and meets it.
- Effective communication—The team combines the collaborative work, including every outcome and error; daily meetings are known as the interaction.
- Continuous Impact—The team's focus and their workflow center around the opinions and changes from customers.
- Respect—Each member of the team is recognized for his feedback and takes part in the effort. Despite the organizational status, the administration admires every team member and gives respect, and the developing team does the same with the customer's demands.
- Courage—The groups are stimulated instead of accepting failure during these projects, and the focus is not on their propaganda.

1.5.2. Extreme Programming Rules

Dawn Wells [6] published the first XP rules in 1999 to clarify the key components of software development rules and list the activities that gradually became the working part of the XP model.

1.6. Management

- The Project Manager establishes a workshop to start work for their team;
- The Project Manager determines the project's speed;
- Stand-up Meetings are held on a daily basis;
- The performance of every team member is checked and reinstated when any procedural or dispute problems arise;
- The development approach changes if the XP method does not properly incorporate the team member [7].

1.7. Designing

- Start with a simple design to avoid delay in completion;
- Functional increases in design should be saved later;
- A class responsibility collaboration (CRC) card is used.

1.8. Coding

- Collective code ownership is applied such that any developer can change the code, detect or repair the errors.
- Using system residences (each team member including clients can explain how the system works).
- Programming is paired to write code and send it to production.
- Integration of code is done every few hours and is stored [8].

1.9. Testing

- Acceptance test requirements are repeated.
- Unit testing is performed regularly before issuing the code.

1.10. Kanban

Kanban is best recognized as a scheduling system, which has a great advantage over production inventory. Kanban limits the amount of extraordinary work. This prevents businesses and teams from being neutral and chaotic, which damages inventory track records. Kanban only uses the time-consuming approach in which the project managers enable them to use their resources [9].

1.10.1. Kanban Project Management Principles

The Kanban technique works on four elementary principles [10] that describe Kanban's process. These principles are as follows.

1.10.2. Start Now

Kanban does not work in a phase and does not work in the cross-ending method. It can be applied to any workflow in the project life cycle. Kanban is already under development and can be applied to a project.

1.10.3. Save Current Processes, Role, and Responsibilities

Contrary to other methods, there are certain rules and frameworks that must work in the project. Kanban teams can continue to work with the same responsibilities in their current roles before they accept and synchronize them. This involves old business practices as well as deliberately fraudulent frameworks.

1.10.4. Leadership Actions

Kanban's flexible model allows transparency in the working frame. Since there is no such classification for the Kanban perspective, every member of the team knows how to access the host for the top and bottom of the project. Kanban promotes leadership power over all stages with its power mode, which is available for every team member. This requires the team to retain a mentality of continuous upgrading on a discrete level.

1.10.5. Kanban Practice

The team follows actions that must be completed for the successful Kanban process.

1.10.6. Workflow

Kanban encourages teams to work in projects and tasks within the work of quick and efficient workflow.

1.10.7. Adjustable Boards

The Kanban Board is a pictorial demonstration of the workflow at each phase of the project life cycle. The board has three parts: development, completion, and application. It serves as a real-time inventory that is constantly updated to include any obstacles, problems, and constraints that are potential threats.

1.10.8. Kanban Cards

A section of the Kanban board is made of Kanban cards. Cards have information such as job details and their development assigned to it as well as its history. These cards serve as a center of information that is accessible to the entire team. One of the most impressive and interesting features of the Kanban cards is the ability to move cards from one side to another, with smooth drag and drop functionality.

1.10.9. Limit Work in Progress

The best use of Kanban is to bind WIP (work in progress). Tasks are assigned extreme limits (which are located in the column of the counting board in this case). Number of tasks in progress cannot exceed the number of jobs requested. In a second step through a Kanban system, teams must work together to narrow down the limitations.

1.10.10. Manage Flow

Workflow through the Kanban system should be free from any roadblocks. Efficient workflow flows from one phase to another phase in a continuous mode, providing evidence that team efficiency and work metrics are fruitful and have business value.

1.10.11. Continuous Improvement

Kanban does not promise successful implementation. The teams need to work on their weaknesses and improve their strengths to maintain stability and expedite quick delivery.

1.11. Application Software Development

Appropriate software development is a framework consisting of adaptive modes, which adopt an adaptive process while building large and complex products via mutual cooperation. The best practices of application software development are emphasized to accommodate adaptive, growing patterns, so that we can dynamically cope with unusual conditions when they occur.

