

Practical experience in the successful delivery of large scale software based systems

Francois Retief

FALK Systems Engineering and Consulting

30 De Waterbosch, Curlew Way, Somerset West, 7130, South Africa

082 328 6543 francois@falk-se.com

Copyright © 2008 by Francois Retief. Published and used by INCOSE SA with permission.

“Software is a place where dreams are planted and nightmares harvested, an abstract, mystical swamp where terrible demons compete with magical panaceas, a world of werewolves and silver bullets.”

Brad J. Cox

Abstract. According to the findings of the Standish Chaos Report based on data of more than 50,000 software projects, most software projects are still seriously challenged or fail completely. Yet there are groups that consistently deliver on time, within budget, and manage to keep customers and management happy. This paper presents an overview of the process elements that has been found to ensure the desired predictability, visibility and stakeholder satisfaction in telematics systems development - a large, complex software development environment. It includes an introduction to the agile and iterative software development practices that form the basis of the approach followed. Project phases, team composition, levels of responsibility and involvement at various stages are discussed. Tools and artifacts to improve communication and efficiency are presented, combining theory with practical experience.

The challenges of telematics systems development

“There is no silver bullet.”

Frederick P. Brooks

Or in the words of Ad Sparrus: “Tailor your process!” It is important to understand the environment for which the processes discussed in this paper were adopted and developed. Besides shedding light on the reasons behind process modifications, it also bears relevance to the applicability of the process to other environments.

The standard Telematics System consists of a mobile device transmitting positional and telemetry data to a software back-end system. More than one type of communication and positioning system is often used, based on least cost or best quality of service. Besides providing positional data, the mobile device also frequently interfaces to a number of external modules and sensors. Figure 1 shows a typical instance of a telematics system.

Back-end servers then distribute the vehicle information to various users via a combination of thick desktop clients, smart clients and web clients. Data is also normally available to a monitoring bureau that is typically responsible for stolen vehicle recovery, fleet management reporting and services, customer service, system health management and mobile support.

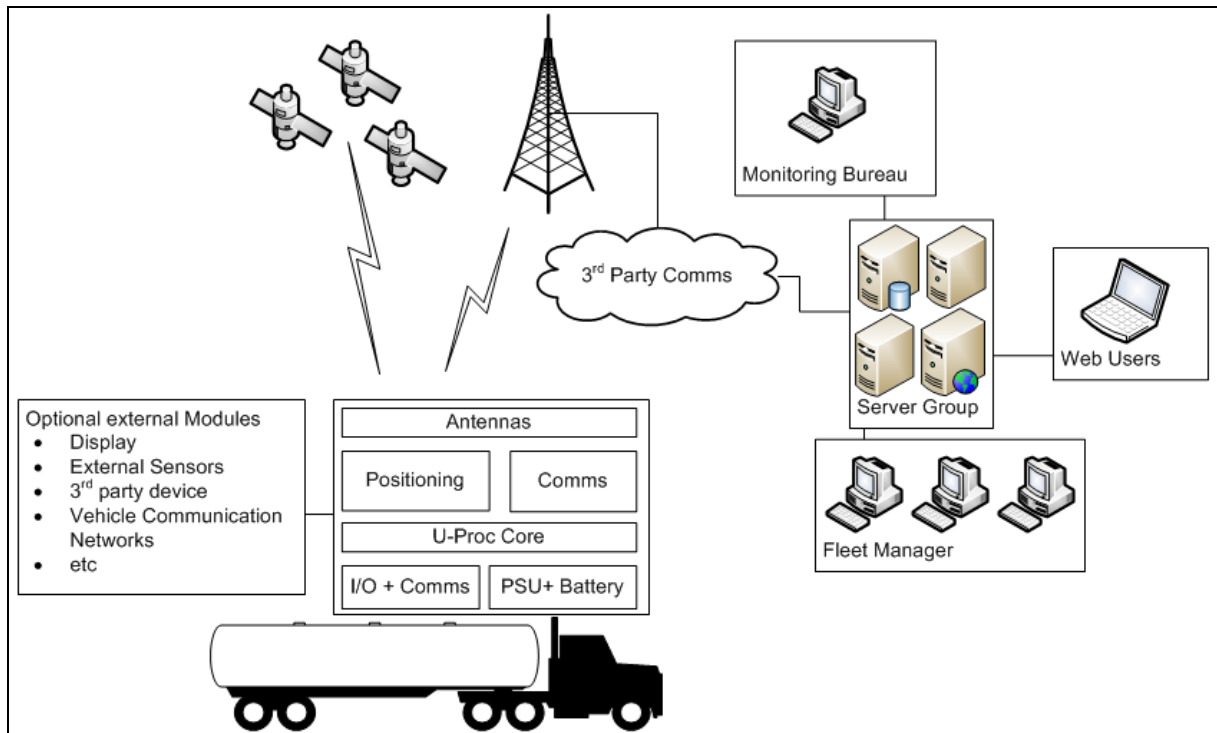


Figure 1. Typical Telematics System (Single Instance)

The trivial case is however complicated in that a typical instance of the system normally consists of 200 or more customers, i.e. fleet operators, resulting in between 10,000 and 100,000 vehicles in total on a single system, and a bureau with up to 20 full time operators.

In addition, there are normally a number of these deployments in key geographical regions around the world, typically Africa and the Middle East, Europe, South East Asia, Australia, North America and South America, depending on how widely the system is sold.

This single, generic solution is normally developed and maintained by a team of 15 to 35, consisting of Software, Firmware and Hardware Engineers, Component and Systems Testers, Systems Engineers and Architects, Team Leads and Managers. It is their responsibility to provide regular system releases meeting the needs of all the regions around the world, where each region deals with multiple divergent customer requests, tenders, contracts and identified strategic opportunities.

