System Development

Scrum

Datamatiker /Computer Science

2nd Semester

Fall 2017

• Knowledge of Scrum as a process model

– How to document and estimate customer requirements

– How to turn requirements into an operational format the

developers can use to control their daily work

– How to monitor and manage the development effort

– How to calculate team velocity, meaning how much work a team can handle in time-boxed period

– How to work in an iterative manner where software is build

piece by piece

**Henrik Kniberg** *Scrum and XP from the Trenches*

<https://www.infoq.com/minibooks/scrum-xp-from-the-trenches-2>

Pages : *pp. 1-13 day 1*

*pp. 14-50 day 2 pp. 51-68, 75-92 day 3*

How to Develop an IT System?







Phase 1 – idea/analysis Phase 2 – design



Phase 3a – fundament Phase 3b – walls Phase 3c – root

Phase 3 – construction

Phase 4 – test

General comparison of two methodology paradigms

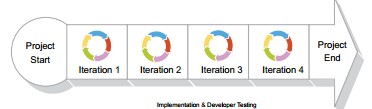
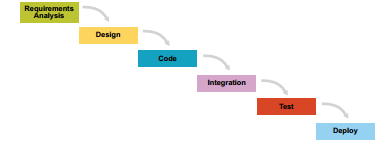
Traditional “waterfall” development depends on a perfect understanding of the product requirements from the beginning and minimal errors made in each phase.

Scrum in a Nutshell



• The Scrum is iterative process

– Many small water falls, usually called **sprints**

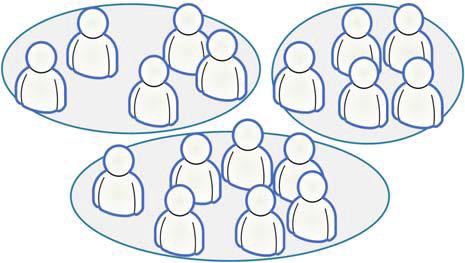


**TRADITIONAL WATERFALL**

**SCRUM - ITERATIONS – MINI WATERFALLS**

• Split your organization into small, cross-functional, self

organizing teams.



*Source: Kniberg ” KANBAN AND SCRUM – MAKING THE MOST OF BOTH”*

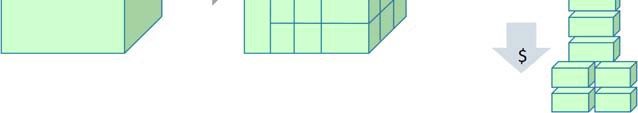
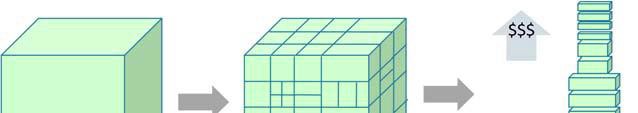
• Split your work into a list of small, concrete

deliverables.

– Sort the list by priority

– Estimate the effort of each item

*Source: Kniberg ” KANBAN AND SCRUM – MAKING THE MOST OF BOTH”*



• Split time into short fixed-length iterations (usually 1 – 4

weeks), with potentially shippable code demonstrated

after each iteration.





• After each iteration …

– Optimize the release plan and update priorities in collaboration with the customer, based on insights gained by inspecting the release

– Optimize the process by having a retrospective after each iteration.

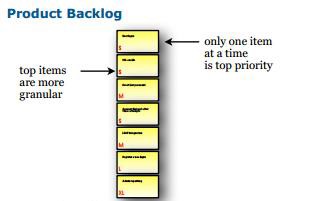
*Source: Kniberg ” KANBAN AND SCRUM – MAKING THE MOST OF BOTH”*

• A prioritized list of everything that might be

needed in the product

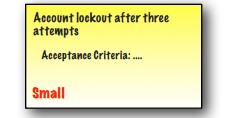
– requirements, features etc.

– things that the customer wants, described using

the customer’s terminology

• Often called (user) story, or just PBI.

• Example:

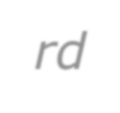
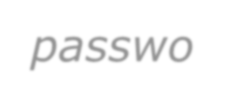
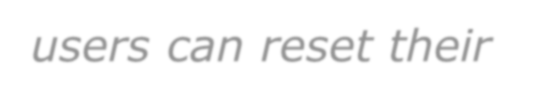
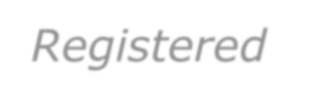


