Distributed Agile Patterns: An Approach to Facilitate Agile Adoption in Offshore Software Development

Maryam Kausar

College of Science and Technology
School of Computing, Science and Engineering
University of Salford, Manchester, UK

Submitted in Partial Fulfilment of the Requirements of the
Degree of Doctor of Philosophy
October 2017

Table of Contents

List of Tables

List of Figures

Acknowledgements

First and foremost, this research would not be possible without the blessing of my Lord ALLAH, the most Merciful, who has given me the patience and strength to finish this PhD journey.

I would like to express my gratitude and sincere appreciation to my supervisor, Dr. Adil Al-Yasiri, for his assistance, guidance, and feedback during my PhD.

Special thanks to my parents, Kausar Iqbal Malik and Khalida Parveen, for their encouragement and prayers.

I would like to extend my thanks to the members of the School of Computing, Science, and Engineering at the University of Salford, for supporting me during my PhD.

Finally, I would like to say thanks to everyone who helped me achieve success in this research.

Declaration

Parts of the research presented in this thesis has been published in the following papers and presentations:

1- Kausar, M., & Al-Yasiri, A. (2015). ***Distributed Agile Patterns for Offshore Software Development.*** Presented in the 12th International Joint Conference on Computer Science and Software Engineering (JCSSE 2015). IEEE Conference 2015.

2- Kausar, M., & Al-Yasiri, A. (2016). *Using Distributed Agile Patterns for Developing Offshore Projects*. Presented in the Proceedings of the CSE 2016 Annual PGR Symposium (CSE-PGSym 16), University of Salford, Manchester, United Kingdom, 2016

3- Kausar, M., & Al-Yasiri, A. (2017). *Using Distributed Agile Patterns for Supporting the Requirements Engineering Process*. Presented in the Requirements Engineering for Service and Cloud Computing. Springer International Publishing, 2017. 291-316.

Abstract:

Over a decade, companies have been using agile methods for the development of software. However with the increasing trends of offshore software development, companies are becoming more interested in using agile methods for such projects. While offshore development has several dynamic benefits such as cost reduction, flexibility, proximity to market, concentration on core processes and easy access to talent, they have introduced new challenges, such as trust, socio-cultural, communication and coordination, and knowledge transfer issues. These challenges not only affect the development process but also affect the applicability of agile practices in offshore development. As a consequence, companies have been modifying and adapting agile practices to overcome these challenges. However there has been little effort put to collect and document the common practices that have been used repeatedly to solve recurring problems in offshore development.

Using the systematic literature review approach and applying customised search criteria based on the research questions, we identified and reviewed over 200 cases from literature. As part of this research we also conducted semi-structured interviews, in which we involved practicing professionals who were working with distributed teams. As a result, we identified and documented a number of solutions to address the common agile issues in software development, which we classified as distributed agile patterns.

This research presents the challenges caused by offshore development, how they affect the applicability of agile practices in offshoring. We have then developed a catalogue containing the identified fifteen distributed agile patterns and have classified them into four categories. We have used a case study to explain how these patterns can be applied in offshore software development. To verify and validate our catalogue, we conducted a reflection workshop, in which we invited professionals to review and comment on the patterns. The participants engaged in reviewing the patterns and gave constructive feedback, which helped in improving the catalogue.

Based on their feedback, the distributed agile patterns catalogue was finalised. The catalogue can help practitioners make a more informed decision while choosing agile for their offshore projects.

Chapter 1 Introduction

Recent advances in software development have made it possible and attractive for the industry to move towards Global Software Development (GSD). GSD is referred to the practice in which companies move some/all part of their software project to an offshore location. For some companies the motivation is to cut down on cost, while for others the motivation is to benefit from the variety in talents available from all over the world, or closeness to market. Nonetheless, whatever the reason was to making such a decision, it is very important for them to consider which type of global software engineering business model they will be using. Work has been done on proposing different frameworks from which organisations can select which type of GSD model they want to use (Robinson et. al, 1996; Prikladnicki et al. 2007 and Agerfalk et al. 2008). We have presented an overview of different models in Section 2.2.1.

The term "offshore outsourcing or offshoring" is commonly used for the practice of a company to subcontract work to third party companies, which are placed at countries like India, Pakistan, China and Russia (Prikladnick et al., 2007). This way, a company can focus on its core business and leave the development of products to companies that can perform them in higher standards or at cheaper rates (Pilatti et al., 2006).

The concept of offshoring isn't new or exclusive to software development; many companies for years have been using subcontractors to provide assistance in multiple areas, from cleaning services to consultations (Jahns, et al., 2006). What has changed is that with the help of technology the scope of offshoring has expanded internationally (Van, 2004). As technology has reduced the cost of communication, companies can exchange work products to far away places, cheaply and efficiently.

A decade ago the trend of offshoring had increased mainly in the sector of manufacturing (Garner, 2004) but as communication became cheaper and faster more companies from different sectors started adopting the practice. In the IT sector, offshoring has been changing how organisations develop software around the world.

The main reason why companies choose offshoring is because of lower production cost (Pilatti et al., 2006) which is mainly because of low salaries in countries outside Europe, the US and Japan. For example in India an average ICT worker earns one seventh of the amount earned by a British employee (Global Wages Comparison, 2013).

Offshoring can offer additional commercial advantages beyond cheaper labour; for example in a country like India, which profits 90% of the revenues of all IT and service offshore activity, produces two million university graduates per year (Kobayashi-Hillary, 2005). With this offshore companies can see new opportunities such as access to larger pool of skilled people, shared best practices and proximity to markets and customers (Smite et al., 2011). Kobayashi-Hillary who went to Bangalore to open a software facility comments "The issue now was not finding good people; it was filtering the excellent from the good" (Kobayashi-Hillary, 2005).

There are many advantages of offshore development but as GSD continues to grow it has been noted that it causes some issues, which are caused as teams become distributed over different geographical locations (Damian et al., 2006). Issues such as trust, socio-cultural, communication and co-ordination and knowledge transfer are examples of the challenges facing GSD. On one hand such issues represent barriers to the distributed teams; on the other hand it has particular impact on Agile teams which favour face-to-face communication and co-working between team members. A number of teams have experimented and adapted various agile practices in order to overcome the above challenges. Such efforts (although documented) remain individual and difficult to share. In this research we have provided solutions for some of these challenges by identifying agile practices that are being used repeatedly for addressing global software development issues. We believe that by identifying and cataloguing these practices they can be easily retrieved and applied for similar problems.

1.1 Research Problem

Traditionally, software was developed at one location where the whole team worked from the same office. Similarly the clients/sponsors of the project were situated in the same country. But as onshore software development became expensive companies started looking for alternative ways to cut down on software development cost (Pilatti, Audy, 2006). One such alternative was to move some of their processes to offshore locations, as due to decrease in global communication rates companies could coordinate work via online communication tools and setup cost in countries like India and China is cheaper in comparisons to countries like USA, UK and Japan (Prikladnick & Audy, 2012; Bush, Tiwana, and Tsuji, 2008; McLaughlin, 2003)

But as companies continued to switch to offshore development it became clear that achieving reduced cost is not as straightforward as it initially seemed. As even if companies manage to cut down cost on development by setting up cheap offices and finding cheap labor (Global Wages Comparison, 2013), offshoring introduces new challenges. These challenges appear because companies distributed their team to different locations having different time zones, different language and social values, causing challenges such as:

- The issue of trust causes difficulty in forming alliances among firms that do not know each other. It also introduces a risk of project failure as clients are concerned with whether the offshore service provider will deliver the work promised without compromising on their requirements. Lack of trust can cause misunderstanding and conflicts between the firms (Battin et al., 2001). This issue will be discussed in detail in section 2.3.1.

- Socio- Cultural conflicts, such as a difference in languages, national traditions, values and norms cause problems in offshore software development (Carmel, 1999; Hofner et al., 2007). As the difference in language or language style can cause problems while developing the code as the offshore team leaves comments for the onshore team regarding what they have done but due to

difference in language the onshore team cannot understand their comments hence resulting in a delay in the project development. This issue will be discussed in detail in section 2.3.2.

- The team faces communication and coordination issues (Beulen et al., 2002), as the team members are distributed on different locations and time differences, which makes it difficult to have real-time face-to-face communication. This can cause problems in sharing information. This issue will be discussed in detail in section 2.3.3.

- Companies started to face knowledge transfer issues they moved their business and development process to offshore locations, as any mistake in the transfer process can cause projects to fail (Kedia et al., 2007). This issue will be discussed in detail in section 2.3.4.

These challenges were not part of traditional software development because the team worked in the same country having the same time zone, face-to-face communication, and same social values (Meyer, 2006). If these challenges are not handled properly they can make the overall cost of the project several times higher than if the project was developed at any onshore location (Iacovou et al., 2008; Hartmann et al.,2011).

In offshoring, companies also have to deal with additional tasks as the team is distributed over different time zones (Evaristo et al., 2004). The additional tasks are:

i) They have to maintain good communication with all its offshore offices otherwise the information flow will be affected (Carmel et al., 2005),

ii) The onshore office has to decide which work/project should be sent to offshore location and which should be developed in-house (Lamersdorf et al., 2008) and

iii) Once the project or process is selected they need to be coordinated with the onshore office in order to develop a product that meets the clients requirements (Aundh et al., 2009).

Another problem with offshoring is that the development methodology for offshore software development is different from the traditional approach as now companies can either follow a global standard methodology for all its offices or it can have different development methodologies for distributed locations (Sengupta et al., 2006).

Similarly the standard development methodologies starting from waterfall development life cycle to agile assume that the team is located at the same location. So if a company applies any of the standard development techniques by the book, offshore projects are bound to fail. Many companies have customised agile methodologies for offshore projects, but there are still limitations to how successful those projects will be as agile methodology focuses on face-to-face meetings such as daily-stand up meeting and retrospective meetings (Beck et al., 2001).

The above challenges can directly cause problems in the software development life cycle, which can result into a project to fail; more discussion is provided in Section 2.4. The problem that this research will tackle is how to overcome the identified challenges between the onshore and offshore teams as it affects agile practices on offshore projects. This is particularly important, as the emphasis of agile methodologies is on face-to-face meetings such as daily-stand ups, sprint meetings and sprint review meetings (Benk et al. 2001), which are difficult to conduct in offshoring as the team is distributed geographically in different time zones. Similarly agile focuses on the team to be self-organised (Benk et al. 2001) but due to lack or delay in communication between the onshore and offshore team, it is difficult to distribute work among the team members efficiently. These issues are discussed in detail in Section 2.5.2.

1.2 Research Aim and Objectives

In order to address the challenges between onshore and offshore teams such as trust, socio-cultural, communication and coordination, and knowledge transfer issues, this research aims to propose a pattern based approach by identifying repeating solutions which have been proven in overcoming those challenges so that practitioners can apply this approach in order to mitigate the effect of the challenges in their projects. In addition to the main goal of the research, we have four specific objectives and they are summarised as the following:

i. To conduct systematic literature review to understand and analyse the existing literature on offshore software development. This will build up the research foundation and form the research questions for this research.

ii. Identifying the key challenges from literature that occur while developing software on offshore locations and investigate how agile practices can be used to overcome those challenges and. Based on literature and interviews identify if there is a recurring agile practices being used to solve the same challenge that we can classify as a pattern.

iii. Develop a catalogue of the identified patterns that will be documented, as it will focus on facilitating offshore software development projects by providing as a guideline for practitioners that want to offshore their projects.

iv. Validating the Distributed Agile Patterns catalogue by using Kerth's reflection workshop method (Kerth, 2001) and evaluate the catalogue by comparing the existing solutions presented in literature.

1.3 Research Questions

During the course of this research, a number of questions have been identified to be addressed:

RQ: What are the recurring adaptations of agile practices that are being used within offshore software development in order to address the identified issues?

To be more specific, the study's focus was on the following two questions:

RQ1: What are the agile practices that are being commonly used to deal with offshore challenges?

RQ2: Are the challenges identified in RQ1 recurring in offshore software development?

1.4 Research Contributions

There is a lack of effort in collecting common practices that have been used repeatedly to solve recurring problems in offshore development. In this research we have studied over 200 cases from the literature and interviewed practicing professionals that work in distributed teams. As a result we have observed a number of solutions for agile issues in distributed development settings, which we have classified as Distributed Agile Patterns. We defined **distributed agile patterns** as adaptation of an agile practice that is being repeatedly applied in order to solve a recurring challenge in a distributed project scenario. The difference between distributed patterns and distributed agile patterns is that distributed patterns only focus on practices that are being repeatedly applied in order to solve a recurring challenge in an offshore project irrespective of if those practices are agile or not whereas distributed agile patterns focus on repeating agile practices.

This research confirms that there are four main challenges in offshore software development, which are trust, socio-cultural, communication and coordination, and knowledge transfer issues, that every organisation faces and that they have adapted agile practices to overcome those challenges.

The findings of this research will strengthen the existing literature on distributed software development and provide a knowledge base for agile practitioners who use or intend to use agile approaches in offshore software development. The documented findings have been observed from literature and practitioners in order to prepare the patterns catalogue.

Practitioners can use the pattern catalogue in the beginning of offshore projects and make informed decisions about how to adopt agile approaches as generalized patterns make it easier for other companies to reflect on and to apply the results to their own cases.

1.5 Research Methodology

The following research methodology is developed and adopted for this research. The main stages of the methodology are outlined in Figure 1.1 and detail of each stage is presented in Chapter 3 Research Methodology.

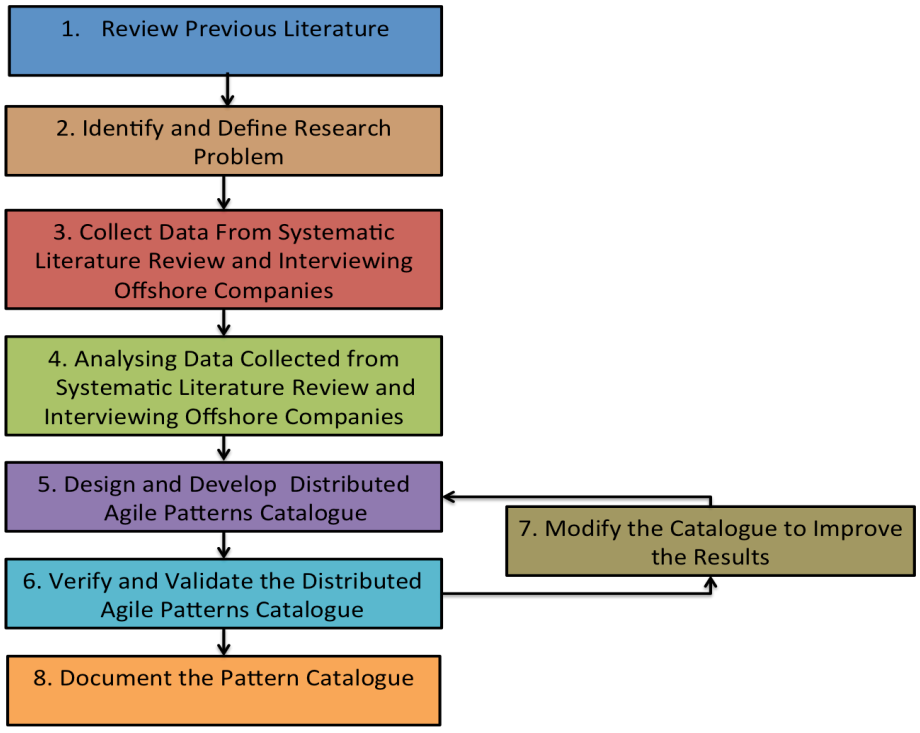


Figure 1.1 Research Methodology

Step 1: Review Previous Literatures and Relevant Offshoring Challenges.

In this step the previous relevant works is reviewed, to identify challenges practitioners faces while developing their projects offshore. This helps in acquiring a good understanding of what factors affect offshore projects. This step also helps in answering the RQ1 mentioned in Section 1.3, as there was a need to study what agile practices are being used in GSD.

Step 2: Identify and Define the Research Problem.

The research starts by making assumptions on how to successfully develop software on offshore locations to meet the objectives of the research. To do this, the research focuses on studying and analysing factors that affect offshore software development, based on the case studies of different companies as this helps in answering the RQ1 mentioned in Section 1.3 in order to identify what agile practices are being used and also to answer *RQ2* that is to identify if those practices are recurring or not.

To perform this step semi-structured interviews are conducted with companies who had chosen offshore processes for their projects. The research also analyses how many companies use agile development methodologies for their offshore software development projects and what limitations they face.

Step 3: Collect Data from Systematic Literature Review and Interviewing Offshore Companies.

This research is carried out by, following Kitchenham's guidelines for conducting Systematic Literature Review (Kitchenham et al. 2007). This is done to select primary studies from the existing literature. We also conducted 20 semi-structured interviews with offshore companies to get practitioners point of view on the challenges they face while working on offshore projects.

Step 4: Analyse Data collected from Systematic Literature Review and Interviewing Offshore companies.

To analyse the data collected we use content analysis as proposed by Kripendorff to verify the results of our findings (Kripendorff, 2004). Details of this step is presented in section 3.3.2.

Step 5: Design and develop the Distributed Agile Patterns Catalogue.

Based on the findings a Distributed Agile Patterns Catalogue is developed; this is presented in Section 4.5.

Step 6: Validate and Evaluate the Pattern Catalogue.

In order to validate the Distributed Agile Patterns Catalogue, we conducted a workshop by using Kerth's keep/try reflection workshop method (Kerth, 2007). By doing this, feedback collected from the companies (who would have been interviewed) is sought to obtain their views on the catalogue, and its effectiveness. Based on their answers the catalogue is validated and this helps in answering the *RQ2* mentioned in Section 1.3 as it confirms that these agile practices are being used to solve recurring problems in GSD.

To evaluate the pattern catalogue we compare the catalogue with other solutions mentioned in the literature for solving GSD challenges, which is presented in Section 5.3.

Step 7: Modify the Catalogue to Improve Results.

The necessary modifications are made in the catalogue based on the feedback given by the companies. This step allows the research to improve the results produced in the catalogue.

Step 8: Document the Pattern Catalogue.

The final stage of the research is to document the patterns catalogue, which incorporates the answers of all the research questions mentioned in Section 1.3.

Based on the above discussion it can be seen that we have used an inductive approach, as the research moves from specific to general. As we first identified and designed a specific research problem and then developed a generic catalogue of distributed agile patterns. In section 3.3.2 we have presented detail explanation on how we have applied inductive approach in our research approach.

1.6 Structure and How to Read the Report

This thesis presents the complete Distributed Agile Patterns Catalogue. The remainder of the thesis is arranged in the following chapters. Chapter 1 presents the research aim and objectives, the research problem, the research questions we will answer in this study and the overview of the research methodology that is carried out to answer the aim and objectives set for this study. Chapter 2 Offshore Software Development, presents the background of offshore software development which concentrates on describing different types global software development models as well as what are the benefits and challenges of offshore software development and what role agile plays in overcoming those challenges. It also focuses on how patterns can be used to overcome the challenges of offshore development. In Chapter 3 we present the research methodology that has been followed in order to conduct this research. Chapter 4 consists of the final version of the Distributed Agile Patterns catalogue, as we didn't want to confuse the reader by presenting two versions of the catalogue, the unrevised version can be found in Appendix F. Chapter 5 presents how the catalogue is validated and evaluated with the use of reflective workshop and comparing other solutions present in literature for offshore software development. To help practitioners understand how the Distributed Agile Patterns can be applied, Chapter 6 presents a case study on how the requirement phase is conducted using the distributed agile

patterns catalogue and finally in Chapter 7, the conclusions has summarised the findings of this research and we have presented an overview of the future work.

1.7 Chapter Summary

This chapter presented an introduction to the thesis through defining the research problem and discussing the research motivation, then the aim of the research, the objectives, and research questions were presented. The research contributions have been defined along with the research methodology, in addition to defining the outline of the thesis.

In summary, this thesis presents a catalogue of distributed agile patterns which practitioners can use while offshoring their projects. In the next chapter, a background for the research field is presented.

Chapter 2 Offshore Software Development

2.1 Introduction

This section summaries the work that has been completed to-date. It presents the background research that has been done on offshore software development so far. The background involves reviewing the models of offshoring and the benefits it provides. The review also covers and highlights the main challenges and issues offshore software development which we identified using Systematic Literature Review. We have also mentioned how agile can be used to overcome those challenges and what are the limitations of agile. We have also presented five approaches used in literature to facilitate practitioners in agile adoption. Lastly we have mentioned how patterns can aid in agile adoption in offshore software development by presenting an overview of current patterns being used for agile adoption and giving an overview of the distributed agile patterns.

2.2 Background of Offshore Software Development

In this section we have presented a background study on offshore software development to understand why organisation move towards offshoring their processes. We gave an overview of different types of offshore models based on what type of business model they chose to follow while offshoring their projects. We also presented what are the key benefits that organisations consider while deciding to go for offshore software development.

2.2.1 Overview of Offshore Models

When an organisation decides to move some of their processes to other locations, it is very important for them to consider which type of global software engineering business model they will be using. Robinson provided a framework to categorise these

business models based upon relationship structure and geographical location (Robinson et al., 2004).

Prinkladnicki used an adaptation of Robinson's framework based on the most common models used in practice, which was than presented in Agerfalk and Fitzgerald work (Prinkladnicki et al., 2007; Robinson et al. 2004; Agerfalk et al., 2008). In this thesis we have used a further adapted version with inputs from the Organisation for Economic Co-operation & Development report (OECD, 2004) shown in Figure 2.1. The reason for choosing this version is because it clearly identified the four main business models selected by organisations that decide to move their processes to a different location, where its in the same country or an offshore location. This section discusses each of these models in detail.

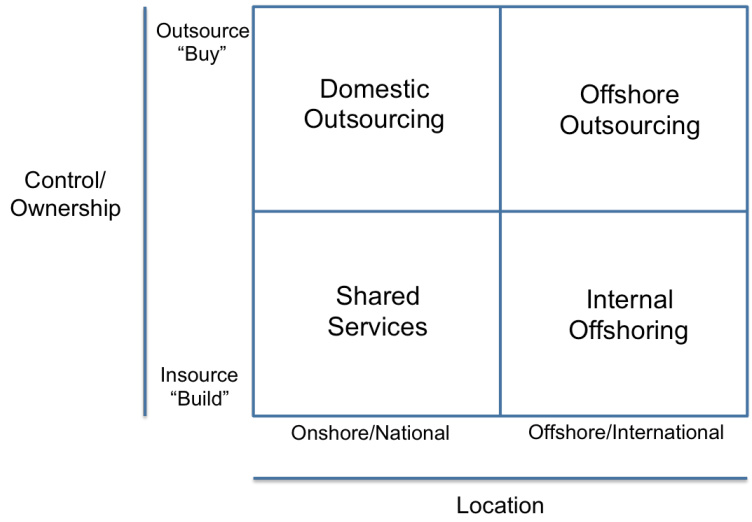


Figure 2.1 Business Models for Global Software Engineering (OECD, 2004).

2.2.1.1 Domestic Outsourcing

Domestic outsourcing is also referred as onshore outsourcing (Robinson et al., 2004). In this business model an external company acts as a subcontractor for providing software development services or software products to an organisation. In this scenario the subcontracting company is located onshore (Prikladnicki et al., 2012). The

companies that select this type of business model go for a joint-venture offshoring approach.

In joint venture offshoring, an organisation collaborates with a local company to develop software. This collaboration can take many different forms. In some cases the revenue stream is separated from rest of the company's business as the servers being outsource generates its own revenue which results in reduced risk for the company offshoring their services. Both the company share a percentage of the revenue earned.

In other cases, joint ventures can be between two or more companies, the goal being to build an offshore centre with multiple owners in order to reduce the start-up costs and operational risks. Hence in joint ventures companies' collaboration can be equal that is both of the company have equal stake or the companies can be independent but just contribute their resources to each other. The goal of this model is to benefit from the strengths of the organisations, in order to achieve a win-win situation for both of them (Babu, 2005).

To motivate both the companies to go for joint venture contracts, many companies include a build-operate-transfer (BOT) clause. This clause helps the companies to decide on how they will work together in order to complete the products (Vashistha et al., 2005) and furthermore it also allows the onshore companies to sell their stakes to foreign companies after achieving agreed-upon milestones.

Joint venture offshoring has many advantages such as both the companies learn from each other. It also helps build relationships across the venders and both the companies share their recourses to develop the best product (Babu, 2005).

2.2.1.2 Shared Services

Shared Services is also referred as onshore insourcing. In this business model there is a department in the company's building or a subsidiary in the same country that provides software development services for the internal projects (Prikladnicki et al.,

2012). This model is the most simple and known business model as not many risks are involved in adopting it since all the teams are working in the same country (Prikladnicki et al., 2007).

Many companies choose onshore insourcing in order to avoid the risk of moving their processes to an offshore location. A study done by Amiti in the United States, the United Kingdom, and many other industrialised countries showed that jobs in these countries are preferred to be insourced rather than to outsource them because they are concerned that if they outsourced more jobs they will be losing a net amount of jobs for their nationals (Amiti et al., 2005).

Onshore insourcing is considered the better alternative to outsourcing as the jobs stay in the country and they do not have to invest in the start-up costs (Prikladnicki et al., 2012). Similarly companies can avoid challenges such as communication and coordination problems, language issues, software quality standard issues and trust issues (Prikladnicki et al., 2007).

2.2.1.3 Internal Offshoring

Internal Offshoring is also referred as offshore insourcing. In this business model, a company internally offshores its services by creating its own software development centre (subsidiary) in foreign country (Prikladnicki et al., 2012). Some companies opt for the ultimate approach, “do it yourself ” that is go out and built your own subsidiary centre in an offshore location (Vashistha et al., 2005). Many companies go directly in for a subsidiary or local offices instead of opting for joint ventures, as there management is comfortable in dealing with international and local market operations. The common terms used for this kind of model include offshore development centre (ODC), captive development centre, branch or local office.

Companies usually use such a model when they already have very large physical presences in the countries involved in the offshoring. Sometimes the captive centres

run independent businesses with their own budget and bottom-line accounting. GE, HSBC, and American Express are considered the most sophisticated at deploying this model. Another example is Citibank who does commercial banking in Brazil and Portland.

The advantages of this model are, it guarantees market for a company's services and it helps in establishing management hierarchy. It also alleviates some organisational issues such as control and politics that manage the back-office offshore activities with external vendors (Robinson et al., 2004). This approach has one key challenge that apart from internationalisation and localisation of business management, managers are concerned about finding the right staff, line works, technical experts and line managers from multicultural backgrounds (Aron et al., 2005).

Many large organisations are comfortable in managing their technology development and innovation at local offices. Large software development companies including IBM, Microsoft and Oracle already have global marketplace and are comfortably moving work around the world (Babu, 2005).

2.2.1.4 Offshore Outsourcing

Offshoring Outsourcing is also referred, as offshoring. It is the most commonly used business model by companies. In this business model external suppliers that are located in other countries (offshore) provide software development services (Prikladnicki et al., 2012). Carmel defines offshoring as moving business processes that were being done at a local company to a foreign country in order to take advantage of cheap labour (Carmel et al., 2005). While selecting this business model companies can choose from three different approaches such as:

i. Service-provider offshoring,
ii. Dedicated centres and
iii. Third-party transplant.

In service-provider offshoring, companies subcontract development work to offshore organisations in order to focus on core business activities (Vashistha et al., 2005). The advantages of choosing this type of offshoring rather than building subsidiary outlets is similar to the advantages provided by offshoring rather than keeping all the functionalities in house such as getting product developed at low cost and concentrating more on core business activities (Vashistha et al., 2005). Hence in order to avoid the risks attached with offshoring and capitalize from the benefits it provides companies choose to outsource projects, program, and individual work orders to offshore vendors.

This model is becoming increasingly popular and it encompasses a wide range of work including small projects to multi-year contracts containing of budget up-to millions of dollars (Aron et al., 2005). Popular companies such as GE, American Express, VeriSign and Abbey National are already using this model successfully (Vashistha et al., 2005).

Another approach of this model is to create a dedicated centre that is some companies are too cautious about quality, which prevents them from hiring an offshore service provider, and they also have issues with cost, which prevents them from building their own offshore centre. Such companies take an alternative approach in which they build dedicated centre relationship. In this model an offshore supplier operates a dedicated centre for a company but the staff, equipment and resources are all wholly dedicated to the company.

Such centres do share some processes and long-term risks, which include co-ownership or co-leasing of resources. An example of a company using this model is Wipro. Wipro is a dedicated centre for Sun Microsystem as it runs the “Orbit”, which is a highly technical developer assistance centre for Sun Microsystem’ Solaris Operating System (Sadagopan, 2002). After working as a dedicated centre for 10 years they now have access to Sun’s Wide Area Network and troubleshoots developer needs from around the world.

In third-party transplant approach, a third-party rather than a company builds and maintains the offshore presences of a company (Vashistha et al., 2005). Many companies that already outsources their work on onshore third- party locations use this model as it's a natural progress to move their work to offshore locations in order to either cut down on cost or to gain profit by margining with a third-party, or both.

An example of a company using such a model is Accenture. It has more than 83,000 employees; over 5,000 are based in low-cost delivery centres in locations such as India and the Philippines (Robinson et al., 2004). This allows its client companies to that advantage from the low- priced labour without having to go through the risk of starting up in those markets. The objective of Accenture is to provide a faultless outsourcing experience for corporations (Hendrik et al., 2011).

This model allows companies like Accenture to distribute and manage their activities across multiple global locations at lower rates without risk. It also answers to faster time-to-market requests by dividing work among onshore and offshore locations (Robinson et al., 2004). Many large companies that hire global outsourcers prefer this distributed approach as this model saves clients from huge investments on team employment and it adapts to the changing requirements of the client.

2.2.2 Benefits of Offshore Software Development

Based upon the framework mentioned in Figure 2.1 the focus of this report is on the offshoring. In this section we will discuss the benefits due to which companies choose offshore software development (Hitt et al., 2002).

2.2.2.1 Economic Benefits

As mentioned in Chapter 1 that the main reason for companies to switch to offshoring is to cut down on cost (Pilatti et al., 2006; Radlo, 2016). Similarly a research done by Smite confirms that the main reason for companies to adopt offshore development is to reduce cost (Smite et al., 2011). Cut-down on cost is achieved by paying lower

salaries to their employees (Smite et al., 2011). As in India, a software developer's base annual salary is U.S $15,000 that is one quarter of the salary of an Irish developer who earns half in comparison to a US developer (Conchúir et al., 2009). Other economical factors that encourage offshoring are interest rates, development of capital markets, capital cost and development of technology centres.

Among the above-mentioned factors, development of technology centres plays an important role in finding the required resources. We know that talented developers are the key factor for great development productivity and quality however it is difficult to find them in general. With the help of offshoring this constraint has been solved as it provides companies' access to technical experts from all over the world (Smite et al., 2011). For example countries like India and China are highly populated as they have 1 billion people and they produce hundreds of thousands of software engineers per year (Conchúir et al., 2009).

Many software companies get into offshore software development in order to provide their clients with a wide horizon of experts to choose from. As they gather specialised people with specific skill set and ability to excel in one area, which enables them to deliver perfect-engineered applications.

Another advantage provided by development of technology centres is that it provides competitive advantage by getting innovative ideas from different countries. This has been also confirmed by a manager at Global Investments who said "Having people coming from different backgrounds will always help, getting different views from different people, since people coming from different parts of the world would have different ways of doing something." (Conchúir et al., 2009). Research done by Porter and Stern shows that Asian economies show a high rate of investment into national innovative capacity in comparison to Latin American economies (Porter et al., 2001).

2.2.2.2 Political- Legal Benefits

Political-legal benefits include taxation laws (Hitt et al., 2002), rights and working hours of labours, trade barriers such as tariff and non-tariff barriers (Stack et al., 2005). Treaties and agreement within trade unions such as MERCOSUE, NAFTA and ASEAN play an important role in encouraging companies to go for offshoring. An example of trade barriers is that for the past 10 years they keep on decreasing, which aid in facilitating offshore agreements (Jahns et al., 2006; Oshri et al., 2015).

An example of flexible labour laws can be see from the success of an overseas IT services provider Wipro. It is India's most successful outsourcing consultant. It gets 12% of its work from UK companies (The Economist, 04-03-2004) and in 2012 it has achieved the award of 'Offshoring Project of the Year' by UK's National Outsourcing Association for their project 'BT Recognized' (Wipro, 14-12-2012). In a study done by Economic Times showed that India is still the No. 1 destination for IT offshoring (The Economic Times 21-12-2010).

Considering that UK economy only represents one-sixth of the west-European total economy shows how much UK companies are inclined towards offshore business. One major reason behind this is the flexible labour laws in the UK, which makes it relatively easy for UK companies to relocate jobs offshore.

As the political and legal factors provide certain benefits they also have a downside such as due to them labour conditions are exploited and some debate that due to this the national labour market is effected as the nation losses jobs to offshore countries though there is still no consent on this matter (von Campenhausen, 2005). An example of this is that in a debate, titled "Offshoring in 2012 and beyond" held by British Computer Society concluded with that the UK is not going to necessarily lose IT-related jobs to overseas professionals as in order for offshoring to work successfully they need a highly skilled team on the onshore location, which creates new job opportunities for the people in the UK (British Computer Society, 2012).

2.2.2.3 Demographic and Geographic Benefits

Socio-demographic benefits include population size, age structure, education levels, work force motivation and time zone difference. As mentioned earlier that in countries like India and China are highly populated. They have 1 billion people in which 53% of the population is under the age of 25 (Robinson et al., 2004) hence producing hundreds of thousands of software engineers per year (Conchúir et al., 2009). Another significant factor is the availability of English-speaking population in counties like India, South- Africa and Philippines, which makes it easy for companies to outsource customer services such as call-centres to such locations (van Zoest, 2004).

Considering the time zone difference between the offshore countries companies cut down on cost by increase development time by adopting "follow the sun" workflow which means it allows 24 hours development as due to different time zones a company's employees can do development 24hrs a day (Carmel, et al., 2001). For example an employee works from 9 a.m. to 5p.m. in the USA. At 5 p.m. she hands over the incomplete task to a colleague in Australia who works from 9 a.m. to 5 p.m. based on his time zone. At 5 p.m. according to his country, he transfers the updated task to a colleague in Poland who works on the updated task for the next eight-hours and then forwards it to his colleague in the USA (Gupta, 2009).

While the employee in the USA had left work two of her colleagues worked on her task as when she will come to the office next morning a lot of the work would have been done. This work scenario takes advantage of the geographical distances as it allows people from different time zones to work round-the- clock in order to build software (Gupta, 2007).

The work distribution among the team can be done in two ways. First either the 3 teams distributed on different geographical locations work on the same task and each team keeps updated the task as mentioned in the above scenario or secondly a most efficient way is that we divided different aspect of the same problem among the team

for example in Figure 2.2 we can see how a problem has been distributed among different teams all over the world.

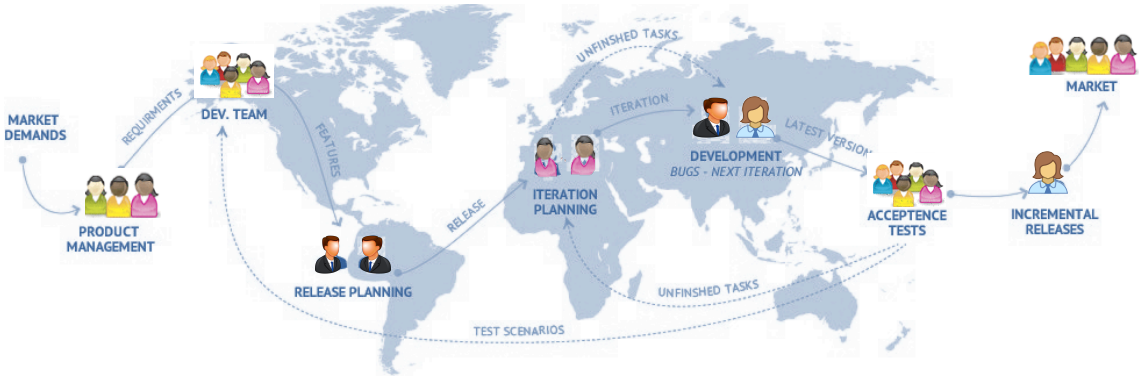


Figure 2.2. Distribution of Work among Multiple Distributed Teams.

2.2.3.4 Technological Benefits

Technological benefits are referred to the development of telecommunication and transportation technologies especially the Internet and mobile communication. Most of these developments happened due to the dramatic increase in the memory size of microprocessors, storage, increase in the integration of information technologies and telecommunications (Picot et al., 2003).

Many companies started moving towards offshoring as it became cheaper to communicate with offshore offices via conference calls and video conferencing (Prikladnicki et al., 2012). With the help of cheaper technology available the organisational and national boundaries remained no longer important as now companies can relocate their offices to offshore locations without facing a huge hurdle of communication and transportation costs (Jahns et al., 2006; Oshri et al., 2015).

Similarly use of standard tools for development of software across countries helps information sharing as all the information is represented in the same format (Beulen et al., 2005). From a service-provider point of view use of standard global tools supports

consistent and cost effective service provisioning, which means it allows consistent follow of information throughout the team and it encourages efficient delivery of services (Beulen et al., 2005).

2.3 A Study on Identifying the Challenges of Offshore Software Development

As highlighted in the previous section there are many benefits to adopting offshore development, however as it continues to grow (Damian et al., 2006) it has been observed that it causes some challenges, due to temporal, geographical and socio-cultural differences (Holmstrom et al., 2006). As software development is a human-centric and socio-technical process, it depends on complex interaction, attitudes, behavioural norms and communication approaches, which can lead to misunderstandings due to any misinterpretation of the project aims that can result into conflicts, mistrust and underutilization of talent (Ozawa et al., 2013). Studies done by Carmel (Carmel et al., 2005) and by Sahay (Sahay et al. (2003) show that these challenges can cause complications to the project processes such as communication, coordination and control of project activities (Damian, 2002, Jan et al., 2016).

It has also been observed that due to the significant differences in engineering culture/style around the world collaborations between countries have unique flavours (Herbsleb et al., 2005). In this section we will study how temporal, geographical and socio-cultural differences can pose different challenges for companies adopting offshore software development. The study focuses on the challenges that occur in the-offshoring business model, offshore outsourcing.

As the characteristics of onshore and offshore are different causing different challenges. Based on an extensive literature review on offshore software development, we identified four key challenges that occur as the team is distributed over different time zones and these challenges also affect the adoption of agile practices in offshore development. In this section we study the challenges that are inherent to offshoring and then we discuss how they affect agile practices. Table 2.1

shows the results of the study that we carried out to identify the inherent challenges, which are trust; socio-cultural; communication and co-ordination; and knowledge transfer issues. The reason for choosing to focus on these four challenges is because of their repeated occurrence in literature. Table 2.1 shows the frequency that these challenges have occurred in the literature. The first column shows the categories of the challenges constructed from the data extracted from the evidence that are presented in the second column. Each occurrence has the same weight, thus the frequency of the occurrence shows the number of times a category has occurred in literature. The selected studies are presented in Appendix A.

Table 2.1. Challenges in Offshore Software Development.

No.	Challenge	Evidence	Occurrence
1	**Trust**	E1, E2, E3, E4, E5, E6, E7, E11, E13, E19, E20, E24, E28, E29, E30, E33, E38, E41, E42, E45, E46, E47, E49, E54, E56, E57, E62, E67	29
2	**Socio-Cultural**	E1, E2, E3, E4, E6, E7, E11, E12, E16, E18, E19, E24, E26, E27, E28, E31, E32, E35, E41, E42, E46, E47, E44, E48, E52, E62, E66, E67	29
3	**Communication and Co-ordination**	E1, E4, E6, E8, E9, E10, E16, E16, E18, E19, E21, E22, E23, E24, E25, E28, E36, E37, E39, E40, E41, E43, E46, E47, E48, E50, E52, E53, E55, E58, E59, E60, E61, E62, E63, E64, E65	37
4	**Knowledge Transfer**	E1, E10, E14, E15, E22, E32, E34, E46, E47, E51, E53, E55, E60,	13

2.3.1 Trust Issues

The first major challenge to offshore development is the factor of trust. Trust is an important aspect of interpersonal (Boon et al., 1991) as well as inter-organisational relationship (Ring et al., 1994) in order to have successful partnership and alliances among the firms (Das et al., 1998). It is also considered as a crucial factor for building business relationships as trust enables open communication which, results in high performance of the team, high quality of projects and satisfactory decision making process (Kanawattanachai et al., 2002; Morgan et al., 1994; Rousseau et al., 1998).

In terms of software outsourcing, trust helps in establishing an open exchange of information and cooperative behaviour among the companies. It also helps in removing conflict and negotiation on software development cost and it also improves response to any crises, which might occur during a project (Rousseau et al., 1998). Moreover trust also enables ease in development process as it encourages an open discussion in the requirement gathering process and helps remove unnecessary expensive documentation (Humphrey, 1989).

