Product Backlogs Using User Stories

User stories describe pieces of software to deliver in the language of someone who will use the software. A **short story title** is written on a card, sticky, or in a list as **a token for conversation**. Stories split into smaller stories and gain more detail over time and through many conversations with contributors in every role.

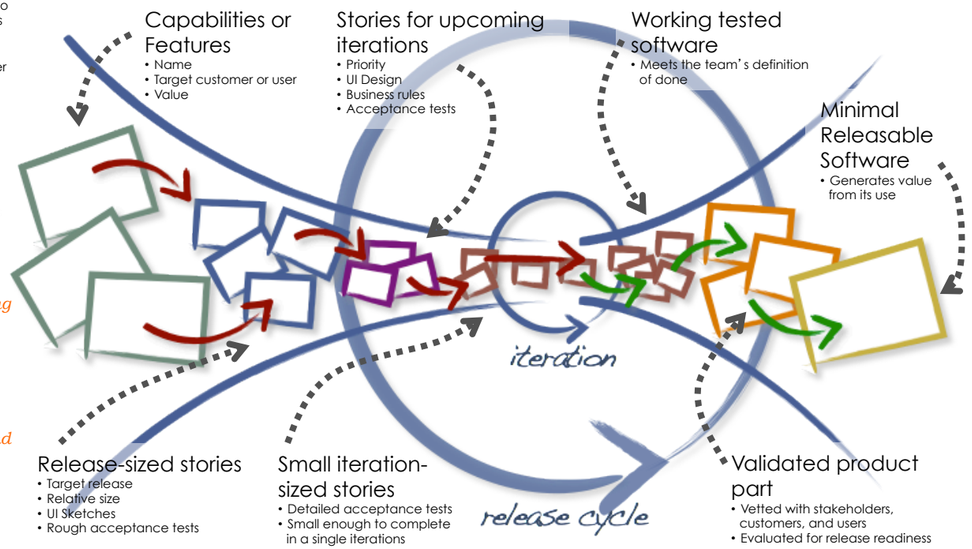
At minimum a story needs:
- Concise title
- Description
- Acceptance criteria

This popular template helps to think through stories from a user and benefits perspective:

as a [type of user]
I want to [perform some action]
so that I can [reach some goal]

Use this template as a thinking tool, not a writing format. Stories without concise titles become hard to organize, discuss, and prioritize.

*Think about **who** uses the software, **what** they'll do, and **why**.*



Visible Progress

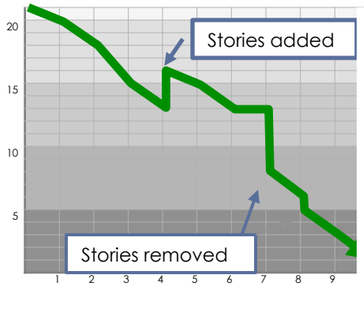
Burn Charts

A burn chart makes progress over time visible. A ***burn down*** chart shows work remaining, while ***a burn up*** chart shows work complete. Watching the slope of the curve on the chart lets us detect early if we're on track to finish all the scheduled work on time.

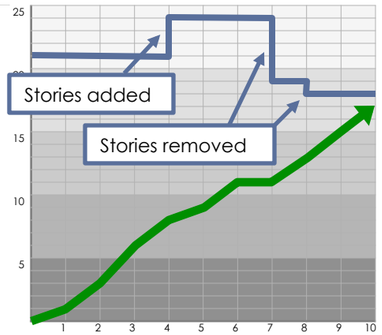
Iteration Day	Story Points In Scope	Story Points Completed	Story Points Remaining
1	21	1	20
2	21	3	18
3	21	6	15
4	24	8	16
5	24	9	15
6	24	11	13
7	19	11	8
			5
			3
			1

A spreadsheet houses the daily totals

Burn Down Chart



Burn Up Chart



Daily Standup or Daily Scrum

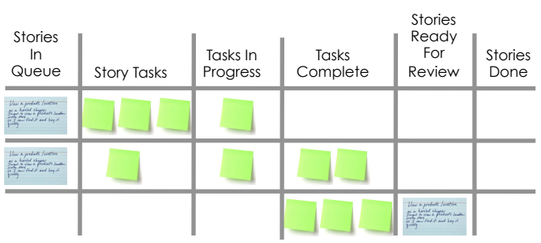
Daily, usually at the start of the day, the team meets for a short planning meeting. Keep the meeting less than 15 minutes. Focus the discussion on what it will take to meet iteration goals.

