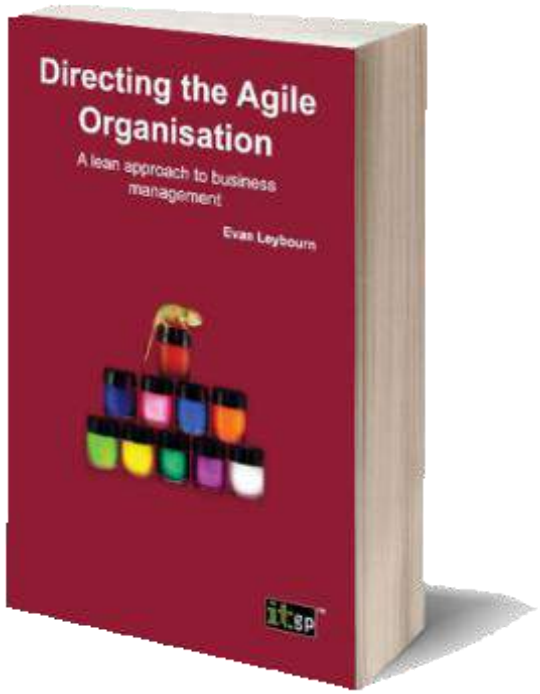
Introduction to Scrum

Student Guide

Evan Leybourn
evan@theagiledirector.com
Twitter: @eleybourn



OTHER WORKS BY EVAN LEYBOURN

DIRECTING THE AGILE ORGANISATION – BY EVAN LEYBOURN

http://theagiledirector.com/book

- Embrace change and steal a march on your competitors
- Discover the exciting adaptive approach to management
- Become the Agile champion for your organisation

Business systems do not always end up the way that we first plan them. Requirements can change to accommodate a new strategy, a new target or a new competitor. In these circumstances, conventional business management methods often struggle and a different approach is required.

Agile business management is a series of concepts and processes for the day-to-day management of an organisation. As an Agile manager, you need to understand, embody and encourage these concepts. By embracing and shaping change within your organisation you can take advantage of new opportunities and outperform your competition.

Using a combination of first-hand research and in-depth case studies, Directing the Agile Organisation offers a fresh approach to business management, applying Agile processes pioneered In the IT and manufacturing industries.

TABLE OF CONTENTS

WHAT DOES AGILE MEAN?

'On two occasions I have been asked, "Pray, Mr Babbage, if you put into the machine wrong figures, will the right answers come out?" [...] I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question.'

Charles Babbage, 1864

Notes:

The Agile Manifesto

The “Agile Software Development Manifesto” was developed in February 2001, by representatives from many of the fledgling “agile” processes such as Scrum, DSDM, and XP. The manifesto is a set of 4 values and 12 principles that describe “What is meant by Agile".

The Agile Values

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

The Agile Principles

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time-scale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximising the amount of work not done – is essential.
11. The best architectures, requirements, and designs emerge from self-organising teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

Notes:

Agile Methods

The term Agile actually refers to a concept, not a specific methodology. There are many, and sometimes conflicting, methods that can be used under the Agile umbrella. These include;

- Agile Unified Process,
- Behaviour Driven Development (BDD),
- Crystal Clear,
- Dynamic Systems Development Method (DSDM),
- Extreme Programming (XP)
- Feature Driven Development (FDD),
- Kanban
- Lean Development,
- Rapid Application Development (RAD),
- IBM - Rational Unified Process (RUP),
- Scrum,
- Test Driven Development (TDD),

Key Points

All of the above methods have four key points in common.

1. Iterative design process
2. Continuous stakeholder engagement
3. Aims for quality and reliable software
4. Short development cycles (up to a month) allows to regular delivery of software

This shows that an Agile approach is appropriate in contexts where the outcomes are not known (or can't be known) in advance and where the delivery of the outcomes cannot be fully controlled.

Notes:

The following figures[1] are an excellent example of the differences between traditional (or phased) software development vs. the Agile approach of iterative development.

FIGURE 1: THE TRADITIONAL APPROACH (PHASED DELIVERY OF KNOWN OUTPUTS)

FIGURE 2: THE AGILE APPROACH (ITERATIVE DELIVERY TO MEET CHANGING EXPECTATIONS)

UNDERSTANDING WASTE

The techniques and frameworks within Agile aim to increase development efficiency, by eliminating all 'wasteful' processes. Drawing on the successful concepts from the Lean manufacturing frameworks, we can define 3 major forms of waste.

- **Mura (Unevenness):** Mura exists where there is a variation in workflow, leading to unbalanced situations, most commonly where workflow steps are inconsistent, unbalanced, or without standard procedures.

[1] Images with thanks from Jeff Patton: http://www.agileproductdesign.com/

Notes:

- **Muri (Overburden):** Muri exists where management expects unreasonable effort from personnel, material or equipment, most commonly resulting from unrealistic expectations and poor planning.
- **Muda (Waste):** Muda is any step in the production workflow that does not add direct value to the Customer. The original seven wastes, as defined by the Toyota Production System (TPS), were:
    1. Transport,
    2. Inventory,
    3. Motion (moving more than is required),
    4. Waiting,
    5. Overproduction,
    6. Over Processing (from poor design), and
    7. Defects (the effort involved in inspecting for, and fixing, defects).

Critical Success Factors

The successful application of an agile methodology depends on the relative maturity of an organisation in relation to Customer Engagement, Staff Resources, Technology, and Processes. These measures are defined as follows:

- Customer Engagement – Product owners involved in teams daily activities, defines requirements, drives the prioritisation of requirements, and has decision making delegation of authority.
- Staff – have experience in an agile method, are skilled in the Standard Operating Environment (SOE) toolsets, have an understanding of the underlying data and technical infrastructure, and are conversant in the development, testing, and configuration and release procedures.
- Technology – a stable and well documented technology stack, with clearly defined ownership and service levels, providing discreet development, testing and release environments that are sized and supported for the delivery of projects, and controlled through rigorous configuration and release management.
- Processes – business processes exist for all domains, with cross stream interdependencies defined and service levels agreed, and clear business ownership and delegations of authority identified.

Notes:

COMMON MISCONCEPTIONS

Being a generic term, Agile means different things to different people. Therefore, before we go much further, I should clarify some of the more common misconceptions surrounding Agile.

- **Agile is ad hoc, with no process control:** First of all, Agile isn't a lack of process. Agile provides a range of formal processes, and methods, to inform work processes, customer engagement and management models. Conversely, Agile isn't about blindly following the prescribed 'agile' methods and processes. Agile is about using your common sense to apply processes, as determined by the current situation, and shaped by the agile philosophy.
- **Agile is faster and/or cheaper:** Agile isn't significantly faster, or cheaper, than alternative frameworks. Put another way, in most cases you can't get significantly more effort out of your Teams by moving to an agile approach. While there is an overall efficiency gain when utilising agile methods, well-managed Agile and non-Agile Teams will deliver products and services in approximately the same time and effort.
- **Agile teams do not plan their work or write documentation:** Agile is not an excuse to avoid appropriate planning or writing documentation. It is an on-demand, or Just-In-Time, approach that encourages continuous planning and documentation, but only when needed for specific Customer Requirements. This allows Customers and Teams to determine if the planning, or document, adds value to the process or product. It creates an opportunity to emphasise valuable documents, and eliminate anything that isn't useful.
- **An Agile project never ends:** While this may be true in some situations, the benefit of Agile is that work will continue while the Customer continues to gain business value, and that value is worth more than the cost of developing it. Most projects, in any industry, have a point of diminishing returns. This is the ideal time for an agile project to end.
- **Agile only works for small organisations:** Agile works for projects, teams and organisations of any size, not just small projects. That is not to say that it will work for all organisations, but size is rarely a factor. Large and complex projects and organisations are often excellent candidates for Agile transformation, where it is difficult, or impossible, to know all your Customer's Requirements in advance.

Notes:

- **Without upfront planning, Agile is wasteful:** This assumes that your Customer knows the detail of all of their Requirements in advance. If this is true, then by all means, undertake comprehensive upfront planning. However, in reality this is rare, and usually leads to the greater 'waste' of having undertaken design and development work that was ultimately unnecessary. Agile Business Management encourages minimal upfront planning, ensuring everyone is working towards the same goal, and reduces the risk of miscommunication.
- Finally, **Agile is not the solution to all your problems.** It is a change in approach and culture that comes with its own set of benefits and issues.

SCRUM OVERVIEW

Scrum is described as a 'framework within which you can employ various processes and techniques', rather than a process, or a technique, for building products. The Scrum framework is primarily team based, and defines associated roles, events, artefacts and rules. The three primary roles within the Scrum framework are:

1. The product owner who represents the stakeholders,
2. The scrum master who manages the team and the Scrum process
3. The team, about 7 people, who develop the software.

Each project is delivered in a highly flexible and iterative manner where at the end of every sprint of work there is a tangible deliverable to the business. This can be seen in the following diagram.

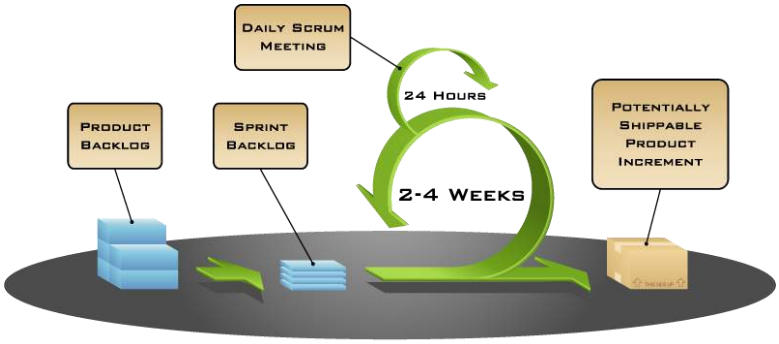


FIGURE 3: SCRUM FRAMEWORK

Notes:

The requirements that form the basis of the project are collated into what is called a Project Backlog, and is updated regularly. The features that are associated with these requirements are termed User Stories. This relationship is illustrated in the following diagram:

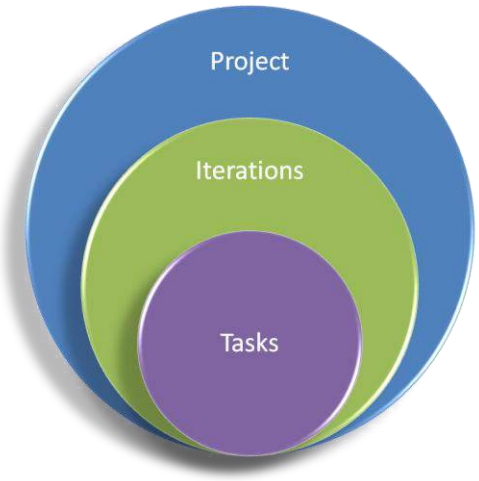


FIGURE 4: SCRUM PROJECT STRUCTURE

The work is time-boxed into a series of 1 to 4 week cycles where the business and project team estimate which User Stories in descending priority order are achievable each cycle, or Iteration. This subset of User Stories from the Project Backlog form the basis of the Iteration Backlog planned for delivery over that two week period.

Under Scrum, there are 3 timeboxed (or fixed duration) meetings held during an Iteration plus a daily stand-up meeting for the team, scrum master and (ideally) the product owner. At the beginning of a sprint, features to be developed during the sprint are decided during the sprint planning meeting. At the end of the Iteration are another 2 meetings, the Iteration review and Iteration retrospective where the team reviews the product and demonstrates the use of the software, as well as reflect on, and improve, the Iteration process itself.

After the sprint is complete, the next set of User Stories is selected from the Project Backlog and the process begins again. Burn rate is monitored to determine when funding will be exhausted.