Trust plays an important role in offshore software development however it is difficult to establish because it is difficult to develop a relationship with unknown foreign partners that are timely and geographically distant (Prikladnicki et al., 2012). The main reason behind this is that the companies do not have any relationship beyond the project itself, which is of a limited duration. Moreover trust can be difficult to establish as most projects are developed through structural mechanisms, which include deliverables, penalty clauses and reporting arrangements whereas in-house development relays more on trust rather than on details structured reporting (Lander et al., 2004).

While realising the difficulty in establishing trust in offshore project it is still needed in order to cooperated with foreign companies as distrust can hurt the performances as it leads to finger pointing and each organisation starts focusing on how other organisations may have hurt the project whereas trust improves the performance

(Sabherwal, 1999). Issues in trust have led some organisations to create their own software development centres in countries like India, China, Russia and Brazil (Prikladnicki et al., 2012). In Section 2.4 we have demonstrated how trust effects each stage of software development lifecycle.

2.3.2 Socio-Cultural Issues

In offshore development distributed teams face many socio-cultural differences, such as difference in languages, national traditions, values and norms. Kluckhohn identify five areas which all-cultural groups have fundamental through differing beliefs, which are listed as follows (Kluckhohn et al., 1961):

- How a culture views human nature,
- The relationship of its people with nature
- Priority it gives to traditional customs, future plans or present events.
- How is the society organised, is it linear in hierarchy?
- How is business and life conducted? Publically or privately, or a mix of both?

Similarly another widely refereed work on culture is by Hofstede (1980,1997), who developed a set of cultural indices. He identified five cultural dimensions, which are:

- Power Distance
- Individualism/Collectivism
- Masculinity/ Femininity
- Uncertainty Avoidance
- Long-Term/ Short-Term Orientation.

The socio-cultural distance is the measure of one person's understanding of another person's values and norms (Pilatti et al., 2006). It is commonly observed that people from one society strangely perceive the actions of people from another society as noted by Kotlarsky, culture can have a huge effect on how people understand and react to certain situations (Kotlarsky et al., 2005). It has also been observed that team

members with different values cause conflicts within the team resulting in decrease in motivation, involvement and cohesiveness with the team members (Ozawa et al., 2013).

Trompenaars and Hampden-Turner build their work on Hofstede's cultural dimensions with the focus on the impact of intercultural variances on business and management process (Trompenaars et al., 2004). They developed a set of seven values for dimension, which are shown in Table 2.2 below:

Table 2.2 Seven Values for Cultural Dimensions (Trompenaars et al., 2004).

No.	Cultural Value Dimension	
1.	**Universalisms vs. Particularism**	
	Rules to be followed under all circumstances	Special consideration based on the uniqueness of the situation.
2.	**Individualism vs. Communication**	
	People believe in personal freedom and achievement	People believe that the group is more important than individual achievement.
3.	**Specific vs. Diffuse**	
	Keep work and personal lives separate. Focused on work-oriented relationships.	Overlap between work and personal life. Believe that in order to have good relationships with co-works is vital for the success of their business.
4.	**Neutral vs. Emotional**	

	Reluctant to reveal their feelings and make an effort to control their emotions.	Display their thoughts and feelings openly. People find ways to express their emotions, even spontaneously at work.
5.	**Achievement vs. Ascription**	
	The organisational culture values you based on your performance at work. They believe that you are what you do.	Status in the organisation is based on a variety of factors such as power, title and position.
6.	**Sequential Time vs. Synchronous Time**	
	Like events to happen in a sequential order. They place a high value on punctuality, planning and sticking to schedules.	Focus on past, present and future events as interwoven periods. Work on several projects at once and consider planning and commits flexibly.
7.	**Internal Direction vs. Outer Direction**	
	Control nature of their environment to achieve their goal. This includes how they work with teams within their organisations.	Consider that the nature of the environment controls them and that they must work with their environment to achieve their goals. At work they focus their actions on others in order to avoid conflicts where possible and need constant reassurance that they are going a good job.

Tylor, a well known Sociologist and Anthropologist defined culture, as “Culture is that complex whole which includes knowledge, belief, art, morals, law, customs and any other capabilities and habits acquired by man as a member of society” (Tylor 1871). From this definition of culture it is clear that culture plays a great role in all aspect of life. Similarly in offshore software development the socio-cultural distance between distributes teams is a complex dimension as it involves organisational culture, national culture and language, politics and motivation of individuals and work ethics (Holmstrom et al., 2006).

Ozawa et al. (2013) identified three main challenges causes due to socio-cultural differences, which are:

i) Difference in the openness in the society.
ii) Difference in the willingness to adopt new techniques and technologies.
iii) Difference in communication methods, implicit over explicit communication

The most obvious disadvantage is language, as English is not the first language in counties like India, Pakistan and China so extra effort is required to communicate. Differences in language can lead to miscommunication due to language style or incorrect use of vocabulary (Carmel, 1999; Hofner et al., 2007). Some reports show that language problem affects the product and the code itself as the comments in the code from an offshore team whose first language is not English may seem odd or not understandable by onshore team. Hence adding a hidden cost of code maintenance (Matloff, 2005).

In a research done by Lee, shows that cultural differences affect the product and the process of development (Lee et al., 2001). A lot of work has been done on how much cultural differences affect the end product in order for people of a region to engage with the product (Yeo, 2001). For example an employee in a US company can directly pass work to his peer in India easily bypassing the management hierarchy whereas an employee in India cannot do so without upsetting higher management. This shows

how cultures of two different countries affect the workplace environment as in India, they follow strict management hierarchies and in US they don't (Matloff, 2005).

A similar experience was observed when a Japanese company offshored their software project to a company in China (Ozawa et al., 2013). They were facing difficulties due to the low quality deliverables and a high turnover rate of Chinese members because of socio-cultural differences. Based on Ozawa et al. (2013) we present a summary of the socio-cultural difference between Japan and China in Table 2.3 (Ozawa et al., 2013).

Table 2.3. Cultural Comparison between Japan and China (Ozawa et al., 2013).

No.	**Japan**	**China**
1.	A job is for a lifetime.	Acquire new skills and switch jobs.
2.	Hire fresh graduates and teach them to fit in their organisation rather than to hire experienced people.	Prefer hiring experienced people.
3.	Focus on avoiding failure and use existing methodologies rather then focusing on attaining success and trying new technologies.	Want to acquire new skills and try new methodologies and technologies. More focused on success.
4.	Tight grouped communities, so they understand each other without "implicitly" saying them.	More diversity, so use more explicit information in communication.
5.	Questioning considered humiliation.	Questioning considered humiliation.

Due to the differences in the teams' socio-cultural values, the Japanese company found it hard to retain its Chinese employees, as their motivation to work was low. The project specifications were not clear, as the Japanese company relied more on implicit

information, which were not clear to the Chinese team members and since questioning is considered humiliation in both the cultures, the Chinese team members didn't ask the Japanese clients to clarify the requirements hence resulting in low quality deliverables.

MacGregor identified 5 cultural patterns, which highlight the problems caused by socio-cultural challenge when the team is distributed over different locations (MacGregor et al., 2005). Table 2.4, gives an overview of their patterns. Though they are calling them patterns they have classified some of them as anti-patterns.

Table 2.4. Cultural Patterns in Software Process Mishaps (MacGregor et al. 2005).

No.	Pattern Name	Assumption	Pattern Overview
1.	**Yes (but No)**	Person A has assumed Person B meant "Yes I will do it, when Person B meant I heard what you said."	- Person B perceived Person A to be higher in hierarchy. - B doesn't want to affect the deadline. - Make clarification and rephrase request.
2.	**Proxy Pattern**	Companies are not willing to invest in intercultural training	- Place bi-coded individuals in a position where they can translate between the cultures.
3.	**We'll- take-you-literally (Anti-Pattern)**	Different cultures may have different perceptions for a development practice, process or	- Resulting in inefficient, frustration and distress among different cultural teams.

		artefact.	
4.	**We're-one-single-team**	Agile encourages "flat team" however in some cultures they follow a more hierarchical approach depending on power distance	- Formal communication with people on higher hierarchy level. - Developers don't feel they have a right to take decisions regarding the system they are developing.
5.	**The-customer-is-king (Anti-Pattern)**	Less hierarchy dominant structure encourage open and direct communication between the client and developers. While high hierarchy organisation prefer indirect communication between the client and developers.	- When problem arises, developers from high hierarchy organisation, the developers cannot contact the client directly hence causing delay in development time. - Similarly the client might feel that the communication is limited. - As nature of social culture also affects the relationship between the developers and client such as where it is work-oriented or relationship oriented.

From the above examples, we can see that socio-cultural conflicts can cause problems in the development of an offshore project. In order to solve such conflicts, the organisations need to create an environment that encourages teams to communicate with each other. They need to continuously adapt team roles based on the changing relationship between the team members. This can reduce dissatisfaction among the employees and increase trust resulting in giving each other more responsibilities, thus improving the quality of software.

2.3.3 Communication and Coordination Issues

In offshore development as the team is separated and implemented at different geographical locations they face many communication and coordination issues because the team members work on different time zones or time shifts in a day (Alzoubi et al., 2016). This reduces the opportunity for real-time communication (Al-Zaidi et al., 2017). The basic issue in offshoring is handling the complex communication and team coordination, as they can't have frequent face-to-face communication (Sahay et al., 2003). Insufficient team communication often creates challenges such as trust, relationships and efficiency of the team (Lanubile et al., 2013).

A research done by Ebert shows that approximately half of all the distributed projects fail due to insufficient communication and trust among the team members (Ebert, 2012). In order to increase communication among the team members, awareness is necessary. Awareness in distributed teams helps in ensuring that individual contributions, contribute to the whole group's effort to develop software successfully. Paul Dourish and Victoria Bellotti described group awareness as " an understanding of the activities of others, which provides a context for your activity" (Dourish et al., 1992).

Group awareness can help overcome some challenges of offshore projects (Lanubile et al., 2013). There are four types of group awareness, which are (Gutwin et al., 1996):

- **Informal awareness** provides information about which team members are around and available for work.

- **Group-structural awareness** focuses on the knowledge roles on team members and structure of the team.

- **Workspace awareness** gives information about the interactions the team members have with shared resources at a workspace.

- **Social awareness** consists of information that team members maintain about each other in conversational context for the purpose of social connections within the team.

In order to achieve group awareness in distributed teams, technology plays an important role. In the survey conducted by Lanubile focused on which technologies and tools support group awareness and collaboration such as for informal awareness, IM (Instant Message) and VoIP (Voice over Internet Protocol) tools can be used similarly for workspace awareness emails and RSS can be used (Lanubile et al., 2013).

Another way to increase the communication between the teams is to keep the working hours flexible so that the onshore and offshore teams can achieve overlapping hours with each other. For example in order to get real-time communication, occasionally one team has to stay late and the other team has to come early to have a combined meeting via video/audio conferencing tools (Yu et al., 2016). Hence in order to organise work between geographically distributed teams we must consider temporal distance to facilitate the real-time communication (Holmstrom et al., 2006). Temporal distance is a measure of the dislocation in time experienced by two actors who wish to interact (Pilatti et al., 2006).

Asynchronous tools are seen as a crucial part for communication and coordination among remote locations. Due to temporal distance the increase in the response time

creates a feeling of “being behind” as when one location sends a request they get reply the next day. As seen in the case study done by Boland and Fitzgerald asynchronous communication overnight can be overwhelming for a developer beginning work in the morning (Boland et al., 2004). Also a study done by Holmstrom shows that limited overlap with colleges causes delay in response which makes people lose track of the overall work process which leads to problems in distributed yet time-crucial work (Holmstrom et al., 2006). Similar results were noted by Herbsleb, the drag in the problem and response could cause increase in the cost (Herbsleb, 2007). As a result response time increases when working hours do not overlap between the remote locations (Sarker et al., 2004).

2.3.4 Knowledge Transfer Issues

Knowledge is a very common and widely used concept. In order to clarify what is referred as knowledge we have used the definition provided by Davenport and Prusak (Davenport et al., 1998). They defined knowledge as “A fluid mix of framed experience, values, contextual information, and expert insight that provides a framework for evaluating and incorporating new experiences and information. It originates and is applied in the minds of knowers. In organisations, it often becomes embedded not only in documents or repositories but also in organisational routines, processes, practices, and norms.” (Davenport et al., 1998)

Using the above-mentioned definition of knowledge, as companies started moving their business and development process to offshore locations, knowledge transfer became critical (Zahedi et al., 2016). A study done by Accenture shows that people, patent and knowledge comprise for 70% of Exchange-listed companies value whereas in 1980 it was just 20% (Aronsson, 2007). This shows how important knowledge is for companies and any mistake during a knowledge transfer can cause a negative affect on the companies. In offshoring the process of knowledge transfer is not straightforward as when transferring knowledge from one country to another the factor of cultural differences causes challenges (Kedia et al., 2007). Culture does not automatically affect

the knowledge transfer process but if there is poor management it can cause projects to fail (Javidan et al., 2005). Similarly in a study done by Radoff's shows that if we do not understand the cultural differences between the countries it can cause a negative impact on outsourcing (Radoff, 2006). As we need to understand the cultural differences in order to successfully manage the projects. The report also shows that the companies that educate their employees of intercultural communication increase their productivity by 30%.

One major challenge organisations face, which directly affects the knowledge transfer process, is when they decide to offshore their process before they have tested the readiness of their management. This includes the difficulties of keeping awareness, and knowledge cohesion when various working groups concurrently access it (Khan et al., 2014). These difficulties directly affect how work is distributed through task allocation in distributed environment. According to Sajjad the factors presented in table 2.5 should be considered while allocating tasks when the team is distributed to overcome the knowledge transfer challenges (Sajjad et al., 2015).

Table 2.5. Factors Affecting Task Allocation Process in Offshore Development (Sajjad et al. 2015).

No.	Factors	Percentage	Affect of Factor on Task Allocation Process in Offshore Development
1.	**Site Technical Expertise**	68%	Select site with appropriate domain expertise and knowledge is crucial.
2.	**Time Zone Different**	63%	Allocation of tasks should be based on the time zone. There are two common approaches used by practitioners while considering the time zone, they either allocate tasks based on Time-Zone Band or Follow-the-Sun. The selection is made considering the fact that it should allow the different sites to be able to have synchronous communication to

			allow knowledge sharing.
3.	**Resource Cost**	47%	Project Managers do aim to assign work units to low cost labour sites in order to cut down on development cost.
4.	**Task Dependency**	44%	Keeping the cost benefit in mind, it is however advised to not only consider the cost while assigning the tasks but to also consider task dependencies. As high task dependent tasks should be assigned to the same location in order to save time on communication and coordination.
5.	**Vendor Reliability**	36%	The perceived reliability of a particular vendor helps clients to better manage tasks allocation in global software development.
6.	**Task Size**	29%	Task size has also been considered important as it helps in deciding how much time the team will spend to develop it.
7.	**Vendor Maturity Level**	21%	It is important to consider the maturity level of the vendor as, if they are new to offshore development, they may face issues in code integration.
8.	**Local Government Regulations**	13%	This factor isn't directly related to the software development process, however it has been perceived as an important factor to be considered while task allocation.

9.	**Requirements Stability**	7%	Based on the type of project, requirements stability is important in order to break them into tasks for the team to develop. However it has only been considered to be an important factor by 7% based on the selected literature of this research, as it is a universal fact that requirements do change and organisations have placed requirement change management models in place to handle this issue.
10.	**Product Architecture**	7%	Since the requirements keep changing, the product architecture keeps evolving throughout the software development lifecycle. This has been considered as an important factor by 7% based on the articles selected from literature.
11.	**Intellectual Property Ownership**	2%	In distributed software development, as code is being developed at different locations, it is important to consider the issue of intellectual property however this factor isn't considered critical for task allocation as the client is usually the one who has the intellectual property of the code being developed.

The common problems between with the company that is transferring knowledge and the company that is receiving the knowledge are: i) communication problem, ii) different ways to conduct work, iii) different work attitudes, and iv) different decision-making process (Radoff, 2006).

2.4 Critical Analysis of Offshore Challenges on Software Development Phase

In this section we will discuss how the above, identified challenges affect the different phases of software development and present a critical analysis.

The traditional software development lifecycle consists of five stages, which are, Requirements, Design, Coding, Testing and Integration. Figure 2.3 below shows the software development lifecycle.

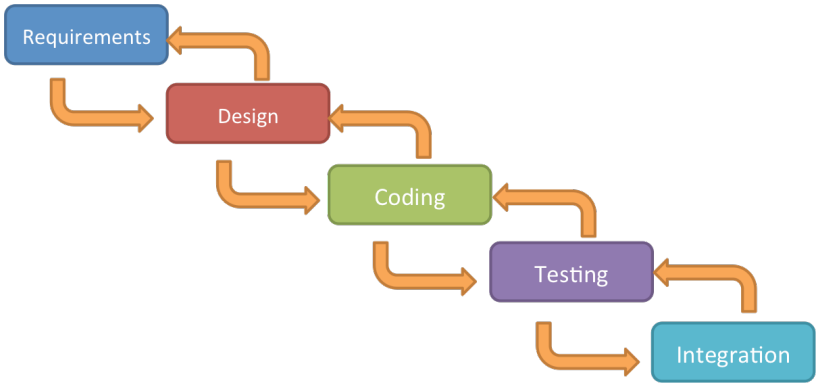


Figure 2.3. Traditional Software Development Lifecycle.

The execution of these phases depends on whichever software development approach the development team decides to selected. An overview of the existing approaches is presented in the Appendix B showing how these phases are executed based on the selected approach.

Based on the literature review, we present in Table 2.6 how each phase of the traditional software development lifecycle is affected when organisations choose to move their projects offshore. We have mapped the four key challenges of offshore development on each lifecycle phase to show how these offshore challenges effect the execution of each phase.

Table 2.6. Offshore Challenges Affecting Software Development Phases.

No.	Phase	Offshore Challenge	Effect of Offshore Challenge on Software Development Phase
1.	**Requirements**	Trust	• Establishing correct estimates of requirements is difficult, as onshore team does not have clear understanding of offshore team skills (Alnuem et al., 2012). • Getting the main objective and core functional requirements of the project clear to all the team members located at different sites is difficult (Herbsleb et al., 2001).
		Socio-Cultural	• Differences in cultural values and language can cause misunderstanding of requirements (Damian et al., 2003). • As at the beginning of the project the client isn't sure of all the requirements, which can result in vague requirements, which need to be later clarified to the teams who aren't on-site (Bird et al., 2009). • Difference in ways to express dissatisfaction, perception regarding punctuality, scheduling, hierarch,

		urgency or risk often leads to misunderstanding among the client and team members hence spoiling the relationship (Damian et al., 2003; Niaz et al., 2012; Bhat et al., 2006).
	Communication and Coordination	• Changes in the requirements, needs to be communicated to all the distributed teams. Any mistake in recording the change can cause problems in the development of the system (Sengupta et al., 2006).
	Knowledge Transfer	• As the project is being developed at different locations, there can be inconsistencies in the work done and documented user stories (Bhat et al., 2006). • Maintaining bidirectional traceability of requirements across different sites is difficult as each site is working on different requirements (Berenbach et al., 2006). • Managing requirements, on-site customer and daily meetings become difficult with distributed teams (Bhat et al., 2006).

2.	**Design**	Trust	• Lack of standard method that can proactivity verify organisation structure can create mistrust between the client and the team (Herbsleb, 2007).
		Socio-Cultural	• It is difficult to communicate architecture design decisions to geographically and culturally dispersed teams (Clerc, 2008). • Synchronous meetings of architecture and developers are difficult, which impinge building of common understanding of architecture and developers (Bosch et al., 2010)
		Communication and Coordination	• Coordination problems occur due to lack of common understanding of architecture design (Jaakkola et al., 2010). • It is difficult to discover the correlation between architecture decision and the coordination requirements they will enforce (Herbsleb, 2007).
		Knowledge Transfer	• Non-textual artefacts make it difficult to highlight and manage change (Sengupta, et al., 2006).

			• Lack of documents that describe the overall architecture of the software under development (Ovaska et al., 2003). • Out-dated documentation leads to rework (Cataldo, et al., 2007).
3.	**Coding**	Trust	• Insufficient social integration can degrade the performance, lower motivation and satisfaction of the developer. Resulting in creating mistrust between the client and the development team (Koehne et al., 2012).
		Socio-Cultural	• Due to cultural differences, developers often misinterpretation each other, causing wrong requirements to be coded (Herbsleb et al., 2005). • Misinterpretations of technical vocabulary due to dissimilarity in technical cultures causes misunderstanding, rework and consequently delays in the projects (Herbsleb et al., 2005). • Developers at remote sites, insufficient use version control systems due to different organisational cultural habits

		(Cataldo et al., 2007).
		• Developers belonging to higher economy countries usually do not help people belonging to low economy countries in fear of losing their jobs to them (Bird et al., 2009).
	Communication and Coordination	• Usage of different process and disparity in process maturity at different sites can cause coordination problems (Bird et al., 2009; Herbsleb et al., 2005).
		• Staff members at new offshore sites have a tendency to not reply quickly to emails of onshore members (Herbsleb et al., 2005).
		• Communicating all the details and norms of the process to be followed to remote teams is very time consuming (Mullick, et al., 2006).
	Knowledge Transfer	• Difficulties in tracking information in distributed development (Herbsleb et al., 2005; Koehne et al., 2012).
		• Divergence in tool usage among different teams causes knowledge transfer problems (Bird et al., 2009).

4.	**Testing**	Trust	• Customers feel insecure to trust their real-life data with offshore organisations, as they haven't worked with them before, hence they generate mock databases for the purpose of testing (Sengupta et al., 2006). • In acceptance testing, as feedback is being provided indirectly, vital information can be lost, causing the client to not trust the development team (Liskin, et al., 2012).
		Socio-Cultural	• Offshore testers don't get sufficient cooperation from the onshore members due to misunderstanding (Tervonen et al., 2013). • Communication is difficult between the onshore testers and offshore testers due to the absence of a common native language and different vocabulary (Camacho et al., 2013).
		Communication and Coordination	• Face-to-face meetings are difficult and expensive causing communication and coordination gaps among the team members (Tervonen et al., 2009). • Reduced awareness of work due to lack of informal contact and

			geographical dispersion (Tervonen et al., 2013).
			• Limited budget restricts communication, coordination and collaboration mechanism (Tervonen et al., 2013).
		Knowledge Transfer	• It is hard to spot a developer of the code where a bug is found (Grechanik et al., 2010).
			• Testers in the development countries are not familiar with the clients testing tools and code, which is to be tested (Liskin, et al., 2012).
			• Due to frequent misinterpretations of requirements and interface specification confusions and misunderstanding occur, which cause inconsistence, which are not exposed until integration testing (Sengupta et al., 2006).
5.	**Integration**	Trust	• Inadequate and incomplete interface specifications are identified only at the integration phase, which cause mistrust among the client and development team (Sengupta et al., 2006).

	Socio-Cultural	• It is difficult to resolve difficulties with unfamiliar remote participants while integrating the code (Conchúir et al., 2006). • Due to lack of cohesiveness and cooperation between the different locations, the offshore team feels they are integrating components of competitors (Herbsleb, et al., 2005).
	Communication and Coordination	• Inadequate communication and lack of domain knowledge lead to lack of understanding in developers about requirements, which often lead to misinterpreted and conflicting modules (Damian et al., 2003; Bhat, et al., 2006; Sengupta et al., 2006). • Communication between remote site only through a single mediator can because the cause of integration failure (Čavrak et al., 2012).
	Knowledge Transfer	• Implementing different code branches at each development site can lead to complex integration problems (Herbsleb, et al., 2005). • Inadequate knowledge and expertise in GSD of integration team can increase

		risks of integration failure (Kommeren et al., 2007). • Risk of integration failure increases with the increase in interdependencies between modules to be developed (Herbsleb, 2007).

Based on the above table we can see how each phase of the development life cycle is effected by offshore challenges. From these findings we can observe how the offshore challenges are interlinked and understand how these challenges are interrelated and affect the development lifecycle such as we can now say that trust, socio-cultural, communication and coordination and knowledge transfer issues effect all the phases of the development lifecycle.

To overcome the challenges mentioned in Table 2.6, work has been done using different approaches such as patterns, ontology-based and agile methodology. Table 2.7 gives an overview of such approaches.

Table 2.7. Overview of Existing Approaches used to Overcome Offshore Challenges.

No.	Approach	Overview of the Approach
1.	**Use of Patterns for Global Software Development**	MacGregor (MacGregor et al., 2005), Shah (Shah et al., 2012), Paasivaara (Paasivaara et al., 2003), Bricout (Bricout, 2004), Lescher (Lescher, 2010), van Heesch (van Heesch, 2015), Valimaki (Valimaki et al., 2009), Pehmöller (Pehmöller et al., 2010) and Salger (Salger et al., 2010) provided patterns to be used in Global Software Development. Details of their patterns can be found in Section 2.7.
2.	**Using Agile Practices to**	Beecham proposed to use agile practices to solve

	Solve Global Software Development Problems	global software development problems (Beecham et al., 2014). The aim of their research was to answer the research question: • What GSD problems can agile methods solve? They interviewed 24 practitioners from FS group and from that they presented 16 issues that were raised by the practitioners while they developed software offshore. They answered 9 issues using agile practices and partially answered 3 and left 4 unanswered. Detail of their solution can be seen in Appendix C.
3.	**Ontology-Based Multi-Agent System to Support Requirements Traceability in Multi-Site Software Development Environment**	Pakdeetrakulwong proposed an ontology-based solution using multi-agents to collect requirements when the project is being developed is offshore (Pakdeetrakulwong et al., 2015). Their proposed architecture had four agents: i) User Agent ii) Recommender Agent iii) Ontology Agent iv) Evolution Agent The main capabilities of these agents are to: • Handle change in the requirements (add/update/deleted). • Identify requirements traceability information and generate traceability matrix. • Analyse change impact on requirements and

		software artefacts. • Send message to notify relevant user about the impact of requirement change; and • Recommend incomplete requirement information.
4.	**Experiments for offshore project to address centrifugal forces.**	Based on Carmel's identified five centrifugal forces that pull people, team and product group apart which are as follows (Carmel, 2010): • Geographical dispersion • Coordination breakdown • Loss of communication richness • Loss of team-ness • Cultural Differences Crag Larman and Bas Vodde designed experiments to be conducted by offshore teams in order to address these issues (Larman et. al., 2010). An overview of their experiments is listed below: • Try.. Fewer sites • Try .. Think "multisite" even when close. • Avoid.. Believing in multi-site daily scrum magic or that multisite forces are inconsequential. • Avoid.. Thinking distributed must mean dispersed. • Try.. One iteration (Sprint) for the product, not for site. Similarly they designed experiments related to team

		and site structures, interaction and coordination, multisite culture and norms, tool, and tests.
5.	**Understanding Collaborative Practices in Distributed Agile Development using Theoretical Concepts**	Modi presented a research proposal in which they used an interpretative qualitative approach along side case studies, to gain deeper understanding into how teams collaborated in distributed agile development scenario (Modi et al., 2013). There analysis was based on theoretical concepts such as common ground, boundary objects and awareness,. The aim of their research was to answer the following research questions: • How do global teams collaborate to establish common ground/shared understanding? • How does knowledge sharing take place? • What transformation takes place? • What can we learn from existing distributed agile teams? Based on their selected theoretical concepts they explained on how they can help distributed teams to solve the collaboration challenges. Below we have defined each concept and given an overview of how they can help improve collaboration in distributed scenario: • *Common Ground:* According to Clark theory of common ground regarding mutual knowledge, beliefs and assumptions maintain that the participants must have a shared awareness to

		carry out a joint activity (Clark et al., 1991). According to Modi's research failure to establish and maintain a common ground can result in serious breakdowns in collaborative work (Modi et al., 2013). • *Boundary Objects:* Star defined boundary objects as objects, which are both plastic enough to adapt to local needs and constraints of the several parties employing them, yet robust enough to maintain a common identify across site (Star et al., 1989). Based on Modi's research, in GSD the project artefacts such as user stories, the shared code and test cases are the integral part of agile and can be viewed as boundary objects in-use, as they provide a common focus to the project goals and act as mediators for communication, coordination and cooperation among the team members distributed over different sites (Modi et al., 2013). • *Awareness* is defined as an understanding of the activities of others, which provide a context for the project activates. According to Modi's research the 3C collaboration model, defines collaboration as the union of communication, coordination and cooperation efforts (Modi et al., 2013).

In Table 2.7 we presented different approaches that are being used to overcome offshore challenges however they have limitations for example using agile practices to solve global software development problems has limitation that even though this

approach answers GSD challenges it does not discuss in detail how practitioners can solve them as they only presented single line solutions similarly using ontologies for requirements approach only focuses on overcoming the challenges of the requirement phase by using ontologies and it provides a very technical solution which is difficult to be used by product owners that aren't from IT background. Further we have discussed these approaches in table 5.8.

2.5 Agile Offshore Software Development

In this section we will first explain what agile is and then how we can use agile methodologies for offshore software development. We have also mentioned what are the limitations of agile when used for offshoring.

2.5.1 Agile Software Development

In 2001 the formation of agile manifesto has brought unprecedented changes to how software is developed (Beck et al. 2001). The manifesto focuses on four points which are: i) individuals and interaction over process and tools ii) working software over comprehensive documentation iii) customer collaboration over contract negotiation and iv) responding to change over following a plan. Broadly speaking agile encourages collaborative development in which people are involved throughout the development process allowing them to openly exchange which helps bridge the gap between clients and developers resulting in a successful product.

Agile software development focuses on 12 principles which are as following (Beck et al., 2001):

- "Customer satisfaction by rapid delivery of useful software
- Welcome changing requirements, even late in development
- Working software is delivered frequently (weeks rather than months)
- Working software is the principal measure of progress
- Sustainable development, able to maintain a constant pace

- Close, daily cooperation between business people and developers
- Face-to-face conversation is the best form of communication (co-location)
- Projects are built around motivated individuals, who should be trusted
- Continuous attention to technical excellence and good design
- Simplicity—the art of maximizing the amount of work not done—is essential
- Self-organizing teams
- Regular adaptation to changing circumstances".

Agile software development is an iterative and incremental development model were the requirement of the project and its solution keeps on changing based on the collaboration and coordination of self-organised and cross-functional teams (Larman, 2004). Agile development life cycle has many methodologies such as Agile Modelling, Agile Unified Processes, Crystal Methods, Dynamic System Development Method (DSDM), Extreme Programming (XP), Scrum and Lean etc. (Dybå et al., 2008). In general agile methodologies emphasizes on frequent delivery of product increments, time boxes, communication and collaboration, working software as a measure of progress, adaptive planning and meetings.

A lot of work is being done on agile and it is being widely adapted by many companies to develop software (Abrahamsson et al., 2003). A literature search in the ISI Web of Science showed that 1551 research papers have been published on agile software development from 2001 till 2010 (Dingsøyr et al., 2012).

2.5.2 Agile Methodology in Offshoring

Companies that opt for offshoring perceive it as they are taking risk in some aspects of the software development process in order to reduce cost of software production (Nisar et al., 2004). As there is tension between agile benefits and difficulties of implementing agile. Agile methods help build trust and confidence between clients as it considers customers as part of the team. It also helps in reducing the risk of production of low quality software as by using agile methods like unit testing, pair

programming, continuous integration etc. ensures good quality software (Danait, 2005). However the process of adopting agility with distributed projects is not straightforward (Šmite et al., 2010). Global software engineering should focus on how to cleverly use information and communication technology to compensate for the inherent problems of distributed works and bridge the remote sites together (Hanssen, 2011).

In industry there are two different opinions when it comes to agile:

- Some companies believe that agile methods are a best fit for offshore development as offshore projects involve high risks of security, decreased development visibility, management and integration. Thus with the help of agile methods such projects can be better planned and executed (Simons, 2002, Hayes, 2003, Massol, 2004).

- Some believe that Agile methods are a mismatch with offshoring (Kontio et al., 2004). Taylor claim that projects that use agile for offshore development go through a lot of difficulties because of the differences in development practices as well as due to complex development environment (Taylor et al., 2006). A characteristic comparison done by Šmite shows how agile development is different from offshore development shown in Table 2.8 below (Šmite et al., 2010):

Table 2.8. Comparison of Agile Development verses Offshore Development (Šmite et al. 2010).

No.	Characteristics	Agile Development	Offshore Development
1.	**Communication**	Informal Face-to-face Synchronous Many-to Many	Formal Computer- mediated Often Synchronous Tunnelled

2.	**Coordination**	Change- driven Mutual adjustment, self management	Plan- driven Standardisation
3.	**Control**	Lightweight Cross-functional team	Command-and-control Clear separation of roles

2.6 Agile Adoption in Offshore Software Development

In this section we will highlight approaches used by organisation to transition from traditional software development to agile methodology, we also present different approaches used to adopt agile in offshore environment.

2.6.1 Transition from Traditional Software Development to Adopting Agile Practices

In this section we have focused on approaches present in literature that focus on facilitating organisations to transit from any traditional software development model to adopting agile practices. We haven't considered the complexity added by offshore software development.

2.6.1.1 Facilitators to Assist Agile Adoption

Javdani conducted a study consisting of 33 agile experts across 13 different countries to identify facilitators that would help organization to transition and adopt of agile methods (Javdani et al., 2014). They identified 8 facilitators that would help agile practitioners to do their job, which are as follows:

- Training
- Good coaching and mentoring
- Management buy-in

- Team member buy-in
- Right people selection and empowering team
- Continuous meeting and negotiation
- Agile champions
- Incentive factors

Based on their results it is clear that for the success of agile adoption, the team members and management need to have common interest. However the limitation of this research is that it only focuses on co-located team members and does not discuss the complexity added due to offshore teams.

2.6.1.2 Framework to Support the Evaluation, Adoption and Improvement of Agile Methods

Qumer also suggested a framework to support the evaluation, adoption and improvement of agile methods (Qumer et al., 2008). In their framework they developed an Agile Toolkit, which facilitated the construction and evaluation of processes in complex software projects. Figure 2.4 shows the main components' of their agile software solution framework, which are Agile Toolkit, Method Core and Software Technology.

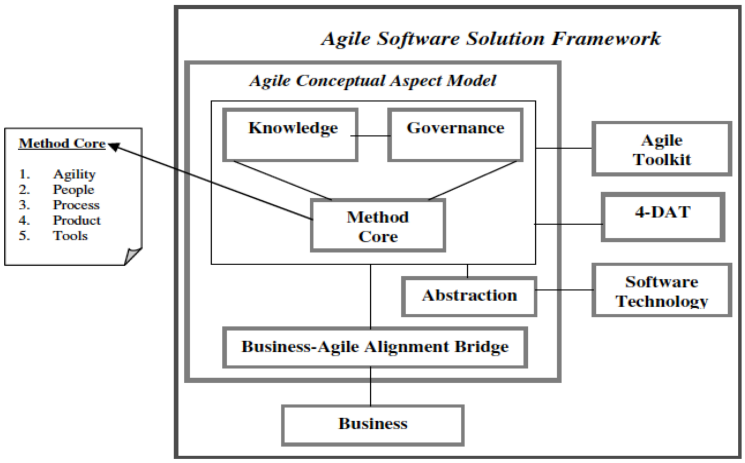


Figure 2.4. The Main Components of the Agile Software Solution Framework (Qumer et al. 2008).

They also developed an analytical tool that evaluated the degree of agility in a development practice. They also designed an Agile Adoption and Improvement Model (AAIM) based on industry analysis and a grounded theory research methodology. The AAIM has 3 agile blocks: i) Agile Block: Prompt, ii) Agile Block: Crux and iii) Agile Block: Apex which are further divided into six stages as shown in the Figure 2.5 below. This model focuses more on how to evaluate the degree of agility adopted in an organisation rather then providing a guideline on how agile should be adopted and it doesn't not explain how agile should be adopted in offshore software development.

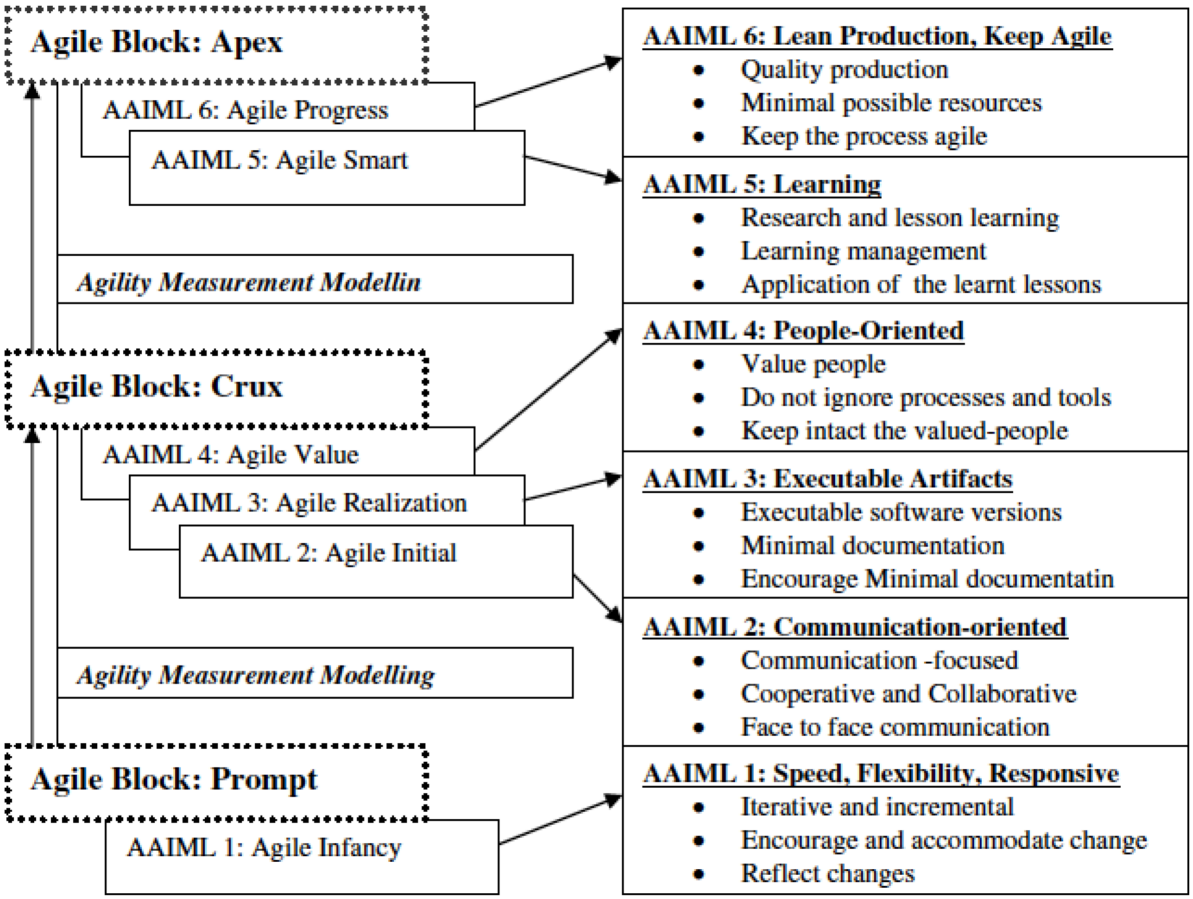


Figure 2.5. Agile Adoption and Improvement Model (Qumer et al., 2007)

2.6.1.3 Shared Mental Models to Understand Agile Practices

Yu used a theory from cognitive psychology known as shared mental models to help practitioners understand and apply agile practices, as according to empirical research

it was found that the perceived benefits of agile software development are not fully understood in research or by organisations (Yu et al., 2014). Hence causing to failures in achieving the required results. There research emphases on answering three research questions:

i) Why is improved interaction needed among development teams and customers?
ii) What aspects of interactions should be emphasised among the development team and customers?
iii) How does increased interaction improve collaboration during software development?

They focused on three Scrum and XP practices, which were System metaphor, Standup meetings and on-site customer and linked them to the types of shared mental model that would facilitate understanding and adoption of agile practices. For example, according to their research, standup meetings using shared mental model practice reflexivity can enhance developers understanding of the project. Similarly that with efficient reflective discusses during the standup meeting, the team understand about the project goals, challenges, limitations and system requirements.

2.6.2 Agile Adoption in Offshoring

In this section we have presented different approaches present in literature that are used to adopt agile practices in offshore software development.

2.6.2.1 Use of Patterns in Agile Adoption

As mentioned in Section 2.4 work has been done on using patterns in global software development. In this section we will give an overview of what is a pattern and how they can be used to facilitate agile adoption in offshore software development. We further conducted a study on identifying existing patterns used in GSD and then investigating specific patterns that are being used for agile adoption, which is presented in Section 2.7.

The term ‘pattern’ was introduced in software development as an inspiration of Christopher Alexander work on architectural patterns. He defined patterns as “Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice” (Alexander et al. 1977). He used this definition for buildings and town patterns however what he says are true for software patterns as well.

Generally a pattern has four essential elements (Gamma, et al., 1997):

- The **patter name:** to give a high level of abstract to the pattern. It gives us the idea of what problem the pattern is providing a solution for in a word or two. Giving a name to a pattern makes it easy for us to talk about it with people and for documentation.