Each person think about:

1. What you worked on yesterday
2. What you plan to work on today
3. What issues you have blocking them from getting work done

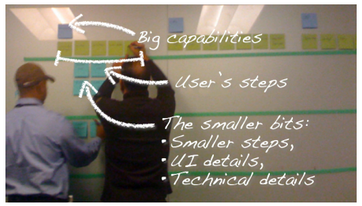
Task Wall

A task wall is used to visualize work in progress during the development cycle. Moving stickies across the wall helps the team coordinate, and signal progress to each other and people outside the team.



Organize user stories as a map

Organize large capabilities roughly in the order users use them. Decompose them top to bottom. This makes the backlog easy to understand and easy to prioritize.



Agile Development & Scrum

Principles and process guide

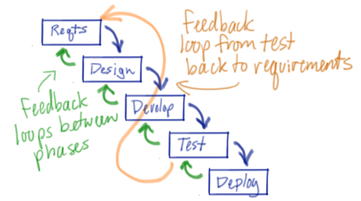
Agile Development is nothing new

The origins of agile thinking are as old as software development. What we call "Agile Development" today is a contemporary word applied to a process style that's been emerging since the 1970s.

Waterfall considered risky

In his 1970s paper Winston Royce is credited with drawing the model we commonly think of as the "waterfall model." But he drew the diagram in a progression on his way to more sophisticated models. He pointed out that you'll need feedback loops between every phase, and feedback all the way from test back to requirements. Royce was trying to describe an iterative development cycle, NOT a sequential process.

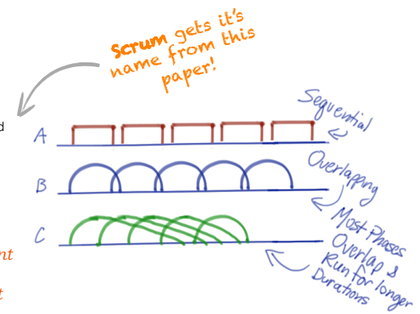
"I believe in this concept, but the implementation described above is risky and invites failure."
--Royce



More like a team sport, less like an assembly line

In the 1986 Harvard Business Review paper "The New New Product Development Game," Takeuchi and Nonaka studied a variety of product design teams and found that the most successful had derived a process that looked chaotic on the outside, but was ultimately faster and more effective at creating great products. They compared these teams work-practice to the sport Rugby.

"Under the rugby approach, the product development process emerges from the constant interaction of a hand-picked, multidisciplinary team whose members work together from start to finish." – Takeuchi & Nonaka



"Agile" identifies specific **values and principles**

In 2001 17 software professionals gathered to discuss what they had in common. They were concerned about the continued growth of formalized waterfall style process. These professionals disagreed on specific process, but agreed strongly on the values and principles that drive process creation.

Scrum was one of those agile processes. Today Scrum has grown in popularity. While teams may identify their process as "Scrum." the way they work borrows practices from most other agile processes.

The agile manifesto describes the values and principles that guide process decisions.

The Agile Manifesto:

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 (below in this case), we value the items on the left more (bold and above in this case).

You can find the manifesto & principles at:
http://agilemanifesto.org/

Remember, the goal isn't to follow an agile process, but to consistently deliver high-quality software your customers and users love.

The principles of Agile development

The principles of Agile Software Development, written the same time as the manifesto, give principles that help guide choices in how we choose to work.

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

Values & Principles Guide Process

Values & Principles

Values describe what's important to individuals, and collectively to an organization.

Principles are the rules of thumb we create and use to make day-to-day process decisions.

Your Context

The process you follow needs to take into account your context.

- How many people are involved?
- How risky is the project?
- How time-sensitive is delivery?
- How difficult is the problem you're solving?
- What does "quality" mean in your context?

New Knowledge

We continue to discover new concepts and leverage them to create new and better practice and tools. For example:

- A system's throughput is only as fast as its slowest step so we use Kanban style boards to visualize the flow of work and find bottlenecks faster.
- Product ideas are hypothetical until proven out after delivery so we use MVP Tests to validate product ideas before delivery.

Process & Practices

Practices like test-driven development support values like being able to respond quickly to change, and principles like technical excellence.

Processes combine practices that support each other and are used by different roles.

Processes help many people coordinate their activities to accomplish a common goal.