Notes:

TABLE 1: KEY SCRUM CONCEPTS

Concept	Description
Project	Discreet set of end user requirements that have been grouped, prioritised and funded.
Requirement	The end user statement that outlines their information need.
Sprint	A sprint is a 1 to 4 week time-boxed event focused on the delivery of a subset of User Stories taken from the Project Backlog.
Project Backlog	The Project Backlog is the current list of User Stories for the Project. User Stories can be added, modified or removed from the Backlog during the Project.
Sprint Backlog	Subset of User Stories from the Project Backlog that are planned to be delivered as part of a Sprint.
User Stories	The User Story is a one or two line description of the business need, usually described in terms of features.
Tasks	Tasks are the activities performed to deliver a User Story.
Technical Debt	This refers to items that were either: • missing from the Planning meeting; or • deferred in favor of early delivery.

Notes:

PROJECT ROLES

'So Mr Edison, how did it feel to fail 10,000 times?'
'Young man, I didn't fail, I found 9,999 ways that didn't work'

Thomas Edison, anecdotal (on his invention of the incandescent light)

Notes:

PROJECT TEAM

The project team is a self governing group capable of independently delivering to a customer's requirements. As a result the team requires cross functional representation of skills and knowledge in the data, tools and infrastructure domains.

A typical project team can be comprised of the following:

- Product Owner
- Scrum Master
- Architects / Analysts
- Designers
- Developers

The project team characteristics and responsibilities are as follows:

- 7 ± 2 (5 to 9) resources who are allocated full-time to an Sprint
- Cross functional in nature across skills, applications, data and organisational knowledge
- Self empowered
- Responsible for delivering the product
- Determine the tasks required to deliver each feature
- Estimate the effort for each task
- Develop the features
- Resolve issues

Ideally the project team, including the product owner are co-located.

INTERESTED AND COMMITTED

Interested roles are individuals who have an "interest" in the software development. Whilst they should be kept informed of progress, they do not have the same level of responsibility and input into the development as committed roles. Interested parties include the Users, the Customers and the Product Owner.

Committed roles are responsible for the software development, and are the people who "do" the work. Committed parties include the Scrum Master, the Team and Testers.

Notes:

FIGURE 5: PIGS AND CHICKENS

PRIMARY ROLES

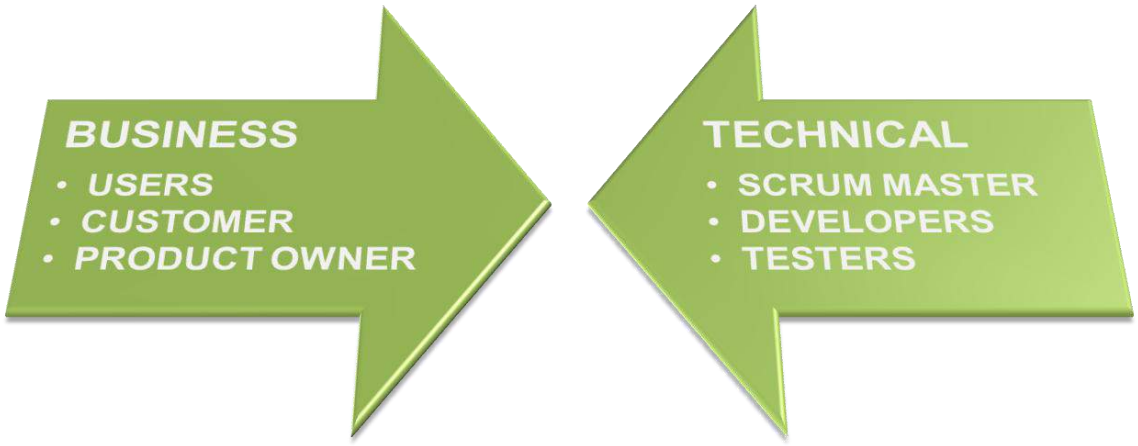


FIGURE 6: BUSINESS VS TECHNICAL ROLES

Notes:

TABLE 2: PRIMARY SCRUM ROLES

Role	Primary Responsibility	Typical	Does Not
Users *Interested Role*	• Use the software • Identify issues & • Provide feedback	• There are no typical users.	• Set Scope • Test Work
Customers *Interested Role*	• Define, start and end the project	• Internal managers • External Clients	• Direct Work
Product Owner *Interested Role*	• Manage the product backlog • Set the scope • Approve Releases	• Project Manager • Product manager • Customer	• Manage the Team
Scrum Master *Committed Role*	• Manage the Agile process • Report on progress	• Project manager • Team Leader • Team member	• Prioritise features
Developers *Committed Role*	• Develop features • Resolve issues	*cross functional* • Developer • Designers • Writers • Administrators	• Prioritise features
Testers *Committed Role*	• Test • Approve or reject features for release	• Existing developers • Dedicated testers	• Test their own code

Notes:

SPRINT 0

aka Feasibility

aka Project Initiation

aka Sprint 0 (XP)

'It is always wise to look ahead, but difficult to look further than you can see.'

Winston Churchill, ~1960

Notes:

Specifications in Agile?

Contrary to common opinion, it is very important to have a good specification before starting an agile project. By building this specification (or backlog in agile terminology) a project will;

- Reduce Risk & Uncertainty
- Improve Decision Making and integrate With Long term Goals
- Improved cost planning (including Staff Hiring)
- Prioritise research and information Gathering

Beginning the Process

Unlike traditional waterfall methods, the specification phase of an agile project is very short; usually no more than 1 or 2 days, and the full team should be available at inception. During the period the customer should be made fully aware of their role. The design should contain the following;

- Problem statement that needs to be addressed
- Desired business objectives, outcomes and benefits for this project
- Identified key stakeholders
- High Level Business Requirements
- Architectural and technical scope
- Testing requirements

Outcomes

The outcomes from Project Initiation, or Sprint 0, are:

- The team should be identified and brought together
- If not part of the team, identify and train the product owner
- Create the product backlog in low detail. Allow customers to slowly build the product requirements throughout the process.
- Estimate the product backlog.
- Plan length of the sprint, anywhere from 1 day to 4 weeks. Each sprint should accomplish something releasable. Short sprints short can reduce overtime.
- Add any team training to the backlog as tasks.

Notes:

BACKLOG

FIGURE 7: PRODUCT BACKLOG

The backlog contains all the User Stories (features) for the product. The Product Owner is responsible for defining the User Stories, and assigning each with a priority for delivery. The order may also be influenced by Story dependencies or business value e.g. a lower priority Story may need to be done first before a higher priority Story can be started.

The User Stories describe the set of features that will satisfy each Information requirement. The high priority User Stories that are candidates for the next Sprint require sufficient detail for the team to solution, and should be sized to fit within two weeks.

Each User Story should meet the INVEST characteristics, as defined by Bill Wake.

- Independent: Each Requirement should be as self-contained as possible, with minimal dependencies on any other Requirement. This allows for easy reordering or removal, as Customer Requirement's change.

Notes:

- Negotiable: The Customer can change a Requirement at any time, up to the point it enters the Sprint Backlog (see Chapter 4: Work, the Agile Way).
- Valuable: Each Requirement should deliver a tangible, and measurable, benefit to the Customer.
- Estimatable: The definition of each Requirement is such that the Team can estimate it.
- Small: The estimate, and delivery, of a Requirement, should be within a few days, or a single Sprint.
- Testable: Each Requirement should have appropriate quality control and quality assurance metrics, so the Customer can validate their Deliverables against the original Requirement.

Each feature should contain, at a minimum, the function, priority, an estimate of the effort to develop and the estimate risk (0 - 100%) based on how accurate the team feels the estimate is.

In can be helpful to structure each User Story in the following format.

As a [role]

I want a [goal/desire]

So that [benefit]

The [role] is the expected end-user of the User Story, who is usually different from the Customer. The [goal/desire] of each User Story describes what should be delivered, and the [benefit] provides the context, or the Why.

Notes:

Accuracy

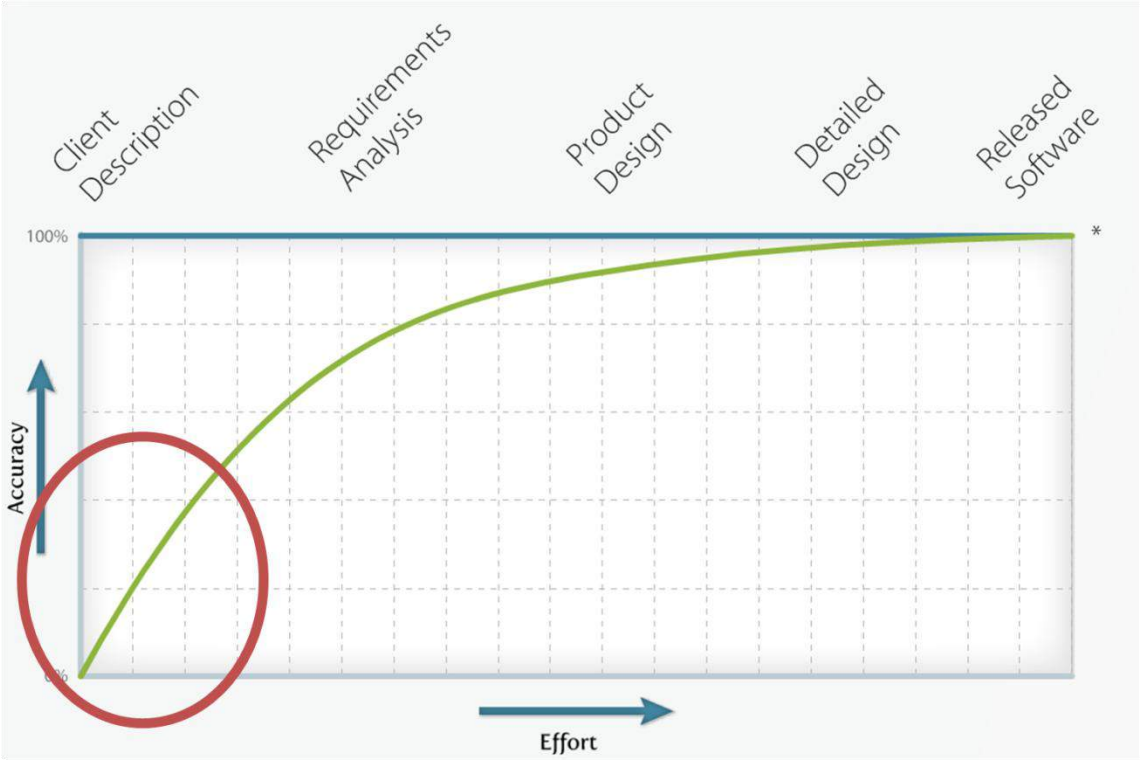


FIGURE 8: ESTIMATE ACCURACY OVER TIME

It is important to note that to increase the accuracy of any estimate, the effort involved increases exponentially. In Agile, we are only interested in the initial estimation.

By keeping Sprints short, we can better review and respond to deviations in the estimate quickly. See the Burndown chart chapter for more information.

Notes:

Estimating Effort

Story Estimation

A Story Point is single number intended to represent all of the work required for the whole team to build the final product. As a result, to perform estimation based on Story Points it requires cross functional representation of all the requisite skills and knowledge in the Business Intelligence Platform data, tools and infrastructure during the Sprint Planning meeting.

Story Points are used for the high level estimation of User Stories. They are usually based on the delivery of similar User Stories in previous Sprints. This assumes that the project team, data and infrastructure are relatively constant between one project and the next. The size of a Story Point for a given project team will eventually normalise over time.

All Stories should be assigned as estimated effort, or cost, to implement. We use a modified Fibonacci series, such as 1, 2, 3, 5, 8, 13, 20, 40, and 100, to represent effort. This encourages features to be split into the smallest task possible, and provides a more realistic estimate range.

Task Estimation

Task estimation is performed at the Sprint level. During the Sprint Planning session the team breaks the User Stories into their composite tasks to determine how they will be delivered. The process of solution decomposition may reveal additional tasks, or more complex tasks that were not apparent during the high level Story Point based estimation that impact what was planned to be delivered. The scope of the Sprint can be renegotiated with the product owner if this is the case. Unlike Story Points, these estimates are an indicative idea of how much time they will take in an ideal world.

