Agile Planning

Petri Heiramo

Agile Coach, CST

An Agile Plan Is Not a “Rough Guide”

- Some teams think that, if they did not finish all stories, that was OK, …”we are agile”
- Postponing stories was seen as an acceptable (and often used) option.
- This is WRONG
- The point of an iteration is to provide a hard dead-line – a time box – to help the team to focus on the really important stuff
- If you drop stories to the next iteration you do not have a time-box at all: you only have regularly scheduled meetings
- That leads to issues of trust and waste

Scrum Planning

- Scrum plans have to be
  - Iterative
  - Incremental
  - Risk and value driven
  - Time-boxed
- Without ALL four, it's not Scrum!
  - It may work for you, but don't call it Scrum

"Often detail adds no more usefulness – only a false appearance of validity."

- Edward de Bono

Involve the Customer!

- Customer and users are involved in almost every activity in planning through Product Owner role and stakeholder participation
- Consider how to make the participation as broadband as possible
  - Live participation is almost always the best
- Continuous participation!

Focus of Scrum Planning

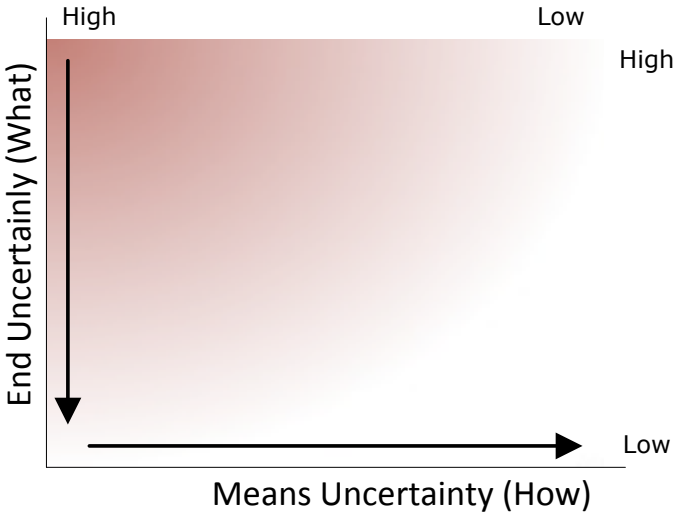
- Strategy
- Portfolio
- Product
- **Release**
- Sprint
- Day

Scrum's focus is on planning at product, release and iteration levels

Daily planning is team's concern

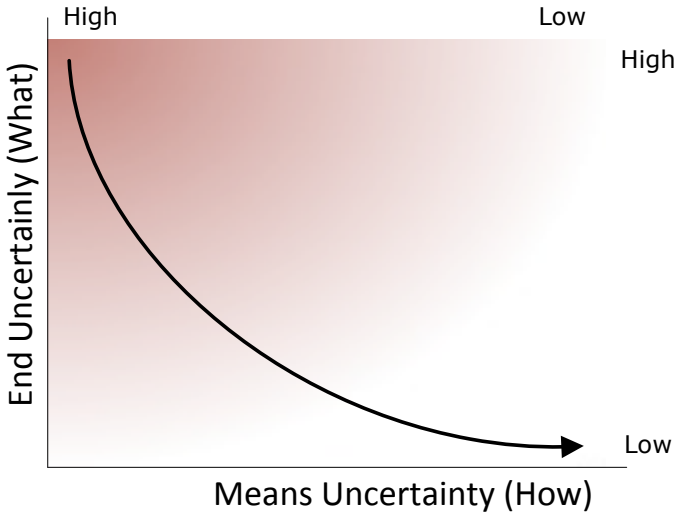
Reducing Uncertainty

- Requirement specification defines “what” first, then design defines “how”



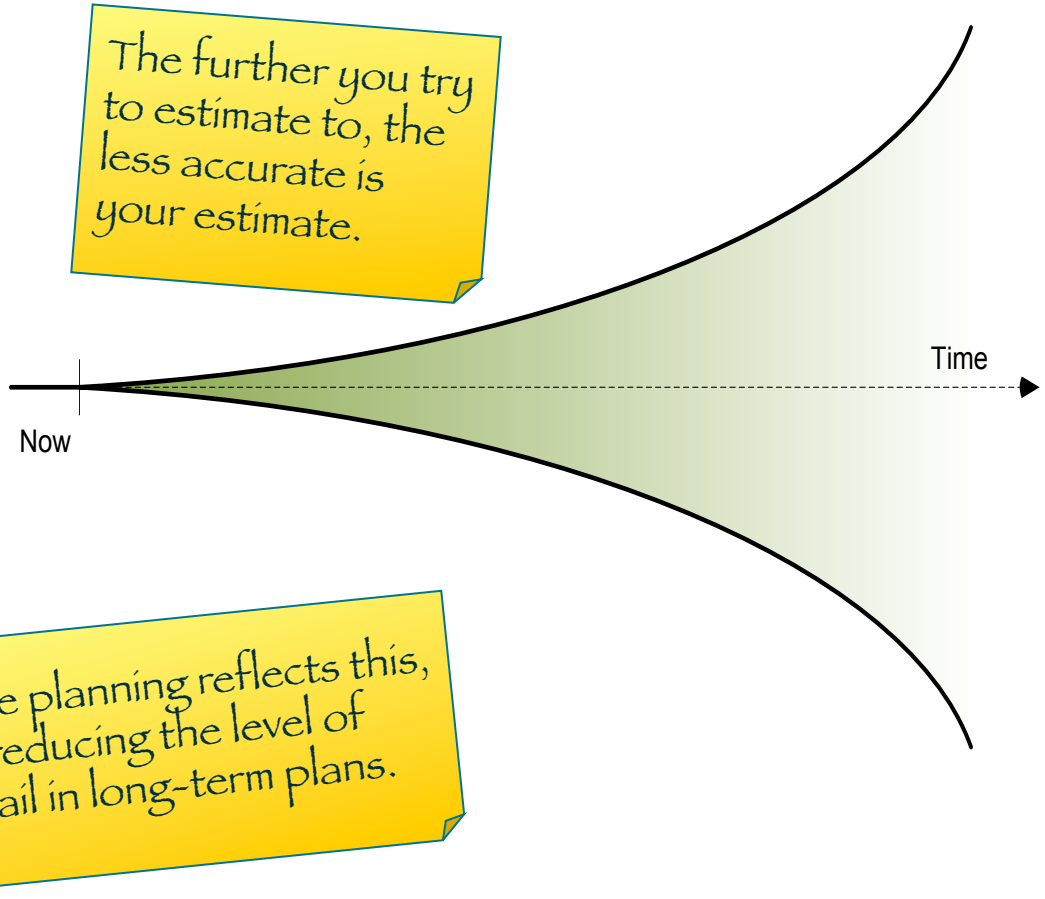
Waterfall / phased

- Initial focus on clarifying “what” with natural transition to “how”