• … is short, simple description of a feature told from the

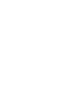
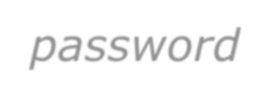
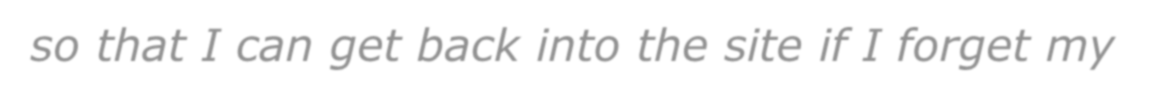
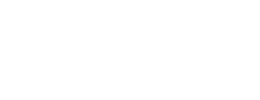
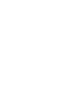
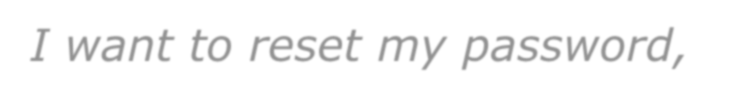
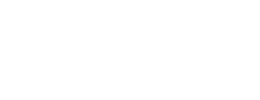
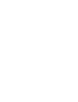
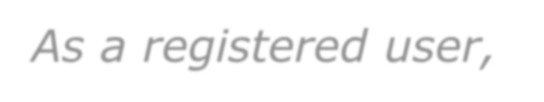
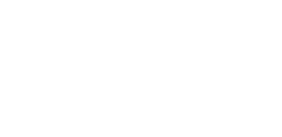
perspective of the person who desires the new capability

(typically user or customer)

• User stories can be written informally:



*Registered users can reset their password*



• Or use a more formal template

*As a registered user,*

*I want to reset my password,*

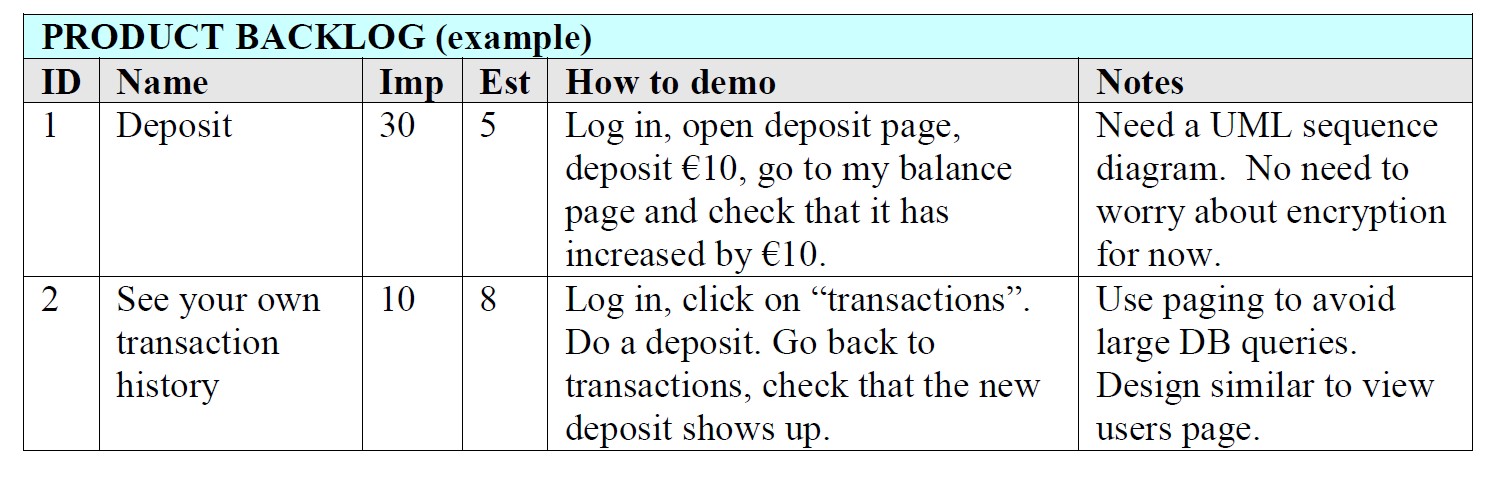
*so that I can get back into the site if I forget my password*

• Notice that a feature description is specified in “How to

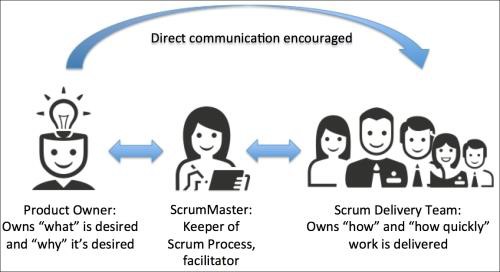
demo” field = description of test steps (acceptance

criteria) (Kniberg p. 10)

**17**



Scrum Roles



**Responsible for the business value of the project**

**Responsible for the team is functional and productive**

**Responsible for getting the work done – is self- organized**



• Represents the stakeholders (= customer voice)

• Is responsible for maximizing product value

• Is responsible for managing the PBL:

– Create Product Backlog items (user stories)

– Prioritize Product Backlog items

– Ensure the teams understands items

• The Scrum Master is the process owner

– responsible for ensuring Scrum is understood and enacted

– Helps the team perform at their highest level (coach)

