# METHODS & TOOLS

**Global knowledge source for software development professionals** **ISSN 1023-4918**

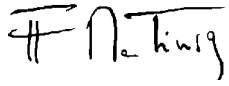
Winter 2004 (Volume 12 - number 4)

## Thinking Above the Method Level

This issue is focused on some aspects of the Agile approaches. With a simplified view, I see agility as a "back to simplicity" movement. It is for me a reaction to the (R)UP trend, a little bit like the RAD approach followed the wave of the structured methodologies. The fact that some of you may never have heard of methods like SSADM, Jackson System Development or Information Engineering is a reminder that methods can be rapidly out of view. In my opinion, the problems often do not lie in the chosen approach, but rather with the fact that organisations are inclined to adopt methods without thinking, or worse being able to think, how they fit within their software development context. Despite all the literature about it, software development teams work as if they still believe that a "silver bullet" exists, but you just have to change it from time to time ;-)

The fact that method promoters rarely take enough time to expose why and when they work well... and why and when they may fail reinforces this situation. Each project has its own characteristics and lives in a specific context of organisation, customers, developers, etc. A large outsourced project for a US financial company would perhaps not require the same approach that an internal development of software for a medical device in India. We lack however frameworks to examine the project's context and to relate what kind of approach work the best in these circumstances. Some will say that processes like RUP can be customised for specific projects, but in reality you have rarely the time or budget to perform customisation. Once upon a time, I used to collaborate with part to the Swiss IT consulting department of one of the Big Seven. They had a large methodology with "process roadmaps" that should be tailored for every project. The stuff was already heavy to swallow for consultants and the recommendation was to hide it from the final customers.

I think we also miss the people that try not to "sell" only one method, but work to understand the context in which the project will live and try to create the best approach for a success. The Agile approaches put some emphasis on simplicity. This is attractive and positive in my opinion. It could not work in every situation, but even the CMM recommend starting with some basic practices in your journey to level 2. In this issue, you will also see that there is diversity within agile approaches. Let's encourage and use this diversity in all areas of software development! As Grant Cause writes it: "Draw you own conclusions as to whether this methodology is right for you and your firm."



## Inside

# Agile, Multidisciplinary Teamwork

Gautam Ghosh, gautam@userminded.no
Userminded, Oslo, Norway, www.userminded.no

**Abstract**

By not segregating customers and users from the designers and developers, but rather enabling them to work together in a single team, it is possible to use the agile approaches such as DSDM, Turboprototyping, SCRUM to achieve perceptible results. Multidisciplinary teamwork is based on being able to find suitable team members, doing work in workshops and visualising requirements, ideas and decisions with lo-tech tools. This formula has enabled successful teamwork in a number of IS projects in recent years.

During recent years, my colleagues and I have had to adopt new approaches in most of our IS projects. Two of the contributing factors have been the influence of dot.com boom and the increased focus on a well-designed user experience. Having alternated between traditional and agile methodological approaches, I have ensconced myself in the agile camp. So most of my recent experiences are from agile processes and projects using either the DSDM project framework or TurboPrototyping, a teamwork-based approach we pioneered (Ghosh 1999).

## The challenge

The main adversaries of agile processes are established, rigid documentation regimen, lack of time and lack of stakeholder buy-in. I have experienced the following challenges when working in with teams in such projects:

- Being able to explore different solutions within the agreed time frame;
- Getting there without having covered all bases;
- Satisfying diverse (and often contradictory) client and user expectations;
- Having something up and running quickly;
- Ensuring team preparedness for an agile approach;
- Documenting enough, but not too much;
- Enabling a multidisciplinary group of people to work together as a team;
- Aiding team-members who don’t see themselves as being creative to participate creatively;
- Ensuring the team’s buy-in to the process and resulting products;
- Supporting a shared understanding of both needs, constraints and solutions;
- Making sure that decision-making is based on informed choices.

## The process

Multidisciplinary teamwork is one of the success criteria in both user-centred approaches and agile methods. It is considered to be essential for agile processes in order to meet project objectives and to ensure stakeholder buy-in to both process and results (Stapleton 1998). It must be stressed that though this article concentrates on the teamwork, other factors such as clear goals, planning, iterative delivery, etc. also play a vital role in the success of agile projects.

In my experience, multidisciplinary teamwork depends on

- Putting together, establishing and sustaining a team;
- Doing all important team-based work in facilitated workshops;
- Visualising and documenting the team's efforts throughout the process.

**Enabling a diverse team to function well**

A multidisciplinary team needs to be made up of all the different stakeholders in the process. This includes project sponsors, domain experts, users, designers, developers and architects – just to mention a few. It is important that all the different interests are represented. This can be a difficult job because you can end up with a team that is too unwieldy to manage. So selecting team members can be a painful process.

When building a team, each participant's potential contribution to the process has to be evaluated. Involving the right stakeholders in the process increases ownership from the users' and customer side. The difficult part is excluding some of potential participants.

The ideal participant

- is a part of the team and identifies himself/herself with the team's goals;
- is empowered and entrusted with the task;
- has the support of his/her manager;
- is experienced and knowledgeable in his/her field;
- represents colleagues;
- has the time to participate;
- wants to contribute to the success;

Real-life team participants obviously represent these attributes to a varying degree. The participants from the customer or user groups are often hard to get hold of because they have important responsibilities in the organisation. Quite often some of our team members are picked for us, so we have to work with people who in one way or another represent 'less' than the ideal participant.

Since teams don't function well when decisions have to be made elsewhere, we try to avoid representation through proxy, e.g. the boss sending a secretary or the system architect sending a junior programmer to represent them.

An ideal team includes people from different disciplines with various skills. I don't split the team into separate user/owner/domain expert teams or designer/developer teams because that often leads to an 'us & them' attitude and reduces ownership. In our teams it is always 'us'. On a recent project, my team consisted of town planners, government and municipal bureaucrats, building contractors, a system architect, web designer and project manager.

**Keeping the team energised and productive**

Teamwork can only be productive if the team members appreciate the goals and strive towards reaching them. So it is important that they actually represent the different aspects of the problem or solution space and are aware of their own role in the team.

- Sponsors are key stakeholders because they represent the visions and can make necessary high level decisions;
- Domain specialists contribute towards the content and process;
- Users can enhance the usability;
- Designers/developers have the needed competence in creating the solution;
- The facilitator is the team's catalyst.

The team must treat people's ideas and concerns as equally important, irrespective of each team member's position or power, or they won't feel motivated to contribute. The team must also have the flexibility to change its course - we easily abandon one idea for another. This enables experimental thinking, but can lead to lower levels of buy-in and commitment.

**Facilitated workshops as a teamwork arena**

The use of facilitated workshop as an arena to bring together people to work is hardly a secret, but I consider them particularly important in getting a team of people with different skills, experience and aspirations to work together. It is possible to instill a common purpose in the team by keeping all activities related to exploration, creativity and decision making in the workshops.

In a recent project, the teamwork was initiated by facilitating stakeholders to find their way through a maze of requirements, user needs and ideas. Later on, after a short brainstorming session, we used one of the participant's ideas to mould a part of the solution we were working on. This enabled the team to trigger off each other's ideas. But as important it is to generate ideas, it is equally important to enable the team with decision-making (Cohn 2004).

In order to stay agile, the facilitator's task in the workshops is to enable the team to deliver results quickly without rushing. Ensuring that all issues are addressed and that the entire team understands them is important. However everybody doesn't need to understand everything all the time. Very often we can move on when the team feels that some of team members have understood the issue and have taken ownership of it.

One of the ways teams can achieve a shared understanding, come to an agreement and commit to group decisions is by visualising requirements, ideas and decisions in the workshops. This obviously requires focussed management so that the workshops are productive. So we go in for thorough planning and preparation in preshops (pre-workshop meetings) to reduce the risk of failure. In design workshops I make sure that the solution designers have considered several alternatives to the requirements before the workshops, enabling a flexibility and agility in the teamwork.

The success of multidisciplinary teamwork partially depends upon no single team member feeling substantially inferior to any of the others. In my experience, abstract and complex modelling leaves some of the team members out of the picture, so we skip heavy analysis and modelling in workshops. We do use some models sparingly – navigation models and information architecture, sometimes system architecture (occasionally Use Case models). Technical models like class diagrams; ERM models, event handling queues, etc. are never a part of the teamwork – despite the fact that the developers and architects in their own work use them.

Achieving something is important to successful teamwork, so I try to make sure that each workshop has achievable goals. The team has to decide towards the end of each workshop whether we have actually fulfilled the goals we set. This confirmation helps the team evaluate its own progress. Post-workshop documentation of the team's results ensures completeness.

**The walls are our tools**

Visualising all requirements, ideas and decisions in the workshops is the most important means of creating a shared understanding and commitment.

We 'paint' the walls with ideas and decisions, using a set of lo-tech visualisation tools. All the documentation in our workshops is wall based, so we avoid using a scribe or secretary to take minutes. By visualising requirements on the fly we level the playing field and enable changes as we move on. By writing things up, team members can share their ideas with every other participant. So the wall serves as an archive of ideas and decisions for the team – visible for all.

We keep track of the changes and document results by photographing the walls frequently. The photographs are later used in written documentation.

Our lo-tech visualisation tools include

- Wallware – coloured card in different shapes and colours;
- Flipcharts – for lists and sketches;
- Whiteboards – visualising concepts;

- MUI – Magnetic User Interface;
- Personas – illustrating user characteristics.

We do use some hi-tech tools, namely

- Projection screens – visualising design;
- Cameras – to document everything.

**Capturing ideas with Wallware**

Wallware comprises of cards in different shapes and colours that are fastened to the wall with blue-tack. In workshops, we use Wallware by noting each idea in large letters on a card using wide-tip pens. These cards are hung up on a wall under appropriate headings (goals, users, benefits, requirements, etc.). At regular intervals, the ideas are sorted and grouped, structuring them as much as possible. When the session is over, the end-result is documented using a digital camera. The content of photographs is used to document the proceedings, decisions and results from workshop.

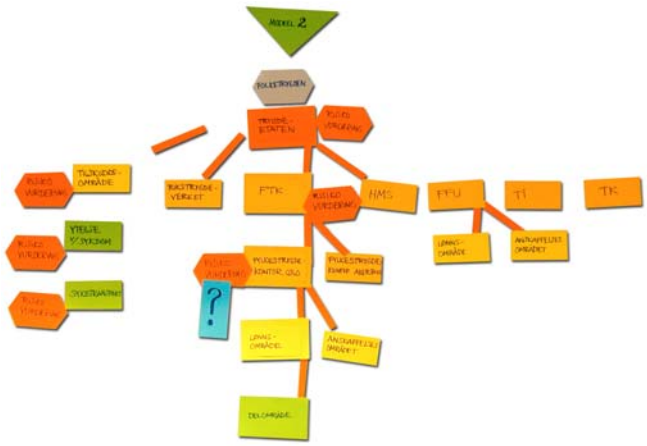



Figure 1 Wallware

The wall serves as a workspace, thereby ensuring that the ideas are visible to all the participants in the workshop. One portion of the workspace is assigned as a “parking-lot” so that ideas outside of the scope of the process can be effectively parked.

**MUI - Magnetic User Interface**

Instead of using abstract models, our approach has been to visualise solutions using web pages as a metaphor – the solution to every problem is a web page! This visualisation technique is described in detail earlier (Ghosh 1999)

Design-suggestions for each web page are drawn in workshops using MUI elements on the whiteboard. The team modifies each web page making sure that there is agreement about the contents and navigation scheme for each page. Navigational details are noted in the margins outside the MUI page. Finally each page is documented by photographing it with the digital camera.

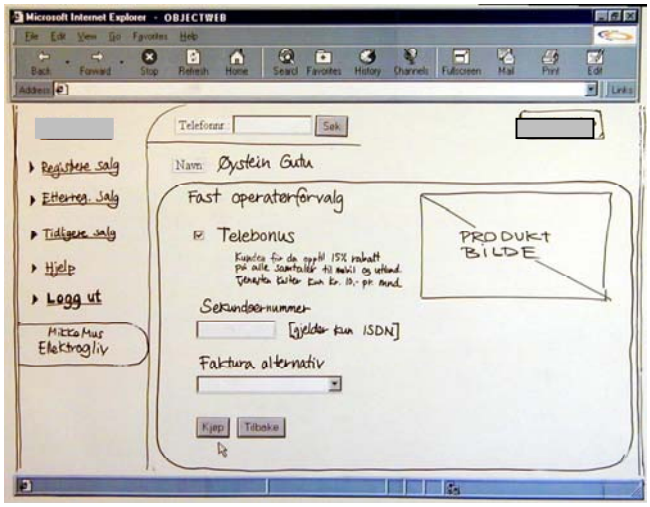



Figure 2 - A typical MUI prototype

The Magnetic User Interface (MUI) toolkit is made up of components that are printed onto a magnetic metal sheet. The toolkit contains a web browser, buttons with predefined texts (OK, Cancel, Print, Save, etc) and screen components like drop-down lists, check boxes and radio-buttons.

The components are to scale and can be used on a 120cmx90cm whiteboard (most whiteboards are larger – thus letting us use superfluous area on the board for comments). By visualising screen components, workshop participants can make realistic decisions about the solution. It is important to encourage and permit the team-members to participate in the design process by letting them move things around if they want to, since this often leads to the revelation of details that might have been otherwise overlooked.

**Representing users as personas**

Many projects conduct a user analysis resulting in a large set of users and their characteristics. This information is grouped in user profiles and is used to evaluate the validity and need for diverse aspects of the project. Unfortunately many of the user profiles I have seen are often vague, contradictory and very often of little use to the project team.

Alan Cooper (1999) has introduced personas into the design work. Personas are descriptions of imaginary persons that include information about goals, personal relationships, background, work and system related expectations.

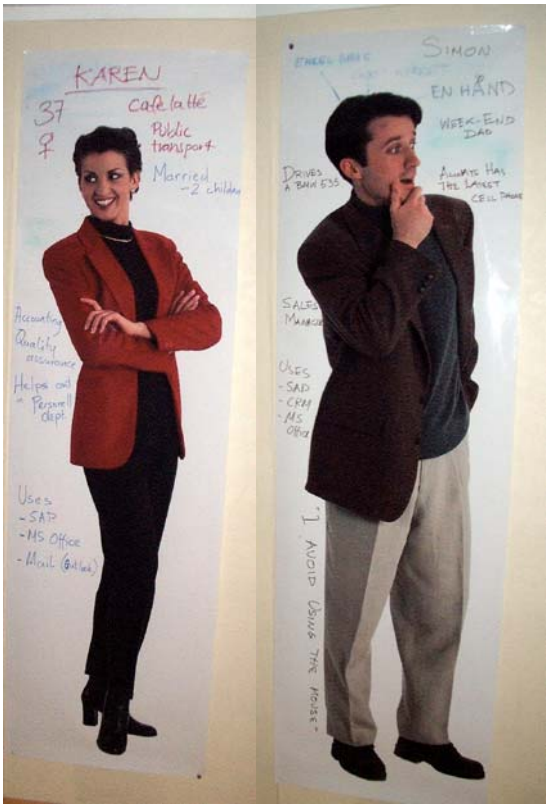



Figure 3 - Full-scale personas

Usually we construct 2-3 personas based on a high-level user analysis, using full size laminated photographs of people and writing the persona characteristics on the photographs. This approach lets us add and remove detail as time goes by - the lamination allowing us to wipe away characteristics that need to be amended. The persona photographs are hung on the wall in our project environment and in workshops. This leads the team to believe that they know the personas and as the project progresses, our team members refer to what the personas would say or do in given situations, e.g. “Simon would never bother to fill out any of the optional fields”. Above all, the use of personas has helped our teams to focus, eliminating many of the endless discussions of what the “user” would want, expect or need. Our users are few and have names.

**Lessons learned**

My experience with the agile teamwork approach is that it enhances communication and collaboration between designers, programmers, stakeholders and users. Some of the other observations are:

- Visualisation speeds up the decision making process;
- Wall documentation levels the field;
- It is not always easy to get hold of representative participants;
- The approach ties up team members for days at a time;
- Democratic solutions can be mediocre and contain compromises;
- Using MUI enables team to create realistic interactive solutions;
- Team members become more demanding in the course of the process;
- Clear rules of the game facilitate creativity and decision making;

- The process empowers the participants
- The process results in a shared understanding of problems and solutions.

Some traps and dangers I have experienced using this approach is

- You have to watch out for scope creep, creativity can take off;
- Wallware notations can obscure details, since everything is noted in short sentences;
- It is easy to be sidetracked by unimportant details;
- Strong personalities can side-track the process;
- The team loses faith if the process doesn't lead to results;
- Challenging participants' statements can lead to unwanted uncertainty.

**Conclusion**

Bringing together people from different environments with diverse goals requires planned facilitation to enable proper teamwork, but the team can work to produce results in an agile manner under the right working conditions. The main components of their success are having the right team members, working in a transparent way in workshops and using lo-tech tools to visualise all ideas and decisions.

**References**

1. Ghosh, G (1999) *Turbo-prototyping: Ultra rapid user centered web development* in S. Brewster, A. Cawsey & G. Cockton (Editors): Human-Computer Interaction - INTERACT'99 (Volume II), IFIP press

2. Stapleton, J. (1998) *DSDM – Dynamic systems development method – The method in practice*. Addison Wesley Longman Limited, Harlow, England

3. Cohn, M (2004) *User Stories Applied – For Agile Software Development*. Addison Wesley Pearson Education.

4. Cooper, Alan. (1999) *The inmates are running the Asylum*. SAMS, Indianapolis.

**Note**

This paper is based on a presentation with the same name - 'Agile Multidisciplinary Teamwork' – presented by the author and Øystein Gutu at the Agile Business Conference 2004 in London on 22.11.04.

# Adaptive Project Management Using Scrum

Craig Murphy, www.craigmurphy.com

Scrum is a lightweight process that can manage and control software and product development: it is a project management process. However, instead of promoting the traditional analysis, design, code, test, deploy "waterfall" approach, Scrum embraces iterative and incremental practices. Similarly, instead of being "artifact-driven", whereby large requirements documents, analysis specifications, design documents, etc. are created, Scrum requires very few artifacts. It concentrates on what's important: managing a project or writing software that produces business value.

## What is Scrum?

The first question that you'll probably want answered is: is Scrum an acronym for something? You would not be alone in thinking this; most other project management processes and techniques are acronyms. Scrum however is just a name, it's not an acronym. Similarly, it has very little to do with rugby either, although there is a "daily meeting" that might be compared to a traditional rugby scrum.

Scrum borrows from the "agile" world through its promotion of iterative and incremental practices. It has a simple implementation that is designed to increase productivity and reduce the time it takes to benefit from a software/product development. Importantly, it embraces adaptive and empirical systems development.

That last paragraph introduced two of Scrum's key facets: adaptive and empirical. The majority of project management methods and techniques are very prescriptive; they tie us down to a fixed sequence of events and offer little in the way of flexibility. Similarly, newer, less mature methods often claim to be a panacea, yet they lack any definitive proof or track history. Fortunately, Scrum is mature; it has a track record going back as far as 1995 and earlier. Similarly, it is scaleable: Scrum can be used on projects of any size. Whilst I will not be discussing it in this article, "Scrum teams" allow Scrum to be used to manage enterprise-level projects.

Scrum also promotes and lends itself to managing eXtreme Programming (XP) based projects. XP uses "customer on site" as a means of ensuring that the development team's questions are answered but also to ensure that what the development team is producing what the customer actually wants.

## The Opposite of Waterfall

Traditionally, we followed a cycle involving requirements gathering, analysis, design, develop, test, deploy with each stage being completed before moving on. This was known as the waterfall approach. Whilst there is a place for the waterfall approach, it has been the subject of a torrent of abuse in recent years. Most notably, the waterfall approach promotes the creation of up-front documentation before any real business value is created. This is confounded by the fact that product development is started downstream, or much later in the project's expected timeframe. This has the obvious disadvantage of delaying the point at which business value can be realised.

But it gets worse: the customer/user endeavour to ensure that all of their requirements are documented during the early stages, thus the feature set is top-heavy. Failure to prioritise the feature-set often results in low quality systems that are overloaded with features that the

customer/user does not actually require in "version 1". Indeed followers of Mary Poppendieck's work (www.poppendieck.com) will know that "80% of a product's value comes from 20% of its features".

With this in mind, can we conceivably build a product (a software product) that provides 20% of the feature set? Yes, we can and we do it iteratively. We can delivery "version 1" with 20% of the features, then, a little later, "version 2" with a further collection of features. The beauty of this approach is that development of 20% of the features should not take 100% of the project's expected schedule and budget: we can realise business value much earlier in the cycle.

Scrum embraces the opposite of the waterfall approach whereby we start working on the analysis as soon as we have some requirements, as soon as we have some analysis we start working on the design, and so on. In other words, we work on small pieces at a time. This approach can be called iterative. Each iteration consists of some requirements gathering, some analysis, some design, some development and some testing culminating in an iterative release cycle (many deployments).

**Scrum Basics**

Scrum revolves around the ethos of simplicity, resulting in delivery of something that moves the project forward. It achieves this by proposing the following questions (referred to as Scrum's three questions).

| Scrum asks… | Fundamental Project Management issue |
|---|---|
| What have you done during the last 24 hours? | This is progress, it's work completed to date |
| What do you plan to do in the next 24 hours? | This is forward planning, it is work you are about to do |
| What's stopping you getting on with the work of the next 24 hours? | These are your impediments or obstructions, it might be things you need in order to work… more forward planning. It's also identification of immediate risks. |

**Scrum Roles**

Scrum uses three "roles": Product Owner, ScrumMaster and Project Team.

The Product Owner is possibly a Product Manager or Project Sponsor, a member of Marketing or an Internal Customer.

The ScrumMaster is key, he or she "represents management to the project". Such a role usually filled by a Project Manager or Team Leader. They are responsible for enacting Scrum values and practices (more about these shortly). Their main job is to remove impediments, i.e. project issues that might slow down or stop activity that moves the project forward.

The Project Team should consist of between 5-10 members. The team itself should be cross-functional, involving individuals from a multitude of disciplines: QA, Programmers, UI Designers, etc.

**The Process**

"Out of the box" Scrum is best described by Figure 1. Most projects have a list of requirements (type of system, planning items, type of application, development environment, user

considerations, etc.) Scrum records requirements in a Product Backlog. Requirements need not be precise nor do they need to be described fully. As with most projects, the requirements are sourced from the expected users or "the business". The Product Owner prioritises the Product Backlog: items of importance to the project/business, i.e. those items that add immediate and significant business value, are bubbled up to the top.

The Project Team responsible for doing the actual work then creates a Sprint Backlog: this comprises of Product Backlog items that they believe can be completed within a 30 day period. The Project Team may liaise with the Product Owner and others in order to expand item(s) on the Sprint Backlog. After 30 days have elapsed, the team should have a "potentially shippable product increment". I will discuss the make-up of the Project Team later in this article, under the topic: Scrum Roles.

The Product Owner, the ScrumMaster and the Project Team will make an initial pass over the Product Backlog items where they work out roughly how long each item will take. Initially, these are estimates, best guesses. As time progresses, well within 30 days, we'll know if the estimate was even close.

Scrum lets us refine our estimates on-the-fly: if we believe that a task will take longer than envisaged, we have the ability to say so before the tasks starts. By only ever working with small work packages (time-boxed to 30 days), any schedule/requirement issues are dealt with as soon as they are identified, not much further downstream where the cost of recovery is considerably higher.

What does this "potentially shippable product increment" actually mean? Put simply, every 30 days, the team should provide something of value to the business, something they can use or something that provides considerable direction.

The beauty of the 30 days approach is this: if the Product Owner, customer or the business likes what they see at the 30 day interval, they can then re-prioritise the Product Backlog. This is important as it means that we are only ever producing goods that will be used by the business: remember that only 20% of a [software] product's features are used frequently. In other words, 80% of a software product's "value" comes from 20% of it features.

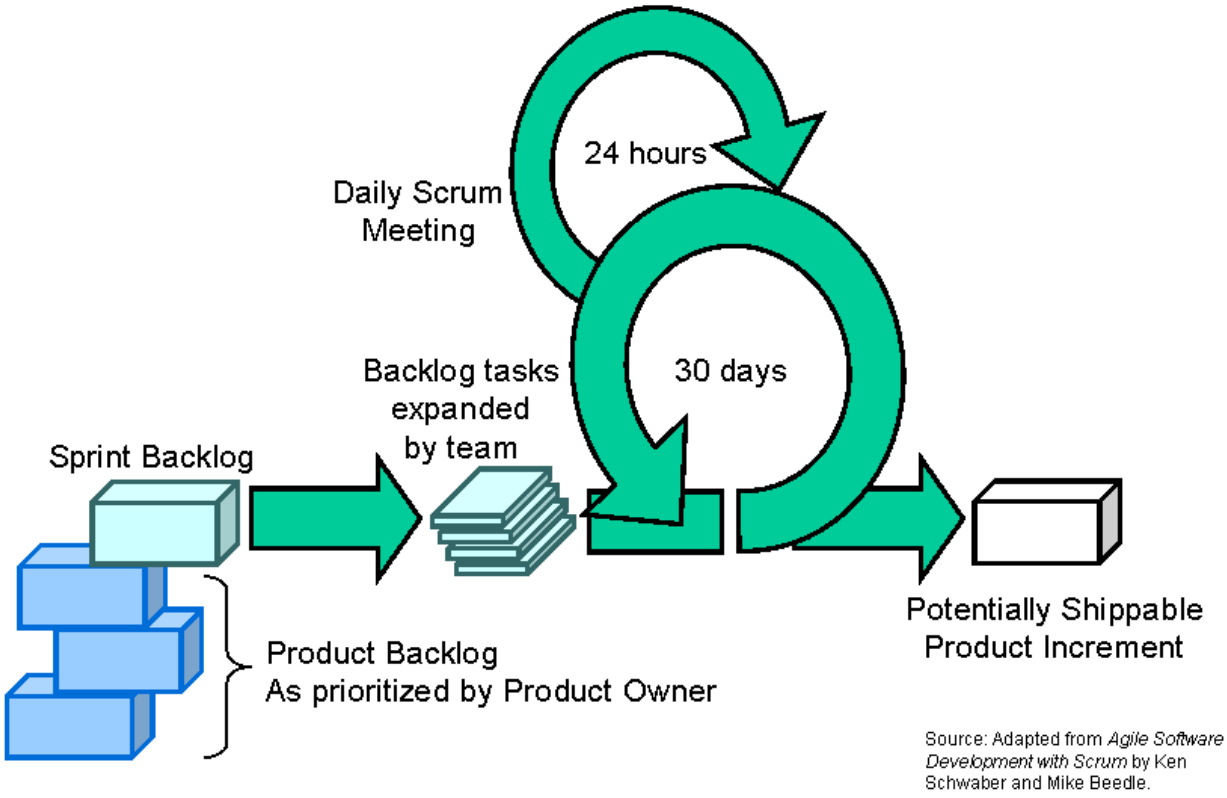



Figure 1 - The Scrum Process

**Daily Scrum Meeting**

One of Scrum's primary practices is the 24 hour cycle shown in figure 1: the Daily Scrum meeting.

How many of you attend meetings on a regular basis? If you are in the corporate environment, meetings seem to be the norm and often they have little business value apart from to act as a caffeine injection mechanism. Folks who sit down at meetings get too comfortable: they attend meetings for the coffee and doughnuts, not for the project's sake. Once relaxed, those same folks often stray off the meeting agenda and start discussing items that are either on the project periphery or are not even project related. I am sure you've all been in such meetings…

Scrum resolves these issues using two simple approaches.

Firstly, Scrum-managed project meeting rooms are devoid of chairs. Attendees have to stand up. This might sound cruel, but it focuses the mind and those folks who are capable of sitting in

(often unproductive) meetings for hours on end are soon discouraged.

Secondly, Scrum-managed meetings are time-boxed or time limited. The Daily Scrum meeting is typically time-boxed to 15 minutes. Only extraordinary projects should require more than 15 minutes. Here is the crux: if you can't say what you have to say succinctly in a short space of time, you're waffling, get your coat. Keeping it time-boxed focuses folks' minds and helps keep agenda items targeted at what's important: Moving the project forward towards delivery of "something"… and identifying and removing obstacles that prevent this goal being met.

The purpose of the Daily Scrum meeting is to answer Scrum's three questions:

1. What did you do yesterday?
2. What will you do today?
3. What obstacles are in your way?

But there is more! The ScrumMaster and the Project Team are the only people who are allowed to talk: outsiders may listen in, but are removed or silenced should they say anything. This is all about who is committed to the project or not. Outsiders tend to volunteer items that are important to them, but not necessarily to the project and its immediate goal: delivery business value within the 30 day sprint.

Scrum refers to those who are committed to a project as "pigs" and those who are not as "chickens". The story goes like this: A chicken and a pig where chatting about setting about a business together. The pig asked the chicken: "What kind of business would we set up?" The chicken thought for a moment and said "How about a restaurant?" The pig liked the idea but asked "What would we serve?" The chicken responded "How about ham'n'eggs?" At which point the pig refused to take the business venture any further. The chicken was confused and asked "Why?" The pig responded "Well, you would only be involved, whereas I would be committed."

One thing I should mention is the fact that Scrum expects late-comers to any of its meetings to pay a nominal £1, $1, or €1 fine: at various milestones within the project the fines are donated to charity. You may find this slightly humorous; however it does focus the mind and it does prevent meetings dragging on because of late-comers (we have all been in meetings that have to start over because of late-comers, right?) Late-comers are simply told to pay their fine, the meeting continues without further ado.

## Scrum Artifacts

Scrum has remarkably few artifacts. There are three artifacts, each of which can be managed using nothing more than an Excel spreadsheet. More advanced / complicated tools exist, many are expensive, web-based or are still under development. Web-based tools are great, but they are not much good if there is no "off-line" operation: I conduct much of my project management (and thinking) in airport lounges where Internet connectivity is a pay-for luxury.

Figure 1 hints at Scrum's artifacts: there is a Product Backlog and a Sprint Backlog. The Product Backlog is a prioritised list of first cut refinements. I say first cut, because the Product Owner is free to adjust the order in which Product Backlog items are development, they are even free to add new items this is the spirit of "agile".

Graphical feedback that can be printed out and displayed in a public place makes progress reporting very visible. Scrum encourages this kind of reporting and offers Burndown Charts as a means of graphically displaying a project's progress or not as the case may be. We will take a look at burndown charts later in this article.

I mentioned earlier that Scrum's artifacts can be managed by nothing more than a spreadsheet. Indeed, the Product Backlog can be represented using an Excel worksheet. Each Sprint Backlog then occupies another sheet within the same workbook.

Figure 2 presents a sample Product Backlog. Keeping with the ethos of Scrum and "agile", you shouldn't be planning more than one sprint into the future. Indeed, the contents of a Sprint are usually defined during a "Sprint planning meeting".

**Product Backlog**
**Estimating System Upgrade**

| Sprint | ID | | Backlog Item | Owner | Estimate (days) | Remaining (days) |
|---|---|---|---|---|---|---|
| 1 | 1 | Minor | Remove user kludge in .dpr file | BC | 1 | 1 |
| 1 | 2 | Minor | Remove cMap/cMenu/cMenuSize from disciplines.pas | BC | 1 | 1 |
| 1 | 3 | Minor | Create "Legacy" discipline node with old civils and E&I content | BC | 1 | 1 |
| 1 | 4 | Major | Augment each tbl operation to support network operation | BC | 10 | 10 |
| 1 | 5 | Major | Extend Engineering Design estimate items to include summaries | BC | 2 | 2 |
| 1 | 6 | Super | Supervision/Guidance | CAM | 4 | 4 |
| | 7 | Minor | Remove Custodian property from AppConfig class in globals.pas | BC | 1 | |
| | 8 | Minor | Remove LOC_ constants in globals.pas and main.pas | BC | 1 | |
| | 9 | Minor | New E&I section doesn't have lblCaption set | BC | 1 | |
| | 10 | Minor | Delay in main.releaseform doesn't appear to be required | BC | 1 | |
| | 11 | Minor | Undo modifications to Other Major Equipment in formExcel.pas | BC | 1 | |
| | 12 | Minor | AJACS form to be centred on the screen | BC | 1 | |
| | 13 | Major | Extend DUnit tests to all 40 disciplines | BC | 6 | |

Figure 2 – A sample Product Backlog

Using the information in Figure 2, we can derive another worksheet, the Sprint Backlog (Sprint 1). Figure 3 presents a sample Sprint Backlog: it contains a list of the things that will be "done" during the Sprint. Each Sprint item has an estimate of how long it should take to complete, usually measured in hours. Looking at figure 3 again, you can see that none of the six tasks have been worked on.

During the Sprint's 30 day period, the Project Team must update the Sprint Backlog. For example, if BC spent four hours on item 1, Remove user kludge in .dpr file, on Tuesday 2nd (November in this case), he would enter '4' into Sprint day 2's Backlog item 1 column/row combination.

**Sprint 1**

01/11/2004

**19 days work in this sprint**

| | Backlog Item | Backlog Item | Owner | Estimate | Sprint Day | 1 Mo | 2 Tu | 3 We | 4 Th | 5 Fr | 6 Sa | 7 Su |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | Hours remaining | 152 | 152 | 152 | 152 | 152 | 152 | 152 |
| 1 | Minor | Remove user kludge in .dpr file | BC | 8 | | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
| 2 | Minor | Remove cMap/cMenu/cMenuSize from disciplines.pas | BC | 8 | | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
| 3 | Minor | Create "Legacy" discipline node with old civils and E&I content | BC | 8 | | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
| 4 | Major | Augment each tbl operation to support network operation | BC | 80 | | 80 | 80 | 80 | 80 | 80 | 80 | 80 |
| 5 | Major | Extend Engineering Design estimate items to include summaries | BC | 16 | | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| 6 | Super | Supervision/Guidance | CAM | 32 | | 32 | 32 | 32 | 32 | 32 | 32 | 32 |

Figure 3 – Sprint 1 at inception

Keeping the Sprint Backlog updated is key: not only does it allow us to work out how fast a team can work (their velocity), it is an early warning indicator.

