Module: Scrum Basics

"In the long history of humankind (and animal kind, too), those who learned to collaborate and improvise most effectively have prevailed." — Charles Darwin

Scrum Roles – Brief Introduction

Product Owner

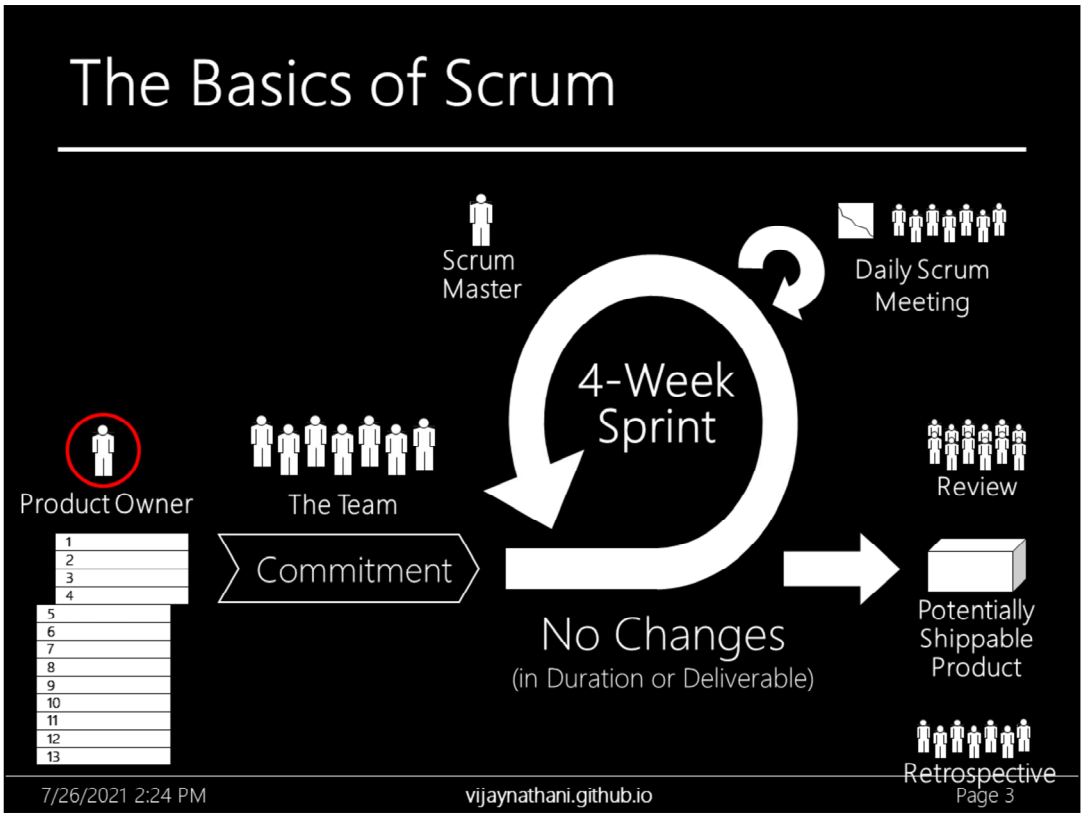
Determines what should be produced, to maximize ROI for the business

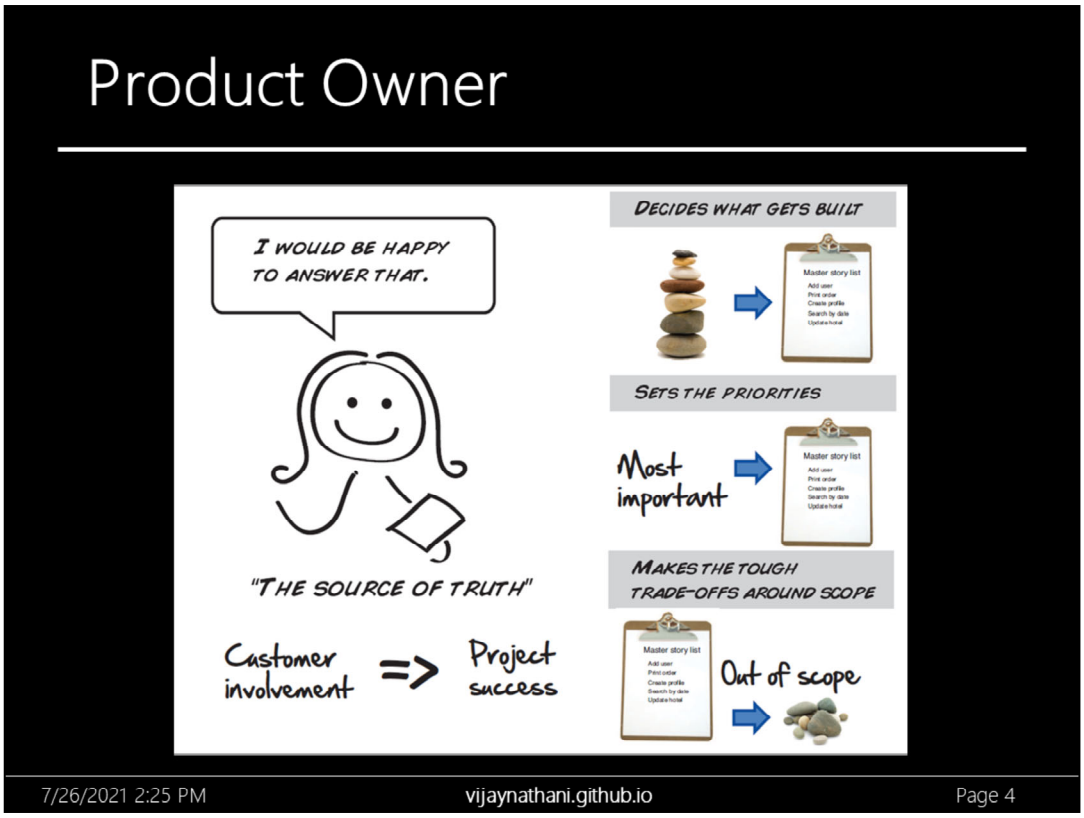
The Team

Cross-functional and self-organizing team of 7 people +/- 2

ScrumMaster

Protects and serves the Team, and supports their practice of Scrum


The Basics of Scrum
Scrum Master
Daily Scrum Meeting
4-Week Sprint
Product Owner
The Team
Review
1
2
3
4
5
6
7
8
9
10
11
12
13
Commitment
No Changes
(in Duration or Deliverable)
Potentially Shippable Product
Retrospective
7/26/2021 2:24 PM
vijaynathani.github.io
Page 3



Owns definition of success

Charged with maximizing ROI and managing project risk

Responsible for taking all inputs into what the team should produce, and turning it into a prioritized list. Inputs include:

- Customer
- Team
- Executives
- Competitors

Other Stakeholders

Determines release plan and communicates it to upper management and the customer

==========

One study found that all the bugs that developers are responsible for, in 37% of the cases, they just did not know that they were supposed to do that check. Systems are so complex today, that one person rarely knows the entire system.

So closely working with the PO is important.

