



## THE DEVELOPMENT OF A SOFTWARE DEVELOPMENT FRAMEWORK BY COMBINING TRADITIONAL & AGILE METHODS TO ADDRESS MODERN CHALLENGES

Bernadette Nortier<sup>1</sup>, Konrad von Leipzig<sup>2</sup> and Corne Schutte<sup>3</sup>

<sup>1</sup>Department of Industrial Engineering  
University of Stellenbosch, South Africa  
[bernadette.j.nortier@gmail.com](mailto:bernadette.j.nortier@gmail.com)

<sup>2</sup>Department of Industrial Engineering  
University of Stellenbosch, South Africa  
[kvl@sun.ac.za](mailto:kvl@sun.ac.za)

<sup>3</sup>Department of Industrial Engineering  
University of Stellenbosch, South Africa  
[corne@sun.ac.za](mailto:corne@sun.ac.za)

### ABSTRACT

This paper is focused on the development of a Software Development Framework (SDF) by applying the principles of Agile with phase management to produce an Agile SDF that is focused on delivering working software in time and on budget within an environment where requirements are not well identified nor defined up-front. This paper will focus on delivering a development process, which handles uncertain requirements and can adapt to software requirements that change late in the development cycle.

The framework is based on Rational Unified Process (RUP) as its skeleton and supplemented with Scrum to produce a risk and value driven Agile methodology covering the entire development lifecycle. The principle behind the amalgamation of these methodologies is to maximize the individual strengths of both traditional and Agile software development methodologies, while controlling their weakness. Some level of the predictability, stability, and high assurance is compromised for agility to create a process that can easily adapt to rapid changing business requirements and still produce high quality software.



## 1. INTRODUCTION

Today, the mainstream software industry has a "poor track record" when it comes to delivering working software on time and within budget. According to the Chaos Report [13] a small percentage of projects (16.2%) are completed on-time, on-budget and manage to meet all requirements. 52.7% of projects are either over-budget, delivered late and/or do not meet requirements. 31.1% of projects are cancelled at some point during the development. The organisation under discussion in this article is no different and faces many problems that inhibit its ability to meet its requirements. This article will look at the problems faced by the organisation and how it can be resolved by combining Scrum and RUP.

## 2. THE ORGANISATION AND ITS INDUSTRY

The specific organisation is one of the subsidiaries within a global organisation, which is one of the world's leading technology providers of intelligent utility metering systems. The software produced by the organisation focuses on the vending and tariff systems for pre-paid electricity; as well as vending application management, customer contracts, account management and meter data management within the pre-paid electricity environment. The following section will look the organisation's development environment and structure.

### 2.1 Project team structure

#### 2.1.1 *Product Management*

Product management is performed by product managers within the Marketing department. They are responsible for driving the future of the various software products within the organisation through what is called a product roadmap. This product roadmap outlines the functionality that should be developed in the system over an approximate three to four year period and is revised annually.

#### 2.1.2 *Project Management*

Project managers within the project office of the organisation are responsible for the project management processes to ensure the software systems/products are developed on time and within budget.

#### 2.1.3 *R&D Systems Department*

R&D Systems is the development department within the organisation and consists of Architects, Software Analysts, Testers, Database Analysts and a small group of developers.

Analysis and design is primarily undertaken by the analysts and architects within this department.

This department is responsible for implementing the development. 60% of the development need is outsourced to third party developers.

Once the development of the software is completed, functional and integration testing is performed in-house by the organisation.



#### **2.1.4 Steering Committee / PAC**

The Steering Committee is responsible for corporate governance and decision-making on all projects and programs. It is the responsibility of a Steering Committee to decide if the development of a product and deliverables makes sense in terms of strategy, the roadmap, market opportunity, feasibility and the business plan. Furthermore, the Steering Committee reviews each phase and milestones.

### **3. PROBLEMS EXPERIENCED BY THE ORGANISATION UNDER STUDY**

The following section will focus on discussing the problems faced by the organisation with their current software development processes.

#### **3.1 Not able to meet deadlines & budget**

The biggest problem that the specific organisation faces is the fact that software releases are not meeting the ever-changing market needs and that the time frames between these releases are too wide spread, resulting in severe negative implications for customers. Furthermore, the organisation struggles to deliver development projects within budget and on time.

#### **3.2 Uncertain and unclear requirements in the beginning of projects**

Due to the nature of the organisation's environment, it does not have the luxury of static requirements. This can be ascribed partially to the fact that requirements are not well-defined, neither well established at the beginning of projects. Subsequently, changes are made throughout the development of the project as a clearer view is ascertained. Another contributing factor is the fast moving industry which the organisation competes in; trends move fast, which requires the organisation to adapt and change to keep up.

#### **3.3 Organisation distribution**

The organisation's structure lends itself to outsourcing, however certain parts of the Software development lifecycle (SDLC) need to be in-house. This creates a challenge in itself since applying Agile development across a distributed organisation is complex and creates the following obstacles to overcome:

- The alignment of both organisations' processes to support the Agile development domain without disrupting the other organisation's development cycle.
- Streamlining both development cycles, while ensuring timely integration of the processes at crucial points in both development cycles.
- The management of the process across the organisations, while allowing both development cycles the necessary freedom that Agile requires and at the same time maintain enough control over organisation A's development cycle to ensure the delivering of deliverables and input required by organisation B's Agile processes. This requires more documentation than Agile typically warrants.

The situation is even more complicated because multiple project teams exist within the organisation, each having different focuses/responsibilities over the course of the SDLC.

Taking all the above items into consideration, the question of what checkpoints to employ and how to apply them in order to ensure that milestones are reached timeously, while simultaneously allowing the necessary freedom for the development cycle to change and adapt where necessary.