Although the system consists of custom developed hardware, external and third party supplied components and a number of interfaces it soon becomes apparent in practice that the bulk of the challenges in meeting stakeholder requirements revolves around the software development.

Careful, it is not as easy as it seems

“The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time.”

Tom Cargill

It is often said that 80% of software projects are late and over budget. This general perception is well supported by the Chaos Report. This research project, conducted by the

Standish Group is probably the best quoted with regards to the difficulty and risks related to software development. It is the source of now familiar statistics such as “45% of developed features are never used” or “84% of projects fail or are significantly challenged.” The Standish Group’s SURF database containing the source data holds information on over 50,000 projects from around the world since 1994. (Hartmann 2006).

Summary of the 2006 Chaos report findings. Figure 2 shows that for the 2006 data only 35% of software projects were successful. 46% did not meet the initial aims but was reworked or accepted as is, and 19% of projects completely failed.

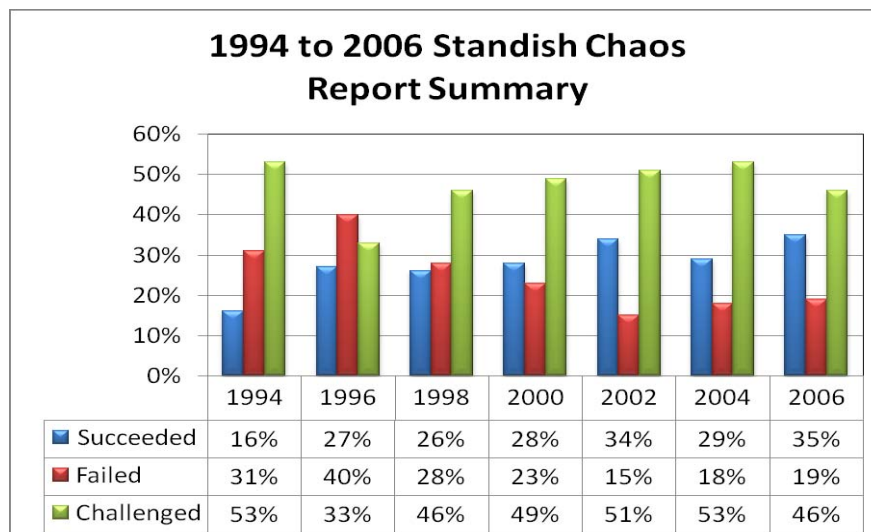


Figure 2. CHAOS 2004- 1994 to 1995 Resolution Statistics

Somewhat more sobering, it also shows that little progress has been made over the previous 10 years improving the statistics. This brings us to the more relevant question of what elements contributed towards successful software projects. The top 10 reasons for success in decreasing order of importance are as follows (Hartmann 2006):

1. User involvement
2. Executive management support
3. Clear business objectives
4. Optimising project scope
5. Agile process
6. Project management experience
7. Financial management
8. Skilled resources
9. Formal methodology
10. Standard tools and infrastructure

Motivation for agile and iterative development. Interestingly, skilled resources and formal methodology only rates a distant 8th and 9th on the list. Of much greater importance are user involvement, executive management support, clear business objectives, optimised scope and an agile development methodology.

Time boxed *iterative and agile software development methodologies* that started to evolve in the mid 1990’s incorporate these items. In the remaining two sections a brief theoretical introduction into agile software development will be provided and practical experience shared.

The power of agility

“Successful software always gets changed.”

Frederick P. Brooks

Waterfall vs Iterative development. The traditional waterfall model shown in Figure 3 is one of the original and most widely used methodologies in engineering and development. Its success is to a large extent based on the fact that it reduces risk and saves large sums of money by investing the relatively cheap design effort upfront, minimising the amount of expensive production and operational rework time. This is due to the cost of a change increasing exponentially as the project progresses.

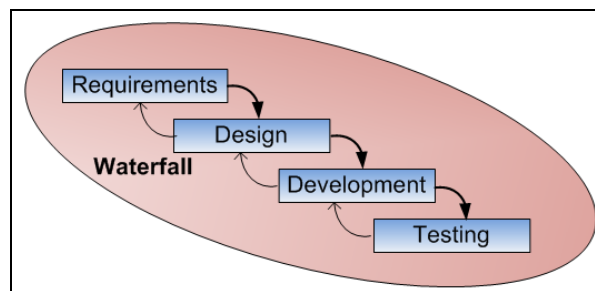


Figure 3. Waterfall Model

Software development fundamentally differs from hardware development. When a fault is discovered on the mobile device hardware, the designs and production data packs have to be changed. Following that, a huge amount of rework needs to be done to existing hardware. In some cases existing stock must be scrapped. Even without recalls from the field the costs and time impact is astronomical. Just imagine the impact of a requirement or design fault made while building a skyscraper, dam or bridge!

For software however the cost of change is very low. One can change a number of lines of code and within seconds recompile a modification for free. This opens up a number of interesting possibilities in software development methodology. The first being that, because of the ease of modularisation, an iterative approach can be followed where the development is broken up into a number of complete sub-projects as shown in Figure 4.

