# Agile Team Practises with Scrum

PluralSight - <https://app.pluralsight.com/library/courses/agile-team-practice-fundamentals>

# Agile and Why It Works

## Comparing Development Processes

**Plan Driven** – Waterfall – Originated in 1970 – Gated, staged phases a project would go through. One phase wouldn’t begin until the previous was complete. The original idea was for iterative phasing. Lessons should be learnt in each phase and inform the phase before it. Entire process should be run through twice before being released to client.  
  
Royce - “Unfortunately, for the process illustrated, the design iterations are never confined to the successive steps. ”

We can never design a system in full at the beginning. The design will evolve the more we understand the requirements.

This seems to have been *lost* from Waterfall development process!

**Agile** – 2001 – Agile Alliance – Manifesto for Agile Software Development.

“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

* Individuals and Interactions over Processes and Tools
* Working Software over Comprehensive Documentation
* Customer Collaboration over Contract Negotiation
* Responding to Change over Following a Plan

That is while there is value in the items on the right, we value the items on the left more.”

## The Big Difference

2 significant differences.

**Planned Driven Model** – Delivering one increment of software at the end of the project.

**Agile** – Small changes of software, more frequently throughout the lifecycle of the project.

**Planned** – Activities are sequential. One phase completed and delivered before the next begins.

**Agile** – Embracing a little of each activity all the time. Planning something that continues ALL the time. Team is multi-disciplinary containing people analysts, developers, testers, integration, etc. Allow decision makers to learn from small releases and make decisions that can adjust to market feedback.

Comparing Methodologies

|  |  |
| --- | --- |
| **Plan Driven Methodologies** | **Agile Methodologies** |
| Change is bad, therefore discouraged and controlled. | Change is inevitable and valuable, therefore encouraged and embraced. |
| Adherence to the plan determines success or failure. | Incentives are often based on customer satisfaction and ROI. |
| I am done when my part of the plan is signed off. | I am done when the customer is happy. |
| Lots of gate to control quality. | Highly iterative to achieve quality. |
| Inspect product when it is complete. | Inspect work as it is being done. |
| Start by predicting what will be delivered. | Start with a goal of filling a need. |

## Graphing Agile v Plan Driven

A picture containing chart

Description automatically generated

Plan Driven

* Beginning – Entire organisation involved in building a plan.
* Mid – Work is progressing, but visibility of results and current situation can be hidden away.
* End – Business stakeholders become concerned, alert and pushing for results no matter what state the software is in. The plan said it would be done…where is it?

A picture containing graphical user interface

Description automatically generated

Agile tries to enable higher adaptability over a longer period of time encouraged by frequent delivery. Business decision makers urged to change mind based on what they learn about the market into which the product is being sold. Priorities may be changed, architectures adapted results in systems that are more adaptable over long periods of time.

A picture containing diagram

Description automatically generated

Agile tends to deliver business value sooner.

Phase One: Agile team tackling what is perceived as highest value first and delivering fully functional versions of features the customer requested. Plan Driven team working on high level design or architecture.

Phase Two: Not uncommon for an Agile project to stop in the middle of planned project lifespan, not because of failure but because enough value has been delivered that an organisation wants to invest elsewhere. Plan Driven is finally beginning to produce some software. Towards the end of this phase Plan Driven will have lots of features with not much functionality and still no saleable or deliverable product.

Phase Three: Agile has been developing revenue for a lot longer!

A picture containing diagram

Description automatically generated

Agile allows us to choose riskiest items first. Even use it to determine if the project resonates with customers at all.