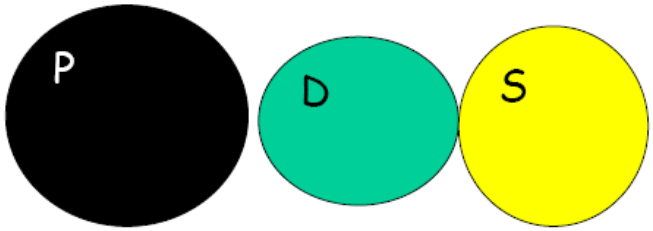


Agile / iterative

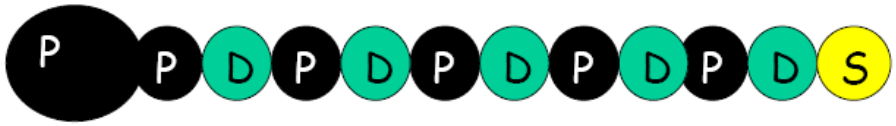
Cone of Uncertainty (Reversed)



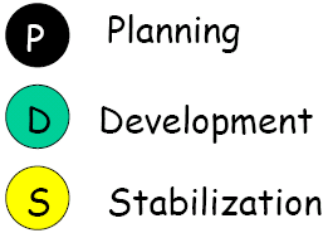
Scrum Planning Is Just-In-Time



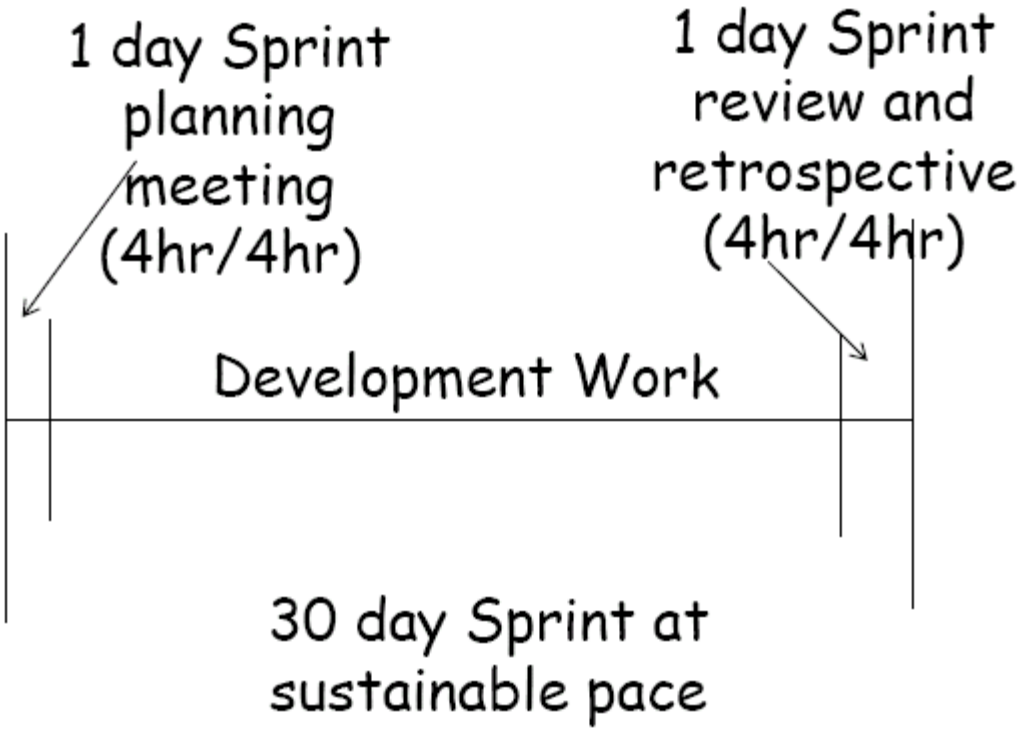
Traditional Project



Scrum Project



Time Spent Planning



For a shorted sprint, use correspondingly shorter time-boxes.

You probably have to do more planning in the background than this, especially early in the project

Also, reserve up to 10% of time for refining future items in product backlog

Making a Plan

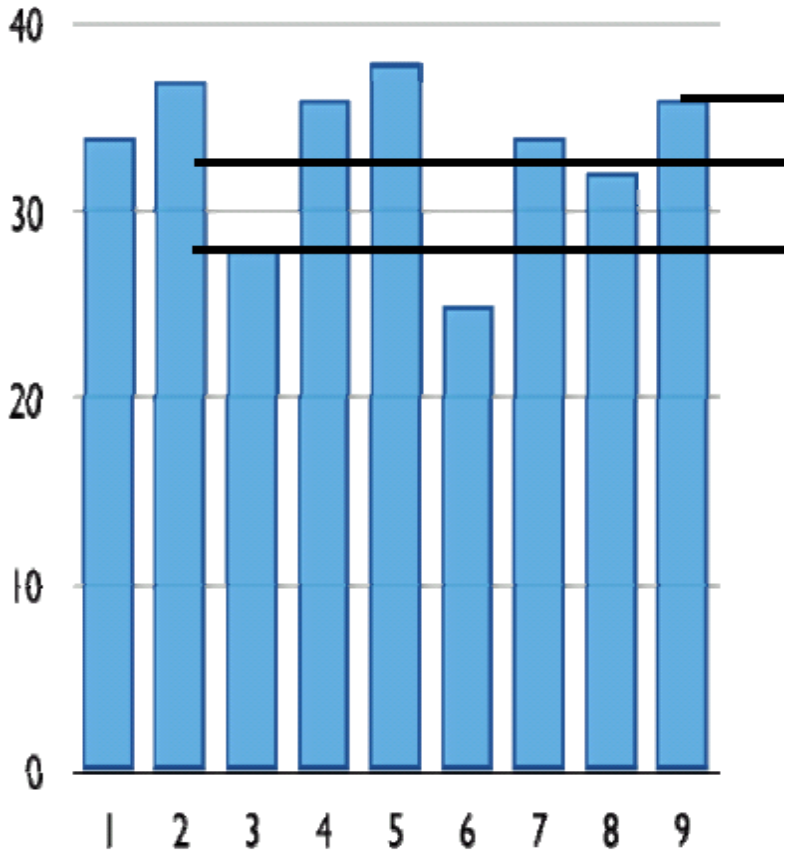
- We already know project size (either in story points or ideal days)
- We need measure for development capacity and progress
- Measure of Progress: **Velocity**
- Measured as realized functionality / sprint
- Initially an estimate, based on past performance in subsequent sprints
  - Either ”yesterday’s weather” or average from several sprints