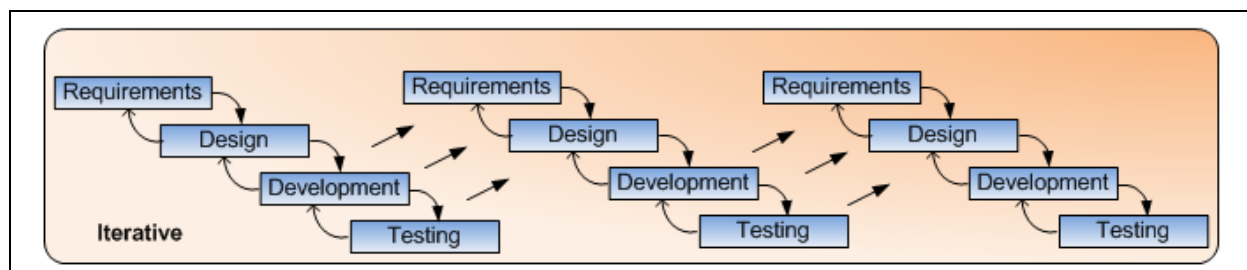


Figure 4. Iterative Development

Immediate apparent advantages are:

- Providing feedback up the chain is a lot easier and happens sooner
- Integration risk is greatly reduced
- It is significantly easier to adapt to changing requirements or needs

Iterative is not agile. A common mistake companies interested in adopting an agile approach makes is to think it is about transitioning from a waterfall to an iterative development model. While agile development is certainly iterative in nature there is a lot more to being agile than just being iterative.

There are a number of agile processes to choose from for example: SCRUM, eXtreme Programming, Feature Driven Development, Lean Software Development and Crystal. Central to all of them is the *Agile Manifesto* which states the following (Larman 2004):

- *Individuals and interaction over process and tools,*
- *Working software over comprehensive documentation,*
- *Customer collaboration over contract negotiation and*
- *Responding to change over following a plan.*

Not for a moment is the value of the statements on the right dismissed. But while the statements on the right are important, the ones on the left are *more important*. Understanding and following these values lead to a number of *key differences* from more traditional development methodologies including:

- Rather than closely managing resources a clear product vision is maintained and communicated and the teams then provided with the correct supporting structure to achieve the set goal. A lot of emphasis is placed on small, self organising teams with cross functional skills. These teams are given a huge amount of autonomy, allowing them to decide how they want to tackle their challenges.
- Documentation, especially for upfront design becomes a lot less important. The teams focus on providing working functions through vertical feature slices. Diagrams and artefacts are used to document design decisions and capture critical information for future reference, but these are merely supporting in nature.
- Similarly, the customer is expected to form an integral part of the development process. As a result, less emphasis is placed on getting him to understand his complete requirement before development starts and tying this down in a contract. In contrast he follows the development, seeing his product unfold and making choices as a part of the team throughout the development process.
- Because we live in a rapidly changing world and a lot of new possibilities and options become apparent during the development process, the team spends less time developing a comprehensive plan upfront and then following it. There is still a very clear vision, but this is used more for direction than for indicating the ultimate destination. At regular intervals in the process the vision, aims and direction are re-evaluated and changed as needed. This also implies a lot less up-front design, with decisions rather being made later when a clearer picture of the options emerges.

An agile process overview. A process overview is shown in Figure 5 (Van der Merwe 2006). A full, prioritised and roughly estimated feature list is maintained. Features in this list are prioritised based on business value (benefit versus cost) and risk. Development is done in a set of time boxed iterations with a duration of between 1 and 6 weeks. At the start of the iteration the development team selects enough features in order of priority to fill the available time. These are then planned and implemented. The aim of the release is to demonstrate the work done as best as possible to all stakeholders. In the ideal case the output of an iteration will be a subset of features that are completely integrated, tested and ready to be deployed commercially. On large system developments this is however not always feasible. Regardless of the complexity, the aim remains to deliver vertical functional feature slices as close as possible to the final version.

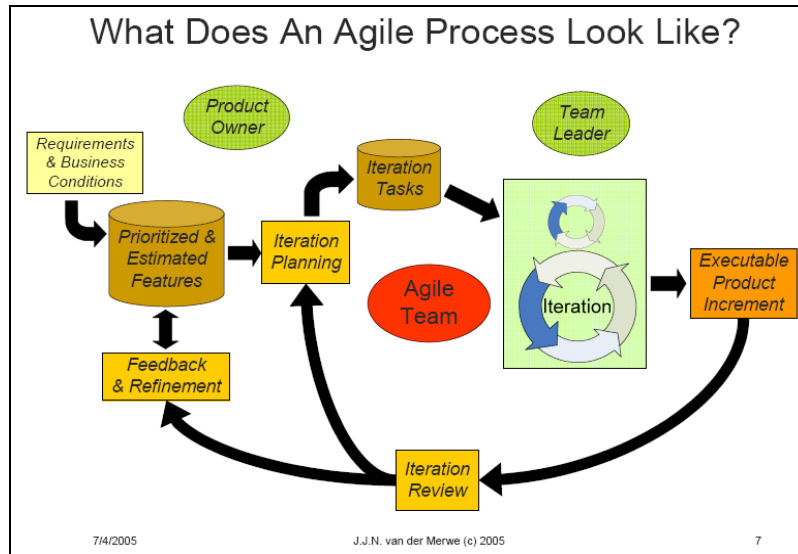


Figure 5. Basic Agile Development Process Overview

After every iteration the result is evaluated by all the stakeholders and the planning updated to incorporate what has been learnt in the iteration. The team also evaluates and improves their processes. The feature list is updated, re-estimated and prioritized, and the goals and features for the next iteration are decided.

Agile in large integrated projects. In a large and integrated project the Unified Process framework (Kruchten 2001) can be used where a number of iterations, consisting of all the necessary disciplines are executed. See Figure 6 for a representation of this model. Throughout the project the focus for an iteration moves from business modeling through requirement analysis, design, implementation testing and deployment but with each of these activities still performed in every iteration.