- The **problem:** helps in describing when the pattern can be applied. It provides details of the problem and its context. It may include lists of conditions or scenarios, which must be met in order to apply the pattern.

- The **solution:** describes the elements that makeup the patterns, their relationships and responsibilities. The solution doesn’t describe a particular concrete practice or implementation, because a pattern is like a template that can be applied in many different scenarios. A pattern does provide an abstract description of a problem and how a general arrangement of elements/practices can solve it.

- The **consequence**: describes the outcome of applying the pattern. They are critical for evaluating a pattern and for understanding the benefit of applying a pattern to see if it helped in solving the problem and if yes, up to what extend. For software the consequence often refer to space and time trade-offs.

Currently there are six types of patterns:

- **Requirements Patterns:** Robertson defined requirement patterns as patterns that provide high-level abstraction of system requirements (Robertson, 1996). They help in the creation of better and precise requirement descriptions in lesser time.

- **Anaylsis Patterns**: Fowler defined analysis patterns as patterns that “reflects conceptual structures of business processes rather than actual software implementations" (Fowler, 1997).

- **Design Patterns:** Gamma defined as “descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context (Gamma, et al., 1997).

- **Architectural Patterns:** Buschmann defined architecture patterns as patterns that contain best practices for decomposing a software system into sub-systems (Buschmann et al., 1996). They also specify the responsibilities between the sub-systems that include rules and guidelines for organizing the relationships between them.

- **Anti-Patterns:** Brown defined anti-patterns as those patterns that describe negative examples of solutions (Brown et al., 1998). They describe repeated negative practices observed while solving a problem and provide better solutions.

- **Idioms:** They are patterns of the lowest level of abstraction. They show common programming techniques and conventions that recur while solving programming tasks. They are often language- specific.

Work has also been done on identifying offshore software development patterns and agile patterns in offshore development. MacGregor and Shah designed cultural patterns to manage the cultural difference challenges in offshore development (MacGregor et al., 2005; Shah et al., 2012). Paasivaara, Bricout, Lescher and van Heesch designed communication and coordination patterns to overcome communication and coordination challenges that onshore and offshore team members face while working together (Paasivaara et al., 2003; Bricout, 2004; Lescher, 2010; van Heesch, 2015). Valimaki designed patterns for project management (Valimaki et al., 2009) and Pehmöller designed testing patterns (Pehmöller et al., 2010). Salger worked on designing patterns for the requirements engineering process (Salger et al., 2010). Overview of these patterns is presented in Table 2.12.

Similarly work has been done on identifying agile offshore patterns. Hvatum designed magic backlog patterns to manage the backlog when the project is being developed at multiple sites (Hvatum et al., 2015). Fowler designed daily-stand up patterns (Fowler, 2016) and Välimäki designed nine distributed project management and scrum patterns (Välimäki et al., 2008). Cordeiro designed multi-site software development patterns (Cordeiro et al., 2007) and Elssamadisy focused on the dynamics of agile adoption (Elssamadisy et al., 2006). Overview of these patterns has been presented in Table. 2.13

2.6.2.2 Factors Contributing to the Success and Failure of Agile Adoption

Malik identified factors that contributed to the success and failure of agile offshore software development, which are as follows (Malik et al., 2010):

- Development Strategy
- Project Type
- Communication Channel
- Cultural Difference
- Split Location

- Size of Project

According to their research, coding and testing phases should be considered to be offshored and that critical modules such as project planning and design should be done on onshore location. The type of projects that have most success in offshore development are websites and web applications. Figure 2.6 shows the result of their research on finding what project type is suitable for offshore development.

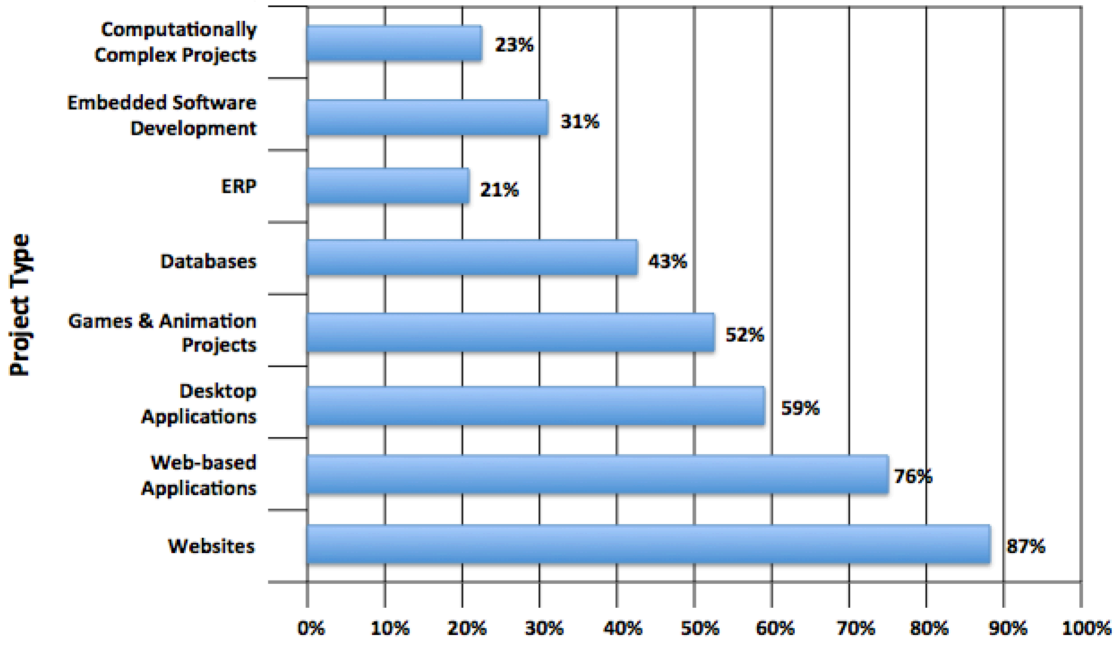


Figure 2.6. Software Project Success Rate for Offshore Projects (Malik et al., 2010).

2.6.2.3 Use of Tools in Agile Adoption in Offshore Development

As mentioned in Section 2.3.3, the main issue with offshoring is communication gap as agile is a big mismatch as its methods focus on regular face-to-face communication hence increasing the demand for communication (Beck et al., 2001). An alternative to this approach is that agile methods solve communication problem by "Offshore Development requires more communication, Agile methods provides more communication" (Nisar et al., 2004).

Now the question is how to apply agile methods in such a way that we can reduce the gap without providing face-to-face communication as the teams are geographically split. Many companies have solved this issue with the use of technology as they (Fowler, 2006; Simons, 2002; Danait, 2005):

- Hold online voice or video chat sessions as they provide real-time face-to-face meetings, which allow the teams to have an efficient conversation (Kotlarsky, Oshri, 2005).

- Collaborative tools such as Wiki, IM and discussion boards help improve the quality of communication of information.

Another issue is the transfers of processes as in offshoring companies move some of their business or development processes to offshore location, which causes problems when companies outsource critical modules of domain specific projects. General Motors is an example of such offshoring projects where they outsourced the development of their web-based "New Owner Centre" project. Agile provides a solution to this problem by sending ambassadors from one offshore site to another in order to help gather information of the business domain. But the disadvantage to this approach is that this practice is costly (Fowler, 2004).

Companies also face trust issues in offshoring as mentioned in Section 2.3.1, agile suggests to use web conferencing for virtual white-boarding to share and discuss project designs and information (Braithwaite et al., 2005). This helps to solve the issue of knowledge transfer mentioned in section 2.3.4 as the whole team is directly sharing information face-to-face via Internet. Agile methods also suggest using a shared code-repository as it allows the teams to see each other's work (Danait, 2005).

Some companies even opt to collocate the team during the initial period of the project so that the team members can get to know each other (Cottmeyer, 2008). This helps in building trust among the team and develops sense of that they all are one team despite located at distributed locations (Danait, 2005). However to collocate the team

is a costly process (Nisar et al., 2004). In a literature study done by Paasivaara et al. (2009) they classified how different agile practices contribute in offshore agile projects, Table 2.9 summarises their classification.

Table 2.9. Agile Practices used for Offshore Development (Paasivaara et al., 2009).

No.	Name of Agile Practice	Description
1.	**Daily Scrums**	For coordination and communication which in offshore projects is done via video/audio professional conferencing tools such as Skype (Berczuk, 2007; Danait, 2005; Jensen, 2003).
2.	**Sprint Planning Meeting**	Meeting done at the start of every sprint. If difficult to schedule it for whole team due to time-zone difference, only team leads can hold this meeting using synchronous communication tools. (Berczuk, 2007; Holmstrom, Conchùir, Agerfalk, Fitzgerald, 2006; Layman, Williams, Damian, and Bures, 2006).
3.	**Sprint Review Meeting**	The whole team including the offshore team members are present in this meeting using synchronous communication tools to discuss the progress of the project (Berczuk, 2007).
4.	**Demonstration of Working Functionality**	After each sprint, new functionalities are added due to the sprint review meeting. These new requirements need to be shared via videoconferencing or through desktop sharing (Berczuk, 2007; Danait, 2005; Fowler, 2006).
5.	**Proxy/Remote Customers**	Some companies choose proxy customers in order to improve quality of product and to add a third party

		which communicates with the real customers and the team. The proxy customers can take decisions on behalf of the real team. They communicate with the team via videoconferencing or email (Kircher, Jain, Corsaro, and Levine, 2001; Layman, Williams, Damian, and Bures, 2006; Nisar, and Hameed, 2004).
6.	**Distributed Scrum of Scrums**	This is used when the team is very large. Only scrum masters meet every 2-3 days either in person or via videoconferencing to discuss the progress of the project (Jensen and Zilmer, 2003; Smits, and Pshigoda, 2007).

2.6.3 Effect of Offshoring on Agile Adoption

In the Section 2.3 we identified four key challenges that affect offshore software development, which also affect the adoption of agile practices in offshore software development (Ghafoor et al., 2017). For example, consider the trust issue; it can introduce certain difficulties in agile practices such as dispute over collective ownership of code and maintaining a sustainable pace of the project development. Similarly, issues caused due to socio-cultural differences also affect agile practices such as frequent delivery of code, self-organising teams, embracing change and responding in a timely manner.

For teams wanting to adopt agile, communication and coordination issues are very important as they impact most of the agile practices. Likewise, knowledge and transfer of knowledge are central to the principles of agile software development as they aid in achieving sustainable pace of development, motivated individuals and continuous code review. In Table 2.10 below we have highlighted some agile practices that are affected by the offshore challenges.

Table 2.10. Agile Practices affected by Offshore Challenges.

No.	Offshore Challenge	Agile Practice	Effect of Challenge on agile practice
1.	**Trust**	Collective Ownership	Dispute over code ownership among the onshore and offshore team members
		Sustainable Pace	Difficulties in maintaining a sustainable pace of project development
2.	**Socio-cultural**	Iterative and incremental development	Delays in frequent delivery of code.
		Self-organising teams	Problems in understanding each other's cultural and social values can cause a barrier in the formation of self-organising teams.
3.	**Communication and Coordination**	Sprint Planning	As the team is distributed, due to lack of sufficient communication, it can cause projects in designing a correct sprint.
		Continuous Integration	Multiple versions of code developed at different location can cause any build to break

			due to errors being integrated in the code.
4.	**Knowledge Transfer**	Product Backlog	Any change in product backlog not documented correctly can cause a project to fail.
		Sprint Review	Due to multiple locations of sprint development. It causes problems in determining the progress of the work done.

Table 2.10 demonstrates that agile practices cannot be used as it is in offshore software development and that is why practitioners have been modifying and adapting agile practices. We believe that patterns can make the agile adoption process easier for practitioners.

A case study done by Paasivaara showed how R&D unit of Ericsson, a multinational telecommunication equipment and services company, integrated its global sites into the lean and agile transformation (Paasivaara et al., 2013). As part of this study they observed how Ericsson integrated three sites, which were Finland, Hungary and US. They started the agile adoption process with the Finland site in which they transitioned to agile methodology using an evolutionary model, that is they gradually transformed their processes one at a time and then eventually the whole site shifted to using agile practices for their projects. Table 2.11, demonstrates an overview of how agile practices were adopted at the three sites. Later we have explained what were the success and failures of their transition:

Table 2.11. Detail of Three Sites Adopting Agile Practices at R & D Unit of Ericsson (Paasivaara et al., 2013).

No.	Finland	Hungary	US
1.	Transition Using four phase: • Studying and Planning • Pilot Teams • Full-Scale Roll Out • Maturation and Continuous Improvement	Transitions using two phases: • Piloting • Roll Out	Transition using three phases: • Studying • Planning • Team Start Working
2.	Hiring external consulting firm to train in agile.	Hired a person from a competitor who had started using agile	The acquired US company had a similar product with the existing customer. Finnish team coached them.
3.	Impediment backlog was at Finland.	Didn't have electronic access, however would get updates using phone call or sometimes pictures of the backlog.	The information was made available to the US team members as soon as possible.
4.	Used an evolutionary change approach.	Used a very dramatically change approach.	The management did final team selection.
5.	Most developers in	Developers were mostly	The US team has been

	Finland have over 10 years experience working in Ericsson.	young and inexperience. Hired right out of school.	using lean and agile practices for development and that was evident from their work.
6.	Had more product knowledge compared to Hungarian developers.	Before the transition, the Hungarian team was viewed more as a sub-contracting site working only on specific product parts.	What was different between the Hungary and US teams is that they US team were involved in all the levels of the project.
7.	Causing in competency gaps between the Finland and Hungary developers as a result an exchange program was designed.	Some of the developers left the Hungary team, as they didn't want to adopt agile practices.	The team started integrating with other sites by starting their processes by intensive collaboration planning phase
8.	Language barrier stopped Finnish team members to commit to long stays in Hungary.	Felt the support of the management, as they visited the Hungary team.	Mutual visits between the sites at all levels were encouraged between the US and Finnish teams.
9.	Due to the national and organisational culture, the Finnish team members were more comfortable forming	However the Hungary team members felt they need to be told what they should do.	Finnish experts visiting and supporting at the US site, during the first few months of transition, helped the

	self-organising teams	teams to understand each other's cultural differences.

Based on the above presented transition process between the three sites, Ericsson, found success by following the practices mentioned below:

- Early involvement of global sites
- Broad involvement of at all organisational levels.
- Competence Exchange Program.
- Constant Communication and Cross-site visits.
- Joint Infrastructure.

However having presented the success factors above, Ericsson still faced some challenges during the integration process, which are:

- *Creating a shared understanding of the change*, that is even though the Finland and Hungary sites, were able to create a shared understanding at the management level, but communicating it to the lower levels was difficult.

- *Enabling End-to End Development,* as the concept of delivering a working piece of code within 2-4 week duration was a new concept for the development team.

- *Bridging cultural differences*, as all the three sites had different natural and organisational cultural values.

- *Creating Transparency between the sites,* as the Hungary site felt as if they were not part of the actual team as all the software artefacts were at the Finland site and were not available to the Hungary site electronically. This issue was later resolved by using online wikis.

Researchers such as Holmstrom have done work on how we can solve communication and coordination issues using agile practices as it is such as (Holmstrom et al., 2006):

- **XP Pair Programming-** Helps to increase time overlap, which in turn helps in reducing *temporal distance.* However they did not discuss about how much time would be wasted in order to just coordinate different time zones.

- **Scrum Simple Planning**: Helps increase "team-ness", which in turn helps reduce *geographical distance*. The limitation to this approach is as above mentioned the authors didn't discuss how much effort would be required to communication and coordinate the distributed teams.

- **XP Pair Programming and Scrum Pre-Game Phase**: Helps increase mutual understanding and collaboration between the teams, which in turn helps reduce sociocultural distance. Similarly the researchers didn't discuss how efficiently the teams should collaborate in order to achieve this benefit as in order to collaborate with geographically distributed teams requires coordination of time from all the distributed sites.

2.7 A Study on the Use of Patterns in Agile Adoption in Offshore Development

In this section we have given an overview of patterns currently present in literature for agile adoption and we have also presented an overview of our distributed agile patterns catalogue.

2.7.1 Current Patterns for Agile Adoption

Based on the definition of patterns mentioned in Section 2.6.2.1, we define **agile patterns** as "focus on how an agile practice is being repeatedly modified and used in order to solve a recurring agile problem in a particular context". The difference between an agile practice and agile pattern is that agile practices are a collection of methods and techniques put together to support the application of agile methodology for developing a project, whereas an agile pattern focuses on agile best practices that occur repeatedly while applying agile methodology for developing software. That means an agile pattern consists of best agile practice that is being repeatedly used to overcome a specific challenge in applying a practice.

For example, daily standup meeting is an agile practice, which helps the team to coordinate their daily activity by answering three questions: *i) What did you do yesterday ii) What are you going to do today? and iii) What is getting in your way?*, whereas *daily morning standup meeting* is an agile pattern as it has been observed that conducting daily standup meeting in the morning is more effective than conducting it during mid-day or at the end of work-day. The repeated observation of the modified practice led to the creation of daily morning standup meeting patterns. With the help of this pattern we addressed the challenge of knowledge transfer and communication and coordination issues.

As companies are choosing to develop software offshore, new patterns are being designed. Noll has worked on making a decision support system for offshore development to help practitioners by providing them with patterns that occur in offshoring (Noll et al., 2014). Lescher provided collaboration patterns for global development that he observed in Siemens (Lescher, 2010). He identified 5 patterns which focused on improving the general communication and coordination of the team in offshore development for example one pattern he identified was *Tailored Training* in which the team is co-locate early on in the project for training to familiarize the whole team with technologies needed in the project. Similarly van Heesch presented two collaborative patterns for offshore development, which focused on how the

onshore and offshore team should be formed in order to improve collaboration among the teams (Heesch et al., 2015). Table 2.12 demonstrates the offshore software development patterns presented by practitioners and researchers:

Table 2.12. Existing Offshore Software Development Patterns.

No.	Scope	Detail
1.	**Cultural Patterns**	Based on the exploration and evidence from the literature designed a few patterns and some of them might be considered as anti patterns (MacGregor et al., 2005). Below is the list of their catalogue: • Yes (but No) • Proxy Pattern • We'll-take-you-literally (Anti-Pattern) • We're-one-single-team (Anti-Pattern) • The customer-is-king (Anti-Pattern) Shah presented the idea of "cultural models" as patterns that govern conventional behaviours (Shah et al., 2012). As according to them patterns identified using dimensions specified by Hofstede's and Hampden-Turner, limits the meaning of culture, make culture to be viewed as a static entity and characterises teams based on nationality rather then individuals. Following are models presented by them: • Unproductive Productivity • Hesitant to always say Yes • Owning rather than modularizing
2.	**Communication and Collaboration**	The identified four types of communication patterns, which are as following (Paasivaara et al., 2003):

	Patterns	• Problem Solving • Informing, monitoring and feedback • Relationship Building • Decision Making and Coordination Bricout presented thirteen communication patterns to deal with communication between the team and different parts of the organisation, which are (Bricout, 2004): • One Project • Explicit Communication Strategy • Early Bonding • Face-to-face Every 2 Months • Iterate as you Meet • Common Terminology • Continually Aligned Process • Responsibility Model • Culture Awareness • Flexible Hours • Temporary Engagement • Social Funds • Join for Completion Based on Lescher observation of offshore software development at Siemens, identified five collaborative patterns, which are (Lescher, 2010): • Tailored Training • Co-located Analysis Phase • Onsite Management Visits • Cross-site Delegation • Unfiltered Line Manager Communication

		van Heesch presented two collaborative patterns (van Heesch, 2015) which he linked with the work done by Lescher and Bricout (Lescher, 2010; Bricout, 2004), which are : • Experience Mix • Dissolve Geographical Boundaries
3.	**Patterns for Project Management**	A catalogue was presented for global software development patterns for project management, which included the following (Valimaki et al., 2009): • GSD Strategy • Fuzzy Front-end • Communicate Early • Divide and Conquer each Iteration • Key Roles On-site • Communication Tools • Common Repositories and Tools • Work Allocation • Architectural Work Allocation • Phase-based Work Allocation • Feature-Based Work Allocation • Use Common Processes • Iteration Planning • Multi-level daily Meetings • Iteration Review • Organise Knowledge Transfer • Manage Competence • Notice Cultural Differences
4.	**Testing Patterns**	A research conducted with Technische Universit¨at

		München (TUM) in cooperation with Capgemini, a German software and consultancy company to identify the problems in testing a GSD project and provide a solution to those problems (Pehmöller et al., 2010). They developed 16 testing patterns for offshore projects, which are: • Test cases as Memorandum of understanding of Knowledge Transfer • Light-weight Per-Acceptance Test • Communication on Eye Level • Use tool for Bug Tracking • Moving on-/off Business Analyst • Complementing Testing Attitudes • Align understanding of General Testing Approach • Continuous Integration Testing • Mirrored Team Manager • Extension of the Day • Traceable Test Cases • Tester's Sparring Partner • Complement test Skills • Synchronised Test Environment • Central Test Environment • Evaluation of Constraints of test data
5.	**Requirement Engineering Patterns**	Salger presented "Specification Patterns", which describe practices that analysts should keep in mind while writing down the software requirement specification document for an offshore project (Salger et al., 2010). The identified the following patterns:

		• Define ‘Business Rule’ • Define ‘Use Case’ • On-board Business Analyst during Requirement Engineering • Ship Test Cases • Use Work Packages and Handover Checkpoints • Use Bidirectional Cross References • Map Business Terms to Entity Attributes.

Work has also been done on identifying agile patterns in offshore development. Cordeiro combined organisational patterns and scrum to provide a solution for offshore development (Cordeiro et al., 2007). They identified 6 patterns and proposed a pattern language structure based on literature and later adapted the patterns based on the authors’ experience of running some multi-site projects. Välimäki presented patterns for distributed scrum, which focused on finding practices to support project management in distributed scrum projects that use Application lifecycle management such as *Establish Application Lifecycle Management tool (*Välimäki et al., 2008). Table 2.13 below lists the agile patterns identified so far:

Table 2.13. Patterns for Agile Software Development.

No.	Pattern Name	Detail
1.	**Magic Backlog Patterns**	Identified nine backlog patterns to help practitioners document correct story cards as a need to formalise the backlog increased with the size of the project. The pattern catalogue was designed based on their 50+ years experience in system development, however their focus has been on co-located teams (Hvatum et al., 2015). Their patterns are: • Backlog Frame • Backlog Views

		• Backlog People • Backlog Tales • Backlog Usage Models • Backlog Placeholders • Backlog Plans • Backlog Connections • Backlog Answers
2.	**Daily Standup Patterns**	Fowler presented patterns for daily standup meeting however he didn't consider software being developed offshore (Fowler, 2016). The list of his patterns is: • Work Items Attend • Yesterday Today Obstacles • Improvement Board • Walk the Board • Obstacles are not Raised
3.	**Distributed Project Management and Scrum Patterns**	Nine distributed project management and scrum patterns were presented, which are (Välimäki et al., 2008): • Have a Kick-off Meeting • Make a Release Plan with some Sprints • Organise needed Scrum Roles in each Sites • Establish Application Lifecycle Management Tool • Establish a fast and reliable infra • Establish efficient communication methods • Knowledge Transfer • Visualise Status of Project

		• Have merge Daily Scrums
4.	**Multi-site Software Development Patterns**	A pattern language was designed to manage multi-site software projects (Cordeiro et al., 2007). They combined organisational patterns and scrum to form their pattern language, in which they proposed 6 patterns, listed below: • Surrogate Customer • Stories Rework Subsystem • Inversion of Control • Code Line (Code Ownership) • Integration Build • Plan Bugs on Sustainable Pace
5.	**Agile Adoption Patterns**	Elssamadisy focused on the dynamics of agile adoption that is focused more on understanding the ideas, values and philosophy supporting agility in order to optimise the agile adoption process (Elssamadisy et al., 2006). He designed a pattern catalogue of seven patterns that intended to focus on the granularity of agile practices and how it would help practitioners in adopting agile. The list of this proposed patterns in as following: • Reciprocal Visibility • Static Information Radiator • Dynamic Information Radiator • Evocative Documentation • Standup Meeting • Reaffirmation Ritual

	• Solidarity Ritual

Elssamadisy provided a template for their adoption pattern, which is (Elssamadisy et al., 2006):

- **Name:** Unique name of the pattern.
- **Sketch:** A story that acts as a 'sketch' for the design of the pattern.
- **Context:** Who and in what circumstances the pattern is useful.
- **Forces:** Used to elaborate the context section and give specific issues that can be resolved with the help of the pattern.
- **Therefore:** Description of the pattern.
- **But:** Consequences and limitations of the pattern.
- **How:** Guidelines of how the pattern should be adopted. Also includes **smells** to indicate where a pattern adoption can go wrong.
- **a.k.a:** Similar published patterns.

They proposed the customised format for the purpose of documenting their patterns; they introduced the concept of smells in patterns and introduced the explicit use of "Abstract Patterns". Abstract pattern might be seen as a category of patterns as it lacks an implementation independence of its concrete patterns.

According to Amr Elssamadisy and), in order to achieve successful agile adoption, it is not just enough to focus on what practices should be adopted rather we need to answer additional questions such as (Elssamadisy et al., 2006):

- Which practices do I adopt first?
- Which practices relate to others?
- Can I incrementally adopt a given practice or incrementally adopt from a set of practices?
- Can I adopt the form of a practice without altering its substance?
- Can I add to or delete from a specified set of practices?

- What values and assumptions are presupposed by a given practice?
- And, consistent with the spirit of agility, what business value does each practice deliver?

They identified 7 patterns, which are mentioned in Table 2.13. Overview of Elssamadisy patterns is explained in Table 2.14 (Elssamadisy et al., 2006).

Table 2.14. Overview of Agile Adoption Patterns (Elssamadisy et al., 2006).

No.	Pattern Name	Overview
1.	**Reciprocal Visibility**	Impediment Change is recorded and made visible to the whole team.
2.	**Static Information Radiator**	Scrum Master makes common Scrum Board chart for the whole team to view
3.	**Dynamic Information Radiator**	Daily Standup meetings and retrospective take place in order to exchange information.
4.	**Evocative Document**	Document is done using symbols and informal style so that it is easier to understand.
5.	**Standup Meeting**	Daily standup meeting are conducted with the team and this is a time-boxed activity.
6.	**Reaffirmation Ritual**	The organisation's atmosphere is kept laid-back in order to keep the team motivated to work.
7.	**Solidarity Ritual**	The team has its own consolidated space to work and conduct their meetings.

However their focus was on how to improve communication and coordination among the team members, which are co-located and did not consider the complications added due to distributed teams located in different time zones.

2.7.2 Distributed Agile Patterns

The work done so far is either generic to patterns observed in offshore software development, or the ones that target agile development which tend to focus on project management and coordination of a project. In contrast, our research focuses on identifying distributed agile patterns that will help practitioners' adopt the agile practices in an offshore software development context. We studied many cases from the literature and observed some common practices that companies utilise to overcome the challenges of offshore development by applying an agile methodology. From our observation we found recurring solutions for recurring offshore problems that the team face from requirement gathering to deployment of the project. Building on the previous definition of agile patterns, we define a **distributed agile pattern as adaptation of an agile practice that is being repeatedly applied in order to solve a recurring challenge in a distributed project scenario**. In our research, we applied the systematic literature review and content analysis research methods to develop a catalogue of distributed agile patterns. As a result we have identified 15 distributed agile patterns, which we have organised in four categories. Details about the distributed agile patterns catalogue have been discussed in Chapter 4.

2.8 Chapter Summary

This chapter presented an overview of the work done in the area of agile offshore development. The chapter started by presenting a background on offshore development in which we discussed different offshore models and what are the benefits of offshoring. We then presented a study in which we identified challenges in offshore development and how those challenges affected the software development

process. We also discussed about agile software development and how agile is being adopted in offshore software development. We then presented a study on the use of patterns in agile adoption in software development in which we presented the current patterns that are being used in agile adoption and gave an overview of our distributed agile patterns.

In the next chapter, the research methodology used in this research has been presented.

Chapter 3 Research Methodology

3.1 Introduction

Research methodology can be defined as something people undertake in order to find out things in a systematic way thereby increasing their knowledge (Saunders et al., 2007). There are many ways in which research methods can be explained. In this chapter we will firstly discuss different stages of the research methodology with the help of Saunders's research onion and then we will map those stages to how we carried out our research (Saunders et al., 2007). Based on our research methodology, we have discussed the research philosophy we have followed throughout the research which is realism and then presented our research approach and strategy based on which we designed our research questions and discussed the data source and search strategies used to collect the data for the research. Lastly we have demonstrated the validation process used for validating the results of this research.

3.2 Overview of Research Onion

The research onion was developed by Saunders to illustrate the stages that must be covered when undertaking research (Saunders et al., 2007). Each outer layer in the research onion describes a more detailed stage of research process and contains different ways in which a layer of research can be conducted. This approach provides an effective progression through which a research methodology can be designed and its usefulness lies in the fact that it can adapt to almost any type of research methodology and can be used in a wide range of contexts (Bryman, 2012).

The Figure 3.1 demonstrates an overview of Saunders's Research Model showing the main stages of the research process.

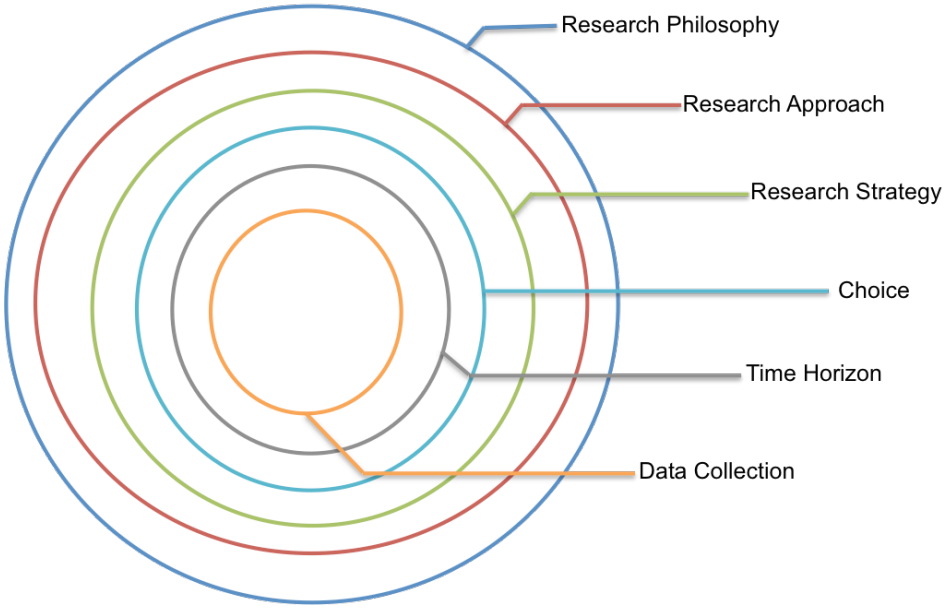


Figure 3.1. The Research Onion (Saunders et al., 2007)

According to Saunders's research onion showed in Figure 3.1, the research needs to start by defining the research philosophy as it helps in creating a starting point for the selection of the appropriate research approach, which according to them is the second step of the research process (Saunders et al., 2007). Once the researcher has selected the research what research approach they will be using they move on to the next stage of the research process in which they will decide what research strategy they will use, that is what plan they will follow to answer their research questions. The fourth step is deciding the research approach that will be used by the researcher, that is are they going to use mono-method, mixed method or multi-method. The fifth step is defining the time horizon for the research, as that will determine the time frame for data collection. The last step according the research onion is data collection, in this stage the researcher decided how he/she will be collecting data for their research, is it going to be by using primary data or secondary data.

In order to explain each phase of the research onion we have designed Table 3.1, which gives an overview of each research step and provides detail of what different type of processes researchers can use to conduct their research in a systematic way with the help of Saunders's research onion (Saunders et al., 2007).

Table 3.1. Overview of Research Onion.

No.	Research Phase	Definition	Type	Detail
1.	**Research Philosophy**	Reflects the way researchers think about the development knowledge, which affects the way, the researcher does his research.	**Positivism**	Assumes that the reality exists independent from the variables under observation. This type of research philosophy is usually considered quantitative as the problem is identified from literature hence only focusing on certain variables.
			Realism	Focuses on the way people make sense of the world, especially by sharing their experiences (Hussey et al., 1997). This type of research is usually considered qualitative, as it requires the researcher to examine real-life events in order to explain how and why certain obstacles occurred during the research.
2.	**Research Approaches**	Defines the approach the research choses to conduct his research.	**Deductive**	In this approach the researcher starts with theoretical proposition and them moves towards concrete empirical evidence. That is the researcher develops the hypothesis upon a pre-existing theory and then

				formulates his research approach to test it. This type of research owes to positivism research philosophy as it allows the researcher to formulate hypothesis and statistical testing for the expected results.
			Inductive	Characterised as to move from specific to general, that observation are considered a starting point and patterns are looked at from the data to formulate a research problem and find its solution. This type of research follows a more realism research philosophy.
3.	**Research Strategies**	It is a plan to answer research questions, which will satisfy the research objectives.	**Experiment**	Refereed to the strategy of creating a research process that examines the results of an experiment against the expected results. This research strategy is used when you have a limited number of factors and you want to study the relationship between them and judge them against the expected research results.
			Survey	Tend to be used in quantitative research projects and involve

		sampling of different proportions of variables.
	Case Study	Used when the researcher wants to gain in-depth understanding of the context of the research problem. Yin also recommended to use case study strategy for when the researcher has little control over events and when the focus in on contemporary events (Yin, 2003).
	Grounded Theory	It is a qualitative methodology that draws on an inductive approach in which patterns are derived from the data as preconditions for the study (May, 2011).
	Ethnography	Involves close observation of people and examines their cultural interactions and their meaning (Bryman, 2012).
	Action Research	Characterised as a practical approach to a specific research problem, within a community of practice (Bryman, 2012). It involves reflective practices, in which professionals get feedback

				on their research, to improve their results.
			Archival Research	In this research strategy, the research is conducted from existing material (Flick, 2012). This form of research may involve systematic literature review, where the researcher examines patterns in order to establish the sum of knowledge on a particular study or to examine the application of existing research to identify a specific problem.
4.	**Choice**	Outlines the research by decided what research approach the researcher will use.	**Mono Method**	As the name suggests it involves the researcher using one approach for his research.
			Mixed Method	In mixed method, the researcher uses two or more methods of research such as using both quantitative and qualitative methodology (Bryman, 2012)
			Multi-Method	This involves that mixed methods combine create a single dataset (Flick, 2011)
5.	**Time Horizon**	Focuses on what time is allocated	**Cross Sectional**	The time frame is already established according to which

		to complete the research.		the data has to be collected.
			Longitudinal	Collection of data repeatedly over an extended time period and is used where an important factor is being examined over a change of time (Goddard et al., 2004).
6.	**Data Collection**	There are a variety of ways through which data can be collected. In quantitative methodology, the researcher is focused on measuring variables or counts their occurrence. Whereas in qualitative methodology, the research emphasises on the experience of a phenomena.	**Primary Data**	Refers to data that is derived from first-hand sources such as data collected through interviews, direct observation, participant observation, questionnaires and surveys.
			Secondary Data	Data is derived from the work or opinions of other researchers (Newman, 1988). For example archival records, organisational documentation, publications and annual reports.

3.3 Research Methodology used to Design Distributed Agile Patterns

In this section we have focused on the how we have used Saunders research onion in order to conduct our research (Saunders et al., 2007).

3.3.1 Research Philosophy

Based on the definition of research philosophy mentioned in Table 3.1 we have used **Realism** research philosophy for conducting our research. Since the focus of this research is to understand how practitioners develop projects offshore while using agile practices. We are studying these variables in the real-life events. This is reflected throughout our research methodology mentioned in Figure 1.1 Research Methodology.

As from Step 1: Review Previous Literature, we are focusing on identifying case studies that demonstrate how practitioners are using developing software at offshore location in order to formulate our research questions, which is Step 2 of our research methodology that is to Identify and Define the Research Problem.

In order to conduct Step 3: Collect Data from Literature and Interviews, to identify recurring agile practices that are being used by practitioners to overcome challenges in offshore development, we focused on examining the how and why practitioners chose to adapt an agile practices and if that solution has been used recurrently to solve the same offshore challenge by other practitioners.

In Step 4: Analysing Data Collected from Literature and Interviews, we used Krippendorff's content analysis (Krippendorff, 2004). Since the data being analysed was based on real-life events in order to explain how practitioners adapted agile practices for their offshore projects, hence this reflects our research philosophy of realism.

Based on the data collected in Step 3 and the analyses done in Step 4. We designed our Distributed Agile Pattern Catalogue. As Step 3 and Step 4 were conducted using real-life data from events, the pattern catalogue is based on patterns observed from literature and by practitioners. Hence Step 5: Design and Develop Distributed Agile Patterns Catalogue has been done following the realism research philosophy.

Step 6: Validate and Evaluate the Distributed Agile Patter Catalogue we involved experts to review our pattern catalogue and based on their feedback Step 7: Modify the Catalogue to Improve the Results is conducted in which we have modified our catalogue.

Based on the overview of each step of our research methodology it is evident that the research philosophy chosen by us for this research is realism.

3.3.2 Research Approach

As mentioned in Table 3.1 there are two types of research approaches that are Deductive and Inductive. The research approach used for this research is **Inductive**. As the research started by reviewing the previous literature on Global Software Development at mentioned in Figure 1.1 Research Methodology's Step 1: Review Previous Literature. In that step we focused on studying cases in literature to identify what are the challenges in Global Software development and based on that study we defined our research problem and formulated our research questions, which is Step 2: Identify and Define Research Problem.

To further refine our research problem we conducted Step 3: Collect Data from Literature Review and Interviews, to identify how specific challenges in offshore software development affect the application of agile practices and we moved towards developing generic distributed agile patterns that practitioners could use while planning to move their processes to offshore locations.

In Step 4: Analysing Data Collected from Literature and Interviews, we go into more detail by using Krippendorff's content analysis approach to analyse the data we collected in Step 3, to get meaningful information (Krippendorff, 2004). Content Analysis is a research tool used to determine the presences of certain words or concepts within text or set of texts. Researchers use it to quantify and analyse the presences of meaning and relationship of words and concepts. They are usually used for the following purposes:

- Reveal internal differences in communication context.
- Detect the existence of propaganda.
- Identify the intentions, focus or communication trends of an individual, group or institutions'.
- Describe attitudinal and behavioural responses to communication.
- Determine psychological or emotional state of a person or group of people.

The reason we selected this approach was to identify the trends of global software development and understand the attitudinal behaviour of offshore development and how it effects the adoption of agile in offshore projects. There are two types of content analysis, conceptual and relational, defined below (Writing @CSU, 2004):

- **Conceptual Analysis:** They only focus of finding the existence and frequency of a text/word but do not focus on if that text/word has any relation with words around it.

- **Relational Analysis:** It is one step further from conceptual analysis and it focuses on examining the relationships among concepts in a text and it also identifies words that have similar meaning.

For the purpose of our research we have used relational analysis as we are not only focusing on counting the frequency of "Global Software Development" rather we also

consider its relationship with agile development and we also considered words with similar meaning. Table 3.2 shows our selected key concepts and their similar meaning.

Table 3.2: Key Concepts Selected for Relational Analysis.

Type	Key Concepts	Similar Meaning
1	**Global Software Development**	Global Software Engineering
		Offshore Software Development
		Offshoring
		Distributed Software Development
		Distributed Development
2	**Agile Methodology**	Agile
		Agile practices
		Agile methods
		Scrum
		XP

Relational Analysis has eight key steps that are (Writing @CSU, 2004):

- Identify the question, as it indicates were the research is headed and why?

- Choose a sample or samples for analysis that is you need to decide how much information is required for analysis. This step needs to be conducted carefully as the sample will determine your results and insufficient samples can limit your results and too much samples can cause difficulties in coding the sample.

- Determine the type of analysis. There are three types of analysis you can select from: Affect extraction, Proximity analysis and Cognitive mapping. The approach we selected was proximity analysis as it evaluates concepts based on co-occurrences because we wanted to analyse global software development

and its effect on agile software development. Where as affect extraction just focused on evaluating individual concepts and cognitive mapping focused on creating models, which isn't relevant to our research.

- Reduce the text to categories and code for words or patterns. At this stage coding is done to identify mere existence of a concept/word.

- Explore the relationships between concepts. That is to determine the degree to which two or more concepts are related to each other and whether that relationship is positive or negative. Lastly focus on the direction of the relationship e.g. impact of X on Y and in our case it would be study the impact of global software development on agile software development.

- Code the relationships. In this step statements and relationships between concepts are coded. Assigning codes to relationships is an effort to determine whether the ambiguous words are just filters or holds information.

- Perform statistical analyses. In this step we explore the difference or look for relationships among the key words/concepts.

- Map out the representations. You can use graphical notations to represent your findings. However we did not use this approach as our focus was on identifying the relationship and developing our distributed agile pattern catalogue.