Velocity is a good planning tool, but a bad metric. Why?

Estimating Project Scope

- Possible Scope = Duration * Velocity
  - Possible Scope means the sum of all features that can be developed in a given Duration with a given Velocity
- The different velocities give different estimates
  - Allows for risk management regarding desired functionality
- Alternatively, if there is certain desired scope
  - Duration = Desired Scope / Velocity
  - Can be used to determine the minimum duration needed before possibility of release

Different Velocities

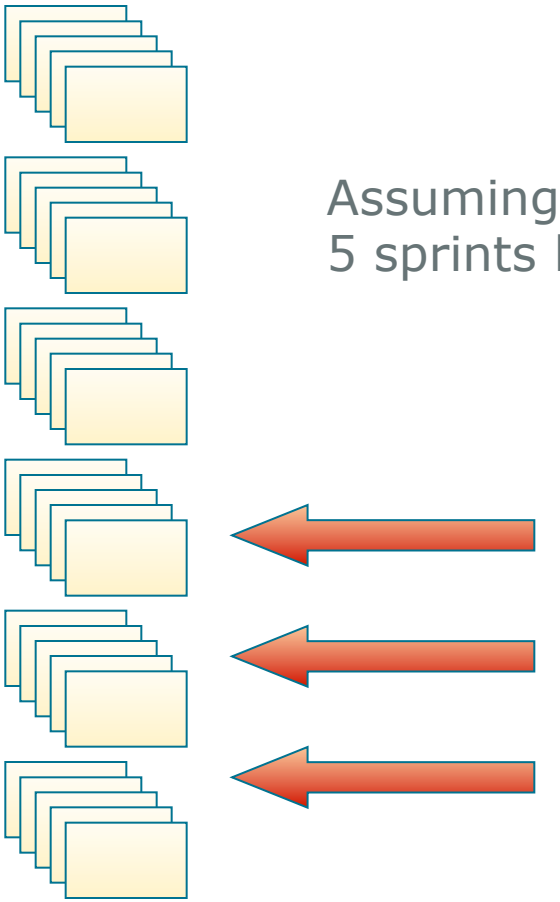


Latest observation = 36

Avg. of last 8 = 33

Worst 3 of last 8 = 28

Extrapolate from Velocity



Assuming we have
5 sprints left:

At our lowest velocity, we get here (5 * 28)

At our average velocity, we get here (5 * 33)

At our latest velocity, we get here (5 * 36)

Product Backlog

- The Product Backlog is the heart of Scrum planning
  - Requirements
  - Release planning
  - Progress tracking
- What to put in the product backlog?
  - Everything related to the developed solution
    - Stories (requirements), errors, spikes, ...
  - Anything that needs a business decision
    - E.g. significant development decisions with trade-offs
- What not to put in there?
  - Tasks
  - Most other development activities (setting up environment, etc.)
  - General quality requirements (put them in the Definition of Done)

Product Backlog ≠ Requirement Spec

- Two important differences to traditional requirement specification
    1. Product Backlog is not expected to contain all requirements at the start of the implementation
    2. All the stories in the PB are not expected to be delivered by the team
        - It may be the goal in a specific project, but there is no general expectation that it is so

- The level of detail is different
    - Requirement specifications try to be comprehensive and accurate to the detail
    - Product backlog is a overview and placeholder for details passed in written or oral form