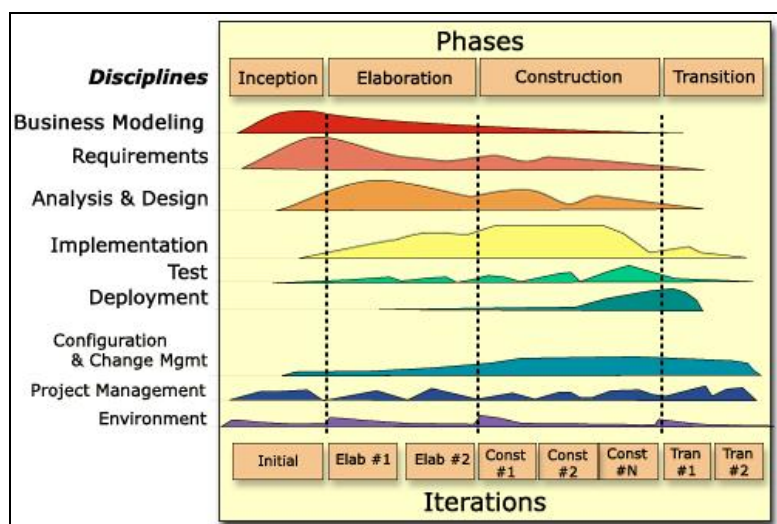


Figure 6. Rational Unified Process Overview

What about long term planning? One of the first concerns to surface while evaluating the agile process tends to be “What about planning and design?” The answer is that planning is refined continuously. At the start of the project a rough estimate is done. This allows the

team to start working quickly against tracked goals. After every iteration this end goal is re-estimated, updated, tracked and managed to provide the best value proposition to all stakeholders. The final effect is less like a canon being fired and more like a missile with adapting guidance.

Agile practices summary. Agile practices falls mostly into Management Practices and Programming Practices (Van der Merwe 2006).

Management Practices include requirement analysis, planning and project management and contains the following practices:

- Involve the customer closely
- Iterate regularly and early
- Agile practices for feature estimation and planning
- Agile practices for iteration and release planning
- Tracking the team velocity and
- Using agile tools to track progress.

Programming practices contain the following:

- Test driven design (Designing and writing tests before coding starts)
- Simple and clean design (Each iteration ends with a design that is as good as is needed for the features in that design, no more but also no less)
- Continuous refactoring of code when problems are detected, both in the normal run of things as well as through scheduled tasks
- Domain driven design

The agile value proposition. Figure 7 shows the advantages of agile software development over the traditional waterfall model.

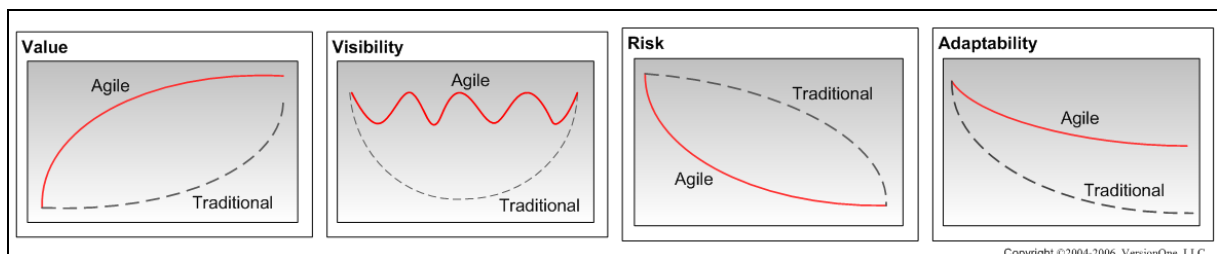


Figure 7. The Agile Value Proposition

Following agile practices provides higher value to the customer from an early stage due to the fact that complete features are being developed and that the product can be released earlier. It also provides better end value through better focus and adaptation to changing customer requirements. Visibility remains higher through the regular iterations. Risk, especially integration and requirement risk is also driven down much faster and provides longer term adaptability.

Practical experience in the development of telematics and other complex software systems

“Good judgement comes from experience, and experience comes from bad judgement.”

Frederick P. Brooks

This section describes some of the principles and tools that have proven successful in the

development and delivery of complex software systems in general and telematics systems in particular. Following is an account of the author's experience:

“When I started working in the telematics environment my company were on the verge of finishing version 1.0 of their product. They followed a ‘cowboy’ approach that enabled them to get a product to market quickly and cost effectively. One of my first tasks was to qualify the first version of the mobile device *over a weekend!* Development ended late Friday and production had to start on Monday if the deadlines were to be met.

The challenges associated with the rest of the product life cycle quickly became apparent. Units had to be reworked in the field. Changing Flash chips on hidden installations in trucks in the field rapidly became very costly. The software team had to implement a very complex and marginally effective workaround because a protocol element that was developed on the mobile unit and subsequently produced in volume could not be scaled. Qualification cycles exploded and timelines were unpredictable.

I helped implement a traditional waterfall development model based on strong systems engineering principles. This provided more certain timelines, reduced rework and improved implementation efficiency through unit and interface specifications. Unfortunately it also increased timelines. It proved difficult to do continuous incremental development and because the planning became more complex hardware and firmware development were often triggered early on risk. Running waterfall phases concurrently improved feedback between the different phases but increased the risk that decisions were left open too long and also still delayed qualification (and user input) until right at the end of the project requiring huge risk contingency in project plans.

Then I was presented with the challenge to help found a new telematics company. The challenge was to, within 9 months, expand a pure firmware/hardware team to include software development, systems engineering and testing and to release a first version of a product that had to compete with companies that have been working on solutions for more than five years. The company succeeded in this goal to a large extent through the application of the agile principles described in this paper. The iterative and agile principles maximised business value of the releases, enabling effective and efficient communication between all stakeholders. It created unity of purpose and commitment throughout the organisation, and provided an efficient tool for accurate estimation and progress tracking.