Roles

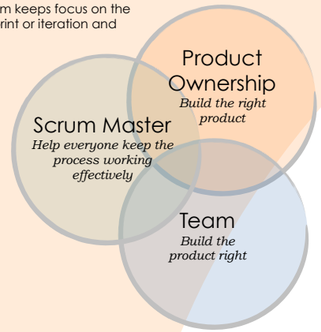
Agile processes and Scrum use three "super roles" that satisfy the concerns of building the right product, the product right, and tuning the process so people can perform at their best. Traditional software development roles are often mapped to one or more of these super roles.

Remember these roles represent concerns we all share and not individual people. In healthy agile teams people will "change hats" from one role to another.

Scrum Master or Coach

The scrum master focuses on process success

The Scrum Master focuses on making sure the process is working, that everyone understands and fills their role, that collaboration is effective, that visibility is kept high, and that the team keeps focus on the goals of the current sprint or iteration and product release.



Product Ownership

Product owners focus on product success

A single person may have the title "product owner," but the ideal product owners is a great leader. Since a successful product must be valuable, usable, and feasible, a product owner usually leads a small **product ownership team** that includes the skills and roles needed to be successful:

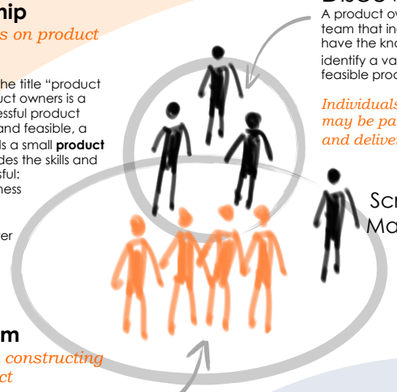
- Product manager or business representative
- UX practitioner
- Engineer, architect, & tester

The Delivery Team

The team focuses on constructing a high quality product

The team is composed of all the roles and skills necessary to build, test, and document software of sufficient quality that it could be released. This includes:

- engineers
- architects
- testers
- business analysts
- UI designers
- technical writers



Discovery Team

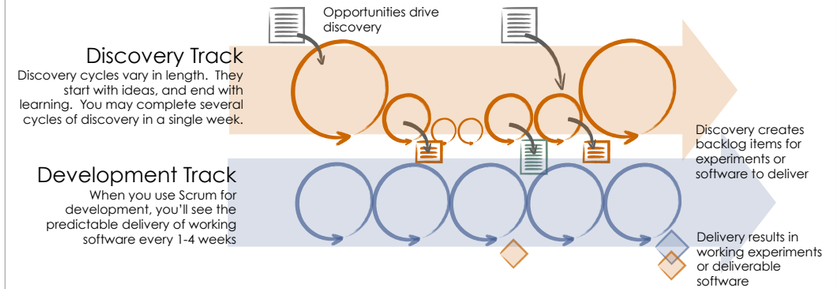
A product owner leads the discovery team that includes individuals that have the knowledge and skills to identify a valuable, usable, and feasible product.

Individuals like lead engineers may be part of both discovery and delivery teams

Scrum Master

Dual-Track Development

Discovery work happens continuously alongside development work.



Process

This starting process builds from the essential Scrum Framework to add practices that wrap the sprint for product discovery, getting to ready, and getting to release.

Remember that process isn't skill. Success or failure is up to the whole team. The right process gives just enough structure for teams to effectively collaborate.

Product Discovery

Using collaboration, design thinking, and validated learning answer the questions:

1. What **problems** are we solving and for who?
2. What will customers and users **value**?
3. What can users **use** to reach their goals?
4. What's **feasible** to build given the tools and time we have?

The speed of discovery is measured by cycle time of learning. Try to experiment and learn several times per week.

Opportunity Backlog

Ideas to explore, and problems to solve.

View your organization's roadmap not as fixed items to deliver, but as ideas to explore and validate.

Product Backlog

Deliverable pieces of a product or experiment we'd like the team to build

While discovering, use backlog items for high-fidelity prototypes or live-data prototypes that requires development time to create

Product Delivery

The product delivery team strives for predictable, high quality, delivery.

Product Team Planning

The product team meets routinely to discuss release progress, select stories for upcoming sprints, and plan the work needed to get stories ready for the delivery team.