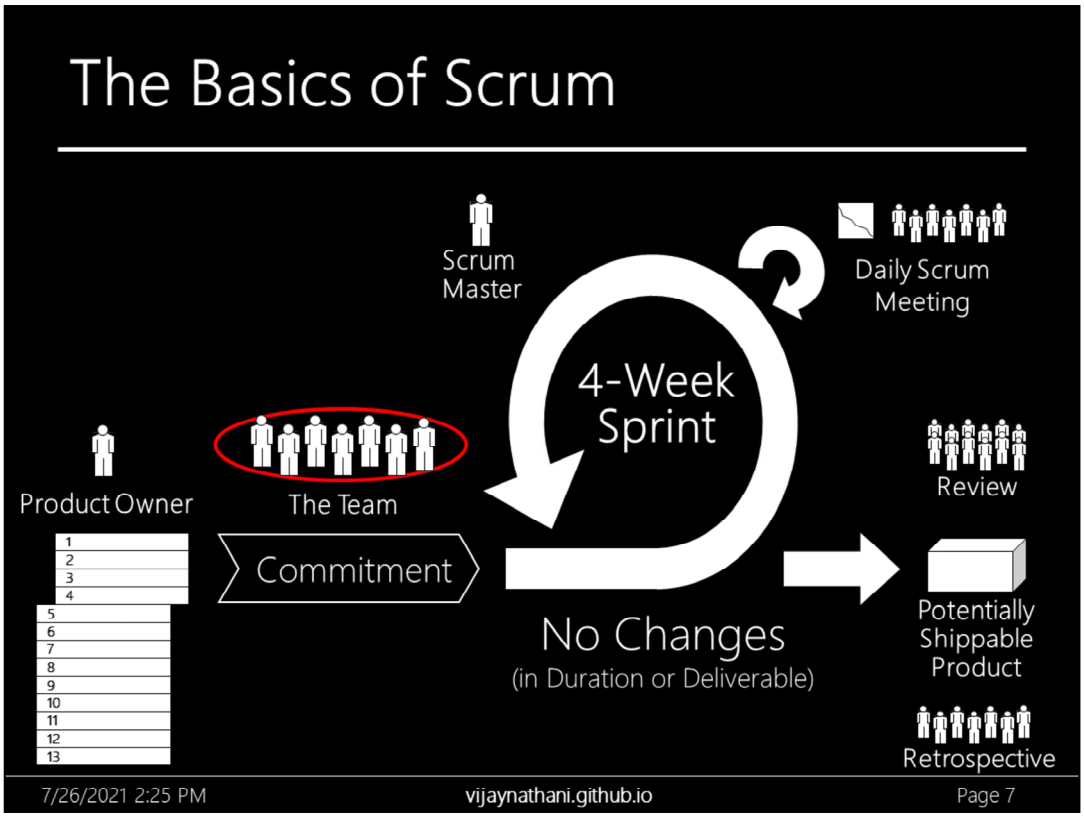
If possible, the PO should sit in the same areas as the team.

Responsible for all Features

There is one Product Owner. Only one.

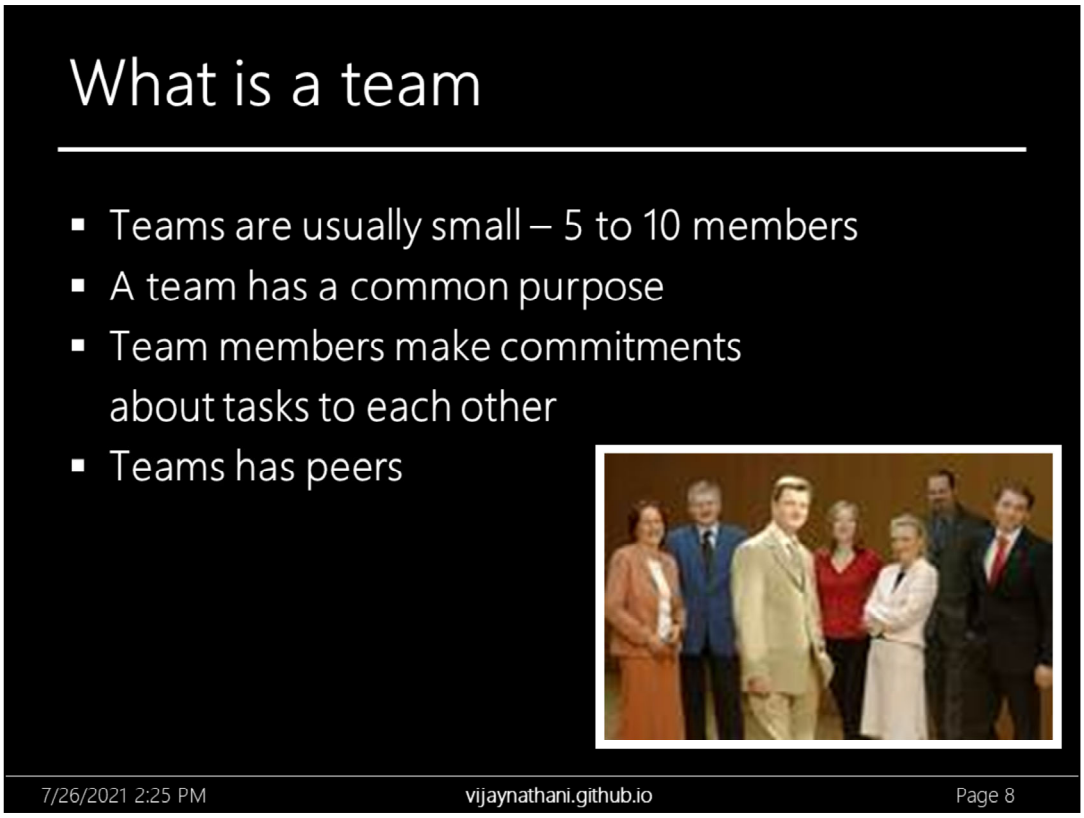
Product Owner

- Represents all stakeholders
- Decides where the team should go.
  - Not how they should get there
  - Not their speed.
- Defines scope. Prioritizes.
- Owns the product backlog
- Not a line manager.
- Does not estimate stories



The power of We is stronger than the power of Me.

A team is a small number of people with complementary skills who are committed to a common purpose, performance goals, and approach for which they hold themselves mutually accountable.