Subsequently, I was involved in a third telematics company with unreliable software development estimation. In one case a 3 month software development project eventually took 15 months to complete. Some projects were on schedule, but results were highly unpredictable. I assisted in implementing an agile planning paradigm that repeatedly reduced software development overrun to at most weeks.“

High level process overview. A telematics system is a complex but single product competing against similar products in a rapidly expanding market. It requires regular releases of new functionality. The optimum balance between frequency and releasability appears to be between 2 and 4 system releases a year. A high level process as indicated in Figure 8 was developed.

The model borrows from the Rational Unified Process where all key activities are performed throughout the development but with emphasis on certain activities at a specific point in the process (See Figure 8). Within these high level project phases one or more SCRUM based sprints are executed, resulting in complete and integrated deliverables.

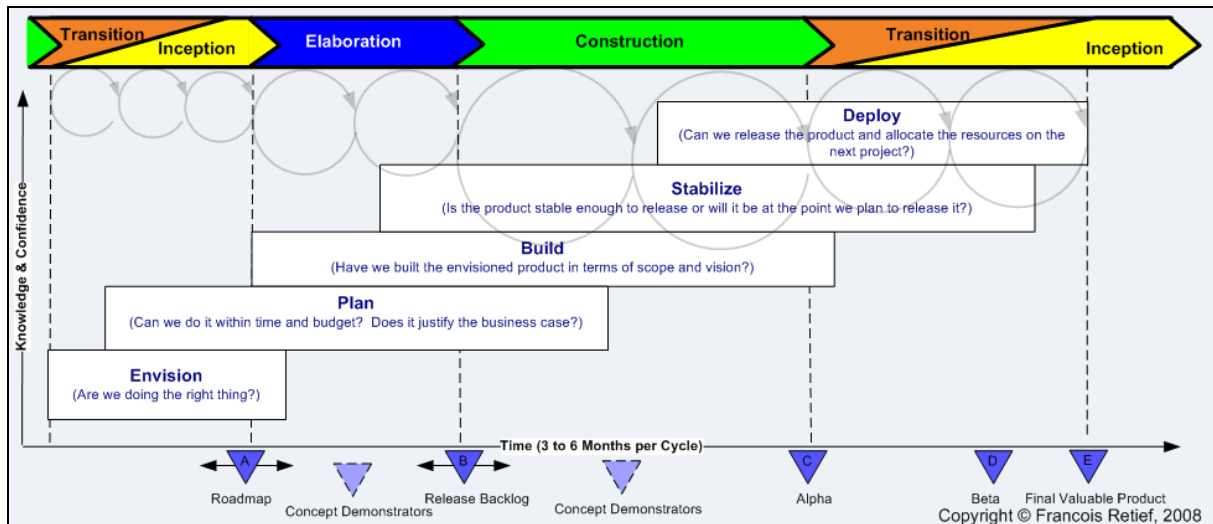


Figure 8. Development Model Overview

Transition and inception of the next incremental product release happens in the same time space. It is important to note that this is not a waterfall based model. Each high level phase has one or more iterations consisting of different levels of requirement analysis, design, development, qualification and acceptance testing.

Before starting

The following paragraphs describe specific elements that were found to require upfront attention to ensure the best results:

Team structure. The cross-functional team approach was initially implemented in a company lacking systems engineering skills and resources. It has proven highly beneficial in ensuring all requirements are represented. In addition, it bridges intercompany divides and provides a higher level of commitment from the organisation towards a release. Figure 9 demonstrates the use of teams across the development life cycle.

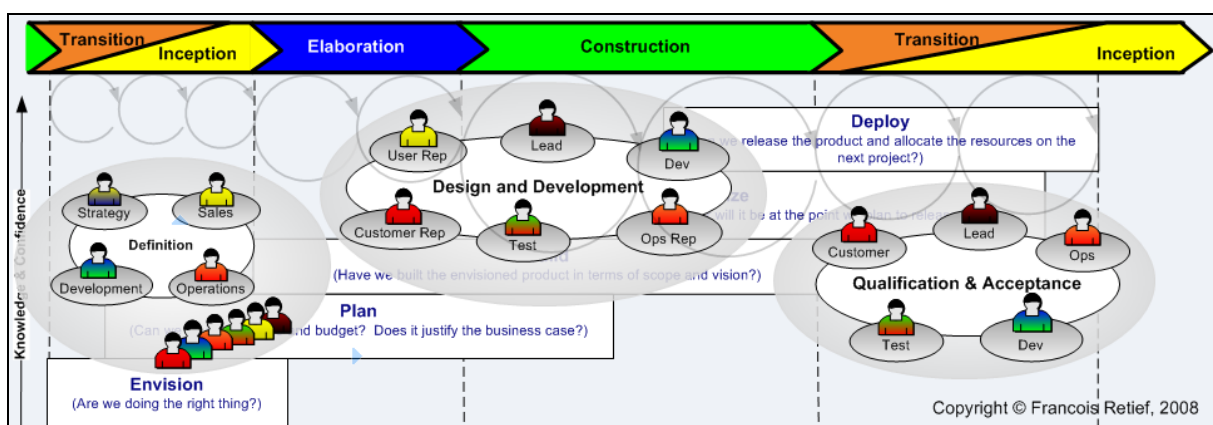


Figure 9. Cross-functional autonomous teams

This approach requires the establishment of cross functional self organising teams for the different stages of the project. It starts with a plan of who needs to be involved in every phase of the project and who else needs to have knowledge of what is happening. (This also allows for a high degree of systems thinking and total life cycle design throughout the development.)