Detail of the analysis is presented in Section 3.3.3, based on the results of the analysis we moved towards Step 5: Design and Develop Distributed Agile Patterns Catalogue. The main follow of the steps followed to develop and design the catalogue is demonstrated in Figure 3.2 below:

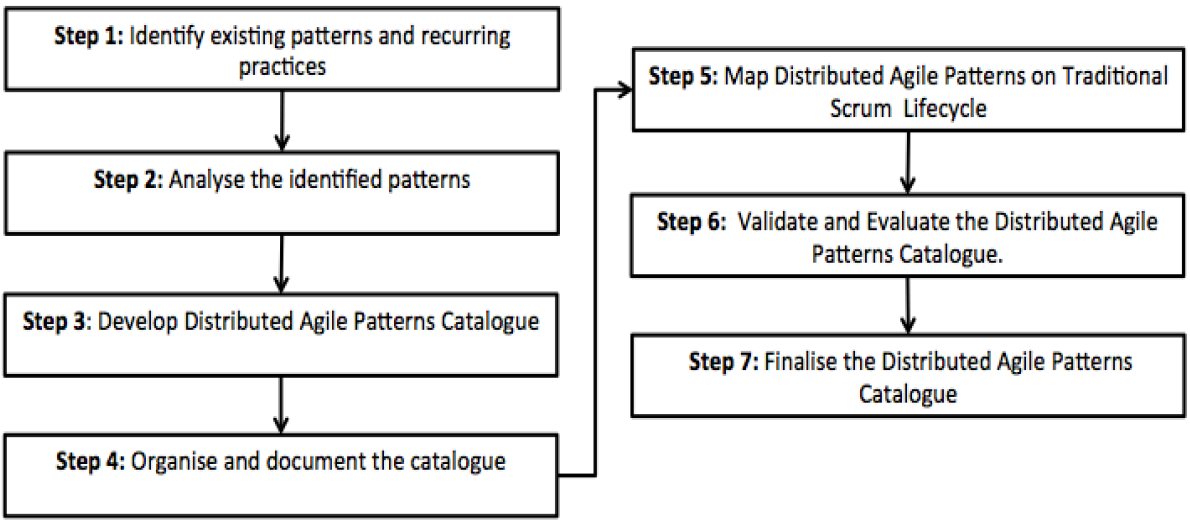


Figure 3.2. Overview of Design and Development of Distributed Agile Pattern Catalogue

An overview of what happens in each step is given below:

Step 1: Identify existing patterns and recurring practices.

As mentioned in Section 3.3.1, the research started by reviewing the existing literature from which we identified our research problem. In order to find a solution for the identified problem, we conducted systematic literature review to identify patterns used to solve challenges in offshore software development. Based on this step we were able to collect data that was required for conducting this research. We also conducted semi-structured interviews to identify patterns. Details of how this step is executed have been presented in Section 3.3.3.

Step 2: Analyse the identified patterns.

In order to focus on collecting the right data for our research, we first filtered case studies based on the identified research questions mentioned in Section 1.3 and to further refine our data we set a selection criteria which was that the papers should address the following questions:

- Does a paper address a challenge in applying any Agile Practice in distributed projects?

- Does a paper discuss any real life experience of using Agile practices for distributed projects?

- Is their any Agile Practice that has been repeatedly adapted and used to solve issues with distributed projects?

Based on the above- identified criteria, we wanted to identify patterns in literature that are being used to solve an offshore development challenge. Further we also focused on identifying solutions that practitioners used to overcome offshore challenges and analyse if any other practitioner has used a similar solution. We have demonstrated in detail the filtration and selection process in Section 3.3.3.

After the data collection phase was completed we moved towards studying literature on existing patterns used in software development. Currently there are six types of patterns mentioned in Section 2.6.2.1.

Step 3: Develop Distributed Agile Patterns Catalogue.

From the systematic literature review and semi-structured interviews, we identified 15 patterns that were being used in distributed agile software development. In our research we have classified an agile practice as a distributed agile pattern if it has been repeated in more then at least 2 articles and has been used to solve a recursive problem. Table 3.4 presents an overview of how many times agile practices were used to solve an offshore challenge in literature, which we further classified as distributed agile patterns.

Step 4: Organise and document the catalogue.

Based on the existing template mentioned in Section 2.6.2.1, we selected Gamma's Design Patterns template in order to preserve familiarity, as they are perceived as the first pattern catalogue to be documented by the software community. Detail of this step has been discussed in Section 4.2.

The distributed agile patterns were organised into four categories based on the type of problem they solved. The four categories are management, communication, coordination and verification patterns. Detail of each category is presented in Section 4.3.

Step 5: Map Distributed Agile Patterns on Traditional Scrum Lifecycle.

The identified 15 distributed agile patterns were mapped onto the traditional scrum lifecycle in order to help practitioners decided which distributed pattern can be used at different stages of the scrum which is shown in Figure 4.2.

Step 6: Validate and Evaluate the Distributed Agile Pattern Catalogue.

In order to validate the pattern catalogue we used reflection workshop and to evaluate it we compared our solution to the existing solutions presented in literature. Detail of this step is presented in Chapter 5.

Step 7: Finalise the Distributed Agile Patterns Catalogue.

Based on the feedback given to us as part of the reflection workshop, we modified our pattern catalogue. In order to avoid confusion for the reader we presented the original patterns in Appendix F and the Revised version of the catalogue in Section 4.5.

According to the above discussion it can be seen that we have used an inductive approach, as the research moves from specific to general. As we first identified and designed a specific research problem and then developed a generic catalogue of distributed agile patterns to be used by practitioners while choosing to develop their projects at offshore locations.

3.3.3 Research Strategy

As defined in Table 3.1, research strategy determines the plan a researcher intends to follow in order to answer the identified research questions. Yin (2003) pointed out that in order to select the correct research strategy, it is important to consider three aspects of the research, which are:

- The type of research questions posed;
- The extent of control and investigator has over actual behavioural events; and
- The degree of focus on contemporary as opposed to historical events.

There are seven ways a researcher can achieve this. We have selected three ways to answer our research questions mentioned in Section 1.3, which are: archival research, case study and action research.

In order to achieve that we start from Figure 1.1 Step 1: Review Previous Literature, in which we study cases from literature, this form of research is referred as archival research as research is conducted from existing material. Systematic Literature Review is a technique to identify, analyse and interpret relevant published primary studies with reference to a specific research questions. It provides a summary of reported evidence available for a given area of interest. It is different from ordinary literature surveys as they are formally planned and methodically executed. Kitchenham recommended to use SLR as a review methodology because it allowed the researcher to (Kitchenham et al., 2007):

- Systematically summarise existing evidence from literature.

- Identify gaps in research.
- Provide a framework to position future research activities.

To identify challenges in offshore development we conducted the first Systematic Literature Review. The details of the SLR are mentioned in Section 2.3 and the studies that were selected as evidence are presented in Appendix A. Based on the literature review we moved to the Step 2: Identify and Define Research Problem. The board objective of this study was defined as to answer the following question:

RQ: What are the recurring adaptations of agile practices that are being used within offshore software development in order to address the identified issues?

To be more specific, the study's focus was on the following two questions:

RQ1: What are the agile practices that are being commonly used to deal with offshore challenges?

RQ2: Are the challenges identified in RQ1 recurring in offshore software development?

As mentioned earlier this type of research strategy involves Systematic Literature Reviews. To identify patterns in offshore software development we conducted the second Systematic Literature Review following Kitchenham and Charters guidelines, which is Step 3: Collect Data from Literature Review and Interviews**.** To conduct this step we searched for papers that are written in English and that where available online. The search strategy includes electronic databases and manual searches of conference proceedings. The following electronic databases are used:

- IEEEXplore (www.ieeexplore.ieee.org/Xplore)
- ACM Digital Library (www.portal.acm.org/dl.cfm)
- Google Scholar (www.scholar.google.com)
- Elsevier Science (www.sciencedirect.com)

- AIS eLibrary (www.aisel.aisnet.org)
- SpringerLink (www.springerlink.com)
- Taylor Francis Online (http://www.tandfonline.com)

We also searched the following conference proceedings for the papers on the use of Agile in offshore software development: *1) Agile Conference and 2) Agile Processes in Software Engineering and Extreme Programming.* The papers ranged from industrial experience reports, theoretical, empirical and experimental academic papers. Figure. 3.3, shows the review process and the number of papers identified at each stage.

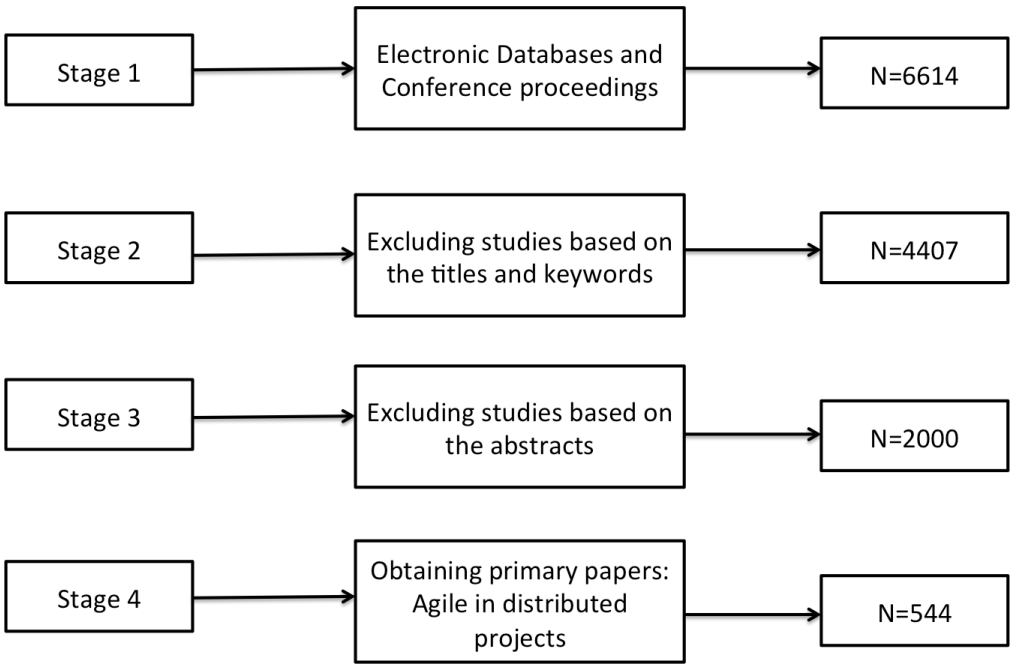


Figure. 3.3. The Selection Process of Primary Papers.

We designed Table 3.3 based on our key concepts selected in Table 3.2. In stage 1, we selected the databases using the search items listed in Table 3.3. Category 1 has more keywords and shows many variations of the same term "Global Software Development". All these search items were combined using the Boolean "AND" operator, which requires that an article that focuses on both Agile and Global Software Development, will be retrieved. Hence we searched every possible combination of the keywords from Category Type 1 AND Category Type 2. Our search excluded articles

that addressed editorials, prefaces, article reviews, discussion comments, news, and summaries of tutorials, workshops, panels and poster sessions. The search strategy resulted in a total of 6614 “hits”.

Table 3.3. Search Terms used in this Review.

Type	Category	Keywords
1	**Global Software Development**	Global Software Engineering
		Global Software Development
		Offshore Software Development
		Offshoring
		Distributed Software Development
		Distributed Development
2	**Use of Agile Practices**	Agile
		Agile practices
		Agile methods
		Scrum
		XP

We used the following screening criteria to ensure that the papers addressed our research topic:

1. Does a paper address a challenge in applying any Agile Practice in distributed projects?

2. Does a paper discuss any real life experience of using Agile practices for distributed projects?

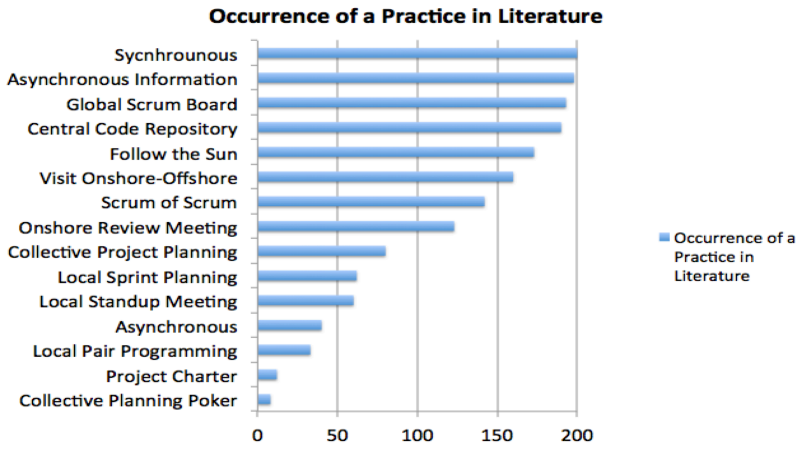
3. Is their any Agile Practice that has been repeatedly adapted and used to solve issues with distributed projects?

In our study we also collected secondary data by considering “lesson learnt” reports based on expert opinion that addressed how Agile practices are used in offshore projects. These 3 points provided a measure of the extent to which we are confident that a selected paper could make a valuable contribution to understand the current use of Agile practices in distributed settings and to identify repeating practices that are being used to solve offshore problems. Each of the 3 criteria was graded on a dichotomous (“yes” or “no”) scale.

We selected 212 papers out of 544 articles by carrying out a quality assessment based on the 3 screening criteria. We accepted a paper that satisfied all 3 criteria and graded as all “yes”. For example we excluded papers that did not show an agile practice being used to solve a repeatedly occurring problem. For example, if an organisation used split pair programming teams but no other organisation used it for better code, we didn’t include that as part of our catalogue. In our research we have classified an agile practice as a distributed agile pattern if it has been repeated in more than at least 2 articles and has been used to solve a recurring problem.

In the Table 3.4, we have shown how many times an agile practice occurred in literature for example synchronous communication practices occurred in over 200 papers while collective planning poker activity occurred in 10 papers. Based on the frequency of a practice we identified patterns.

Table 3.4. Occurrence of Agile Practices in Literature



We also used case study strategy for our research as according to Yin, a case study examines a phenomenon in its natural setting, to answer how and why questions when the researcher has little control over the events. The case study method covers both the phenomenon of interest and its content, resulting in producing a larger number of potentially relevant variables.

According to Yin six sources of evidence should be used for data collection for case study, which are shown in Table 3.5 (Yin, 2003).

Table 3.5. Six Sources for Data Collection Comparison (Yin, 2003).

No.	Data Source	Advantage	Limitation
1.	**Interviews**	• In-depth knowledge and can extract required information.	• Bias due to poorly constructed questions. • Response is bias. • Inaccurate data, due to poor recall of events by the researcher.
2.	**Documents**	• Broad coverage. • Unobtrusive- not created as a result of the case study.	• Biased selectivity if collection is incomplete. • Access may be deliberately blocked • Reporting biases based on the authors' bias reflection.
3.	**Archival Records**	• Precise and quantitative. • Broad coverage of data. • Not created as a result	• Accessibility blocked due to privacy issues. • Biased as it is based on the selection of the researcher.

		of the case study.	
4.	**Direct Observation**	• Covers events in real-time. • Focused on covering the context of the event.	• Time consuming • Added cost based on the hours needed by the human resource. • Events may take place differently as they are being observed.
5.	**Participant Observation**	• Insightful in understanding interpersonal behaviour and motives. • Events are covered in real-time. • Events are focused on context.	• Biased as it is based on the researchers manipulation of results. • Time consuming. • Added cost due to the hours human resource spend on the research.
6.	**Physical Artefacts**	• Insightful into cultural features. • Insightful in understanding technical operations.	• Selectivity- as it depends on the researcher what he selects. • Depends on the availability of the artefacts.

Each source of data has its advantages and limitations. For this research we have used interviews and archival records. According to Easterby-Smith et al. (2012) interviews are the best method of gathering information. There are three types of interviews:

Table 3.6. Type of Interviews (Easterby-Smith et al., 2012).

No.	Type	Definition
1.	**Fully Structured Interview**	Use questionnaire to predefine the questions as standard of research.
2.	**Semi-Structured Interview**	The researcher has a list of themes and questions that cover the phenomenon under study. However these questions can change based on the answers given by the interviewee in order to get more clarification (Robson, 2002; Saunders, 2007).
3.	**Unstructured Interview**	This is used to explore in depth a general area of interest for the researcher. The basic objective is to put the interviewee at ease and allow them to express themselves.

We conducted 20 semi-structured interviews to collect primary data, in which we asked nine open-ended questions covering different aspects of offshoring in order to get expert opinion regarding the practices. According to Easterby-Smith semi-structured interviews are appropriate when (Easterby-Smith et al. 2004):

- The researcher wants to understand what constructs the interviewee uses as basis for his beliefs and opinion about the phenomenon under study.
- Aim is to understand the "respondent's world", so that the researcher might influence it and challenge it.
- Step by step logic is not clear, that is some information be confidential or sensitive and the interviewee might be reluctant to share the true situation.

Jankowicz suggested that semi-structured interviews are a powerful data collection technique when used in case study (Jankowicz, 2005). The reason for using semi-structured interviews in this research is because it provides flexibility to the researcher

to modify the questions in order to understand the phenomenon under study. The sample used to conduct the semi-structured interviews is presented in Appendix D. We also signed consent form from the organisations we interviewed due to confidentiality issues, which is available in Appendix E. The companies we interviewed were selected based on their experience in offshoring and as mentioned in Table 3.7 we further divided them into three categories based on the organisational type, which are startup, breakeven and profitable. Companies that are still in their early stages of earning revenue and have not yet reached breakeven are categorized as *startup;* companies that have reached breakeven based on finances as *breakeven* and lastly the companies that are generating profit as *profitable*. Table 3.7 shows an overview of the organisations that were selected.

Table 3.7. Detail of Companies Interviewed.

Category	Experience in offshoring (years)	Offshore Projects	No. Of Companies
Startup	1-2	3	4
Breakeven	>3	8-9	6
Profitable	>5	>10	10

Based on the collected data, we moved towards Step 4: Analysing Data Collected from Literature and Interviews which has been discussed in Section 3.3.2, based on which we identified 15 distributed agile patterns that is Step 5: Design and Develop Distributed Agile Patterns Catalogue. We validated the catalogue using an action research technique, which is Step 6: Validate and Evaluate the Distributed Agile Patterns Catalogue, of our research methodology.

We conducted a reflection workshop based on Kerth's **"The keep/try reflection workshop"** for verification and validation of the catalogue. For this workshop we invited four organisations to take part (Kerth, 2001). The focus of this workshop was to get feedback from companies in order to obtain their views on the identified patterns. Based on their answers, an assessment was aimed for identifying the completeness of

the catalogue and the usefulness of the identified patterns in adopting agile for developing offshore projects and overcoming offshore challenges. Detail of the validation and evaluation process has been presented in Chapter 5.

3.3.4 Choice

The choice outlined in the research onion mentioned in Table 3.1 includes three methods, which are mono method, mixed method and multi-method. We selected to use multi-method approach to conduct our research as we used both quantitative and qualitative methods for determine our results. The main difference between mixed method and multi-method approach is that is provides the researcher with a wider selection of methods to use (Bryman, 2012).

In multi-method researchers can divide their research into separate segments, with each segment producing a specific dataset; each is dataset is then analysed using techniques derived from quantitative or qualitative methodologies (Felizer, 2012).

As mentioned in Section 3.3.4 we have used both quantitative and qualitative methods for collecting and analysing the data. In Step 3 and Step 4 of our research methodology showed in Figure 1.1 Research Methodology, we started with reviewing the existing literature (Step 1) from which the research questions were formulated (Step 2) based on which we conducted systematic literature review and interviewed practitioners (Step 3) and then analysed the collected data (Step 4). Table 3.8 shows an overview of the quantitative and qualitative methods used in this research.

Table 3.8. Overview of the Quantitative and Qualitative Methods Used in this Research.

No.	Method Type	Used in the Research
1.	**Quantitative Methods**	**Step 1: Review Previous Literature.** **Step 3**: **Collect Data From Literature Review and Interviews:**

		- Systematic Literature Review (SLR). **Step 6: Validate and Evaluate the Distributed Agile Patterns Catalogue:** - For evaluation: Filter the SLR to identify solutions other than patterns and compare how the Distributed Agile Patterns catalogue is a comparatively a better and easy solution.
2.	**Qualitative Methods**	**Step 3: Collect Data From Literature Review and Interviews:** - Semi-Structure Interviews. **Step 4: Analysing Data Collected from Literature and Interview:** - Content Analysis (Relational Analysis). **Step 5: Design and Develop Distributed Agile Patterns Catalogue:** - Filtering the studies identified in the SLR to identify: o Solutions for offshore software development challenges (FSD). o Recurring agile solutions for adoption practices in offshoring. o Patterns in agile adoption in offshoring. **Step 6: Validate and Evaluate the Distributed Agile Patterns Catalogue:** - For validation: Kerth's, "The keep/try reflection workshop". - Evaluation of the catalogue was done by reviewing existing work done on designing solutions other

		then patterns, to overcome offshore development challenges

3.3.5 Time Horizon

The Time Horizon is defined as the time frame within which the intended research is to be completed in. There are two types of time horizons in which research is usually conducted within, which are cross sectional and the longitudinal (Bryman, 2012). For the purpose of this research the time horizon is cross sectional as this research was conducted as part of a PhD Thesis and we had a fix time duration for data collection. However since the duration of the PhD was 4-years we had the flexibility on deciding when the data collection should take place based on the progress of the research.

Table 3.9 shows the time frame in which the research methodology was conducted during the course of the four years of PhD Research.

Table 3.9. Time Horizon across Four-Year PhD Research.

No.	PhD Year	Research Methodology Step Conducted
1.	Year 1 (2013)	**Step 1**: Review Previous Literature. **Step 2**: Identify and Define Research Problem.
2.	Year 2 (2014)	**Step 3:** Collect Data From Literature and Interviews. **Step 4:** Analysing Data Collected from Literature and Interviews.
3.	Year 3 (2015)	**Step 5:** Design and Develop the Distributed Agile Patterns Catalogue.
4.	Year 4 (2016)	**Step 6**: Validate and Evaluate the Distributed Agile Patterns Catalogue.

	Step 7: Modify the Catalogue to Improve the Results. **Step 8:** Write up the PhD Thesis.

3.3.6 Data Collection

The data collection and analysis approach is dependent on the research methodology a researcher decides to use (Bryman, 2012) which is Step 3: Collect Data From Literature Review and Interviews, of our research methodology in Figure 1.1. There are basically two main approaches through which data is collected which are primary and secondary data collection techniques.

In this research we have used both primary and secondary techniques to collect data. We used semi-structure interviews as the primary source of information as it allowed us to discuss in detail the factors the participants considered to affect the development process and how they had been adapting agile practices for their offshore projects. The interviews were open ended, allowing us to probe for more information and gave us the flexibility of rephrasing questions and simplify a specific question if it was too complex for the participant to understand. The detail of the interview has been presented in Section 3.3.3 and the list of questions asked can be found in Appendix D.

For secondary data we studied over 200 cases from literature to identify patterns of how practitioners were adapting agile methods for distributed agile projects and to identify specific challenges in offshore development we conducted systematic literature review, details have been presented in Section 2.3 and the studies that were selected as evidence are presented in Appendix A.

3.4 Chapter Summary

In this chapter we presented the research methodology used to conduct this research. We first gave an overview of Saunder's research onion and then applied it on our research methodology as this helped us present our methodology in a systematic way. The research methodology was presented in six stages, which are research philosophy, research approach, research strategy, choice, time horizon and data collection.

In the next chapter, we have presented our distributed agile patterns catalogue and how practitioners can use it.

Chapter 4 Distributed Agile Patterns

4.1 Introduction

In this section we present the distributed agile patterns that were identified from systematic literature review and semi-structured interviews. We also present how the template for the distributed agile patterns is designed and demonstrate the process through which practitioners can choose which pattern is suitable. Further the catalogue is divided into four categories such as management, communication, collaboration and verification, based on what type of challenge the pattern solves. The distributed agile patterns are then mapped onto the traditional scrum lifecycle to help practitioners choose patterns based on what functionality they want to perform. Lastly the full distributed agile patterns catalogue is presented.

4.2 Designing Template for Distributed Agile Patterns

Patterns are described using a standard format in order to provide structured information that makes them easier to understand and compare. In Table 4.1 we have provided a comparison between the existing different templates that are used for the patterns mentioned in Chapter 2 (Robertson, 1996; Fowler, 1997; Gamma et al., 1997; Buschmann et al. 1996; Brown et al. 1998) such as requirement, analysis, design, architectural, anti-patterns and idioms. The purpose of this comparison is to understand what are the key components required for documenting a pattern.

Table 4.1. Comparison of Existing Patterns.

Requirement Patterns	Analysis Patterns	Design Patterns	Architectural Patterns	Anti Patterns	Idioms
Name	Name	Name	Name	Name	Name

Context	Also known as	Intent	Problem	Problem	Intent
Solution	Evolution	Also known as	Category	Context	Also known
Related Patterns	Structural Adjustment	Motivation	Context	Solution	Motivati-on
	Problem	Applicability	Solution	Consequence	Solution
	Motivation	Structure	Rational	Design Rational	Sample Code
	Context	Participants	Consequence	Example	Known uses
	Applicability	Collaborat-ion	Example	Related Anti Patterns	
	Requireme-nts	Consequen-ces	Known uses		
	Modelling	Implement-ation	Related Patterns		
	Consequen-ces	Sample Code			
	Example	Known uses			
	Known uses	Related			

	Patterns
Related Patterns	

From the above table we can see that all the current templates cover detail descriptions of a problem and its solution. As only graphical notations aren't sufficient to capture the process of an end product, in order to reuse a solution we need to record all the decisions that lead to form a pattern. Similarly we can see all of them have put in sample code or example section as they help in understanding how a pattern can be used in practical scenarios.

In this research, the proposed catalogue is developed based on literature and interviews and we have used Gamma's pattern template in order to preserve familiarity, as they are perceived as the first pattern catalogue documented by the software community. A customised template was then developed to capture the specific findings related to distributed agile practices. The distributed agile patterns template contains the following sections:

- **Pattern Name:** As patterns represent generic knowledge it is vital to give a good name that would make it recognizable and reusable. A good name also helps in facilitating communication among practitioners about the pattern.

- **Intent:** A short statement that highlights the issues and problems that are required to be solved by applying the pattern.

- **Also known As:** The pattern's other well-known names, if any are mentioned in this section.

- **Category:** Based on the similarities of the patterns we grouped them into different categories to be able to provide an abstract view of all the patterns.

- **Motivation:** It consists of the description of the problem and why the pattern should be used in order to avoid the problem from recurring. It provides scenarios that help understand the abstract description of the pattern.

- **Applicability:** Under what conditions the pattern can be applied.

- **Participants:** The participants are those people that are required in applying the pattern.

- **Collaboration:** How participants will coordinate with each other in order to fulfil their responsibilities that are required to complete the projects.

- **Consequences:** Discuss the trade-offs of applying the patterns such as advantages and difficulties faced when applying it.

- **Known uses:** Examples of real scenarios found that follow the pattern in order to provide clarity of how the pattern can be used.

- **Related Pattern:** List of similar patterns in order to identify which patterns can be used together to improve a particular situation.

4.3 Selecting the Right Pattern from the Distributed Agile Patterns Catalogue

Based on the identified distributed agile patterns and how we have organised them. We have developed a process through which practitioners can select which pattern would help them while choosing to develop their software offshore. Following are the steps the practitioner should follow in order to select a distributed agile pattern:

- To provide ease to the practitioners we have used **names** similar to agile practices, for example, if the practitioner wants to use the agile practice daily standup meeting, he can look for **Pattern Name** Local Standup Meeting for

distributed development or for more clarification he can look at **Also Known As**, which is Daily Scrum meeting or daily meeting. Another alternative starting point can be selecting which pattern the practitioner wants to use based on the **Category** the practitioner wants to solve. For example, if they are facing management issues, they can start by looking at the patterns that solve management problems such as distributed scrum of scrum, local standup meeting, local sprint planning, local pair programming and asynchronous retrospective.

- After shortlisting a pattern the practitioner, can further understand what is a pattern does by reading the **Intent**, as it highlights what issues and how the pattern solves it.

- To understand in detail what problem a pattern solved and what is the best way of applying it, the practitioner should read the **Motivation** next.

- Every pattern has some advantages and some limitations. In order to clearly understand them the next section the practitioner should read is **Consequences**.

- To understand under which conditions a pattern is applicable and who should participate and how while applying a pattern, the practitioner should read **Applicability**, **Participants** and **Collaboration** sections.

- To get more information on how existing organisations and practitioners have used this pattern, the practitioner can read **Known Uses**.

- If the practitioner wants to find patterns that are related they can read **Related Patterns** section.

Figure 4.1 shows the follow of how the practitioner should read a distributed agile pattern in order to select a pattern from the distributed agile patterns catalogue for

their offshore projects. A practitioner can either start by reading a patterns name or by the category as we have divided our pattern catalogue into four categories such as management, communication, collaboration and verification patterns. The categories help the practitioner in deciding which type of challenge they want to overcome.

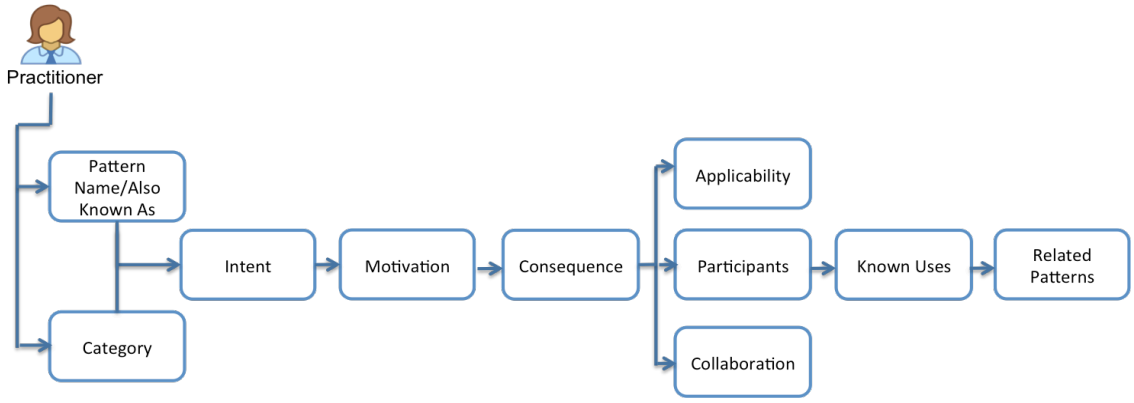


Figure 4.1. Process of selecting patterns from the Distributed Agile Pattern catalogue.

The distributed agile patterns are spread across the Scrum software development lifecycle as shown in Figure 4.2. The practitioner can pick and chose whichever pattern he would like to apply and whichever Scrum phase. The patterns names are written in red and the blue box under them shows on which scrum practices the distributed agile pattern can be applied.

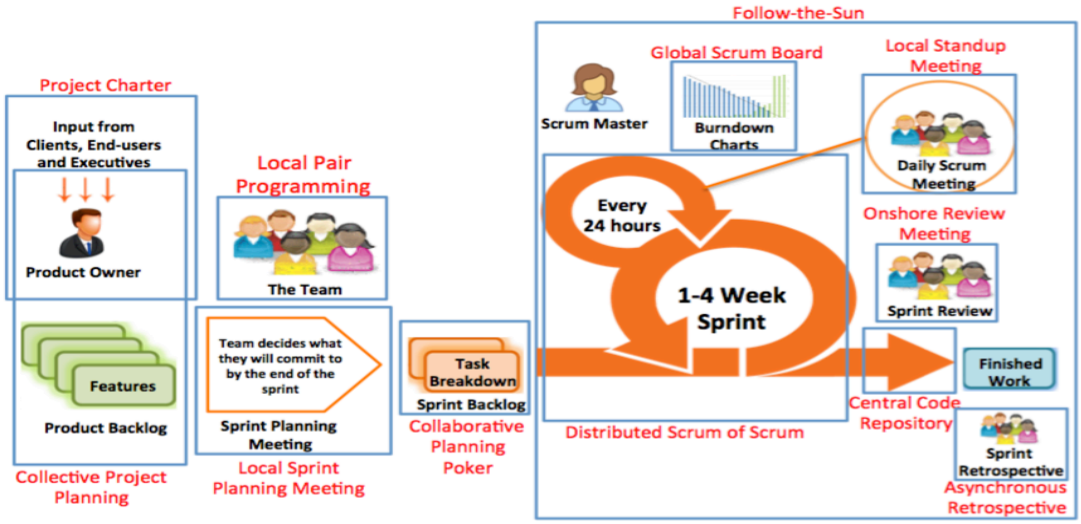


Figure 4.2. Distributed Agile Patterns Application on Traditional Scrum Lifecycle

4.4 Organising the Distributed Agile Patterns Catalogue

Distributed agile patterns vary in their granularity and level of abstraction. Because there are many distributed agile patterns, we have organised them in 4 categories, which are **management**, **communication**, **collaboration** and **verification patterns**. This classification helps in learning and identifying which pattern is to be used in a specific scenario. We have classified distributed patterns based on the problem they solve. Following are the definition of the 4 categories:

- **Management Patterns:** As in offshoring the team is distributed over different time zones, it is difficult to manage all the distributed team members and as they are working on different modules of the project it is difficult to determine the overall progress of the project. In order to handle this problem we use management patterns as they consist of practices that help in managing the onshore and offshore team members and their activities to effectively apply agile in a distributed environment.

- **Communication Patterns:** Since the team is distributed geographically over different time zones they have minimum overlapping working hours, which makes it difficult to maintain real time communication between the onshore and offshore team members. Communication patterns focus on providing solutions to how distributed team members can maintain an effective communication channel in an agile setting using different online tools which provide both synchronous and asynchronous method for communication.

- **Collaboration Patterns:** Even if the nature of the project is offshoring, there is still a need to conduct some collective team activities in order to improve the coordination among the onshore and offshore team members. Because if the onshore and offshore team members don't meet each other it creates mistrust and misunderstanding among the team members, which affects the overall team's productivity. In order to solve this issue collaboration patterns provide

solutions regarding which activities the onshore and offshore team members should conduct together to improve team coordination and project progress.

- **Verification Patterns:** As in agile we focus on building the right product according to the satisfaction of the client. But as the team is distributed on different locations it is difficult to set a standard guideline for all the distributed development sites and how to show project progress to the client. In order to solve this problem verification patterns focuses on how efficiently the clients can get a distributed project developed according to their requirements and monitor the progress of what has been developed.

Table 4.2. Categories of Distributed Agile Patterns.

	Category			
	Management Patterns	**Communication Patterns**	**Collaboration Patterns**	**Verification Patterns**
Pattern Names	Distributed Scrum of Scrum Local Standup meeting Local Sprint Planning Local Pair Programming Asynchronous Retrospective	Global Scrum Board Central Code Repository Asynchronous Information Transfer Synchronous Communication	Collaborative Planning Poker Follow-the-sun Collective Project Planning Visit onshore-offshore	Project Charter Onshore Review Meeting

4.5 Full Distributed Agile Patterns Catalogue

In this section we have presented our full finalised Distributed Agile Patterns Catalogue. In order to avoid causing confusion for the reader we presented the original unrevised patterns in Appendix F.

4.5.1 Management Patterns

In this section we have described the detail of each management pattern.

4.5.1.1 Distributed Scrum of Scrum Pattern

In agile methodology, Scrum is an iterative and incremental project management approach that provides a simple framework that " inspect and adapt" (Hossain, Babar, and Paik, 2009). We observed that in offshore projects the onshore and offshore team practices separate scrums in order to develop the project. Based on this observed practice we have designed the following pattern details:

Pattern Name

Distributed Scrum of Scrum Pattern

Intent

To apply scrum, sub-teams are formed based on location. Each team has its own scrum. Scrum of scrum meetings are arranged to discuss the progress of the project, which is attended by key people.

Also Known As

Scrum meeting or Meta Scrum

Category

Management category as this pattern helps the onshore and offshore team manage their separate scrums and keep each other updated of the project progress.

Motivation

The motivation of this pattern is to address the communication and co-ordination, and knowledge transfer challenges. For example consider a team that is divided into sub-teams based on location and they are working on different tasks of a project. It is difficult to have both onshore and offshore teamwork on the same scrum as they both work on different time zones so in order to work on the same project, both teams work on separate scrums.

To coordinate work both teams arrange a scrum of scrum meeting, which is attended by key people from both teams to update each other of the progress of the project.

Applicability

Use Distributed Scrum of Scrum when:

- Team is distributed over different time zones.
- The overlapping working hours between the onshore and offshore team is less.

Participants

- Distributed onshore and offshore agile team.
- Scrum Masters of agile sub-teams and Product owner.

Collaboration

- Key members from onshore and offshore teams decide time for Scrum of Scrum meeting.

Consequences

The Distributed Scrum of Scrum pattern has the following benefits and liabilities:

1. It prevents the onshore and offshore team from wasting time on collaborating tasks with each other through online tools as both the teams are working on their own scrum so they don't have to wait for each other to complete tasks. This helps overcome the communication and coordination challenges

2. It provides control to both onshore and offshore team to work on their scrum, which avoids the offshore team from having to adjust working hours based on onshore teams availability. This helps overcome the communication and coordination challenges.

3. It allows key people such as Scrum Masters and Product owners' to discuss the progress of the project without having the whole team present which keeps the meeting time boxed and helps in knowledge transfer among the teams.

4. Its limitation is that due to minimum collaboration between the onshore and offshore team, both sub-teams don't feel they are one team.

5. Since only key people attend the Scrum of Scrum meeting, it limits face-to-face interaction of both onshore and offshore team, which affects trust building between both teams.

Known uses

When CheckFree decided to move their work to an Indian offshore consulting firm they used Scrum of Scrum to gather and review the over all team statistics and progress of the project (Cottmeyer, 2008). Similarly, multiple case studies

done on organisations using Scrum for distributed teams also used Scrum of Scrum to coordinate work with offshore team (Hossain et al., 2009; Paasivaara et al., 2009). Siemens also used Scrum of Scrum for two large distributed projects in which the development teams were located in USA, Europe and India. In their Scrum of Scrum meetings they covered technical and managerial issues that occurred in multiple teams (Avritzer et al., 2010)

Related Patterns

Distributed Scrum of Scrum pattern is often used with Local Sprint Planning Pattern and Asynchronous Retrospective meeting Pattern as the onshore and offshore team members are working on different story cards and at the end have their separate retrospective meetings to discuss the sprint

4.5.1.2 Local Standup Meeting

Agile methodology focuses on conducting a daily standup meeting. We observed that in offshore projects the onshore and offshore team conduct their own separate daily standup meetings and use online tools such as Wiki's to share meeting minutes with each other. Based on this observed practice we have designed the following pattern details:

Pattern Name

Local Standup Meeting Pattern

Intent

To discuss daily updates on work done, each local team will conduct their own standup meetings.

Also Known As

Daily Scrum meeting or daily meeting

Category

Management category as this pattern helps the team members of both onshore and offshore team manage their daily activities and update each other with the work done.

Motivation

The motivation of this pattern is to address the communication and co-ordination, and knowledge transfer challenges. For example consider a team that is divided into sub-teams that are located on different time zones. To have a collaborative daily standup meeting is difficult and time consuming as the offshore team either has to come early to work or stay late to attend the meeting. To avoid this, the onshore and offshore team conducts separate local standup meetings in which they answer the following questions:

- What did you do yesterday?
- What are you doing to do today?
- What is getting in your way?

After conducting local standup meetings both onshore and offshore team share, meeting minutes via online tools such as Wiki's to keep both the teams up to date with the progress of the project.

Applicability

Use local standup meeting when:

- Team is distributed over different time zones.
- The overlapping working hours between the onshore and offshore team is less.

Participants

- Distributed onshore and offshore agile team.

Collaboration

- Both onshore and offshore team share-meeting minutes with each other using online tools.

Consequences

The local standup meetings pattern has the following benefits and liabilities:

1. It prevents the offshore team from waiting for the onshore team's availability to conduct the daily standup meeting. This helps overcome the communication and coordination challenges.

2. It allows both onshore and offshore team flexibility to conduct their own standup meeting at whichever time they want. This helps overcome the communication and coordination challenges.

3. It allows the onshore and offshore team to transfer knowledge by sharing meeting minutes with each other.

4. It limits the onshore and offshore team from real-time communication and both team heavily rely on the meeting minutes so any mistake can lead to misunderstanding in the progress of the project.

Known uses

Organisations such as PulpCo (Paasivaara et al., 2009) and Wipro Technologies (Sureshchandra et al., 2008) use local standup meetings for communicating the progress of the project with team members.

Related Patterns

Daily Standup meeting pattern is often used with Global Scrum Board Pattern as once the onshore and offshore team members have conducted their meetings they share the meeting minutes on a shared tool so that both can see the project progress.