A Manager are not a part of developer team

The structure of a team is network and not hierarchy

Teams in Agile Ecosystems

Some people think that "Agile" means

fewer processes,

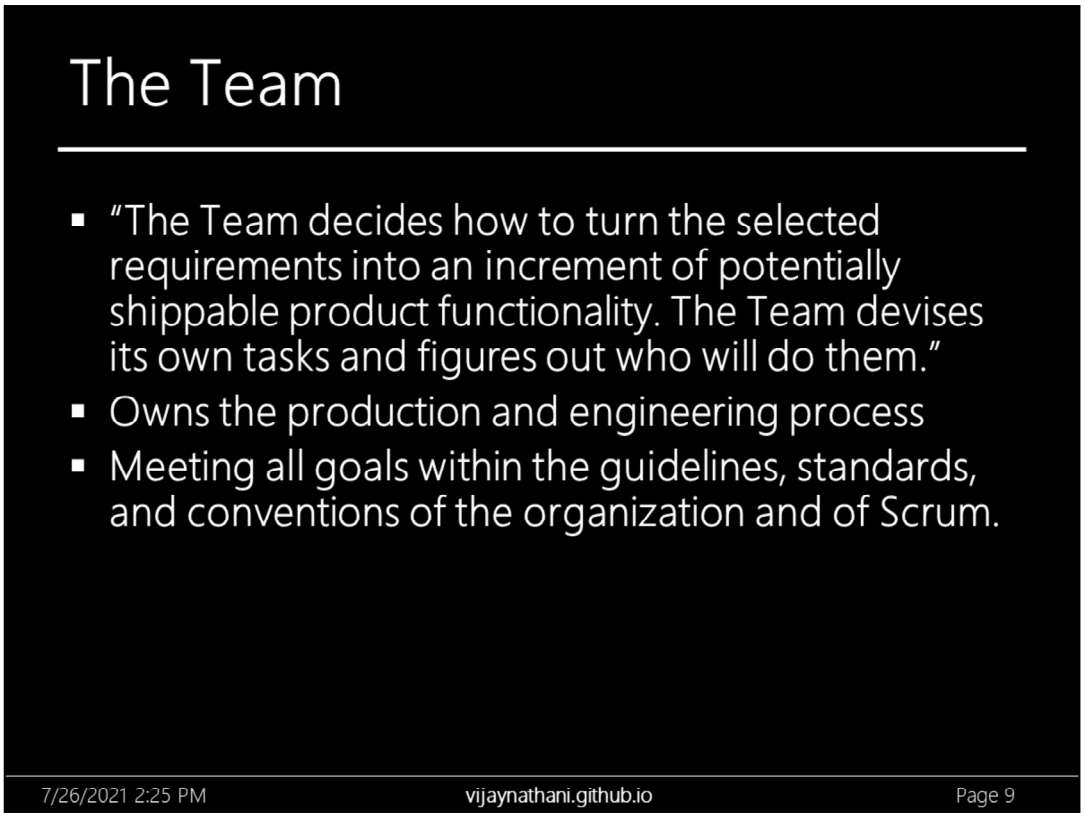
less ceremony, and

briefer documents,

but it has a much broader perspective.

Although fewer processes and less formality might lower development costs, they are not enough to produce agility.

Focusing on people and their interactions and giving individuals the power to make quick decisions and to self-adapt their own processes are key to Agility.



Dee Huck said,

"Simple, clear purpose and principles give rise to complex, intelligent behavior"

"Complex rules and regulations give rise to simple Stupid behavior"

Self Organizing Teams are Superior to Command n' Control Teams

* People in a self-organized team are able to make decisions themselves and accordingly adapt to changing situations. Command and control grunts have to wait for the boss to tell them what to do. That introduces latency in the development process as the team waits for the leader to shake free to deal with a decision. It can also sap a developer of any energy to contribute to the design and approach if they know they don't have any say in the matter. It's a negative incentive to increase their intellectual participation in the project, and that can only hurt the team.

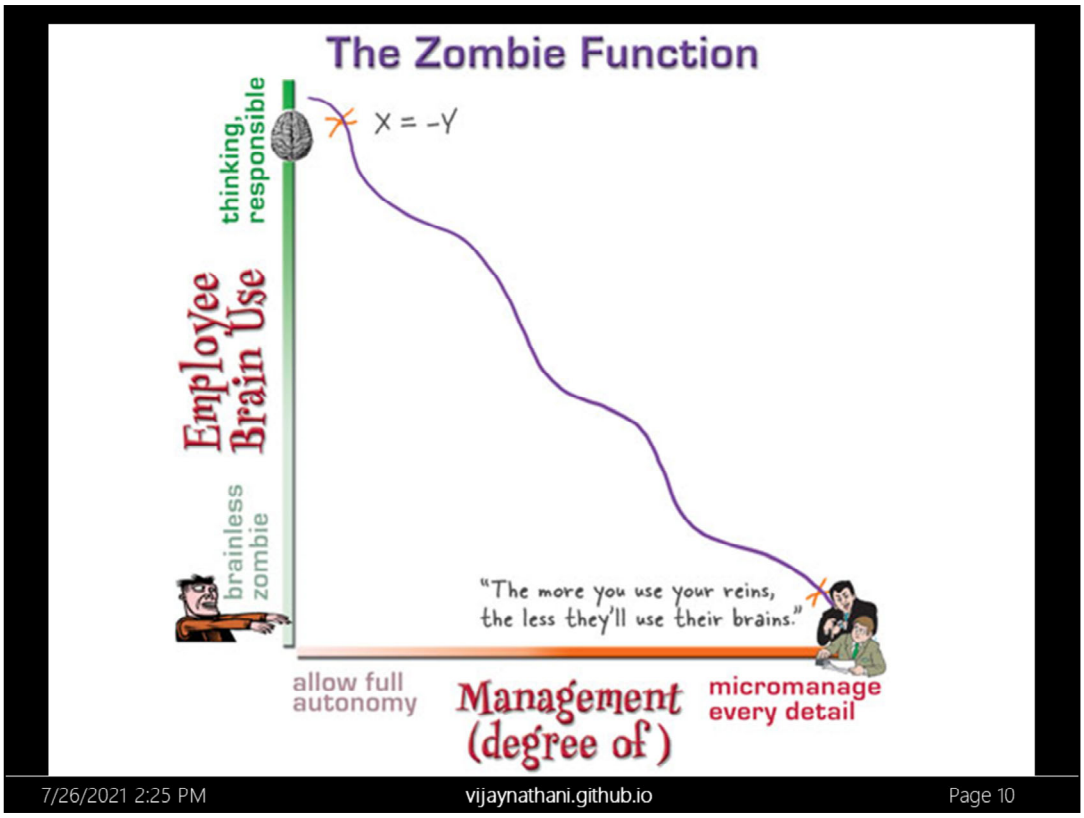
* Self organized teams do a much better job of utilizing the talents of the team because more minds are involved in any activity. By investing one person with all decision making authority, you effectively shut down the other team member's contributions to the design and planning. They become drones.

* Self organized teams have much more communication between team members. A command and control team chokes communication by running too much through one person.

* Command and control organizations don't provide as many chances for personal development. The best way to learn is to have actual responsibility and opportunities to do new things. If all you do is what you're told without question, you don't get to learn how to make decisions.

* A self organized team is collectively aware of the upcoming work and much better able to bootstrap themselves with new work when they complete their existing task. I've learned that there's a lag between telling a developer to write a feature and the beginning of coding because it takes some time to understand what's required. You can cut down that lag by doing planning and design together as a team.

* Self organized teams spread knowledge around much better and make decisions together. That makes each team member more effective because they have much more background on the "why" of the coding assignments. A command and control team member often lacks an understanding of why a decision was made because they weren't involved with that decision. That hampers their ability to follow a design or approach.



If people are not empowered to make improvements, they stop thinking about making improvements.