How?

Tasks can be estimated in 4 ways.

1. **Expert opinion:** The team member with specific understanding, or who is most likely to develop the task, can provide a more accurate estimate of effort.

   E.g. A database administrator can better estimate effort for database tasks.

Notes:

2. **Comparison:** Comparing a task to another, already estimated, task.

   e.g. "Task A is about twice the effort of Task B"

3. **Components:** If a task is too large to accurately estimate, break it into small sub-tasks.

   e.g. User management can be broken into interface, login, ACL, etc.

4. **Planning poker:** If using Planning Poker, estimates must not be mentioned at all during discussion to avoid anchoring. A timer may be used to ensure that discussion is structured; any person may turn over the timer and when it runs out all discussion must cease and a round of poker is played.

   Each person lays a card face down representing their estimate of the task, and then simultaneously turns their cards over.

   People with high estimates and low estimates are given an opportunity to discuss their estimate before anyone else can speak.

   Repeat the estimation process until a consensus is reached.

FIGURE 9: PLANNING POKER CARDS

Notes:

ESTIMATING TIME

Converting an estimated cost into estimated time is very simple. There are 2 primary modifiers that we use. Staff overhead and estimate accuracy (or risk).

STAFF OVERHEAD

This is a percentage modifier for staff availability to work on specific project tasks. It allows you to take into account factors such as estimated leave, illness, breaks, scrum meetings etc. The industry standard modifier is 25%-40%, though you should modify this as required. To calculate staff overhead use the following process;

*working hours = (hours per day * days per sprint * staff) – planned leave*

project hours = sum of actual (from last sprint)

staff overhead = (working hours/project hours) – 1

CALCULATION

Story Cost x (Staff Overhead + 1) x (Estimate Risk + 1)

e.g.

4 x (25%+1) x (50%+1)

= 4 x 1.25 x 1.5

= 5 to 7.25 hours

Notes:

COST / TIME / SCOPE

"How much is this going to cost?" - "As much as you're willing to spend."

"How long is this going to take?" - "As long as it necessary."

"What am I going to get?" - "Whatever you tell us you want."

FIXED COST

Where a customer asks for a fixed price quote prior to agreeing to project commencement, but is flexible on what is delivered and how long it takes.

- Work in absolute customer priority order – reducing the time spent on technical helper tasks will help meet short-term budget constraints (at the cost of long term, post-project, efficiency)
- Release in short (1-2 week) sprints – similar to longer waterfall projects, longer sprints have a tendency to cost overruns to deliver on time
- Monitor velocity and burn rate – this is your key indicator of cost

FIXED TIME

Where a customer asks for delivery by a certain date, but is flexible in scope and cost.

- Work in absolute business value order – increases the number of user stories complete in a given sprint (high business value = simple, moderate-high priority)
- Enforce sprint duration – Your project will be defined by a fixed number of sprints, and extending an sprint will push out your final date

FIXED SCOPE

Where a customer asks for a fixed set of deliverables, but is flexible in the time it takes to deliver and the cost of delivery. This is sometimes known as "heavy agile".

- Focus on backlog definition and estimation during Sprint 0 to ensure accurate scope definition

Notes:

Fixed Cost and Scope

Where the customer asks for a fixed price quote for a fixed set of deliverables. In this situation, the final date for delivery is flexible. As well as the points in fixed cost and fixed scope;

- Increase the estimate risk during Sprint 0 – to ensure your quote for the project allows for unexpected delays (which would impact on your cost to deliver)
- Update delivery date as required

Fixed Cost and Time

Where the customer asks for a fixed price quote by a fixed time. In this situation, the exact set of features (or scope) for delivery is flexible. As well as the points in fixed cost and fixed time;

- Calculate total cost as cost per sprint – which makes your quote to the customer very simple.

Fixed Time and Scope

Where the customer asks for a fixed set of deliverables by a fixed time. In this situation, the total cost to the customer is flexible. As well as the points in fixed time and fixed scope;

- Pre-assign work to sprints during Sprint 0 – which will define the scope delivery timetable.
- Pad schedule with extra sprints – to cater to unexpected defects or technical debt
- Increase the size of the team 3-4 sprints prior to the end of the project if required – to ensure the set of features are completed in time.

Fixed Cost, Time and Scope

Where the customer gives no flexibility in the project.

Cancel the project – this is not an agile project. This should be run using a waterfall methodology such as PRINCE2 (and even they are likely to fail without some flexibility)

Notes:

STARTING AN SPRINT

'Make everything as simple as possible, but not simpler.'

Albert Einstein (paraphrased), 1933

Notes:

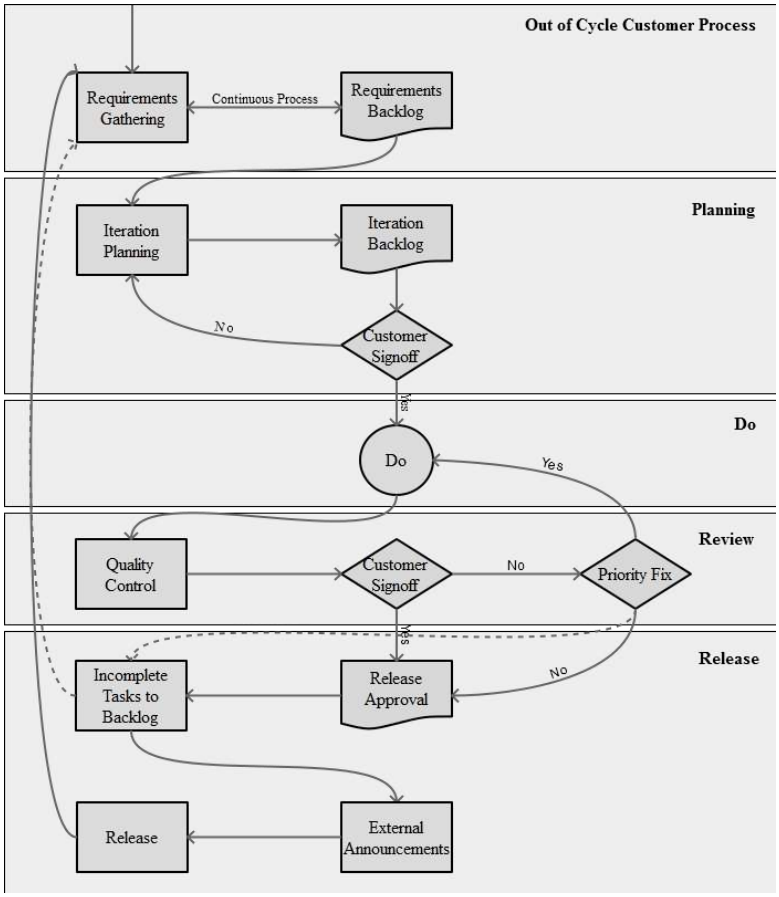


FIGURE 10: EXAMPLE BUSINESS PROCESS FLOWCHART

Notes:

SPRINT PLANNING MEETING

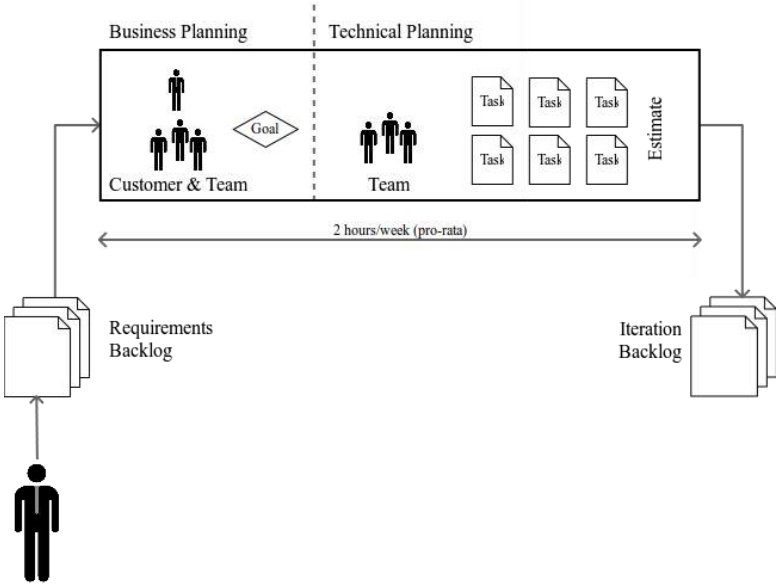


FIGURE 11: SPRINT PLANNING

The Sprint Planning Meeting is run before the start of each Sprint, and allows the customer and developers to discuss the requirements and work required for the next release. This step in the Scrum process focuses on determining the target scope of delivery for a Sprint, and defines the sprint backlog. The Sprint Planning Meeting should be no more than 8 hours long (4 weeks pro rata).

The Product Owner and Scrum Master are responsible for updating the Project Backlog in preparation for the Sprint Planning meeting. This includes clarification, prioritisation and in some cases investigation of the feasibility of the collated User Stories. This activity also needs to take into consideration any technical debt inherited from previous Sprints.

PART 1 – BUSINESS SPECIFICATIONS

The first part of the sprint planning meeting aims to convert features from the backlog into a realistic goal for this sprint. The product owner is a part of this process and sets the priority of the tasks. This also provides the Product Owner with the opportunity to communicate the required scope of delivery, provide the business context and priority, and address any questions the project team may have to assist with performing the solution

Notes:

decomposition and estimation steps. This part of the meeting should take no more than ¼ of the time.

Ensure that a copy of the Project Backlog has been distributed prior to the Sprint Planning meeting to provide the project team with time to consider solution options for discussion during the workshop, and prepare clarification questions for the Product Owner.

The participants for this session are as follows:

- Product Owner
- Scrum Master
- Team
- Testers

PART 2 – TECHNICAL SPECIFICATIONS

The second part of the Sprint Planning meeting is technical, and usually without the product owner. This is the solution decomposition and estimating step in the planning process and aims to estimate the effort for all features in the release and (optionally) write the test cases.

The general flow of activity in this step is described in the following diagram:

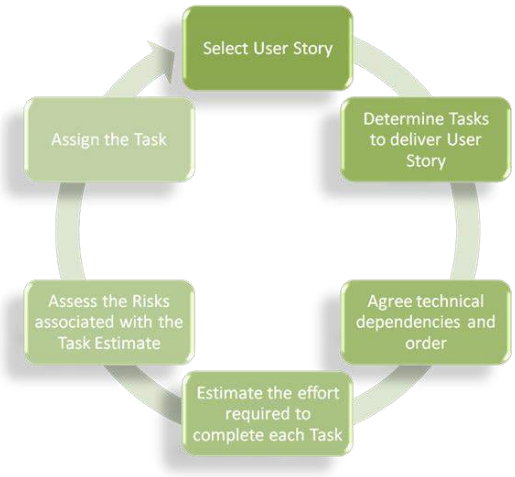


FIGURE 12: DESIGN AND PLANNING WORKFLOW

Notes:

As a guideline, large tasks should be broken into tasks preferably no longer than 1 day and tasks that involve waiting should be split into separate tasks. Research tasks should have a high estimate risk. This is done to enable accurate tracking and the calculation of velocity.

For complex User Stories or those with a high number of interdependencies it may be necessary to split the task decomposition and estimating activities across 2 days and allow the team members an opportunity to consult with external parties on feasibility and obtain input into the estimating process.

Always remember that tasks can be created, but features can't.

The participants for this session are as follows:

- Scrum Master
- Team
- Testers

TABLE 3: KEY PLANNING ELEMENTS

Planning Element	Description
User Story	The User Story is a description of the business need, usually expressed as a feature.
Story Identifier	Every User Story will be assigned a unique identifier for tracking purposes.
Task	A task is typically a single activity that can be described in one sentence that contributes to the delivery of a User Story. • Generally a task takes no longer than 4-8 hours of effort to complete • There may be one or many Tasks per User Story • The task can only be assigned to and owned by one person at a time
Task Identifier	A Unique identifier will be assigned to track each Task, and show which User Story they are associated with.
Project Function	This describes the architectural layer where the Task activity will be performed.