### 3.4 The need to adhere to the global high-level Development Process (DP)

Across the global organisation a high-level Development Process (DP) is defined to manage all development. The high-level DP is comprised out of 5 phases of which all projects and programs have to adhere to:

- Phase 1: Concept
- Phase 2: Feasibility
- Phase 3: Development and Verification
- Phase 4: Validation
- Phase 5: Production, Commercialization and Deployment

Furthermore, the global process incorporates phase management whereby phases are controlled by gates/milestone with a formal "Gate Approval" by the Steering Committee. It is the responsibility of the Steering Committee to decide whether the development of a product/system and its deliverables makes sense in terms of strategy, market opportunity, and feasibility as well as supervise the progress of the project throughout. Approval is based on checklists specific to each phase. A phase ends by crossing a Gate/milestone successfully.

The high-level DP is the top process within the global organisation's process hierarchy. Below it is more detailed processes called the mid-level DP. These processes are specifically geared towards the operation of the various subsidiaries of the organisation. Furthermore, each mid-level DP focuses on specific product development processes for either software or meters within the various subsidiaries. Each of the mid-level processes is also required to adhere to the principles of the high-level DP. The SDF is such a mid-level process and subsequently should also follow a phase management approach with milestone at end of each phase.

Despite the fact that the succession of phases in the high-level DP denotes an overall Waterfall Model globally, each phase may have its own development model (Waterfall, V-model, Spiral, Iterative, Agile methodology, etc.), further, it may have several co-existing models.

### 3.5 Maturity level of the development department within the organisation

Since the organisation's in-house development department is relatively young, the development and testing teams have not reached expert capabilities and the required maturity level (at least 10 000 logged hours in their field [14]). This is problematic since most Agile methods are based on the assumption that projects are staffed by technical experts and highly motivated individuals as required by principles 6 and 7 of the Agile Manifesto [15]. Some of them state that it is a requirement if an organisation wants to succeed in Agile, citing that Agile places responsibility on project team members where they are to identify the correct path rather than enforcing the correct path through process controls as the reason [15].

To overcome the maturity level of the organisation's development team (especially in the beginning) more checkpoints and gates might be required than what Agile promotes.

### 3.6 Daily Operations

The testing team is overly involved in deployment, User Acceptance Testing (UAT), handling of operational and maintenance problems of systems. Subsequently, the majority of the testing team's daily tasks are geared towards this. Consequently, the software systems are cursorily validated and tested due to lack of time, leading to poor quality



software products delivered by the organisation. Upon further investigation, it was discovered that the nature of the problem was due to the improper knowledge transfer between the development team and Operations. Therefore, Operations were not able to fulfil their responsibilities in resolving operational and maintenance issues or deploy the system to customer's live environments.

#### 4. THE APPROACH FOLLOWED IN DESIGNING THE SOFTWARE DEVELOPMENT FRAMEWORK (SDF)

After studying various Agile development methodologies (Dynamic System Development Method (DSDM) and Extreme Programming (XP) primarily), it was concluded that the organisation would like to implement Scrum. It was evident that Scrum alone would not suffice as it only solved a small portion of the organisation's problems. Subsequently, it was decided to implement Rational Unified Process (RUP) with Scrum. The following section discusses the reasoning behind these decisions and the factors involved in scaling Agile to fit the organisation's needs, as well as touching on the recommended software development process to be employed by the organisation.

##### 4.1 Scaling agile to fit the organization

According to the Agile Scaling Model (ASM) there are two approaches to scaling Agile to fit an organisation. These are:

- **Disciplined Agile development processes**  
Disciplined Agile development is focused on the self organisation of teams within the development process by applying risk and value driven approaches within an appropriate governance framework. It is best suited for environments with small, co-located teams which deliver fairly straightforward systems. Disciplined Agile development is achieved by scaling core agile methods (e.g. Scrum, XP, etc.) to address the full development lifecycle by combining practices from several methods or adopt a method which already spans the entire development lifecycle. Traditional practices as up-front requirements elicitation and architecture modelling can also be adopted and tailored to achieve this goal, providing they are adapted to reflect agile principles.
- For **Agility at scale models**, disciplined Agile development processes are scaled further based on eight factors: team size; geographical distribution; regulatory compliance; organisational complexity; technical complexity; organizational distribution; and enterprise discipline. Therefore, adopted agile approaches need to be accordingly scaled to the complexity of the organisation structure and development environment [1]. The following figure depicts the different agile approaches according to the ASM as well as where RUP and Scrum fits within this picture.

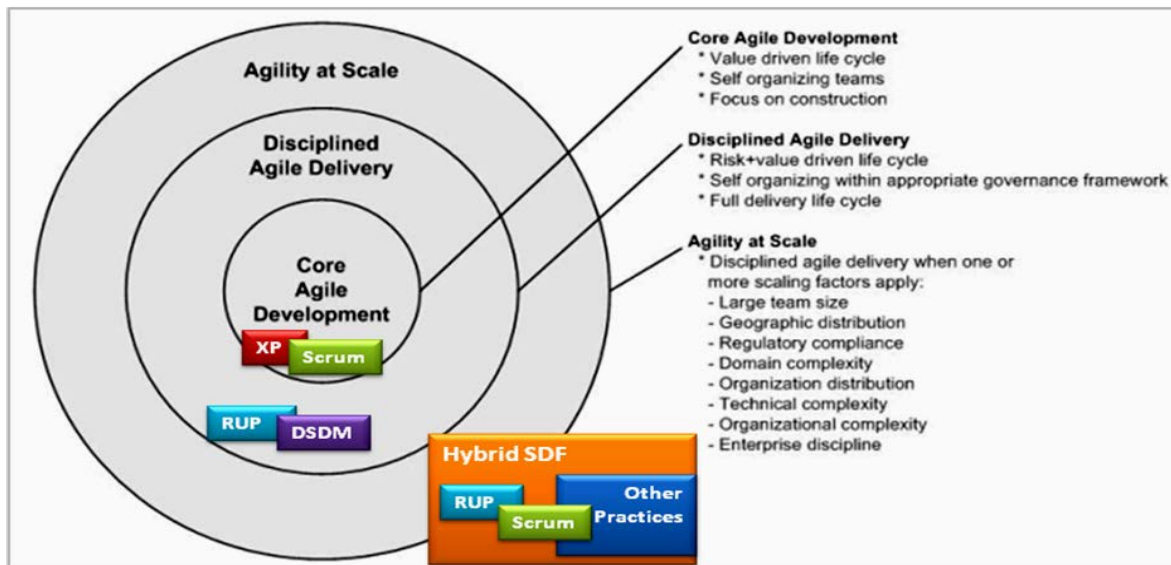


Figure 1: The Agile Scaling Model [1]

For the SDF, it was decided to create an Agility at scale model because of the problems faced by the organisation as discussed in this article. The first step was to ensure that the SDF addresses the full development lifecycle as promoted by disciplined Agile development. The second step was to identify which factors are applicable to the organisation's project team and development environment. Next, was to tailor the adopted strategy to address the range of complexities faced by the team and consequently gear them more appropriately for the real-world needs of the organisation [1].