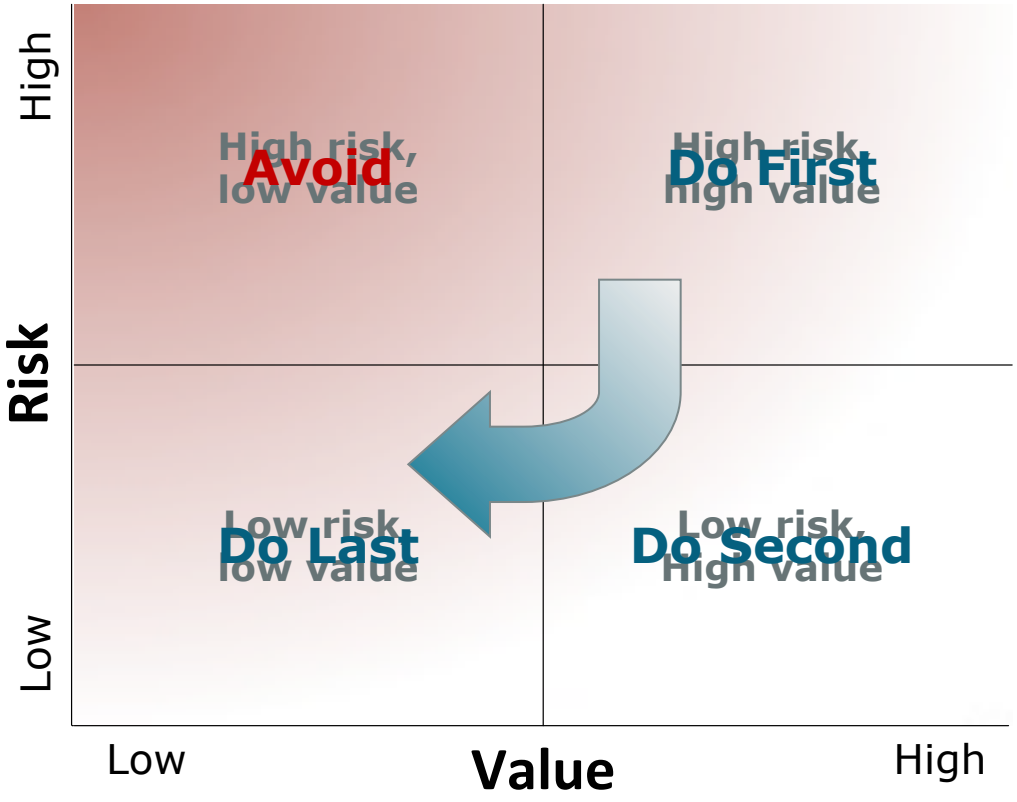
Managing the Product Backlog

- The product backlog is owned by Product Owner
  - Decides on content, priority and implementation order
  - ScrumMaster often supports by managing the normal maintenance
- Anyone can propose items to product backlog
  - The product backlog can contain any number of items
  - Only features placed into sprint by PO are actually made
- Every item should have a size estimate
  - Can be excepted with items that will be done far into the future, if at all
  - The team is responsible for providing the size estimates

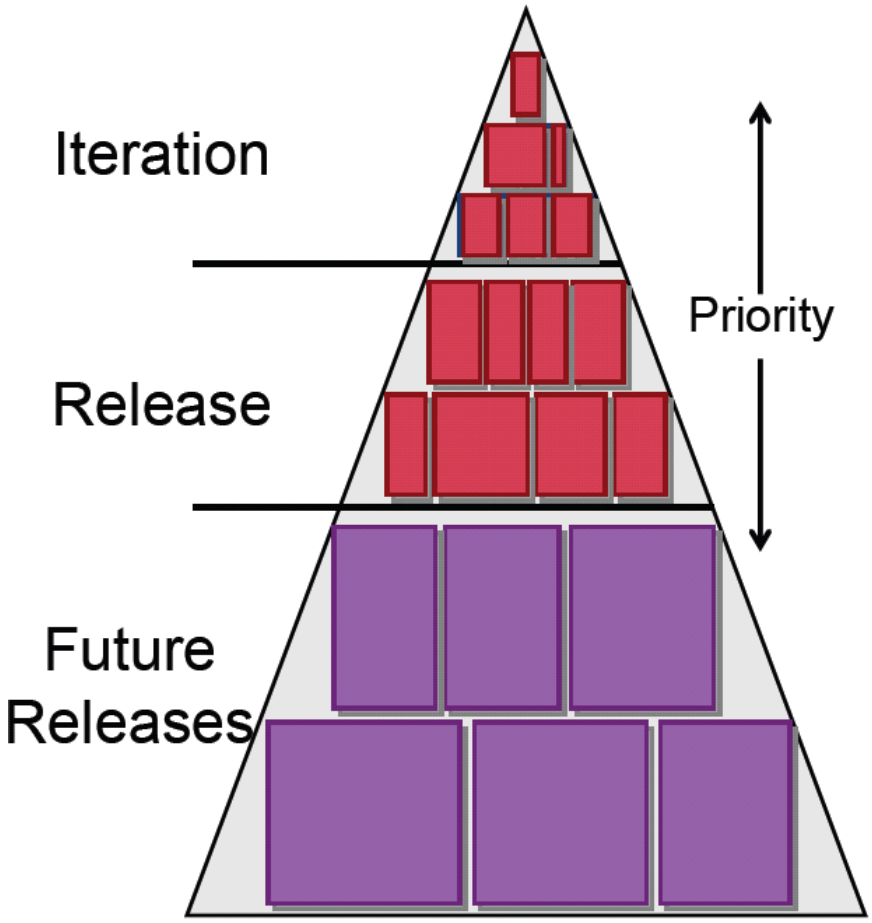
Managing Scope

- Scope management is dependent on business objectives
  - Fixed budget → Make a cut-off point
  - Feature-driven → Adjust budget and schedule estimate to include desired stories
  - Value-driven → Develop features as long as they provide enough value-added
  - Fixed schedule → Prioritize features and adjust team size to fit desired most important features into schedule
  - Combinations of above
- Scope-creep is very possible, if not guarded against
- Having everything at top priority is not scope management
  - Results in arbitrary results as no real prioritization is made
- Most large features have different priorities for sub-features
  - Split large stories to smaller ones for meaningful prioritization

Managing Risk



Granularity Increases

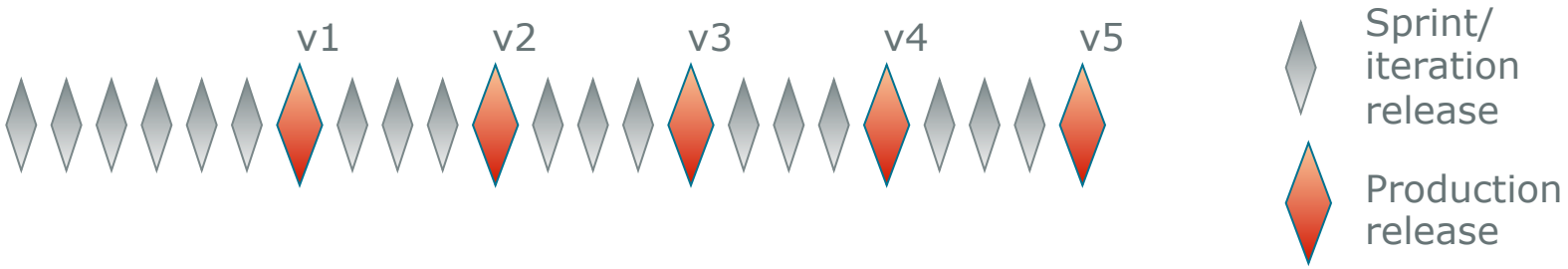


When planning releases, low granularity simplifies planning.