Story Workshop

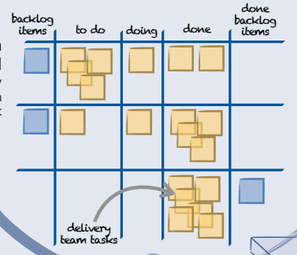
Product team members meet with delivery team member regularly to work through story details and agree on acceptance criteria

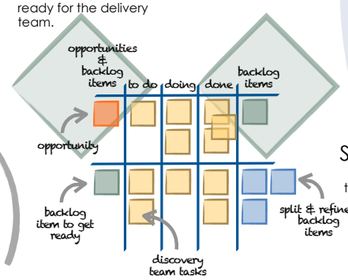
Sprint Planning

The product owner shows up "ready" with details for high priority backlog items. The team builds their plan and commits.

Delivery Team Task Board

Use to routinely plan and re-plan delivery work





Sprint Backlog

The delivery tasks that turn backlog items into working software

Product Team Task Board

Use to routinely plan and re-plan discovery work and work to get items ready to deliver.

Release Burndown

Makes release progress visible. How much is left to build before we can release?

Sprint Burndown

Makes progress visible during this sprint. Are we making progress?

Sprint

A fixed time-box for delivering software usually 1-4 weeks

Day

The smallest cycle of work – you can't extend this one if you don't finish what you planned

Daily Scrum

Reflect on what was done the prior day
Plan what to do today
Raise issues stopping progress

Sprint Review & Retrospective

Demonstrate and critique the working **Product**

Discuss the **Progress** relative to the plan

Reflect on the way you've been working (your **Process**) and change it as necessary

Potentially Shippable Software Increment

It may take more software to be valuable to users, but it had better not require more testing and bug fixing

Customer & User Product Testing

Routinely validate the working software with genuine users and customers

Stakeholder Product Review

Keep progress visible to stakeholders outside the team

Monitor Released Software

Almost every day take a look at metrics , bug reports, and customer feedback for software you've already released

Viable Product Release

Built iteratively and incrementally each sprint, this is the product that'll get you the outcomes you want from the users you've targeted.

Artifacts

Artifacts, or things we see and touch. They help make information, ideas, and status visible to the whole team. Basic artifacts include backlogs, burndown charts, and task boards.

If no one uses the artifacts, remove them because they're likely not working.

Product Discovery

Pull from a backlog of opportunities use discovery work to identify the product you should build.

- Understand the business motivation for building the product
- Understand the customers and users that will use your product and the problems your product solves for them
- Ideate: consider multiple solutions to solve your customer's problems
- Iteratively build and test prototypes with your customers and users
- Use backlog items to develop higher fidelity and live-data prototypes where you'll need development work
- Describe your validated solution using stories in a product backlog you can use to manage the delivery of your software.

Work together to explore details and describe the software you want to build in the next time-box.

Getting to Ready

To help the team move quickly and predictable, your product team lead by a product owner must better understand and describe the details of what to build.

- The **product team plans** by identifying **user stories** 1-3 iterations in advance. The product team identified that work they'll need to do to design and describe the software to build.
- **Collaborating with the team** using a workshop approach to build shared understanding about what to build. Members of the product team and delivery team work together to answer these three questions: What exactly will be build? How will we know when we're done? How will we demonstrate this new software to others?

Work together to explore details and describe the software you want to build in the next time-box.

Sprint

(Iteration or Development Time-box)

Iterations (Sprints in Scrum) are short fixed time-boxes, usually 1-4 weeks. At the end of each iteration demonstrate finished, tested, high quality software. While it's critical to keep the software high quality, don't expect it to be releasable after early iterations.

- The team plans the sprint by working through the details of what they'll need to build
- Keep visibility high during the iteration using a task wall or burn chart
- Keep collaboration high between the product team and delivery team
- At the end of the iteration use a **product review** to evaluate the product and discuss the pace of delivery, or **velocity**
- Use a **heartbeat retrospective** to adjust your process.

Use Sprints to build software, measure, and learn. Use what you learn to improve your product, and the way you work.

Getting to Release

The team may be confident in what they've built, but we'll need to validate it with customers, users and stakeholders.

- Take working software out to customers and users to test it. Is it valuable? Is it usable?
- Keep progress and user feedback visible to stakeholders.

Continue validating the software you've built with customers, users, and stakeholders to avoid surprises at release time.

Learn after Release

After the release monitor metrics and scorecards, and observe people using your product. Remember, your project may be over, but the life of your product has just begun.

The most valuable improvements to your product come from observing people using it today.