– Protector of the team



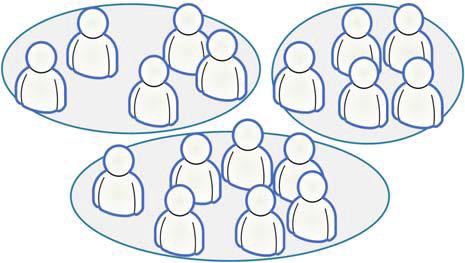
• Cross functional

• Self-organizing

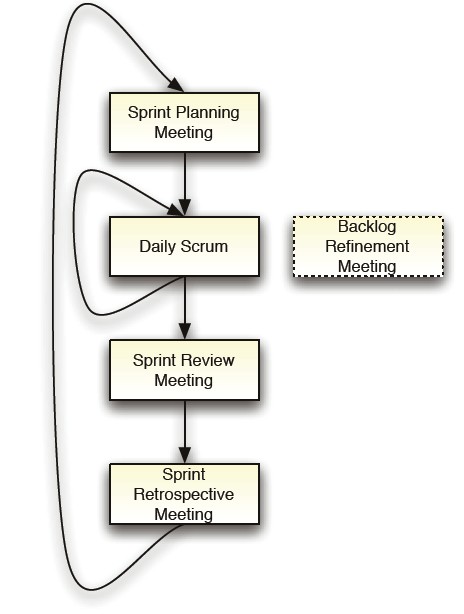
• Negotiates commitments with the Product Owner, one sprint at a time

• Has autonomy regarding how to reach commitments

• Collaborative

• Co-located

• 7 ± 2 members



• Scrum meetings

• Let’s “attend” a backlog refinement meeting by

watching a video (13 minutes++):

<http://scrumtrainingseries.com/>

• We will see Product Owner, Scrum Master and Team in

action!

• Home work: Watch 15 minutes video by Henrik

Kniberg

[http://blog.crisp.se/author/henrikkniberg](http://blog.crisp.se/?s=Agile%2BProduct%2BOwnership%2Bin%2Ba%2BNutshell)

• Make team contract

– Consider Scrum Master role

• Material for inspiration

– [Agile manifesto](http://agilemanifesto.org/)



PAUSE

Good User Stories



|  |  |  |  |
| --- | --- | --- | --- |
| 1. | Identify stories | = | PO responsibility |
| 2. | Write stories | = | PO responsibility |
| 3. | Estimate stories | = | team responsibility |

Apply INVEST criteria for each story

I – Independent

N – Negotiable

V – Valuable

E – Estimable

S – Small

T – Testable

• Stories are easiest to work with if they are independent.

• We'd like stories to not overlap in concept

• We'd like to be able to schedule and implement stories

in any order.

• A good story is negotiable

• Story isn’t an explicit contract for features; Rather, details will be co-created by the PO and Team.

• A good story captures the essence, not the details

• A story needs to be valuable to the customer

• What about Tech Stories? (H. Kniberg p. 39)

– Examples:

• Install continuous build server

• Write a system design overview

• Refactor the data layer

• Update bug tracking system

– What do to?

1. Transform into normal story

2. Be a task in another story

3. Define and keep in separate list

– Let Product Owner see, but not edit

– Negotiate with Product Owner

• Valuable – to who?

– Customer (purchaser & user)

– Secondarily developer

Examples:

**Valued by purchaser, but maybe not the users**:

*“All configuration information is read from a central location”*

*“The development team will produce the software in accordance with CMM Level 3”*

**Valued by both customer and developer**… if changed from

*”All error handling and logging is done through a set of common*

*classes”*

into this text:

*“All errors are presented to the user and logged in a consistent*

*manner”*

• A good story can be estimated

• We don't need an exact estimate, but just enough to help the Product Owner rank and schedule the story's implementation

• Being estimable is

– partly a function of being negotiated, as it's hard to estimate a

story we don't understand

– Also a function of size: bigger stories are harder to estimate

• And of the team: what's easy to estimate will vary

depending on the team's experience

• Why difficult to estimate stories?

1. Developers lack domain knowledge

2. Developers lack technical knowledge

3. The story is too big

Solutions

1. Discuss with customer

2. Turn into two stories:

a) a quick spike to gather information

b) a story to do the real work.

3. Decompose into smaller, constituent stories

• Good stories tend to be small

• Stories typically represent at most a few person-weeks worth of work (that is actually long time)

– Often teams try to restrict them to a day of work

• Above this size, it seems to be too hard to know what's in the story's scope

• A good story is testable

• Writing “how to demo” accept criteria carries an implicit promise: "I understand what I want well enough that I could write a test for it."

• Will be used in sprint review – is the story done?

Done User Story

**13**



• Bring the project from *”It Works as Coded”* to *“It Works*

*as Intended”*

• Are *conditions* that a story must satisfy to be *accepted*

by a user/customer/other stakeholder (PO in Scrum)

• Are a set of *statements*, each with a *clear pass/fail result*, that specify both functional and non-functional requirements