When planning sprints, high granularity is a necessity.

Breaking stories smaller is a continuous activity through the project.

Planning Release Schedule



- Considerations
  - Given your organization and users, how often can you do production releases?
  - How often can you expect to get feedback?
  - How big is the overhead in doing a production release?
  - Are there external parties that need releases at particular times?

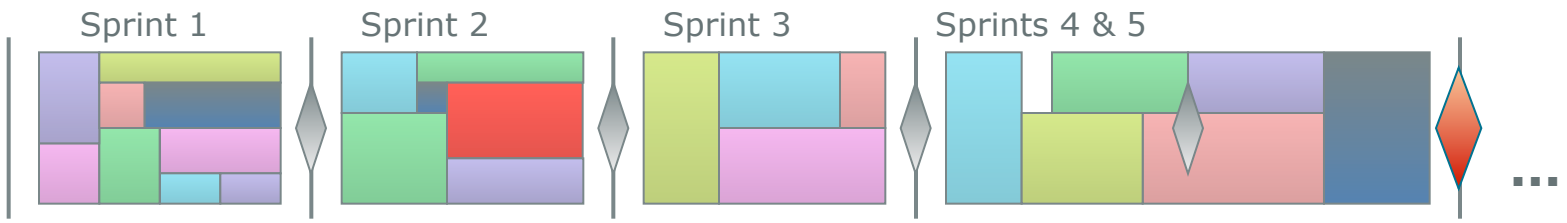
- Typically, the more often you can do production releases, the more feedback you will get

Planning Release Content

- Focus on key features (epics)
  - These act as vision elements for the release
- Calculate overall expected capacity
- ”Paint with a large brush”
  - It probably isn’t useful to allocate things to individual sprints
- Identify key milestones within the release
  - External dependencies (either from or to outside)
  - Internal dependencies between teams
  - These milestones are usually quite fixed
    - Plan buffers and slack to ensure their requirements are met

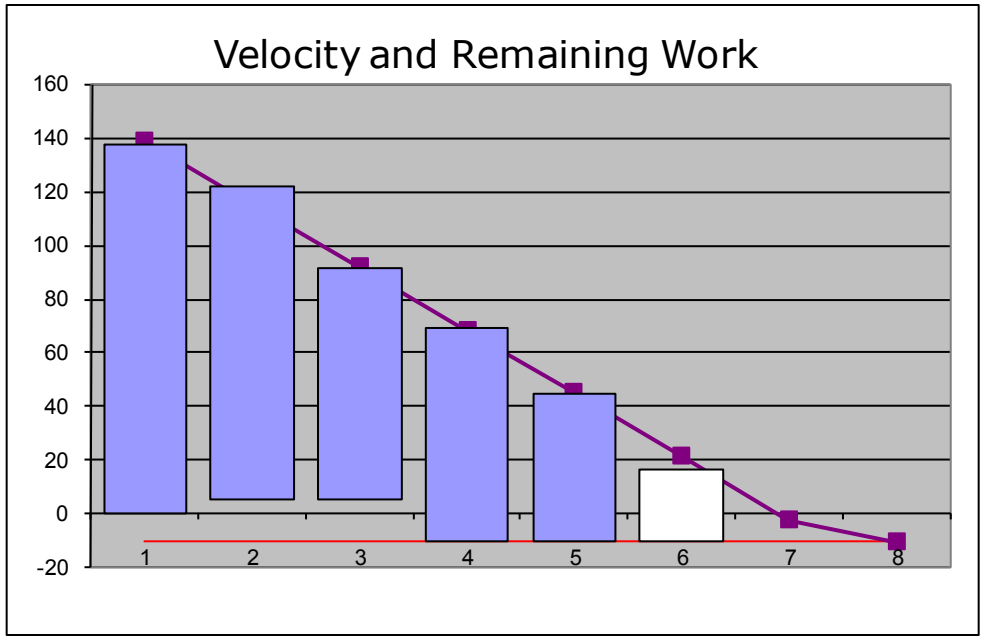
Planning the Near Future

- Usually it is necessary to plan a few sprints ahead at a higher level of detail
  - Usually 2-3 sprints ahead
- These plans should already be quite detailed
  - Right sized stories, reliable estimates
- Still not a commitment to develop specific stories
  - There is still room for change



Product Burndown Chart

- Project progress is visualized in the product burndown chart
- The same chart can provide visual planning feedback



More detail on burndown charts in the tracking section.

Planning Sprints

- Each sprint is planned at the beginning of the sprint
  - Sprint goal
  - Implemented stories
  - Sprint backlog (list of development tasks)
- Acceptance criterias are agreed
  - Story scope
  - Acceptance tests

Sprint Goal

- A short statement of what the work will be focused on during the sprint

Database
Make the application run on SQLserver in addition to Oracle

Life sciences
Support features necessary for population genetics studies.

Financial Services
Support more technical indicators than company ABC with real-time, streaming data

Content by Martine Devos

Planning Rules

- Product Owner prioritizes the features
- Product Owner sets sprint goal
- The team provides estimates for
  - Story sizes
  - Velocity
- PO cannot assign more work than what the team estimates it can commit to
  - Up to velocity normally
- The customer commits to agreed scope for the duration of the sprint
- The team will do all it can to deliver the agreed stories
- The team will maintain the agreed (high) quality

Bugs – How to Plan?

- Agile teams have the goal to fix bugs IN the iteration they are discovered
- Estimates should include that work
- Automated tests help us getting better at that
- Defects found later need to be treated the same as user stories – put on backlog, prioritized, acceptance tested…
- Merge “Bugzilla”… with new stories and prioritize like all other work

Maintenance?