## What Agile Is and Is Not - Dispelling Common Myths

**Principles**

* Iterative Delivery
* Transparency and Openness
* Frequent Feedback
* Lean Principles

|  |  |
| --- | --- |
| **Processes and Methodologies** | **Techniques and Practises** |
| **Working with People** | **Working with Software** |
| Planning | Design |
| Teamwork | Coding |
| Engaging customers | Testing |
| Providing leadership | Deploying |
| Collaboration | User Experience |
| Learning |  |

**Agile Is**

|  |  |
| --- | --- |
| Iterative | Easy to Understand |
| Adaptive | Hard to Implement |
| Value Based |  |

**Agile Is Not**

|  |  |
| --- | --- |
| Just about writing code | Whatever you want it to be |
| Undisciplined | A placebo for pesky developers |
| Unstructured |  |

## Contemporary Agile Methodologies

**Extreme Programming (XP)**

Ancestor of most Agile methodologies – Kent Beck 1999

Blends processes and practises. Found success in smaller teams. Controversial due to advocates teaching it as *dogma*.

Basic Idea - Take observed effective team practises and push them to extreme levels

|  |  |
| --- | --- |
| **Good Practise** | **Pushed to Extreme** |
| Code Reviews | Pair Programming |
| Testing | TDD and constant regression |
| Software Design | Relentless Refactoring |
| Simplicity | The simplest thing that could work |
| Integration Testing | Continuous Integration |
| Short Iterations | The Planning Game |

12 Practises

|  |  |  |
| --- | --- | --- |
| 1. The Planning Game | 5. Testing | 9. Collective Ownership |
| 1. Small Releases | 6. Refactoring | 10. On-site Customer |
| 1. Metaphor | 7. Pair Programming | 11. The 40-hour Week |
| 1. Simple Design | 8. Continuous Integration | 12. Coding Standards |

**Scrum**

An iterative project management process – Originated with Schwaber and Sutherland in 1990s. Does not advocate any specific engineering techniques and as such is used beyond software development. Blends well with XP development. Above all is a Simple idea.

Diagram

Description automatically generated

Project Backlog is a list of prioritised and always estimated requirements. Highest priority item is taken from top of list and broken into smaller tasks by the team who is going to be doing the work. This list becomes the ‘Sprint Backlog’. It is this set of tasks that will be developed in the next 2 – 4 week Sprint. Sprints contain a series of 24-hour iteration blocks in which the team makes progress every day and communicates within the team how much progress has been made daily. Goal is a fully functional product – working software. This may not be 100% shippable but features that are present are complete and ready.

**Lean**

More a set of guidelines than a formal methodology. Originally comes from a manufacturing space (Toyota), now applied in software development although can be applied to improve any process. Focused mainly on Continuous Improvement (Kaizen) and value flow from inception to customer.

Principles

Eliminate Waste – Remove anything that is not of value to the customer.

Amplify Learning – Culture of learning. Encourage individuals to go beyond their area of domain expertise.

Respect People – People most knowledgeable about the work being done are those performing it.

Build Quality – Quality is a first-class concern and not something to be injected after completion.

Defer Commitment – Waiting to make a decision at last possible moment – Longer we wait, more information available to make a successful decision.

Deliver Fast – Fast and frequent release to aid feedback gathering and amplify learning.

Recognise and Optimise the Whole – Focus improvement on the whole rather than individual processes.

**Feature Driven Development**

Based on time-honoured engineering practises. Originated in software development with Jeff De Luca in 1997. Advocates modelling as the base currency of process and adheres to strict processes (UML, formal class design, etc.). Some reports claim FDD scales more effectively than Scrum.

|  |  |
| --- | --- |
| **Activities** | **Milestones** |
| 1. Develop Overall Model | 1. Domain Walkthrough |
| 1. Build Feature List | 2. Design |
| 1. Plan by Feature | 3. Design Inspection |
| 1. Design by Feature | 4. Code |
| 1. Design by Feature | 5. Code Inspection |
|  | 6. Promote to Build |

# Agile Requirements and Estimation

## Creating Effective Requirements

What is a Requirement?

* A feature, behaviour, or constraint to be added to a system.
* A prelude to a conversation between stakeholders.
* A request for someone to do work.
* A request for software to change.

Requirements = Desirements until fully implemented!

A Requirement is NOT

* A solution design.
* A decision about implementation.
* Typically illustrative of the final deliverable.
* The source of truth.

Agile Estimating and Planning – Mike Cohn – Proposes acronym INVEST for qualities of an effective requirement.