4.5.1.3 Local Sprint Planning Meeting Pattern

In agile, a scrum consists of many sprints. The duration of a sprint varies from 2 weeks to 4 weeks depending on the size of the project. At the start of every sprint the team has a sprint-planning meeting in which the team defines the goal of the sprint and prepare the sprint backlog. When the team is divided and is working on different modules of the project it has been observed that the onshore team members and offshore team members conduct their own separate sprint planning meetings. Based on this observed practice we have designed the following pattern details:

Pattern Name

Local Sprint Planning Meeting Pattern

Intent

Each team will have their own sprint planning meetings

Also Known As

Sprint Planning Meeting or Iteration Meeting

Category

Management category as this pattern helps the onshore and offshore team work on their separate module and conduct independent scrum and sprint planning meetings.

Motivation

The motivation of this pattern is to address the communication and co-ordination, and knowledge transfer challenges. For example when a project is distributed to a team that is divided over different time zones, and are working on different modules of the project and are conducting their own scrums. As the onshore and offshore teams conduct their separate scrums, they also conduct separate sprint planning meetings to decide what they will develop

during a sprint. Both teams prepare their sprint backlogs, which are shared using online tools.

Applicability

Use Local Sprint Planning Meeting pattern when:

- Team is distributed over different time zones and is working on different modules/subsystem of the project.

Participants

- Distributed onshore and offshore agile team.

Collaboration

- The onshore team and offshore team share sprint backlog with each other to show the work they will be doing over the next sprint.

Consequences

The Local Sprint Planning Meeting pattern has the following benefits and liabilities:

1. It allows both teams to work independently without having to wait for the onshore team to be available to conduct the meeting, which helps overcome the communication and coordination challenges.

2. It provides control to both onshore and offshore team to work on their scrum and conduct their own sprint planning meetings, which avoids the offshore team from having to adjust working hours based on the onshore team availability. This helps overcome the communication and coordination challenges.

3. Both teams can share their sprint backlog with each other, which provides visibility of the project progress and helps overcome the knowledge sharing challenges.

4. As both the teams are working independently it can cause the teams to feel, as they are not part of one team, rather create an effect that they are two separate teams.

Known uses

When CheckFree decided to move their work to an Indian offshore consulting firm they used local sprint planning meetings to plan their sprint activities (Cottmeyer, 2008).

Related Patterns

Local Sprint Planning Patterns in often used with Distributed Scrum of Scrum Pattern and Global Scrum board Pattern as the meetings minutes of the planning meeting are shared with both onshore and offshore team members.

4.5.1.4 Local Pair Programming Pattern

In agile, pair programming consists of two programmers that share a single workstation that is they share one screen, keyboard and mouse. The programmer using the keyboard is usually called the "driver", the other, is called “navigator” as he is activity giving his remarks on the code and helping the driver to write the code. The programmers are expected to switch roles after every few minutes. It has been observed that when the team is divided on different locations, the team members that are co-located form pairs as it is difficult to form pairs with other locations team members due to the time difference. Based on this observed practice we have designed the following pattern details:

Pattern Name

Local Pair Programming

Intent

Make pair programming teams from the same location.

Also Known As

Pair Programming or Paired Programming

Category

Management category as this pattern helps the local team members to form pairs and work on their story card.

Motivation

The motivation of this pattern is to address the communication and co-ordination challenges. For example when a team is distributed over different locations based on time zones, it is difficult to form pairs between them due to the time difference in working hours. So each team forms pairs based on their location as they are co-located and can work together. Pair Programming helps improve the quality of code as instead of one person writing the code the other person is checking the code.

Applicability

Use Local Pair Programming pattern when:

- Team is distributed over different time zones and is working on different modules/subsystem of the project.

Participants

- Distributed onshore and offshore agile team and working on different story cards.

Collaboration

- The onshore team and offshore team members share a keyboard with a fellow team member from their respective location.

Consequences

The Local Pair Programming pattern has the following benefits and liabilities:

1. The offshore team members don't have to wait for the availability of onshore team members to start work, which helps overcome the communication and coordination challenges.

2. Some organisations feel it's a waste having two resourcing working on the same thing.

Known uses

In a case study conducted by Maruping (2010) on an organisation that had its development centers in India, U.S West Coast, U.S Mid-West and U.S East Coast showed that pairs where made on the bases of physical locations.

Related Patterns

Since in Local Pair programming we are selecting team members from the same location, we often use it with Local Sprint Planning Meeting.

4.5.1.5 Asynchronous Retrospective Meetings Pattern

In agile, when a team is using Scrum at the end of every sprint after the sprint review meeting, a retrospective meeting is conducted which is attended by only the team members and the scrum master. In this meeting the team discusses all the good and bad things that happened during the sprint and how they can improve their work. They also discuss the feedback given by the client. It has been observed that when the team is divided on different time zones, teams conduct their own retrospective meeting, as due to the time difference it is difficult to have a collective retrospective at the end of each sprint (Kamaruddin, 2012). Once both the onshore and offshore teams have conducted their retrospective meeting they share the meetings minutes with each other using online tools. Based on this observed practice we have designed the following pattern details:

Pattern Name

Asynchronous Retrospective Meetings

Intent

Teams conduct separate retrospective meetings based on location and share the key information via email. The Scrum Masters discuss possible improvements with the team based on the feedback from the client.

Also Known As

Retrospective Meetings or Iteration Retrospective

Category

Management category as this pattern helps the onshore and offshore team members to review their sprint and discuss their performance. The Scrum Master advises the team on how they can improve their performance and discusses the feedback of the client.

Motivation

The motivation of this pattern is to address the communication and co-ordination, and knowledge transfer challenges. For example when a team is divided on different time zones and are working on different modules/subsystems of a project and conducting their own independent Scrum and sprints it is difficult to have a collective retrospective meeting. The onshore and offshore team members conduct their separate retrospective meetings to discuss what went good and bad in the sprint and how they can improve their work in the next sprint. The Scrum master attends these meetings and gives feedback on the performance on the team and discusses the remarks given in the sprint review meeting by the client. Once both teams have conducted their retrospective meetings they share the meeting minutes with each other using an online tool.

Applicability

Use Asynchronous Retrospective Meetings when:

- Team is distributed over different time zones and is working on different modules/subsystem of the project.

Participants

- Distributed onshore and offshore agile team members.
- Scrum Master

Collaboration

- The onshore team and offshore team members share meeting minutes at the end of their retrospective meetings.

Consequences

The Asynchronous Retrospective Meetings pattern has the following benefits and liabilities:

1. It allows onshore and offshore team members to conduct retrospective meeting independently of each other's availability, which helps overcome the communication and coordination challenges.

2. It helps team members discuss their independent problems and doesn't make both onshore and offshore team to be present for the meeting and can share meeting minutes using online-sharing tools. This helps overcome the knowledge transfer challenges.

3. Since onshore and offshore team members conduct separate retrospective meetings they don't understand each other's problems.

Known uses

Elastic Path, a Vancouver, British Columbia-based company decided to offshore their work to Luxsoft, an outsourcing partner used asynchronous retrospective

sessions to discuss the sprint progress and well as what improvements they can make. Once all locations had conducted their retrospective meetings, they posted comments and results on SharePoint, which were then viewed by Scrum Master and technical lead for their remarks (Vax, 2008).

Related Patterns

We often used Distributed Scrum of Scrum Pattern with Asynchronous Retrospective meeting Pattern. It is also often used with Local Sprint Planning Pattern as in order to review the progress of a sprint and the team we use retrospective meeting. After all the distributed teams have conducted their retrospective meetings they share the meeting minutes with each for which they use Global Scrum Board Pattern.

4.5.2 Communication Patterns

In this section we have described the detail of each communication pattern.

4.5.2.1 Global Scrum Board Pattern

Agile has many artefacts such as product backlog, sprint backlog, storyboard, task board, team velocity and burndown charts which help the team in managing the project. It has been observed that when the team is divided to different locations they maintain a online record of all these artefacts so that they can share them with each other using online tools such as Wiki's, Rally and Jira (Danait, 2005; Berczuk, 2007; Cottmeyer, 2008). Based on this observed practice we have designed the following pattern details:

Pattern Name

Global Scrum Board Pattern

Intent

An online-shared Scrum board, will be used by, both onshore and offshore teams to view the product backlog, storyboard, task board, burn down charts and other agile artefacts using online tools

Also Known As

Scrum Board or Agile Story Board

Category

Communication category as this pattern helps the onshore and offshore team communicate with each other using an online tool to view each other's work and understand the progress of the overall project.

Motivation

The motivation of this pattern is to address the trust, socio-cultural; communication and co-ordination, and knowledge transfer challenges. For example when a project is distributed to a team that is divided over different time zones, and are working on different modules of the project, to share their work they use an online tool to display agile artefacts. Based on the work done by both teams it is easier to see the progress of the project and it helps understand if there is a problem with a team.

Applicability

Use Global Scrum Board pattern when:

- Team is distributed over different time zones.

Participants

- Distributed onshore and offshore agile team.

Collaboration

- The onshore team and offshore team share agile artefacts with each other to show their progress.

Consequences

The Global Scrum board pattern has the following benefits and liabilities:

1. It allows the whole team to see the requirements, which creates visibility of the project and helps in overcoming trust challenges. The scrum board is designed keeping the socio-cultural differences in mind.

2. It allows the onshore and offshore teams to understand the progress of the project, which helps overcome the communication and coordination challenges.

3. It increases the visualization of the work done by each team, which helps overcome knowledge transfer challenges.

Known uses

FAST, a search company with headquarters in Norway while building a search application on top of their core search platform experimented with couple of online tools to keep both teams updated with the progress of the project. They tired XPlanner and Jira and settled for Jira, which is a web-based tool that allowed the remote team members to view the backlog and update tasks whenever they wanted (Berczuk, 2007). Similarly in a study done by Cristal on an organisation that has development centres across North America, South America and Asia concluded with that the use of a global scrum board could help improve the productivity of global agile teams (Cristal et al., 2008). Similarly companies like Valtech (Danait, 2005), Telco (Ramesh et al., 2006), BNP Paribas (Massol, 2004), Aginitys LLC (Armour, 2007) and SirsiDynix (Sutherland et al. 2007) used online tools to share agile artefacts with their offshore team members.

Related Patterns

Global Scrum board pattern is often used with Central Code Repository Pattern as the team shares all the agile artefacts and code using an online tool.

4.5.2.2 Central Code Repository Pattern

In agile, when a team is using Scrum and XP, the team members are divided in pairs of two and are working on different tasks during a sprint. When a task is completed the team members commit their code to a share repository for continuous integration of the code. It is observed that even when the team members are geographically apart they still use a share code repository where they commit their code so that all the team members can see the code as well as determine the progress of the project. Based on this observed practice we have designed the following pattern details:

Pattern Name

Central Code Repository

Intent

The whole team will maintain a central code repository so that both team can see each other's code and see the progress of the work done.

Also Known As

Source Code Repository or Global Build Repository

Category

Communication category as this pattern helps the onshore and offshore team members to write code and share it on a central code repository where all team members can see the code and edit it if required.

Motivation

The motivation of this pattern is to address the communication and co-ordination, and knowledge transfer challenges. For example when a team is divided on different time zones and are working on different modules/subsystems of a project they use a central code repository to share their work with all team members. They can use online tools such as GitHub for committing their code and maintain versions of the project (Räty, 2013). This

helps the whole team to see the code and provides visibility of the project progress.

Applicability

Use Central Code Repository when:

- Team is distributed over different time zones and is working on different modules/subsystem of the project.

Participants

- Distributed onshore and offshore agile team members.

Collaboration

- The onshore team and offshore team members share a keyboard with a fellow team member from their respective location and once they have finished a task they commit their code to a central code repository.

Consequences

The Central Code Repository pattern has the following benefits and liabilities:

1. It allows onshore and offshore team members to view each other's code, which helps overcome communication and coordination challenges.

2. It helps in determining the progress of the project, which helps overcome knowledge transfer challenges.

3. As all the team is committing to a central repository, if a team commits code with errors it can affect the whole build of the project.

Known uses

WDSGlobal is a leading global provider of knowledge-based services to mobile operators, manufacturers and application and sales channels. In 2004 they combined their developments, which were located in UK, USA and Singapore.

They shared their code on a central code repository to minimize duplications and reduce cost of maintenance (Yap, 2005). Many companies use central code repository for their distributed projects such as Extol International (Kussmaul et al., 2004), Valtech (Danait, 2005), Manco (Ramesh et al., 2006), Aginity LLC (Armour, 2007), SirsiDynix (Sutherland et al., 2007), CEInformant (Bose, 2008) and ABC Bank (Modi et al., 2013).

Related Patterns

Central Code Repository pattern is often used with Global Scrum Board Pattern.

4.5.2.3 Asynchronous Information Transfer Pattern

Agile emphases on close face-to-face communication between the team members rather than detailed documentation. When a team is distributed on different time zones it has been observed that the teams adopted asynchronous tools for sharing information with each other such as emails, wikis and SharePoint. Based on this observed practice we have designed the following pattern details:

Pattern Name

Asynchronous Information Transfer

Intent

Due to the time difference between the onshore and offshore team use online tools to exchange information with each other. Each team should response to queries within 12 hours.

Also Known As

Information Transfer or Knowledge Sharing

Category

Communication category as this pattern helps the onshore and offshore team members to answer each other's queries within 12 hours.

Motivation

The motivation of this pattern is to address the communication and co-ordination, and knowledge transfer challenges. For example When a team is divided on different time zones they may have queries about work but due to the time difference they cannot get a direct reply at that time so they use emails to communicate queries, which are then answered within 12hours max. Organisations have set standards for response time in order to avoid delays in work (Vax, 2008).

Applicability

Use Asynchronous Information Transfer when:

- Team is distributed over different time zones.

Participants

- Distributed onshore and offshore agile team members.

Collaboration

- The onshore team and offshore team members share information and ask queries using asynchronous tools.

Consequences

The Asynchronous Information Transfer pattern has the following benefits and liabilities:

1. It allows onshore and offshore team members to exchange information when synchronous communication cannot be conducted due to working hours time difference. This helps overcome the knowledge transfer challenges.

2. It helps team members from waiting for onshore team member availability to ask a query. This helps overcome the communication and coordination challenges.

3. If the team members don't respond timely it can cause delays in the project.

Known uses

VTT Technical Research Centre of Finland and National University of Ireland conducted a research on two organisations that were developing a system together. One organisation was a customer organisation in U.S and the other organisation was a development organisation located in Ireland. Based on their findings the companies used asynchronous tools for communication. They used wikis for storing documents and meeting minutes and used Emails for decisions and queries (Korkala, 2010). Similarly Valtech used Twiki for asynchronous communication (Danait, 2005).

Related Patterns

Asynchronous Information Transfer pattern is often used with Global Scrum Board and Synchronous Communication Pattern.

4.5.2.4 Synchronous Communication Pattern

Agile emphases on close face-to-face communication between the team members rather than detailed documentation. When a team is distributed on different time zones it has been observed that the teams adopted synchronous tools for sharing information with each other such as voice, video conferencing and document sharing tools. Based on this observed practice we have designed the following pattern details:

Pattern Name

Synchronous Communication Pattern

Intent

In order to discuss issues the teams uses synchronous tools for voice, video conferencing, document sharing, application sharing etc.

Also Known As

Synchronous Knowledge Transfer

Category

Communication category as this pattern helps the onshore and offshore team members to answer each other's queries within 12 hours.

Motivation

The motivation of this pattern is to address the trust, socio-cultural communication and coordination, and knowledge transfer challenges. For example when a team is divided on different time zones they may have queries which they can discuss with the onshore team using synchronous tools to get real-time response (Vax, 2008).

Applicability

Use Synchronous Communication Pattern when:

- Team is distributed over different time zones.

Participants

- Distributed onshore and offshore agile team members.

Collaboration

- The onshore team and offshore team members share information and ask queries using synchronous tools.

Consequences

The Synchronous Communication pattern has the following benefits and liabilities:

1. It allows onshore and offshore team members to exchange information. This helps overcome knowledge transfer, communication and coordination challenges.

2. Team members can ask each other questions which builds trust and help understand each others socio-cultural differences, which helps overcome trust and socio-cultural challenges.

3. It assists team members from waiting for onshore team member availability to ask a query. This helps overcome the communication and coordination challenges.

4. If the team members don't respond timely it can cause delays in the project.

Known uses

CampusSoft is a UK based company that used synchronous communication when they moved to Agile with their offshore suppliers in India and Romania. They used video conferencing facilities for planning sessions and later shifted to WebEx sessions and GoToMeeting so that they could share desktops with the remote team members. For daily Scrum meetings they preferred to use Skype call and made everyone wear headsets to make the meeting easier. For sprint review meetings they used sharing desktop tools as well as conference phones so that members from both end could talk with each other (Summers, 2008).

Related Patterns

Synchronous Communication pattern is often used with Global Scrum Board and Asynchronous Information Transfer Pattern.

4.5.3 Collaboration Patterns

In this section we have described the detail of each collaboration pattern.

4.5.3.1 Collaborative Planning Poker Pattern

An agile team plays planning poker to put point's estimation on each story card. The product owner also takes part in this activity. He tells the team the intent and value of a story card based upon which the development team assigns an estimation on the card. Based on the points assigned the team members who assigned the lowest and highest estimation will justify their reasons. The team will have a brief discussion on each story and assign an estimation upon which the whole team agrees on.

It has been observed that even when the team is distributed the planning poker activity is conducted when both teams are co-located for the project planning activity. Based on this observed practice we have designed the following pattern details:

Pattern Name

Collaborative Planning Poker Pattern

Intent

Onshore and offshore team members take part in this activity.

Also Known As

Planning Poker or Scrum Poker

Category

Collaborative category as this pattern helps the onshore and offshore team to discuss the duration of a story card.

Motivation

The motivation of this pattern is to address the trust, socio-cultural, communication and co-ordination, and knowledge transfer challenges. For example when a project is distributed to a team that is divided over different time zones, it is important that all the team members agree on the time duration of a feature before they start developing the project. This helps estimate the duration of the project completion as well as it provides visibility

of project progress. For this purpose the onshore and offshore team members play planning poker in order to collectively agree on the estimation of a story card. Once the estimation is decided they write it down and approved by the product owner/client and move on to the next story card, till all the story cards are estimated.

Applicability

Use Planning Poker pattern when:

- Team is distributed over different time zones and will be working on different story cards in a sprint.

Participants

- Distributed onshore and offshore agile team.
- Product owner/Client.

Collaboration

- The client approves the estimation made by the team members.

Consequences

The Planning Poker pattern has the following benefits and liabilities:

1. It allows the onshore and offshore teams to agree on a story card estimation, which helps the team establish their team velocity. Since members from both locations are present during this activity, this helps overcome trust and socio-cultural challenges.

2. It provides the product owner/client with estimation of project completion, which helps overcome the communication and coordination, and knowledge transfer challenges.

3. If all team members don't agree on estimation on a story card it can lead to a long discussion, resulting the planning poker to prolong.

Known uses

UShardware has development centres across North America, South America and Asia. When transitioning to distributed agile environment they used planning poker activity for estimation of their story cards (Wildt et al., 2010).

Related Patterns

Planning Poker Pattern is often used with Collective Project Planning as its better to conduct this pattern when the whole team is co-located. The estimated story cards are then shared on the Global Scrum board so that whole team can view them during the project.

4.5.3.2 Follow-the-Sun Pattern

When a agile team is distributed over different time zones it has been observed that companies cut down on cost by increasing development time by adopting "follow the sun" workflow which means it allows 24 hours development due to the difference in time zones allowing a company's employees to do development 24hrs a day (Carmel et al., 2001; Yap, 2005; Kroll, et al., 2012). Based on this observed practice we have designed the following pattern details:

Pattern Name

Follow-the-Sun Pattern

Intent

Onshore and offshore teams will work 9:00 a.m-5:00 p.m. according to their own time zones.

Also Known As

24-hours Development Patterns

Category

Collaboration category as this pattern helps the onshore and offshore team manage their work and handoff their work to the other team before they leave from work.

Motivation

The motivation of this pattern is to address the communication and co-ordination, and knowledge transfer challenges. For example consider a team that is divided into sub-teams that are located on different time zones. To adjust the working hours of the offshore team for both the onshore and offshore team work together is difficult as the offshore team has to come in the evening and stay till early morning. This makes the offshore team feel they are less important in comparison to the onshore team and it affects the employees' social life.

In order to avoid that, the onshore and offshore team use "follow-the-sun" approach. For example an employee works from 9:00 a.m. to 5:00 p.m. in the USA. At 5:00 p.m. she hands over the incomplete task to a colleague in Australia who works from 9:00 a.m. to 5:00 p.m. based on his time zone. At 5:00 p.m. according to his country, he transfers the updated task to a colleague in Poland who works on the updated task for the next eight-hours and then forwards it to his colleague in the USA (Gupta, 2009).

While the employee in the USA had left work two of her colleagues worked on her task as when she will come to the office next morning a lot of the work would have been done. This work scenario takes advantage of the geographical distances as it allows people from different time zones to work round-the-clock in order to build software (Gupta, 2007).

The work distribution among the team can be done in two ways. First either the 3 teams distributed on different geographical locations work on the same task and each team keeps updated the task as mentioned in the above scenario or

secondly a most efficient way is that we divided different aspect of the same problem among the team for example in Figure 4.3 we can see how a problem has been distributed among 3 teams (Gupta, 2009):

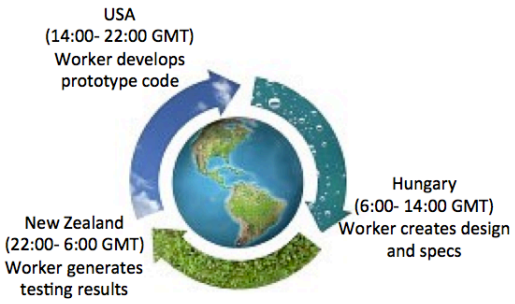


Figure 4.3. Distribution of Work among Three Distributed Teams (Gupta, 2009).

Applicability

Use follow-the-sun pattern when:

- Team is distributed over different time zones.
- The onshore and offshore teams are working on different tasks.

Participants

- Distributed onshore and offshore agile team.

Collaboration

- Both onshore and offshore team hand over their work to each other at the end of every working day using online tools.

Consequences

The follow-the-sun pattern has the following benefits and liabilities:

1. It allows continues development during different working shifts across different time zones, which helps overcome the communication and coordination challenges.

2. It allows both onshore and offshore team to work according to their time zone and share their work without having to either come early to work or stay late till early morning. This helps overcome the knowledge transfer challenges.

3. It reduces the development life cycle or time-to-market (Denny, et al., 2008).

4. It limits the onshore and offshore team from real-time communication as the overlapping working hours between different time zones can be less or zero.

Known uses

Yahoo! used follow-the-sun approach when they offshored their Yahoo! Podcast product from Sunnyvale, California campus to Yahoo! Bangalore, India Campus (Drummond et al., 2008). Similarly organisations like WDSGlobal (Yap, 2005) and Wipro Technologies (Sureshchandra, et al., 2008) use follow-the-sun approach.

Related Patterns

Follow-the-sun Pattern is often used with Local Standup meeting Pattern and Distributed Scrum of Scrum Pattern as both onshore and offshore team members are working separately.

4.5.3.3 Collective Project Planning Pattern

Agile focuses on individuals and interactions over processes and tools. While planning for the project the whole team is present. Unlike the traditional development where a project manager hands a project plan to the team. In agile the whole team takes part in the planning activity in order to determine when and how the project will be developed. It has been observed that even if the project is of a distributed nature it is better to co-locate the team onshore and offshore team for the project planning activity. Based on this observed practice we have designed the following pattern details:

Pattern Name

Collective Project Planning Pattern

Intent

Both the onshore team and the offshore team will collectively work in the project-planning phase. Once both team have engaged in the project planning activity, the team will prepare the project backlog.

Also Known As

Project Planning or Agile Project Planning

Category

Collaboration category as this pattern helps the onshore and offshore team to work together and come up with a project plan.

Motivation

The motivation of this pattern is to address the trust, socio-cultural communication and co-ordination, and knowledge transfer challenges. For example consider a team that is divided into sub-teams that are located on different time zones and both the teams come to one location to do the project planning activity. In the beginning of any distributed project, the offshore team is invited to the onshore location so that they may work together and understand each other's requirements.

While the teams are co-located they worked on preparing the product backlog and they spend at least one or two sprints together before the offshore team leaves and starts working on the project (Cottmeyer, 2008; Therrien, 2008). This helps the onshore team by making the offshore team understand their working style and work standard.

Applicability

Use Collective Project Planning pattern when:

- Team is distributed over different time zones.

Participants

- Distributed onshore and offshore agile team.

Collaboration

- Onshore team and offshore team work together to make a product backlog.

Consequences

The Collective Project Planning pattern has the following benefits and liabilities:

1. It allows the onshore and offshore teams to work together and understand each other. This helps build trust among the team members and overcome communication and coordination challenges.

2. Onshore team works with the offshore team and makes them understand what type of work they want. This helps overcome the socio-cultural and knowledge transfer challenges.

3. It adds additional cost of travel and stay of the offshore team at the onshore location.

Known uses

FAST, a search company with headquarters in Norway while building a search application on top of their core search platform used collective project planning to co-locate the team and make them work together in project planning activities (Berczuk, 2007). Siemens also used collaborative planning for their distributed projects (Avritzer et al., 2010; Avritzer et. al, 2007) in which team members from multiple sites got involved in the early stages of the project in order to create an open communication channel and high level of trust among the distributed team members (Avritzer et. al, 2010)

Related Patterns

Collective Project Planning Pattern is often used with Project Charter Pattern as it provides a central document that consists of the goal and objectives of the project written by the client.

4.5.3.4 Visit Onshore-Offshore Team Pattern

As agile emphases on close face-to-face communication between the team members it has been observed that when the team is divided on different time zones, the team members travel quarterly or annually to visit each other. This activity helps build trust among the team members and helps them understand each other's cultural differences (Ramesh, 2006; Therrien, 2008; Summers, 2008; Paasivaara et al., 2014). Based on this observed practice we have designed the following pattern details:

Pattern Name

Visit Onshore-Offshore Team Pattern

Intent

Both onshore and offshore teams should quarterly / annual visit each other in order to build trust, exchange cultural values and improve team coordination.

Also Known As

Travel Onshore-Offshore

Category

Collaboration category as this pattern helps the onshore and offshore team members to co-locate and understand each other and build a good relationship, which improves team coordination.

Motivation

The motivation of this pattern is to address the trust, socio-cultural communication and co-ordination, and knowledge transfer challenges. For

example when a team is divided on different time zones they don't feel that they are both part of one team and they don't trust each other. They don't understand each other's cultural values and work ethics. In order to solve these issues the onshore and offshore team visits each other to develop the feeling of trust and understand each other's cultural and working values. During these visits they attend training together as well as engage into informal activities to better understand each other. This helps build a bond between the team members, which results in good team coordination.

Applicability

Use Visit onshore-offshore Team when:

- Team is distributed over different time zones.

Participants

- Distributed onshore and offshore agile team members.

Collaboration

- The onshore team and offshore team members visit each other to improve team coordination.

Consequences

The Visit onshore-offshore Team pattern has the following benefits and liabilities:

1. It allows onshore and offshore team members to exchange cultural values with each other and work ethics. This helps overcome socio-cultural and communication and coordination challenges.

2. It helps team members to feel they are part of one team, which develops trust among onshore and offshore team members. This helps overcome trust and knowledge transfer challenges.

3. The travelling adds additional cost to the project budget.

Known uses

Ericsson is a Swedish multinational provider of communications technology and services. To build a XaaS platform and a set of services they used agile software development methodologies. The development team was distributed over 5 sites located in 3 countries. Four of the sites were located in Europe and one was located in Asia. They conducted workshops, which were attended by team members from different locations. The purpose of these workshops was to create a common vision for the whole organisation by setting common values as well also to improve the collaboration between the sites, thus build trust (Paasivaara et al. 2014).

Related Patterns

Visit onshore-offshore Team pattern is often used with Collective Project Planning Pattern as planning is better done when the whole team is co-located.

4.5.4 Verification Patterns

In this section we have described the detail of each verification pattern.

4.5.4.1 Project Charter Pattern

In project management, project charter is a statement that defines the scope, objectives and participants of a project. It is used to explain the roles and responsibilities, outline of the project objectives and identify main stakeholders. It has been observed that while starting a distributed project using agile many organisation use project charter to clarify the goals and objectives of the project to both onshore and offshore team (Galen, 2009). Based on this observed practice we have designed the following pattern details:

Pattern Name

Project Charter Pattern

Intent

Before starting the project planning activity, agile teams use project charter in order to have a central document between the onshore and offshore team that defines the project.

Also Known As

Project Definition or Project Statement

Category

Verification category as this pattern helps the onshore and offshore team to have a central document clarifying the project goals and objectives, which is written by the product owner/client.

Motivation

The motivation of this pattern is to address the trust, communication and co-ordination, and knowledge transfer challenges. For example when a project is distributed to a team that is divided over different time zones, a central document is written known as the project charter, which clarifies the onshore and offshore team the goals and objectives of the project. It also identifies the roles and responsibilities of the onshore and offshore team. The purpose of this activity is to have a document that helps the team in the project-planning task.

Applicability

Use Project Charter pattern when:

- Team is distributed over different time zones.

Participants

- Distributed onshore and offshore agile team.
- Client.

Collaboration

- The client gives the project charter to the onshore team and offshore team to clarify the goals of the project.

Consequences

The Project Charter pattern has the following benefits and liabilities:

1. It allows the onshore and offshore teams to understand the project. This helps overcome communication and coordination, and knowledge transfer challenges.

2. Since it is a single document stating the goals and objectives of the project it helps establish trust between the onshore and offshore team members.

3. It is intended to clearly set the stage for the project by aligning the team and settings goals and expectations.

Known uses

IONA Technologies used Project Charter for their distributed projects in order to have a central document that clarifies the goals of the project to both onshore and offshore team members (Poole, 2004). Similarly in a case study conducted by Brown (2011) on Agile-at-Scale Delivery it was observed that organisations use project charter.

Related Patterns

Project Charter pattern is often used with Visit onshore-offshore Team pattern.

4.5.4.2 Onshore Review Meeting Pattern

In agile, at the end of each sprint, a sprint review meeting is held in which the team meets with the clients and shows them the work they have done through a demo. In this meeting the client gives remarks about the work and if they require any changes

they tell the team. It has been observed that when the team is distributed on different locations, then the team that is co-located with client conducts the review meeting. Based on this observed practice we have designed the following pattern details:

Pattern Name

Onshore Review Meeting

Intent

The onshore team will present the demo as they are located where the client is.

Also Known As

Sprint Review Meeting

Category

Verification category as this pattern helps the client see the progress of the project as well as they can suggest early changes.

Motivation

The motivation of this pattern is to address the trust, socio-cultural communication and co-ordination, and knowledge transfer challenges. For example consider a team that is divided into sub-teams that are located on different time zones and one of the team is located in the same country as the client. In order to show the progress of the project it is more convenient that the team, which is located near the client, presents the demo in order to have a face-to-face meeting. Once the meeting has ended the onshore team briefs the offshore teams the remarks of the client.

Applicability

Use Onshore Review Meeting pattern when:

- Team is distributed over different time zones.
- The onshore is located in the same country as the client.

Participants

- Distributed onshore and offshore agile team.
- Client

Collaboration

- Onshore team presents demo to the client and discusses the meeting minutes with the offshore.

Consequences

The Onshore Review Meeting pattern has the following benefits and liabilities:

1. It allows the client to meet the development team face-to-face and give feedback. This helps overcome the communication and coordination challenges.

2. It allows both onshore and clients to discuss what changes they want and how much time will be required based on the modification. This helps overcome the knowledge transfer challenges.

3. It limits the offshore team from meeting the clients which can cause the offshore team from discussing the changes and giving their remarks on the clients feedback.

Known uses

Wipro Technologies, a global service provider company used onshore review meetings so that they could get quick feedback from the customer/business user, which was then shared with the remote team over mail and/or teleconference (Sureshchandra et al., 2008). Similarly when SirsiDynix, USA outsourced their work to Starsoft, Ukraine they also used onshore review meetings to show the demo of the work done (Sutherland et al., 2007)

Related Patterns

Onshore Review Meeting pattern is often used with Asynchronous Retrospective Meetings Pattern because after the demo both onshore and offshore team members conduct their separate retrospective meetings.

4.6 Chapter Summary

This chapter presented the distributed agile patterns catalogue. The chapter starts by discussing the templates of existing patterns and then presents how the template of distirbuted agile was designed. We then discuss how the pattern catalogue is organised and can be used by the practitioners. At the end of the chapter we presented the whole distributed agile patterns catalogue.

Chapter 5 Validation and Evaluation of the Distributed Agile Patterns Catalogue

5.1 Introduction

In this section we have validated and evaluated the Distributed Agile Patterns catalogue. In order to validate the catalogue we used reflection workshop technique as proposed by Kerth (Kerth, 2001). For the workshop we invited 4 companies to take part. From each company two representatives took part. The workshop lasted from 9:30 a.m.- 4:00 p.m., in which the catalogue was presented to the participants and then they gave feedback on how to improve the catalogue. Based on their suggestions the pattern catalogue was modified. For evaluation we compared the catalogue with other solutions presented in the literature and discussed their limitations and how our pattern catalogue overcomes those limitation.

5.2 Revised Distributed Agile Patterns

In order to verify and validate the identified distributed agile patterns, we conducted a reflection workshop based on Kerth's **"The keep/try reflection workshop**" (Kerth, 2001). The reason for choosing this method was to get expert opinions on the patterns and find out if the patterns actually exist based on the practitioners point of view and if they help practitioners in applying agile in offshore projects. To conduct this workshop four companies were invited to take part; details are shown in Table 5.1.

Table 5.1. Details of the Companies.

Company	Category	Type of Business	HQ	Loc. of Business	Exp. in Agile
C1	Startup	IT services	Pakistan	Dubai, U.K	2 years

C2	Breakeven	Product-based	Pakistan	U.S.A, U.K	5 years
C3	Breakeven	Multinational IT service provider	Pakistan	Dubai	8 years
C4	Profitable	Software solutions	U.S.A	Pakistan	10 years

From each company, we invited two participants to take part in the workshop so that we can have a managerial and development point of view for our patterns. Table 5.2 shows the details of the participants and their experience in agile and distributed agile practices (DAP):

Table 5.2. Details of the Participants Attended the Workshop.

Company	**Participant Name**	**Job Title**	**Role in Agile**	**Exp. in Agile (years)**	**Exp. in DAP**
C1	P1	CEO	Scrum Master	2	2
	P2	Developer	Developer	2	1
C2	P3	CEO	Product Owner	5	5
	P4	Senior Developer	Developer	5	5
C3	P5	CEO	Scrum Master	8	8
	P6	Developer	Developer	5	4
C4	P7	Senior	Developer	8	5

	Developer			
P8	Senior Software Engineer	Developer	8	4

The workshop was conducted at C1's boardroom and the duration was 7 hours. The agenda of the workshop is shown in Table 5.3 below:

Table 5.3. Agenda of the Reflection Workshop.

No.	Agenda Items	Time
1.	Tea and Networking	9:30 a.m. -10:00 a.m.
2.	Introduction of what is reflection workshop and discussed how we will conduct it	10:00a.m. -11:00 a.m.
3.	Discussion on Distributed Agile Patterns	11:00 a.m. –2:00p.m.
4.	Lunch	2:00 p.m. – 3:00 p.m.
5.	Feedback and recommendation	3:00 p.m.-4:00p.m.

The workshop was started by explaining to the participants what a reflection workshop is and what is expected from them at the end of the workshop. Based on Kerth **keep/try reflection workshop** method, this workshop focused on capturing three things on a flip chart, which was then displayed for the group to see and discuss. (Kerth, 2001). The three things are:

- What should we keep?
- Where are we having on-going problems?
- What do we want to try in the next time period?

The format of the flip chart looks like in the Table 5.4 below:

Table 5.4. Flip Chart Format for the Reflection Workshop (Kerth, 2001).

Keep these	Try these
Problems	

The patterns catalogue was presented in the form of a document so that the participants can have a copy and make notes during the presentation (Figure 5.1). Since the document was new to the participants, the catalogue was presented one pattern at a time. The participants were then invited to discuss among themselves before moving to the next pattern. The document was presented as shown in Figure 5.1 to help make the documentation on the flip chart easier while referring to a pattern:

Figure. 5.1. Document Presented in Reflection Workshop to the Participant

As an example to show how we documented the reflection workshop, Table 5.5 shows the detail of a flip chart of Company 3 Participant 5 (C3P5). In the ***Keep these,*** section we have mentioned all the patterns that C3P5 thinks should stay as they were presented, that is he wants section 1.2-1.5 to remain the same. In the ***Problems*** section, C3P5 wants Section 2.2 to be changed, as according to him the pattern *Central Code Repository* is too generic.

In the section ***Try these,*** C3P5 has given suggestions on how we can improve the patterns, such as he wants us to change the name of pattern *1.1 Scrum of Scrum* to distributed Scrum of Scrum because the Scrum of Scrum word is usually used in agile for a Scrum inside a Scrum whereas in this pattern we meant two separate scrum working independently.

Table 5.5. Flip Chart of Company 3 Participant 5 (C3P5).

<table>
<tr><td>Keep these
Keep 1.2,1.3,1.4,1.5
Keep 2.2,2,3,2,4
Keep 3.1,3.2,3.4
Keep 4</td><td rowspan="2">Try these
1.1: Try changing name of 1.1 to distributed Scrum of Scrum as just scrum of scrum is confusing.
3.3: Try coming up with generic set of preferences and then let the distributed team define exactly how they want to collaborate.</td></tr>
<tr><td>Problems
2.2: Need to change this pattern as it is too generic, that is add details on what code repository should be used.</td></tr>
</table>

A summary of the flip charts by all participants is shown in Table 5.6:

Table 5.6. Summarised Flip Chart of the Companies.

Keep these	**Try these**
• Keep 1.2,1.3,1.4,1.5 • Keep 2.2,2,3,2,4 • Keep 3.1,3.2,3.4 Keep 4	• 1.1: Try changing name of 1.1 to distributed Scrum of Scrum as just scrum of scrum is confusing. • 1.1: Try making 1.2,1.3,1.4,1.5 part of 1.1 as they are related to each other. • 3.3: Try coming up with generic set of preferences and then let the distributed team define exactly how they want to collaborate. • 4.2: Onshore review is good and recommended, however in some demos offshore team should also be there especially if they are visiting the onshore team. This would boost their moral and help them understand the client better. • To all patterns, add details on which challenge they help overcome and modify the consequences section to highlight how a benefit can aid in overcoming a challenge.
Problems	
• 2.2: Need to change this pattern as it is too generic, that is add details on what code repository should be used. • 3.1: Whole team should be part of planning poker, as it will help the team to understand each other's culture. • 3.2: Only applicable teams that are in the time zone next to each other e.g. 5:00p.m is 9:00 a.m. in another country.	

Table 5.7. Feedback on the Challenges Distributed Agile Patterns Solve.

Pattern Name	Trust	Socio-Cultural	Communication and Coordination	Knowledge Transfer	Agreed
Distributed Scrum of Scrum			☒	☒	80%
Local Stand-up Meeting			☒	☒	80%
Follow the sun			☒	☒	50%
Onshore Review Meeting			☒	☒	75%
Collaborative Project Planning	☒	☒	☒	☒	100%
Project Charter	☒		☒	☒	75%
Collaborative Planning Poker	☒	☒	☒	☒	100%
Global Scrum Board	☒	☒	☒	☒	100%
Local Sprint Planning			☒	☒	80%
Local Pair Programming			☒		80%
Central Code Repository			☒	☒	100%
Asynchronous Retrospective Meeting			☒	☒	65%
Asynchronous Information Transfer			☒	☒	80%
Synchronous Communication	☒	☒	☒	☒	100%
Visit Onshore-Offshore Teams	☒	☒	☒	☒	100%

After receiving feedback on the presented catalogue, the discussion was then directed towards the usefulness of the patterns to help overcome the offshore challenges. According to the results collected from the workshop, Table 5.7 shows what pattern helps solve which offshore challenge based on the feedback of the participants. As shown in Table 5.7:

- 50% of the participants agreed that the **follow the sun pattern** helps in improving communication and coordination of the team. However the other half believe that by applying follow the sun pattern, the team has less

overlapping working hours which results in less real-time communication among the team members.

- 75% of the participants agreed that **project charter pattern** would help solve the trust, communication and coordination and knowledge transfer issue. As it is a single document that is shared between the onshore and offshore team, defining the goal and objective of the project. However 25% of the participants believed that it is difficult to establish trust at the start of the project through a document as to develop trust among the team members, they need to communicate with each other frequently throughout the project.

- Similarly 80% of the participants agreed that **distributed scrum of scrum pattern** helps solve communication and coordination as well as knowledge transfer issue in offshore software development. As it allows each team to conduct their own scrum and at the end both teams share their work with each other hence providing synchronous exchange of communication and information.