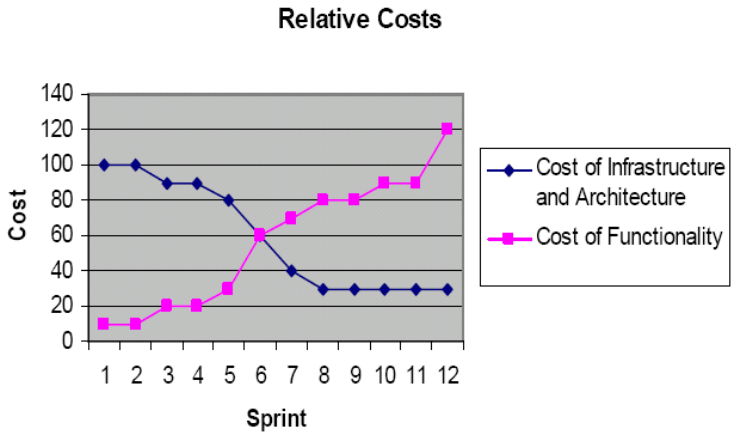
- Reduce team capacity by certain amount (e.g. 20%) to account for time needed
  - Base estimate on history, if possible
- If you can, arrange a dedicated team or persons
  - Less interruptions, better performance
- If not, the team can track and manage the support activities through the daily scrums
  - Or let them plan a better way

Acceptance Tests

- In optimal case, the customer can deliver acceptance tests for the team
  - Detailed requirements in executable format
  - E.g. Fit/Fitnesse
- Many projects don't have such optimal situation
  - Written acceptance tests or test summaries
- There has to be an agreement how the acceptance testing is arranged in the project
- The test criteria are used to evaluate sprint acceptance in the sprint review meeting
  - Not the only criteria, though

Developing Architecture

- Architecture and infrastructure are high priority non-functional requirements
- Must be completed to prove that functional requirements can be implemented satisfactorily
- Every Sprint still must deliver at least some piece of business functionality
  - To prove that architecture or infrastructure works
  - To prove to customer that work they care about is taking place



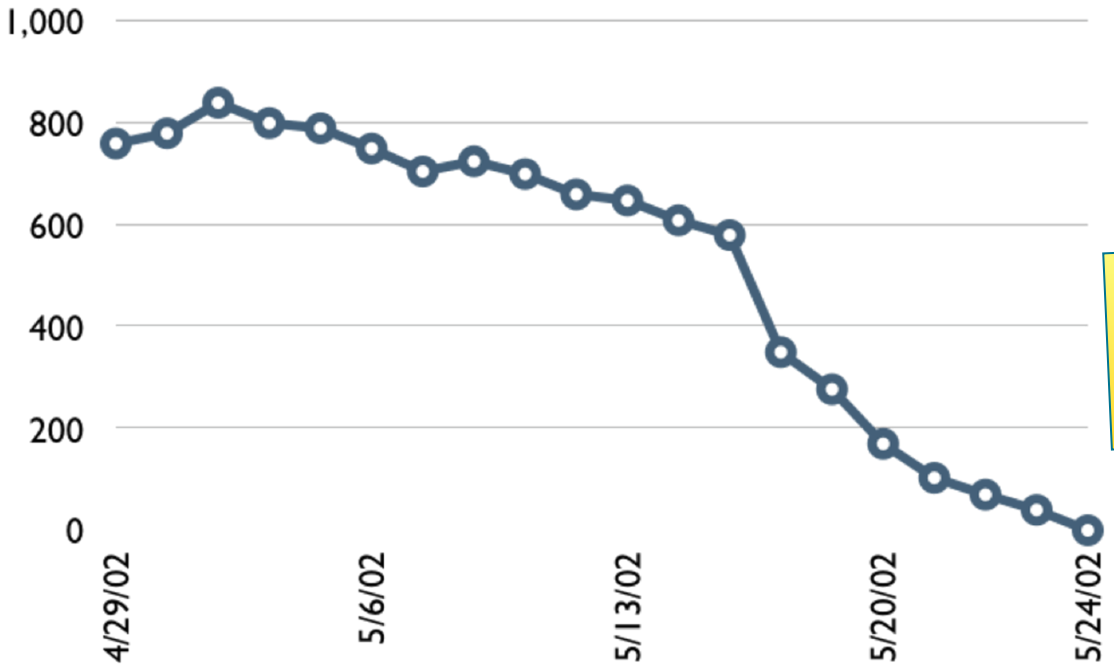
Sprint Backlog

- Team plans the **tasks** needed to develop the agreed stories
- Estimation unit: (ideal) hours
- Ideal task size: 2 – 10 lh
- The team owns the sprint backlog and is free to modify it during the sprint
  - Add or remove tasks
  - Update task definitions
  - Update estimates
- The goal is to complete the stories committed to in the sprint, not to complete tasks

Maintaining the Sprint Backlog

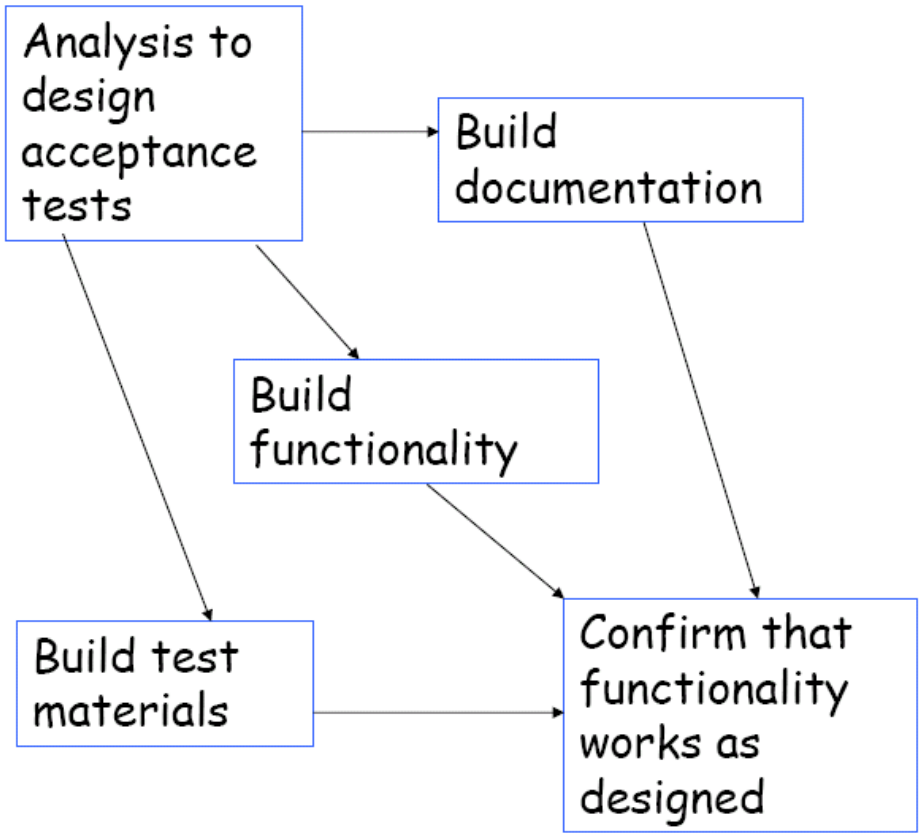
- The sprint backlog should be updated daily regarding
  - Tasks completed
  - Remaining hours updated
  - New identified tasks added, unnecessary ones removed
- A logical time for update is before daily scrum
  - Up to date information provided for everyone
- ScrumMaster updates the sprint burndown chart
  - Preferably daily
  - NOTE: The information is not updated for the SM; the ScrumMaster merely records down to help the team
- Remember, sprint backlog is owned by the Team – they can modify it at will