Graphical user interface, application

Description automatically generated with medium confidence

**Independent** – Idea that each and every requirement into a system needs to be executable in and of itself. This avoids dependencies across features and avoids confusion in planning. If in estimation, it is hard to determine if a feature is independent, break it down further.

**Negotiable** – Concept that requirements are not contracts or promises. Too many details yet to be worked out to understand the end solution. If a requirement is treated a contract, then there is a perception that there is no need to talk about it further, which can lead to defects.

**Valuable** – Requirements need to be valuable to the users or customers, not necessarily the implementors.

**Estimable** – Requirements are used in planning. If knowledge is lacking domain or technical knowledge or the requirement is too large, it may not be estimable.

**Sized Appropriately** – Work should be able to be accomplished within half the time of an iteration period. Any that don’t need to be broken down into smaller requirements.

**Testable** – Needs to be demonstrable as having met customer requirements. Confirmation should be automatable if possible, to a greater degree.

## User Stories

A User Story is a way to express a requirement in a usable, human-readable form. Very effective means of communication between the people who want and the people who do.

**User Story Recipe**: As a <role> I want <feature> so that <benefit>.

Roles – Important to use appropriate contextual role, not a generic ‘user’.

Why do User Stories Work Well?

* They are simple to write and understand.
* Software requirements is a communication problem.
* They elicit detail in further conversation.
* Requirements analysis is effective when performed collaboratively.
* Full intent can rarely be modelled or represented 100%.

User Stories can generate smaller sub-stories based on conversations.

Signs Stories are Working

* Focus shifts from writing to talking.
* Stories are understood by customer and developer.
* At estimation time, they are the right size.
* Participative design is occurring.
* Emphasis is on the user goals, not the systems.

## Scenarios

Recipe for communicating when a requirement has been met: Given <context> [and <more context>], when <something happens> then <outcome> [and <another outcome].

These recipes can be seen as scenarios to communicate clearly and quickly (e.g. Scenario 1 – User Cancels Booking, etc.).

The Whole Story on a Card – Put whole user story on a single ‘card’ – Title, Description and Success Criteria (plus fields for cost estimates – Business Value, Development Effort, ROI).

## Estimating Work

Estimates Are Necessary

* To plan and proceed deliberately
* To get a feel for costs
* To calculate potential ROI
* To understand the size of something
* To know if work can even be done
* To weigh options

Ways to Estimate Software (not necessarily the best ways!)

* Darts! 😉
* Give it to a Manager
* Ask the Expert
* Without ‘bothering’ the Developers

Deadly Estimation Warning Signs

* Estimates are given without looking at historical performance.
* Someone other than the team is doing the estimation (e.g. ‘Ask the Manager’).
* Estimates are treated as promises.
* Estimates are rejected because they don’t fit an already existing plan.

How do we measure software work? Not able to measure in ‘usual’ ways (e.g. Lines of Code, Coffees Drunk, etc.).

## Story Points

What is a Story Point?

* Very common way to estimate work
* Based on size and complexity, not duration
* Unitless and numerically relative
* Different for each team of estimators
* Points are additive, unlike time
* Based on historical reality
* Easy to use and understand

**Using Story Points**

Each User Story is given a value (e.g. Bug 1 – Cost 20). Can see immediately which stories ‘cost’ the most.

Story Point Values – Use values that ‘make sense’ – cumulatively bigger as scale goes up. Many teams use based on Fibonacci Sequence-ish (e.g. 1, 2, 3, 5, 8, 13, 20, 40) or power of 2 (e.g. 1, 2, 4, 8, 16, 32). Do not use a scale where numbers sit next to each other (e.g. Scale of 1 – 10). Instead use historical estimates to help: “This piece of work last week was a 5. This piece of work is bigger, but how much bigger 2x, 3x...”  
  
Estimates are NOT Promises – If estimates are used against you this is a people problem, not a problem with the estimates. Address it.

## Planning Poker

Estimating with Groups – Group derived estimates are demonstrably more accurate than by individuals – “Together, we are smarter than any one of us”

Relative Estimation – If Task A is 8 - 10 times larger than Task B, it is likely that it will take 8 – 10 times longer to complete (unless we introduce some aspect of tooling). Rather than asking “How long will this take”, a better question is “How big is something in relation to something else”.