- All the participants agreed that **visit onshore-offshore team pattern** helps build trust among the onshore and offshore team members. It also helps team members understand each other's socio-cultural differences and as the team is co-located, it will help in improving the team's communication and coordination and knowledge transfer.

5.3 Evaluating Distributed Agile Patterns

In this section we will evaluate the distributed agile patterns (DAP) approach with the other existing approaches present in literature and discuss what advantages our approach has over the existing solutions that were mentioned in Section 2.4.

Table 5.8. Existing Solutions in Comparison to the DAP Catalogue.

No.	Type	Existing Solutions for Global Software Development
1.	**Approach**	**Using Agile Practices to Solve Global Software Development Problems.**
	Offshore Challenges answered in this approach	Beecham proposed to overcome the challenges of poor communication, lack of control, low staff morale and ambiguous requirements (Beecham et al., 2014). They achieved this by identifying agile practices that solve the identified challenges. As mentioned in Table 2.7 that the solution presented did not provide details of how the identified practices will overcome the offshore challenges. Further they have not addressed offshore challenges such as trust, socio-cultural and knowledge transfer.
	Limitation	The limitation to their approach is that even though they have considered the challenges of GSD, they have not discussed in detail how practitioners can solve them. For example for the challenge identified as "vague/missing requirements" the solution they presented was a one line answer, " Product owner role; Planning game; Frequent interactions; on-site customer." They didn't give any detail on how or what affect these practices will have.
2.	**Approach**	**Ontology-Based Multi-Agent System to Support Requirements Traceability in Multi-Site Software Development Environment.**
	Offshore Challenges	This approach, focuses on communication, coordination

	answered in this approach	and knowledge transfer challenges. They have used software engineering ontology as a communication framework to enable knowledge sharing and reuse. The proposed framework supports automated requirements traceability tasks. As mentioned in Table 2.7, Agent Communication Language is used for requirement traceability. This approach only focuses on requirement traceability in a distributed environment and they did not consider offshore challenges such as trust and socio-cultural issues.

	Limitations	A major limitation to their approach is that it is too technical considering in agile the requirements/user stories are written by the product owner. A simple request to update a requirement using the Agent Communication Language (ACL) looks like below:
		
		`((action (agent-identifier`
		`:name UserAgent@platform1`
		`:addresses (sequence`
		`http://192.168.0.4:7778/acc))`
		
		`(UpdateRequirement`
		`:name FR03`
		`:resourceType Requirements)))`
		
		For a technical person it isn't a difficult piece of code, but considering the fact that product owners do not have a technical background. It is difficult to use this system. Secondly they have only focused on the requirements phase thus providing us with an incomplete solution to the challenges affecting the offshore development process

3.	**Approach**	**Experiments for Offshore Project to Address Centrifugal Forces.**
	Offshore Challenges answered in this approach	Larman has suggested experiments to overcome offshore challenges such as communication and coordination (Larman et al., 2010). But they have just provided us with brief description of experiments that they think based on their experience with Valtech should predict success for other organisations offshoring their projects using agile. For example one of their experiment is to: *"Try.. Outside-the-site agile coaches*, in which they state that every organisation benefits by bringing in outside agile coaching experts to act as viral agent. This is doubly true for offshore organisations steeped in traditional command-and-control managements and process culture. If the offshore organisation is multinational start by looking for in-company coaches from other nations." This approach focuses on providing agile experiments in distributed environment and they did not consider offshore challenges such as trust, socio-cultural and knowledge transfer issues.
	Limitations	Limitation of their approach is that they have not provided example on how and why should practitioners perform that experiment. They have not provided samples of organisations doing that experiment and what results did they achieve by doing that experiment and what difference does it make if practitioners do not perform this

		experiment.
4.	**Approach**	**Understanding Collaborative Practices in Distributed Agile Development using Theoretical Concepts**
	Offshore Challenges answered in this approach	Modi research presented a new way of solving the collaboration challenge (Modi et al., 2013). They have presented a model representing the interrelationship with common ground, boundary objects and awareness as according to their proposed solution interlinking these theoretical models within the globally distributed agile setting, can lead to increasing awareness regarding the project artefacts which in turn contributes to establishing and negotiating common ground for joint collaborative practices to be held at different distributed sites. This approach focuses on collaboration challenges and does not consider trust, socio-cultural and knowledge transfer challenges.
	Limitations	Although this research presented a new way of solving the collaboration challenge; they haven't presented any results as their idea was presented as a research proposal. They have just mentioned what research methodology they will use and what their expected results will be based on their assumptions.
5.	**Approach**	**Distributed Agile Patterns for Offshore Software Development**
	Offshore Challenges answered in this	As part of this research we designed a catalogue of 15 distributed agile patterns. The focus of these patterns was

	approach	to overcome offshore challenges such as trust, socio-cultural, communication and coordination, and knowledge transfer. In chapter 4 we have presented the catalogue.
	Strengths	Compared to the above-mentioned approaches, our pattern catalogue covers the main challenges of offshoring; we have categorised the catalogue based on what type of problem it solves such as management, communication, collaboration and verification patterns. The pattern catalogue also provides details of how practitioners can use the patterns and in each pattern there is a section on known uses in which examples are given on which companies have used the pattern.

5.4 Chapter Summary

This chapter presented the validation of the distributed agile patterns catalogue by using Kerth's (2001) keep/try reflection workshop method. For the workshop we invited four companies to take part and we presented our patterns catalogue to them. Based on their feedback we modified our patterns catalogue. In order to evaluate the results of this research we compared our catalogue with other existing solution present in literature such as ontology-based multi-agent system to support requirements and experiments for offshore project to address centrifugal forces. For the different approaches we identified what challenges they helped solve and then discussed their limitations and further explained how our patter catalogue addresses the offshore challenges.

In next chapter a case study has been presented which shows how the distributed agile patterns can be used in the requirement phase.

Chapter 6 Case Study to Show Applicability of Distributed Agile Patterns

6.1 Introduction

In this section we present a case study on how the Distributed Agile Patterns catalogue can be used for offshore software development to give an overview to the practitioners on how to select patterns from the catalogue. However we only focus on the requirement-engineering phase of the software development lifecycle. The chapter starts by presenting an overview of the requirements engineering process and how requirements are gathered in agile software development. We then present the existing work done on improving the requirements engineering process and lastly we show how distributed agile patterns can be used to overcome the requirements engineering challenges in offshore development that we highlighted in Chapter 2 and then we map the selected distributed agile patterns onto the requirements engineering lifecycle to show how the patterns improve each step of the lifecycle in a distributed project.

6.2 Overview of Requirements Engineering Process

Wiegers defined requirements as something, which a system must have or satisfy or perform which is being identified by the client side (Wiegers, 2007). Similarly Zave stated that requirements engineering is the branch of software engineering that is concerned with the real-world goals for, functions of and constraints on software system (Zave, 1997). The process of requirements elicitation is one of the most challenging and critical tasks in software development. According to Jacobs the cost of incorrect, misunderstood and not agreed upon requirements affects all of us in terms of time, money and lost opportunities (Jacobs, 2007). Fowler argued that everything else in software development depends on the requirements, as without having a stable set of requirements to start with, the development team cannot start coding

(Fowler, 2005). Darke, Davis and Anthony have also done similar work by explaining how critical the requirement process is and how it directly impacts the success and failure of a project (Darke, 1997; Davis, 1989; Anthony, 1992).

6.3 Requirements Engineering Process in Traditional Software Development vs. Agile Methodology

In traditional software development methodologies, the client would predefine all their requirements to the development team before the start of subsequent phases. The team would then analyse the requirements and finalise a software requirements specification (SRS) document. Once the client has approved the document, they would start the development phase. Figure 6.1 shows the traditional requirements engineering process. This process of requirements elicitation has problems such as; a long time is spent in preparing this documentation, which causes issues in dealing with future requirements change requests from the client once the actual development phase starts. Over decades of software development we have learnt that requirements change is inevitable during the development stage, because neither the client nor the developers are 100% sure of all the requirements of the system at the start of the project.

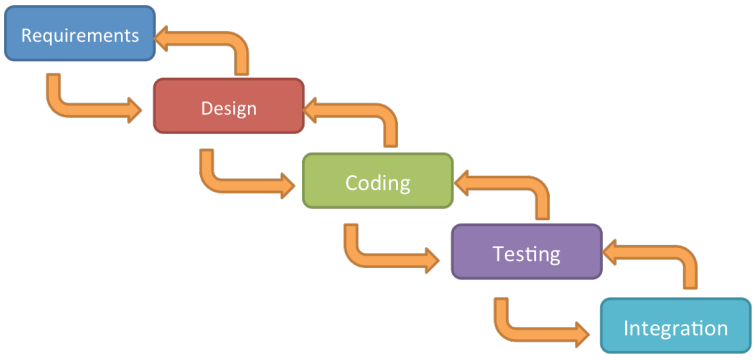


Figure 6.1. Traditional Requirements Engineering Process.

In agile software development, this problem is solved with the use of story cards, which is a lighter process for the definition of very high-level requirements and is an artefact of methods such as SCRUM and XP. They contain just enough information for

the developer to be able to estimate how much effort and time will be required to develop them and can handle change requests with little effort. There is also an agile requirements change management process to control changing requirements throughout the software development lifecycle. Based on this process, new requirements can be added and reprioritized based on the client's request. Figure 6.2 illustrates the agile requirements change management process:

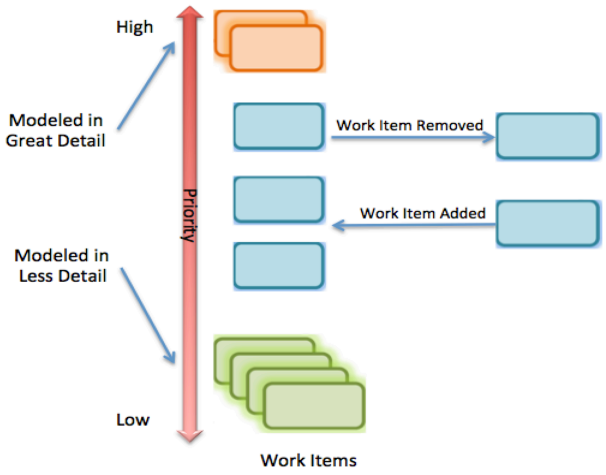


Figure 6.2. Agile Requirements Change Management Process.

6.4 Approaches to Improve Requirements Engineering Process in Agile

Tools have been made to ease the process of documenting story cards. One such tool is SoBA (Patel et al., 2008). The tool addressed issues that are remaining to be solving regarding story card based development, which are:

i. To explore story card for test driven development and design for testability.
ii. To structure story cards with user stories and acceptance testing.
iii. To address non-functional requirements and effort estimation for story cards.
iv. To express traditional software development best practices as guidelines with a tool support.
v. To capture story cards and plan tasks supporting pair-wise development.

This model can capture user stories in five formats, which are face-to-face communication, electronic discussions, knowledge and experience from similar systems, graphical notations and voice. This tool consists of five stages, which they have based on the traditional model for collection requirement, which are: capturing user stories and refining them, develop story cards, structuring story cards, effort estimation and priorities story cards.

Figure 6.3 shows the methodology used in SoBA tool to document story cards and what steps are performed before a user story can go into development phase. That is the customer and the tem capture the user stories, which can be collected in five different multi-media format such as face-to-face, electronic discussion, knowledge systems, graphical notation and voice. Once the user stories are captured they are refined and documented into story cards with indexing. The next steps are structuring those cards and assigning effort estimation and putting the cards in prioritization order. Once the cards are ready they are moved to design and development phase of the project.

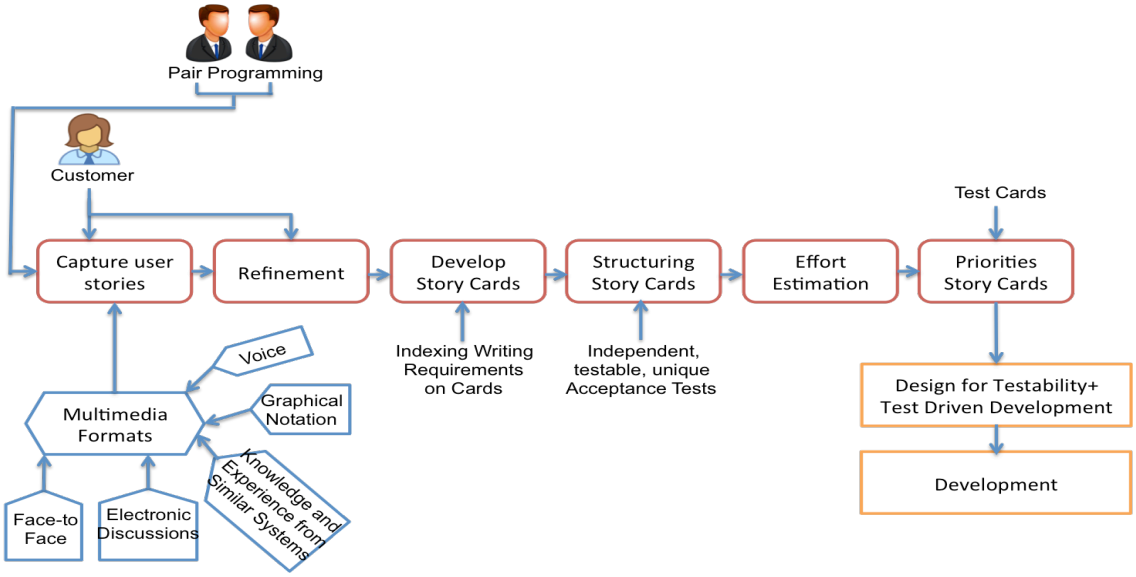


Figure 6.3. Story Card Based Methodology followed by SoBA Tool (Patel et al., 2008).

Khan conducted a systematic literature review to identify factors that generate risks during the requirements engineering process in a global software development

environment. Based on their research, they found 74 factors, which they grouped into 8 classifications, which are (Khan et al., 2014):

i. Communication and Distance.
ii. Cultural, Background, Language, Organisational and Time Differences.
iii. Knowledge Management and Awareness.
iv. Management.
v. Tools, Technologies and Standards.
vi. Stakeholders.
vii. Project and Process
viii. Requirements.

These classifications can facilitate the practitioners by serving as a checklist to consider while capturing requirements however since this study has only collected data from literature and has not validated their findings from practitioners, we cannot consider this as a standard classification nor we don't know how much effect it has on the requirements process if the practitioner uses it.

Work has also been done on how agile requirements can be prioritised in a large-scale outsourced system. Daneva conducted an empirical study that aimed prioritise agile requirements but keeping the following points in mind (Daneva et al., 2013):

- Focus on not only creating value,
- But also accumulating it over time, while minimizing the risk for the development team.

According to their study the task of prioritising requirements was the job of product owner or the client and the contextual factors affecting the requirements prioritisation process are as follows:

- The need to embrace change,
- Project constraints
- Project Scope

- The number of project staff.

As every software project is subjected to the need of change in requirements and every project has constraints, Daneva, focused on the affects of project scope and number of staff working on the project as according to their study, the scope determines the complexity of the project and the number of staff is directly linked to risk, as larger number of clients often leads to contradiction in requirements. They also classified requirements into six types of dependencies, which are (Daneva et al., 2013):

i. *Inter-domain dependencies* are referred to dependencies that are concerned with requirements that are overlapping multiple business areas such as client policies and setting up client complaint management system.

ii. *Intra-domain dependencies* are those dependencies caused due to the close links between the processes and entities.

iii. *Dependencies due to downstream activities,* such as sequencing requirements that maximises the use of available human resource.

iv. *Team-based dependencies* deal with avoiding multiple teams having to do the same work or on dependent artefacts.

v. *Dependences among user stories* these are imposed by the order of activities in a specific business process.

vi. *Dependencies among delivery stories,* these are referred to the dependencies caused by non-functional requirements.

As part of their study they designed a user story delivery process, which is shown in Figure 6.4.

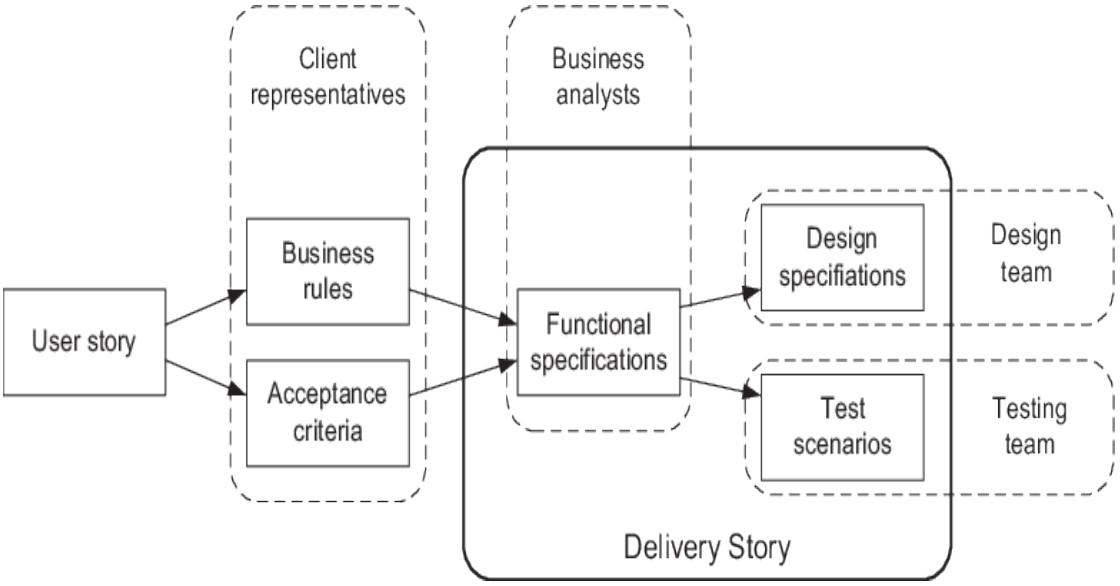


Figure 6.4. The User Story Delivery Process (Daneva et al., 2013).

They also identified 10 characteristics practices that should be followed while gathering and prioritising user stories. An overview of the practices is given below:

i. Maintain traceability between the user stories and delivery stories. In their research they introduced the concept of delivery stories. A delivery story consists of the following:

- Number of use cases
- Required time for use cases implementation in person days
- Number of GUI screens
- Required time for GUI screens implementation in person days
- Business rules and validations
- Number of test scenarios
- Required time for test cases preparation in person days
- Required time for data models in person days
- Total time required for sub-process implementation

ii. Establish a dedicated Delivery Story Team. According to their study in larger projects, we need to have a separate team to handle the requirements process. As unlike small projects which do not have any detailed requirements, in large projects there is a need to add detail in requirements to make it easier for the offshore team members and clients to understand what they have to do.

iii. Run series of workshops, as it helps build trust between the client and development sites.

iv. Establish a Domain Owner. As a result of their study, they have introduced a new role of Domain Owner, and they classified it as a critical role for acquiring knowledge about the core business processes and operational procedures.

v. Understand the dependencies of requirements before carrying out the design process.

vi. The Product Owner role should be set-up at the client's site.

vii. Set-up a requirement change analysis management process, as the requirements do change over the software development life cycle.

viii. Directly ask the Product Owner to confirm any change in the requirements.

ix. In order to encourage transparency, use status reports, which should be shared with the client after every three weeks.

x. Training on the domain should be gradual and the team should be prepared for it. Most of the time trainings were supported by the client's organisation.

However, SoBA helps in documenting and managing user stories it does not consider the complexity added by distributed agile developments such as any change in the requirements needs to be communicated over different locations and due to cultural

and language differences, requirements can be misunderstood (Patel et al., 2008). Similarly the work done by Daneva does facilitate the practitioners by providing us with a checklist of 10 practices to consider while documenting and prioritising user stories but they did not consider the affect of cultural differences and how language affects the requirements process and they introduced two new concepts such as dedicated team for requirements and Domain owner, which add additional cost to the development, while as mentioned in Section 2.2.1, the main reason for organisations to choose to offshore their projects, is to cut down on cost (Daneva et al., 2013).

6.5 Distributed Agile Patterns used to Overcome Requirements Engineering Challenges

In this section we will discuss how distributed agile patterns can be used to overcome the challenges we identified in Section 2.4. However we will focus only on the requirements engineering process. The selection of patterns was done following the process identified in Figure 4.1.

Table 6.1. Using Distributed Agile Patterns to Address Requirements Engineering Challenges in Agile Offshore Development.

No.	Requirements Challenge in Agile Offshore Development	Distributed Agile Pattern	Solution
1.	**Requirement Estimation**	Collective Planning Poker	By doing the planning poker activity together all team members will get a better understanding of each others skills.
2.	**Objective and core**	Project Charter	Having a project charter document

	functional requirements		written at the start of the project, will give all the team members a clear understanding of the project's objective and core functional requirements.
3.	**Misunderstanding of requirements**	Collective Project Planning	Since the whole team will be part of the planning activity, the chances of misunderstanding a requirement will be reduced.
		Asynchronous Information Transfer	In case of any misunderstanding, the team members can communicate with each other using asynchronous tools.
		Synchronous Communication	For a real-time response to any misunderstanding, the team members can use synchronous methods for communication.
4.	**Vague requirements**	Visit Onshore-Offshore	To clarify vague requirements, onshore team members should visit offshore team members and vice versa to discuss the requirements in order to avoid defects.
5.	**Changes in the requirements**	Global Scrum Board	Any change in the requirements, will be updated on the global

			scrum board, which is accessible by all team members in real-time.
6.	**Inconsistencies**	Central Code Repository	To avoid any inconsistency between the work done and the user stories, all the team members use a central code repository, which is accessible by the whole team. All the code is committed in that repository, enabling the whole team to see what work is done and what is remaining.
7.	**Bidirectional traceability of requirements**	Central Code Repository	With the help of a central code repository, we can map each requirement with its code, allowing traceability of all the requirements being developed at different locations.
		Global Scrum Board	Status of all requirements being developed can be recorded using a global scrum board, which helps in maintaining traceability of requirements.
8.	**Managing requirements**	Local Sprint Planning	Each site can manage their requirements and daily meetings using local sprint planning.

6.6 Mapping Distributed Agile Patterns on the Requirements Engineering Lifecycle

In this section we map the distributed agile patterns onto the traditional requirements engineering lifecycle, to show how these patterns facilitate the requirements engineering process. As shown in Figure 6.5 the four key features of this process are as following:

- **Feasibility Study:** In this process, we decide whether or not the proposed system is worthwhile. By writing down the Project Charter document in the beginning of the project, we can achieve this. As stated in the Project Charter we define the aim, objectives and core functional requirements of the proposed system.

- **Requirements Elicitation and Analysis:** The purpose of this process is to identify the application domain; the services it will provide and what are the limitations. Three distributed agile patterns are applied in these phases. Collaborative project planning helps the team to find out what the requirements of the application to be developed. Local sprint planning helps team members in their respective locations to discuss in detail the user stories allocated to them and determine the constrains associated with them. Collaborative Planning Poker, allows the team members to discuss the services that the system will provide and estimate how much effort is required to develop them.

- **Requirements Specification**: Once the requirements have been identified, we document them in the SRS document. As in agile software development, requirements are documented using user stories, in distributed agile software development; we use a Global Scrum Board, where all the story cards are placed. Hence any change or modification in the requirements will be updated on the Global Scrum Board, which is accessible by all the team in real-time.

- **Requirements Validation:** In this process, the requirements are validated, by reviewing them to make sure that the identified requirements are in accordance with what the client wants. Four distributed agile patterns make sure that the correct requirements are identified. Central Code Repository helps the client verify that the code being developed is according to requirements. Asynchronous Information Transfer provides a platform to the team members and the clients to communicate and coordinate the progress of the system being developed and in case of any confusion, the team can either use asynchronous or synchronous tools for communication, according to the communication standards defined using Synchronous Communication Pattern. For further clarification of requirements, distributed agile patterns suggest that both onshore and offshore team members should visit each other.

In Figure 6.5 we have presented how our distributed agile patterns map onto the four phases of traditional requirement engineering in order to provide an overview to practitioners on which distributed agile pattern they can use while collecting requirements for their offshore projects. For example in order to perfome the requirements elicitation and analysis phase in the requirement engineering process of an offshore project, the practitioner can use collaborative project planning, local sprint planning and collaborative planning poker patterns

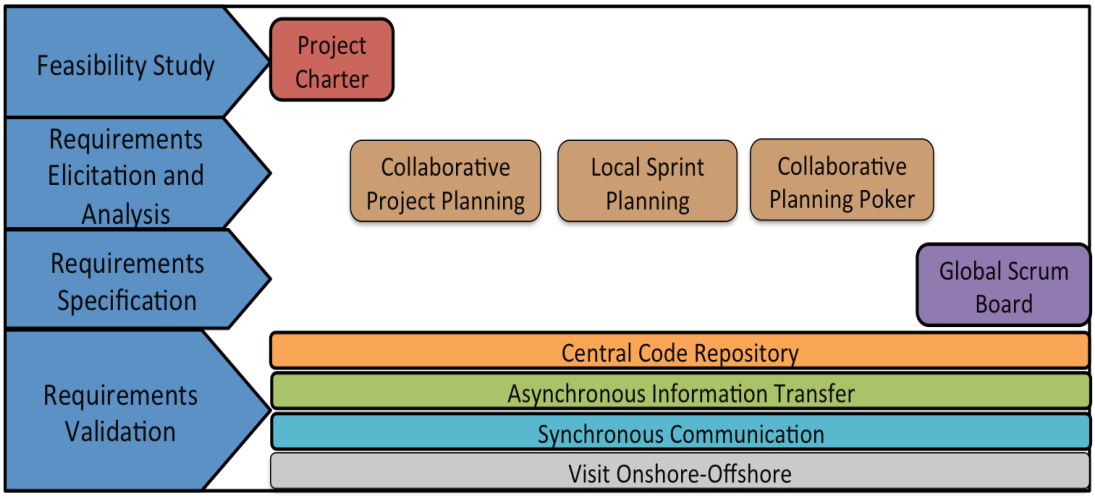


Figure 6.5. Mapping Distributed Agile Patterns on Traditional Requirements Engineering Process.

6.7 Chapter Summary

This chapter presented a case study to show how the distributed agile patterns can be used in offshore software development. The focus of this case study was on the requirement phase of the software development lifecycle so the chapter starts by giving an overview on the requirement engineering process. Then we provided a comparison between the requirement process in traditional software development vs. agile methodology. Several approaches to improve the requirement process in agile were also discussed in this chapter. Based on the offshore challenges identified in chapter 2 we discussed how the distributed agile patterns catalogue can be used to overcome them. Lastly we mapped the distributed agile patterns on the requirement engineering lifecycle.

The next chapter concludes the thesis, discussing the main achievements of the research and proposing a set of recommendations for improvements in future research.

Chapter 7 Conclusions and Future Work

7.1 Introduction

This research has studied the factors that effect software development in offshore environment. It aimed to identify and address the factors that affect the adoption of agile practices in offshore software development. In order to achieve the research aim and objectives, to answer the research questions and to maximize the quality of the case study finding there was a need to choose the most appropriate research approach and strategy for the researcher to follow in collecting and analysing the data. The selection of an appropriate methodology for this research came by following a literature review based on the research topic, setting the aim and objectives and after consideration of literature on research methodologies.

Based on the nature of this research, the phenomenological research paradigm and the use of both qualitative and quantitative philosophy were reasoned to be as discussed in Section 3.3, the perfect means to undertake the research due to its subjectivity and in order to gain in-depth understanding and to identify the factors that affect the adoption of agile practices in offshore software development. As adopting agile practices in offshore software development is not a straightforward process. In this research we have conducted a systematic literature review and semi-structure interview to identify agile practices that are being repeatedly used in offshore development to solve recurring problems. As a result, we identified 15 distributed agile patterns, which were presented in Section 4.5. To validate the patterns catalogue we used a reflection workshop and for evaluation we compared our solution to other existing solutions present in literature to overcome offshore challenges, which we have presented in Chapter 5.

In order to help practitioners understand how the distributed agile patterns catalogue can be used at any development phase we presented a case study in Chapter 6, on

how the catalogue can be used in the requirement phase of scrum based software development.

7.2 Meeting the Aim and Objectives and Answering the Research Questions

The main research question was answered by achieving the aim of this study, which was to address the issues between the onshore and offshore teams that are developing software at offshore locations and to develop a solution by identifying repeating solutions from literature and interviewing professionals. This aim has been achieved by addressing the research objectives as follows:

The first objective was to review the relevant literature on offshore software development. The literature included studies focusing on why organisations chose to moves their projects to offshore locations, what are different types of global software development business models, what are the advantages and limitations of choosing to develop software at different locations, how did organisation overcome offshore software development challenges.

The second objective was to identify key challenges that occur while offshore software development and if any recurring agile practices are being used to overcome an offshore challenges. We achieved this by conducted a SLR and semi-structured interviews as we wanted to identify patterns.

Based on the data collected and analysis, we identified agile practices recurrently being used to solve offshore software development challenges. Based on observed recurring practices we designed our distributed agile patterns catalogue, which is the third objective of our research.

The fourth objective was to validate and evaluate the distributed agile patterns catalogue, which we achieved by conducting reflection workshop and based on the feedback we revised our patterns catalogue and for evaluation we compared our

patterns catalogue with other solutions such as using ontology-based multi-agent approach, experiments for offshore projects designed by Larman et at. (2010), the detail is presented in Section 5.3. The last objective was to write this thesis and present it in the viva.

7.3 Recommendation for Future Work

Future work has been planned for this research, which includes ways on how we can improve and extend the research presented in this study. The following recommendations arise from the present study for future academics and professional research that want to work in offshore software development and agile methodology:

- Researchers could investigate and identify the factors that effect the development of software in domestic outsourcing, shared services and internal offshoring as in this research we have only focused on identifying the challenges in offshore outsourcing business model.

- One major direction for future work would be to apply this catalogue for the development of an offshore software project that want to adopt agile practices and study the impact of each pattern on different development stages.

- Design a decision making tool to help practitioners enter the challenge they are facing or what process they want to apply and get a list of recommended distributed patterns that could be used to overcome it. The purpose of the tool would be to provide ease to practitioners, as going through all fifteen patterns can be time consuming.

- Further research is needed into identifying how development of the code can be improved and identifies that is design a set of rules to be followed on how frequent the developers need to commit their code and identify patterns from

how practitioners are developing their code in order to provide a guideline on how to manage and write quality code in offshore software development.

- Researchers can compare other development lifecycles such as ad-hoc, linear, evolutionary, iterative and incremental approaches as defined in Appendix B, to identify patterns for offshore software development and evaluate them against the distributed agile patterns catalogue to determine if they produce better results or not.

- Finally researchers can explore designing solutions other then patterns and evaluate their solutions with distributed agile patterns catalogue and determine if their solutions present better results then our catalogue.

7.4 Limitation of the Study

According to Yin (2003) every research study is limited by the constraints placed upon the research and the research environment, and this research is no exception. In this research we have made every effort to overcome these limitation to ensure that this study could be delivered smoothly, but it is not possible to control all the factors that were likely to affect the outcomes.

The limitations of this research as are following:

- Some cases identified in literature, just mentioned names of practices and did not provide details of how they are to be implemented or what the results were after they implemented an adapted agile practice.

- Another limitation was that researchers had identified offshore software development challenges but had not considered the complexity add by adopting agile practices, which limited our data from literature and we had to

reply on interviews to verify agile practices actually used in offshore development.

- A final limitation was that as we also collected data from interviews, the human factor can not be ignored, that is since the organisation chose to remain anonymous, some facts were not allowed to be made public such as what was the nature of the projects being offshored, how critical the data being shared to offshore locations was and what legal arrangements they had with the offshore developers regarding sharing of information for the purpose of research.

7.5 Chapter Summary

This chapter has presented an overall conclusion for the thesis. The aim of the research, the objectives, and the techniques that were used by the researcher to achieve the study objectives were addressed; and in final conclusion, the chapter outlined potential future directions, which could be adopted as further research work.

References:

Abrahamsson, Pekka, Juhani Warsta, Mikko T. Siponen, and Jussi Ronkainen (2003). "New directions on agile methods: a comparative analysis." In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pp. 244-254. IEEE.

Agerfalk, Par J., and Brian Fitzgerald (2008). "Outsourcing to an unknown workforce: Exploring opensourcing as a global sourcing strategy." MIS quarterly 32, no. 2, 385.

Alexander, Christopher (1977). *A pattern language: towns, buildings, construction*. Oxford University Press, 1977.

Alnuem, Mohammed Abdullah, Arshad Ahmad, and Hashim Khan (2012). "Requirements Understanding: A Challenge in Global Software Development, Industrial Surveys in Kingdom of Saudi Arabia." In *2012 IEEE 36th Annual Computer Software and Applications Conference*, pp. 297-306. IEEE, 2012.

Alzoubi, Yehia Ibrahim, Asif Qumer Gill, and Ahmed Al-Ani (2016). "Empirical studies of geographically distributed agile development communication challenges: A systematic review." *Information & Management* 53, no. 1 (2016): 22-37

Al-Zaidi, Areej, and Rizwan Qureshi (2017). "Global software development geographical distance communication challenges." *Int. Arab J. Inf. Technol.* 14, no. 2 (2017): 215-222.

Anthony Byrd. T, K.L.C., Robert W. Zmud, (1992). A Synthesis of Research on Requirements Analysis and Knowledge Acquisition Techniques. MIS Quarterly, 1992. **Vol. 16**(No. 1): p. pp. 117-138

Armour, Phillip G (2007). "Agile… and offshore." *Communications of the ACM* 50, no. 1 (2007): 13-16.

Aron, Ravi, and Jitendra V. Singh (2005). "Getting offshoring right." Harvard business review 83, no. 12, 135.

Aronsson, Cecilia. (2007) ”Kriget om talangerna”, *Veckan Affärer,* nr 8, p 44.

Avritzer, Alberto, Francois Bronsard, and Gilberto Matos (2010). "Improving Global Development Using Agile." In *Agility Across Time and Space*, pp. 133-148. Springer Berlin Heidelberg, 2010.

Avritzer, Alberto, William Hasling, and Daniel Paulish (2007). "Process investigations for the global studio project version 3.0." In *Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference on*, pp. 247-251. IEEE, 2007.

Babu, Mohan (2005). Offshoring IT services: a framework for managing outsourced projects. Tata McGraw-Hill Education.

Beck, Kent, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning et al. (2001). "Manifesto for agile software development".

Beecham, Sarah, John Noll, and Ita Richardson (2014). "Using Agile Practices to Solve Global Software Development Problems--A Case Study." In *2014 IEEE International Conference on Global Software Engineeering Workshops*, pp. 5-10. IEEE, 2014.

Berczuk, Steve. "Back to basics: The role of agile principles in success with an distributed scrum team." In *Agile Conference (AGILE), 2007*, pp. 382-388. IEEE, 2007.

Berenbach, B. (2006). Impact of organizational structure on distributed requirements engineering processes: lessons learned. In *Proceedings of the 2006 international workshop on Global software development for the practitioner* (GSD '06). ACM, New York, NY, USA, 15-19.

Beulen, Erik, Paul Van Fenema, and Wendy Currie (2005). "From Application Outsourcing to Infrastructure Management:: Extending the Offshore Outsourcing Service Portfolio." *European Management Journal* 23, no. 2, 133-144.

Bhat, J.M., Gupta, M., Murthy, S.N. (2006). Overcoming Requirements Engineering Challenges: Lessons from Offshore Outsourcing. *IEEE Softw. 23, 5 (September 2006), 38-44.*

Bricout, V., Heliot, D., Cretoiu, A., Yang, Y., Simien, T., and Hvatum, L., (2005). Patterns for managing distributed product development teams. Tech. rep., Schlumberger Oilfield Services.

Bird, C., Nagappan, N., Devanbu, P., Gall, H., and Murphy, B. (2009). Does distributed development affect software quality? An empirical case study of Windows Vista. In *Proceedings of the 31st International Conference on Software Engineering* (ICSE '09). IEEE Computer Society, Washington, DC, USA, 518-528.

Braithwaite, Keith, and Tim Joyce (2005). "XP expanded: Distributed extreme programming." In *Extreme programming and agile processes in software engineering*, pp. 180-188. Springer Berlin Heidelberg.

Bryman, Alan (2012). Social research methods (5th ed.). Oxford: Oxford University Press.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., Sommerlad, P., & Stal, M. (1996). Pattern-oriented software architecture, volume 1: A system of patterns.

Bosch, J., and Bosch-Sijtsema, P. (2010). From integration to composition: On the impact of software product lines, global development and ecosystems. *J. Syst. Softw. 83, 1 (January 2010), 67-76*

Boon, S., and Holmes, J. (1991). The dynamics of interpersonal trust: Resolving uncertainty in the face of risk. *In R. Hinde and J. Groebel (Eds.). Cooperation and Prosocial Behavior. Cambridge University Press, Cambridge, UK*. 190–211.

Bose, Indranil (2008). "Lessons learned from distributed agile software projects: A case-based analysis." *Communications of the Association for Information Systems* 23, no. 1 (2008): 34.

Brown, W.J., Malveau, R.C., McCormick, H.W.S., Mowbray, T.J. (1988): AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. J. Wiley, 1998.

Čavrak, I., Orlić, M., and Crnković, I. (2012). Collaboration patterns in distributed software development projects. In *Proceedings of the 34th International Conference on Software Engineering* (ICSE '12). IEEE Press, Piscataway, NJ, USA, 1235-1244

Camacho, C., Marczak, S., and Conte, T. (2013). On the Identification of Best Practices for Improving the Efficiency of Testing Activities in Distributed Software Projects: Preliminary Findings from an Empirical Study. In *Proceedings of the 2013 IEEE 8th International Conference on Global Software Engineering Workshops* (ICGSEW '13). IEEE Computer Society, Washington, DC, USA, 1-4.

Carmel, Erran (1999). Global software teams: collaborating across borders and time zones. Prentice Hall PTR.

Carmel, Erran, and Ritu Agarwal (2001). "Tactical approaches for alleviating distance in global software development." *Software, IEEE* 18.2, 22-29.

Carmel, Erran, and Paul Tjia (2005.). Offshoring information technology: sourcing and outsourcing to a global workforc. Cambridge University Press.

Cataldo, M., Bass, M., Herbsleb, J.D., Bass, L. (2007). On Coordination Mechanisms in Global Software Development. In *Proceedings of the International Conference on Global Software Engineering* (ICGSE '07). IEEE Computer Society, Washington, DC, USA, 71-80.

Clark, Herbert H., and Susan E. Brennan (1991). "Grounding in communication." *Perspectives on socially shared cognition* 13, no. 1991 (1991): 127-149.

Clerc, V. (2008). Towards architectural knowledge management practices for global software development. In *Proceedings of the 3rd international workshop on Sharing and reusing architectural knowledge* (SHARK '08). ACM, New York, NY, USA, 23-28.

Conchúir, Eoin Ó., Pär J. Ågerfalk, Helena H. Olsson, and Brian Fitzgerald (2009). "Global software development: where are the benefits?." *Communications of the ACM* 52, no. 8 .127-131.

Conchúir, E., Holmström, H., Ågerfalk, P.J. and Fitzgerald, B. (2006). Exploring the Assumed Benefits of Global Software Development. In *Proceedings of the IEEE international conference on Global Software Engineering* (ICGSE '06). IEEE Computer Society, Washington, DC, USA, 159-168.

Cordeiro, Lucas, Cassiano Becker, and Raimundo Barreto (2007). "Applying Scrum and Organizational Patterns to Multi-site Software Development." (2007): 46-67.

Cottmeyer, Mike (2008). "The good and bad of Agile offshore development." In *Agile, 2008. AGILE'08. Conference*, pp. 362-367. IEEE, 2008.

Cristal, Mauricio, Daniel Wildt, and Rafael Prikladnicki (2008). "Usage of Scrum practices within a global company." In *Global Software Engineering, 2008. ICGSE 2008. IEEE International Conference on*, pp. 222-226. IEEE, 2008.

Damian, Daniela, and Deependra Moitra (2006). "Guest Editors' Introduction: Global Software Development: How Far Have We Come?." Software, IEEE 23, no. 5, 17-19.

Damian, Daniela E., and Didar Zowghi (2003). "RE challenges in multi-site software development organisations." *Requirements engineering* 8, no. 3 (2003): 149-160.

Danait, Ajay (2005). "Agile offshore techniques-a case study." In *Agile Conference, 2005.*

Proceedings, pp. 214-217. IEEE, 2005.

Daneva, Maya, Egbert Van Der Veen, Chintan Amrit, Smita Ghaisas, Klaas Sikkel, Ramesh Kumar, Nirav Ajmeri, Uday Ramteerthkar, and Roel Wieringa. (2013). "Agile requirements prioritization in large-scale outsourced system projects: An empirical study." *Journal of systems and software* 86, no. 5 (2013): 1333-1353.

Darke, Peta (1997). G.G.S., User viewpoint modelling: understanding and representing user viewpoints during requirements definition. Information Systems Journal, 1997. **Volume 7**(Issue 3): p. pages 213-219.