Notes:

Assignee	This is the person who will be responsible for delivering the task. This can done at any point in the Sprint. The person assigned to the completion of the Task may also change at any point in the Sprint.
Estimate	The estimate in hours is the amount of effort the team agrees is required to complete the specified Task. The estimate includes: • Analysis • Build • Unit Test • Migrate from DEV to TEST • Integration Testing • Documentation
Estimate Risk Modifier	This is a measure of the confidence level associated with the estimate provided and represented as a numeric modifier.

NOTES

- Prepare beforehand.
- This is a creative, problem solving process. Encourage brainstorming.
- Ensure the planning room has plenty of paper, a whiteboard and a computer with Google access.

Notes:

DURING AN SPRINT

'Treat your men as you would your own beloved sons. And they will follow you into the deepest valley.'

Sun Tzu, ~6th Century BCE

Notes:

DAILY LIFECYCLE

The daily lifecycle of team activities is as follows;

1. Team members select the next Task to work on
2. Undertake the task as described
3. Commit and share the completed task with the rest of the team
4. Write and run the tests that will be used to verify they have been completed successfully. Verification, unit testing and documentation need to be completed prior to migrating the deliverable from DEV to SIT.

The assignee for a Task may change at any of these steps. Team members will proactively interact will their colleagues and any internal parties as required to progress the assigned Task to completion, including any quality assurance and review.

The governance of the daily lifecycle is through the daily Scrum Meeting.

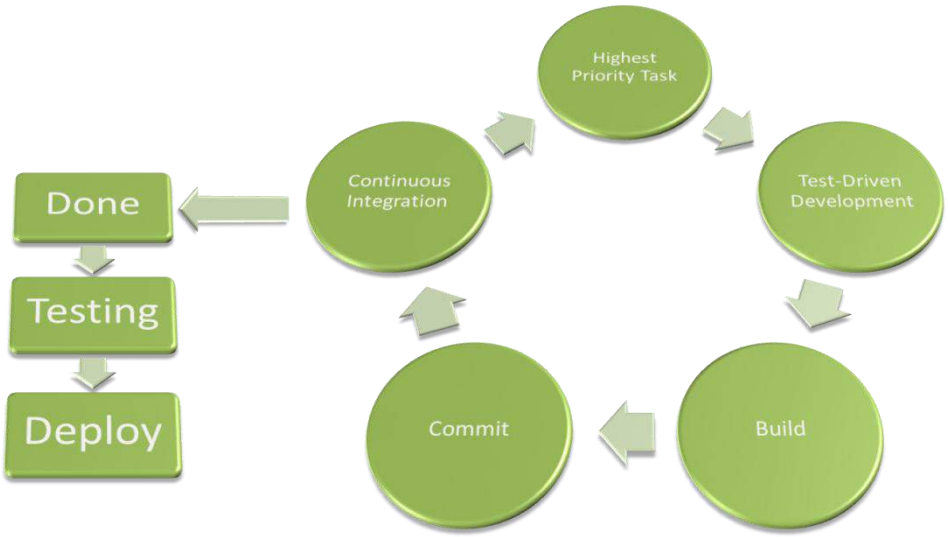


FIGURE 13: WORK LIFECYCLE

Notes:

TASK LIFECYCLE

Based on Kanban, a task will progress through a minimum of 4 different states during its lifecycle. Each task and state should be visible to the team, product owner and customer; commonly this is done through a card wall or integrated dashboard.

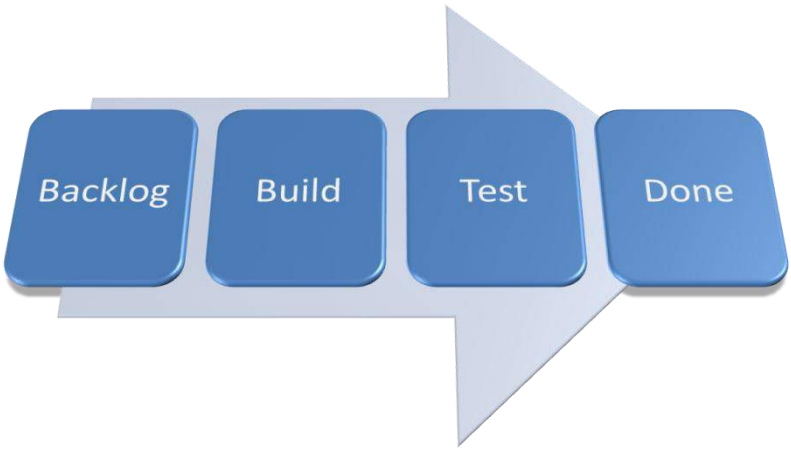


FIGURE 14: BASIC TASK LIFECYCLE

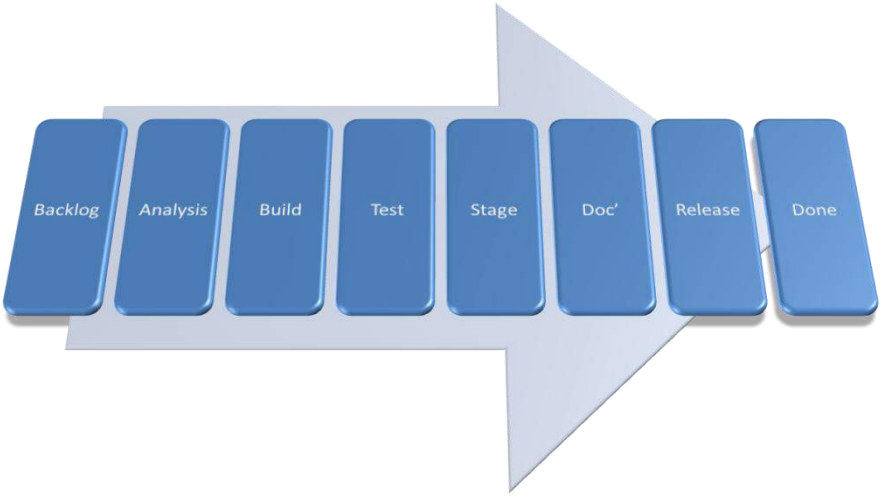


FIGURE 15: COMPLEX TASK LIFECYCLE

Notes:

Sprint Backlog

The Sprint Backlog is the subset of User Stories from the Project Backlog that the Product Owner and project team agreed would be delivered for this Sprint.

The User Stories will be broken into tasks which are put into the Sprint Backlog. Based on the logical sequencing of tasks and agreed prioritisation, the project team members select the next task to work on and promote this to the "In Progress" state.

In Progress

In Progress items are Tasks that are actively being worked on. This includes both development and unit testing activities. Once the task has been completed it is promoted to the "Testing" state.

In Progress Tasks include the following types of activity being performed:

- Analysis
- Build
- Unit Test
- Documentation

When a Task has been completed the deliverable will be changed to the "Testing" state. In the case of code based artefacts these will be promoted from the development environment to the test environment.

Blocked

Blocked items are Stories and or Tasks that have upstream or downstream dependencies external to the project team that are preventing progress. These impediments are moved to this holding state to highlight these issues to the Scrum Master and Product Owner for escalation and resolution.

Testing

Testing, in this context, is performed by the team's specialist testing resources. Unit testing is expected to be undertaken by the developers.

Done

Tasks are considered "Done" when:

Notes:

- Code has been produced, meets development standards, and has been checked in and run against current version in source control
- Unit tests written and passed
- System tested and passed
- Quality Assurance reviewed
- Builds without errors for deployment
- Relevant documentation, including diagrams have been produced or updated and communicated

When the Sprint has completed the User Stories selected by the team to be delivered are either "Done" or "Not Done". The decision over whether a User Story is Done is based on whether all the pre-requisite Tasks associated with this Story have been completed. The completed User Stories are presented to the Product Owner in the Sprint Review for acceptance and sign-off.

Not Done

Tasks that are "Not Done" are reviewed in the context of the User Stories that they belong to and if this impacts whether the User Story can be considered delivered. The not done tasks may be rolled into a new User Story for the next Sprint, accrued as technical debt, or it may be decided that they are no longer required and are removed.

Development Hints

Features

Get the highest priority feature from the Sprint backlog. Allow developers to choose their work, don't assign it. The backlog can be updated mid-sprint if critical changes are required.

Develop

Agile also makes some suggestions on improving the development process. These are;

- **Pair Programming:** Two developers working together, the first as a coder and the other as a reviewer. These roles should change regularly, and the pairs themselves should switch pairs each day
- **Code Standards:** A common coding style (Documentation, Names, Whitespace, etc)
- **System Metaphor:** All classes and functions should be named such that their purpose is understood.

Notes:

COMMIT

Everyone must commit every day, and should never commit broken code. (Continuous Integration)

TRANSPARENCY

Key to Agile is transparency between the product, the team and the customers.

Customers can:

- Attend scrums. However they should not talk. Questions should be directed to the Product Owner or Scrum Master. An alternative is to record the scrums and make the recording available to the customer.
- See the product and sprint backlog in its current state.
- See the state of each task via a card wall or integrated dashboard.
- Access a test version of the software from the development environment.

TEST DRIVEN DEVELOPMENT

KEY POINTS

Tests are written by the Customer and Developer together and are written before the code. Both automated unit tests and user acceptance tests should be written. There is no issue with using standards such as IEEE 829 and IEEE1008 to document and write tests.

By using TDD, the team can prove how well a project is going and how close to completion. This in turn, allows customers and product owner to make informed decisions about the project.

TEST COVERAGE

The tests should cover;

- Software functions,
- Boundary cases,
- User interface,
- User experience,
- Non-functional components,
- Performance

Notes:

Test Types

There are 4 types of tests that can be written.

1. Defect
2. Functionality
3. Usability
4. Data

TDD in Development

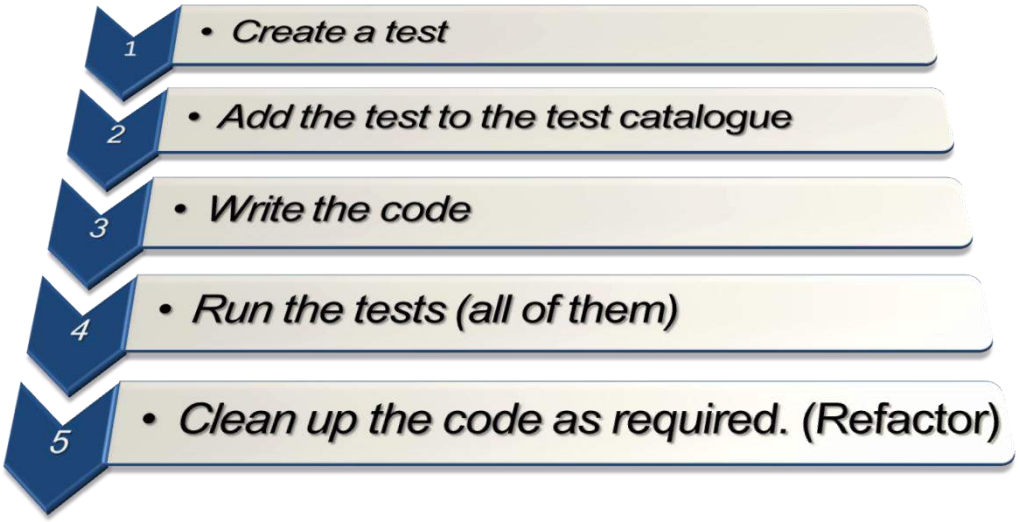


FIGURE 16: TDD WORKFLOW

Continuous Integration

Unit Testing

Runs predefined tests to identify software defects

- Create tests for each class and function
- Create tests for all parameter combinations
- Create tests for all edge cases
- Create tests to examine the database for logical errors
- Create tests to detect interface defects (Selenium)
- Tests should be kept in the version control repository
- Test in a clone of the production environment