Myth – With more time, estimates get significantly more accurate

Estimation is Expensive – We can get fairly accurate quickly with a good historical record and ability to estimate work relative to work already been performed.

Planning Poker Cards – Deck of Cards with sequence on (e.g. 1, 2, 3, 5, 8, 10, 20, 40…and a ‘?’).

Why Planning Poker Works

* Emphasises relative sizing
* Focuses most estimates within an order of magnitude
* Everyone is heard
* Finds hidden requirements and details
* Estimators must justify estimates
* It is iterative

Planning Poker Rules

1. Each estimator has a deck of estimation cards
2. Customer/Product Owner reads a story and it’s briefly discussed
3. Each estimator selects a card that’s his or her estimate from their deck and places face down
4. Cards are all turned over at same time so all can see them
5. Discuss differences (especially outliers)
6. Re-estimate until estimates converge.

Options for Handling Conflict – Aim for consensus, not unanimous agreement

* Wait for convergence
* Average the estimates
* Toss out high and low
* Send the item back for re-definition

# Scrum Fundamentals – Essential Vocabulary and Behaviours

## Origins of Scrum

1986 – Harvard Business Review – ‘The New New Product Game’ – Discussed phases of product development cycles.

Type A – Sequential | Type B – Overlap of Cycles | Type C – Concurrent Cycles

Lean Manufacturing – Toyota Production System – Taiichi Ohno

Goals: Design out overburden (Muri) | Design out inconsistency (Mura) | Eliminate Waste (Muda)

Techniques: Continuous Improvement (Kaizen) | Relentless reflection (Hansei) | Production smoothing (Heijunka) | Go see for yourself (Genchi Genbutsu)

Learning Organisations – Peter Senge

* Are adaptive to their external environment
* Continually enhance their capability to change/adapt
* Develop collectively as well as individual learning
* Use the results of learning to achieve better results

Scrum Masters – Jeff Sutherland & Ken Schwaber – 1995 – Co-presented Scrum as a formal practise.

* The Enterprise and Scrum / Agile Project Management with Scrum - Schwaber

## Principles Behind the Methodology

Scrum Promotes These Principles

* Small working teams
* Embracing changing requirements
* Deliver finished work frequently
* Small batches
* Enable releasing product whenever required

Characteristics of Scrum

* Activities are time boxed
* All project metrics, reports and deadlines are prominently displayed on Scrum Boards
* Multi-disciplinary and self-organising teams
* No specific engineering practises prescribed

More Characteristics!

* Product progresses in a series of Sprints
* Requirements are captured as items in a Product Backlog
* Continuously test product as it is created

## The Basic Scrum Framework

What Is It?

* A project management technique
* One of many Agile practises
* An effective way to deliver products

In a Nutshell

* Schedule a demo with the customer
* Make product to show at demo – functioning software
* Get feedback from the demo and use it to guide next development work

## Who Uses Scrum and Why

Microsoft, Yahoo, Google, Electronic Arts, Lockheed Martin, Phillips, Capital One, BBC, etc.

Certainly not a fad! Different team sizes proving Scrum is scalable.

## Roles in Scrum

A team succeeds of fails as one

Three Roles in the Model

* Scrum Master
* Team Members
* Product Owner

**Scrum Master** - - *serves* - - > **Team Members** - - *serves* - - > **Product Owner** - - *serves* - - > **CLIENT**

**Product Owner**

* Is responsible for the profitability (ROI)
* Defines the features of the product
* Prioritises features
* Decides on a release date and content
* Can change features and priority of each iteration
* Accepts or rejects iteration results

**Scum Team**

* Typically 5 -9 people
* Cross functional
* Members should be full-time
* Teams are self-organised
* Membership changes only between sprints

**Scrum Master**

* Represents management to the product
* Responsible for enacting Scrum values and practises
* Ensure that the team is fully functional and productive
* Removes impediments
* Facilitates team meetings
* Enable close cooperation across all roles and function
* Shield the team from external interferences

Scrum Masters Contend With:

* Waterfall expectations of inattention
* The illusion of command and control
* The pain of transparency
* Belief in magic!