– Functional example: *When a user clicks on the ‘Reports’*

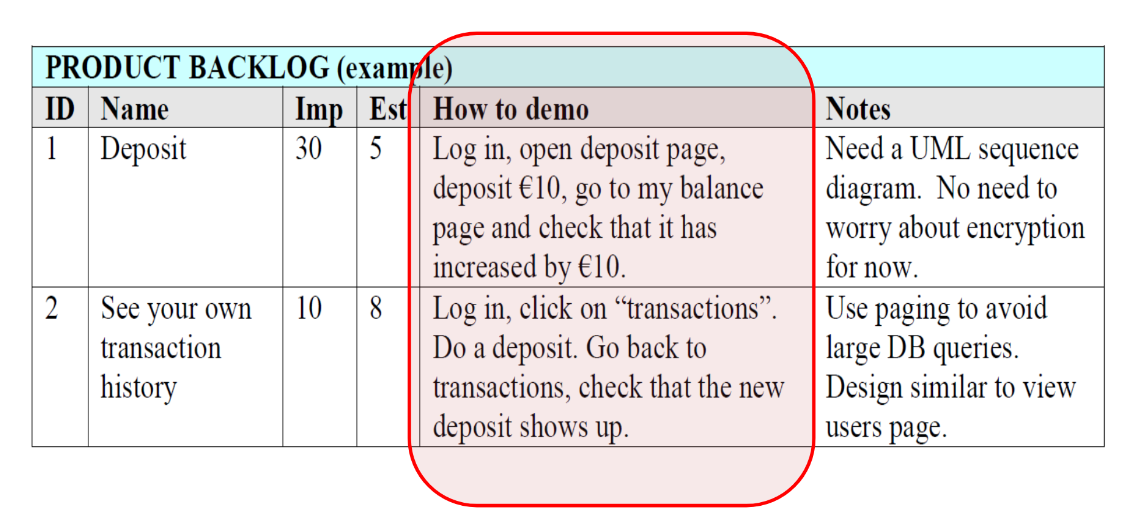
*dropdown, a list of available reports will be displayed*.

– Non-functional example: *Form edit buttons will be blue, and Form*

*workflow buttons will be green.*

Source: <http://www.seguetech.com/blog/2013/03/25/characteristics-good-agile-acceptance-criteria>

Accept criteria:





User Story Estimation

Scrum 2 **16**

• **S, M, L and XXXXL**

• Each estimator has four cards S, M, L and XXXXL (epic)

• Each estimator privately selects one card to represent his estimate for a story. All cards are revealed at the same time

• If consensus, that will be the estimate

• If not, discussion will lead to re-estimation until

consensus

– Possibly decompose stories into smaller stories

• A deck of **Planning Poker** cards with values like **1, 2, 5,**

**8, 13, 20, 40, 100** and ? (I don’t know), coffee cup (I

want a break)

– The values represent number of story points, ideal days, hours, or other unit in which the team calculates its estimations

• Each estimator privately selects one card to represent his estimate for a story. All cards are revealed at the same time

• If consensus, that will be the estimate

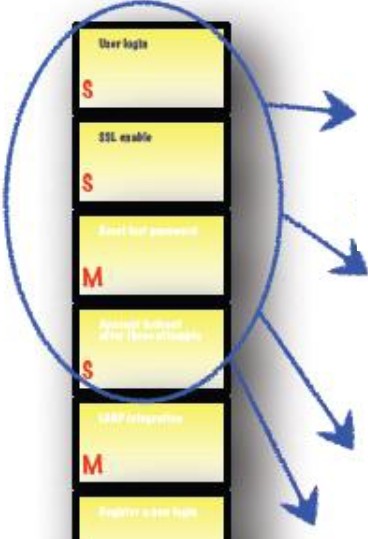
• If not, discussion will lead to re-estimation until

consensus (Possibly decompose stories into smaller stories)

Produet Backlog

Sprint Backlog

UurloQitt



SeIeeted

Produet

Increment

L



XI.

*Source:* [*http://scrumreferencecard.com/ScrumReferenceCard.pdf*](http://scrumreferencecard.com/ScrumReferenceCard.pdf)



• Contains **committed stories** negotiated

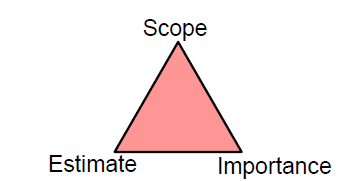
between the team and the Product Owner during the Sprint Planning Meeting

• **Initial tasks** are identified by the team during

Sprint Planning Meeting

• Team will discover **additional tasks** needed to meet the fixed scope commitment during Sprint execution

• Sprint planning meeting with team decision based on:



Scope question example

*“Does the ‘delete user’ story include going through each pending transaction for that user and canceling it?’”*

In some cases the answer will be surprising to the team, prompting them to change their estimates

In some cases the time estimate for a

story won’t be what the PO expected.

This may prompt the PO to change the importance of the story. Or change the scope of the story, which in turn will cause the team to re-estimate, etc. etc.