Das, Tarun K., and Bing-Sheng Teng (1998). "Between trust and control: developing confidence in partner cooperation in alliances." *Academy of Management review* 23, no. 3, 491-512.

Davenport, Thomas, H., and Prusak, Laurence, (1998). *Working knowledge.* Boston: Harvard Business School Press.

Davis, A.M (1989). Software Requirements: Analysis and Specification. 1989: Prentice Hall. 352 pages.

Denny, Nathan, Shivram Mani, Ravi Sheshu Nadella, Manish Swaminathan, and Jamie Samdal (2008). "Hybrid offshoring: Composite personae and evolving collaboration technologies." *Information Resources Management Journal (IRMJ)* 21, no. 1 (2008): 89-104.

Dingsøyr, Torgeir, Sridhar Nerur, VenuGopal Balijepally, and Nils Brede Moe (2012). "A decade of agile methodologies: Towards explaining agile software development." *Journal of Systems and Software* 85, no. 6, 1213-1221.

Dourish, Paul, and Victoria Bellotti. (1992). Awareness and coordination in shared workspaces. *Proceedings of the 1992 ACM conference on Computer-supported cooperative work. ACM*.

Drummond, B., and John Francis Unson (2008). "Yahoo! Distributed Agile: Notes from the world over." In *Agile, 2008. AGILE'08. Conference*, pp. 315-321. IEEE, 2008.

Dybå, Tore, and Torgeir Dingsøyr (2008). "Empirical studies of agile software development: A systematic review." *Information and software technology* 50, no. 9, 833-859.

Easterby-Smith, M., Thorpe R. and Lowe A. (2004) Management Research: An Introduction,2nd Edition. SAGE Publications Ltd. London.

Easterby-Smith, Mark, Richard Thorpe, and Paul R. Jackson (2012). *Management research*. Sage, 2012.

Ebert, Christof (2011). *Global software and IT: A guide to distributed development, projects, and outsourcing*. Wiley-IEEE Computer Society Press

Elssamadisy, Amr, and David West (2006). "Adopting agile practices: an incipient pattern language." In *Proceedings of the 2006 conference on Pattern languages of programs*, p. 1. ACM, 2006.

Flick, Uwe. (2011). Introducing research methodology: A beginner's guide to doing a research project. London: Sage.

Fowler, Martin (1997). *Analysis patterns: reusable object models*. Addison-Wesley Professional, 1997.

Fowler, Martin. *Daily Standup Patterns*. Available at: http://martinfowler.com/articles/itsNotJustStandingUp.html

Fowler, Martin (2005). The New Methodology. 2005. <http://www.martinfowler.com>.

Galen, Robert (2009) SCRUM Product Ownership – Balancing Value from the Inside Out RGCG LLC 2009. ISBN: 978-0-578-01912-3

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1997.

Ghafoor, Fawad, Ibrar Ali Shah, and Nasir Rashid (2017). "Issues in Adopting Agile Methodologies in Global and Local Software Development: A Systematic Literature Review Protocol with Preliminary Results." *International Journal of Computer Applications* 160, no. 7.

Garner, C. Alan (2004). Offshoring in the service sector: Economic impact and policy issues. Economic Review-Federal Reserve Bank of Kansas City 89, 5-38.

Global Wages Comparison from http://www.paywizard.co.uk/main/pay/global-wage-comparison

Goddard, W. & Melville, S. (2004). Research Methodology: An Introduction, (2nd ed.) Oxford: Blackwell Publishing.

Grechanik ,M., Jones, J.A., Orso, A., and van der Hoek, A. (2010). Bridging gaps between developers and testers in globally-distributed software development. In *Proceedings of the FSE/SDP workshop on Future of software engineering research* (FoSER '10). ACM, New York, NY, USA, 149-154.

Gupta, Amar (2007). "Expanding the 24-hour workplace." *The Wall Street Journal*: 15-16.

Gupta, Amar (2009). "Deriving mutual benefits from offshore outsourcing."*Communications of the ACM* 52, no. 6 : 122-126.

Gutwin, Carl, Saul Greenberg, and Mark Roseman (1996). "Workspace awareness in real-time distributed groupware: Framework, widgets, and evaluation." *People and Computers.* 281-298.

Hayes ,L.G (2003). “Everything you know about Offshore Outsourcing is Wrong”, Datamation Magazine, Feb 28, 2003.

Hendrik C. Jahn, Alexandra Gazendam and André Schlieker (2011) Accenture: The high-performance insurer of the future, Accenture. [Available at: http://nstore.accenture.com/Geneva/HP_Insurer_of_the_Future_report.pdf]

Herbsleb, James D., Audris Mockus, Thomas A. Finholt, and Rebecca E. Grinter (2001). "An empirical study of global software development: distance and speed." In *Proceedings of the 23rd international conference on software engineering*, pp. 81-90. IEEE Computer Society, 2001.

Herbsleb, J.D. (2007). Global Software Engineering: The Future of Socio-technical Coordination. In *Future of Software Engineering* (2007). IEEE Computer Society, Washington, DC, USA, 188-198. DOI=http://dx.doi.org/10.1109/FOSE.2007.11.

Herbsleb, J.D. Paulish, D.J. and Bass, M. (2005). Global software development at siemens: experience from nine projects. In *Proceedings of the 27th international conference on Software engineering* (ICSE '05). ACM, New York, NY, USA, 524-533.

Hitt, Michael A., R. Duane Ireland, and Robert E. Hoskisson (2002). *Strategic Management: Competitiveness and Globalization*. South-Western College Pub., Canada.

Hofner, Gerd, and V. S. Mani (2007). "TAPER: A generic framework for establishing an offshore development center." In Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference , 162-172.

Hofstede, Geert (1980). Culture's consequences. Beverly Hills, CA: Sage, 1980.

Hofstede, Geert (1997) *Culture and organizations--Software of the mind*: McGraw-Hill, 1997.

Holmström, Helena, Brian Fitzgerald, Pär J. Ågerfalk, and Eoin Ó. Conchúir. "Agile practices reduce distance in global software development." *Information Systems Management* 23, no. 3 (2006): 7-18.

Holmstrom, H., Conchúir, E. Ó., Agerfalk, J., & Fitzgerald, B. (2006). Global software development challenges: A case study on temporal, geographical and socio-cultural distance. In Global Software Engineering, 2006. ICGSE'06. International Conference), 3-11.

Hossain, Emam, Muhammad Ali Babar, and Hye-young Paik (2009). "Using scrum in global software development: a systematic literature review." In *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*, pp. 175-184. Ieee, 2009.

Humphrey, Watts S (1989). *Managing the Software Process* . Addison-Wesley Professional.

Hussey, Jill, and Roger Husse (1997)y. "Business research." *Hampshire: Palgrave* (1997).

Hvatum, Lise B., and Rebecca Wirfs-Brock (2015). "Patterns to build the magic backlog." In *Proceedings of the 20th European Conference on Pattern Languages of Programs*, p. 12. ACM, 2015.

Jaakkola, H., Heimbürger, A., and Linna, P. (2010). Knowledge-oriented software engineering process in a multi-cultural context. *Software Quality Control 18, 2 (June, 2010). 299-319.*

Jacobs, D., Requirements Engineering so Things Don't Get Ugly, in Companion to the proceedings of the 29th International Conference on Software Engineering. 2007, IEEE Computer Society. p. 159-160.

Jahns, Christopher, Evi Hartmann, and Lydia Bals (2006). "Offshoring: Dimensions and diffusion of a new business concept." Journal of Purchasing and Supply Management 12.4 , 218-231.

Jan, Syed Roohullah, Faheem Dad, Nouman Amin, Abdul Hameed, and Syed Saad Ali Shah (2016). "Issues in Global Software Development (Communication, Coordination and Trust) A Critical Review." *training* 6, no. 7 (2016): 8.

Jankowicz, A. Devi (2005). *Business research projects*. Cengage Learning EMEA, 2005.

Javdani Gandomani, Taghi, Hazura Zulzalil, Abdul Azim Abdul Ghani, Abu Bakar Md Sultan, and Khaironi Yatim Shairf (2014). "Exploring facilitators of transition and adoption to agile methods: a grounded theory study." *Journal of Software* 9, no. 7 (2014): 1666-1678.

Javidan, Mansour, Günter K. Stahl, Felix Brodbeck, and Celeste PM Wilderom (2005). "Cross-border transfer of knowledge: Cultural lessons from Project GLOBE."*The Academy of Management Executive* 19, no. 2, 59-76.

Jensen, B. and Zilmer, A. (2003). Cross-continent development using Scrum and XP. *Proceedings of XP*. Springer Berlin, , pp.146-153

Kanawattanachai, Prasert, and Youngjin Yoo (2002). "Dynamic nature of trust in virtual teams." *The Journal of Strategic Information Systems* 11, no. 3,187-213.

Kamaruddin, Nina Kamarina, Noor Habibah Arshad, and Azlinah Mohamed (2012). "Chaos issues on communication in Agile Global Software Development." In *Business Engineering and Industrial Applications Colloquium (BEIAC), 2012 IEEE*, pp. 394-398. IEEE, 2012.

Kausar, Maryam and Adil Al-Yasiri (2015). *"*Distributed Agile Patterns for Offshore Software Development" 12th International Joint Conference on Computer Science and Software Engineering (JCSSE), IEEE 2015

Kausar, Maryam and Adil Al-Yasiri (2016). Distributed Agile Patterns. [Available at: http://stp872.edu.csesalford.com/distributedagilepatterns.html]

Kedia, Ben L. and Lahiri, Somnath, (2007), *International outsourcing of services: A partnership model*. Memphis: Journal of International Management, 13, 22–37

Kerth, Norm (2001). "Project Retrospectives: A Handbook for Reviews." *Dorset House Publishing* (2001).

Khan, Huma Hayat, Mohd Naz'ri bin Mahrin, and Suriayati bt Chuprat. "Factors generating risks during requirement engineering process in global software development environment." *International Journal of Digital Information and Wireless Communications (IJDIWC)* 4, no. 1 (2014): 63-78.

Kircher, M., Jain, P., Corsaro, A. and Levine, D. (2001). Distributed Extreme Programming. Proceedings of the International Conference on eXtreme Programming and Flexible Processes in Software Engineering, Sardinia, Italy, May 20 - 23,2001.

Kitchenham, B. & Charters, S. (2007), Guidelines for performing Systematic Literature Reviews in Software Engineering, EBSE2007-001, Keele University and Durham University Joint Report.

Kluckhohn, Florence R., and Fred L. Strodtbeck (1961). "Variations in value orientations." (1961).

Kobayashi-Hillary, M. (2005). *Outsourcing to India: The offshore advantage*. Springer Science & Business Media.

Koehne, B., Shih, P.C., and Olson, J.S. (2012). Remote and alone: coping with being the remote member on the team. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work* (CSCW '12). ACM, New York, NY, USA, 1257-1266.

Kommeren, R., and Parviainen, P. (2007). Philips experiences in global distributed software development. *Empir. Softw. Eng.* 12, 6 (December, 2007), 647-660. DOI=http://dx.doi.org/10.1007/s10664-007-9047-3.

Kontio, Jyrki, Magnus Hoglund, Jan Ryden, and Pekka Abrahamsson (2004). "Managing commitments and risks: challenges in distributed agile development." In *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*, pp. 732-733. IEEE

Korkala, Mikko, Minna Pikkarainen, and Kieran Conboy (2010). "Combining agile and traditional: Customer communication in distributed environment." In *Agility Across Time and Space*, pp. 201-216. Springer Berlin Heidelberg, 2010.

Kotlarsky, Julia, and Ilan Oshri (2005). "Social ties, knowledge sharing and successful collaboration in globally distributed system development projects." *European Journal of Information Systems* 14, no. 1, 37-48.

Krippendorff, Klaus. (2004). *Content analysis: An introduction to its methodology*, Thousand Oaks, CA: Sage Publications.

Kroll, Josiane, Alan R. Santos, Rafael Prikladnicki, Estevão Ricardo Hess, Rafael A. Glanzner, Afonso Sales, Jorge Luis Nicolas Audy, and Paulo Henrique Lemelle Fernandes (2012). "Follow-the-Sun Software Development: A Controlled Experiment to Evaluate the Benefits of Adaptive and Prescriptive Approaches." In *SEKE*, pp. 551-556. 2012.

Kussmaul, Clifton, Roger Jack, and Barry Sponsler (2004). "Outsourcing and offshoring with agility: A case study." In *Extreme Programming and Agile Methods-XP/Agile Universe 2004*, pp. 147-154. Springer Berlin Heidelberg, 2004.

Lander, Maria Cristina, Russell L. Purvis, Gordon E. McCray, and William Leigh (2004). Trust-building mechanisms utilized in outsourced IS development projects: a case study. *Information & Management* 41, no. 4 .509-528.

Lanubile, Filippo, Fabio Calefato, and Christof Ebert (2013). Group Awareness in Global Software Engineering. *IEEE Software* . 18-23.

Larman, Craig (2004). *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley. p. 27. ISBN 978-0-13-111155-4

Larman, Craig (2010). *Practices for Scaling Lean and Agile Development: Large*. Addison-Wesley Professional, 2010.

Layman,L.,Williams,L.,Damian,D.and Bures,H. (2006.) Essential communication practices for extreme programming in a glob- al software development team. *Information and Software Technology,* vol. 48, no. 9, pp. 781-794

Lee, Sang M., and Suzanne J. Peterson (2001) . "Culture, entrepreneurial orientation, and global competitiveness." *Journal of world business* 35, no. 4 (2001): 401-416.

Lescher, Christian (2010). "Patterns for global development: how to build one global team?." *Proceedings of the 15th European Conference on Pattern Languages of Programs*. ACM, 2010.

Liskin, O., Herrmann, C., Knauss, E., Kurpick, T., Rumpe, B., and Schneider, K. (2012). Supporting Acceptance Testing in Distributed Software Projects with Integrated Feedback Systems: Experiences and Requirements. In *Proceedings of the 2012 IEEE Seventh International Conference on Global*

Software Engineering (ICGSE '12). IEEE Computer Society, Washington, DC, USA, 84-93.

MacGregor, Eve, Yvonne Hsieh, and Philippe Kruchten (2005). "Cultural patterns in software process mishaps: incidents in global projects." In *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1-5. ACM, 2005.

Malik, Fareesa, and Hammad Majeed (2010). "Effect of Development Strategies and Project Types on Offshore Software Development using Agile Paradigm–A Study." In *2010 Agile Conference*, pp. 67-74. IEEE, 2010.

Massol, Vincent (2004), Technical talk on "Agile Offshore Methods" TheServerSide.com, Jan 6, 2004

Maruping, Likoebe M (2010). "Implementing Extreme Programming in Distributed Software Project Teams: Strategies and Challenges." In *Agility Across Time and Space*, pp. 11-30. Springer Berlin Heidelberg, 2010.

Massol, Vincent (2004). ""Case Study: Distributed Agile Development." *TheServerSide. com* (2004).

Matloff, Norman (2005). "Offshoring: What can go wrong?." *IT professional* 7, no. 4, 39-45.

May, Tim (2011). *Social research*. McGraw-Hill Education (UK), 2011.

Modi, Sunila, Pamela Abbott, and Steve Counsell (2013). "Negotiating common ground in distributed agile development: A case study perspective." In *Global Software Engineering (ICGSE), 2013 IEEE 8th International Conference on*, pp. 80-89. IEEE, 2013.

Morgan, Robert M., and Shelby D. Hunt (1994). "The commitment-trust theory of relationship marketing." *the journal of marketing* , 20-38.

Mullick, N., Bass, M., Houda, Z., Paulish, P., and Cataldo, M. (2006). Siemens Global Studio Project: Experiences Adopting an Integrated GSD Infrastructure. In *Proceedings of the IEEE international conference on Global Software Engineering* (ICGSE '06). IEEE Computer Society, Washington, DC, USA, 203-212.

Newman, Isadore. (1998). Qualitative-quantitative research methodology: Exploring the interactive continuum. Carbondale: Southern Illinois University Press.

Niazi, M., El-Attar, M., Usma, M., and Ikram, N. (2012). GlobReq: A framework for improving requirements engineering in global software development projects: Preliminary results. In *proceedings of the 16th International Conference on Evaluation & Assessment in Software Engineering (*EASE 2012). (May 14-15, 2012) 166-170.

Nisar, Muhammad F., and Tahir Hameed (2004). "Agile methods handling offshore software development issues." In *Multitopic Conference, 2004. Proceedings of INMIC 2004. 8th International*, pp. 417-422. IEEE.

Noll, J., Richardson, I., & Beecham, S. (2014). Patternizing GSD Research: Maintainable Decision Support for Global Software Development.

OECD (2004). OECD information technology outlook: Organisation for economic co-operation and development. Retrieved January 2009, from http:// www.oecd.org/dataoecd/22/18/37620123.pdf.

Oshri, Ilan, Julia Kotlarsky, and Leslie P. Willcocks. *The Handbook of Global Outsourcing and Offshoring 3rd Edition*. Springer, 2015.

Ovaska, P., Rossi, M. and Marttiin, P. (2003). Architecture as a coordination tool in multi-site software development. Softw. Process: Improve. Pract. 8(2003) 233–247.

Ozawa, H., & Zhang, L. (2013). Adapting Agile Methodology to Overcome Social Differences in Project Members. In *Agile Conference (AGILE), 2013* (pp. 82-87). IEEE.

Paasivaara, Maria, and Casper Lassenius (2003). "Collaboration practices in global inter-organizational software development projects." *Software Process: Improvement and Practice* 8, no. 4 (2003): 183-199.

Paasivaara, Maria, Sandra Durasiewicz, and Casper Lassenius (2009). "Using scrum in distributed agile development: A multiple case study." In *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*, pp. 195-204. IEEE, 2009.

Pakdeetrakulwong, Udsanee, Pornpit Wongthongtham, and Naveed Khan (2015). "An Ontology-Based Multi-Agent System to Support Requirements Traceability in Multi-Site Software Development Environment." In *Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference*, pp. 96-100. ACM, 2015.

Patel, Chetankumar, and Muthu Ramachandran (2008). "SoBA: A Tool Support for Story Card Based Agile Software Development." In *SETP*, pp. 17-23. 2008.

Pehmöller, Anneke, Frank Salger, and Stefan Wagner(2010). "Patterns for testing in global software development." In *Proceedings of the 13th International Conference on Quality Engineering in Software Technology*. 2010.

Picot, Arnold, Ralf Reichwald, and Rolf T. Wigand. (2003). "Die grenzenlose Unternehmung.".

Pilatti, Leonardo, and Jorge Luis Nicolas Audy (2006). "Global Software Development Offshore Insourcing Organizations Characteristics: Lessons Learned from a Case Study." In Global Software Engineering, 2006. ICGSE'06. International Conference, 249-250.

Prikladnicki, Rafael, Jorge Luis Nicolas Audy, Daniela Damian, and Toacy Cavalcante de Oliveira (2007). "Distributed Software Development: Practices and challenges in different business strategies of offshoring and onshoring." In Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference, 262-274.

Prikladnicki, Rafael, and Jorge Luis Nicolas Audy (2012). "Managing Global Software Engineering: A Comparative Analysis of Offshore Outsourcing and the Internal Offshoring of Software Development." Information Systems Management 29, no. 3: 216-232.

Prikladnicki, Rafael, Sabrina Marczak, Erran Carmel, and Christof Ebert (2012). "Technologies to Support Collaboration across Time Zones." *Software, IEEE*29, no. 3, 10-13.

Poole, Charles J (2004). "Distributed product development using extreme programming." In *Extreme Programming and Agile Processes in Software Engineering*, pp. 60-67. Springer Berlin Heidelberg, 2004.

Qumer, Asif, Brian Henderson-sellers, and Tom Mcbride (2007). "Agile adoption and improvement model." (2007).

Qumer, Asif, and Brian Henderson-Sellers (2008). "An evaluation of the degree of agility in six agile methods and its applicability for method engineering." *Information and software technology* 50, no. 4 (2008): 280-295.

Qumer, Asif, and Brian Henderson-Sellers (2008). "A framework to support the evaluation, adoption and improvement of agile methods in practice." *Journal of Systems and Software* 81, no. 11 (2008):

1899-1919.

Radlo, Mariusz-Jan (2016). "Offshoring and Outsourcing in Economic Theories." In *Offshoring, Outsourcing and Production Fragmentation*, pp. 41-97. Palgrave Macmillan, London, 2016.

Radoff, Sandy (2006). "*Improved Cross-Cultural Communication Increases Global Sourcing Productivity"*. United States: Accenture

Ramesh, Balasubramaniam, Lan Cao, Kannan Mohan, and Peng Xu (2006). "Can distributed software development be agile?." *Communications of the ACM* 49, no. 10 (2006): 41-46.

Räty, Petteri, Benjamin Behm, Kim-Karol Dikert, Maria Paasivaara, Casper Lassenius, and Daniela Damian (2013). "Communication Practices in a Distributed Scrum Project." *CoRR* (2013).

Ring, P., and Van de Ven, A (2004). Developmental processes of cooperative interorganizational relationships. Acad. Mgt. Rev. 19, 1 .90–118.

Robertson, S. (1996). Requirements Patterns Via Events/Use Cases. In *Proceedings Pattern Languages of Programming*.

Robinson, Marcia, and Ravi Kalakota (2004). Offshore outsourcing: Business models, ROI and best practices. Mivar Press.

Robson, C (2002). Real world research: A resource for social scientist and practitioners (2nd Edition, Blackwell Publishers, Oxford.

Rousseau, Denise M., Sim B. Sitkin, Ronald S. Burt, and Colin Camerer (1998). "Not so different after all: A cross-discipline view of trust." *Academy of management review* 23, no. 3, 393-404.

Sabherwal, Rajiv (1999). "The role of trust in outsourced IS development projects. *Communications of the ACM* 42.2 .80-86.

Sadagopan, S (2002), Indian IT Companies – Do you know Wipro?, Times of India, May 13, 2002

Sahay, Sundeep, Brian Nicholson, and Shenai Krishna (2003). Global IT outsourcing: software development across borders. Cambridge University Press.

Salger, Frank, Jochen Englert, and Gregor Engels (2010). "Towards Specification Patterns for Global Software Development Projects-Experiences from the Industry." In *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the*, pp. 73-78. IEEE, 2010.

Saunders, M., P Lewis (2007) - Research methods for business students, 2007

Sarker, Suprateek, and Sundeep Sahay (2004). Implications of space and time for distributed work: an interpretive study of US–Norwegian systems development teams. *European Journal of Information Systems* 13.1 .3-20.

Sengupta, B., Chandra, S., and Sinha, V. (2006). A research agenda for distributed software development. In *Proceedings of the 28th international conference on Software engineering* (2006). ACM, New York, NY, USA, 731-740. DOI=http://doi.acm.org/10.1145/1134285.1134402.

Shah, Hina, Nancy J. Nersessian, Mary Jean Harrold, and Wendy Newstetter (2012). "Studying the influence of culture in global software engineering: thinking in terms of cultural models." In *Proceedings of the 4th international conference on Intercultural Collaboration*, pp. 77-86. ACM, 2012.

Simons, Matt (2002). "Internationally agile." *Inform IT*.

Smite, Darja, and Claes Wohlin (2011). "A whisper of evidence in global software engineering." Software, IEEE 28.4, 15-18.

Šmite, Darja, Nils Brede Moe, and Pär J. Ågerfalk (2010). "Fundamentals of Agile Distributed Software Development." In *Agility Across Time and Space*, pp. 3-7. Springer Berlin Heidelberg.

Smits, H. and Pshigoda, G (2007). Implementing scrum in a distributed software development organization. Proceedings of AGILE 2007, pp. 371-375.

Summers, Mark (2008). "Insights into an Agile adventure with offshore partners." In *Agile, 2008. AGILE'08. Conference*, pp. 333-338. IEEE, 2008.

Sureshchandra, Kalpana, and Jagadish Shrinivasavadhani (2008). "Adopting agile in distributed development." In *Global Software Engineering, 2008. ICGSE 2008. IEEE International Conference on*, pp. 217-221. IEEE, 2008.

Sutherland, Jeff, Anton Viktorov, Jack Blount, and Nikolai Puntikov (2007). "Distributed scrum: Agile project management with outsourced development teams." In *System Sciences, 2007. HICSS 2007.*

40th Annual Hawaii International Conference on, pp. 274a-274a. IEEE, 2007.

Stack, Martin, and Ricard Downing (2005). "Another look at offshoring: Which jobs are at risk and why?." *Business Horizons* 48, no. 6 , 513-523.

Star, Susan Leigh, and James R. Griesemer (1989). "Institutional ecology,translations' and boundary objects: Amateurs and professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39." *Social studies of science* 19, no. 3 (1989): 387-420.

Taylor, Philip S., Des Greer, Paul Sage, Gerry Coleman, Kevin McDaid, and Frank Keenan (2006). "Do agile GSD experience reports help the practitioner?." In *Proceedings of the 2006 international workshop on Global software development for the practitioner*, pp. 87-93. ACM.

Tervonen, I., Haapalahti, A., Harjumaa, L., Simila, J. (2013). Outsourcing Software Testing: A Case Study in the Oulu Area. In *Proceedings of the 2013 13th International Conference on Quality Software* (QSIC '13). IEEE Computer Society, Washington, DC, USA, 65-74.

Tervonen, I., Mustonen, T. (2009). Offshoring Test Automation: Observations and Lessons Learned. In *Proceedings of the 2009 Fourth IEEE International Conference on Global Software Engineering* (ICGSE '09). IEEE Computer Society, Washington, DC, USA, 226-235.

Therrien, Elaine (2008). "Overcoming the Challenges of Building a Distributed Agile Organization." In *AGILE*, pp. 368-372. 2008.

Trompenaars, Fons, and Charles Hampden-Turner (2004). *Managing people across cultures*. Chichester: Capstone, 2004.

Tylor, Edward Burnett (1871). *Primitive culture: researches into the development of mythology, philosophy, religion, art, and custom*. Vol. 2. J. Murray, 1871.

Välimäki, Antti, and Jukka Kääriäinen (2008). "Patterns for Distributed Scrum—A Case Study." *Enterprise interoperability III*. Springer London, 2008. 85-97.

Välimäki, Antti, Jukka Kääriäinen, and Kai Koskimies (2009). "Global Software Development Patterns for Project Management." In *European Conference on Software Process Improvement*, pp. 137-148. Springer Berlin Heidelberg, 2009.

van Heesch, Uwe (2015). "Collaboration patterns for offshore software development." *Proceedings of the 20th European Conference on Pattern Languages of Programs*. ACM, 2015

Van Zoest, Anna (2004). Offshoring practices in the UK – Where are the Limits? . Institute for Public Policy Research. .

Vashistha, Atul, and Avinash Vashistha (2005). *The offshore nation: the rise of services globalization*. New York: Tata McGraw-Hill Publishing Company Limited.

Vax, Michael, Stephen Michaud (2008). "Distributed Agile: Growing a Practice Together,"AGILE Conference, pp. 310-314, Agile 2008.

von Campenhausen, C. (2005). "Offshoring rules–Auslagern von unterstützenden Funktionen." *Zeitschrift für Betriebswirtschaft* 75, no. 1, 5-13.

Wiegers, K. (2007) . Process Impact. 2007. <http://www.processimpact.com>.

Wildt, Daniel, and Rafael Prikladnicki (2010). "Transitioning from Distributed and Traditional to Distributed and Agile: An Experience Report." In *Agility Across Time and Space*, pp. 31-46. Springer Berlin Heidelberg, 2010.

Wipro (2012). Available at: http://www.wipro.com/newsroom/Wipro-Project-for-BT-Recognized-as-the-Offshoring-Project-of-the-Year-by-UKs-National-Outsourcing-Association

Yap, Monica (2005). "Follow the sun: distributed extreme programming development." In *Agile Conference, 2005. Proceedings*, pp. 218-224. IEEE, 2005.

Yeo, Alvin W (2001). "Global-software development lifecycle: An exploratory study." In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 104-111. ACM

Yu, Liguo, Zhong Guan, and Srini Ramaswamy (2016). "The effect of time zone difference on asynchronous communications in global software development." *International Journal of Computer Applications in Technology* 53, no. 3 (2016): 213-225.

Yin, R. K. (2003) Case Study Research Design and Methods, Thousand Oaks, 3rd Edition, Sage Publications, Inc.

Yu, Xiaodan, and Stacie Petter (2014). "Understanding agile software development practices using

shared mental models theory." *Information and Software Technology* 56, no. 8 (2014): 911-921.

Zahedi, Mansooreh, Mojtaba Shahin, and Muhammad Ali Babar (2016). "A systematic review of knowledge sharing challenges and practices in global software development." *International Journal of Information Management* 36, no. 6 (2016): 995-1019.

Zave, Pamela (1997). "Classification of research efforts in requirements engineering." *ACM Computing Surveys (CSUR)* 29, no. 4 (1997): 315-321.

Appendix A: Selected Studies for Offshore Challenges

Selected Studies for identifying the challenges in offshore software development:

[E1]. Avritzer, Alberto, and Adailton Lima. "An empirical approach for the assessment of scheduling risk in a large globally distributed industrial software project." In *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*, pp. 341-346. IEEE, 2009.

[E2]. Bannerman, Paul L., Emam Hossain, and Ross Jeffery (2012) . "Scrum practice mitigation of global software development coordination challenges: a distinctive advantage?." In *System Science (HICSS), 2012 45th Hawaii International Conference on*, pp. 5309-5318. IEEE.

[E3]. Battin, Robert D., Ron Crocker, Joe Kreidler, and K. Subramanian (2001). "Leveraging resources in global software development." *Software, IEEE* 18, no. 2 (2001): 70-77.

[E4]. Boden, Alexander, Bernhard Nett, and Volker Wulf (2007). "Coordination practices in distributed software development of small enterprises." In *Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference on*, pp. 235-246. IEEE.

[E5]. Boon, S., and Holmes, J. (1991). The dynamics of interpersonal trust: Resolving uncertainty in the face of risk. *In R. Hinde and J. Groebel (Eds.). Cooperation and Prosocial Behavior. Cambridge University Press, Cambridge, UK*. 190–211.

[E6]. Bosch, Jan, and Petra Bosch-Sijtsema (2010). "Coordination between global agile teams: From process to architecture." In *Agility Across Time and Space*, pp. 217-233. Springer Berlin Heidelberg.

[E7]. Carmel, Erran (1999). Global software teams: collaborating across borders and time zones. Prentice Hall PTR.

[E8]. Carmel, Erran, and Ritu Agarwal (2001). "Tactical approaches for alleviating distance in global software development." *Software, IEEE* 18, no. 2: 22-29.

[E9]. Cataldo, Marcelo, Patrick A. Wagstrom, James D. Herbsleb, and Kathleen M. Carley (2006). "Identification of coordination requirements: implications for the Design of collaboration and awareness tools." In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pp. 353-362. ACM.

[E10]. Cataldo, Marcelo, Charles Shelton, Yongjoon Choi, Yun-Yin Huang, Vytesh Ramesh, Darpan Saini, and Liang-Yun Wang (2009). "Camel: A tool for collaborative distributed software design." In *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*, pp. 83-92. IEEE.

[E11]. Cusick, James, and Alpana Prasad (2006). "A practical management and engineering approach to offshore collaboration." *Software, IEEE* 23, no. 5: 20-29.

[E12]. Damian, Daniela E., and Didar Zowghi (2003). "An insight into the interplay between culture, conflict and distance in globally distributed requirements negotiations." In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, pp. 10-pp. IEEE.

[E13]. Das, Tarun K., and Bing-Sheng Teng (1998). "Between trust and control: developing confidence in partner cooperation in alliances." *Academy of Management review* 23, no. 3, 491-512.

[E14]. Davenport, Thomas, H., and Prusak, Laurence, (1998). *Working knowledge.* Boston: Harvard Business School Press.

[E15]. Desouza, Kevin C., Yukika Awazu, and Peter Baloh (2006). "Managing knowledge in global software development efforts: Issues and practices." *IEEE software* 23, no. 5: 30

[E16]. Dorairaj, Siva, James Noble, and Petra Malik (2011). "Effective communication in distributed Agile software development teams." In *Agile Processes in Software Engineering and Extreme Programming*, pp. 102-116. Springer Berlin Heidelberg.

[E17]. Dourish, Paul, and Victoria Bellotti. (1992). Awareness and coordination in shared workspaces. *Proceedings of the 1992 ACM conference on Computer-supported cooperative work. ACM*.

[E18]. Ebert, Christof (2011). *Global software and IT: A guide to distributed development, projects, and outsourcing*. Wiley-IEEE Computer Society Press

[E19]. Evaristo, J. Roberto, Richard Scudder, Kevin C. Desouza, and Osam Sato (2004). "A dimensional analysis of geographically distributed project teams: a case study." *Journal of Engineering and technology Management* 21, no. 3: 175-189.

[E20]. Gotel, Olly, Vidya Kulkarni, Des Phal, Moniphal Say, Christelle Scharff, and Thanwadee Sunetnanta (2009). "Evolving an Infrastructure for Engineering, Communication, Project Management and Socialization to Facilitate Student Global Software Development Projects." In *Proc. of the Indian Software Engineering Conference (ISEC 2009), Pune, India*.

[E21]. Green, R., T. Mazzuchi, and S. Sarkani (2010). "Communication and quality in distributed agile development: an empirical case study." *Proceeding in World Academy of Science, Engineering and Technology* 61: 322-328.

[E22]. Grundy, John, John Hosking, and Rick Mugridge (1998). "Coordinating distributed software development projects with integrated process modelling and enactment environments." In *wetice*, p. 39. IEEE.

[E23]. Gutwin, Carl, Saul Greenberg, and Mark Roseman (1996). "Workspace awareness in real-time distributed groupware: Framework, widgets, and evaluation." *People and Computers.* 281-298.

[E24]. Herbsleb, James D., Daniel J. Paulish, and Matthew Bass (2005). "Global software development at siemens: experience from nine projects." In Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference , 524-533.

[E25]. Herbsleb, James D (2007). "Global software engineering: The future of socio-technical coordination." In *2007 Future of Software Engineering*, pp. 188-198. IEEE Computer Society.

[E26]. Hofner, Gerd, and V. S. Mani (2007). "TAPER: A generic framework for establishing an offshore development center." In Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference , 162-172.

[E27]. Hofstede, Geert, Michael Harris Bond, and Chung-leung Luk (1993). "Individual perceptions of organizational cultures: A methodological treatise on levels of analysis." *Organization Studies* 14, no. 4: 483-503.

[E28]. Holmström, Helena, Brian Fitzgerald, Pär J. Ågerfalk, and Eoin Ó. Conchúir. (2006): "Agile practices reduce distance in global software development." *Information Systems Management* 23, no. 3 7-18.

[E29]. Hsieh, Yvonne (2006). "Culture and shared understanding in distributed requirements engineering." In *Global Software Engineering, 2006. ICGSE'06. International Conference on*, pp. 101-108. IEEE.

[E30]. Humphrey, Watts S (1989). *Managing the Software Process* . Addison-Wesley Professional.

[E31]. Iacovou, Charalambos L., and Robbie Nakatsu (2008). "A risk profile of offshore-outsourced development projects." *Communications of the ACM* 51, no. 6: 89-94.

[E32]. Javidan, Mansour, Günter K. Stahl, Felix Brodbeck, and Celeste PM Wilderom (2005). "Cross-border transfer of knowledge: Cultural lessons from Project GLOBE."*The Academy of Management Executive* 19, no. 2, 59-76.

[E33]. Kanawattanachai, Prasert, and Youngjin Yoo (2002). "Dynamic nature of trust in virtual teams." *The Journal of Strategic Information Systems* 11, no. 3,187-213.

[E34]. Kedia, Ben L. and Lahiri, Somnath, (2007), *International outsourcing of services: A partnership model*. Memphis: Journal of International Management, 13, 22–37

[E35]. Kobayashi-Hillary, Mark. (2005). Outsourcing to India: the offshore advantage. ISBN 3-540-23943-X Springer-Verlag Berlin Heidelberg New York.

[E36]. Korkala, Mikko, and Pekka Abrahamsson (2007). "Communication in distributed agile development: A case study." In *Software Engineering and Advanced Applications, 2007. 33rd EUROMICRO Conference on*, pp. 203-210. IEEE.

[E37]. Korkala, Mikko, and Frank Maurer (2014). "Waste identification as the means for improving communication in globally distributed agile software development." *Journal of Systems and Software* 95: 122-140.

[E38]. Lander, Maria Cristina, Russell L. Purvis, Gordon E. McCray, and William Leigh (2004). Trust-building mechanisms utilized in outsourced IS development projects: a case study. *Information & Management* 41, no. 4 .509-528.

[E39]. Lanubile, Filippo, Daniela Damian, and Heather L. Oppenheimer (2003). "Global software development: technical, organizational, and social challenges." *ACM SIGSOFT Software Engineering Notes* 28, no. 6: 2-2.

[E40]. Lanubile, Filippo, Fabio Calefato, and Christof Ebert (2013). Group Awareness in Global Software Engineering. *IEEE Software* . 18-23.

[E41]. Layman, Lucas, Laurie Williams, Daniela Damian, and Hynek Bures (2006). "Essential communication practices for Extreme Programming in a global software development team." *Information and software technology* 48, no. 9: 781-794.

[E42]. Lee, Seiyoung, and Hwan-Seung Yong (2010). "Distributed agile: project management in a global environment." *Empirical Software Engineering* 15, no. 2: 204-217.

[E43]. Maruping, Likoebe M (2010). "Implementing Extreme Programming in Distributed Software Project Teams: Strategies and Challenges." In *Agility Across Time and Space*, pp. 11-30. Springer Berlin Heidelberg.

[E44]. Matloff, Norman (2005). "Offshoring: What can go wrong?." *IT professional* 7, no. 4, 39-45.

[E45]. Morgan, Robert M., and Shelby D. Hunt (1994). "The commitment-trust theory of relationship marketing." *the journal of marketing* , 20-38.

[E46]. Paasivaara, Maria, Sandra Durasiewicz, and Casper Lassenius (2009). "Using scrum in distributed agile development: A multiple case study." In *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*, pp. 195-204. IEEE.

[E47]. Paasivaara, Maria, Casper Lassenius, Daniela Damian, Petteri Raty, and Adrian Schroter (2013). "Teaching students global software engineering skills using distributed scrum." In *Software Engineering (ICSE), 2013 35th International Conference on*, pp. 1128-1137. IEEE.

[E48]. Pilatti, Leonardo, and Jorge Luis Nicolas Audy (2006). "Global Software Development Offshore Insourcing Organizations Characteristics: Lessons Learned from a Case Study." In Global Software Engineering, 2006. ICGSE'06. International Conference, 249-250.

[E49]. Prikladnicki, Rafael, and Jorge Luis Nicolas Audy (2012). "Managing Global Software Engineering: A Comparative Analysis of Offshore Outsourcing and the Internal Offshoring of Software Development." Information Systems Management 29, no. 3: 216-232.

[E50]. Qumer Gill, Asif, and Deborah Bunker (2013). "Towards the development of a cloud-based communication technologies assessment tool: An analysis of practitioners' perspectives." *VINE* 43, no. 1: 57-77.

[E51]. Radoff, Sandy (2006). "*Improved Cross-Cultural Communication Increases Global Sourcing Productivity"*. United States: Accenture

[E52]. Ralyté, Jolita, Xavier Lamielle, Nicolas Arni-Bloch, and Michel Léonard (2008). "A framework for supporting management in distributed information systems development." In *Research Challenges in Information Science, 2008. RCIS 2008. Second International Conference on*, pp. 381-392. IEEE.

[E53]. Ramasubbu, Narayan, Mayuram S. Krishnan, and Prasad Kompalli (2005). "Leveraging global resources: A process maturity framework for managing distributed development." *Software, IEEE* 22, no. 3: 80-86.

[E54]. Ring, P., and Van de Ven, A (2004). Developmental processes of cooperative interorganizational relationships. Acad. Mgt. Rev. 19, 1 .90–118.

[E55]. Robarts, Jane M (2008). "Practical considerations for distributed agile projects." In *Agile, 2008. AGILE'08. Conference*, pp. 327-332. IEEE.

[E56]. Rousseau, Denise M., Sim B. Sitkin, Ronald S. Burt, and Colin Camerer (1998). "Not so different after all: A cross-discipline view of trust." *Academy of management review* 23, no. 3, 393-404.

[E57]. Sabherwal, Rajiv (1999). "The role of trust in outsourced IS development projects. *Communications of the ACM* 42.2 .80-86.

[E58]. Sahay, Sundeep, Brian Nicholson, and Shenai Krishna (2003). Global IT outsourcing: software development across borders. Cambridge University Press.

[E59]. Sarker, Suprateek, and Sundeep Sahay (2004). Implications of space and time for distributed work: an interpretive study of US–Norwegian systems development teams. *European Journal of Information Systems* 13.1 .3-20.

[E60]. Sindhgatta, Renuka, Bikram Sengupta, and Subhajit Datta (2011). "Coping with distance: an empirical study of communication on the jazz platform." In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pp. 155-162. ACM.