The team is both empowered and accountable to deliver the goods. The team does their job when they self-organize, self-manage and self-achieve the objectives of the Sprint.

> For many organizations, this turns things upside down.
>
> The hierarchical-technical-management-directive approach is essentially eliminated with Scrum.
>
> The Product Owner now sets the objectives and priorities, the team figures out how to achieve them, and no one need tell them how to do that along the way.

Software Development is a social Activity

If you want to be incrementally better: Be competitive. If you want to be exponentially better: Be cooperative.

> A strong player is not necessarily an ace programmer. A strong player may be an average programmer, but someone who works well with others. Working well with others, communicating and interacting, is more important that raw programming talent. A team of average programmers who communicate well are more likely to succeed than a group of

superstars who fail to interact as a team.

It isn't what you don't know that gets you in trouble. It's what you know that isn't so. The biggest problem, in what people "just know" about software development, is that they are focused on individual action. What actually matters is not how any given person behaves, but how the individuals behave as part of a team.

Games are either co-operative / competitive

In cooperative games, the people work either to win together or to continue the game as long as they consider it worth playing. Storytelling, playing jazz, etc are non-goal-seeking cooperative games. In these latter games, the players do not seek to end the game by reaching a goal as fast as possible. They come to an end only when enough people get tired of playing and step out.

Cooperative and goal-seeking. A team of rock climbers work together to reach the top. They will evaluate the climb based on how well they climbed together and how much they enjoyed themselves, but the first measure of success is whether they reached the top. Reaching the endpoint is a primary goal, and the game is over when they reach the top.

(If you are a rock climber, you might well interrupt me here. For many rock climbers, the moment of reaching the end of the climb is a sad one, for it signals the end of the game. That is true of cooperative games in general. The game comes to an end when the endpoint is reached, but if the players have been enjoying themselves, they may not want to stop. Similarly, sometimes software developers do not want to finish their design, because then the fun part of their work will be over.)

Load bearing. The climbers must actually support their weight on their hands and feet. This is a particularly valuable point of comparison between the two: Software must run and produce reasonable responses. While multiple solutions are possible, not just any solution will do.

Team. Climbing is usually done in teams. There are solo climbers, but under normal circumstances, climbers form a team for the purpose of a climb.

Individuals with talent. Some people just naturally climb better than others do. Some people will never handle certain climbs.

Skill-sensitive. The rock climber must have a certain proficiency. The novice can approach only simple climbs. With practice, the climber can attack more and more difficult climbs.

Training. Rock climbers are continually training on techniques to use.

Tools. Tools are a requirement for serious rock climbing: chalk, chucks, harness, rope, carabineer, and so on. It is important to be able to reach for the right tool at the right moment. It is possible to climb very small distances with no tools. The longer the climb, however, the more critical the tool selection is.

Resource-limited. A climb usually needs to be completed by nightfall or before the weather changes. Climbers plan their climbs to fit their time and energy budget.

Plan. Whether bouldering, doing a single-rope climb, or doing a multiple-day climb, the climbers always make a plan. The longer the climb, the more extensive the plan must be, even though the team knows that the plan will be insufficient and even wrong in places.

Improvised. Unforeseen, unforeseeable, and purely chance obstacles are certain to show up on even the most meticulously planned climbing expeditions unless the climb is short and the people have already done it several times before.

Therefore, the climbers must be prepared to change their plans to improvise at a moment's notice.

Fun. Climbers climb because it is fun. Climbers experience a sense of flow while climbing, and this total occupation is part of what makes it fun. Similarly, programmers typically enjoy their work, and part of that enjoyment is getting into the flow of designing or programming. Flow in the case of rock climbing is both physical and mental. Flow in the case of programming is purely mental.

Challenging. Climbers climb because there is a challenge: Can they really make it to the top? Programmers often crave

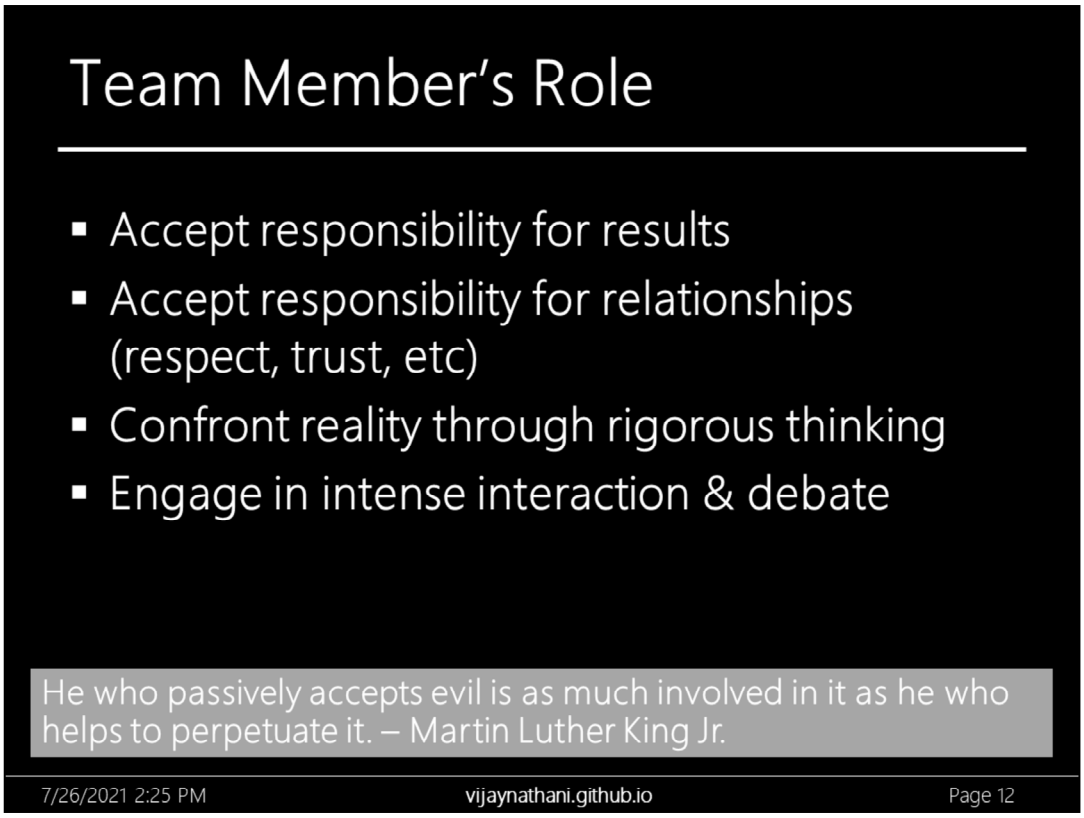
this challenge, too. If programmers do not find their assignment challenging, they may quit or start embellishing the system with design elements they find challenging (rather like some of the poets mentioned in the epic poetry project).

Dangerous. Probably the one aspect of rock climbing that does not transfer to software development is danger. If you take a bad fall, you can die. Rock climbers are fond of saying that climbing with proper care is less dangerous than driving a car. However, I have not heard programmers express the need to compare the danger of programming with the danger of driving a car.