Story:

As **an online store owner**,

I want **to view my products**

so that **I can review what is current available on my site**

Split story into tasks (examples):

1. Create database table

2. Populate table with a few sample data

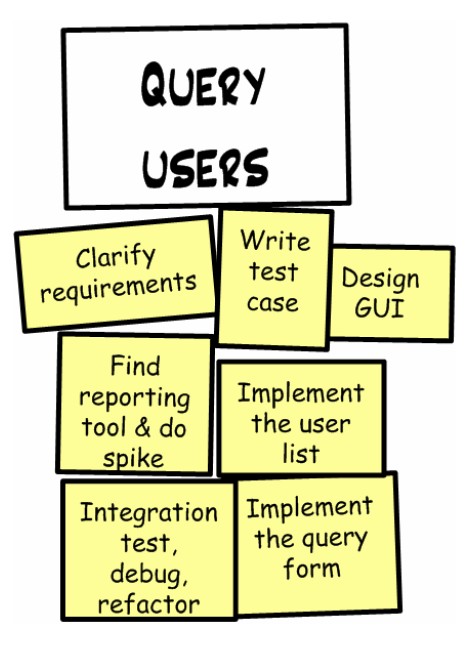
3. Create select SQL script

4. Create UI for viewing my products

5. …

6. Create automated functional tests for viewing functionality

**6**

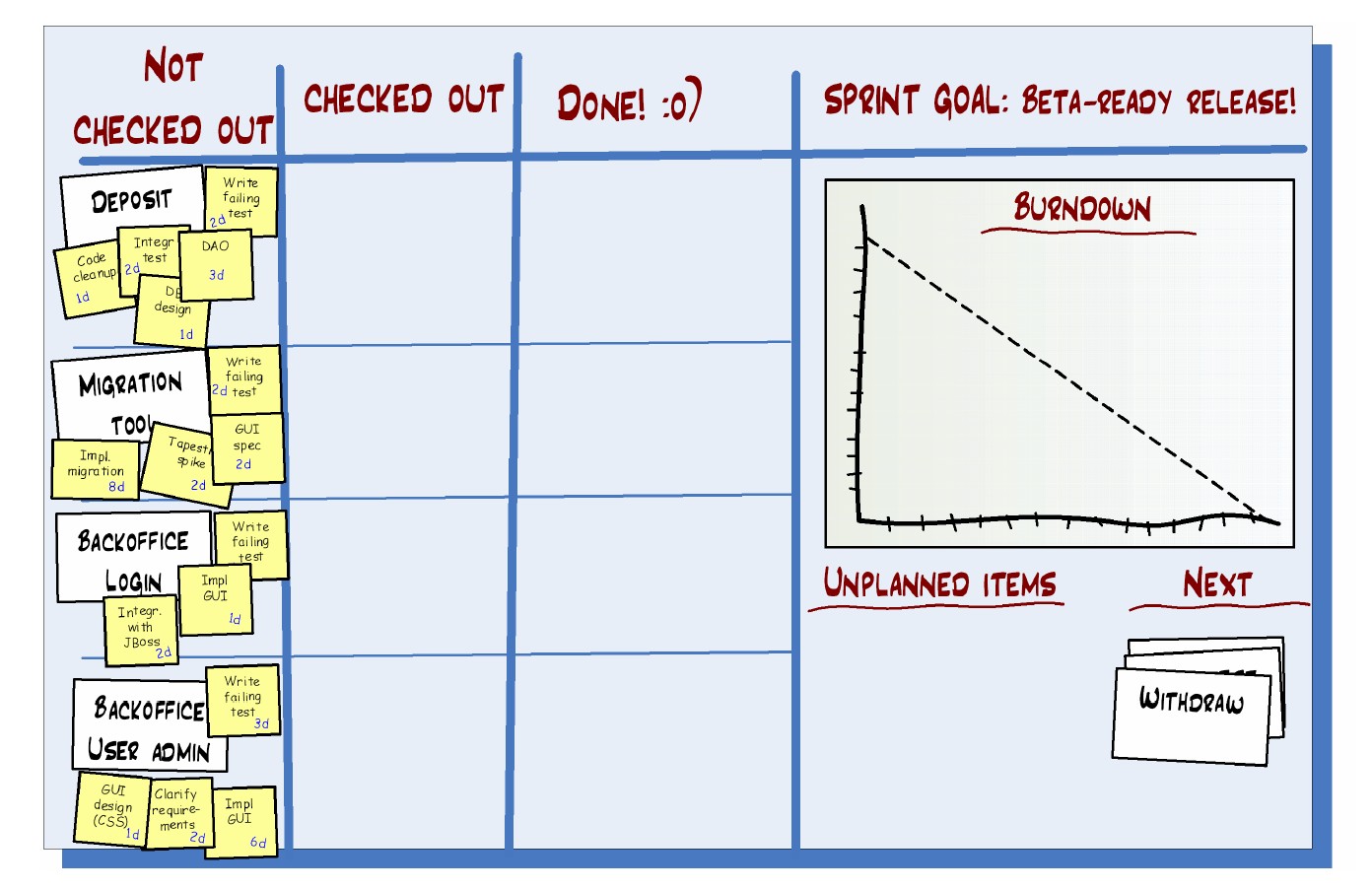
• **Stories:** deliverable things at PO (business value) level

• **Tasks:** non-deliverable things that

PO doesn’t care about

Example:

Scrum 2

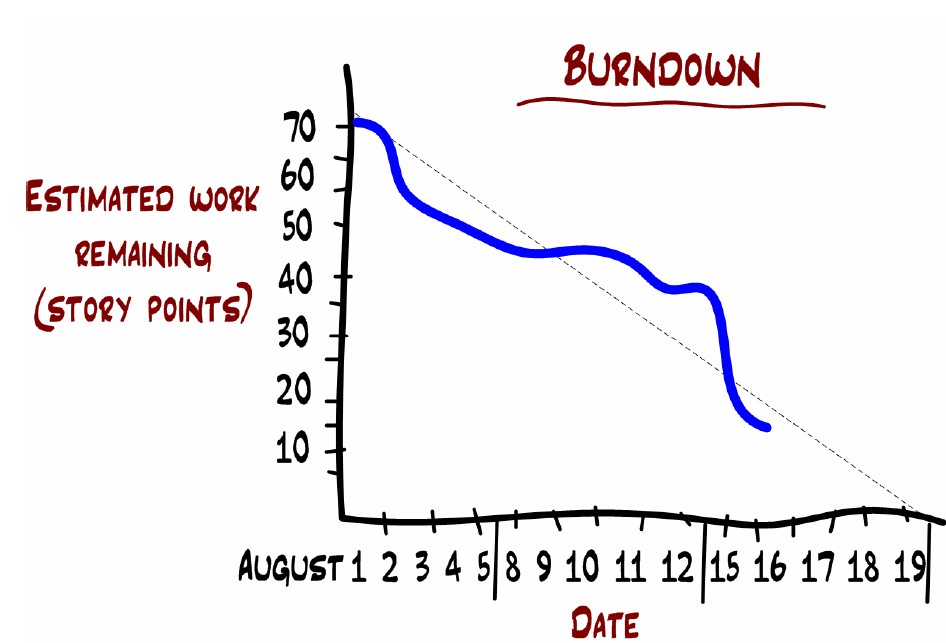


Sprint Backlog Format Kniberg p. 46

Taskboard + Burndown Chart should always visible to team:

**8**

Scrum 2



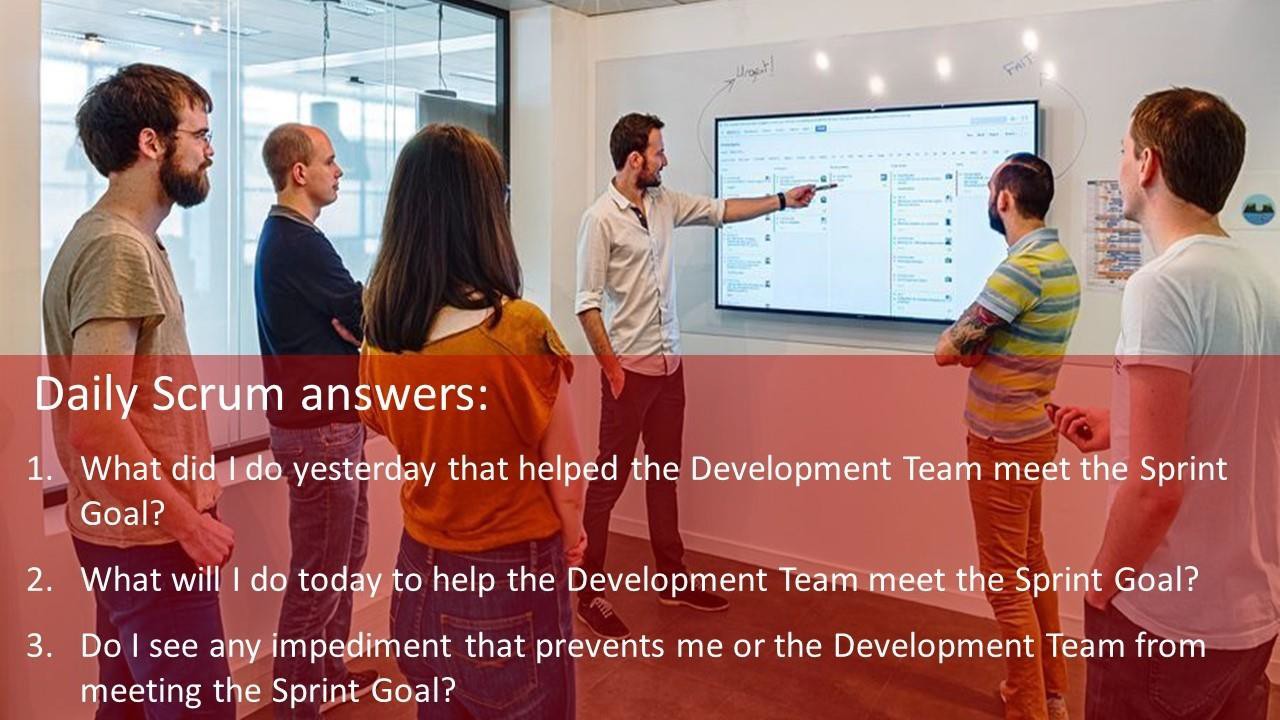
• Tracking progress during sprint.

– The graph shows, each day, a new estimate of how much

work remains until the team is finished.