Sprint Burndown Chart



More detail on burndown charts in the tracking section.

Realizing Product Backlog



Cross-functional, collaborative team effort, coordinated by continuous communication

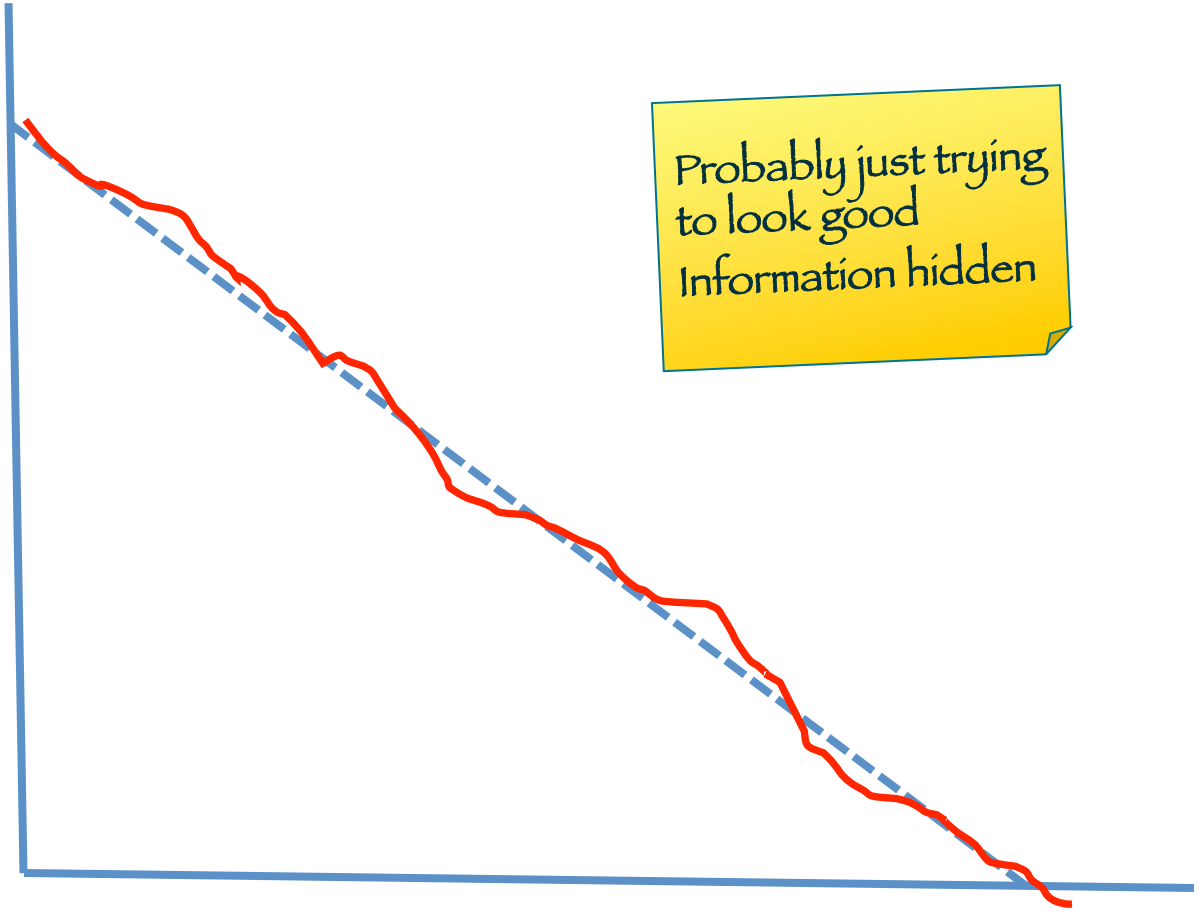
TRACKING

Team Walls

- A physical team wall, with tasks tracking and ”big visible charts”, is considered the most effective method for tracking sprint progress
  - Some colocated teams prefer technical tools, which is fine
  - Most distributed teams use some kind of computer-based tool
- Team’s tool
  - Not ScrumMaster’s, even if he/she helps the team a lot in maintaining the information on the wall

More Sample Team Walls

Good Burndown?



It's Team's Responsibility

- As a ScrumMaster, it is not your responsibility to worry about the burndown
  - If you get nervous, you might ask how the team feels about the burndown. If they're cool with it, fine. If they fail to deliver, they will learn.
- Provide the information so that the team can make up-to-date interpretations of their status
  - If the team seems to ignore the burndown and it is a problem, make it bigger ☺
  - Seek to provide information that will highlight the areas the team is having problems with
    - Don't point the finger at it, just make it visible
- Help the team carry its responsibility