It seems essential to keep the involvement team as small as possible but to make sure all project stakeholders are represented in the team. The larger the team, the longer it takes to reach a decision and the greater the risk that a member will not participate.

Once the correct team is assembled (both in terms of capability and mix) and they understand the goal and process, the responsibility shifts to ensuring they have the necessary resources (tools, information, management support etc.) to achieve this. The responsibility for the actual goal as well as the method to achieve it resides with the team.

Risk management. The aim is to target risk and continuously drive it down. This starts with selection of the highest risk items and working on them first. Mitigating tasks and contingency plans must be identified upfront. Development is then done in small vertical feature increments that are unit and system tested as they are completed throughout the development. A feature freeze period with focused systems testing and stakeholder demonstrations for feedback is executed at the end of each iteration.

Expectation management. It is critical to create the right expectation with everyone. A lot of effort must be invested to ensure that everyone involved in the project knows exactly what the aims are and how the process will work. Based on experience working with Microsoft's Professional Services Division, a daylong workshop with customers unfamiliar with the process of development is incorporated into the process (Hammond 2006). During this workshop the project vision and critical success factors are defined and captured.

Change management. Changes should be welcomed but all the stakeholders must understand that while an iteration is in progress no changes can be made from the outside on what the team is working on. Changes are discussed and implemented between sprints.

Gathering and prioritising requirements

A distributed requirement gathering network was found to work well in the telematics environment where multiple stakeholders are distributed over the world. A central product steering group manages the product development strategy. This group consists of representatives from Development (responsible for creating the *ability to produce a product* as well as *actually producing* it), Marketing and Sales (responsible for creating opportunities to deliver this product to customers), Operations (responsible for commissioning the product and ensuring it remains operational) and Strategic Management (concerned with the long term goals of the product and the organisation). Figure 10 illustrates this composition.

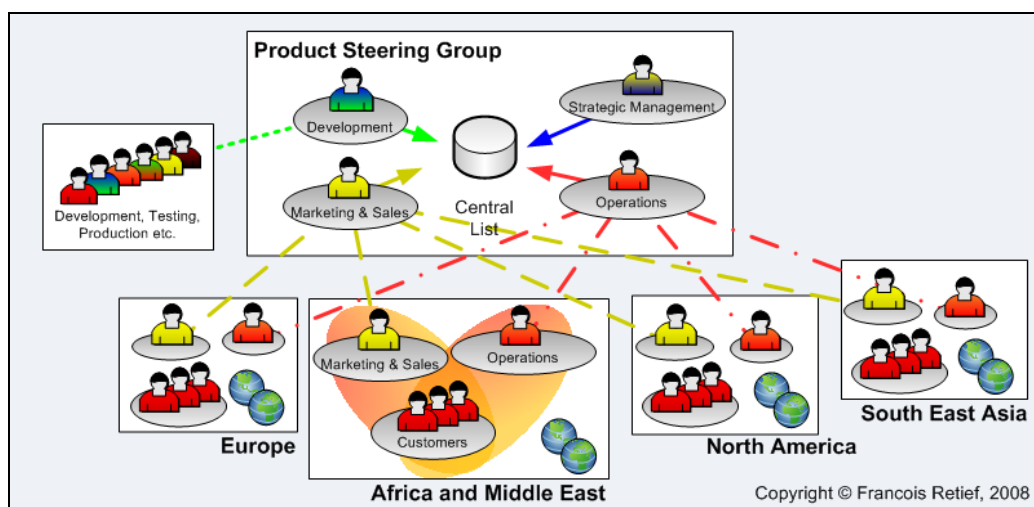


Figure 10. Requirement Gathering Network

Each member of the Product Steering Committee represents his members in the meetings. Between them they propose their most important requirements and negotiate consensus on importance to the organisation. They are responsible for deciding what features ends up where on the roadmap. They are also the group that does change approval.

Release and iteration planning

“Even the best planning is not so omniscient as to get it right the first time.”

Frederick P. Brooks

Planning is done in a number of fast cycles at the beginning of each iteration. The targeted release date is determined and a calculation is made of the number of development hours available. Quick high level planning is then done for all the targeted features in priority sequence. This ends when the available number of hours have been allocated. Feature groups are introduced to minimise the amount of items that simultaneously needs to be handled. As it becomes apparent that features are too large to fit in, or that the difference in priority of the sub features is too great, sub-items are created.

Product backlog. Selection of the scope for the next release starts with the list of identified requirements from all stakeholders as shown in Figure 11 block 1.

Product Backlog				
Backlog Item	Importance	Risk	Quick Estimate	
Value Item 1	1	2	25	
Value Item 2	2	2	18	
Value Item 3	2	3	3	
Value Item 4	1	1	57	
Value Item 5	3	1	25	
Value Item 6	2	2	22	
Value Item 7	2	1	34	
Value Item 8	1	3	42	
Value Item 9	3	3	1	
Value Item 10	3	2	2	
Value Item 11	2	2	5	
Total				234

Figure 11. Initial Product Backlog

The development team, that is actually going to do the work, does a quick risk assessment and estimation of each item (Figure 11 block 2). This is a best effort estimate, done based on a quick candidate design penned out, typically taking 5 to 45 minutes, but not more than 2 hours. The attempt is to get it as accurate as possible, but if it is within order of magnitude it is good enough.

Based upon this, a rough timeline can be generated. Taking the example in Figure 12, given that the development team can do 50 units per 30 work days, it means that it will take roughly 5 cycles of 30 days each before all the features can be implemented. (An overhead factor can be added for the amount of interdependency as well as risk).