Notes:

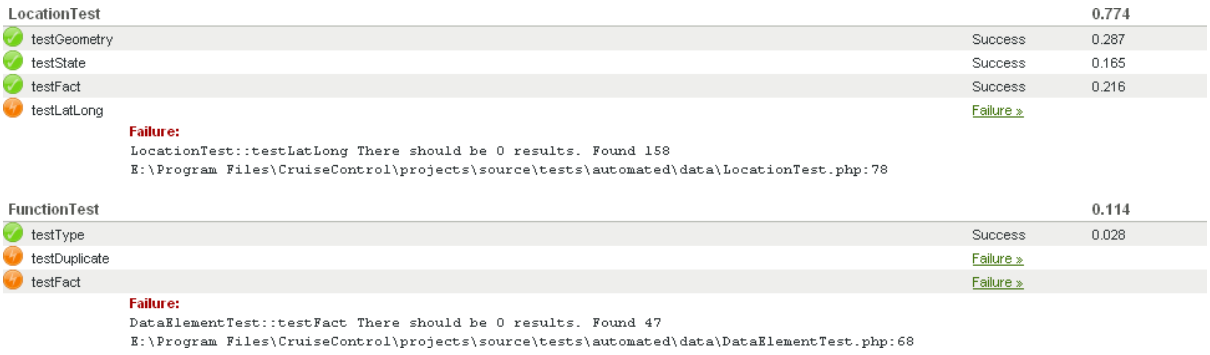


FIGURE 17: UNIT TEST SCREENSHOT

CODE STANDARDS

Inspect the developed code for deviations from the internal code standard

- Check for correct inline documentation (docblock)
- Check for correct variable naming conventions
- Check for correct whitespacing conventions
- Check for complex code that may require refactoring
- Check for incomplete or unused functions

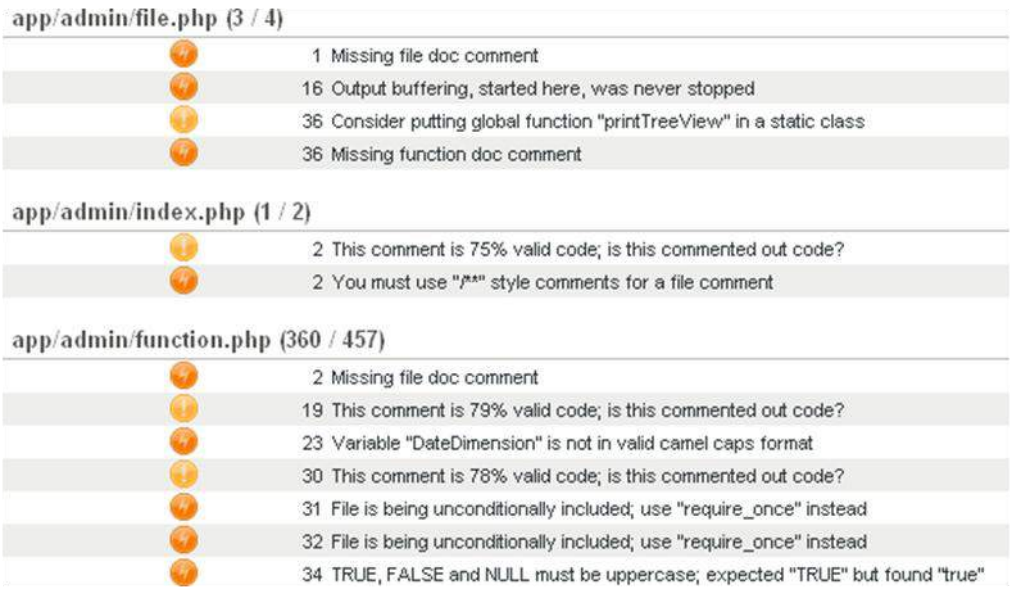


FIGURE 18: CODE STANDARD SCREENSHOT

Notes:

Documentation

The following should be commented;

- Files
- Classes
- Functions
- Class Variables
- Complex Structures

Comments should contain;

- Description
- Author
- Usage
- Parameter description
- Return description
- References to other functions
- Copyright (file comments)

Code Coverage

Calculate and display how much of the software is covered by unit tests. Aim for 80-90% code coverage.

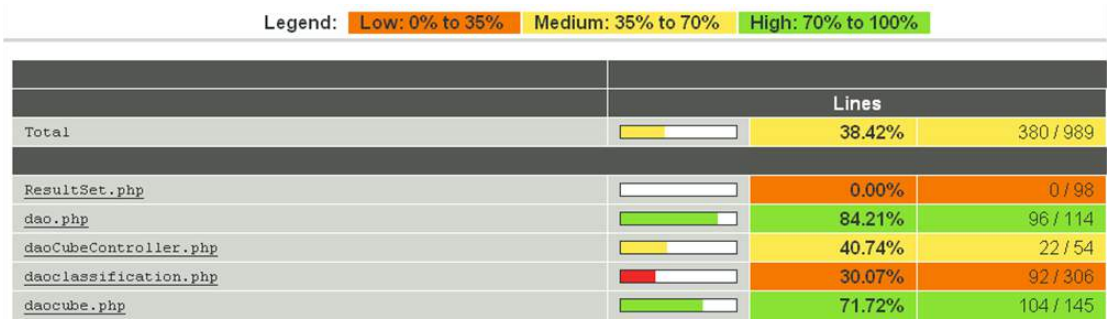


FIGURE 19: CODE COVERAGE SCREENSHOT

Compile

Run any compile or make scripts. All commits should compile.

Notes:

SCRUM MEETING (AKA DAILY STAND-UP)

The purpose of the daily stand-up meeting is to provide a consistent focus on incremental progress and delivery demonstrated to the Product Owner. It is intended to be informative and interactive and align the team's understanding of what is being worked on, by whom and its current status.

This meeting consists of the Product Owner, Scrum Master, and the project team and is time-boxed to 15 minutes.

All participants should answer the following three questions:

1. What did you achieve yesterday?
2. What will you achieve today?
3. What impediment may prevent you from achieving your goal today?

It is the objective of the Scrum Master to remove any impediment identified by the team.

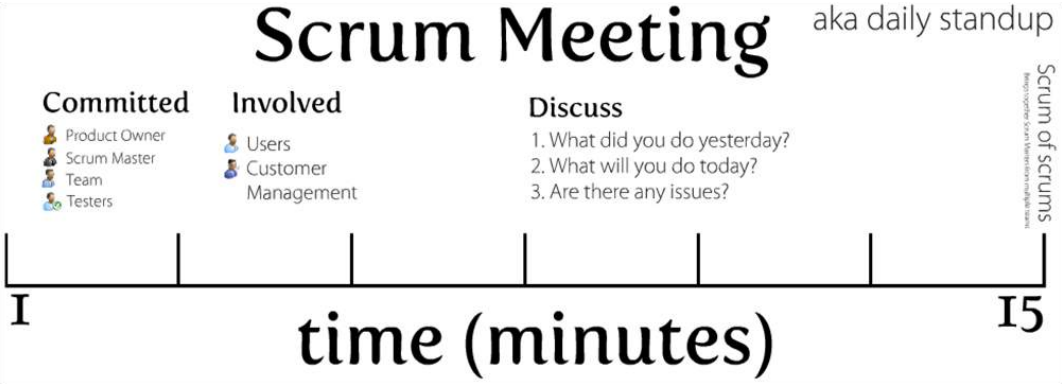


FIGURE 20: DAILY STANDUP WORKFLOW

Projects with multiple teams should hold a scrum of scrums, also timeboxed to 15 minutes, after the initial scrums. This meeting should bring together Scrum Masters from multiple teams to answer the same 3 questions as before, but relating to their teams.

Notes:

INSPECTION

PROJECT VELOCITY

Project Velocity is a measure that determines the capacity and resulting output from the project team over time. In this case it refers to how many User Stories the project team estimate that they can deliver in the project.

The methods available to determine Project Velocity are as follows:

- Use historical values - this assumes similarity between previous User Stories in terms of relative size, data, infrastructure etc
- Run an Sprint – derive the estimate based on observed Velocity over 2-3 Sprints
- Make a forecast – this approach is used where we do not have existing historical values and it is not feasible to run several Sprints to observe the team's velocity. User Stories are disaggregated and estimated at a Task level.

The scope of the project, any changes to the scope, and the planned and actual delivery of User Stories can be represented in a burndown or burnup chart. This chart can also be used to forecast when the project team will have completed all the User Stories in the Project Backlog.

SPRINT VELOCITY

Effort based Velocity determines the capacity and resulting output from the project team for a given Sprint i.e. how many User Stories can be delivered over the next two weeks. This velocity can be calculated as an average at the team level, or if there is a significant variation in the working hours as an aggregate of the individuals.

Sprint velocity is measured in terms of Potential and Forecast capacity:

- Potential Sprint Velocity is the sum of allocated working hours for the project team
- Forecast Sprint Velocity is the estimated productive workable hours that can be attributed to Sprint tasks i.e. takes into account Staff Overhead

Notes:

BURNDOWN & BURNUP CHARTS

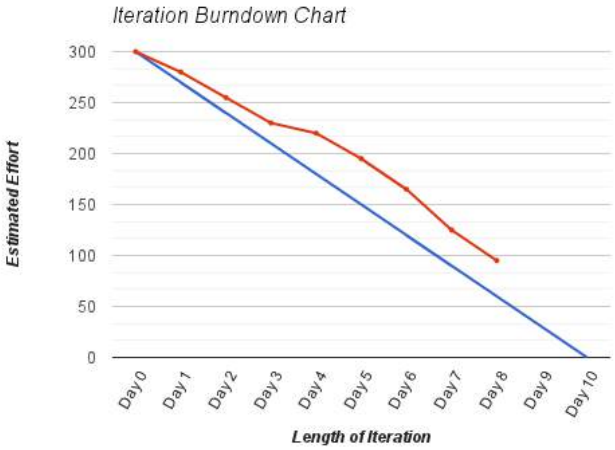


FIGURE 21: EXAMPLE BURNDOWN CHART

Burndown and Burnup charts help managers and customers to view progress against the release, improve future estimates and identify problem trends early. The Burndown (or burnup) chart should be available to everyone involved in the project.

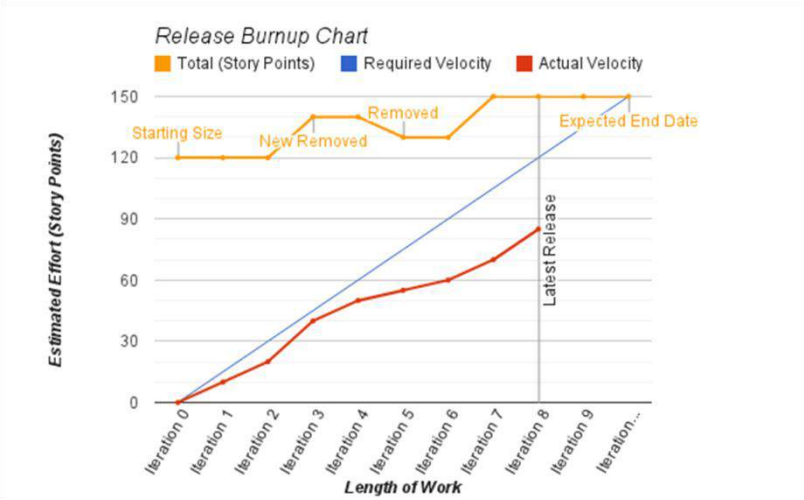


FIGURE 22: EXAMPLE BURNUP CHART

Notes:

PROGRESS PROBLEMS

All about risk mitigation

DISCOVERY

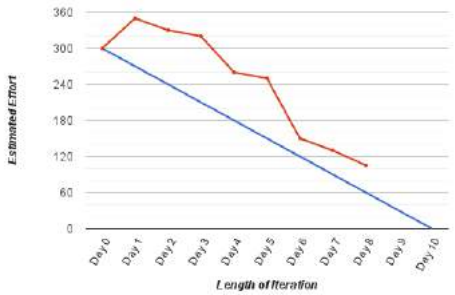


FIGURE 23: PROBLEM BURNDOWN (DISCOVERY)

Issues identified after the sprint begins or refined estimation after the sprint begins. Watch the progress carefully. If necessary review the tasks in the Sprint

SCOPE CREEP

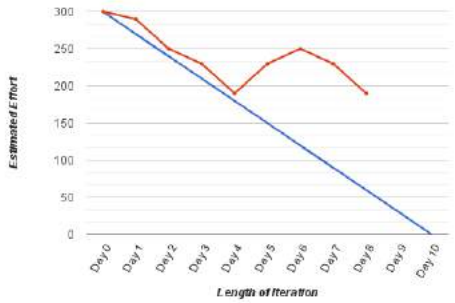


FIGURE 24: PROBLEM BURNDOWN (SCOPE CREEP)

Tasks are being added mid-release. Identify who is adding tasks and stop this behaviour.