Software development has been compared with many other things, including math, science, engineering, theater, bridge building, and law. Although one can gain insight from looking at any of those activities, the rock-climbing comparison is the most useful for the purpose of understanding the factors involved in the activity.

==========

Rock climbing has one limitation when used as a metaphor for software development: When people finish a rock climb, they walk back down to their cars and drive off. When people finish a software project, they stay around and enhance the system or build the neighboring system. They have to live with the results of what they did, and they have to integrate other people into the world they created.



Once a team begins to jell, the probability of success goes way up. It doesn't need

to be managed in a traditional way, and they certainly don't need to be motivated.

It has momentum.

Team members must have

trust in one another.

Mavericks may have to be

excluded from the team

A team gains consensus

- All members commit
- Decisions are made highly visible
- Agreement is through consensus

- Check on consensus to quickly find the disagreements
- Fist of five approach:
  - 5 = wild, unbridled support
  - 4 = this is a fine idea, wish I'd thought of it
  - 3 = I can live with and support it
  - 2 = I have reservations I'd like to think about
  - 1 = I am very opposed; we shouldn't

7/26/2021 2:25 PM vijaynathani.github.io Page 13

The IQ of the crowds is better than the IQ of a single person.

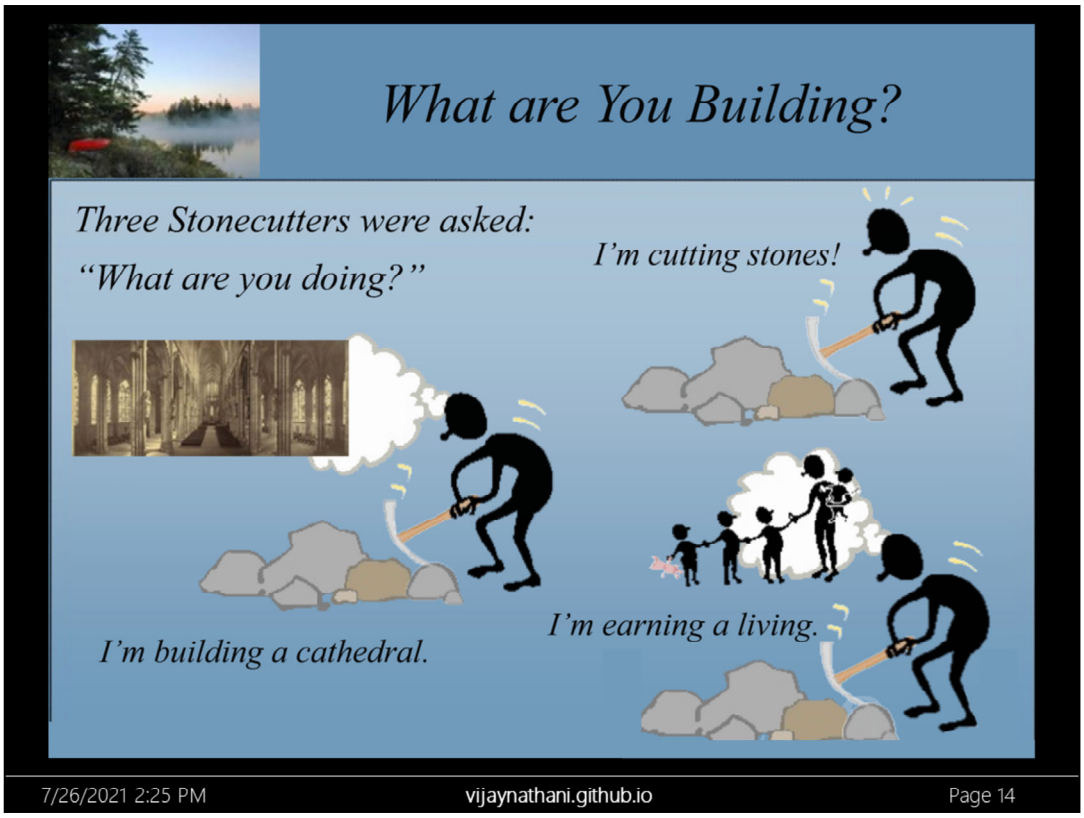
"Wisdom of Crowds" book:

•"Who wants to be millionaire?" show offered the participants 3 choices: 50-50, phone a friend, audience poll. The success of audience poll was 91 %. The success of phone a friend is 65%.

•A professor took a different jars of beans and asked the students to guess the number of beans. The average consistently was better. The 1 / 2 person who performed better than the average could not retain their correctness. So we score better by going with the average, rather than one single person.

For this approach to work, each person must think independently. A person must not take decision based on the opinion of others.

e.g. Two equal lines were drawn on the board. If a large number of (biased) people said that one is bigger than the other, the unbiased participant usually said the same. This defeats the purpose of independent thinking.