What is useful about the Sprint Backlog is its ability to be displayed graphically. Scrum uses burndown charts to represent "work done". Figure 4 presents a burndown chart where no work as been performed in the sprint. The idea behind burndown charts is that they should demonstrate a steady drive to zero hours remaining: it represents a pace of work that should be sustainable. In reality however, some work takes longer than others, and some are even shorter, so the burndown graph may not be a perfect straight line.

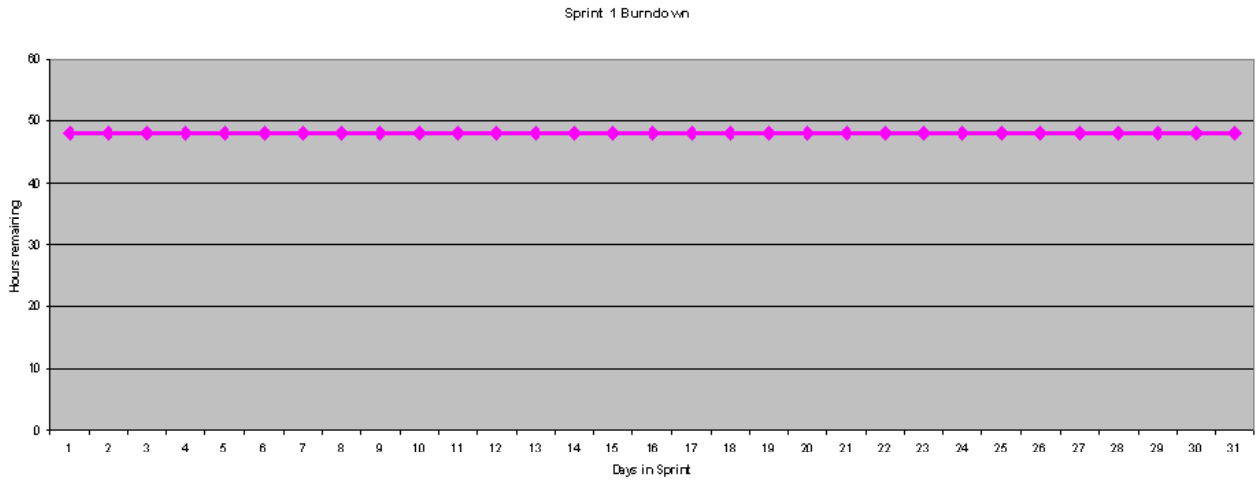



Figure 4 – Sprint Burndown, no work performed

Figure 5 presents a Sprint backlog that has been updated after work has been performed. Following the "Remove user kludge in .dpr file" item through, BC performed 4 hours work on Wednesday 3rd, followed by 2 hours work on Thursday 4th and finished the task on Friday by spending 2 hours on it.

You will also notice that on the same Wednesday, BC spent 4 hours on "Remove cMap/cMenu…". Assuming a working day of 8 hours, BC has split his time between two tasks.

**Sprint 1**

| 01/11/2004 | | | | Sprint Day | 1 Mo | 2 Tu | 3 We | 4 Th | 5 Fr | 6 Sa | 7 Su |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **19 days work in this sprint** | | | Hours remaining | 152 | 150 | 140 | 130 | 118 | 118 | 118 |
| | Backlog Item | Backlog Item | Owner | Estimate | | | | | | | |
| 1 | Minor | Remove user kludge in .dpr file | BC | 8 | 8 | 8 | 4 | 2 | 0 | | |
| 2 | Minor | Remove cMap/cMenu/cMenuSize from disciplines.pas | BC | 8 | 8 | 8 | 4 | 0 | | | |
| 3 | Minor | Create "Legacy" discipline node with old civils and E&I content | BC | 8 | 8 | 8 | 8 | 6 | 0 | | |
| 4 | Major | Augment each tbl operation to support network operation | BC | 80 | 80 | 80 | 80 | 80 | 78 | 78 | 78 |
| 5 | Major | Extend Engineering Design estimate items to include summaries | BC | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| 6 | Super | Supervision/Guidance | CAM | 32 | 32 | 30 | 28 | 26 | 24 | 24 | 24 |

Figure 5 – Sprint 1 after work has been performed

The beauty of the Sprint Backlog is its simplicity. By recording the hours of actual work completed versus the estimated hours to complete, we are able to plot equally simple, but powerful Burndown Chart that should provide you with the confidence that progress is being made. If the Burndown Chart does not indicate that progress is being made, then it has served another good purpose: it gives you an early warning that this sprint contains either too much work or too little.

Figure 6 presents a Burndown Chart based of Figure 5's Sprint Backlog. Ideally, the Burndown Chart should indicate a level velocity, i.e. work is performed at a steady rate. In reality, we need to factor in weekends and other week-day distractions. As we will see shortly, if a Project Team or individual is distracted for a few hours, Burndown Charts are a great way of identifying the distraction very early in the project.

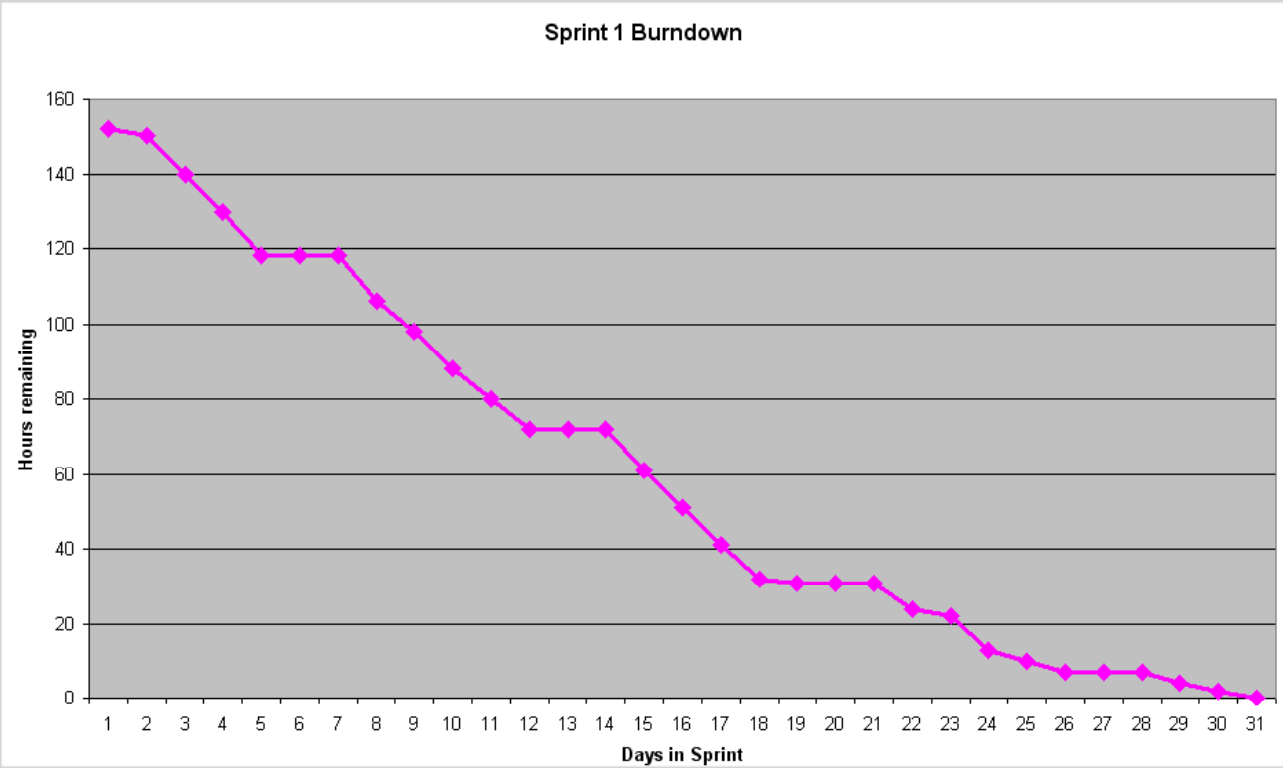



Figure 6 – Sprint Burndown after work has been performed

Figure 7 presents a burndown chart demonstrating that progress is being made, however by no means fast enough. The Burndown Chart gives us a clue that there is something wrong within 2 to 3 days of starting and confirms that work is not progressing at a good speed by days 7 through to 14.

There are a few reasons why this shape of Burndown Chart appears:

1. The Project Team, or individuals are being distracted from their work
2. The Sprint Backlog is not being updated
3. The Sprint Backlog items are too difficult

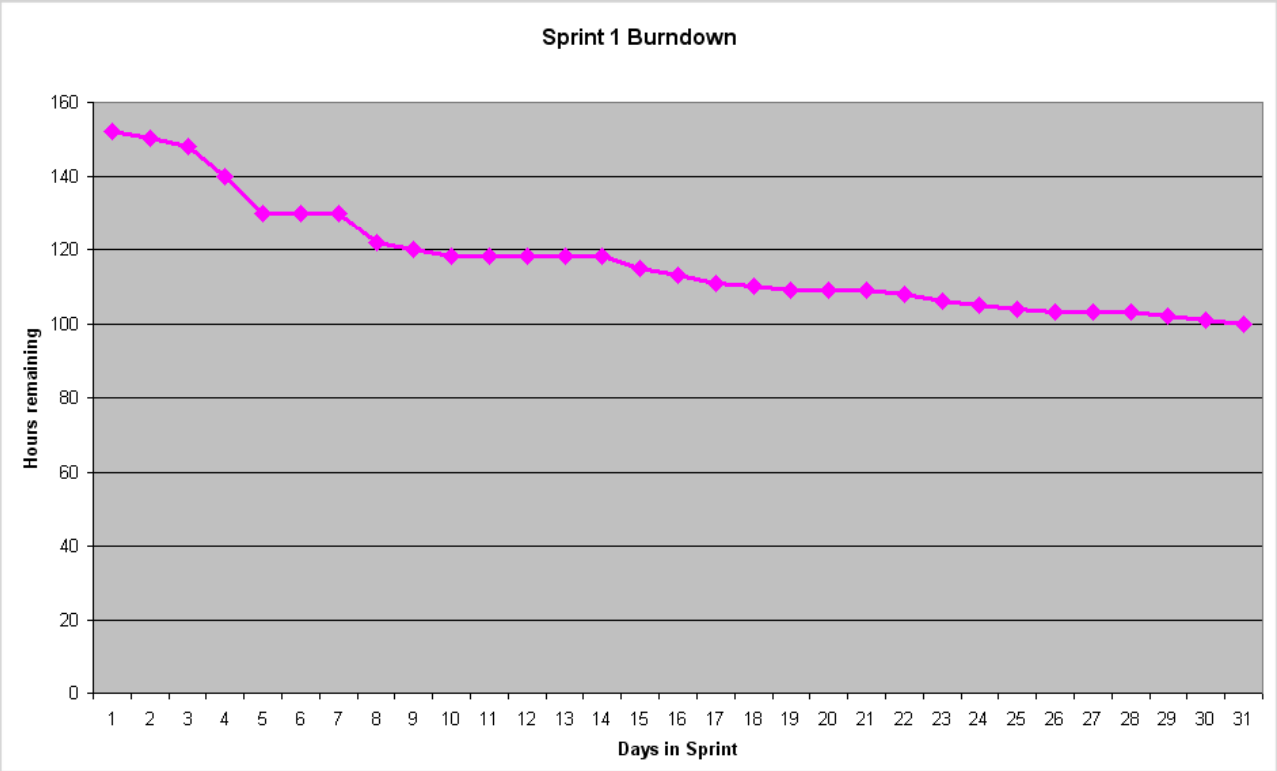



Figure 7 – Sprint Burndown after work has been performed, but not fast enough

Of course, Burndown Charts can reveal that the Sprint Backlog might finish early. For example, if a sprint does not contain enough work, you might end up with a burndown chart similar to figure 8. There are other reasons why a Sprint Backlog might appear to finish early:

Excessive working: if the Project Team works more that an 8 hour day (in this case), work might be completed "ahead of schedule". In reality, excessive working only ever appears to have short-term gain – the project will pay for it either in a loss of product quality and/or tiredness downstream resulting in loss of productivity.

Sprint Backlog item estimates may be incorrect: we may have to revisit our initial estimates as the work is being completed well within the existing estimates. This kind of exception can be caught if the Project Team know to announce the fact that the have completed a task ahead of schedule – the Sprint Backlog simply requires that a task is marked as having "0" hours remaining to indicate that it is complete.

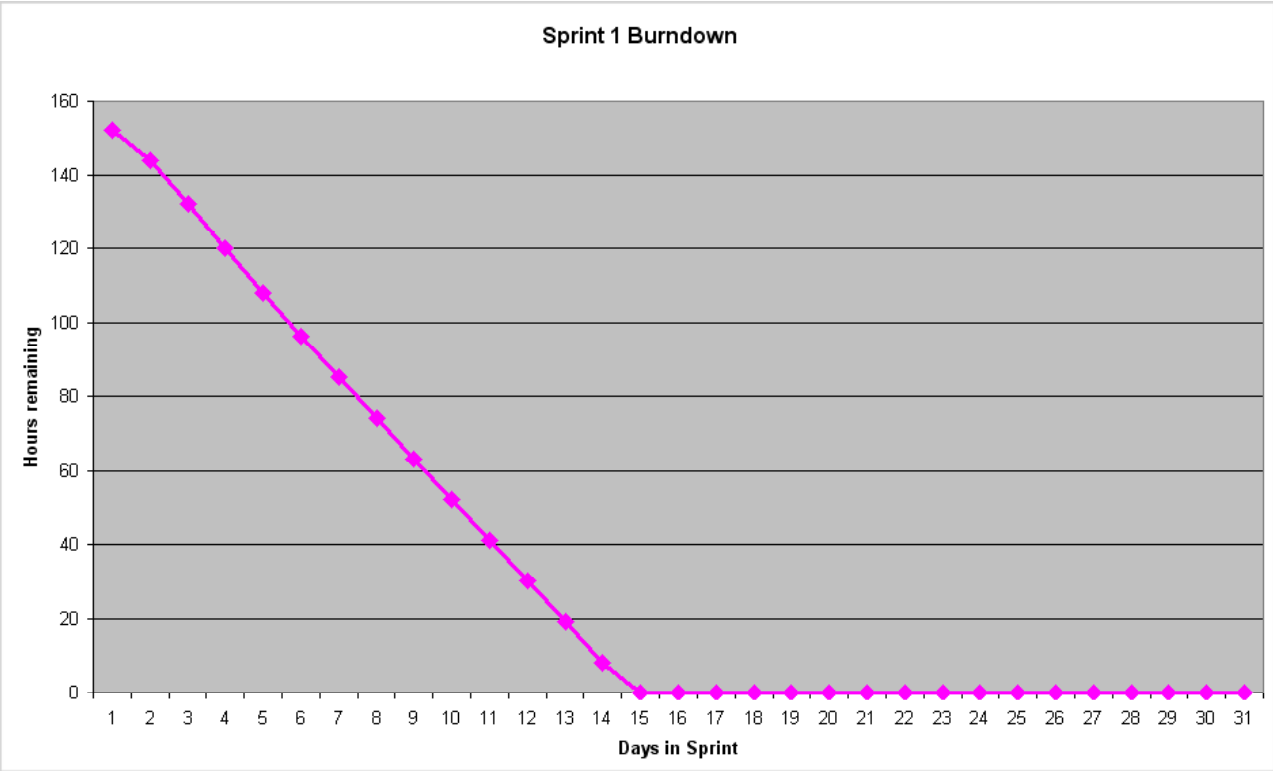



Figure 8 – Sprint Burndown after work has been performed, but too fast (not enough work)

In reality of course, we notice burndown charts that have their ups and downs whilst still meeting their target. If the ups or downs are dramatic, then alarm bells can ring indicating that something is wrong with either the sprint's scope or the sprint's Project Team. Either way, you'll get an early warning indicating that this sprint (less than 30 days of work) is about to encounter a problem. It's better to encounter and solve project issues as soon as they arise: Scrum's burndown charts put the project issues into context, better that you "lose" 30 days early in the project than you have to find time later in the project to fix mistakes made early on.

Sprint Backlogs and Burndown Charts are early warning indicators, they highlight "lack of progress". Similarly, they highlight scenarios where there is not enough work or work is too easy". However, their use does assume that everybody is committed to keeping the Sprint Backlog up to date and that is a job for the ScrumMaster.

With careful use, even these simple spreadsheets can assist in a number of other ways. For example, it is possible to extract "individual" Burndown Charts for each Project Team member. Whilst an overall Burndown Chart might highlight an overall problem with the project, using individual Project Team member Burndown Charts can help identify the one team member who is causing the problem!

Similarly, by allocating Sprint Backlog items to team members, we can respect their working time: ideally no employee should have to work more than a given number of hours per day (in this case 8).

**Why Iterative Development Works**

Consider figure 9, it presents an ideal profit and loss curve comprising of cost over time. Early in most projects, there is an initial hit, a start up cost and that's the period below the line.

After some time passes, typically a number of years, the project starts to make some profit and that's the period above the line.

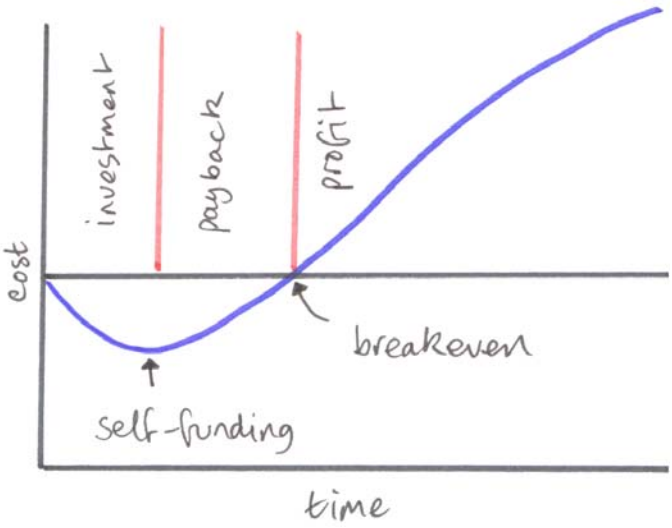



Figure 9 - Profit & Loss using "traditional" development (from Software by Numbers)

The crux is getting the project into profit as quickly as possible. Obviously there are a number of tricks that you can use to achieve this, some of which may involve cutting corners, reducing quality, reducing scope etc. Beware if you do use such techniques – early inducement of the profit position point can be a short-term solution that may affect future profit. Poor quality (of product, of support, etc.) can lead to a reduction in future sales.

Iterative development is a technique that allows a controlled inducement of the profit position: if we can release a product earlier on an iterative and incremental basis, we can realise the benefit of the early release much earlier on in the project. Consider figure 10, it presents a typical profit and loss curve where iterative development has been applied.

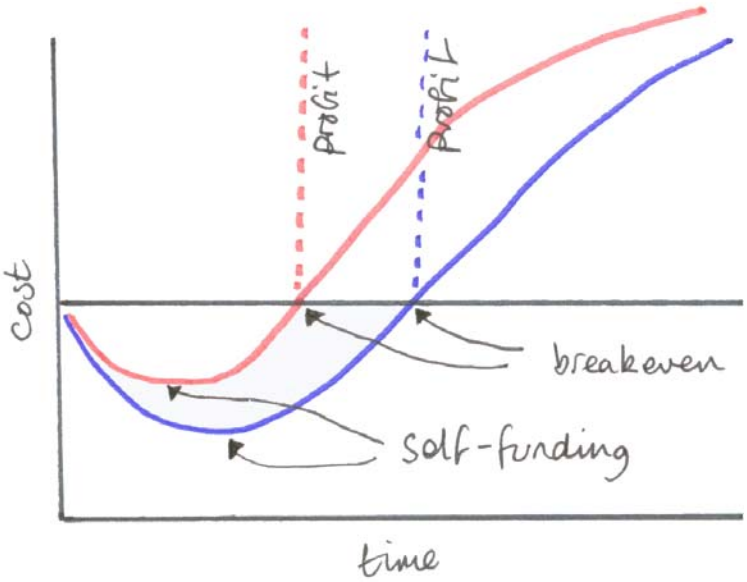



Figure 10 - Profit & Loss using iterative development (from Software by Numbers)

Iterative development means you are able to deploy your application sooner than before. The earlier you are able to deploy your application, the sooner you can reap the profit.

Scrum promotes iterative development: every 30 days you have a "potentially shippable product increment" (or executable). If you are able to move much of the useful functionality into early releases, not only are your customers going to appreciate early access, they'll be in a better position to confirm that they're getting what they asked for.

**Adapting Scrum**

I have enjoyed reasonable success by adapting Scrum's three questions. Early in 2004 I used the three questions in front of a small audience: they really appreciated the simplicity. They also appreciated the honesty that the questions seemed to portray: because of their simplicity, it's difficult to incorporate untruths into the answers to these questions.

I discovered that my upper managers wanted four questions answered:

1. Summary of work completed to date
2. Summary of work completed in the last 14 days
3. Plan of work for the next 14 days
4. Project issues requiring their action

Ignoring question 1 for now, it's clear that questions 2, 3 and 4 map directly on to Scrum's three questions: "what have you done?", "what do you plan to do?" and "what's getting in your way/stopping you working?".

If we keep track of our responses to Scrum's first question, "what have you done?", it lets us answer question 1, it gives us a summary of work completed to date. Indeed, whilst I have not been able to implement Daily Stand Up Meetings, because of geographical and diary time issues, their benefit has not gone unnoticed. I have had some buy in to "stand up" meetings, but not enough: we don't practice them right now.

**Summary**

I hope this article has provided you with a flavour of what Scrum is and how it can help you manage your next (or even current) project and help you deliver software incrementally instead of "big bang".

Keep asking these questions:

1. What is the simplest thing that can move the project forward?
2. Does what I am doing right now move the project forward at all?
3. Are there any impediments that are preventing progress?

I will let you think about figure 11. Where does your current/future project appear? Scrum lets us move our projects from "top right" towards "bottom left". Those projects that try to implement 100% of the requirements move themselves into anarchy; they try to do too much. By doing less work, accepting that 80% of a product's value does come from 20% of its features means you can move through complex and complicated towards simple.

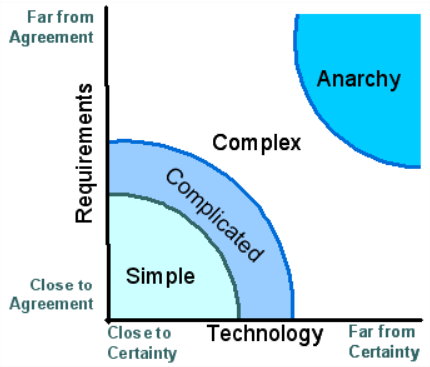



Figure 11 - Where's your project?