## Artifacts of the Process

Scrum Artifacts

**Product Backlog**

* A prioritised complete list of Desirements
* All potential features of the product
* Single source of truth for requirements
* Managed by the product owner (Created by and Prioritised By)
* Reprioritised for each sprint

**Product Backlog Item**

* Unit of deliverable work
* Has measurable business value
* Estimated by the team
* May reference other artefacts (e.g. Large spec docs, mock-ups, etc.)
* Contains criteria for successful completion

Defining “Done”

* What does ‘done’ mean in your current project?
* What issues do you see with this definition of ‘done’?
* What technical problems do you see with this approach?

**Sprint Backlog**

* Created by the Scrum Team
* List of activities the team commits to for a single Sprint
* Drawn from the Product Backlog Items
* Often thought of as a to-do list for the team

Sprint Backlog Item

* + Represents a single deliverable or activity
  + Estimated in ideal hours
  + Managed by a single Scrum Team Member
    - May be done by others on the Scrum Team
    - Reportable each day of the Sprint
  + Ideally no more than 2 days of work
  + No minimum size

Sprint Backlog Characteristics

* + Individuals sign up for work of their own choosing
    - Work is never assigned
  + Estimated work remaining is updated daily
  + Any team member can add, delete, or change the sprint backlog
  + Work for the sprint emerges
  + If work is unclear, define a sprint backlog item with more time and break it down later
  + Update work remaining daily as more becomes known

**Sprint Burndown Chart**

* Shows work remaining for the Sprint
* Updated daily
* Used by Team Members to adjust activities
* Used by Scrum Masters as the tactical project management tool

**Scrum Team Velocity**

* A measure of how much value a team delivers
  + Per Sprint
  + Over time
* Used to inform the Team when pulling work
* Used by Product Owner to plan released and future work
* Will vary significantly from one Sprint to another

## Ceremonies

**Sprint Planning Meeting**

* Sprint goals are created
* Sprint Backlog is created
* Burndown chart is initialised

Sprint Goals

1. High level descriptions of the work to be accomplished in the Sprint
2. Agreed to by the Team and Product Owner

Who is there?

* Product Owner
  + Brings estimated, prioritised Product Backlog
  + Defines ‘Done’ for any ambiguity
  + Answers any other questions about the work
* Scrum Master
  + Facilitates the whole discussion
* Team
  + Create their Sprint Backlog for the next Sprint

Additional Info

* Often first thing on Monday after the last Sprint
* Time boxed to less than 2 hours
* Team selects items from the product backlog they can commit to completing
* Team discusses details of implementation
* Scrum Master brings food!

Workflow for creating the Sprint Backlog

* Team selects the first Product Backlog Item (PBI) on top of the list they can commit to completing
* Team identifies all tasks to be completed to deliver the PBI
* Team estimates all tasks identified in ‘ideal hours’
* Does the set of Sprint Backlog Items (SBI) fit into the Sprint?
  + Yes – add the SBIs to the Sprint Backlog
  + No – discard the PBI or negotiate what can be delivered
* Repeat from start until the Sprint Backlog is full

**The Daily Scrum**

* Team focuses together once per day
* Ideally in morning, to set direction for the day. Same place, same time each day
* Time boxed, typically 15 minutes
* Standing up rule helps time box rule

During the Daily Scrum each Team Member answers:

* What did you do yesterday?
* What will you do today?
* What impediments are in your way?

Good practise to update the Scrum Backlog and Sprint Burndown chart at the Daily Scrum.

**The Sprint Review – Show Your Work!**

* Team presents what it accomplished during the Sprint
* Typically takes the form of a demo of new features or underlying architecture
* Informal – 2 hour prep time
* Whole team participates
* Invite the world!
* Feedback informs future work

**The Sprint Retrospective**

* This is the Kaizan part (Continuous Improvement)
* Discuss what is and is not working
* Typically 15 – 30 minutes
* Done after every Sprint Review
* Whole team participates

Ask 3 Questions:

* What will we start doing?
* What will we start doing?
* What will we continue doing?