#### 4.1.1 Disciplined Agile Development

Scrum does not cover the whole software development lifecycle (SDLC); it only covers the development lifecycle from the requirements specification phase up until the testing phase. Therefore, the SDF incorporates RUP as its skeleton and embeds Scrum into the process to produce a risk and value driven Agile methodology across the entire development lifecycle. The combination of RUP and Scrum maximises the strengths of both traditional and agile methods, while limiting the weakness of each approach. However, some degree of the predictability, stability, and high assurance of RUP is lost in the process to gain agility and adaptability to rapid changing business requirements within the development cycle. RUP is only used as a process framework to build onto; the IBM RUP product itself was not adopted.

#### 4.1.2 Agility at Scale

The following figure depicts the level of complexities faced by the organisation:

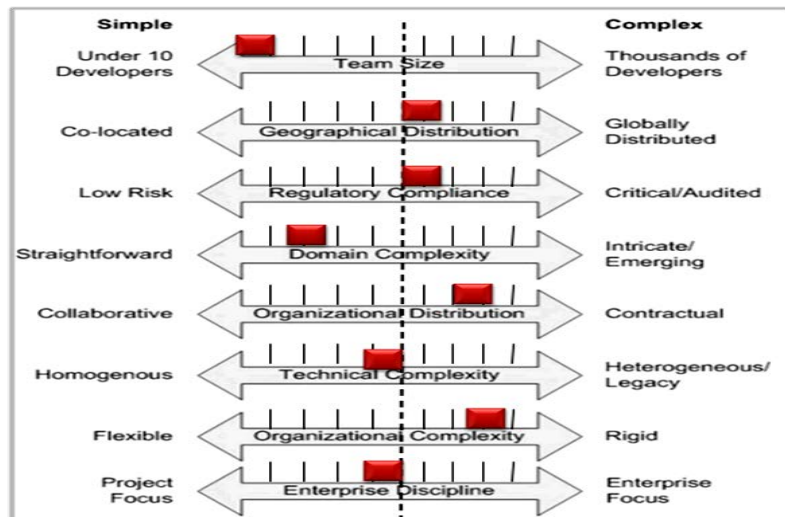


Figure 2: The organisation in terms of the complexity of ASM's scaling factors

The organisation only has four complexities factors to be addressed: the geographical distribution of the organisation; regulatory compliance; organisational distribution; and organisational complexity. Geographical and organisational distribution can be seen as one combined factor since the solution to address the one will automatically resolve the other one.

## 4.2 Why Rational Unified Process (RUP) and Scrum

The following section looks at the strengths of RUP and Scrum and the rationale to why these methods are the foundation of the SDF.

### 4.2.1 Scrum

Scrum focuses more on the managing of software projects by shifting the focus more to the day to day project management responsibilities and general project coordination. Scrum is not only a set of management practices, it is a framework that provides transparency, and a mechanism of "inspect and adapt". Furthermore, Scrum helps to improve the existing development practices in an organisation since it is focused on making visible the dysfunction and impediments impacting the development and team's effectiveness. Scrum does not solve the problems of development; it makes them painfully visible, and provides a framework for people to explore ways to resolve these problems quickly [9].

Subsequently, Scrum is basically a wrapper process since it is primarily an Agile requirement and project management methodology without any established or prescribed development practices. Therefore, making it an ideal methodology to manage whatever development practices are used in an organisation or other development methodologies. This in return makes Scrum an ideal methodology to manage the outsourcing of development since Scrum is independent of the methods and practices employed by third party developers. Consequently, all third party development processes can be managed as a black box that needs to interface with the organisation's own development processes at specific points.

Moreover, Scrum enables projects where the business requirements are hard to quantify upfront:



- It is a methodology that welcomes changes and encourages customers to learn more of their needs as development progresses.
- Changes to requirements can be addressed easier and more efficiently due to short iterations and continuous customer feedback promoted by Scrum.

Furthermore, Scrum also has the ability to address the quality issues experienced by the organisation vis-à-vis meeting customer requirements and delivering software products on time. This is accomplished in the following ways:

- Scrum ensures that all product features and functionality are prioritised according to the highest customer valued items. Subsequently, ensuring that all high priority functionality is developed first and only nice-to-haves are jeopardised when the project timelines are jeopardised.
- Scrum ensures that no major problems within the development cycle are incurred by ensuring that only fully tested and functional components of the system are delivered in the iteration. On occasions where iterations impede to deliver all the expected functionality, the team is forced to confront the issues and resolve them to ensure that future iterations will be able to deliver [6].

Since the organisation under study does not really follow good development or software engineering practises, the organisation's practices are not sufficient to fill the gaps Scrum has in this area. Therefore, Scrum alone will not suffice as a wrapper to the organisation's current practices without adopting another methodology to fill the current gaps.

#### 4.2.2 RUP

RUP is a system development process framework that provides process guidance for all aspects of the SDLC and not just the development phase. Therefore, it is a structure from which one can create and tailor a SDLC as required by the organisation's specific business priorities and skill sets to address its particular needs. Moreover, RUP defines a comprehensive description of the roles (which adhere more strongly to the traditional roles as defined within the organisation), activities, procedures, guidelines and artefacts. These were adopted by the SDF and amended where necessary.

RUP possesses the necessary structure to eliminate requirement uncertainties earlier within the agile development cycle and tightens the process control on the project as it progresses. However, RUP does not assume a fixed set of requirements at the inception of the project, but allows for the refining of requirements as the project evolves. Moreover, it expects and accommodates changes. RUP believes in addressing high risks areas early in the development process and therefore advocates the development of the system's architecture first [2], [7].

RUP is based on several strength areas of best practices commonly used in industry by successful organisations and therefore, will be valuable to incorporate into the SDF. RUP's six best practices are:

- As Scrum, RUP focuses on an **iterative approach to development**, where requirements are evolved iteratively throughout the development process. Therefore, RUP has the ability to accommodate tactical requirement changes without affecting the development. Furthermore, each iteration ends with a shippable release, which leads to:
  - Development teams being more focused on producing results.
  - Frequent status checks which help ensure that the project stays on schedule [9].





- RUP **manage requirements** by emphasising the importance of proper requirements gathering (stakeholder as well as project requirements), defining of project goals and the documentation thereof as far as possible before commencing with the development.

Requirements management within RUP, requires a collaborative effort and does not leave it up to the product owner as Scrum [6]. Furthermore, RUP provides the necessary guidance for defining requirements through use cases, which is lacking in Scrum.

- It believes in baselining the architecture of the system, prior to committing resources for full-scale development. The **architecture-centric** approach promoted by RUP leads to careful consideration and selection of various applicable architectures components and their flexibility. In Scrum, this definition is looser and depends on guidelines and standards given to the team [6].
- RUP promotes **visual modelling** techniques to capture the structure and behaviour of the architecture and components of the system. RUP believes that through visual models one can improve communication; ensuring the consistency of building blocks across the systems and maintain a close correlation between design and implementation [9]. Therefore, RUP is guided by visuals, many of them in UML.

If combining Scrum with RUP, teams will not be forced to build specific visual artefacts, the need for visualising artefacts and the detail thereof will be dependent on the specific iteration's goals and project.

- RUP believes that **quality management** should be a major part of each and every phase of the project from inception to delivery. Furthermore, at the end of each phase, specified milestones should be met before the development can commence to the next phase within the SDLC. This provides management with the necessary visibility into the development process [2].

Scrum already has built-in quality aspects; teams must demonstrate results at the end of each sprint, functionality is tested, measured, and demonstrated in an iterative fashion. However, in larger organisations where higher quality control procedures are required, like ISO 12207, as in the organisation under study, Scrum would benefit from RUP's quality assurance since RUP provides the necessary guidance in regards to the planning, designing, implementation, execution, and evaluation of quality tests [9].

- RUP believes in a proactive approach in resolving **changing customer requirements** and related risks. RUP describes how to control, track and monitor changes to enable successful iterative development.

RUP can be extended to adopt the principle of Scrum where iterations are not allowed to be interrupted by introducing requirement changes into the current iteration. Requirement changes should rather be captured and introduced at the end of the iteration to be developed in the next iteration. Therefore, ensuring that nothing impedes the development process to deliver the functionality agreed on at the commencement of the iteration [6].

RUP is the ideal methodology to incorporate with Scrum since RUP is serial in the large and iterative in the small and therefore enabling one to apply it to the organisation's global

high-level DP lifecycle and phases while adopting iterative and incremental development within the phases themselves.

In addition, RUP is ideal for the phase management approach advocated by the organisation's global high-level DP since each phase in RUP ends in a well-defined milestone(s) at which stakeholders assess the project progression and the plans forward. Subsequently, a go/no-go decision is made whether to proceed with the project. Moreover, the phases within the RUP are well-defined and very similar to those within the high-level DP. RUP will be adapted where necessary to be more in-line with the global high-level DP. The following figure depicts how Scrum and RUP maps onto each other.

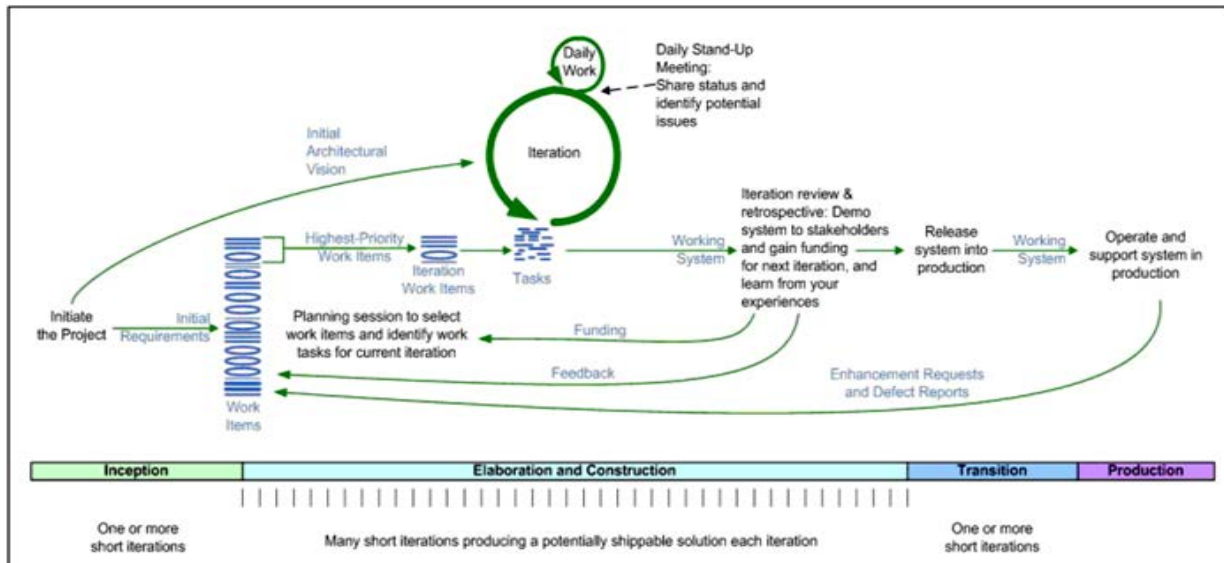


Figure 3: Expanding the Scrum lifecycle by applying RUP [3]

RUP consists of a rich set of software engineering principles which includes disciplines as well as workflows and depicts a semi-ordered sequence of activities to be performed (by specific roles) to achieve particular goals. Therefore, RUP is not generally considered particularly Agile since Agile principles refrain against these extensive guidelines within RUP. What most people don't seem to understand is that the sequence of activities in each of the workflows is based on best practices. It should not be taken as a mandatory sequence. RUP should be tailored to provide an appropriate and customized process for developing software which should be geared towards the project and organisational needs [10]. This implies that RUP can be instantiated anywhere from a very Agile form to a very traditional form or anywhere in between to fit the needs of the organisation.

#### 4.3 Combining RUP and Scrum to form a hybrid process

The four major phases of RUP and their milestones, with some modification, form the framework of the SDF. Scrum provides the necessary project management and tracking mechanisms through its structured ceremonies, roles (scrum master, team & product owner), and artefacts (product backlog, sprint backlog, and burn down chart) while RUP provides the engineering practices required. The various RUP disciplines are applied in the various iterations throughout each phase of the SDF. The daily Scrum meetings, the Sprint planning meetings, and the Sprint review meetings are also incorporated into the SDF and utilised to monitor the adherence of the process to the six fundamental RUP disciplines throughout the process.



Although the SDF complies with many of RUP's principles, it simultaneously complies to various Scrum principles, especially the Development phase since it is directly adopted from Scrum. Development within the SDF occurs in iterations and is directly aligned with Scrum.

The SDF also adopted the product backlog and sprint backlog principles of Scrum but with a RUP spin to it since it makes use of use cases to produce Scrum's user stories and include non-functional, architectural and technical requirements. The SDF refers to the product backlog as the high-level Software Requirement Specification (SRS). The high-level SRS is used for iteration planning and can be seen as a growing document since the necessary analysis is performed to fill in the detail that is needed for the development of that iteration, during the iteration in question. This detailed SRS is then equivalent to a sprint backlog. The rest of the artefacts produced throughout the SDF are more in-line with those recommended by the analysis and design, implementation and test disciplines of RUP.

Like scrum, the SDF does not believe in big up-front requirements, but supports design, analysis and requirement elicitation being done upfront to establish at least the architecture of the system before starting with development and fill the gaps as needed (as Scrum).

#### **4.3.1 SDF Phases**

The following section depicts how the SDF's phases align with those of RUP and Scrum. It also indicates what of the disciplines it adopted from RUP and how it is applied throughout the process.

##### **4.3.1.1 SDF Feasibility Phase**

The Feasibility phase of the SDF is mostly focused on developing and evolving the business case and customer vision. Scrum, RUP and the SDF require that a customer's vision of the system is needed before the project can start. The vision is vague at first, stated in market terms rather than system terms but is elaborated on as the vision becomes clearer as the project progresses.

As part of the business case very high-level requirements and project plan are included, but do not form a substantial part of the analysis in this phase. This phase does however include a risk analysis of taking the project on. Subsequently, this phase of the SDF corresponds with the RUP's Inception phase and Scrum Pre-game phase where the product backlog gets produced along with the high-level architect.

##### **4.3.1.2 SDF High-level planning and design Phase**

As with the Inception and Elaboration phases of RUP, this phase is more dedicated to the Requirements, Analysis and Design disciplines of RUP since the focus of this phase is to produce:

- The necessary use case models and the business models.  
The business modelling discipline of the RUP's Inception phase is executed during this phase of the SDF, as an elaboration on the business case produced during the Feasibility phase.
- More elaborated requirements for the product backlog (including non-functional requirements).



To adhere to the Scrum methodology, this phase aims to produce requirements that are only 60% complete. The requirements will be further elaborated during the iterations that they are implemented in.

- The high-level software architect design  
During this phase the aim is to define a sound architectural foundation:
  - That defines the composition of the major components and ensures that the development of the different components of the system is developed in-line with each other.
  - Upon which the first few iterations of development can be based.
  - To form the common technical focus throughout all iterations.

The detail of the architect can be filled in as needed during the iterative development.

This phase also focuses on producing a project plan, showing the necessary phases (with their milestones) and iterations. The project plan also indicates which requirements are developed during which iteration. Included in the project plan will also be a development case specifying which practices of the SDF will be used during the development and which will be omitted.

#### ***4.3.1.3 SDF Development Phase***

The Development phase is based on an iterative development cycle. An iteration includes the following activities:

- Elaboration on the initial requirements and design stipulated in the product backlog as well as the high-level architectural design
- Development/ Construction
- Testing
- Sprint Review

Therefore, the analysis and design, implementation and testing disciplines of RUP are dominant during the development phase.

The development phase is only focused on the internal validation and verification of the software product/ system. Customer validation and verification of the system only takes place in the next phase of the methodology. Therefore the end of the development phase is signified by an assessment of the deployment baseline against the complete vision and the acceptance criteria for the product.

#### ***4.3.1.4 SDF Customer Validation and Stabilisation Phase***

This phase is focused on validating and verifying the new system against user expectations and requirements. The new system is deployed within the customer's environment and is tested/ validated by the customer.

This phase focuses on attaining stakeholders' agreement that the deployment baselines are complete and are consistent with the evaluation criteria of the vision and can be deployed to the live environment. Therefore, this phase includes:

- Fine-tuning the product/system by engaging in bug fixing, enhancements to the performance and usability of the system identified during customer acceptance testing.
- Ensuring that all software and supporting documentation are acceptable, stable and mature to deploy.

The end of this phase is signified by the agreement of all stakeholders (and the business) that the system is ready to be deployed into the customer's live environment.

#### 4.3.1.5 SDF Commissioning Phase

This phase corresponds to the RUP's Transition phase and Scrums post-game phase. During this phase the system gets deployed to the user's live environment and/ or roll out to marketing, distribution, and sales teams. At this stage the new system might be in operation with a parallel legacy system that it is to replace, to ensure that the system is stable before switching over. Training of users and maintainers takes place simultaneously.

#### 4.3.2 SDF iterations and RUP disciplines

All workflows are executed through one or multiple iterations throughout the various phases of the SDF. Each iteration has its own lifecycle which is composed of the various RUP disciplines; each discipline forms a phase within the iteration's lifecycle. An iteration's lifecycle is comprised of:

- A planning phase which is constituted from RUP's Business Modelling and Requirements disciplines.
- A development phase which is mostly constituted from RUP's Implementation discipline but also contains a section of the deployment discipline.
- A testing phase which draws from RUP's testing discipline.
- A review phase which draws from Scrum's project management discipline.

Depending on the phase within the SDF, iterations might focus on some disciplines more than on others as depicted by the figure below.

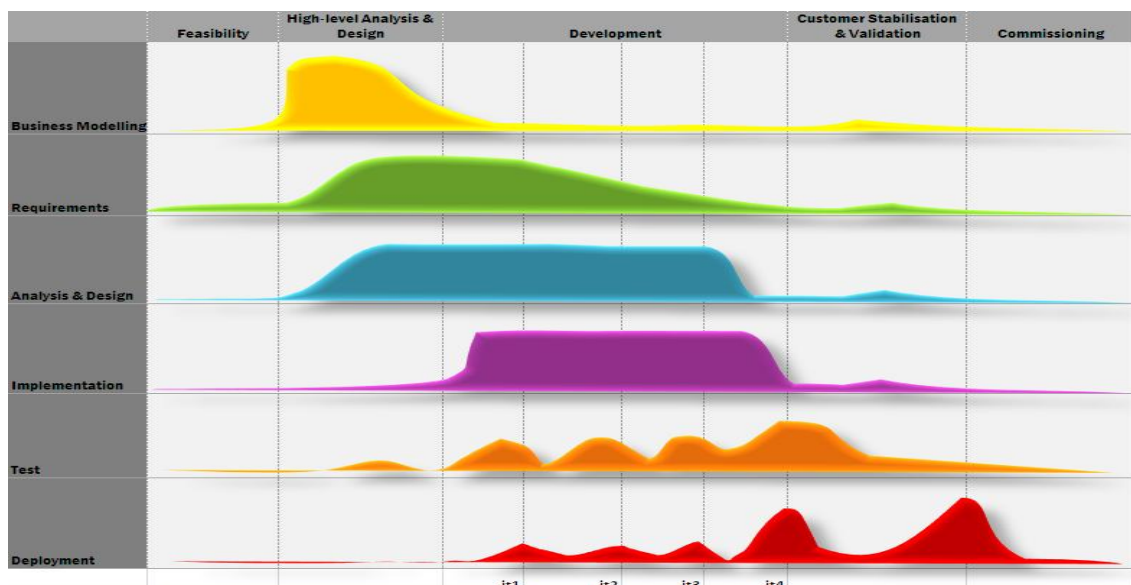


Figure 4: The "Hump chart" of the Hybrid SDF's Phases and Disciplines



## 5. HOW DID THE SDF OVERCOME THE ORGANISATION'S PROBLEMS

The following section depicts how the problems that were faced by the organisation were resolved through the SDF as well as other problems that were encountered during the development of the process.

### 5.1 Not able to meet Deadlines & Budgets

This problem was overcome by adopting Agile methods which by nature address these problems through:

- Treating cost and schedule as fixed constraints opposed to treating requirements as the scaling factor in the equation.
- Ensuring that any problems and issues were resolved early within the development cycle through the sprint review and retrospective meetings.

### 5.2 Uncertain and unclear requirements in the beginning of projects

Once again, the problem was addressed by adopting Agile methods which by nature address these problems through applying iterative and incremental development. Moreover, requirements are elaborated on and defined in detail per iteration as development progresses. Therefore, agile provides customers with the opportunity to learn more about their requirements throughout the development of the project and refine their requirements accordingly.

Furthermore, the SDF applied practices to ensure that a better understanding of the business' problems is obtained before commencing with development.

### 5.3 Outsourced development

Extra checkpoints were built into the practices of the SDF to ensure the controls between the 3<sup>rd</sup> party developers and the organisation. In the framework these are documented as "document sign-offs" but these controls can also be achieved through Scrum principles e.g. demos and sprint planning meetings. These checkpoints within the SDF are focused on:

- Obtaining buy-in and sign-off of all internal stakeholders (the project team) into the Software Requirement Spec (SRS). Thereby, ensuring that everything in the SRS is correct and that the SRS is aligned with the architect and various system operations. This checkpoint can be achieved through a sprint planning meeting where the detailed requirements can be discussed and any problem highlighted/resolved.
- Quality control whereby each software release is verified and validated to ensure that it would not interrupt the testing cycle. This is achieved by the lead developer validating the depth and sufficiency of the unit tests performed during development. When 3<sup>rd</sup> party developers are responsible for development, this checkpoint can be achieved by a sprint demo by the 3<sup>rd</sup> party developers to the rest of the project team.

### 5.4 Maturity level of the development department within the organisation

Due to the immaturity of the development team extra checkpoints, called sub-phase checkpoints were built into the practices of the framework. Effectively these checkpoints can be seen as a format of peer reviews on documents, code produced and test cases. Moreover, these peer reviews can be seen as "pair-programming" since the principle

behind it is to ensure quality as with pair-programming. If the development team involved in the project is mature, these checkpoints can be omitted from the process.

## 5.5 Daily Operations

This issue was addressed by getting the Operations team involve earlier in the lifecycle of the SDF and assigning the following practices to them:

- They are responsible for setting-up testing environments and deploying the system within these environments for validation by the product owner as well as for User acceptance testing (UAT) by the customer. Subsequently, ensuring that they obtain the necessary “know-how” before the official live deployment. Therefore, reducing Testing’s involvement in these practices.
- As part of the close-out of each project, the appropriate knowledge transfer on the technical operation of the system needs to transpire between the development team and the relevant operations support team to ensure that the operations support team will be able to support the product/system going forward. The project will not be able to be closed-off until this knowledge transfer was successful.

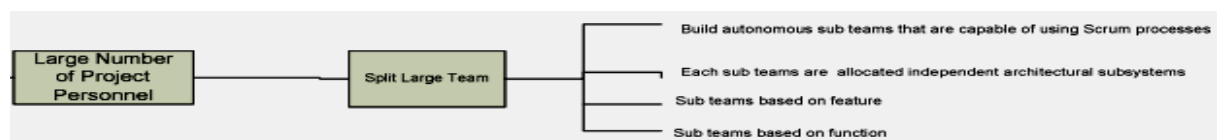
## 5.6 Addressing the Scaling Factors faced by the organisation

Now that all the organisation’s problems are addressed and resolved through the practices within the SDF there is still the outstanding issue to turn the SDF into a Disciplined Agile delivering lifecycle. This is accomplished by scaling various Scrum principles to be more suited for large projects as well when needed.

### 5.6.1 Scrum and large projects

One approach is to divide large projects into teams of teams. Each team is then comprised of various members from the multiple project lines. Another approach is to let members of each project team attend daily Scrum meetings to facilitate a widespread understanding of the project as a whole. It is furthermore recommended that one core team should be responsible for the architecture and standards of the system across the multiple teams therefore, ensuring that all software components adhere to the central architecture and standards [8].

The following diagram (taken from a Conceptual Framework by Hossain, Babar, Paik and Verner [5]) depicts possibilities of how teams can be divided.



### 5.6.2 Scrum and Distributed Teams/ Outsourcing

The following practices can be applied when development is outsourced or for Programs which exist of multiple projects:

- Scrum meetings  
Daily scrum meetings are held by each team independently. The results of these meetings can be posted on a wiki. However, if teams work on pieces of the system which are closely correlated, daily scrum meetings between the teams should rather be arranged via telephone conferencing and application sharing [8].



A scrum-of-scrums meeting should be held once a week. During this meeting the three scrum questions are answered but from the various teams' perspectives. Two additional questions are added to the list: "Have your team put some impediments in the other teams' way?" and "Does your team plan to put any impediments in the other teams' way?" [8].

- Synchronized sprints

It is recommended to keep sprints synchronized across the various teams where possible. At the end of a sprint, the teams can deploy to a collaborative environment for collaborative sprint demonstrations. However, if the software components developed by the various teams is not strongly correlated, the sprint lengths can be varied among the teams according to what makes sense within the project environment and scope.

- Distributed sprint planning meetings

The sprint planning meetings are divided into distributed meetings and local meetings. The objective of these meetings is to review the Sprint backlog and segregate the work amongst the teams. The product backlog should be prioritised before the meeting and the items should have preliminary estimates assigned to them. Furthermore, it is recommended that strongly correlated user stories should be given to one team rather than splitting it across teams. Once the distributed meeting is completed, the teams separate and have their own local meetings to divide the backlog items into more detailed tasks and adjust the estimates made by the product owner [8].

- Separate backlogs for each team

Backlogs should be managed via SharePoint that all team members have access to, but only product owners can update. There is one central, consolidated and prioritized product backlog which is managed by the chief product owner of the teams. This centralised product backlog is then split into various sprint backlogs for each team which are updated by the respective product owners. The chief product owner meets regularly with all product owners to assure coordination of all requirements and that the central product backlog is always up to date [8], [11].

- Shared Electronic Work Spaces

With a distributed team, there is a need for a shared workspace for better visibility into e.g. project status, product backlog item statuses, Burndown charts, updates on documents and the product backlog, etc. Therefore, it is important to ensure that a shared electronic workspace is set-up before the project/program kicks-off. Therefore, it is suggested that the first activity on distributed projects should be aimed at:

- Establishing how communication in the project should be handled e.g. that all development issues will be discussed on the project wiki.
- What tools will be used.
- How a shared electronic workspace will be attained [8].

### **5.6.3 Organisational Complexity**

This scaling factor was resolved by applying RUP principles, phases and practices to the SDF since these allow outsourcing and adhere to the high-level DP as explained throughout this article.





## 6. CONCLUSION

SDF is portrayed as a methodology/guideline/framework to be followed by all projects within the organisation. It lists both mandatory and optional phases and artefacts as a roadmap to success. The SDF should be thought of as a baseline for tailoring practices to meet a specific project's needs. Dependant on the project type or size, the project might follow the SDF from the Feasibility phase through to the Commissioning phase or it might only follow certain sections of the SDF.

Furthermore, to provide the project team with the freedom to act as an independent unit, all the phases and artefacts that can possibly be produced within the SDF is seen as "optional" and the project team should determine the strategy and approach that they are planning to follow for each specific project as part of the project plan upfront. A quick risk analysis should be performed before omitting any sections/artefacts of the SDF to ensure that the process itself is not side-stepped or artefacts not produce in the name of Agile. Therefore, the SDF can be seen as an appropriate governance framework within which the project team can organise themselves.

The framework does not prescribe the precise contents of the artefacts but rather provide a guideline for what these documents should address; it allows the project team the necessary freedom to tailor these documents as needed by their specific project. One of the most important artefacts is the high-level and detailed Software Requirement Specification (SRS) due to the fact that the high-level SRS is seen as the product backlog within the framework. Therefore, the high-level SRS's contents are the only artefact within the framework that is prescribed since it applies RUP's principles of writing "user stories" as use cases.

It is advised that during the tailoring process of the practices and artefacts/models, that the following principles and practices of Agile modelling should be followed:

- Model with a purpose: If you cannot identify why you are creating a specific document or any models within the documentation rather omit it.
- Apply modelling standards: The team should agree in the beginning of the project on a common set of modelling standards on the software project and follow it throughout the project.
- Create simple content: Do not add additional aspects to artefacts unless they are justifiable.
- Depict models simply: Create a simple model to portray the key features under investigation.
- Model in small increments: Model a little, code a little, test a little and deliver a little.
- Only model if
  - A model is used to communicate key aspects of the system to be built.
  - A model is used to understand the system being built, to consider approaches and choose the best one.
  - The model you are planning to use is the most appropriate one for the situation.
- Only update a model or artefact when absolutely needed or if by not doing so it will be detrimental to the team/project later on.
- Use the simplest tool possible to achieve the desired outcome [12].

## 7. REFERENCES

- [1] Ambler, S. 2009. The Agile Scaling Model (ASM): Adapting Agile Methods for Complex Environments. [Online] Available: [https://www.ibm.com/developerworks/mydeveloperworks/blogs/ambler/entry/agile\\_scaling\\_model?lang=en](https://www.ibm.com/developerworks/mydeveloperworks/blogs/ambler/entry/agile_scaling_model?lang=en). [1 July 2011]
- [2] Ambler, S. 2005. A Manager's Introduction to the Rational Unified Process (RUP). [Online] Available: [www.agilemodelling.com](http://www.agilemodelling.com). [1 July 2011]
- [3] Ambler, S. 2010. *Scaling Agile: An Executive Guide*. [Online] Available: [https://www.ibm.com/developerworks/mydeveloperworks/blogs/ambler/entry/scaling\\_agile\\_an\\_executive\\_guide10?lang=en](https://www.ibm.com/developerworks/mydeveloperworks/blogs/ambler/entry/scaling_agile_an_executive_guide10?lang=en). [1 July 2011]
- [4] Deemer, P, Benefiel, G, Larman, C and Vodde, B. The Scrum Primer Version 1.2. [Online] Available: <http://www.scrumalliance.org/resources/339>. [July 2011]
- [5] Hossain, E, Babar, M, Paik, H and Verner, J. 2009. Risk Identification and Mitigation Processes for Using Scrum in Global Software Development: A Conceptual Framework, *16th Asia-Pacific Software Engineering Conference, 2009*, pp 457-464. IEEE Computer Society.
- [6] Krebs, J. 2005, RUP in the dialogue with Scrum. [Online] Available: <http://www.ibm.com/developerworks/rational/library/feb05/krebs/>. [1 July 2011]
- [7] Lines, M, Barnes, J, Holmes, J and Ambler, S. 2008. Agile Rational Unified Process: RUP experiences from the trenches. [Online] Available: [http://www.ibm.com/developerworks/rational/library/edge/08/feb08/lines\\_barnes\\_holmes\\_ambler/index.html](http://www.ibm.com/developerworks/rational/library/edge/08/feb08/lines_barnes_holmes_ambler/index.html). [1 July 2011]
- [8] Paasivaara, M, Durasiewicz, S and Lassenius, C. 2008. Distributed Agile Development: Using Scrum in a Large Project, *IEEE International Conference on Global Software Engineering, 2008*, pp 88-95. (33). IEEE Computer Society.
- [9] Rational Unified Process: Best Practises for Software Development Teams, A *Rational Software White Paper*. [Online] Available: [http://user.it.uu.se/~hessel/project/RUP\\_bestpractices.pdf](http://user.it.uu.se/~hessel/project/RUP_bestpractices.pdf). [1 July 2011]
- [10] Shuja, A. *Chapter 1: Welcome to the IBM Rational Unified Process and Certification*. IBM Rational Unified Process v7.0. [Online] Available: [http://ptgmedia.pearsoncmg.com/images/9780131562929/samplechapter/0131562924\\_01.pdf](http://ptgmedia.pearsoncmg.com/images/9780131562929/samplechapter/0131562924_01.pdf). [1 July 2011]
- [11] Sutherland, J, Viktorov, A, Blount, J and Puntikov, N. 2007. Distributed Scrum: Agile Project Management with *Outsourced Development Teams, Proceedings of the 40th Annual Hawaii International Conference on System Sciences IEEE Computer Society, 2007*. Hawaii: IEEE Computer Society
- [12] Cohen, D, Lindvall, M and Costa, P. 2003. Agile Software Development: A DACS State-of-the-Art Report. [Online] Available: [http://www.google.co.za/url?sa=t&source=web&cd=1&ved=0CByQFjAA&url=http%3A%2F%2Fciteseerx.ist.psu.edu%2Fviewdoc%2Fdownload%3Fdoi%3D10.1.1.94.7%26rep%3Drep1%26type%3Dpdf&rct=j&q=\[12\]%09David%20Cohen%2C%20D%2C%20Lindvall%2C%20M%20and%20Costa%2C%20P.%202003.%20Agile%20Software%20Development%3A%20A%20DACs%20State-of-the-Art%20Report.%20&ei=OlwkTufFL6j0mAXPw6icAw&usq=AFQjCNES6h9VbExJrwXOq6alW2Q7gcARIQ&sig2=1yDOzYefuFs-hMvy5i3n0g&cad=rja](http://www.google.co.za/url?sa=t&source=web&cd=1&ved=0CByQFjAA&url=http%3A%2F%2Fciteseerx.ist.psu.edu%2Fviewdoc%2Fdownload%3Fdoi%3D10.1.1.94.7%26rep%3Drep1%26type%3Dpdf&rct=j&q=[12]%09David%20Cohen%2C%20D%2C%20Lindvall%2C%20M%20and%20Costa%2C%20P.%202003.%20Agile%20Software%20Development%3A%20A%20DACs%20State-of-the-Art%20Report.%20&ei=OlwkTufFL6j0mAXPw6icAw&usq=AFQjCNES6h9VbExJrwXOq6alW2Q7gcARIQ&sig2=1yDOzYefuFs-hMvy5i3n0g&cad=rja). [1 July 2011]
- [13] Ambler, S. 2006-2009. *Examining the "Big Requirements Up Front (BRUF) Approach"*. [Online] Available: [www.agilemodelling.com](http://www.agilemodelling.com). [1 July 2011]
- [14] Gladwell, M. 2009. *Outliers, Limited (UK) Penguin Books*.
- [15] Turk, D, France, R and Rumpe, B. 2003. *Revised and Resubmitted as a Research Review Paper*, *Journal of Database Management*, May, pp 16 -18.