Lastly, if you take one thing away from this article, it has to be Scrum's co-founder, Ken Schwaber being quoted in April 2004, Vienna: "Don't procrastinate, do something, no matter how small…"

Scrum: the ethos of simplicity and the art of the possible.

**This article's resources**


1. Scrum: It's About Common Sense http://www.controlchaos.com
2. Slide Deck accompanying this article: http://www.craigmurphy.com/sd/WhyScrumWorks.zip
3. http://www.scrumalliance.org : Pay your £1, $1, €1 late-comer fee via this site!
4. Software by Numbers: Low-Risk, High-Return Development, By Mark Denne, Jane Cleland-Huang. Prentice Hall PTR, 2003, ISBN 0131407287
5. Agile Project Management with Scrum, Ken Schwaber, Microsoft Press, 2004, ISBN 073561993X
6. Agile Software Development with Scrum, Ken Schwaber, Mike Beedle, Prentice Hall, 2002, ISBN 0130676349

# Delivering Real Business Value using FDD

By Grant Cause, grant@itps.com.au, Managing Director,
IT Project Services Pty. Ltd. www.fddtracker.com

## Introduction

I run my own bespoke software development consultancy firm in Brisbane, Queensland, Australia. I first ran into Jeff De Luca the inventor of the Feature Driven Development methodology about a year ago where he was a guest speaker at a seminar organised by Borland.

I have been in IT for about 20 years now and have heard many people speak on software development methodologies from Waterfall to Extreme Programming. So I guess I was a bit jaded when I went into this seminar, thinking "oh no, not another methodology". To my surprise this wasn't to be the case. In just under an hour Jeff De Luca convinced me that FDD was the way of the future for all software development my firm would be conducting.

The reasons for this are many:

1. At its heart the methodology talks Business and delivering real business value; my firm has differentiated itself on this very point since we first started. So this instantly appealed to me.
2. FDD doesn't rely on gimmicks such as "pair programming" to get the job done and deliver real business results. Instead FDD was just formalising the use of the existing "best practices" we had been using for years. Don't get me wrong, I believe Pair Programming has its place in software development, I just believe it has been abused and its intent confused over time. It should not be overused but used where needed
3. FDD is a simple methodology, it only has five processes. It is easy to adapt and adopt the methodology to suit most projects. And despite its simplicity it is also quite thorough in its coverage of the software development lifecycle.
4. FDD is not some theoretical methodology. It comes from real life experience and has been proven in the field many times over, over many projects, over many years. Put simply ***IT WORKS***!

I certainly do not purport to be an expert in Feature Driven Development having used it for only about a year now. Nor am I an evangelist for FDD. Software development is not a religious experience despite what some people would have you believe. The purpose of this article is simply to introduce the methodology to you and to share my real world experiences in adopting this methodology for my own consultancy firm.

I will let you draw your own conclusions as to whether this methodology is right for you and your firm.

## What is Business Value?

Businesses these days are under extreme pressures from all sides. The pace of business has quickened thanks to advances in more efficient business practices and technology. Businesses don't have the luxury of time that they once had to run large projects and wait for the results to come out at the end. If they did this now, their businesses and their markets would have moved on and the results they were waiting for may no longer be meaningful or relevant.

Businesses are quite simply looking for one thing, ***RESULTS***! These results generally involve some monetary quantification i.e. they can be measured in real dollars. In general the results the businesses are looking for can be broadly categorised as ***PRAISED*** items as follows:

**P** roductivity gains
**R** educed cost
**A** voided cost
**I** ncreased revenue
**S** ervice level improvements
**E** nhanced quality
**D** ifferentiation in the marketplace

If a software development project can deliver on one or more of the PRAISED items then it will have delivered real business value back to the business as a whole. PRAISED items are not part of FDD - it is something I have learned over 20 years about delivering real business value back to businesses.

Many consultancy firms take the wrong approach to delivering real business value. They try and force a piece of technology onto a business with little or no understanding of the business as a whole and the impact that technology will have on the business. Some of this can also be self-inflicted by the business in its desire to gain a competitive edge and have the latest and greatest innovative new technology implemented within its firm without thinking through all the implications of doing so. We will see shortly how FDD avoids this classic mistake from happening.

Another common mistake many consultancy firms make is that they tend to talk technology to business people whom, because this is not their area of expertise, are unfamiliar with the language of technology and all its three letter acronyms. This tends to lead to a lot of confusion and frustration on both sides as they are unable to communicate effectively with each other and as a result fail to get the results they are both looking for. FDD has built in naming templates that avoid this breakdown in communication and allow both parties to use a common language.

As I stated earlier FDD instantly appealed to me as my firm prefers to take a business first approach also and talk business to business people and then technology to technical people once we fully understand the business and its unique business problems. As a result we often take on the role of the trusted advisor acting as the go-between between the business people and the technical people who find they can suddenly communicate in more meaningful ways and achieve results they never before thought possible. Now that we have qualified what we mean by business value we will now explore how FDD delivers on and relates to this.

**FDD and Business Value**

FDD has as its basic tenet that the features being delivered in a software development project must be "***client valued***". That is, they have to have some real tangible benefit to the business to be considered as part of the feature list for that project. Generally if it doesn't have some business case behind it, then it should not be considered as a feature. FDD also reflects business value in the way the feature list is categorised and broken down into a hierarchy of business activities. These business activities are sets of features ("client valued" items) that will be delivered by the project to the business.

This means that when a business person looks at the feature list everything on that list will have some meaning to the business as a whole. For a technical person one of the most frustrating things is to expend time and energy on building something only to be told by the business person "oh that's not what I wanted". Having the business people involved in helping to define the feature list of what is to be built helps avoid this situation.

FDD is an all inclusive methodology in that all the stakeholders in a project get involved in developing the feature list; that way there are no surprises for any of them when the thing is finally built. FDD is not only inclusive at the up-front design and planning stages but all the way through the software development lifecycle. The reporting mechanisms are for all involved.

The design is on-going and iterative involving everyone. By being inclusive the methodology ensures that real business value is delivered as everyone is firmly focused on and working towards the same set of goals.

FDD is also an agile methodology; remember what I said before about a business not being able to wait a long time for results. FDD defines its features as anything that can be built in 1-10 days effort. Anything larger than this is broken down further until it meets that rule. The emphasis is on as granular or fine-grained features as possible. Thus, most features are about 1-3 days of effort.

The FDD development cycle is dependent on how often you choose to do the release meetings - the checkpoint that collects dates and publishes all the reports, etc. We will typically meet with our development teams first and then meet with and report to our clients on a weekly basis. This depends on what we agree with both the client and the development team. We generally run a two week release cycle where we do a build of working software which we will then make available to the end-users for testing. Only the items that have made it to the FDD promote to build milestone will be included in the release.

From what I have observed most FDD projects typically use a two or three week development cycle. Because of this short development cycle the business gets its results quickly. At each steering committee meeting the business people get used to hearing the word “complete”. And when FDD mentions the word “complete” it really does mean complete - the feature is now ready to go into a businesses production environment. This is great for getting management buy-in into the methodology.

Being an agile methodology FDD has an emphasis on producing working software.

Usually traditional “waterfall lifecycle” projects have suffered from a fatal condition known as “analysis-paralysis”. I say fatal because research has shown that 80% of projects don’t actually get past the requirements definition stage. In the Chaos report published in 1995 by the Standish Group, 15% of the projects they studied were considered a success this only rose to 34% by 2003

In the past, they would wait and wait for wads of documentation to be completed before any real results are produced, consequently they would get frustrated by not being able to service their business needs and cancel the project and start again.

With FDD they start getting used to seeing working software instead of wads of documentation and as that working software is rolled out they start realising the business results and consequent business value they were really looking for in the first place.

**A Brief introduction to FDD**

***So what is this FDD thing all about anyway?*** I hear you ask.

Well, the following is a very brief introduction to FDD in layman’s terms.

Firstly I must state that FDD is no silver bullet (The Silver Bullet was first described in the book The Mythical Man month by Fred Brooks). In fact there are no silver bullets to delivering software development projects on-time, to-budget with agreed functionality. That takes hard work, patience, commitment and skill. FDD is a methodology, a good one, but just a methodology none the less. It is not designed to be followed rigidly or to replace good common-

sense. FDD does however require street-smarts to be able to decide where, how and to what extent to adopt and adapt FDD to your particular business.

***What is FDD?***

FDD stands for Feature Driven Development.

Feature Driven Development is a process designed and proven to deliver frequent, tangible, working results repeatedly. FDD is a straight-forward, no-nonsense, common-sense, industry proven approach to software development built around a well recognised core of industry best practices.

***Where did FDD come from?***

FDD was developed in the field; it first came to prominence in 1997 in Singapore on a banking project known as PowerLender where Jeff De Luca was the Project Manager. Jeff had a bunch of processes and practices he'd been using for years. Peter Coad, who was brought on to do an initial up-front modeling exercise on the project, introduced Jeff to the concept of a fine-grained feature. Jeff then took that concept and implemented it throughout his process framework.

Over time a lot of other people have contributed in some way or another to what is now known as FDD. However the main drivers behind FDD have been Jeff De Luca and Peter Coad, with Jeff being the original inventor and owner of the methodology.

***So why use FDD?***

To quote Jeff De Luca “To enable and enforce the reliable delivery of working software in a timely manner with highly accurate and meaningful information to all key roles inside and outside a project.”

FDD is a simple easy to understand and easy to implement methodology that:

- was developed in the field on real projects. It comes from practice and has been proven in practice; it was not a theory or a book first like so many other methodologies
- delivers frequent, tangible, working results repeatedly
- is highly iterative
- is built around a core set of industry best practices
- builds in quality rather than adding it as an afterthought
- provides highly accurate and meaningful progress and status information with minimal impact on the software developers
- is well liked by all project stakeholders, from the technical people through to the business people there is something in it for them all.

FDD makes use of a core set of industry best practices which include but is not limited to:

- Domain Object Modeling
- Developing by Feature
- Individual Class Ownership
- Feature Teams

- Inspections
- Regular Builds
- Configuration management
- High visibility/reporting of results

Domain Object Modeling is used extensively in FDD. The first process of FDD is in fact the development of an object model representing the problem domain for the project. Initially this model is just a straw man representing more shape than content but the details quickly start to be filled in as the project progresses. As each new feature is addressed, a small design phase ensures the details fall into place quickly so the build can proceed.

An example of this modeling technique is shown in Figure 1. Note the use of sticky post-it notes which get put on a wall to show the model. That way the model is dynamic it can be easily changed as new ideas and requirements come to mind. By keeping the modeling rules simple the business people can participate in the modeling exercise without having to know all the technical details.

For those who want to know the technical details, FDD makes use of some innovative modeling techniques. The most notable are the use of Colour and the use of Stereo Archetypes. Colour is used to differentiate between such things a Role (Yellow), a Party / Place / Thing (Green), a Description (Blue) and a MomentInterval (Pink) as shown here in Figure 1:

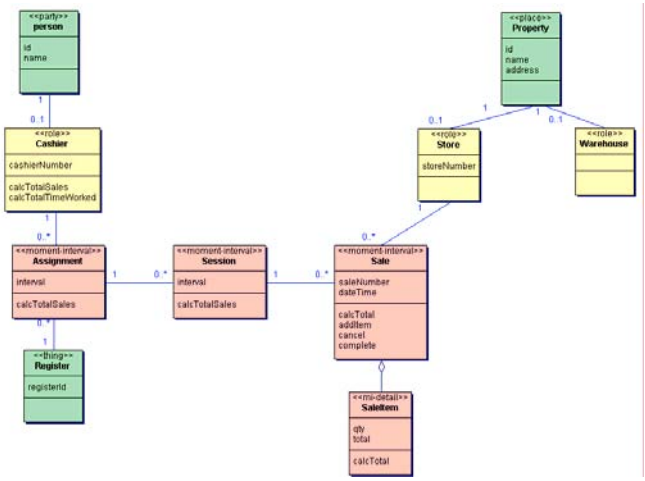



Figure 1 - Colour Modeling example / Copyright © 1997-2004 Jeff De Luca

Each of these is describing a Stereo Archetype. Stereo Archetypes are similar to the concept of Design Patterns but in an object modeling sense, they represent solutions to common problems that keep occurring in the real world. Jeff has put these together with a set of rules that govern them into what he calls the ADS or Archetypal Domain Shape. Almost any problem can be modeled by the use of the stereo archetypes contained in the ADS. Each one of these interacts with the other in a set way typically in the order of Blue->Green->Yellow->Pink. I wont go into all the details of what each of these is as that can get quite complex and is best left as the topic of a separate article.

FDD places emphasis on customer participation, by including the customer in the initial up-front and subsequent design sessions. The customer starts to buy into and take ownership for the project and the final project deliverables. For the developers this allows them the opportunity to see how and why things are done within the customers business and who and what interact with

the feature in question. It is also a great way of building a rapport between the customer and the development team. It also gives both the customer and the development team a new perspective on the issues faced by each of them during the project.

As the name suggests FDD is “Feature Driven”, meaning it makes use of the concept of a “feature”.

The term feature is very specific; a feature is a small, client valued function expressed in the form: <action> <result> <object>.

A feature is:

- Small; 1-10 days of effort are required to complete it. Most are 1-3 days. But they are designed and built in batches. The batch is the workpackage. A workpackage cannot take more than 10 days.
- Client valued; it is relevant and has meaning to the business; in business systems this usually relates to a step within some business activity within a business process.
- Named; <action><result><object> naming template with appropriate prepositions between them <action> the <result> <by|for|of|to> a(n) <object>

Feature examples:

- Calculate the balance of an account
- Validate the PIN number for a bank account
- Authorise a loan for a customer

The use of a naming template for the features helps when you are discussing these features with business people. The business people can understand this language and there is no technical jargon for them to have to decipher. They also understand that anything that makes it onto the feature list is going to have some real value to their business.

FDD places importance on “ownership”. It makes use of “class owners”. When a Chief Programmer (a role within FDD) schedules features for design and build, he puts them in a workpackage. A Workpackage cannot run longer than two weeks but they could be less. At a single point in time there will be multiple workpackages active, each at different stages.

Each feature is assigned to a team member who then becomes a “class owner”; this means they take ownership of that feature. Each feature the developer is given ownership of now forms part of their workpackage. It is now their responsibility to deliver that feature and in doing so become the expert on that feature. If another team member needs information on that feature this person is whom they would turn to first.

Teamwork is an important part of FDD. It has a concept known as feature teams. Feature Teams form dynamically. These teams represent people working on a single feature or group of like features. At any point in time a developer may belong to one or more feature teams collaborating as needed with other project team members within a feature team.

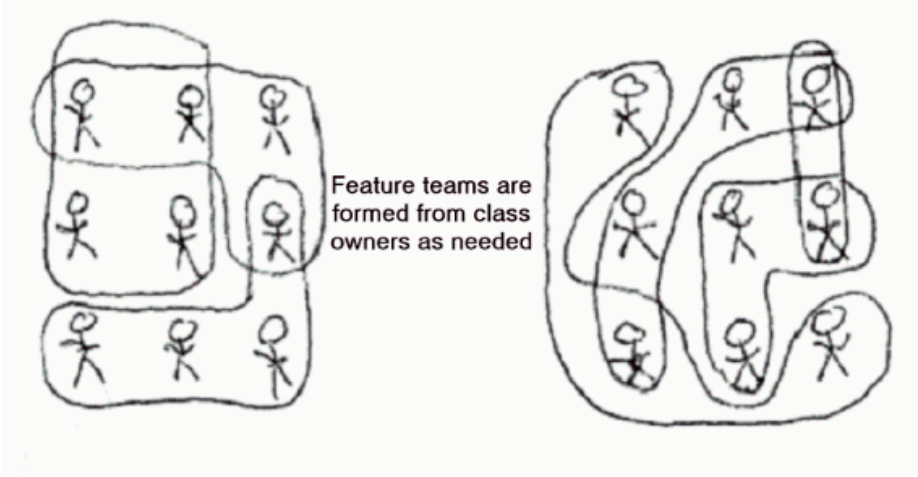



Figure 2 - Feature Teams / Copyright © 1997-2004 Jeff De Luca

FDD tries to prevent problems before they happen. One of the ways it does this is through the use of Inspections. Inspections have been proven over time to be one of the most highly effective yet under utilised methods for preventing errors in software. FDD has two types of inspections. The first is a design inspection. As designs are done they are inspected. This is more of a sanity check than anything to see if the team agrees with the chosen solution. If not the design is scrapped or modified until the team is satisfied with the chosen solution.

The second type of inspection is a code inspection. Code is chosen for inspection and checked for such things as adhering to coding standards, potential bugs or logic errors. Inspections are also great learning opportunities particularly if your teams are made up of a mix of both senior and junior developers. They can be a great opportunity to share knowledge across the team as well as help to identify potential problems before they happen.

FDD is a highly iterative and incremental methodology. One of the most powerful concepts of FDD is the production of regular builds of working software. This concept really represents the use of continuous integration that can prevent many problems that might normally only show up later in a project by identifying these problems early in the projects lifecycle so they can be corrected earlier.

Another powerful concept of FDD is high visibility and reporting of results. FDD provides a complete reporting mechanism for all stakeholders in a project as part of its process. The project reporting happens by default. You no longer have to worry about how you are going to collate all the necessary information in order to know if the project is on track. FDD will accurately record this information by default and provide a simple, easily understood reporting mechanism. To provide this reporting by default FDD makes use of a number of milestones within each feature and assigns each certain weightings as indicated in Figure 3. At its simplest you just sum the weightings for each of the completed milestones in order to get your percentage complete for that feature. So there you have it instant reporting.

| Domain Walkthrough | Design | Design Inspection | Code | Code Inspection | Promote to Build |
|---|---|---|---|---|---|
| 1% | 40% | 3% | 45% | 10% | 1% |

Figure 3 - Feature milestones / Copyright © 1997-2004 Jeff De Luca

The first three milestones above form what is known as the Design by Feature milestone and the last three milestones are known as the Build by Feature milestone.

FDD again makes use of colour to show progress status, White is not yet started, Blue or Yellow is work in progress, Red is needs attention (behind schedule, etc) and Green is complete. Each feature is colour coded to show its progress status on what is known as the Plan View report as shown in Figure 4. When the plan view reports are printed out they quickly give you a sense of how your project is progressing. These can be put up on a wall to give the big-picture view of the project from 30,000 feet. If you want the details on a particular area you can quickly zoom-in (i.e. move closer to the wall).

**Explode Design Model (19)**

| ID | Description | Walkthrough | | Design | | Design Inspection | | Development | | Code Inspection | | Promote to Build | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Plan | Actual | Plan | Actual | Plan | Actual | Plan | Actual | Plan | Actual | Plan | Actual |
| DP036 | add an Exploded Item to an Exploded Item List | 03/10/2000 | 03/10/2000 | 05/10/2000 | 06/10/2000 | 06/10/2000 | 09/10/2000 | 10/10/2000 | 10/10/2000 | 12/10/2000 | 12/10/2000 | 12/10/2000 | 12/10/2000 |
| DP037 | add a List of Exploded Item to an Exploded Item List | 03/10/2000 | 03/10/2000 | 05/10/2000 | 06/10/2000 | 06/10/2000 | 09/10/2000 | 10/10/2000 | 10/10/2000 | 12/10/2000 | 12/10/2000 | 12/10/2000 | 12/10/2000 |
| DP038 | iterate through an Exploded Item List | 03/10/2000 | 03/10/2000 | 05/10/2000 | 06/10/2000 | 06/10/2000 | 09/10/2000 | 10/10/2000 | 10/10/2000 | 12/10/2000 | 12/10/2000 | 12/10/2000 | 12/10/2000 |
| DP039 | explode the Design Items for a Design Product | 03/10/2000 | 03/10/2000 | 05/10/2000 | 06/10/2000 | 06/10/2000 | 09/10/2000 | 10/10/2000 | 10/10/2000 | 12/10/2000 | 12/10/2000 | 12/10/2000 | 12/10/2000 |
| DP040 | determine the list of valid Design Models during an interval for a Design Product | 03/10/2000 | 03/10/2000 | 05/10/2000 | 06/10/2000 | 06/10/2000 | 09/10/2000 | 10/10/2000 | | 11/10/2000 | | 12/10/2000 | |
| | | REMARKS: blocked on domain expert response | | | | | | | | | | | |
| | | REMARKS: Still blocked on [redacted] | | | | | | | | | | | |
| DP041 | determine if a Design Model contains any complex Design Items | 03/10/2000 | 03/10/2000 | 05/10/2000 | 06/10/2000 | 06/10/2000 | 09/10/2000 | 10/10/2000 | 10/08/2001 | 12/10/2000 | 12/10/2000 | 12/10/2000 | 12/10/2000 |
| DP042 | determine if a Design Item is complex | 03/10/2000 | 03/10/2000 | 05/10/2000 | 06/10/2000 | 06/10/2000 | 09/10/2000 | 10/10/2000 | 10/10/2000 | 12/10/2000 | 12/10/2000 | 12/10/2000 | 12/10/2000 |
| DP043 | explode the Design Items for a simple Design Model | 03/10/2000 | 03/10/2000 | 05/10/2000 | 06/10/2000 | 06/10/2000 | 09/10/2000 | 10/10/2000 | 10/10/2000 | 12/10/2000 | 12/10/2000 | 12/10/2000 | 12/10/2000 |
| DP044 | explode the Design Items for a simple reversible Design Model | | | | | | | | | | | | |
| DP045 | explode the Design Items for a complex Design Model with one complex Design Item | | | | | | | | | | | | |
| DP046 | list the Design Models for a complex Design Item | | | | | | | | | | | | |
| DP047 | explode the Design Items for a complex Design Model with two complex Design Items | | | | | | | | | | | | |
| DP048 | select each combination of a Pair of Design Model Lists | | | | | | | | | | | | |
| DP049 | generate an Exploded Item List for a combination selection | | | | | | | | | | | | |
| DP050 | record the Exploded Item List for a Design Model | | | | | | | | | | | | |
| DP051 | recall the Exploded Item List for a Design Model | | | | | | | | | | | | |
| DP052 | explode the Design Items for a complex Design Model with multiple complex Design Items | | | | | | | | | | | | |
| DP053 | iterate through each combination of multiple Design Model Lists | | | | | | | | | | | | |
| DP054 | select each combination in user specified order of multiple Design Model Lists | | | | | | | | | | | | |

Progress sum for these features in business activity "Explode Design Model": **39%**
Expected completion date: **Feb 2001**

Figure 4 - Plan View report / Copyright © 1997-2004 Jeff De Luca

Another innovative reporting mechanism is the summary reporting through the use of the car park or parking lot report. The business activities are shown as if they were the lines in a car park on the concrete with a progress bar and other summary information. Again the same colours the plan view report uses are used to quickly highlight progress status information see Figure 5.

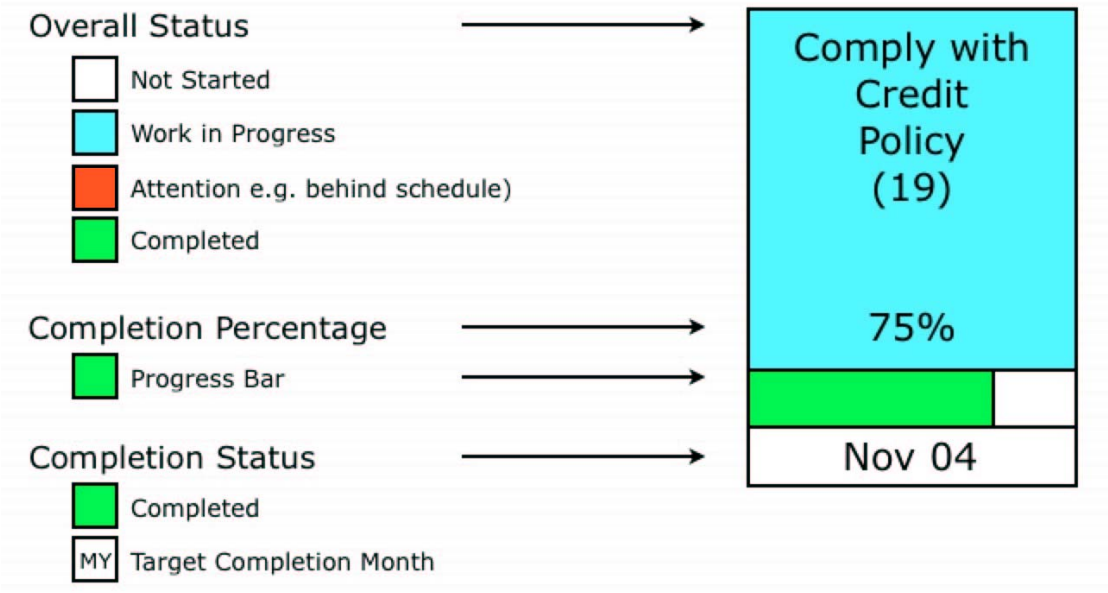



Figure 5 - Car park or Parking Lot report / Copyright © 1997-2004 Jeff De Luca

FDD lends itself readily to the use of repositories or knowledge bases of information and the publishing and dissemination of that information which ensures high visibility and reporting of results. These range from simple pen and paper solutions through to enterprise level solutions where all project stakeholders can see the project progress reports.

FDD consists of just five processes as follows:

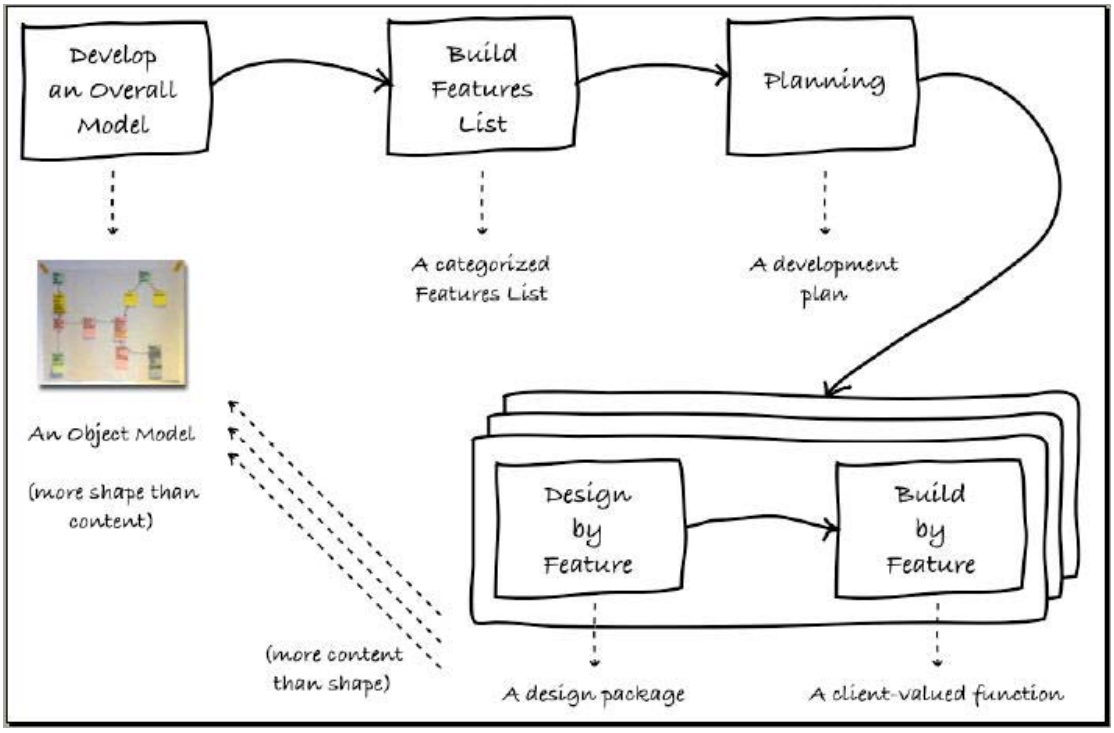



Figure 6 - FDD Model / Copyright © 1997-2004 Jeff De Luca

As can be seen in Figure 6, FDD has two phases - a start-up phase and a construction phase. Each phase has a series of processes associated with it and each process has one or more deliverables associated with it. The start-up phase is short and once off whereas the construction phase is made up of many short iterations (between 1 and 10 days) which together may last many months.

The start-up phase consists of the first three processes.

The first process is to develop an overall model, here technical people work in collaboration with domain experts (usually business people) to derive the shape of the system to be built.

The second process uses the information derived from this modeling activity the team produces a categorised list of features. The categories used are:

Subject Area -> Business Activity -> Feature

In the third process a plan is derived from the features list and features are assigned to class owners who then take responsibility for them through to their implementation. When planning an FDD project the basic rule of thumb is that for every week spent in the up-front design process there will be 3 months of development time required to build the system.

Once the planning process is complete there is enough information for the business to make a go or no-go decision on the project.

By conducting these first three processes the business now has a low-cost, high-value, low-risk assessment as to whether there is a real business case to develop a solution to solve a particular business problem. These first three processes have also taken a very short amount of time to achieve - typically weeks. With all this information in mind the business can now make a well informed decision as to whether to proceed with the project or not.

The construction phase consists of two processes.

Design by Feature and Build by Feature, as their names suggest means there is an initial design process followed by a build process. It is in this design stage that the details of the object model are fleshed out in collaboration with the customers' domain experts. Once all the details have been identified a design is finalised and then inspected and once passed the solution is built and then inspected again before being promoted into the build.

Note the term "promoted to the build", a feature is not considered complete until it has been promoted. That is it is now working software ready to be tested by the customer. It is not half-working, it is not nearly there, there is nothing whatsoever left to do on this feature except for the user to test it.

When FDD says a feature is complete it means it. Developers tend to have a problem with this concept, it is too black and white for them. If you ask a developer if he has finished a particular feature you might get some answer like "Yes its finished, but I just have to do …" this answer means it is incomplete - it will not be promoted to the build. Once the developers get through the first few iterations they quickly realise what the word "complete" means and what it really means to promote something to the build.

By having this "promote to build" milestone FDD reduces the risk of a project failure by ensuring that only working software is ever produced and provided to the business.

**Conclusion**

I know this has only been a cursory look at FDD but even though it is a simple methodology with only 5 processes the depths and nuances that it contains can be quite complex to describe. Indeed books have been written on FDD and even they do not cover all the details. This isn't surprising given that FDD is drawing on 30 years of industry experience and best practices and Jeff De Luca's own insights into how to make best use of them.

I hope this article has given you some insight into its value as a methodology that truly does deliver real business value.

How FDD delivers real business value that can be broadly categorised via the PRAISED items I mentioned earlier.

| PRAISED ITEM | HOW FDD ACHIEVES THIS |
|---|---|
| **P** roductivity gains | Short release cycles with short iterations that deliver working software.<br><br>Focused development efforts through the use of features and workpackages. |
| **R** educed cost | Higher chance of project success by involving the customer in every aspect of the design.<br><br>Being able to develop business systems in a way that allows the business to quickly react to changing conditions in the market. |
| **A** voided cost | By finding problems early through inspections.<br><br>The high cost of "Analysis Paralysis" is avoided.<br><br>Project success rates increase avoiding the cost of failed projects which have to be restarted or abandoned. |
| **I** ncreased revenue | Having business systems delivered to the business in a timely manner allows the business to get on with its core business and react quickly to market trends. |
| **S** ervice level improvements | Having good systems enables the business to better service its customers.<br><br>Employees enjoy a working environment where business systems support their ability to perform their jobs. |
| **E** nhanced quality | The use of inspections means the quality of the business solutions delivered to the business increases. |
| **D** ifferentiation in the marketplace | Being able to deliver on promises to customers by having business systems that support the core capabilities of the business in a timely manner. |

I said at the beginning of this article that I will let you draw your own conclusions as to whether this methodology is right for you and your firm.

All I can do is tell you that it has worked for us. Since adopting this methodology we have had a number of successful projects delivered on-time, to-budget with agreed functionality.

Although we had of course done this in the past, we had never done it with such ease as we have experienced when using FDD. In the past it was always a constant struggle to collate all the information to produce reports for our clients, in some cases it would literally take days to produce these status reports. Now it takes us only a few seconds to print out these reports based on the data we have constantly kept up to date in our FDD knowledge base product.

Since adopting FDD we have had clients compliment us on how well our projects have been run and in particular how we have been able to engage everyone from their organisation in their projects. Our clients have told us how pleased they are with the finished results and how sure they are that the finished results will deliver real business value to them for many years to come.

For my business FDD is definitely the way to do projects. We have only just begun implementing the methodology. There are many more techniques and best practices it contains that we can adopt and adapt that will enhance our ability to deliver real business value to our clients.

And in the end anything that allows us to deliver real business value to our clients is welcome in my business. After all if our clients are doing well, then so are we. FDD has certainly been a welcome addition to our business.

For further reading on this subject I would suggest the following web pages:

| Web Page | Contents |
|---|---|
| www.featuredrivendevelopment.com | The FDD community webpage. It has discussion forums, resource pages and further reading. |
| www.nebulon.com | Jeff De Luca's company home page provides details of the FDD processes as well as training courses and further reading. |
| www.fddtracker.com | My company's home page. The support link provides knowledge base articles that explain certain aspects of FDD. |

Thank you for taking the time to read this article. Your feedback is always welcome.

**References**

Jeff De Luca is the original inventor of FDD he runs his own consultancy based in Melbourne, Australia see www.nebulon.com for more details and further details on FDD. Jeff has kindly allowed me to use some of his artwork for this article and has provided some editing of this article.

Advertising for a new Web development tool? Looking to recruit software developers? Promoting a conference or a book? Organizing software development training? This space is waiting for you at the price of US $ 30 each line. Reach more than 35'000 web-savvy software developers and project managers worldwide with a classified advertisement in Methods & Tools. Without counting the 1000s that download the issue each month without being registered and the 9500 visitors/month of our web sites! To advertise in this section or to place a page ad, simply visit www.methodsandtools.com/advertise.html and send an e-mail to contact@methodsandtools.com

**METHODS & TOOLS** is published by **Martinig & Associates,** Rue des Marronniers 25,
CH-1800 Vevey, Switzerland Tel. +41 21 922 13 00 Fax +41 21 921 23 53 www.martinig.ch
Editor: Franco Martinig ISSN 1023-4918
Free subscription on : http://www.methodsandtools.com/forms/submt.php