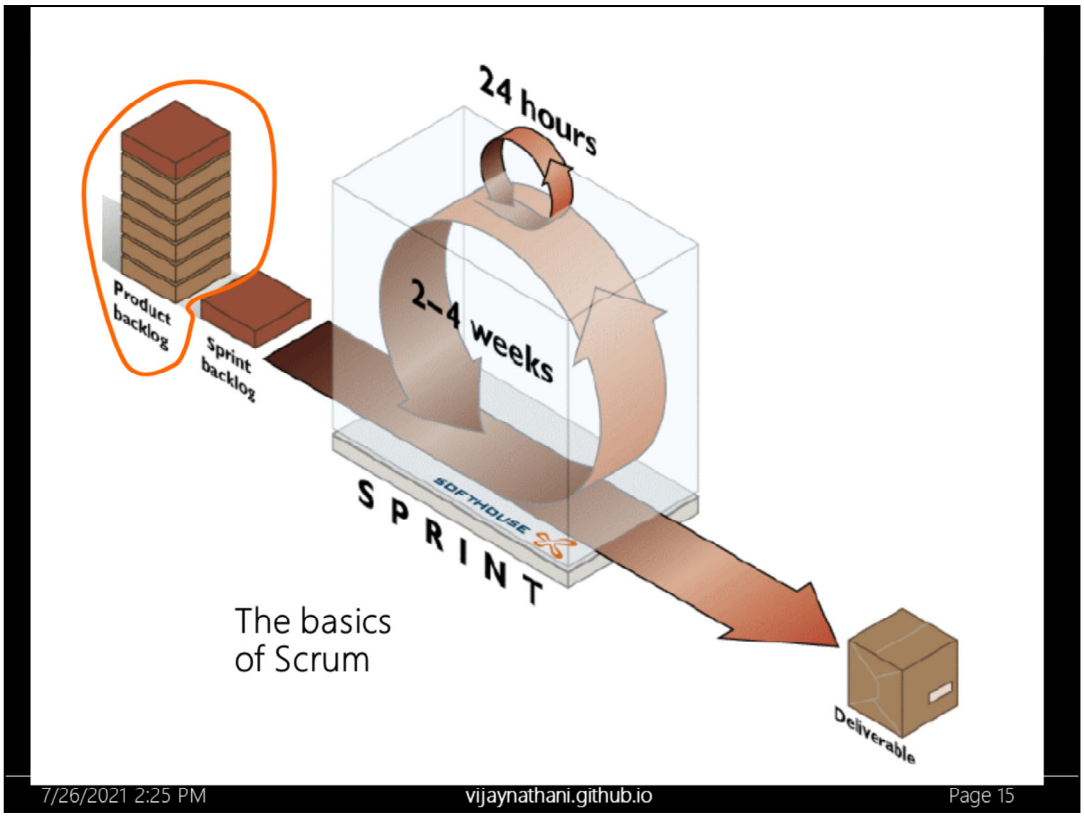


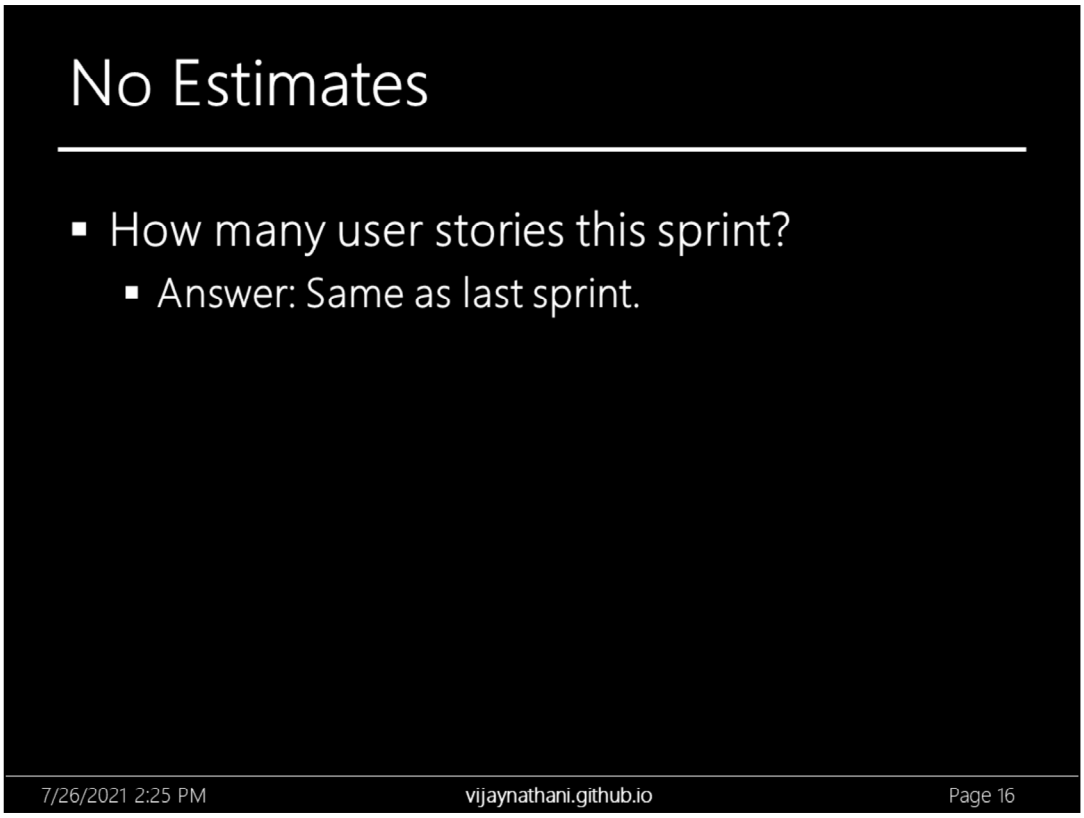
Respect People Principle of Lean means: Move responsibility and decision making to the lowest possible level

In which category do you belong?

Here is the litmus test?

When you are annoyed with your job, do you complain, ignore or fix it?


24 hours
2–4 weeks
Product backlog
Sprint backlog
SPRINT
The basics of Scrum
Deliverable
7/26/2021 2:25 PM
vijaynathani.github.io
Page 15



Estimation is a waste of time. Let the project continue as long as the output is profitable.

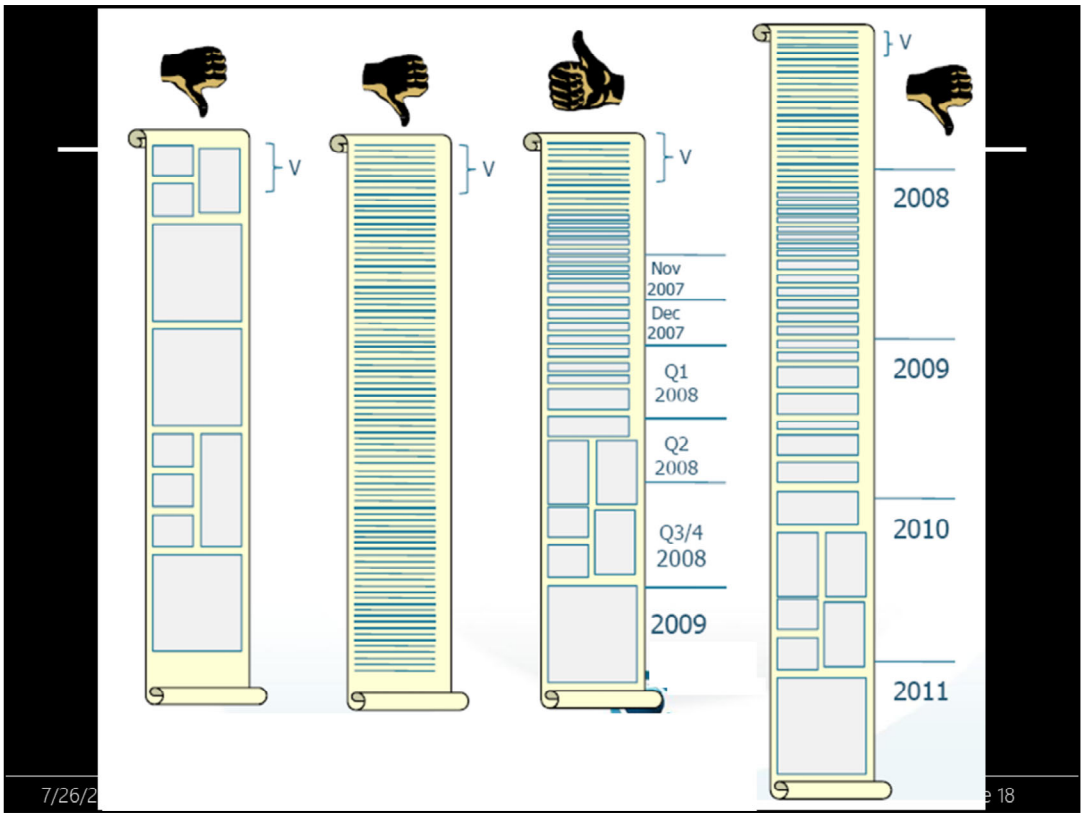
https://vimeo.com/user22258446/review/126266148/4543b8ba38

http://noestimatesbook.com/

Product Backlog

Description	*Rough Est. Size*
Enable all users to place book in shopping cart (mocks and additional details are located here)	20
Upgrade transaction processing module (must be able to support minimum 500 transactions per second)	10
Investigate solutions for speeding up credit card validation (see target performance metrics located here)	20
Upgrade all servers to Apache 2.2.3	40
Diagnose and fix the order processing script errors (bugzilla ID 14823)	20
Enable all users to create / save wishlist	40
Enable all users to add and delete items on their wishlist	10

Product Owner lists items in descending order of priority (highest priority item is listed first, next-highest is second, etc.)