Product Backlog						
Backlog Item	Importance	Risk	Quick Estimate	Priority	Iteration	
Value Item 1	1	2	25	1	1	
Value Item 6	2	2	22	3	1	
Value Item 3	2	3	3	6	1	
Value Item 4a	1	1	30	4	2	
Value Item 2	2	2	18	5	2	
Value Item 4b	1	1	27	4	3	
Value Item 8a	1	3	23	2	3	
Value Item 8b	1	3	19	2	4	
Value Item 5	3	1	25	11	4	
Value Item 11	2	2	5	7	4	
Value Item 10	3	2	2	10	5	
Value Item 9	3	3	1	8	5	
Value Item 7	2	1	34	9	5	
Total			234			

Figure 12. Product Backlog with Priorities and Iteration allocations

Armed with these estimates the stakeholders can refine their priorities, identify logical release points (if needed) and determine which features should go into which release. Higher priority is assigned to items with higher value to the total organisation and to high risk items that impose uncertainty on the release dates. Figure 12 shows the example iteration allocations.

The final result is a preliminary release milestone plan as indicated in Figure 13, with the understanding that the certainty of deliverables decreases the further one goes into the future, but also that the future milestones will be updated frequently with the input of all stakeholders taken into account.

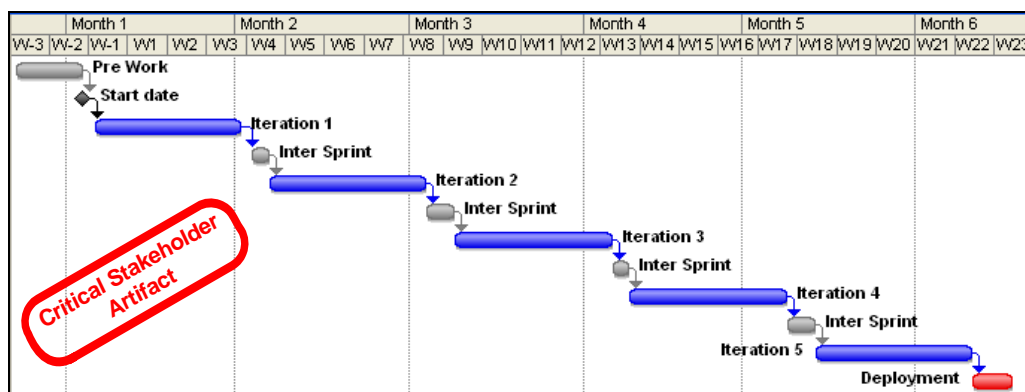


Figure 13. Release Milestone Plan

Using this methodology, a roadmap spanning as far into the future as the organisation requires can be drafted with a reasonable amount of effort.

Innovation. How does one schedule innovation? It is critical to decouple research from development if reliable timelines are expected. Being upfront with the customer is of the essence. If unable to say how to do something and per implication when it will be ready, a separate development stream should be created, decoupled from the production releases (See Figure 14). These research items still have defined deliverables that are tracked with the rest of the items of the iteration but more lenience is given towards scope changes. This allows for research that is budgeted into the main development effort and tracked, but without imposing undue strain on delivery dates. Once the risk and uncertainty have been sufficiently

diminished the item is reincorporated into the production stream.

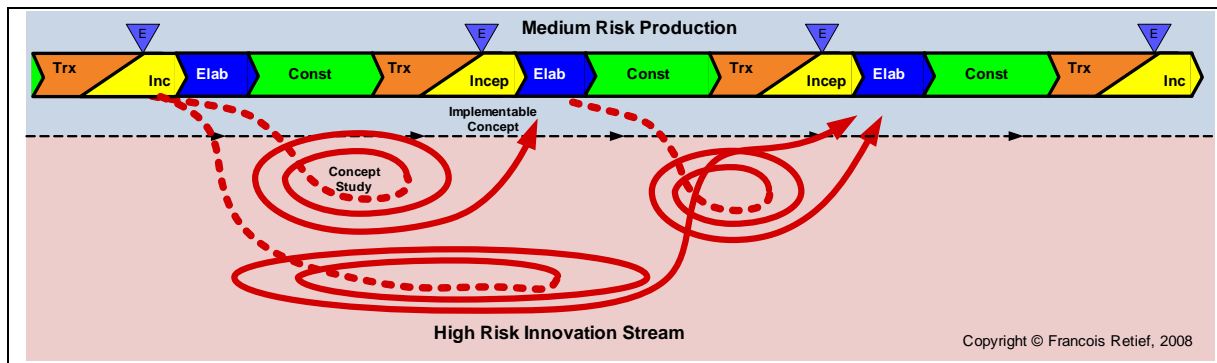


Figure 14. High Risk Development Items

Detail iteration planning and progress tracking

Detail iteration planning is left up to the team. The team lead facilitates a process where all the deliverables for an iteration are broken up into sub-tasks and allocated to team members as indicated in Figure 15 block 1. Tasks should be between 1 and 3 days. Smaller than that and there is an excessive amount of small tasks to manage, larger and it becomes difficult to track progress.

Backlog Item	Risk	Detail Estimate	Priority	Owner	Remain
Value Item 1	2	25	1		20
Task 1.1 (Acceptance Test)		2		AA	0
Task 1.2 (Acceptance Test)		5		BB	4
Task 1.3		4		BB	2
Task 1.4		3		AA	3
Task 1.5		4		CC	4
Task 1.6		5		CC	5
Task 1.7		2		DD	2
Value Item 6	2	22	3		11
Task 6.1 (Acceptance Test)		5		AA	0
Task 6.2 (Acceptance Test)		3		DD	3
Task 6.3 (Acceptance Test)		4		BB	4
Task 6.4		4		DD	0
Task 6.5		2		DD	1
Task 6.6		1		CC	0
Task 6.7		1		CC	1
Task 6.8		2		DD	2
Value Item 3	3	3	6		0
Task 3.1 (Acceptance Test)		1		CC	0
Task 3.2		2		AA	0
TOTAL		50			31

Figure 15. Iteration Backlog

During the development each team member does a daily estimate of how much time is left on each of the items allocated to him as indicated in figure 15 block 2. The remaining time is tracked on a burn down chart as shown in Figure 16.