1.11.1. What Is Distributed Agile?

Project development with distributed agility is an approach to software development that connects the team members from different countries, time zones, and language. Good communication and creating a highly functional environment are the fundamental elements for better operation and good results from the team. Some principles of distributed agile need to be integrated with the agile software development to make it distributed agile development:

- Interactions among team members and a better understanding of the processes and tools;
- Working software with comprehensive documentation;
- Contract negotiation and customer collaboration;
- How to respond to changes in plans.

1.11.2. Distributed Agile Software Development

Distributed agile software development combines the features of an agile model with distributed computing technology and upgrading the agile model to the next level. In distributed agile software development, we make use of external resources and high-profile competent in-house teams. The highlighted outcome for the approach is the minimum time required for the project completion. We can meet deadlines easily and never need to work in high-risk situations. In this development model, we integrate team members across the world and provide the best utilization of this extended team. One of the most appropriate ways of uniting them and working efficiently is to develop an effective communication environment.

1.11.3. Significance of DADDM

- Improves the productivity of dynamic agile programming by presenting intuitive and semiautomatic strategies in various periods for improving requirement engineering;
- Decreases the intellectual multifaceted nature of light-footed programming advances and determines complex choice circumstances effectively by defining scientific models;
- Improves the correspondence and coordination of dispersed agile programming advancement groups by presenting semi-mechanized strategies to the appropriate groups;
- It improves the nature of agile programming resulting in lower levels of risk by considering all significant factors (for example conditions, limits) in numerical streamlining models;
- Agile programming improvement focuses on the best arrangement for a particular setting, and clients' criticisms are addressed by modifying requirements, limits, and needs.

2. Literature Review

Extensive remote software development programs are complex and have a low tolerance rate that creates many challenges. Despite the challenges of handling multiple development teams, the methods of agility are being adopted based on the quick process to support the team's agreement. Artifacts are solid products of the software development process aimed at the teams' perspective on the same development program. The purpose of the study of Bass [11] was to focus on the development process to meet the requirements of remote software development programs concentrating on the infrastructure used in the development process. Using practitioner interviews, 46 software development teams were adopted in nine international companies combined with a documentary approach, documentary sources, and observations. The data analysis used was an open coding memo consistent comparison. This study identified 25 architectures of five types including features, sprints, releases, products, and corporate governance. It was discovered that with the plans of conventional angels to encourage the rules, strategies associated with planning-based methods were encouraged. Study-related experimental evidence has been used to identify the main owner of every artifact and generate a graphic map of specific activities. Finally, the programs developed in the study created a dynamic and plan-oriented architecture to improve the performance of the enterprise's quality and technology strategy, while the risk of failure was also reduced in the agile development environment [11]. It worked well to satisfy the requirements of remote software development programs. However, its applicability is smaller in agile development, which highlights the need for our work on designing and developing dynamic agile distributed development methods.

A combined animated approach and distribution software development (DSD) were used to promote better quality software solutions in less time and at lower cost. The method helps in both the agility and distribution and reduces major challenges and risks. This work was intended to create a broad set of risk factors affecting the output of distributed freeware projects and indicate risk management practices [12]. Their work was unique because it integrated the simulation method with DSD, and it will be more useful if applied in dynamic project development.

The study provided continuous comparisons to analyze deeper interviews of twenty projects from three practitioners and thirteen different information technology (IT) organizations. Field experience was supported by extensive research literature on the management of risk in agile and distributed development. Interviews and project work documents showed the five risk factors of the group under five main risk categories. The risk category was mapped for organizational changes to facilitate results of the real world. Risk factors can be attributed to the DSD, and the development-related issues in agile are presented. Besides, some new risk factors have been tested using this process, and further investigation is needed. Distributed agile development (DAD) is being adopted in organizations to meet the changing business needs caused by global transformation. This study reduced the risk impact and enhanced the possible options to mitigate the influence to a maximum extent [12].

Software development has become mainstream. The authors [13] worked on the Scaled Agile Framework (SAFE) and showed the best practices for studying two industry case studies [13].

Knowledge management (KM) is mandatory for any software project, but especially in global software development, where members of the team are separated from time to time. The knowledge management software is organizing knowledge in various ways to enhance transparency and improve the performance. One way to evaluate these strategies is presented in Reference [14], as they defined seven Knowledge Management areas. The purpose of their research was to study the methods of knowledge and sharing in many distributed ages. This is done by managing a series of interviews during a specified time period. If we apply their findings for the agile development platform, then we can get a high customer satisfaction level. Their results showed that sharing knowledge in remote areas and trusted fiscal projects created a sound base of successful project completion [14].

Adapting mass agility often requires the integration of agile methods and non-dynamic growth elements to build hybrid adaptive procedures. The challenge is to determine what elements or components (freelance or unauthorized) of hybrid evolution procedures are relevant to develop a

reference architecture. The authors [15] addressed this important challenge and used a hybrid creative experiment to produce a hybrid adaptive model. They proved that there are a lot of chances for improving adaptive hybrid and agile-based development approaches using controversial engineering approaches [15].

The options available for project management methods have increased significantly; there are many questions for project managers considering alternatives. Project rating systems and standards are not met with logical project goals, features or environments. The main reason for this is that many software projects do not receive time, budget or client's attention. The purpose of the paper [16] was to identify and categorize the important success factors (CSFs) and to develop a controversial approach based on traditional project-oriented and freelance models. After reviewing the previous work, 37 CSFs were identified from 148 articles for software development projects. These were then ranked in three main CSFs categories: organizational, team, and customer factors. An unsatisfactory model highlighted the need to meet project properties and project management procedures on CSFs. This study suggested a model and helped to develop rules to assess the role of CSF for the success of the project. Although future experimental testing of this conceptual model is necessary, it provided an initial step in collecting the number of databases, provided a detailed experimental analysis for comparative studies, and improved the description of CSF [16]. The importance of critical success factors will become more impactful when it comes with dynamic project development, where the customer is allowed to make changes at any level of the development phased. So, if we take the work on this research as the initial point for our research and integrate it with the dynamic agile development method, we should achieve a high rate of customer satisfaction.

In the past, it was assumed that the global software development and the agile methods are incompatible with each other. The author of Reference [17] explained some of the core difficulties and benefits of using the agile method for global software development. They suggested applying this on an industrial level. This point is the base line of our work and we adopted their approach for the compatibility design of the software development in the distributed agile environment.

Agile distributed development models are currently widely used by most of the software organizations due to the fact of limited time and cost, but this method also creates many additional problems for software development teams. The result was an unstable software architecture. To address this problem, a situational ADSD (agile distributed software development) framework [18] was proposed, which can handle different requirement challenges for different situations. The gaps in situational ADSD also motivated us to propose a dynamic and generic DADDM, which applies to all kinds of distributed software development processes to complete on time and budget.

3. Methodology

3.1. Assessment of the Patterns for Agile-Based Distributed Development

3.1.1. Agile Integration

Organizations need to put things like business segments or administrations together rapidly. At its core, agile development is an approach to complete programming tasks more rapidly. Presently, agile development can be used in conjunction with small groups and a gradual methodology to determine what is required and how to achieve it more successfully instead of making it into a multi-year plan. Thus, agile is not just about software development or external software acquisition. The agile philosophy can also apply to broader concepts of such as integration and infrastructure. However, it brought change to the overall tech industry, driving the fast-moving needs of the customers.

3.1.2. Distributed Environment

Integration has been around for decades, but this approach is closely related to technical concepts and organizational patterns. Both are required in the modern context. The integrated technology

pattern around the ESB (enterprise service bus), where a centralized software component performs integrations to backend systems. The goal of integration was to make the logic easy and accept the changes. In the past, integration was performed by the most prominent people, who were primarily focused on distributed agility. With time, they moved away from the central distributed model. These integration experts now participate in invisible teams of software development, which helps to break the prevailing myths about integration.

Organizations realize that integration has major potential for their business, which can result in the integration of previous office and agile teams. One goal is to rapidly provide software to a qualified organization. Systems and services must be discussed together. If the organization does not unite quickly, it cannot solve the problem rapidly. Modern companies are adopting an organizational model for software like Microsoft. This creates a high failure of software modules being created. All software teams must be able to make this team part of the organization.

With distributed integration, you can separate the full module development, divide the task into pieces, and integrate the work progress using a centralized environment. Some technologies were prohibited five years ago. JBoss Fuse and JBoss AMQ are two examples. Modular deployment in a distribution agile environment is complex. This works best with the principles of containers and APIs (application programming interfaces) as deployment packaging and integration models.

3.1.3. Containers

Containers are independent application bundles, which must ensure rapid deployment and modification. Containers are a way of packaging your software and, in the context of Open shift, they can be fast and can be centrally organized. If you have integration infrastructure, then agile teams can work freely with general libraries and frameworks. It provides a basic structure that ensures good governance, which ensures the soundness of development.

3.2. Application Program Interfaces

Application program interfaces (APIs) are the third pillar of distributed agile development. This acts as an additional layer in the development process. Many companies now want to implement APIs in their primary system such that everyone can use them. Some organizations have hundreds of applications. Many times, they may be ignored, pursued, and used later. The privacy API can be tracked, but sometimes the cloud APIs are not easily tracked. We have a hybrid cloud environment for many customers. Many of them are posted in different elements. They have applications or backup systems in every environment that need to be securely linked with each other. So, 3scale API management can be used to keep users safe. Containers manage backup systems. APIs are a glue layer, and they provide communication. They can integrate data from different sources to create a different environment. You may have software or services and drive them all together to drive customer satisfaction and enhance environmental systems. Each time a customer creates a process of implementation for a single process, it is an island of information. However, it can be brought together with other operational data. Customers must take advantage of current investment and enable new jobs, so these APIs that create new applications are designed to mask and integrate with legacy systems.

3.3. Distributed Agile Development Frameworks

3.3.1. Requirement Hierarchy

Many times, the general user perception of the requirement is the short description of the working module. One of the main issues of requirements is that people do not know exactly what they want. Very detailed levels of requirements must be used. Requirements should always be clear. Different levels of ideas lead to a categorized model, thus reducing the overhead of requirement management for complex systems. Requirements can be classified into three different levels, where epics form the

highest-level requirement group are followed by characteristics at the middle level and end-user stories at the bottom as represented in Figure 2.

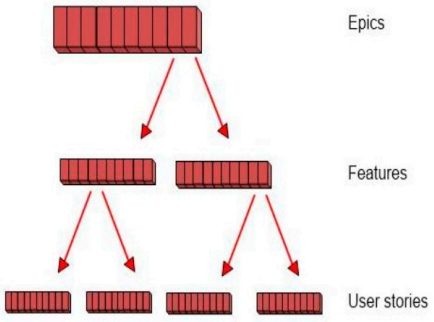

Figure 2. Hierarchical view of requirements.

3.3.2. Communication Advantages of Distributed Agile Teams

Almost all agile development methods involve the advantages of co-located teams and face-to-face communication. However, with the advances in technology and tools, distributed agile teams are growing in numbers, and these provide more advantages as well. Modern-day tools allow more flexibility and mobility, and organizations that do not take advantage of these may be limiting their employee base.

3.4. Assessment of the Patterns and Frameworks

3.4.1. Assessment Criteria for the Framework

Framework documentation provides a paper trail of shared understanding. There is no argument that face-to-face communication is best for reading body language, but we certainly know that there are many cases in which we have different interpretations of a conversation. With documentation, there is a paper trail that often provides what is necessary to make sure everyone is on the same page. If a misunderstanding of a requirement has occurred, the team can go back to the document for understanding rather than referring to a conversation in which there may be debate about exactly what was said. Documentation allows us to capture important details that may be forgotten or misunderstood if we only have a conversation. With today's technologies, many tools provide alternative ways of reaching shared understanding using graphical interfaces, diagrams, pictures, tests or mind maps. Sometimes the tools may even have the capability for these representations to convert to working applications. This is not to say our modern-day communication tools should entirely replace conversations; however, used in conjunction with conversations, tools may provide a more accurate representation of a shared understanding.

3.4.2. Documentation Can Be Shared

Finally, documentation can be shared with anyone. If you rely on face-to-face communication, the only way to share the information is to pass it along, and like a game of "gossip", each version is going to be a little different. Having a common document that the whole team can work from, regardless of where they are located, will prevent the misunderstandings that can occur when your primary form of communication is face to face.

3.4.3. Multiple Sources of Information

Another issue with co-located teams is that often the conversations that can happen around the whiteboard do not get transferred into a tool that everyone has access to. Agile teams often talk about the transparency that is experienced by having a big visible board that people see as they walk by.

However, only those people who walk by that big visible board will see it. By having all the important data in one place that every team member can access, regardless of where they are physically located, we are not limiting our information to only those in the room. If the latest data is only on a whiteboard, rather than in a common tool that is accessed by all, there will be more than one source of data, which can lead to confusion. This is a good reminder that documentation should not be a substitute for having the up-front conversations needed with the appropriate stakeholders.

3.4.4. Significance of Face-to-Face Communication

Face-to-face communication and co-located teams have many advantages. There is no doubt that it is easier to communicate and build strong relationships with face-to-face teaming. However, organizational leaders need to be aware of some of the issues that come from focusing so heavily on face-to-face communication, and they need to ensure they are staying current in a world which is becoming increasingly mobile. Keeping documentation up to date requires consistent, on-going effort. Distributed teams need to work harder to ensure good communication. The key is to use effective communication and collaboration that is appropriate to your situation without physical boundaries.