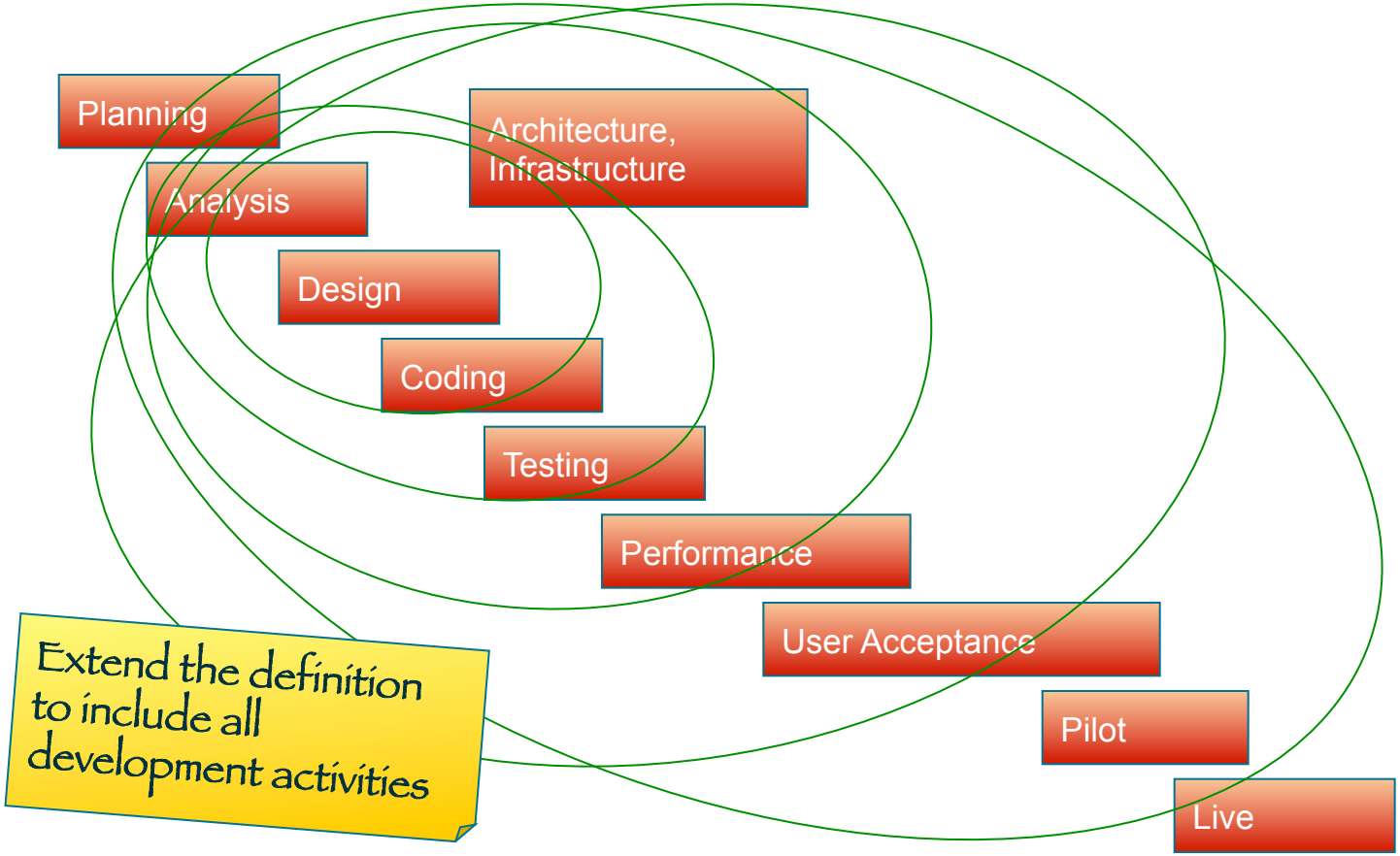
Exercise: Definition of Done

- As a group (at each table), discuss
  - What does “done” mean in your current project?’
  - What issues do you see with that definition of done?
  - How would you address them?
  - What engineering problems do you see with that approach?
  - How would you rectify them?

RTF

- Running Tested Features is the measure by which project progress is measured
  - Not: % complete
- RTF complies with the Definition of Done
- As the definition improves over the duration of the project, so does the requirements for past Done stories
  - This may mean that the team has to add stories to the product backlog to improve the quality or integration of past features

The Scope of ”Done” Changes



Visibility at Daily Level

- Scrum provides very high visibility into the daily progress as it is those people who know where they are being responsible for caring where they are
- As a ScrumMaster, remember that it is not your responsibility to know where the team is. If they do, you do, too, but the important bit is that they do.
    - You measure progress as working features

Visibility at Project Level

- This level is the domain at which management and PO should look at
  - At what rate real features are being added to the project
  - The quality of the software stays high
- It is relatively easy to estimate progress
  - Features are demonstrated in the sprint reviews
  - High test coverage, 100% pass rate, and very low number of released errors prove quality
- No need to micromanage

What Kind of Conclusions Can PO Draw?

- Velocity is increasing
  - The team is picking up speed and things should be fine
  - Is quality staying high?
- Velocity is stable
  - Is the team spending enough time on continuous improvement?
  - Are there unresolved obstacles?
- Velocity is decreasing
  - Is the quality of the software poor, resulting in reducing speed?
  - Are the problems/risks the team is not bringing forward?
  - Are obstacles to productivity not removed?
- Estimates differ from realized results regularly
  - How and based on what information does the team make the estimates?
  - Is there something wrong in the team environment?
- Team is continuously updating estimates
  - This can be a good thing
  - If the direction is continuously the same, maybe there are some systematic errors that need to be fixed

Avoid suspicion or blame – open atmosphere builds trust – trust builds commitment

Re-estimating Partially Completed Stories?

- Favour all or nothing
  - Done, full points. Not done, *"nuls points"*
  - Velocity is lower in first iteration, will then get a bit higher
  - OK, if most interested in average velocity over time
- Sometimes unfinished part does not get done next iteration
  - Take partial credit and re-estimate the rest
    - Combined size does not need to be equal to original
  - Beware of tiny stories – combine when needed

When to Re-estimate?

- Whenever the team feels one or more stories are mis-estimated relative to other stories,

  → re-estimate as few stories as possible to bring the relative estimates back in line

- As a learning experience for estimating future stories

Quality Indicators

- Ability to test new stories within the iteration
- Defect
  - Numbers far smaller
  - Address all hands on deck
- Progress on test automation
- Unit test coverage

Thank you

For more info:

petri.heiramo@gmail.com