## Activities

**Product Backlog Planning**

* Typically 5 – 15% of the Team’s time will be planning
* Planning happens continually during the Sprint
* Planning meetings facilitated by the Product Owner
* Always account for time taken for planning during the Sprint

Purpose / Aims

* Used to build the Product Backlog
* Centres around estimating Product Backlog Items
* Often elicits overall system approach or architecture
* Must be facilitated well to avoid digression

**The Sprint**

* Scrum projects make progress in a series of Sprints
* Typical duration in 2-4 weeks or a calendar month at most
* A constant duration leads to better rhythm
* Product is designed, coded, and tested during the Sprint

Non-sequential execution is key – Do a little of everything rather than all of one thing

Goals Don’t Change in the Sprint

* Sprints are sacrosanct
* Commit to keeping change out of the Sprint
* Even a high change environment can commit to a single plan for only 2 weeks
* Plan Sprint durations around a willingness to commit

## Scaling Scrum

Scaling Up with Scrum of Scrums

* A meeting with Scrum Masters and organisational leaders
* Used to co-ordinate the work of multiple Scrum Teams
* Needs executive sponsorship, ideally attendance
* Executives may own team impediments
* MUST have people in attendance who can make resource decisions
* May not be needed daily

Scaling Up – Answer 4 Questions

1. What has your team done since we last met?
2. What will your team do before we meet again?
3. Is anything slowing your team down or getting in their way?
4. Are you about to put something in another team’s way?

# Agile Planning

## Traditional Plans

A Good Plan Is:

* Clear
* Reliable
* Used
* Available

Idea: Can I explain this to my mother?

Traditional Plan:

* Plans activities not deliverable
* Relies on strict sequencing
* If time over runs, it is passed to the next phase
* Developed for systems instead of features
* Asserts that the end result is known

Why We Need Plans

* Reduce Risk
* Make Informed Decisions
* Reduce Uncertainty
* Establish Trust
* Convey a tangible vision
* So that customers can depend on you

The Unspoken Reality

* We can’t accurately say what will be delivered
* As users see the software the come up with new ideas
* Scope should change as new information is uncovered

## Agile Planning

* Plan constantly, not just at the beginning
* Planning is an activity, not a document
* Don’t try to control change, encourage it
* Be constantly transparent
* Focus on historical performance, not hyper-optimal scenarios
* Changing the plan doesn’t mean changing timing

**Levels of Agile Planning**

Chart, sunburst chart

Description automatically generated

Activities at all levels of the organisation – each level of planning directly supports the one just above it. Less frequent change at higher level (e.g. Mission Statement), at lower levels entirely appropriate for things to be changing almost daily.

**Artifacts**

Chart, sunburst chart

Description automatically generated

## Planning a Release

Basic Types of Release Planning

* Date Target Planning
  + The product will release on a specific date
* Feature Target Planning
  + The product will release when features A, B and C are ready

How does it work in your organisation? ‘We do both’ is not realistic – one or the other will win in the end!

**Rule 1 -** An accurate release plan requires a prioritised and estimated backlog

**Rule 2** - An accurate release plan requires known velocity (amount of work a team can complete in a time period – Best Case, Most Likely, Worst Case)

**Product Roadmap vs. Release Plan**

|  |  |
| --- | --- |
| **Product Roadmap** | **Product Release Plan** |
| Communicate the big picture | Predicts to what extent we are poised to deliver of the Product Roadmap |
| Determine and communicate when releases are needed | Provides tangible targets of functionality and dates backed by the reality of the Product Backlog |
| Determine what functionality is sufficient for each release | Used to make reality-based decisions |
| Focus on business value derived from the releases |  |

## Product Backlog Ownership Techniques

**Themes**

Higher level grouping of a set of product backlog items. Some organisations use ‘Theme’ instead of ‘Project’. They can be organised for any reason necessary.