3.4.5. Phases in API Development

(1) Technical details

Technical details often capture high-level architectural concepts. For example, there are many structural instructions in a detailed database. Subscription or service details and relationships among these cover logical architectural issues, and technical details can also be used as a reminder for minor architectural style and design patterns. The key focus for the technical specifications is on the architects, who should be answerable for the questions related to performance, availability, security, and complaint. Project managers must support the technical details.

(2) Dependencies and relationships

When management needs mainly focus on business needs or opportunities, then attention is usually paid to how these needs can be met and whether they are related to clear, comprehensive, and project targets.

3.5. General Software Modeling Pattern

There is a recurring time period that progresses. These are some features that you want to see in a normal repeat:

a. Team formation

You must get the correct combination of product owners, designers, developers, and initial estimates. Pay attention to making the right people available and engage them in all those tasks that match the agile team member's skills. Try to avoid any changes in your team to ensure a high level of knowledge and good collaboration.

b. Transfer plan

Make sure customers and businesses are close to those who can comment on their plan.

c. Daily tests

Ensure that every member of the team is responsible for their daily development, because this is important for their overall success. Use of communication tools like Skype or Google Hangout are usually performed as a virtual stand-in for ten minutes at a specific time.

d. Process

Does each team member fully understand the process in which they are working? Do you follow traditional analysis, development code review, test, or UAT? Each developer should be well-trained. Otherwise, there are tools to provide the necessary support.

e. Meeting deliverables and objectives

Show that the project is the primary combination of deliverable iterations. The development team can present the objectives of the meeting to the clients or product owners. These objectives can be used to collect comments and ideas on what happens in the next round of development.

f. Circular iteration and team involvement

These meetings must be scheduled at the end of every iteration. This is a valuable way to examine and follow teamwork and methods for the team. Members of a good agile development team can speak openly and honestly.

3.6. Mathematical Model

Given that M assets (designers/developers) and N assignments (codes to be created) with T_{ij} time length units of each assignment are executed in asset j, we want to program the N tasks in the M assets, scanning for the best execution request that satisfies certain states of priority between the tasks characterized in P_{ik}. As characterized below, the input parameters to be utilized are introduced:

- i = 1, 2, ... N, Tasks comparing to product owner prerequisites.
- j = 1, 2, ... M, Resources or designers with various capacities.

In this work, we are working with junior, middle, and senior designers in a dynamic agile distributed development environment.

P_{xy} is the matrix of priorities between undertakings.

On the off chance that $P_{xy} = 1$ shows that task x should be finished before the beginning of task y, but generally $P_{xy} = 0$. T_{ab} is the time matrix of task execution time T_a in an asset M and ai is the total time for the task need to be executed. These calculations can be shaped based on three sorts of designers and the time required to finish an assignment.

U is a constant; if it is large enough then $U = P_{xy}\ T_{ab}$.

The main points of consideration are X_{ij}. If $X_{ij} = 1$, the undertaking (i) is relegated to the asset (j). Generally, $X_{ij} = 0$. If $Y_{ij} = 1$, this means that the undertaking task (i) is executed before the assignment (j), but generally $Y_{ij} = 0$. The start time of undertaking (i) is T_0, and the total time includes all tasks.

The model DADDM involves the accompanying conditions:

$$\text{Min } T_0 \tag{1}$$

$$T_0 \geq X_i = T_{ab} \cdot X_{ij}; \tag{2}$$

$$T_0 \geq X_i \mid X = 1; \text{ for } M_{1<=j} \tag{3}$$

$$ai \geq (ai + T_{ab} \cdot X_{ij}) \cdot P_{xy} \tag{4}$$

$$Y_{ki} + Y_{ik} = 1 \tag{5}$$

$$ai \geq (ai + T_{ab} \cdot X_{ij}) - U \cdot (3 - Y_{ij} - X_{ij} - X_{1.=k}); \tag{6}$$

Equation (2) characterizes the longest time among all designers. Equation (3) shows that each undertaking must be relegated to a developer. Statement (4) ensures that the non-covering time between two undertakings has priority. Statements (5) and (6) guarantee non-covering time. Based on the above, we have formalized an assignment arranging process for the product improvement procedure using light-footed techniques (integrated DADDM with Agile, for this case). We have examined this model for many parameters, and we obtained highly positive feedback from the customers.

3.7. Dynamic Agile Distributed Development Method

We are proposing a DADDM that is one of the most valuable methods of distributed agile project management. The DADDM specializes in information system projects that involve barriers such as extraordinary budgets and very strict dimensions. We have designed DADDM to solve common issues that often cause failures of information system projects, such as

- Unsuccessful delivery;
- Lack of user engagement;
- Administrative issues;
- Cost and budget.