Notes:

PLATEAU

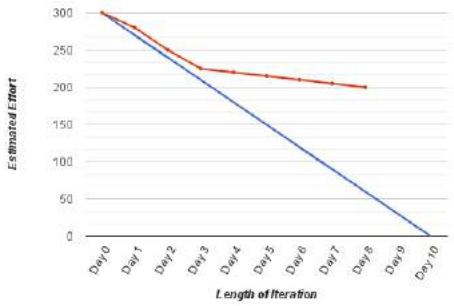


FIGURE 25: PROBLEM BURNDOWN (PLATEAU)

Features are more difficult than estimated or unexpected staffing issues. Review the tasks in the sprint.

TOO MANY FEATURES

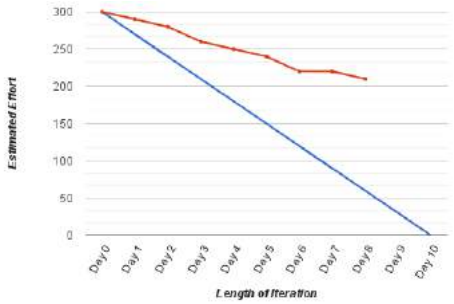


FIGURE 26: PROBLEM BURNDOWN (TOO MANY FEATURES)

Features are more difficult than estimated. Review the estimation process and remove tasks from the sprint.

Notes:

TRACKING EPICS

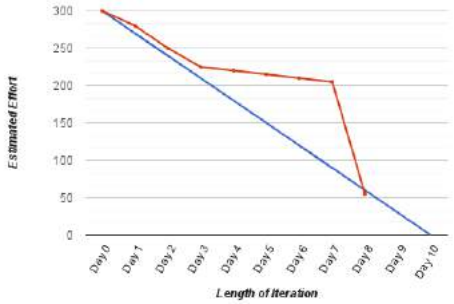


FIGURE 27: PROBLEM BURNDOWN (TRACKING EPICS)

Individual stories are too large and difficult to track. Keep each task under 1 day of work.

Notes:

FINISHING AN SPRINT

'Fall seven times. Stand up eight.'

Old Japanese Proverb

Notes:

SPRINT REVIEW

After the Sprint, the Scrum Master should hold a Sprint Review meeting to demonstrate to the Product Owner and Customer (if different) the completed User Stories for final review for release to production. As the Product Owner should have been involved in the development and verification process on a daily basis this step should be straightforward.

The participants for this session are as follows:

- Product Owner
- Scrum Master
- Project Team

During the meeting the team should:

- Present the completed work to the Product Owner
- Review the work that was completed
- Review the work that was not completed. The User Stories that were not completed may move to the next Sprint.

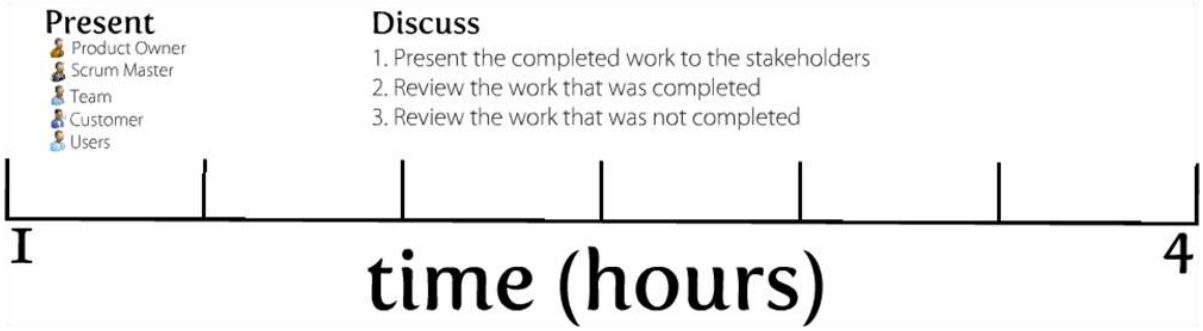


FIGURE 28: SPRINT REVIEW

Notes:

KAIZEN AND THE SPRINT RETROSPECTIVE

After the Sprint Review, the Scrum master should hold a Sprint Retrospective meeting to discuss and improve on the Sprint process itself. During this meeting the team should;

1. Reflect on the sprint
2. Make any process improvements
3. Discuss what went well during the sprint
4. What could be improved during the sprint

The Sprint Retrospective provides the team with the opportunity to reflect on the Sprint just completed and drive continuous process improvement out of the learning's taken from this.

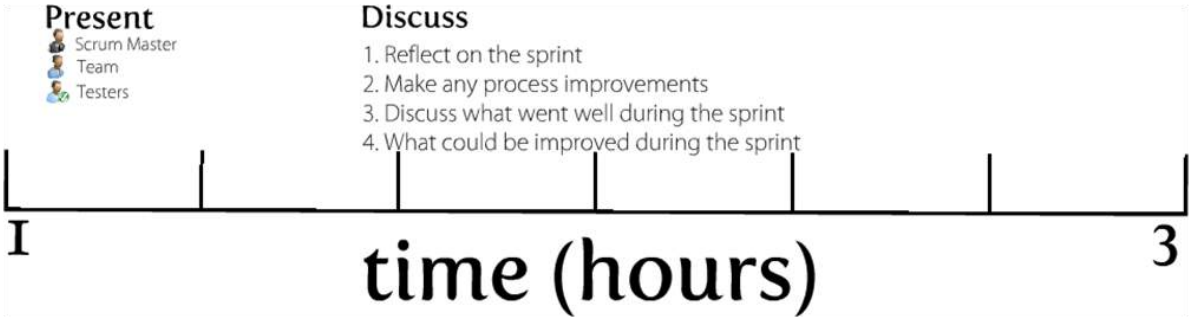


FIGURE 29: SPRINT RETROSPECTIVE

Notes:

REFERENCES

Notes:

Books & Links

Directing the Agile Organisation – Evan Leybourn

Agile Estimating and Planning - Mike Cohn

Managing Agile Projects - Kevin J. Aguanno

http://en.wikipedia.org/wiki/Agile_software_development

http://agilemanifesto.org/

http://en.wikipedia.org/wiki/Agile_testing

http://en.wikipedia.org/wiki/Scrum_(development)

http://www.ambysoft.com

http://en.wikipedia.org/wiki/Iterative_and_incremental_development

http://en.wikipedia.org/wiki/Agile_software_development

http://www.agileadvice.com/

http://en.wikipedia.org/wiki/IEEE_829

http://www.ddj.com/architect/201202925?pgno=4

http://www.scrumalliance.org

Tools

http://trac.edgewall.org/

http://watir.com

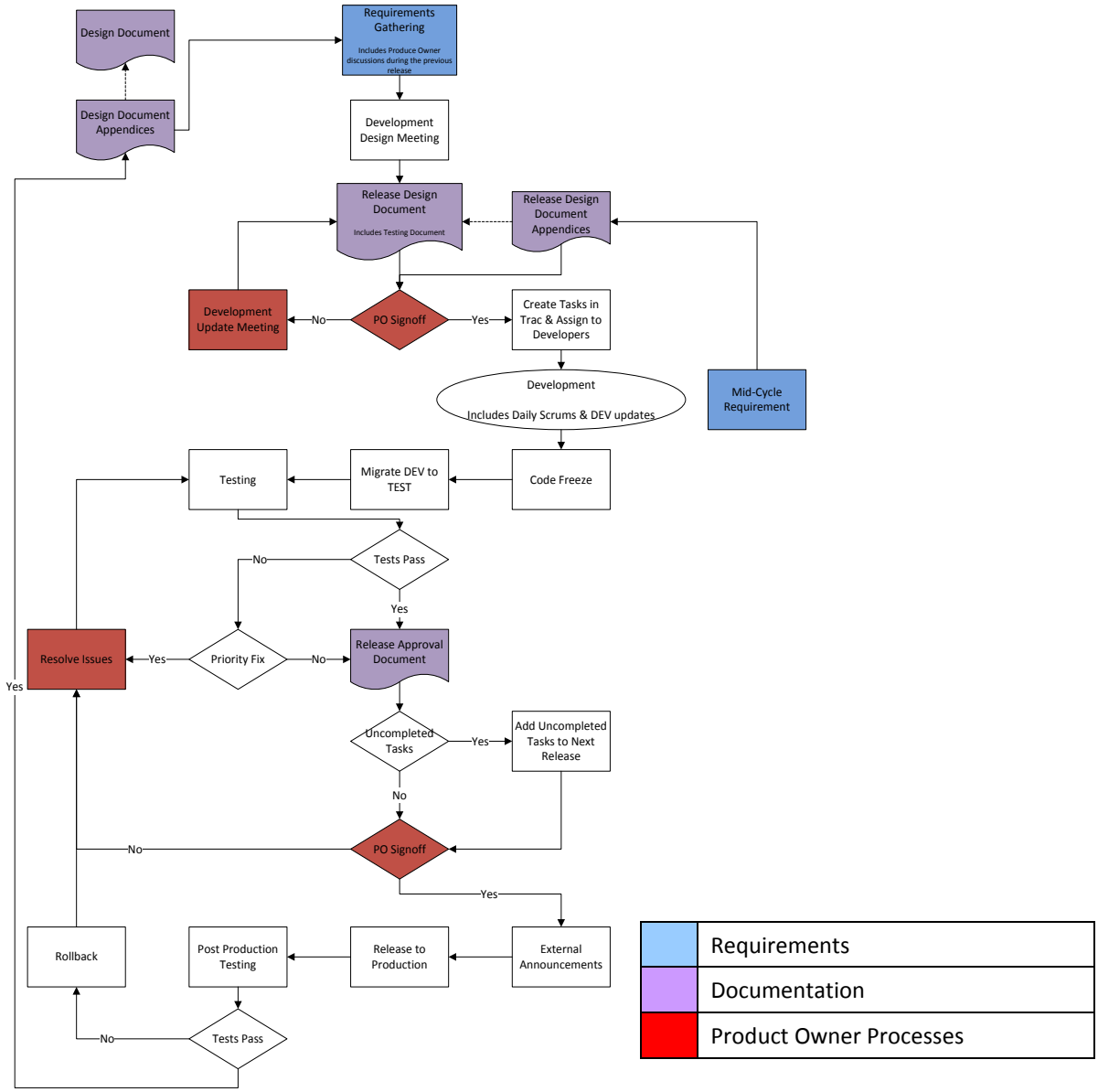
Notes:

SUPPORTING MATERIAL – EXAMPLE RELEASE AND TEST MANAGEMENT PLAN

Notes:

SCHEDULE OF ACTIVITIES & DESCRIPTION OF THE APPROVAL PROCESS

This is an example process for a monthly iteration cycle.