[E61]. Taweel, Adel, Brendan Delaney, Theodoros N. Arvanitis, and Lei Zhao (2009). "Communication, knowledge and co-ordination management in globally distributed software development: Informed by a scientific software engineering case study." In *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*, pp. 370-375. IEEE.

[E62]. Taxén, Lars (2006). "An integration centric approach for the coordination of distributed software development projects." *Information and software technology* 48, no. 9: 767-780.

[E63]. Välimäki, Antti, and Jukka Kääriäinen (2008). "Patterns for Distributed Scrum—A Case Study." In *Enterprise interoperability III*, pp. 85-97. Springer London.

[E64]. Vallon, Raoul, Stefan Strobl, Mario Bernhart, and Thomas Grechenig (2013). *Inter-organizational Co-development with Scrum: Experiences and Lessons Learned from a Distributed Corporate Development Environment*. Springer Berlin Heidelberg.

[E65]. Yadav, Vanita, Monica Adya, Dhruv Nath, and Varadharajan Sridhar (2007). "Investigating an'Agile-Rigid'Approach in Globally Distributed Requirements Analysis." *PACIS 2007 Proceedings*: 12.

[E66]. Yeo, Alvin W. "Global-software development lifecycle: An exploratory study. (2001)." In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 104-111. ACM.

[E67]. Zieris, Franz, and Stephan Salinger (2013). "Doing scrum rather than being agile: a case study on actual nearshoring practices." In *Global Software Engineering (ICGSE), 2013 IEEE 8th International Conference on*, pp. 144-153. IEEE.

Appendix B: Overview of Existing Software Development Approaches

No.	Software Development Approach	Phases Execution in the Software Development Approach
1.	**Ad-Hoc**	In this approach a developer writes code without any formal software requirement specification document. There is no design phase; the developer directly goes to coding phase and follows a build and fix approach.
2.	**Linear**	This the traditional approach, in which the phases are executed linearly. That is when one phase is complete, only then can the developer move to the next phase. For example, the team needs to first complete the design phase before they can go into development phase and once a phase has been completed the development team cannot go back to it.
3.	**Evolutionary**	In evolutionary approach, the development team can move across analysis, design and coding phases, thus providing the developers the freedom to go back to design phase and correct a design mistake. However the team has to stick to the objectives of the project and cannot change them.
4.	**Iterative and Incremental**	Iterative and incremental approach allows the team to repeat a phase, as many times they need to and allows them to move back and forth between different phases based on the teams' requirement.

Appendix C: Overview of Beecham et al. (2014) GSD Solution

Issue raised in Interviews	Agile Practice
Directly Addressed	
role – skills mismatch	Self-organizing teams
unclear roles and responsibilities	Self-organizing teams; defined roles (i.e. Scrum Master, Product Owner)
on-site need for autonomy	Self-organizing teams
disenfranchisement, demotivation	Self-organizing teams; "people over process"
knowledge sharing	Daily stand-up
unsustainable working hours	"Sustainable pace"
lack of defined process	Scrum, XP, or other method
vague/missing requirements	Product owner role; Planning Game; frequent iterations; on-site customer
unpredictable customer expectations	"embrace change" , "customer collaboration"
Partially Addressed	
unclear/poorly written English	"working software over documentation"
proposal document used for wrong purpose	"working software over documentation"
tension between Head Office and on-site	Self-organizing teams
Not addressed	
lack of training	
limited career path	
lack of time overlap (delays; forced to adapt to Head Office hrs)	
excessive travel	

Appendix D: Sample Questions for Semi-Structured Interviews

The duration of the interview is 30-45 minutes. We have divided the interview into 3 section based on the type of questions. Section 1 consists of shorts question that require direct answers. Section 2 consists of questions regarding the methodology used for offshore project and section 3 consists of questions that require detail answers, as they will determine the results for the research.

Section 1: Organisation

In this section we will be asking questions regarding the organisation in order to have a general overview of the experience of the organisation in offshoring projects and agile methodology. As this information will help us understand whether organisations with different experiences with agile and offshoring projects face similar problems with communication and coordination issue among the onshore and offshore teams or not.

1. What is the name and age group of the company?
2. What is the organisation's experience with agile?
3. How many years have they done offshoring?
4. How has their experience been with offshoring?
5. How many projects have they done on offshore locations?

Section 2: Methodology

In this section we start by asking the following question in order to get information about what methodology the organisation uses for developing offshore projects to determine if they use agile methodology or not.

1. What software development methodology does your organisation use for offshore projects?

Based on their reply of the above question we will ask further questions s to understand how the organisation uses agile and whether they face any issues with it:

- Have you used agile methodology for any offshore project?
- Due to the difference in time do you feel you have to wait a lot for feedback/work from the offshore team? Does it make it difficult to complete the project on time?
- How does your organisation manage time different between the teams?
- Does the offshore and onshore teams use the same development methodology?
- Do you feel you have to repeat yourself a lot while communicating with the offshore team? Does this slow down the development process?

Section 3: Research

In this section, we ask detail description of the following questions to understand if the organisation is facing or has faced communication and coordination issues while developing software offshore and if they managed to over come it or not and if they have faced any other challenge while developing offshore projects:

1. How efficient is the current methodology in handling the communication and coordination of work between the onshore and offshore team.
2. Have you faced any difficult in the current methodology? If yes where you able to solve the issue?
3. In the current methodology do you think there is need for some modification that would help improve the offshoring projects?

Appendix E: Consent Form and Data Processing Statement

Below is the consent and data processing statement form, the organisations we interviewed signed.

Interview consent and data processing statement

Name: Maryam Kausar, email: m.kausar@edu.salford.ac.uk

If you consent to being interviewed and to any data gathered being processed as outlined below, please print and sign your name, and date the form, in the spaces provided.

Project Facts:

- This project - 'Agile and Beyond Agile Software Development Techniques for Application in Offshore Development" is being conducted by a research team at the University of Salford.
- All data will be treated as personal under the 1998 Data Protection Act, and will be stored securely.
- Interviews will be recorded and transcribed by the research team.
- A copy of your interview transcript will be provided, free of charge, upon request.
- Data collected may be processed manually and with the aid of computer software.
- I have the right to withdraw from the research at any stage in the research

Consents:

- With respect to information resulting from the study being used in future publication and/or reports outside the research team, I consent to the following:

  My name may be identified
  □ YES □NO

  My words may be quoted
  □ YES □NO

- I am willing to cooperate in the future if a need for further information is required

  □ YES □NO

- I/my employer consent to our interviews being recorded by recording devices.

□ YES □NO

Please print your name: .....

Organisation: .....

Signature: ..... Date: .....

Appendix F: Unrevised Distributed Agile Pattern Catalogue

In this section we have presented the unrevised distributed agile patterns before they were validated. As most of the patterns were not changed we have only presented the ones that were changed after the validation process.

a) Management Patterns

In this section we have described the detail of each management pattern.

1. Scrum of Scrum Pattern

In agile methodology, Scrum is an iterative and incremental project management approach that provides a simple framework that " inspect and adapt" (Hossain, Babar, and Paik, 2009). We observed that in offshore projects the onshore and offshore team practices separate scrums in order to develop the project. Based on this observed practice we have designed the following pattern details:

Pattern Name

Scrum of Scrum Pattern

Intent

To apply scrum, sub-teams are formed based on location. Each team has its own scrum. Scrum of scrum meetings are arranged to discuss the progress of the project, which is attended by key people.

Also Known As

Scrum meeting or Meta Scrum

Category

Management category as this pattern helps the onshore and offshore team manage their separate scrums and keep each other updated of the project progress.

Motivation

Consider a team that is divided into sub-teams based on location and they are working on different tasks of a project. It is difficult to have both onshore and offshore teamwork on the same scrum as they both are working on different time zones so in order to work on the same project, both teams work on separate scrums.

To coordinate work both teams arrange a scrum of scrum meeting, which is attended by key people from both teams to update each other of the progress of the project.

Applicability

Use Scrum of Scrum when:

- Team is distributed over different time zones.
- The overlapping working hours between the onshore and offshore team is less.

Participants

- Distributed onshore and offshore agile team.
- Scrum Masters of agile sub-teams and Product owner.

Collaboration

- Key members from onshore and offshore teams decide time for Scrum of Scrum meeting.

Consequences

The Scrum of Scrum pattern has the following benefits and liabilities:

1. It prevents the onshore and offshore team from wasting time on collaborating tasks with each other through online tools as both the teams are working on their own scrum so they don't have to wait for each other to complete tasks.
2. It provides control to both onshore and offshore team to work on their scrum, which avoids the offshore team from having to adjust working hours based on onshore teams availability.
3. It allows key people such as Scrum Masters and Product owners' to discuss the progress of the project without having the whole team present which keeps the meeting time boxed.
4. Its limitation is that due to minimum collaboration between the onshore and offshore team, both sub-teams don't feel they are one team.
5. Since only key people attend the Scrum of Scrum meeting, it limits face-to-face interaction of both onshore and offshore team, which affects trust building between both teams.

Known uses

When CheckFree decided to move their work to an Indian offshore consulting firm they used Scrum of Scrum to gather and review the over all team statistics and progress of the project (Cottmeyer, 2008). Similarly, multiple case studies done on organisations using Scrum for distributed teams also used Scrum of Scrum to coordinate their work with offshore team (Hossain, Babar, and Paik, 2009; Paasivaara, Durasiewicz, and Lassenius, 2009). Siemens also used Scrum of Scrum for two large distributed projects in which the development teams were located in USA, Europe and India. In their Scrum of Scrum meetings they covered technical and managerial issues that occurred in multiple teams (Avritzer et al., 2010)

Related Patterns

Scrum of Scrum pattern is often used with Local Sprint Planning Pattern (and Asynchronous Retrospective meeting Pattern as the onshore and offshore team members are working on different story cards and at the end have their separate retrospective meetings to discuss the sprint

2. Local Standup Meeting

Agile methodology focuses on conducting a daily standup meeting. We observed that in offshore projects the onshore and offshore team conduct their own separate daily standup meetings and use online tools such as Wiki's to share meeting minutes with each other. Based on this observed practice we have designed the following pattern details:

Pattern Name

Local Standup Meeting Pattern

Intent

To discuss daily updates on work done, each local team will conduct their own standup meetings.

Also Known As

Daily Scrum meeting or daily meeting

Category

Management category as this pattern helps the team members of both onshore and offshore team manage their daily activities and update each other with the work done.

Motivation

Consider a team that is divided into sub-teams that are located on different time zones. To have a collaborative daily standup meeting is difficult and time consuming as the offshore team either has to come early to work or stay late to attend the meeting. To avoid this, the onshore and offshore team conducts separate local standup meetings in which they answer the following questions:

- What did you do yesterday?
- What are you going to do today?
- What is getting in your way?

After conducting local standup meetings both onshore and offshore team share their meeting minutes via online tools such as Wiki's to keep both the teams up to with the progress of the project.

Applicability

Use local standup meeting when:

- Team is distributed over different time zones.
- The overlapping working hours between the onshore and offshore team is less.

Participants

- Distributed onshore and offshore agile team.

Collaboration

- Both onshore and offshore team share meeting minutes with each other using online tools.

Consequences

The local standup meetings pattern has the following benefits and liabilities:

1. It prevents the offshore team from waiting for the onshore team's availability to conduct the daily standup meeting.
2. It allows both onshore and offshore team flexibility to conduct their own standup meeting at whichever time they want.
3. It limits the onshore and offshore team from real-time communication and both team heavily rely on the meeting minutes so any mistake can lead to misunderstanding in the progress of the project.

Known uses

Organisations such as PulpCo (Paasivaara, Durasiewicz, and Lassenius, 2009) and Wipro Technologies (Sureshchandra et al., 2008) use local standup meetings for communicating the progress of the project with team members.

Related Patterns

Daily Standup meeting pattern is often used with Global Scrum Board Pattern as once the onshore and offshore team members have conducted their meetings they share the meeting minutes on a shared tool so that both can see the project progress.

3. Local Sprint Planning Meeting Pattern

In agile, a scrum consists of many sprints. The duration of a sprint varies from 2 weeks to 4 weeks depending on the size of the project. At the start of every sprint the team has a sprint-planning meeting in which the team defines the goal of the sprint and prepare the sprint backlog. When the team is divided and is working on different modules of the project it has been observed that the onshore team members and offshore team members conduct their own separate sprint planning meetings. Based on this observed practice we have designed the following pattern details:

Pattern Name

Local Sprint Planning Meeting Pattern

Intent

Each team will have their own sprint planning meetings

Also Known As

Sprint Planning Meeting or Iteration Meeting

Category

Management category as this pattern helps the onshore and offshore team to work on their separate module and conduct independent scrum and sprint planning meetings.

Motivation

When a project is distributed to a team that is divided over different time zones, and are working on different modules of the project and are conducting their own Scrum. As the onshore and offshore team conducts their separate Scrum, they also conduct separate sprint planning meetings to decide what they will develop during a sprint. Both teams prepare their sprint backlogs, which are shared using online tools.

Applicability

Use Local Sprint Planning Meeting pattern when:

- Team is distributed over different time zones and is working on different modules/subsystem of the project.

Participants

- Distributed onshore and offshore agile team.

Collaboration

- The onshore team and offshore team share sprint backlog with each other to show the work they will be doing over the next sprint.

Consequences

The Local Sprint Planning Meeting pattern has the following benefits and liabilities:

1. It allows both teams to work independently without having to wait for the onshore team to be available to conduct the meeting.

2. It provides control to both onshore and offshore team to work on their separate scrum and conduct their own sprint planning meeting, which avoids the offshore team from having to adjust working hours based on onshore teams availability.
3. Both teams can share their sprint backlog with each other, which provides visibility of the project progress.
4. As both the teams are working independently it can cause the teams to feel as they are not part of one team and create an effect that they are two separate teams.

Known uses

When CheckFree decided to move their work to an Indian offshore consulting firm they used local sprint planning meetings to plan their sprint activities (Cottmeyer, 2008).

Related Patterns

Local Sprint Planning Patterns in often used with Scrum of Scrum Pattern and Global Scrum board Pattern as the meetings minutes of the planning meeting are shared with both onshore and offshore team members.

4 Local Pair Programming Pattern

In agile, pair programming consists of two programmers that share a single workstation that is they share one screen, keyboard and mouse. The programmer using the keyboard is usually called the "driver", the other, is called “navigator” as he is activity giving his remarks on the code and helping the driver to write the code. The programmers are expected to switch roles after every few minutes. It has been observed that when the team is divided on different locations, the team members that are co-located form pairs as it is difficult to form pairs with other locations team members due to the time difference. Based on this observed practice we have designed the following pattern details:

Pattern Name

Local Pair Programming

Intent

Make pair programming teams from the same location.

Also Known As

Pair Programming or Paired Programming

Category

Management category as this pattern helps the local team members to form pairs and work on their story card.

Motivation

When a team is distributed over different locations based on time zones, it is difficult to form pairs between them due to the time difference in working hours. So each team forms pairs based on their location as they are co-located and can work together. Pair Programming helps improve the quality of code as instead of one person writing the code the other person is checking the code.

Applicability

Use Local Pair Programming pattern when:

- Team is distributed over different time zones and is working on different modules/subsystem of the project

Participants

- Distributed onshore and offshore agile team and working on different story cards.

Collaboration

- The onshore team and offshore team members share a keyboard with a fellow team member from their respective location.

Consequences

The Local Pair Programming pattern has the following benefits:

1. The offshore team members don't have to wait for the availability of onshore team members to start work.
2. Some organisations feel it's a waste of having two resourcing working on the same thing.

Known uses

In a case study conducted by Maruping (2010) on an organisation that had its development centers in India, U.S West Coast, U.S Mid-West and U.S East Coast showed that pairs where made on the bases of physical locations.

Related Patterns

Since in Local Pair programming we are selecting team members from the same location, we often use it with Local Sprint Planning Meeting.

5 Asynchronous Retrospective meetings Pattern

In agile, when a team is using Scrum at the end of every sprint after the sprint review meeting, a retrospective meeting is conducted which is attended by only the team members and the scrum master. In this meeting the team discusses all the good and bad things that happened during the sprint and how they can improve their work. They also discuss the feedback given by the client. It has been observed that when the team is divided on different time zones, teams conduct their own retrospective meeting, as due to the time difference it is difficult to have a collective retrospective meeting at the end of each sprint (Kamaruddin, 2012). Once both the onshore and offshore teams have conducted their retrospective meeting they share the meetings minutes with each other using online tools. Based on this observed practice we have designed the following pattern details:

Pattern Name

Asynchronous Retrospective meetings

Intent

Teams conduct separate retrospective meetings based on location and share the key information via online tools. The Scrum Masters discuss possible improvements with the team based on the feedback from the client.

Also Known As

Retrospective meetings or Iteration Retrospective

Category

Management category as this pattern helps the onshore and offshore team members to review their sprint and discuss their performance. The Scrum

Master advises the team on how they can improve their performance and discusses the feedback of the client.

Motivation

When a team is divided on different time zones and are working on different modules/subsystems of a project and conducting their own independent Scrum and sprints it is difficult to have a collective retrospective meeting. The onshore and offshore team members conduct their separate retrospective meetings to discuss what went good and bad in the sprint and how they can improve their work in the next sprint. The Scrum master attends these meetings and gives feedback on the performance on the team and discusses the remarks given in the sprint review meeting by the client. Once both teams have conducted their retrospective meetings they share the meeting minutes with each other using an online tool.

Applicability

Use Asynchronous Retrospective meetings when:

- Team is distributed over different time zones and is working on different modules/subsystem of the project.

Participants

- Distributed onshore and offshore agile team members.
- Scrum Master

Collaboration

- The onshore team and offshore team members share meeting minutes at the end of their retrospective meetings.

Consequences

The Asynchronous Retrospective meetings pattern has the following benefits and liabilities:

1. It allows onshore and offshore team members to conduct retrospective meeting independently of each other's availability.
2. It helps team members discuss their independent problems and doesn't make both onshore and offshore team to be present for the meeting.
3. Since onshore and offshore team members conduct separate retrospective meetings they don't understand each other's problems.

Known uses

Elastic Path, a Vancouver, British Columbia-based company decided to offshore their work to Luxsoft, an outsourcing partner used asynchronous retrospective sessions to discuss the sprint progress and well as what improvements they can make. Once all locations had conducted their retrospective meetings, they posted comments and results on SharePoint, which were then viewed by Scrum Master and technical lead for their remarks (Vax, 2008).

Related Patterns

We often used Scrum of Scrum Pattern with Asynchronous Retrospective meeting Pattern. It is also often used with Local Sprint Planning Pattern (explained in 8.3) as in order to review the progress of a sprint and the team we use retrospective meeting. After all the distributed teams have conducted their retrospective meetings they share the meeting minutes with each for which they use Global Scrum Board Pattern.

b) Communication Patterns

In this section we have described the detail of each communication pattern.

1. Global Scrum Board Pattern

Agile has many artefacts such as product backlog, sprint backlog, storyboard, task board, team velocity and burndown charts which help the team in managing the project. It has been observed that when the team is divided to different locations they maintain a online record of all these artefacts so that they can share them with each other using online tools such as Wiki's, Rally and Jira (Danait, 2005; Beruzuk, 2007; Cottmeyer, 2008). Based on this observed practice we have designed the following pattern details:

Pattern Name

Global Scrum Board Pattern

Intent

An online-shared Scrum board, will be used by, both onshore and offshore team to view the product backlog, storyboard, task board, burn down charts and other agile artefacts using online tools.

Also Known As

Scrum Board or Agile Story Board

Category

Communication category as this pattern helps the onshore and offshore team communicate with each other using an online tool to view each other's work and understand the progress of the overall project.

Motivation

When a project is distributed to a team that is divided over different time zones, and are working on different modules of the project, to share their work they use an online tool to display agile artefacts. Based on the work done by both teams it is easier to see the progress of the project and it helps understand if there is a problem with a team or not.

Applicability

Use Global Scrum Board pattern when:

- Team is distributed over different time zones.

Participants

- Distributed onshore and offshore agile team.

Collaboration

- The onshore team and offshore team share agile artifacts with each other to show their progress.

Consequences

The Global Scrum board pattern has the following benefits:

1. It allows the onshore and offshore teams to understand the progress of the project.
2. It increases the visualization of the work done by each team.

Known uses

FAST, a search company with headquarters in Norway while building a search application on top of their core search platform experimented with couple of online tools to keep both teams updated with the progress of the project. They tired XPlanner and Jira and settled for Jira, which is a web-based tool that allowed the remote team members to view the backlog and update tasks whenever they wanted (Berzuk, 2007). Similarly in a study done by Cristal et al. (2008) on an organisation that has development centers across North America, South America and Asia concluded with that the use of a global scrum board can help improve the productivity of global agile teams. Similarly companies like Valtech (Danait, 2005), Telco (Ramesh et al., 2006), BNP Paribas (Massol, 2004), Aginity LLC (Armour, 2007) and SirsiDynix (Sutherland et al. 2007) used online tools to share agile artefacts with their offshore team members.

Related Patterns

Global Scrum board pattern is often used with Central Code Repository Pattern as the team shares all the agile artefacts and code using an online tool.

2. Central Code Repository Pattern

In agile, when a team is using Scrum and XP, the team members are divided in pairs of two and are working on different tasks during a sprint. When a task is completed the team members commit their code to a share repository for continuous integration of the code. It is observed that even when the team members are geographically apart they still use a share code repository where they commit their code so that all the team members can see the code as well as determine the progress of the project. Based on this observed practice we have designed the following pattern details:

Pattern Name

Central Code Repository

Intent

The whole team will maintain a central code repository so that both team can see each other's code and see the progress of the work done.

Also Known As

Source Code Repository or Global Build Repository

Category

Communication category as this pattern helps the onshore and offshore team members to write code and share it on a central code repository where all team members can see the code and edit it if required.

Motivation

When a team is divided on different time zones and are working on different modules/subsystems of a project they use a central code repository to share their work with all team members. They can use online tools such as GitHub for committing their code and maintain versions of the project (Räty, 2013). This helps the whole team to see the code and provides visibility of the project progress.

Applicability

Use Central Code Repository when:

- Team is distributed over different time zones and is working on different modules/subsystem of the project.

Participants

- Distributed onshore and offshore agile team members.

Collaboration

- The onshore team and offshore team members share a keyboard with a fellow team member from their respective location and once they have finished a task they commit their code to a central code repository.

Consequences

The Central Code Repository pattern has the following benefits:

1. It allows onshore and offshore team members to view each other's code.
2. It helps in determining the progress of the project.
3. As all the team is committing to a central repository, if a team commits code with errors it can affect the whole build of the project.

Known uses

WDSGlobal is a leading global provider of knowledge-based services to mobile operators, manufacturers and application and sales channels. In 2004 they combined their developments, which were located in UK, USA and Singapore. They shared their code on a central code repository to minimize duplications and reduce cost of maintenance (Yap, 2005). Many companies use central code repository for their distributed projects such as Extol International (Kussmaul et al., 2004), Valtech (Danait, 2005), Manco (Ramesh et al., 2006), Aginity LLC (Armour, 2007) , SirsiDynix (Sutherland et al., 2007), CEInformant (Bose, 2008) and ABC Bank (Modi et al., 2013).

Related Patterns

Central Code Repository pattern is often used with Global Scrum Board Pattern.

3. Asynchronous Information Transfer Pattern

Agile emphases on close face-to-face communication between the team members rather than detailed documentation. When a team is distributed on different time zones it has been observed that the teams adopted asynchronous tools for sharing information with each other such as emails, wikis, SharePoint. Based on this observed practice we have designed the following pattern details:

Pattern Name

Asynchronous Information Transfer

Intent

Due to the time difference between the onshore and offshore team use online tools to exchange information with each other. Each team should response to queries within 12 hours.

Also Known As

Information Transfer or Knowledge Sharing

Category

Communication category as this pattern helps the onshore and offshore team members to answer each other's queries within 12 hours.

Motivation

When a team is divided on different time zones they may have queries about work but due to the time difference they cannot get a direct reply at that time so they use emails to communicate queries, which are then answered within 12hours max. Organisations have set standards for response time in order to avoid delays in work (Vax, 2008).

Applicability

Use Asynchronous Information Transfer when:

- Team is distributed over different time zones.

Participants

- Distributed onshore and offshore agile team members.

Collaboration

- The onshore team and offshore team members share information and ask queries using asynchronous tools.

Consequences

The Asynchronous Information Transfer pattern has the following benefits:

1. It allows onshore and offshore team members to exchange information when synchronous communication cannot be conducted due to working hours time difference.
2. It helps team members from waiting for onshore team member availability to ask a query.
3. If the team members don't respond timely it can cause delays in the project.

Known uses

VTT Technical Research Centre of Finland and National University of Ireland conducted a research on two organisations that were developing a system together. One organisation was a customer organisation in U.S and the other organisation was a development organisation located in Ireland. Based on their findings the companies used asynchronous tools for communication. They used wikis for storing documents and meeting minutes and used Emails for decisions and queries (Korkala, 2010). Similarly Valtech used Twiki for asynchronous communication (Danait, 2005).

Related Patterns

Asynchronous Information Transfer pattern is often used with Global Scrum Board and Synchronous Communication Pattern.

4. Synchronous Communication Pattern

Agile emphases on close face-to-face communication between the team members rather than detailed documentation. When a team is distributed on different time zones it has been observed that the teams adopted asynchronous tools for sharing information with each other such as emails, wikis, SharePoint. Based on this observed practice we have designed the following pattern details:

Pattern Name

Synchronous Communication Pattern

Intent

In order to discuss issues the teams uses synchronous tools for voice, video conferencing, document sharing, application sharing etc.

Also Known As

Synchronous Knowledge Transfer

Category

Communication category as this pattern helps the onshore and offshore team members to answer each other's queries within 12 hours.

Motivation

When a team is divided on different time zones they may have queries about work but due to the time difference they cannot get a direct reply at that time so they use emails to communicate queries, which are then answered within 12hours max. Organisations have set standards for response time in order to avoid delays in work (Vax, 2008).

Applicability

Use Synchronous Communication Pattern when:

- Team is distributed over different time zones.

Participants

- Distributed onshore and offshore agile team members.

Collaboration

- The onshore team and offshore team members share information and ask queries using asynchronous tools.

Consequences

The Synchronous Communication pattern has the following benefits:

1. It allows onshore and offshore team members to exchange information when synchronous communication cannot be conducted due to working hours time difference.
2. It helps team members from waiting for onshore team member availability to ask a query.
3. If the team members don't respond timely it can cause delays in the project.

Known uses

CampusSoft is a UK based company that used synchronous communication when they moved to Agile with their offshore suppliers in India and Romania. They used video conferencing facilities for planning sessions and later shifted to WebEx sessions and GoToMeeting so that they could share desktops with the remote team members. For daily Scrum meetings they preferred to use Skype call and made everyone wear headsets to make the meeting easier. For sprint review meetings they used sharing desktop tools as well as conference phones so that members from both end could talk with each other (Summers, 2008).

Related Patterns

Synchronous Communication pattern is often used with Global Scrum Board and Asynchronous Information Transfer Pattern

c) Collaboration Patterns

In this section we have described the detail of each collaboration pattern.

1. Collaborative Planning Poker Pattern

An agile team plays planning poker to put points estimation on each story card. The product owner also takes part in this activity. He tells the team the intent and value of a story card based upon which the development team assigns an estimation on the card. Based on the points assigned the team members who assigned the lowest and highest estimation will justify their reasons. The team will have a brief discussion on each story and assign an estimation upon which the whole team agrees on.

It has been observed that even when the team is distributed the planning poker activity is conducted when both teams are co-located for the project planning activity. Based on this observed practice we have designed the following pattern details:

Pattern Name

Collaborative Planning Poker Pattern

Intent

Only Key people will hold this activity from onshore and offshore team.

Also Known As

Planning Poker or Scrum Poker

Category

Collaborative category as this pattern helps the onshore and offshore team to discuss the duration of a story card.

Motivation

When a project is distributed to a team that is divided over different time zones, it is important that all the team members agree on the time duration of a feature before they start developing the project. This helps estimate the duration of the project completion as well as it provides visibility of project progress. For this purpose the onshore and offshore team members play planning poker in order to collectively agree on the estimation of a story card. Once the estimation is decided they write it down and approved by the product owner/client and move on to the next story card, till all the story cards are estimated.

Applicability

Use Planning Poker pattern when:

- Team is distributed over different time zones and will be working on different story cards in a sprint.

Participants

- Distributed onshore and offshore agile team.
- Product owner/Client.

Collaboration

- The client approves the estimation made by the team members.

Consequences

The Planning Poker pattern has the following benefits:

1. It allows the onshore and offshore teams to agree on a story card estimation, which helps the team establish their team velocity.
2. It provides the product owner/client with estimation of project completion

3. If all team members don't agree on estimation on a story card it can lead to a long discussion, resulting the planning poker to prolong.

Known uses

UShardware has development centers across North America, South America and Asia. When transitioning to distributed agile environment they used planning poker activity for estimation of their story cards (Wildt, Prikladnicki, 2010).

Related Patterns

Planning Poker Pattern is often used with Collective Project Planning as its better to conduct this pattern when the whole team is co-located. The estimated story cards are then shared on the Global Scrum board so that whole team can view them during the project.

2. Follow-the-sun Pattern

When a agile team is distributed over different time zones it has been observed that companies cut down on cost by increasing development time by adopting "follow the sun" workflow which means it allows 24 hours development due to the difference in time zones allowing a company's employees to do development 24hrs a day (Carmel, Agarwal 2001; Yap,2005; Kroll, et al.,2012).Based on this observed practice we have designed the following pattern details:

Pattern Name

Follow-the-sun Pattern

Intent

Onshore and offshore teams will work 9a.m -5p.m according to their own time zones.

Also Known As

24hours development Patterns

Category

Collaboration category as this pattern helps the onshore and offshore team manage their work and handoff their work to the other team before they leave from work.

Motivation

Consider a team that is divided into sub-teams that are located on different time zones. To adjust the working hours of the offshore team for both the onshore and offshore team work together is difficult as the offshore team has to come in the evening and stay till early morning. This makes the offshore team feel they are less important in comparison to the onshore team and it affects the employees' social life.

In order to avoid that, the onshore and offshore team "follow-the-sun" approach. For example an employee works from 9 a.m. to 5p.m. in the USA. At 5 p.m. she hands over the incomplete task to a colleague in Australia who works from 9 a.m. to 5 p.m. based on his time zone. At 5 p.m. according to his country, he transfers the updated task to a colleague in Poland who works on

the updated task for the next eight-hours and then forwards it to his colleague in the USA (Gupta, 2009).

While the employee in the USA had left work two of her colleagues worked on her task as when she will come to the office next morning a lot of the work would have been done. This work scenario takes advantage of the geographical distances as it allows people from different time zones to work round-the-clock in order to build software (Gupta, 2007).

The work distribution among the team can be done in two ways. First either the 3 teams distributed on different geographical locations work on the same task and each team keeps updated the task as mentioned in the above scenario or secondly a most efficient way is that we divided different aspect of the same problem among the team for example in Figure 3 we can see how a problem has been distributed among 3 teams (Gupta, 2009):

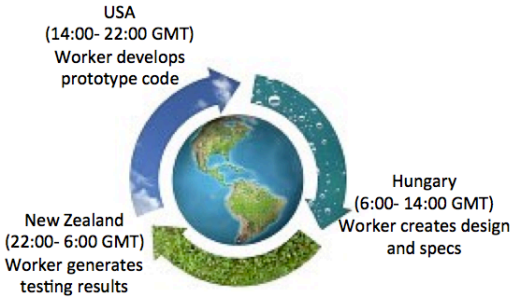


Figure 7.3.1. Distribution of work among 3 distributed teams.

Applicability

Use follow-the-sun pattern when:

- Team is distributed over different time zones.
- The onshore and offshore teams are working on different tasks.

Participants

- Distributed onshore and offshore agile team.

Collaboration

- Both onshore and offshore team hand over their work to each other at the end of every working day using online tools.

Consequences

The follow-the-sun pattern has the following benefits and liabilities:

1. It allows continues development during different working shifts across different time zones.
2. It allows both onshore and offshore team to work according to their time zone without having to either come early to work or stay late till early morning.
3. It reduces the development life cycle or time-to-market (Denny, et al., 2008).
4. It limits the onshore and offshore team from real-time communication as the overlapping working hours between different time zones can be less or zero.

Known uses

Yahoo! used follow-the-sun approach when they offshored their Yahoo! Podcast product from Sunnyvale, California campus to Yahoo! Bangalore, India Campus (Drummond, Unson, 2008). Similarly organisations like WDSGlobal (Yap,2005) and Wipro Technologies (Sureshchandra, et al.,2008) use follow-sun-approach.

Related Patterns

Follow-the-sun Pattern is often used with Local Standup meeting Pattern and Scrum of Scrum Pattern as both onshore and offshore team members are working separately.

3. Collective Project Planning Pattern

Agile focuses on individuals and interactions over processes and tools. While planning for the project the whole team is present. Unlike the traditional development where a project manager hands a project plan to the team. In agile the whole team takes part in the planning activity in order to determine when and how the project will be developed. It has been observed that even if the project is of a distributed nature it is better to co-locate the team onshore and offshore team for the project planning activity. Based on this observed practice we have designed the following pattern details:

Pattern Name

Collective Project Planning Pattern

Intent

Both the onshore team and the offshore team will collectively work in the project-planning phase. Once both team have engaged in the project planning activity, the team will prepare the project backlog.

Also Known As

Project Planning or Agile Project Planning

Category

Coordination category as this pattern helps the onshore and offshore team to work together and come up with a project plan.

Motivation

Consider a team that is divided into sub-teams that are located on different time zones and both the teams come to one location to do the project planning activity. In the beginning of any distributed project, the offshore team is invited to the onshore location so that they may work together and understand each other's requirements.

While the teams are co-located they worked on preparing the product backlog and they spend at least one or two sprints together before the offshore team leaves and starts working on the project (Cottmeyer,2008; Therrien,2008). This helps the onshore team by making the offshore team understand their working style and work standard.

Applicability

Use Collective Project Planning pattern when:

- Team is distributed over different time zones.

Participants

- Distributed onshore and offshore agile team.

Collaboration

- Onshore team and offshore team work together to make a product backlog.

Consequences

The Collective Project Planning pattern has the following benefits and liabilities:

1. It allows the onshore and offshore teams to work together and understand each other.
2. Onshore team works with the offshore team and makes them understand what type of work they want.
3. It adds additional cost of travel and stay of the offshore team at the onshore location.

Known uses

FAST, a search company with headquarters in Norway while building a search application on top of their core search platform used collective project planning to co-locate the team and make them work together in project planning activities (Berczuk, 2007). Siemens also used collaborative planning for their distributed projects (Avritzer, Paulish, 2010; Avritzer et. al, 2007) in which team members from multiple sites got involved in the early stages of the project in order to create an open communication channel and high level of trust among the distributed team members (Avritzer et. al, 2010)

Related Patterns

Collective Project Planning Pattern is often used with Project Charter Pattern as it provides a central document that consists of the goal and objectives of the project written by the client.

4. Visit onshore-offshore Team Pattern

As agile emphases on close face-to-face communication between the team members it has been observed that when the team is divided on different time zones, the team members travel quarterly or annually to visit each other. This activity helps build trust among the team members and helps them understand each other's cultural differences (Ramesh, 2006; Therrien, 2008; Summers, 2008; Paasivaara et al., 2014). Based on this observed practice we have designed the following pattern details:

Pattern Name

Visit onshore-offshore Team Pattern

Intent

Both onshore and offshore teams should quarterly / annual visit each other in order to build trust, exchange cultural values and improve team coordination.

Also Known As

Travel onshore-offshore

Category

Collaboration category as this pattern helps the onshore and offshore team members to co-locate and understand each other and build a good relationship, which improves team coordination.

Motivation

When a team is divided on different time zones they don't feel that they are both part of one team and they don't trust each other. They don't understand each other's cultural values and work ethics. In order to solve these issues the onshore and offshore team visits each other to develop the feeling of trust and understand each other's cultural and working values. During these visits they attend training together as well as engage into informal activities to better understand each other. This helps build a bond between the team members, which results in good team coordination.

Applicability

Use Visit onshore-offshore Team when:

- Team is distributed over different time zones.

Participants

- Distributed onshore and offshore agile team members.

Collaboration

- The onshore team and offshore team members visit each other to improve team coordination.

Consequences

The Visit onshore-offshore Team pattern has the following benefits:

1. It allows onshore and offshore team members to exchange cultural values with each other and work ethics.
2. It helps team members to feel they are part of one team, which develops trust among onshore and offshore team members.
3. The travelling adds additional cost to the project budget.

Known uses

Ericsson is a Swedish multinational provider of communications technology and services. To build a XaaS platform and a set of services they used agile software development methodologies. The development team was distributed over 5 sites located in 3 countries. Four of the sites were located in Europe and one was located in Asia. They conducted workshops, which were attended by team members from different locations. The purpose of these workshops was to create a common vision for the whole organisation by setting common values as well also to improve the collaboration between the sites, thus build trust (Paasivaara et al. 2014).

Related Patterns

Visit onshore-offshore Team pattern is often used with Collective Project Planning Pattern as planning is better done when the whole team is co-located.

d) Verification Patterns

In this section we have described the detail of each verification pattern.

1. Project Charter Pattern

In project management, project charter is a statement that defines the scope, objectives and participants of a project. It is used to explain the roles and responsibilities, outline of the project objectives and identify main stakeholders. It has

been observed that while starting a distributed project using agile many organisation use project charter to clarify the goals and objectives of the project to both onshore and offshore team (Galen, 2009). Based on this observed practice we have designed the following pattern details:

Pattern Name

Project Charter Pattern

Intent

Before starting the project planning activity, agile teams use project charter in order to have a central document between the onshore and offshore team that defines the project.

Also Known As

Project Definition or Project Statement

Category

Verification category as this pattern helps the onshore and offshore team to have a central document clarifying the project goals and objectives, which is written by the product owner/client.

Motivation

When a project is distributed to a team that is divided over different time zones, a central document is written known as the project charter, which clarifies the onshore and offshore team the goals and objectives of the project. It also identifies the roles and responsibilities of the onshore and offshore team. The purpose of this activity is to have a document that helps the team in the project-planning task.

Applicability

Use Project Charter pattern when:

- Team is distributed over different time zones.

Participants

- Distributed onshore and offshore agile team.
- Client.

Collaboration

- The client gives the project charter to the onshore team and offshore team to clarify the goals of the project.

Consequences

The Project Charter pattern has the following benefits:

1. It allows the onshore and offshore teams to understand the project.
2. It is intended to clearly set the stage for the project by aligning the team and settings goals and expectations.

Known uses

IONA Technologies used Project Charter for their distributed projects in order to have a central document that clarifies the goals of the project to both onshore and offshore team members (Poole, 2004). Similarly in a case study conducted by Brown (2011) on Agile-at-Scale Delivery it was observed that organisations use project charter.

Related Patterns

Project Charter pattern is often used with Visit onshore-offshore Team pattern.

2. Onshore Review Meeting Pattern

In agile, at the end of each sprint, a sprint review meeting is held in which the team meets with the clients and shows them the work they have done through a demo. In this meeting the client gives remarks about the work and if they require any changes they tell the team. It has been observed that when the team is distributed on different locations, then the team that is co-located with client conducts the review meeting.

Based on this observed practice we have designed the following pattern details:

Pattern Name

Onshore Review Meeting

Intent

The onshore team will present the demo as they are located where the client is.

Also Known As

Sprint Review Meeting

Category

Verification category as this pattern helps the client see the progress of the project as well as they can suggest early changes.

Motivation

Consider a team that is divided into sub-teams that are located on different time zones and one of the team is located in the same country as the client. In order to show the progress of the project it is more convenient that the team, which is located near the client, presents the demo in order to have a face-to-face meeting. Once the meeting has ended the onshore team briefs the offshore teams the remarks of the client.

Applicability

Use Onshore Review Meeting pattern when:

- Team is distributed over different time zones.
- The onshore is located in the same country as the client.

Participants

- Distributed onshore and offshore agile team.
- Client

Collaboration

- Onshore team presents demo to the client and discusses the meeting minutes with the offshore.

Consequences

The Onshore Review Meeting pattern has the following benefits and liabilities:

1. It allows the client to meet the development team face-to-face and give feedback.
2. It allows both onshore and clients to discuss what changes they want and how much time will be required based on the modification.
3. It limits the offshore team from meeting the clients which can cause the offshore team from discussing the changes and giving their remarks on the clients feedback.

Known uses

Wipro Technologies, a global service provider company used onshore review meetings so that they could get quick feedback from the customer/business user, which was then shared with the remote team over mail and/or teleconference (Sureshchandra et al., 2008). Similarly when SirsiDynix, USA outsourced their work to Starsoft, Ukraine they also used onshore review meetings to show the demo of the work done (Sutherland et al., 2007)

Related Patterns

Onshore Review Meeting pattern is often used with Asynchronous Retrospective Meetings Pattern because after the demo both onshore and offshore team members conduct their separate retrospective meetings.