The daily scrum



• Daily

• 15-minutes

• Stand-up

• Not for problem solving

• Everybody can attend

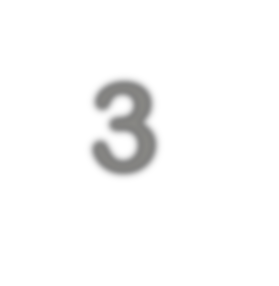
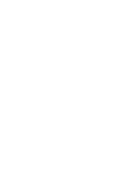
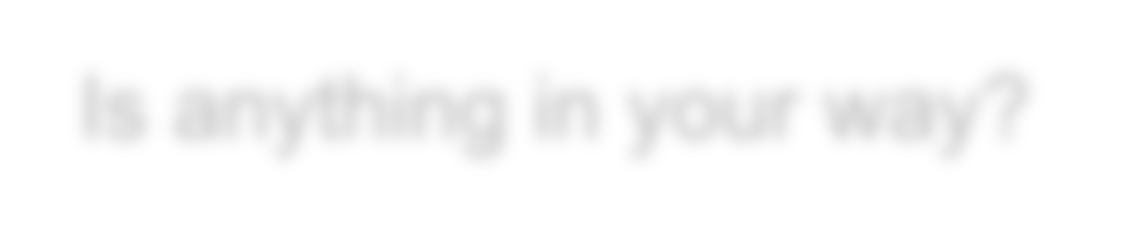
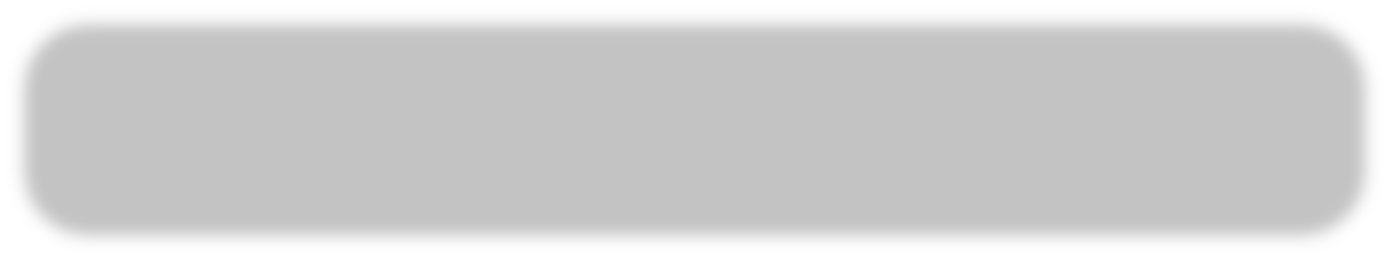
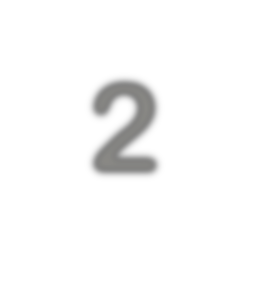
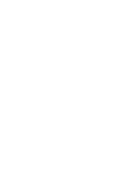
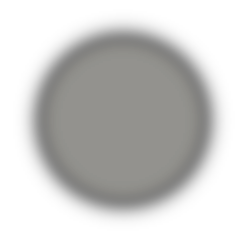
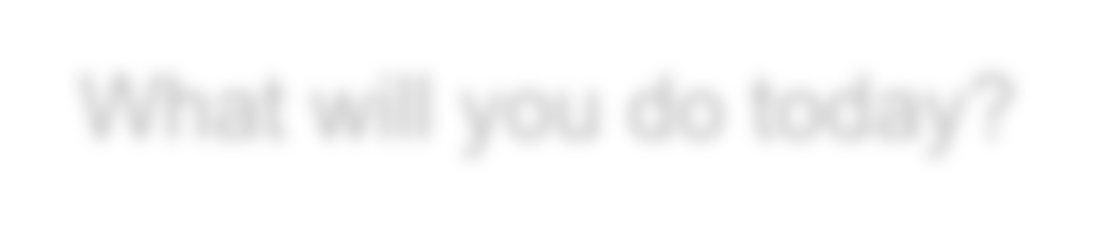
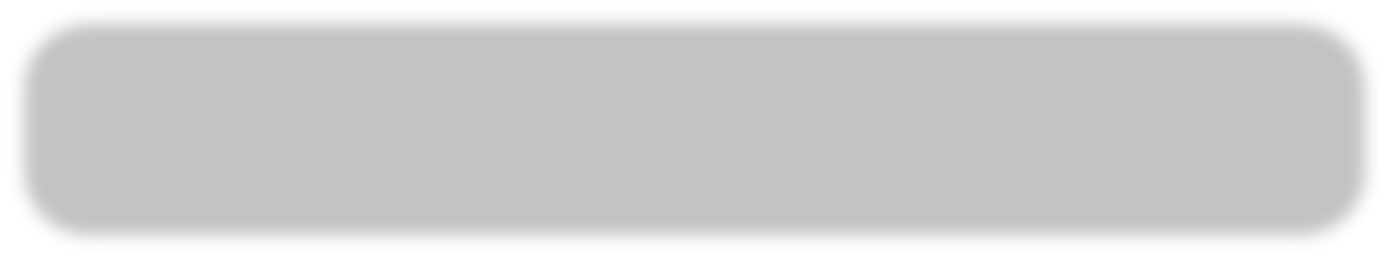
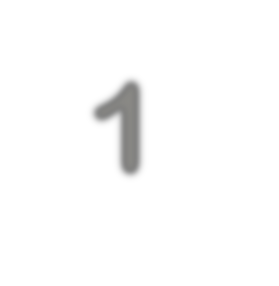
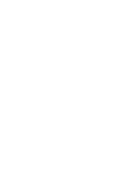
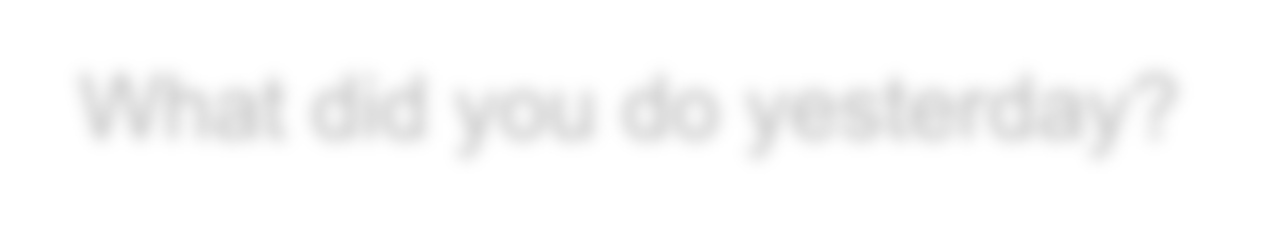
source: https://masterofproject.com/blog/135454/daily-scrum