When you have multiple product backlog owners and multiple product backlogs that feed into one team, having themes as an organisation to group work together is almost indispensable.

Model does require one single point of accountability for ensuring the priorities of the different product are represented fairly. Usually a Chief Product Owner.

**Team Backlog View**

* Team uses this to plan the next iteration of work
* If you are a theme owner and your work items aren’t showing in the Team View, you’re in trouble

**Release Backlog View**

* Absolute reality of what will be in the next release

## Iteration Planning Techniques

**Iteration Planning Meeting** – Pre-Game Meeting

* First day of a new iteration
* Time Boxed
* Everyone involved in the work is present
* The point is to plan the next iteration only
* Goal is to make a to-do list for the upcoming iteration

**Velocity vs. Commitment Based Planning**

|  |  |
| --- | --- |
| **Velocity Based** | **Commitment Based** |
| Uses average velocity over time or uses velocity of last iteration | Team commits based on what they believe to be true right now |
| Most useful with a long historical record | Likely to lead to realistic expectations |
| Unreliable in WHAT will be accomplished | Uncovers future impediments NOW |
| Assumes conditions are constant across iterations | Forces team to be deliberate in their thinking |

**Commitment Based Iteration Planning**

* Discuss the highest priority item on the product backlog
* Decompose it into tasks
* Whole team estimates each task in ideal time
* Team answers “Can we commit to this?”
* If yes, see if we can add another backlog item
* If not, remove this item but see if we can add another item

**Ideal Time**

* How long something would take if
  + It’s all you worked on
  + You had no interruptions
  + Everything you need is available
* The ideal time of a football game is 90 minutes
  + Two 45-minute halves
  + The elapsed time is much longer (2+ hours?)

**Task Decomposition**

Start with a well-defined user story in our product backlog. Using poker planning the team estimates the amount of Story Points for the item. The team now considers for the 1st time, the actual work needed broken into ALL individual tasks needed.

**About Tasks**

* The real work is in the Product Backlog Item
* Tasks don’t typically need a lot of detail
* These items represent a conversation that will occur
* Simply meant to be a to-do item
* Keep it simple

## The Daily Plan

**The Daily Scrum / Stand-up / Planning Session**

* Sharing Commitment
* Communicate daily plans to the team and any observers
* Identify impediments
* Set direction and focus
* Regularly rallying the team builds a stronger team

**Tips for Staying Effective**

* Limit to 15 minutes
* Good stand-ups will feel supportive and respectful
* All team members participate, everyone is heard
* It’s all pig, no chickens – everyone is in it for the team if you’re not, you don’t get a say
* Everyone walks away with actionable commitments
* Co-locate the meetings with information radiators

**Information Radiator**

* A large display of critical team information
* Continuously updated
* Located where the team can see it constantly throughout the day

**Items for the Team Information Radiator**

* Task Board
* A Burnout Chart
* Historical View of Team Velocity
* Current Build Status
* Number of Current outstanding defects
* Number of passing tests
* Current code coverage
* Release Plan

**More Tips to Stay Effective**

* Focus on the Backlog
* Create a parking lot for following up later
  + Problem solving
  + Story telling
  + Impediments
* Signal the end
* Time the meeting
* Keep it fun and interesting
  + Last person to arrive starts the meeting
  + Bring food
  + Fine latecomers
  + Create a ‘Stand-up Duration Chart’
  + Change the Order
    - Draw lots
    - Round robin
    - Pass the token

**Signs the Stand-up is Going Off-Track**

* Starting late
* Missing team members
* Meeting overload
* Socialising
* Gloom and doom
* Impediments aren’t raised
* Impediments aren’t resolved
* The Storyteller taking up all the time micro-analysing

# Scrum in Enterprise Environments

## Co-Existence Models

* Waterfall at the End – Iterative development then hand off to waterfall process for release, regulatory stuff, etc. = Irritating -
* Waterfall up Front – Lot of paperwork at the start before iterative development = Painful
* Waterfall in Tandem – Two or more teams some Agile, some waterfall – same product = Deadly