Notes:

Process	Description	Timeframe	Responsible
Design Document	The base design document. This outlines all baseline features in <Product> when initially released.	Complete	Technical Lead
Requirements Gathering	Requirements are drawn from user and product owner feedback during the previous months. The priorities are set by the product owner which leads to the development design meeting.	Ongoing	Technical Lead
Development Design Meeting	A half day meeting of the development team to discuss the requirements and to decide on the resources and timeframes involved.	Day 1	Development Team
Release Design Document	A formal document outlining the features that will be developed during the next sprint. Not all requirements (from step 1) will be part of the release.	Day 1-2	Technical Lead
Product Owner Signoff?	Does the product owner approve the design document?	Day 2	Product Owner
Development Update Meeting	A short meeting of the development team to resolve any design issues raised by the Product Owner and hindering signoff.	Day 2	Development Team
Create Tasks in management system & Assign to Developers	After the design document is complete, the features are broken down into short tasks and assigned to developers to complete.	Day 3	Technical Lead
Development	The development of <product>. Progress is measured through daily scrums and review of the tasks. Updates to <product> are regularly migrated to the DEV server. Issues in newly developed code are added as defects.	Approximately 20 days	Development Team
Mid-Cycle Requirement Gathering	These are high priority requirements, drawn from user and product owner feedback during the development, and form part of the current release cycle.		Product Owner
Release Design Document Appendices	Appendices to the formal design document outlining any mid-cycle requirements.		Technical Lead
Code Freeze	No new features can be developed during this release cycle. The developers will continue to work on defects/bugs in <product>. Any requirements from the design document which are incomplete are moved to the next release.	Day n-5	Technical Lead

Notes:

	The product owner can choose to extend the current release cycle if the requirements warrant it.		
Migrate DEV to TEST	Migrate all changes from the DEV server to the TEST server for formal testing	Day n-5	Development Team
Testing	<product> testers run the base test cases and any new test cases to identify issues and defects in the current release.	Day n-5	Testers
Tests Pass?	Were the tests successful?	Day n-5	Testers
Priority Fix?	Are the issues raised in testing minor, and can they be delayed until the next release?	Day n-5	Technical Lead
Resolve Issues	Any issues or defects for the current release are resolved by the developers, which then get re-tested.	Day n-3-4	Development Team
Release Approval Document	A document based on the initial design document which outlines all completed features in the current release.	Day n-2	Technical Lead
Uncompleted Tasks?	Are there any features from the original design document which could not be completed in the current release?	Day n-1	Technical Lead
Add Uncompleted Tasks to Next Release	To form part of the next design document.	Day n-1	Technical Lead
Product Owner Signoff?	Does the product owner approve the current release	Day n-1	Product Owner
External Announcements	Send an email to all <product> users describing the new changes to <product> with the updated user manual.	Day n-1	Technical Lead
Backup Current Production	Make a manual backup of the current production release. This is used if the system needs to rollback after the release.	Day n-1	CIO Division
Release to Production	Migrate the changes from TEST to PROD	Day n	CIO Division
Post Production Testing	Run through a series of tests to ensure the production release was successful.	Day n	Testers
Test Pass?	Were the post production tests successful?	Day n	Testers
Rollback	Revert PROD to previous release.	Day n	CIO Division
Design Document Appendices	Appendices to the base design document outlining the changes to <product> that were made in the current release.	Day n+1	Technical Lead

Notes:

Roles, Responsibilities and Resource Requirements

Name	Role	Responsibility
<Name>	Product Owner	Final approval for release
<Name>	Technical Manager	Sign off of issues / tasks within the scope of the release
<Name>	Scrum Manager	Manage the daily scrum and monthly design meeting
Testers (Various)	Tester	Follow test cases to identify issues. Can pass/fail cases at their discretion
<Name>	Infrastructure Lead	Maintaining the Development, Test, and Production hardware Install software onto Production hardware Run post-release deployment tests to confirm successful deployment

Formal testers must have had training in using <product>, but should not be “power users”. That is, highly skilled users often miss the usability issues that a “casual user” can identify. For the same reasons, software developers cannot be testers.

Notes:

ROLL BACK STRATEGY

Rollback occurs if the post-production testing identifies any major issues. It is for this reason that post-production testing occurs as soon as possible after the migration from TEST to PROD. The process is as follows;

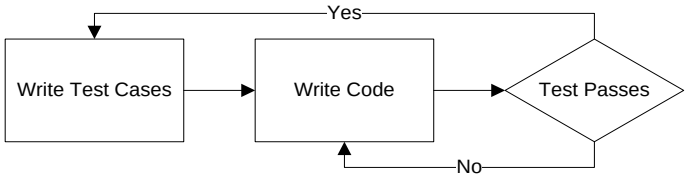
1. Take a complete backup of the current production application. This includes the application data, and the database.
2. Migrate the new version from the Test server to the Production server.
3. Post-Production testing runs through a series of tests to ensure the production release was successful. These tests include;
    a. Comprehensive tool testing to ensure each tool operates as expected.
    b. Speed testing to ensure that each tool (specifically mapping and charting) are responsive within the allowed times.
    c. Authentication testing to ensure that administrators and normal users can access <product> appropriately.
    d. Data quality testing to ensure that the <product> database was updated correctly.
4. If the tests did not pass, and the issue can't be resolved quickly in-situ, begin the rollback process;
    a. Obtain permission from the Product Owner to rollback.
    b. Replace the new version of <product> with the recent backup, including application data, the mapping files and the database.
    c. Email all <product> users informing them of an issue and that the expected upgrade will take place at a later time.
    d. Review the issues with the Product Owner and Development Team to identify what caused the issue and why it was not picked up in initial testing.

Notes:

TESTING METHODOLOGY

TEST DRIVEN DESIGN METHODOLOGY

The <product> testing methodology is based on "Test Driven Design Methodology" (TDD), which is part of the Extreme Programming practices. The core of the methodology is that the basic tests are developed before the software is written.



A major part of TDD is "Unit Testing", a series of automated tests which thoroughly search for logical errors in the software whenever a function is completed. These tests are constantly built on previous tests so if something developed six months ago suddenly breaks under a new update, it is known immediately.

See http://www.developer.com/design/article.php/3622546 for more information

TESTING DETAILS

Testing is performed by <who> staff. These are people who are familiar with the system, but are not involved in the development process.

When a release is scheduled all current test cases are loaded into the issue management system by the Technical Lead. These test cases are assigned to the testers on a random basis and each tester is given a short session discussing what changes have been made and what they should expect.

As each test is completed, the tester updates the test case as either "passed" or "failed". Failed tests are reviewed by the Technical Lead who either accepts the failure or assigns a defect to a developer to resolve the issue. Once resolved the failed test case is reassigned to the original tester to re-test.

Further information can be found under the "Test Report Template" section of this document.

Notes:

Test Types

There are four forms of tests which are run.

1. Defect Testing - Are there any bugs or defects that have been missed?
2. Functionality Testing - Does the release meet the original requirements specification?
3. Usability Testing – Is the release user friendly?
4. Data Testing – Is the data in <product> reasonable and suitable as a policy evidence base?

Manual Testing

All tasks in <product> are managed using a test management tool, including test cases. Each test case is linked to a category and a milestone in which to complete it (e.g. May Release). These test cases are then assigned to a tester to run and go through the following workflow;

- New (Test cases are created)
- Assigned (Test cases are assigned to a tester)
- Tested (Test cases have been run and are awaiting review)
    - Failed test cases go back to assigned once the underlying issue is resolved.
- Closed

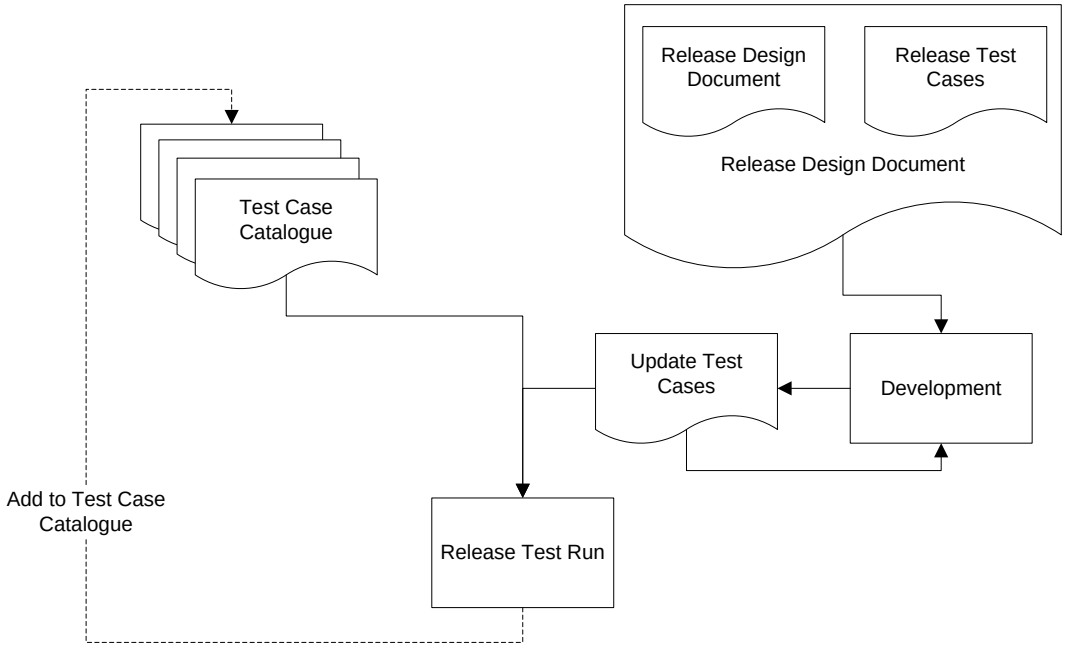
Automated Testing (Unit Tests)

The continuous integration environment runs unit tests for every software update and informs the developers if they have introduced a new defect. If a unit test fails, the responsible developer is notified immediately and the update is removed from the current version until is passes.

Notes:

MAPPING BETWEEN THE TEST SCRIPTS AND THE REQUIREMENTS

As per the development process, basic test cases are developed during the initial design phase. These test cases are further refined during the development cycle as specific functionality is created. This process ensures a close mapping between the test scripts and the requirements.



There are two exceptions to this process.

1. Tests developed prior to this process. These tests form part of the catalogue to ensure good coverage of the test scripts.
2. Core tests that relate to underlying functionality. Underlying functionality is a product of the design process, and does not map to a design requirement.

Notes:

TEST REPORT TEMPLATE

There are 6 details that make up a test case.

1. Reported By: Who is managing the test case? This person is responsible for verifying the result and managing the resolution of any issues arising from failed test cases.
2. Owned By: Who is the test cases assigned to?
3. Priority: How important is this test case?
4. Milestone: Which release is this test case for? Each Milestone in Trac is a monthly release.
5. Component: What part of <product> is the test case for?
6. The description of the test. This is broken into numbered steps. There can be many verifications in a single test, and if any of them fail the entire test fails.

Note that test cases don't use time estimation.

SAMPLE TEST CASES

These testcases are written by the developers and loaded into the tracking system during each testing period.