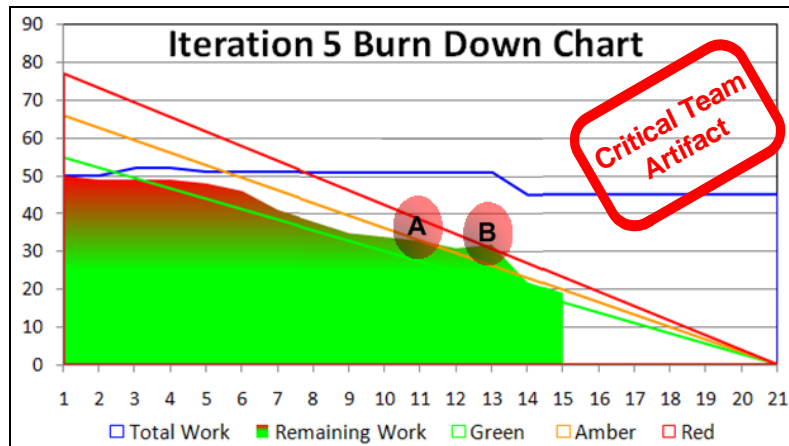


Figure 16. Burn-down chart

The chart also shows the total quoted effort as well as green, amber and red lines. The green line indicates the average amount of hours the team burns over an iteration as empirically calculated. The red line shows the maximum amount of work that the team can do and the amber line lies in the middle of the red and green lines. When the remaining amount of work crosses the amber line (e.g. A in Figure 16) the team needs to devise a contingency plan to get things under control. When the red line is crossed (e.g. B in Figure 16) this plan needs to be implemented. In the example in figure 16 lower priority tasks were removed from the iteration backlog.

Scrum meeting. Daily synchronisation meetings are facilitated by the team lead or “scrum master”. These are quick (15 minute), high energy, stand-up meetings where the team shares information, commitment and success with each other. The meeting is held in the team’s common work area. Critical artefacts (burn down chart, project vision, milestone plan) are updated before the meeting and placed where visible.

The following 4 questions as adapted from (Larman 2004) are addressed by each team member:

- Have I been able to finish what I committed to yesterday?
- What am I comfortable to commit to have done by tomorrow?
- What obstacles do I need removed to meet my goals?
- What obstacles do I foresee I’m going to put in other team members’ path?

These answers are addressed to the team. Anything falling outside this scope is quickly noted for offline discussion.

Consensus release

This may seem trivial but is an important and an often ignored point. The development continues until the stakeholders (including the customer) accept it. This is counterbalanced by the stakeholders continuing to pay for the development while it continues.

Continuous learning and improvement

“Life can only be understood backwards, but it must be lived forwards.”

Soren Kierkegaard

This is one of the ***most important*** aspects of agile development. At the end of every iteration a *retrospective* is done. There are two main points to this:

1. What has been learned about the project and how does it influence the current planning?
2. What was learned about the process being followed? What new elements need to be added to the process and what existing elements are superfluous?

Following this practice ensures rapid process improvement on a continuous basis and contributes towards a "continuous learning mindset". The improvements appear small when viewed in isolation, but are cumulative and has a huge impact over the long term.

Conclusions

Following agile software development principles has proven highly beneficial in developing complex software systems, specifically in the telematics industry. It has resulted in improved business value per release, better visibility, a more manageable environment, improved estimation of delivery, more commitment, motivation and satisfaction and less stress amongst the development team, customers, support personnel and other stakeholders.

To compensate for the challenges on a complex system delivery and higher reliability requirements the following additions to the basic agile methodology were made:

- A more formal requirement gathering and release management process.
- While testing was done continuously at the end of each iteration, a formal feature freeze period was enforced, with focused systems testing. Large scale field trials and a managed deployment process were also done before a final release.

References

- Hammond, Pierre, "Project Initiation- A Practical Approach to Project Workshops", 31 May 2007, INCOSE Western Cape Chapter Meeting.
- Hartmann, Deborah, *Interview: Jim Johnson of the Standish Group* (August 25, 2006). <http://www.infoq.com/articles/Interview-Johnson-Standish-CHAOS>.
- Larman, Craig, *Agile & Iterative Development, A Managers Guide*, Pearson Education, Inc, Boston, 2004.
- Kruchten, Philippe, "What Is the Rational Unified Process?" *The Rational Edge* (January 2001), <http://www.ibm.com/developerworks/rational/library/content/RationalEdge/jan01/WhatIsTheRationalUnifiedProcessJan01.pdf>
- Van der Mewe, J.J.N., "Agile Development In 20 Minutes" (June 22, 2005), Cape Town Software Process Improvement Meeting.

Biography

Francois Retief obtained a degree in Electronic Engineering from the University of Stellenbosch. He has held numerous leadership positions throughout his studies and working life, including junior mayor of Bloemfontein, chairman of the Student Engineering Council of Stellenbosch as well as general secretary of IAESTE-SA. He is also one of the founding members of the Western Cape Branch of INCOSE SA. He has worked as a systems and project engineer, field and test engineer, production manager and systems engineering manager. Currently he has his own business providing consulting and system development services.