Product Backlog

List of everything that could ever be of value to the business for the team to produce

Ranked in order of priority

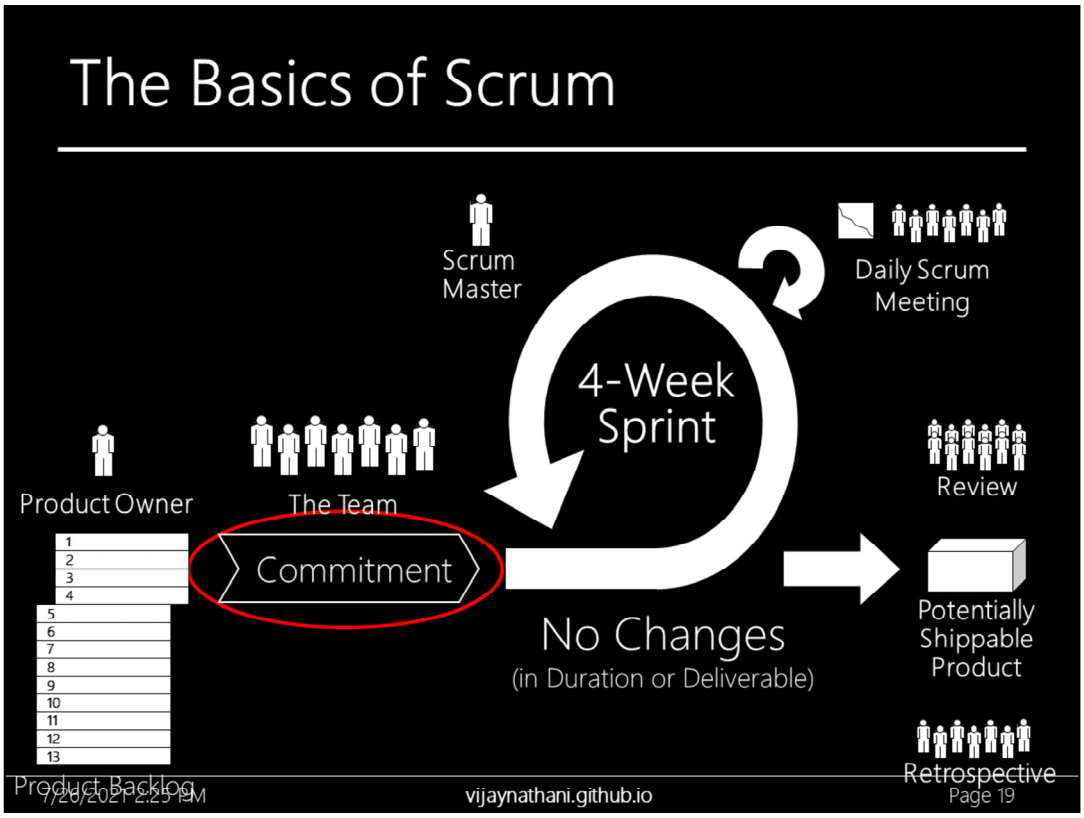
- Priority is a function of ROI and risk

Product Owner can make any changes they want before the start of a Sprint Planning Meeting

- Items added, changed, removed, reordered

The farther down the list, the bigger and less defined the items become

- ~2 Sprints worth are defined in detail


The Basics of Scrum
Scrum Master
Daily Scrum Meeting
4-Week Sprint
Product Owner
The Team
Review
1
2
3
4
5
6
7
8
9
10
11
12
13
Commitment
No Changes
(in Duration or Deliverable)
Potentially Shippable Product
Product Backlog
Retrospective
7/20/2021 2:25 PM
vijaynathani.github.io
Page 19

Sprint Planning Meeting

- Goal of Sprint Planning Meeting:
  For the team to make a good commitment around what it will deliver by the end of the Sprint
- What's a good commitment?
  - Clearly understood by all
  - Shared among the team
  - Achievable without sacrificing quality
  - Achievable without sacrificing sustainable pace
- Attended by Team, Product Owner, ScrumMaster, Stakeholders
- May require 1-2 hours for each week of Sprint duration

Team:

Validates commitment by breaking down into tasks with time estimates

Team decides who will do what, when; thinks through sequencing, dependencies, possible task trades, and so forth.

The Product Owner answers questions but does not direct the team's choices.

The outcome is the Sprint Backlog.

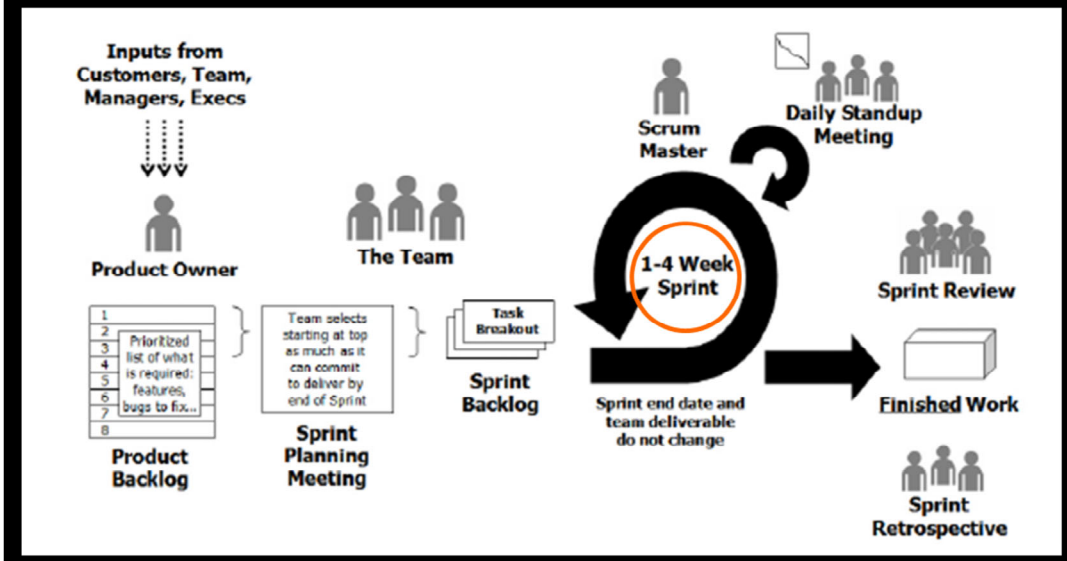
Product Backlog

Description	Rough Est. Size
Enable all users to place book in shopping cart (mocks and additional details are located here)	20
Upgrade transaction processing module (must be able to support minimum 500 transactions per second)	10
Investigate solutions for speeding up credit card validation (see target performance metrics located here)	20
Upgrade all servers to Apache 2.2.3	40
Diagnose and fix the order processing script errors (bugzilla ID 14823)	20
Enable all users to create / save wishlist	40
Enable all users to add and delete items on their wishlist	10

Task Breakdown

Backlog Item	Task	Owner	Initial Time Estimate
Enable all users to place book in shopping cart	Configure database and space IDs for Trac	Sanjay	4 hours
	Use test data to tune the learning and action model	Jing	2 hours
	Setup a cart server code to run as apache server	Philip	2 hours
	Implement pre-Login Handler	Tracy	6 hours
Upgrade transaction processing module (must be able to support 500 transactions /sec)	Merge DCP code and complete layer-level tests	Jing	8 hours
	Complete machine order for pRank	Jing	4 hours
	Change DCP and reader to use pRank http API	Tracy	2 hours

The basics of Scrum

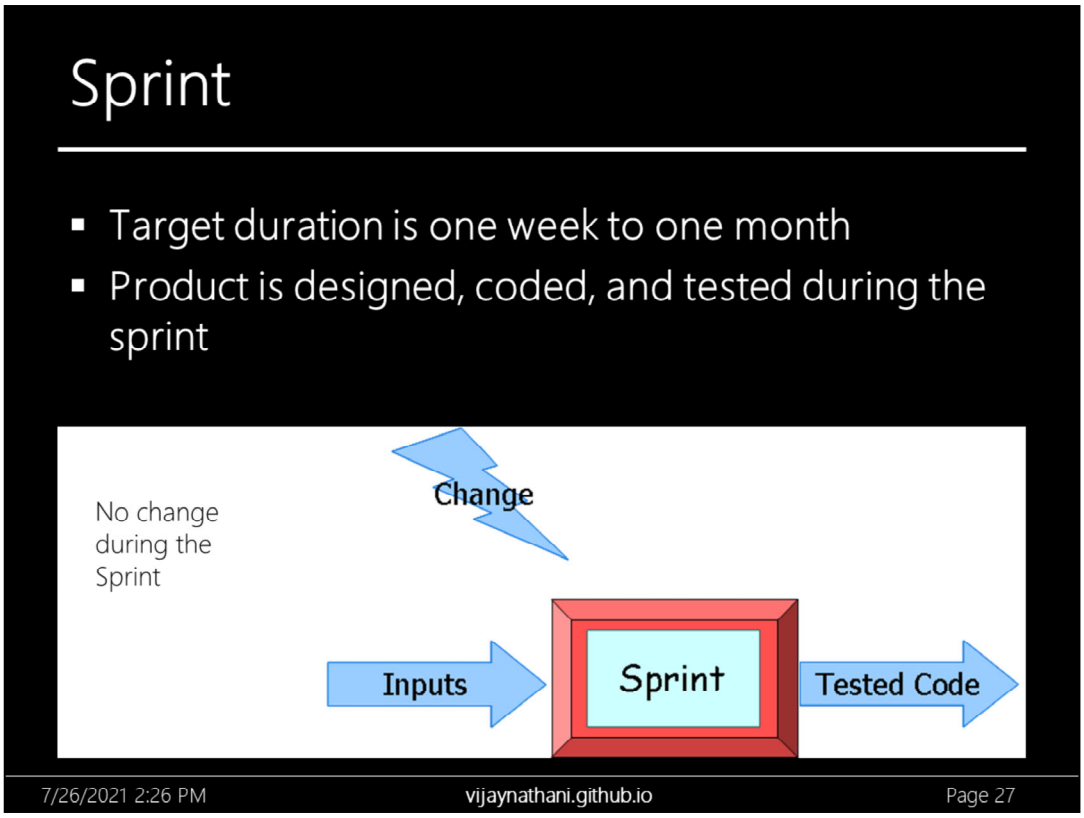


Sprint Commitment

- Team's commitment to the Product Owner: We promise that
  - We believe that we can complete all stories included in the Sprint backlog.
  - We will do everything in our power to reach the Sprint goal and will let you know immediately if we can no longer reach it.
  - We will release for production at the end of the sprint.
  - We will display our progress and status on a daily basis.
  - Every story, what we deliver, is "Done".

Common Misconceptions

- We promise we will achieve the goal.
- We promise that we will deliver all stories included in the sprint backlog.
- Estimates are given by the Product Owner / ScrumMaster



The Sprint

The Sprint is never extended

- Sprint ends on pre-determined date whether or not team has completed what it committed to

Sprint Length

- 4 weeks is standard in literature
- 2 weeks is also common
- 1 week is minimum, 4 weeks is maximum

Factors in deciding your Sprint length

- Length of the release
- Amount of uncertainty
- How long priorities can stay unchanged
- Overhead of Sprint Planning and Review
- Urgency / intensity lag

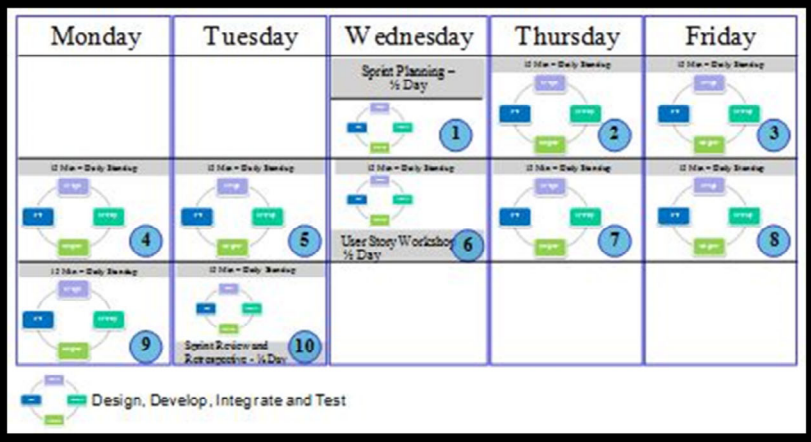
Sprint Cycle: 2-Week Sprint

Mon	Tues	Weds	Thurs	Fri
		1	2	3
6 Sprint Planning Meeting	7 1	8 2	9 3	10 4
13 5	14 6	15 7	16 8	17 Sprint Review & Retrospective
20	21	22	23	24
27	28	29	30	31

Sprint Cycle: 2-Week Sprint

Mon	Tues	Weds	Thurs	Fri
	5	6	1 1-Hr Pre-Meeting for Next Sprint	2 8	3 Sprint Review & Retrospective
6 Sprint Planning Meeting	7 1	8 2	9 3	10 4
13 5	14 6	15 7	16 8	17 Sprint Review & Retrospective
20 Sprint Planning Meeting	21 1	22 2	23 3	24 4
27 5	28 6	29 7	30 8	31 Sprint Review & Retrospective

Cadence


Monday
Tuesday
Wednesday
Thursday
Friday
Sprint Planning – ½ Day
1
2
3
4
5
User Story Workshop ½ Day
6
7
8
9
Sprint Review and Retrospective - ½ Day
10
Design, Develop, Integrate and Test