• Only team, Scrum Master and Product Owner can talk

• Helps avoid other unnecessary meetings

**13**

Everyone answers 3 questions



1

What did you do yesterday?

2

What will you do today?

3

Is anything in your way?

• These are *not* status for the ScrumMaster

• They are commitments in front of peers

**14**

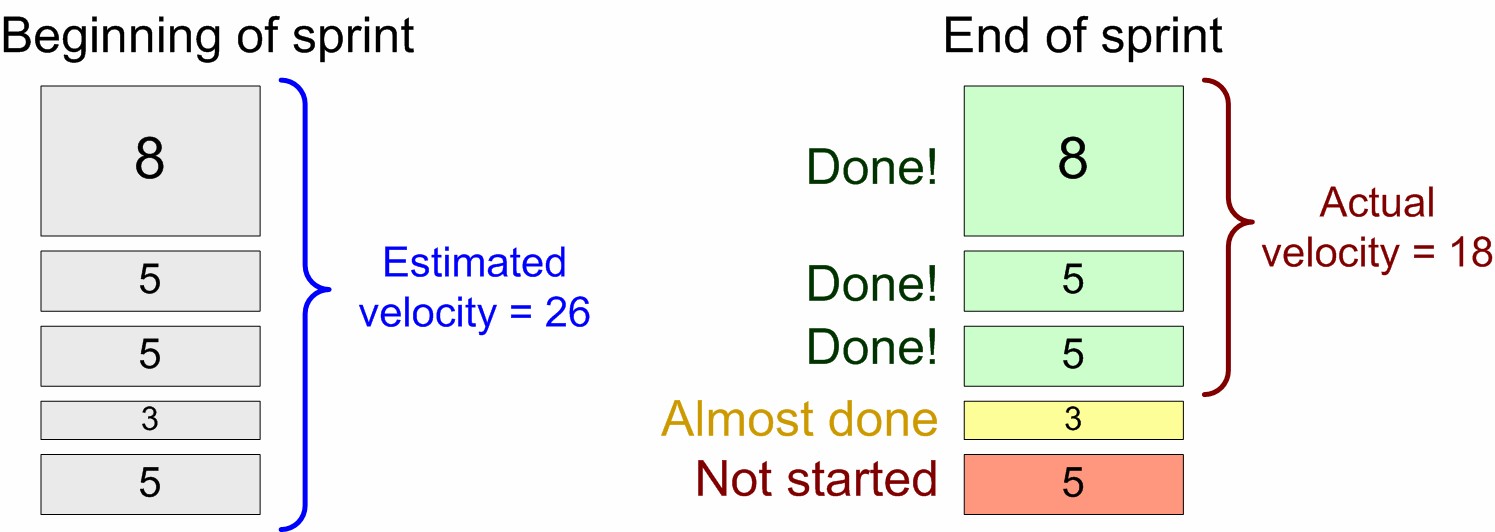
*Can we get better at estimating?*

Velocity is a measure of the amount of work a Team can tackle during a single Sprint and is the key metric in Scrum. Velocity is calculated at the end of the Sprint by totaling the Points for all fully completed [User Stories](https://www.scruminc.com/independent-user-stories/).

A simple way to estimate velocity is to look at team history

– What was their velocity during the past few sprints?

– Then assume that velocity will be roughly the same next sprint.



**[TAIGA.IO](taiga.io)**

**Online Scrum board**