The DADDM is different from agile Kanban and other software development frameworks because it focuses on product delivery compared to team activity in an agile manner (although team support is important in any other product activity process).

3.7.1. Dynamic Agile Distributed Development Method Key Concepts

The basic principles and features that were used to create the DADDM structure were:

- Before work can begin, the users should actively monitor the accuracy of the decisions involved in product development;
- The team must be self-sufficient to make project development and mandatory management and administrative decisions to achieve better results in the distributed environment. There should be a minimum of red tape by management in decisions that the team takes;
- Products will be developed frequently in the initial steps. Early and fast delivery of the product will make sure that users get products and give feedback;
- Product repetition is done in the process of maximizing user feedback and product improvement;
- The product development must follow the necessary path, so if any change is needed, it can be implemented in development;
- During the pre-project phase, the needed capacity and requirements of this project will be explained before the development begins;
- The workflow process works with integrated testing as its main objective. Products are processed during each stage of development;
- Effective communication and cooperation between all the basic teams, management, and stakeholders are necessary for the development process.

3.7.2. DADDM Development Life Cycle

The DADDM framework basically has three stages:

- Pre-project phase;
- Project life cycle;
- Post-project phase.

Medium phase in the project life cycle is the biggest phase of the whole framework. Specific roles, responsibilities, and activities at each stage are described below.

3.7.3. Pre-Project Phases

The pre-planning step occurs before the project starts. Management settings and project requirements are described to provide insight into the project. Figure 3 represents DADDM framework.

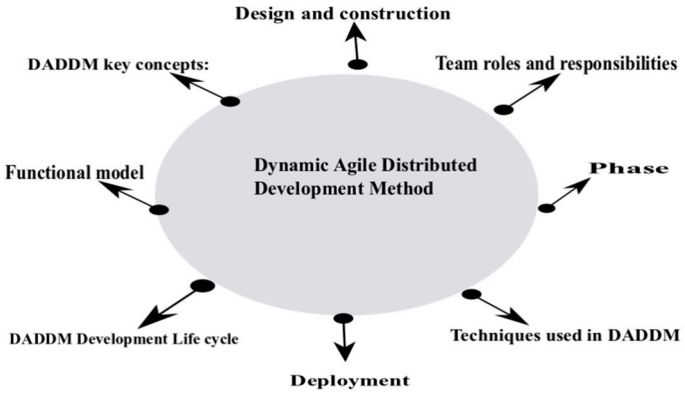

Figure 3. Dynamic Agile Distributed Development Method Framework overview.

3.7.4. Project Life Cycle Phase

Once the project is started and all the necessary details have been decided, the project enters into its life cycle phase. The project life cycle phase is applied in two steps:

- Feasibility study;
- Business study.

These two steps must be followed in an orderly manner. That is, we must first perform a feasibility study. The results of the feasibility study promote the execution of the second step, i.e., the business study.

3.7.5. Feasibility Study

In feasibility studies, the team members coordinate with each other and determine the answers to many questions such as:

- Do established ground rules and project planning result in the production of working products in time?
- Will the fixed budget be enough to complete the work?
- Is the project increasing our market value?

Since DADDM is applicable to information systemssolutions where time is a light commodity, the feasibility as well as the business study stage are conducted as quickly as possible.

3.7.6. Business Studies

In the business Studies stage, the team and information about the business aspect of the project are assessed to generate quick surveys.

- What are the user requirements?
- What specifications should be kept to maintain the business value of the project?
- What technology will be used to meet the business quality of the product?
- What skills will be needed to test and verify the product?

3.8. Use Case: Using DADDM to Develop a System for Predicting Flood Level

We used an example of the development of a software project that deals with flood level detection and an alert generating system, using the concept of dynamic agile distributed development. The initial required engineering results show that the client wants to use this software for provisional disaster management authority flood data collection. In this case, they will visit the villages in the divisional regions and then collect the data. The collected data will then be cross-checked and verified to

identify any possible mistakes. After receiving the initial requirement from the clients with face-to-face communication, the team starts software development. We are using the two software engineering models in this case study. The first is the Normal Agile Model, and the next one is our proposed DADDM. We have 15 team members, including the designers, developer, two managers, and other supported staff members. The cost allocated for this project is 100 dollars, and the time allowed is one month. For the agile development, we completed the task in 25 days within the cost allowed, and the customer was satisfied (the available options are dissatisfied, normal, above normal, well satisfied, and excellent). In the case of the DADDM, we completed the software at 19 days with the same team and budget. The client response was well satisfied. We applied the proposed three phases of the model: pre-project phase, project life cycle, and post-project phase.

3.9. Functional Model

The development team starts with the construction of the initial prototype. An active prototype of the product is the initial version of this function, and it must contain a finished working model.

The prototype is produced following a cycle:

- Active investigation;
- Prototype returns;
- Strengthened prototype.

After this, the prototype is reviewed for:

- Use;
- Performance;
- Capacity;
- Frequency;
- Techniques.

3.9.1. Design and Construction

The product is designed in the design and construction phase. A design model is developed and coded according to the mission statement and is then examined and reviewed. Product design and development occur continually.

3.9.2. Deployment

After reviewing and testing the product, the next step is to deploy the project to the client. A review document is also generated, where the points and actions obtained from the product requirements are obtained. Developers have learned lessons from consumers using the full version.

3.9.3. Post Project Phase

The post project phase confirmed that the end version of the product works efficiently. The DADDM principles are used to ensure the quality of the team's work product; they are also used if there is a need for improvement; then, the product is forwarded to further development in the next phase. Figure 4 shows the complete working model.

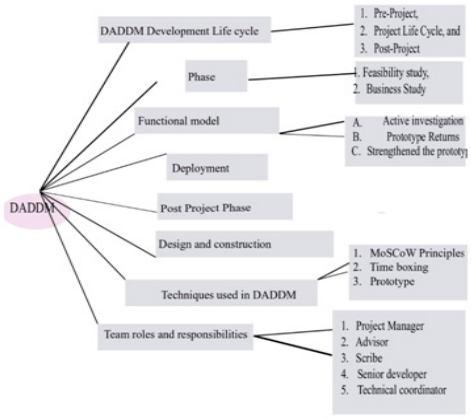

Figure 4. Complete operation of the proposed model DADDM.

3.9.4. Techniques used in DADDM

(1) MoSCoW Principles

The DADDM facilitates projects that do not cover luxury timeframes and flagged funds. Therefore, work blocks are preferred in the development phase. The MoSCoW rule is applied to enhance the importance of functionality:

- Required—Essential things;
- Must be—Essential things for business solutions;
- Maybe—useful things that do not promote problematic bugs and can replace other preferences;
- Will not work—things that are important and can wait until later.

(2) Time boxing

Instead of building combinations on the project road map, DSMD uses time boxing in which an initial cycle consisting of three to six weeks is prepared during which complete testing of tasks and goals is completed.

(3) Prototype

Prototype is the end of the DSDM. If the team is not using lots of prototypes as part of the development process, they are not operating according to the DSDM approach.

Prototypes guarantee the following:

- User contributions and opinions;
- Continual product delivery;
- Incremental product release.

The DADDM classifies prototype activity in four groups:

1. Business Prototype—evaluates the business value angle version;
2. Performance Prototype—ensure user-solving solutions;
3. Prototype of Use—what fault and user interface is free?
4. Ability Prototype—is there room to expand the product?

(4) Team roles and responsibilities in DADDM

1. Project Manager—Monitoring and management of prototype and development process;
2. Advisor—Information expert with practical knowledge about automatic or modifying areas;
3. Scribe—Member who handles all project discussions, plans, results, and objectives;
4. Senior developer—An advanced developer that helps create an access code to provide unusual skills and software development information;
5. Technical coordinator—Responsible for taking care of the technical aspects of the project.

It should be ensured that every person is aware of the technical details of their role and how to execute them.

4. Conclusions

Agile distributed development procedures consist of various frameworks that focus on a wide range of project management methods. Compared with the previous results, the proposed approach of an integrating DADDM model joins different tasks during the prerequisite assembling and configuration stages. It is a novel method that can work in general settings and is not limited to situational ADSD. As requirements are constantly changing, the models ought to be refreshed to integrate the new tasks as they are characterized. Additionally, the agile phase used during the requirement elicitation stage can be the base for the models made during the post-project stage.

The proposed method fixes the responsibilities of every team member. The existing frameworks, patterns, and configured APIs for the agile-based distributed development of software have been evaluated. Key tasks for future research should be a comparison of DADDM with agile and Extreme Programming as there is the need for a framework that incorporates the requirement among researchers and practitioners in classifying CSFs for software development projects. The next step is the evaluation of CSF's impact on our proposed framework—DADDM.

Author Contributions: Conceptualization, M.A.N. Methodology, M.A.N. and S.U.-J.L.; Investigation, S.U.-J.L.; Resources, S.Lee; Data Curation, M.A.N.; Writing—Original Draft Preparation, M.A.N.; Writing—Review and Editing, M.A.N.; Visualization, M.A.N.; Supervision, S.U.-J.L.; Project Administration, S.U.-J.L.

Funding: This research received no external funding.

Conflicts of Interest: There are no conflicts of interest.

References

1. Alzoubi, Y.I.; Gill, A.Q.; Al-Ani, A. Empirical studies of geographically distributed agile development communication challenges, A systematic review. *Inf. Manag.* **2016**, *53*, 22–37. [CrossRef]
2. Alqudah, M.; Razali, R. A review of scaling agile methods in large software development. *Int. J. Adv. Sci. Eng. Inf. Technol.* **2016**, *6*, 828–837. [CrossRef]
3. Permana, P.A.G. Agile method implementation in a software development project management. *Int. J. Adv. Comput. Sci. Appl.* **2015**, *6*, 198–204.
4. Vlietland, J.; van Solingen, R.; van Vliet, H. Aligning codependent Agile teams to enable fast business value delivery, A governance framework and set of intervention actions. *J. Syst. Softw.* **2016**, *113*, 418–429. [CrossRef]
5. Bass, J.M. How product owner teams scale agile methods to large distributed enterprises. *Empir. Softw. Eng.* **2015**, *20*, 1525–1557. [CrossRef]
6. Anwer, F.; Aftab, S.; Shah, S.M.; Waheed, U. Comparative Analysis of Two Popular Agile Process Models: Extreme Programming and Agile. *Int. J. Comput. Sci. Telecommun.* **2017**, *8*, 1–7.
7. Qureshi, M.R.J.; Ikram, J.S. Proposal of Enhanced Extreme Programming Model. *Int. J. Inf. Eng. Electron. Bus.* **2015**, *7*, 37.
8. Anwer, F.; Aftab, S. SXP: Simplified Extreme Programing Process Model. *Int. J. Mod. Educ. Comput. Sci.* **2017**, *9*, 25. [CrossRef]
9. Lei, H.; Ganjeizadeh, F.; Jayachandran, P.K.; Ozcan, P. A statistical analysis of the effects of Agile and Kanban on software development projects. *Robot. Comput. Integr. Manuf.* **2017**, *43*, 59–67. [CrossRef]
10. Ahmad, M.O.; Kuvaja, P.; Oivo, M.; Markkula, J. Transition of software maintenance teams from Agile to Kanban. In Proceedings of the 2016 49th Hawaii International Conference on System Sciences (HICSS), Koloa, HI, USA, 5–8 January 2016; pp. 5427–5436.
11. Bass, J.M. Artefacts and agile method tailoring in large-scale offshore software development programmes. *Inf. Softw. Technol.* **2016**, *75*, 1–16. [CrossRef]
12. Shrivastava, S.V.; Rathod, U. Categorization of risk factors for distributed agile projects. *Inf. Softw. Technol.* **2015**, *58*, 373–387. [CrossRef]

13. Ebert, C.; Paasivaara, M. Scaling agile. *IEEE Softw.* **2017**, *34*, 98–103. [CrossRef]
14. Razzak, M.A. Knowledge Management in Globally Distributed Agile Projects—Lesson Learned. In Proceedings of the 2015 IEEE 10th International Conference on Global Software Engineering, Ciudad Real, Spain, 13–16 July 2015; pp. 81–89.
15. Gill, A.Q.; Henderson-Sellers, B.; Niazi, M. Scaling for agility, A reference model for hybrid traditional-agile software development methodologies. *Inf. Syst. Front.* **2018**, *20*, 315–341. [CrossRef]
16. Ahimbisibwe, A.; Cavana, R.Y.; Daellenbach, U. A contingency fit model of critical success factors for software development projects, A comparison of agile and traditional plan-based methodologies. *J. Enterp. Inf. Manag.* **2015**, *28*, 7–33. [CrossRef]
17. Cooper, R.G.; Sommer, A.F. Agile–Stage-Gate for Manufacturers: Changing the Way New Products Are Developed Integrating Agile project management methods into a Stage-Gate system offers both opportunities and challenges. *Res.-Tech Mgt.* **2018**, *61*, 17–26. [CrossRef]
18. Hashmi, A.S.; Hafeez, Y.; Jamal, M.; Ali, S.; Iqbal, N. Role of Situational Agile Distributed Model to Support Modern Software Development Teams. *Mehran Univ. Res. J. Eng. Technol.* **2019**, *38*, 655–666.