Summary	Test
<Test Summary>	1. Log in to <product> 2. Select <something> 3. Verify <results>

Notes:

SUPPORTING MATERIAL – EXAMPLE HIGH-LEVEL BUSINESS REQUIREMENTS

Notes:

GLOSSARY OF TERMS

Term	Definition
<Term>	<Definition>

INTRODUCTION

OBJECTIVE
<What Is The Objective Of This Product>

NEED/PURPOSE
<What Need Or Purpose Does It Fulfil>

USAGE
<Who Is The Target Audience / Who Is Allowed To Access The Product>

SCOPE OF REQUIREMENTS SPECIFICATION

INCLUSIONS
<What Is Included In The Specification>

EXCLUSIONS
<What Is Not Included In The Specification>

Notes:

Functional Requirements

General

ID	Requirement description	Essential or Desirable
1		Essential

Report Generation & Printing (including Management Information)

ID	Requirement description	Essential or Desirable
1		Essential

Help Facilities

ID	Requirement description	Essential or Desirable
1	Online help will be available	Essential
2	Context sensitive help for users	Desirable
3	Tooltips will appear on mouse-over of objects	Essential
4		

System Management

ID	Requirement description	Essential or Desirable
1		Desirable

Non-Functional Requirements

General

ID	Requirement description	Essential or Desirable
1		

Notes:

Security

ID	Requirement description	Essential or Desirable
1	The product will adhere to Open Web Application Security Project (OWASP) standards and guidelines	Desirable
2		

Availability

ID	Requirement description	Essential or Desirable
1	Core operating hours 8am to 6pm, Monday to Friday	Essential
2	99% availability during core operating hours	Essential
3	Any planned maintenance is expected to be carried out outside these hours	Essential
4	Monthly reports provided on the amount of downtime/uptime	Essential
5		

User Access

ID	Requirement description	Essential or Desirable
1		

Accessibility

ID	Requirement description	Essential or Desirable
1	The product will adhere to World Wide Web Consortium (W3C) accessibility standards and guidelines	Essential
2		

Usability

ID	Requirement description	Essential or Desirable
1		Desirable

Notes:

IT SERVICE CONTINUITY (DISASTER RECOVERY)

ID	Requirement description	Essential or Desirable
1	A disaster recovery process will be in place. The system will be able to be recovered and available within 48 hours of a disaster event.	Desirable
2		

PERFORMANCE

ID	Requirement description	Essential or Desirable
1	Response times for users will be 4 seconds maximum response to all web pages.	Desirable
2		

CAPACITY/VOLUME

ID	Requirement description	Essential or Desirable
1		

INTEROPERABILITY

ID	Requirement description	Essential or Desirable
1		

DATA

ID	Requirement description	Essential or Desirable
1		

Notes:

AUDIT

ID	Requirement description	Essential or Desirable
1		

REQUIREMENTS IMPOSED BY SPECIFIC LEGISLATION STRATEGIES

GENERAL

ID	Requirement description	Essential or Desirable
1		

PRIVACY

ID	Requirement description	Essential or Desirable
1		

CONSTRAINTS

Id	Description
1	Developer resource limits.
2	The product will be a web application and as such the interface needs to be compliant with standard web browsers including but not limited to Internet Explorer 6 & 7 and Mozilla Firefox
3	Compliance with OWASP and W3C standards. This constraint is self-imposed. Refer to sections 3.2.2 and 3.2.4.
4	Due to web standards and accessibility the interface is constrained to HTML, CSS and JavaScript
5	

Notes:

ASSUMPTIONS

Id	Description
1	

RISKS

Id	Description
1	

Notes:

SUPPORTING MATERIAL – EXAMPLE TECHNICAL ARCHITECTURE

Notes:

CONCEPTUAL SYSTEM DESIGN SECTION

Conceptual System Checklist	Responses - Select all that apply
Project Type	☐ New System ☐ Upgrade System ☐ Other (specify):
Development Approach	☐ Commercial Off The Shelf (COTS) ☐ Government Off The Shelf (GOTS) ☐ Custom
Delivery of Functionality	☐ Modular (functionality delivered over time) ☐ Monolithic (functionality delivered all at once)
System Interactions	☐ Business to Business (B2B) ☐ Business to Customer (B2C)
Electronic Commerce	☐ Yes ☐ No
Pilot Prior to Implementation	☐ Yes ☐ No
Security - Regulatory or Privacy Requirements	
W3C Accessibility Compliance	☐ Yes ☐ No
Estimated Total Number of Customers	Total: By Audience: Customer: _____ Employee: ____ Business:____ Other:_____
Estimated Total Number of Concurrent Customers	Total: By Audience: Customer: _____ Employee: ____ Business:____ Other:_____
Average Transaction Response Time Requirements	Platform: Windows XP Pro, PIV 1.2Ghz, 512MB RAM Bandwidth: DSL/Cable (1Mb, 50% utilized) Average File Size: ___ Platform: Windows XP Pro, PIV 1.2Ghz, 512MB RAM Bandwidth: LAN (10Mb, 50% utilized) Average File Size: 30Kb
Production Hours of Operation	
Production Availability	Uptime => Downtime/year (i.e. unplanned)

Notes:

Expectations	99% (2 Nines) => 003d 15h 36m 00s
Application Backup Requirements	Full Backup: __ Real Time __ Daily __ Weekly Incremental Backup: __ Hourly __ Daily __ Weekly
Application Recovery Requirements	Recovery Time Objective:
Disaster Recovery Requirements	Hot Site: __ hour(s) Warm Site: __ days(s) Cold Site: __ days(s)
Hosting Location	
Architectural Approach	☐ SOA ☐ 3/N Tier ☐ Other (specify):
Processing Type	☐ OLTP ☐ OLAP ☐ Other (specify):
Development Platform	☐ J2EE ☐ .NET ☐ PHP ☐ Other (specify):
Architectural Framework(s)	
Architectural Pattern(s)	☐ MVC ☐ Factory ☐ Controller ☐ Data Access Object ☐ Other (specify):
Software Testing	Test Driven Design Methodology (TDD) - basic tests are developed before the software is written. Tests include; • Functional • Performance • Accessibility • User Acceptance
Security Technologies	☐ Identity and Access Management ☐ SSL/TLS ☐ Data Encryption ☐ Cookie Encryption ☐ __ DES __ 3DES __ AES __ Other (specify): ☐ Other (specify):

Notes:

Conceptual System Design Diagram

<Insert Physical Architecture Diagram>

<Descript Physical Architecture Diagram>

Detail System Design Checklist

Detail System Checklist	Responses – Select all that apply
Client Operating Systems	☐ Apple ☐ Microsoft ☐ Linux ☐ Unix ☐ Palm ☐ Microsoft PocketPC ☐ Other (specify):
Client Platforms	☐ Desktop/Laptop ☐ Tablet ☐ PDA ☐ Smart Phone ☐ Other (specify):
Client Footprint by Platform	Specify size of footprint in KB or MB: Desktop/Laptop: Tablet: PDA: Smart Phone: Other (specify):
Client Connection Speed	Specify speed in kbps or mbps: Minimum: Recommended:
Client Richness	☐ Browser Based ☐ Rich Client ☐ Rich Internet (AJAX)
Browsers and Versions Supported	☐ Internet Explorer: ☐ Mozilla Firefox: ☐ Safari: ☐ Opera:
Presentation - Client Side Languages	☐ HTML ☐ DHTML

Notes:

	☐ XML ☐ XHTML ☐ VB.NET ☐ C# ☐ ActiveX Controls ☐ Java Applets ☐ Java ☐ JavaScript ☐ VBScript ☐ C++ ☐ Other (specify):
Application State	☐ Cookies (Non-Persistent Cookies, Persistent Cookies) ☐ Session Ids ☐ State Stored in Hidden Fields ☐ Other (specify):
Virtualization	☐ Server ☐ Storage
Web Server Location	☐ Public Facing ☐ Internal Facing
Web Server Operating System	☐ Windows ☐ Linux ☐ Unix ☐ Other (specify): ☐ Specify Version:
Web Server Software	☐ Apache ☐ Microsoft ☐ Sun ☐ Oracle ☐ Other (specify): ☐ Specify Edition and Version:
Web Server - High Availability	Load Balanced: __ Yes __ No Processor Architecture: __ 64 Bit __ 32 Bit Processor Cores: __ Double __ Single Other (specify):
Web Server - Specifications	Rollout Configuration: Number of Servers: __ CPUs/Server: __ CPU Type: _____ CPU Speed: ____ Amount of RAM: ___ Maximum Configuration:

Notes:

	Number of Servers: __ CPUs/Server: __ CPU Type: _____ CPU Speed: ____ Amount of RAM: ___
Presentation – Server Side Languages	☐ ASP.NET ☐ VB.NET ☐ C# ☐ PHP ☐ JSP ☐ Servlets ☐ Java ☐ Server Side Includes (SSI) ☐ C++ ☐ Other (specify):
Application Server Operating System	☐ Windows ☐ Linux ☐ Unix ☐ Other (specify): ☐ Specify Version:
Application Server Software	☐ Microsoft ☐ IBM ☐ Sun ☐ Oracle ☐ BEA ☐ Other (specify): ☐ Specify Edition and Version:
Application Server – High Availability	Processor Architecture: __ 64 Bit __ 32 Bit Processor Cores: __ Double __ Single RAID Supported: __ Yes __ No SAN Supported: __ Yes __ No Mirroring Supported: __ Yes __ No Clustering Supported: __ Yes __ No Grid/On Demand Supported: __ Yes __ No Other (specify):
Application Server - Specifications	Rollout Configuration: Number of Servers: __ CPUs/Server: __ CPU Type: _____ CPU Speed: ____ Amount of RAM: ___ Maximum Configuration: Number of Servers: __ CPUs/Server: __ CPU Type: _____

Notes:

	CPU Speed: ____ Amount of RAM: ___
Business Rule – Application Languages	☐ VB.NET ☐ C# ☐ PHP ☐ Java (J2SE) ☐ Java/EJB (J2EE) ☐ C++ ☐ Other (specify):
Database Server Operating System	☐ Windows ☐ Linux ☐ Unix ☐ Other (specify): ☐ Specify Version:
Database Server Software	☐ Microsoft ☐ IBM ☐ Oracle ☐ PostgreSQL ☐ MySQL ☐ Other (specify): ☐ Specify Version:
Database Server – High Availability	Processor Architecture: __ 64 Bit __ 32 Bit Processor Cores: __ Double __ Single RAID Supported: __ Yes __ No SAN Supported: __ Yes __ No Mirroring Supported: __ Yes __ No Clustering Supported: __ Yes __ No Grid/On Demand Supported: __ Yes __ No Other (specify):
Database Server - Specifications	Rollout Configuration: Number of Servers: __ CPUs/Server: __ CPU Type: _____ CPU Speed: ____ Amount of RAM: ___ Maximum Configuration: Number of Servers: __ CPUs/Server: __ CPU Type: _____ CPU Speed: ____ Amount of RAM: ___
Data Access – Connectivity Methods	☐ ADO.NET ☐ ODBC ☐ OLE/DB ☐ JDBC

Notes:

	☐ JDO ☐ DB2 Connect ☐ Other (specify):
SQL Languages	☐ T/SQL ☐ PL/SQL ☐ Other (specify):
Stored Procedures Utilization	No Yes __ Data Access only __ Business Rules and Data Access

SYSTEM DESIGN DESCRIPTION

<Insert High Detail Physical Architecture Diagram>

<Describe High Detail Physical Architecture Diagram>

BUSINESS CONTINUITY

<Describe Business Continuity / Disaster Recovery Procedures In Place>

EXPECTED ISSUES

<Describe Issues Or Link To Issues Register>

Notes:

SUPPORTING MATERIAL – EXAMPLE DESIGN PAGE

Notes:

Story: _____

Estimate		**Priority**	
Value		**Risk**	
Purpose		**Test Cases**	
Use Cases		**Notes**	

Notes:

SUPPORTING MATERIAL – EXAMPLE RELEASE APPROVAL

Notes:

Release Approval

Release Name: _____

Version: _____

Date: _____

Compliance Outcome

Requirement	Pass/Fail	Notes

Defect Testing Outcome

Name	Pass/Fail	Notes

Notes:

PERFORMANCE TESTING OUTCOME

Name	Pass/Fail	Tests (All results are the average of 10 loads at SSC level)
		Single User 1000 Users
		Single User 1000 Users
		Single User 1000 Users
		Single User 1000 Users
		Single User 1000 Users
		Single User 1000 Users
		Single User 1000 Users
		Single User 1000 Users

ISSUES TO CARRY OVER TO THE NEXT RELEASE

Notes:

SIGN OFF

Technical	Program	Final
<Name>	<Name>	<Name>

Notes: