The Agile Fundamentals

Agile is a Philosophy guided by the Four Values and Twelve Principle.

We Are Starting With Four Values

Value No. 1 Individuals and Interactions Over Processes and Tools.

- It's Important for People Building a software to interact with Stackeholders(Customer).This not to say that processes and tools are not important they are very important. But being Agile means placing more emphasis on Interactions.
- Interactions between team members or multiple team members are important. Sometimes team members are hung up on how big this software has to be, so what to do, the best way to solve this problem is to have a face-to-face conversation with the team to better develop and have a better estimation of work.

Key takaway here is Interactions are very important between Stackeholders and teammembers.

Value No.2 Working Software Over Comprehensive Documentation.

- This does not mean we need to stop Documentation because without it software navigation becomes very hard. The key word here is Comprehensive.
- What it means to say that focusing more on main deliverable software rather then wasting a lot of time on documentation.
- This can be achieved by **Shared Undestanding**. Having more conversation between team and with the stakeholders and even the customers.
- This shared Undestanding can be achieve again by creating small increaments of software and putting it in the hands of the internal stakeholders or the customer so that they can gain feedback in the software they want.

Key Takeaway here is creating small increatment in software and rasing the shared understanding

of customer and team without too much focusing on documentation.

Value No.3 Customer Collaboration over Contract Negotiation.

- We want to collaborate with customer because we want to delight our customer and also want them to come back to us and rely on our expertise for developing software.
- When we Write Contract we need to keep customer success in mind. So Don't Have strong Contract upfront, collaborate instead to create a win-win situation.
- In Customer Collaration it is actually not fair of development team to ask for all the reqiurement in day one,customer are coming to you in the first place for help, because they want this piece of software to be built by you.

Key Takeaway here Customer Collaboration in software devlopment is important over rigid forced contact and asking requirement for development in small parts over development period.

Value No.4 Responding to Change Over Following a Plan.

- This does not mean have no plan. In Software development we have all kinds of planing. Having a plan is Important.
- Traditionally when we're planing for projects what we are thinking(saying) is that conditions of the universe didnt change between now and the moment we finish it, this rearly happens.
- While developing a complex piece of software somethings will always go wrong or customer requirement change or there may be limitations etc this may render plan useless.

Key Takeaway here Responding to Change in software devlopment is important over just Following the plan. Thing doesnt go as planned always.

Postscript

Agile is really a concept or a philosophy. Off of that, different organizations and companies have built various frameworks that kind of walk you through this step by step of how agile is performed within that various company.

And a lot of those frameworks have become methodology such as **scrum, kanban and scrumban**. And those are the three most popular agile methodologies used by companies today. **In that scrum is the most popular Framework, majority of companies that are utilizing agile are utilizing some version of scrum.**

Kanban is a little bit different way of working through more of a continuous process rather than having various sections and segments that you'll learn about in scrum. Scrumban is really a combination of both. A lot of companies that use scrumban and are using it as a kind of a launching point for them to get from scrum to the kanban methodology.

12 Agile Principles

Agile principles are descriptive guidelines help us make better decisions thought our daily agile activities.

Agile Principle No.1

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

- First thing, is faster or early on; and the second thing, is continuously.
- Try to deliver value faster almost every week or every other week where you can give some kind of valuable software to customer a feature to the customer and they can give you Feedback.

Agile Principle No.2

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

- Responding to change in 21st century is crucial, for any business, even a small business, you need to be able to respond to change.
- So, what this principle is saying is, if somebody is part of an agile team and if they're delivering something to the customer,think about being able to create something for the customer, in case something changes on their business end.
- It's also about delighting customers as well, and really responding to their needs.

Agile Principle No.3

Deliver working software frequently, from a couple weeks to a couple of months,with a preference to the shorter

timescale.

- So, in this principle what we are saying is: focus on the working software. And when we say working software- it is not a requirement; it is not a mockup; it is not a wireframe. **IT IS [A] WORKING SOFTWARE** - something that actually works, a part of the feature that you can give it to the customer, receive feedback, and see how they use it.
- And then the second thing is, the timescale.You are trying to shoot for the shorter timescale a few weeks. And every few weeks, you're trying to have some sort of working software, and you're trying to give that to your customer.
- So, the working software as you can give it to them (customer); so they can get their hands on it. Because most change and most ideas will come out when they actually start working with it and they can see themselves trying to meet that business need. So, that... that's absolutely crucial.

Agile Principle No.4

Business people and developers must work together daily, throughout the project.

- So here what Agile is saying is: There's no need for a creation of this layer. Developers, the team, and the stakeholders or the customers, can actually talk and in fact, it's encouraged, they're talking on almost on a daily basis.
- And the other thing to remember is: we're not talking about business people asking developer when this is going to be done, when this is going to be done? We're talking about a collaboration. What are we building? Has anything changed? So, that kind of communication is happening throughout.
- With agile, it removes the blinders from the developer and allows them to get their questions answered directly from the source rather than going through a layer which doesn't really add much value. Yep, it just helps everybody to get on the same page on what they're building.

Agile Principle No.5

Build projects around motivated individuals. Give them the environment

and support they need, and trust them to get the job done.

- The key concept here is: there's no project manager trying to ask you, you know, what did you do yesterday? or like how much have you done? Like no status reports. The idea here is, you know hire a really good, motivated group of individuals, part of the team who are cross-functional, and actually give them space to actually do amazing work. And when we say work, not just devolve software but also to solve problems for the company or the client.
- So I think the big thing to keep in mind here if you're going to work in an agile team is, regardless of your title these teams are cross-functional. That means, they cross over roles. Your duties and responsibilities are going to evolve and change and that is because, everybody's collaborating together to try to realize that end value.

Agile Principle No.6

The most efficient and effective method of conveying information to and within a development team is face to face conversation.

- What this means is, instead of writing a big piece a requirement document and giving to a developer, we are encouraging more face to face communication. Or even worse creating a ticket, or messaging them, and bothering them all the time, just having meetings face to face, where you are able to communicate what you need.

Agile Principle No.7

Working software is the primary measure of progress.

- This principle is just saying that, whenever you're thinking about progress, let's measure that in terms of our working software. What have we built? Not how much analysis we have done, or how many pages of documentation we have created, or how many mock up pages we've created? This is purely working software.
- And while we may be creating models, as in part of that, and have

them attached to our various items. The working software is really what we're driving towards and what will create the most value.

Agile Principle No.8

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

- In traditional days where business analyst is doing all kind of analysis. They're stressed, and they hit their deadline, they finish, they give it to a developer. Now a development team is stressed out, and they're trying to make the ends meet, hit the deadline, and then finally they're done. Then it goes to Tester. It's Tester's time to be stressful now. They're testing everything, and then implementation time comes in, and everybody stressed out. So, in Agile, what we're trying to do is, instead of this big bang thing wherein everybody's working in phases, we're talking about, like, what are key features? What you need to do?. What is the team's capacity? And think about a sustainable pace that team can develop in.
- I think its really important that because in Agile the whole team is getting together, so, it makes it more sustainable, because it's a cross-functional team, so, everybody is able to help move that that particular project forward to meet the value.

Agile Principle No.9

Continuous attention to technical excellence and good design enhances agility

- This will tackle a little bit about quality in this principle. So, if you build something and if there is no quality in place, you cannot build anything on top of it. So, it's going to be harder for you to be agile in the future. So, agile says: think about quality, think about technical excellence, as you're building certain products.
- You evolve those features and functionality you've built, right? You're making slight changes to them, and additions to them to help drive value later on. If you've got problems early on in that design, or in that initial quality, you're gonna have problems for a

long time.

Agile Principle No.10

Simplicity- the art of maximizing the amount of work not done- is essential.

- When I think about this principle Jeremy, I think about Google's landing page. So when you go to Google's landing page, it is just one search button and it's a white screen. And it is beautiful on the way that it works. Just because you have more and more features, it doesn't make a software pleasant to use. So the idea here, is in Agile, you're also thinking about the outcome, and what ou-... what kind of outcome are you looking. And you can do that by building less and less, and delivering more.
- So, making sure that you're delivering value on everything that you're building, not just building something, or creating something to create it.

Agile Principle No.11

The best architectures, requirements, and designs emerge from self-organizing teams.

- So in agile teams, where the teams are self-organizing and cross-functional, meaning depending on what kind of project you're working on, you're going to have people with the right skill sets. And that's what a team is formed of. Is everybody in the team has unique skill sets and they have what it takes to make this project successful. So, when they need something- a question about an architecture- they already have somebody in that team, so, they can actually talk to somebody, versus borrowing somebody, from somebody else.
- In waterfall, or in other methodologies, a lot of times you need to go ask a manager. "Hey, can I have somebody from your team- from your testing team- to help out with this?". And it usually takes a long time. Where[as] in agile, with it being self-organizing, if you need somebody, you can go tap them on the shoulder and see if they have availability to come help your team, or to join the team potentially.

Agile Principle No.12

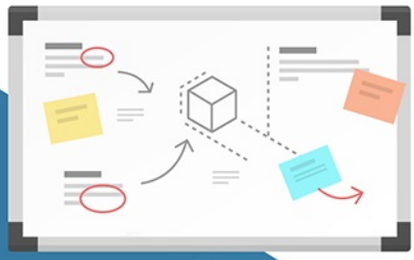
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

- This is actually one of the key principle of agile, that one my favorite principle of agile (instructor)as well. So, as you're working in a product as a team, you're going to learn different things about each other ways of working. So at each interval like every two three weeks, you're getting back and you're looking at what went well, what didn't go well, and how can we improve. and this is a key principle of agile.
- Agile, with it being more iterative, every few weeks, you're actually able to look and see how it's going and make adjustments as necessary, to make the team even more successful.

12 AGILE PRINCIPLES

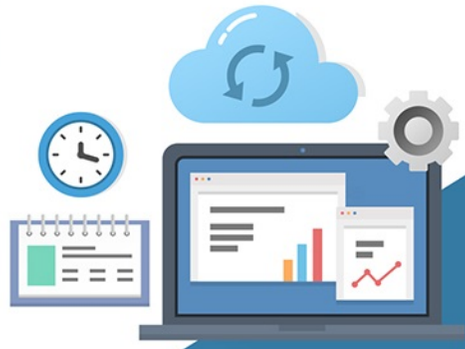
DELIVER VALUE FASTER

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.



WELCOME CHANGE

Welcome changes to requirements, even late in projects. Agile processes harness that change for the customer's competitive advantage.



DELIVER WORKING SOFTWARE FREQUENTLY

Working software should be delivered after a couple of weeks to a couple of months, with a preference to the shorter timescale.

WORK TOGETHER DAILY

Business people and developers must work together daily throughout the project.

BUILD PROJECTS AROUND MOTIVATED INDIVIDUALS

Give them the environment and support they need and trust them to get the job done.

FACE-TO-FACE CONVERSATIONS

The most efficient and effective method of conveying information to and within a development team is with face-to-face conversation.

WORKING SOFTWARE IS KEY

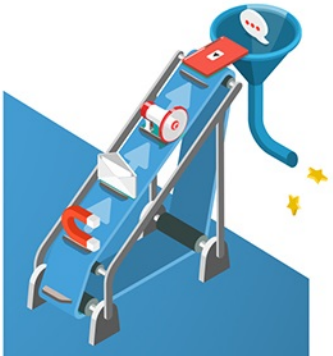
Working software is the primary measure of progress. Not requirements, not models, not designs... working software.

SUSTAINABLE DEVELOPMENT

Agile processes promote sustainable development. The sponsors, project team members (developers), and users should be able to maintain a constant pace indefinitely.

ATTENTION TO TECHNICAL EXCELLENCE

Continuous attention to technical excellence and good design enhances agility.



SIMPLICITY

The art of maximizing the amount of work not done is essential.

SELF-ORGANIZING TEAMS

The best architectures, requirements, and designs emerge from self-organizing teams.

REFLECT AND ADJUST

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

THE AGILE COACH

TheAgileCoach.com

The BAGUIDE
TheBAGuide.com

TheBAGuide.com

Benefits of Agile

Agile was created because of the pitfalls and downfalls of a *traditional waterfall methodology.*You see with waterfall there's various phases and you can't move to the next phase until that previous phase is complete. And once you've moved past a phase, it's very difficult to go back to a previous phase.

So for waterfall you need to identify all the requirements up front, then you need a model and analyze all those requirements. Then you need to develop some type of solution that meets those requirements. Then you need to develop some type of solution that meets those requirements. Then you go through the testing and move to production phases.

But if you're already in the development or testing and you identify some changes, to go back it takes a lot of work for that team to go back to requirements and push that as a separate piece through that waterfall process. As well, because of having to do those things step by step by step the value isn't delivered to the end user and to the customer until the end of the process and so it causes a lot of problem.

As well, **with waterfall it's a much longer period to deliver any type of value or gain any feedback from the users or the customer. that ultimately asked for this solution because you have to**

gather all the requirements, go through design development and testing, that is months, sometimes years to actually receive some type of solution that may or may not meet that business need. Agile was created to help solve those problems.

The number one thing that agile does is it allows you to deliver that **value in smaller increments to the customer**. Not only as a customer that end users get to start utilizing that solution and seeing that value, but also they get to give you feedback. They really like it or they don't really like it or this should be adjusted or the business has changed. And now we said this, but it needs to really be this now.

Agile allows you to do that, you deliver that little bit of value, you gain that critical feedback that will change and adjust the way you move forward on your project. And it really helps to solve a lot of those problems before they become big problems. And because agile is more of an iterative process, the project team is actually able to get feedback on their performance as well. The project team can understand what they could do differently and what they could adjust to make them even more successful and make their solution even more successful.

Some of the challenges to using agile

Well the first thing is agile is actually difficult for existing companies and organizations to implement if they're using some type of methodology like waterfall or other methodology today. And the biggest reason is agile changes everything. It really has to change the whole mindset of the company. It sometimes has to change the organizational structure, has to change the way people and teams in various roles work with each other. **And it really has to be an all or nothing process.**

So a lot of companies will kind of go into and want to move in the agile and they'll do it half heartedly because oh yeah we want to get better and we want our solutions to be out faster and we want to receive all those benefits of using it. But they do it half heartedly and in the end years down the line, they're still using a half agile half not an agile approach. And let's just say it doesn't work.

So that's one of the big challenges is companies have to be all in and ready to spend some money and a lot of time in making that adjustment. Now we're seeing that new companies starting up have a much easier time of making this work. And that's because they don't have existing processes or various things that need to be adjusted. And

that company culture can just be built around that agile philosophy and that agile mindset.

One of the really nice things that a **lot of team members like about agile is the real lack of documentation,** honestly, agile really focuses on more on conversation and communication than writing all of these requirements out in building models and all that stuff. Agile really has that all done through let's sit down, let's discuss it, let's hammer it out and then we'll move forward on whatever is decided there. **That can pose some real challenges when the project's over. And now you have maybe a support team handling this solution that was created. well, there's not any real documentation to tell that support team how it works or some of the common things that come up, it can be a big challenge for them to actually learn and support that particular solution.**

And while we're talking more documentation, **another challenge for the lack of documentation is reusing features or components.** In a traditional methodology such as waterfall you're documenting those features out, your document the design, your document all the analysis points that you've really thought about as you've designed it and as you've developed that solution and that feature, that component can be utilized on additional projects so if you have additional

projects later that are very similar, you can utilize that documentation to implement that feature on that next project. **Because of agile not having much documentation and really that information only being stored or really siloed by the project team members that worked on that particular project, it can be much more difficult in agile to take a feature from a previous project and successfully apply it to the new one.**

And the last challenge we're going to talk about is really there's not a clear role in that agile that takes control or has ownership of that particular project. Instead the team works together collaboratively and everybody chips in and does their part to make sure that the project meets that eventual business need. **The challenge arises when the project goes off course, when it kind of goes into an area that wasn't planned for. Now there's no real role inside of that team to help bring it back. Everybody in the collaborative teams kind of looking at each other and not really sure who should be stepping up to correct the path and get back on plan.**

Scrum

Scrum framework is a very lightweight **framework** that was designed by agilist to **solve complex adaptive problems** that require **sprinting or iterating through solution.**

Scrum's root come from empirical process control framework that values inspection, adaption and transparency. So let's look at inspection and adaption. Inspect and adapt pretty much means that as a team picks up the work and they start working, after some time they can learn more about the problem and the solution and a better way of delivering software.

So scrum says you pick up our problem, you start working on and as you work on it you can look back and inspect and adapt according to what you learn from the process of working itself. Not only you can inspect and adapt on the problems that you're solving, you can inspect and adapt on also how the team is doing and how the team is leveraging the scrum framework.

Transparency on the other hand make scrum framework very effective because all the roles, responsibilities, the meetings and scrum are designed so that team can be more transparent about how they're working and what they're working on.

Let's look at the heart of scrum which is the

sprint. Sprint is just the time selected by the team anywhere between one to four weeks, where they plan the work, they get the work ready, the develop and test and get it production ready so you can put that software in the hands of the customer by the end of the iteration or the sprint.

It's really effective because all the meetings actually happen within the sprint, where do you plan to work, you do a daily checking and you review the work at the end of the timebox or the time that you selected, let's say for example two weeks. And you can also do a retrospective on how the team is doing in terms of working with each other.

Scrum is an agile way of developing software or any project. A lot of times out there people perceive scrum as a methodology, but there's a lot of utility if you look at scrum as a framework. When you look at scrum as a framework, then it's a little bit more than a process. It's more of an adaptive framework that can allow your team to become agile and you can place more value on being agile than just doing perfect scrum. Scrum highly recommend a self-organizing team and cross-functional team. That means if a team is going to take on some work the team members should have the skill sets to be able to develop the solution for their work. If there is a conflict or something that the team doesn't know what to do,

scrum recommends that team discuss among each other and come to a conclusion as a team. **Scrum was initially devolved to manage software products.**

Outside of software development, scrum is used in schools and education, operations, startup, government and marketing. Scrum is unique in his ways because he has very prescriptive roles, responsibilities, ceremonies and artifacts.

Scrum Values

These values are very powerful and effective because scrum is leveraged by a team.And these values really support team members when they are working to solve complex problems.

Openness

Openness just means that team members are open to living the scrum values over just doing scrum. Team members are open to uncover and find effective ways of solving problems. Whenever team members are solved on a tough problems, openness could mean that they are open to inspecting and adapting and figuring out better ways to solve that problem as a team. So being open to other people's ideas and being open to change in general.

Commitment

Whenever people are looking at this value, people

immediately think about commitment to doing the work or executing on the sprint goal or the work that team has committed to, but it's actually more than that. This means that team members are committed to each other in terms of doing their best and coming up with solution collaboratively. This actually also means commitment to not only just delivering software, but delivering software to solve end user's or customer's problems. Commitment also means that team members are going to give their best action and effort to solve a problem versus just focusing on a sprint goal. Focus is another very important scrum value. If you are trying to solve any kind of problems, human beings need to be able to focus. Scrum recommends that team members plan on a certain amount of work and they focus on that for a certain period of time and the whole goal of the team is to get this work that they picked to done.

Focus

Focus is another very important scrum value. If you are trying to solve any kind of problems, human beings need to be able to focus. Scrum recommends that team members plan on a certain amount of work and they focus on that for a certain period of time and the whole goal of the team is to get this work that they picked to done. Focus can also mean that team members are focused on solving a problem that they have

decided as a team to do versus thinking about every shiny thing that's out there and trying to build everything. Focus can also mean that team members are focused on customer happiness and delivering value to the stakeholders

Courage

Mark Twain said: Courage is not the absence of fear, It is acting in spite of it. Courage in this context means that team members show up with courage when they are working on tough problems and are not afraid to try and come up with their bold ideas.Change is usually hard and we are not perfect. However this scrum value encourages team members to do their best despite not being perfect and despite of change being hard.

Respect

Whenever we are working on difficult problems and we are working as a team and we share our success and finish together we will be professional and will respect the team members culture, their background, their ways of working and we will also respect that opinion.

Sprint

Sprint is the heart of the scrum framework. Sprint pretty much just means it is a **time-box, it is a set of time.** Scrum recommends anywhere between **one week to a month where you have** this time-box where **team members** will plan the work along with the **product owner** and **scrum master** and they will **deliver** something called a **product increment** at the **end of the sprint.** So the sprint comprises of all the **scrum ceremonies.** So the **daily standup** happens within the sprint, the **sprint planning** happens within the sprint and the **sprint review and retrospective** it's part of the sprint.

Roles in Scrum framework

Product Owner

One of the main responsibilities of a product owner is to talk to the stakeholders or the users and understand the product vision from them, understand their challenges and what kind of problem they are trying to solve.

Then the product comes back to the team and helps the team understand what are some of the needs and the challenges that the team will be solving with the product that they are developing. And, to do so **product owner writes this very simple description of a feature from an end users**

perspective. And they prioritized that user story in a way where the development team knows exactly which ones to work on.

- So there's a lot of managerial type of stuff that really helps out as a product owner.

Scrum Master

SCRUM MASTER IS NOT A PROJECT MANAGER. HE'S ALSO NOT A TEAM MANAGER. Scrum Master is a **servant-leader role** that is there to **remove the impediments** or the roadblocks in that iteration or time box where the team is very focused on working on those work items. Scrum Master also **coaches teams on the scrum values** to live the scrum values and **facilitate the various scrum meetings or ceremonies.**

- **"What is a standup meeting?" According to The Scrum Guide, "the daily scrum is a 15-minute time-boxed event for the development team" to plan for the next 24 hours.**
- In daily standup the Scrum Master generally asks the following: What did you do yesterday ? What are you going to focus on today ? And do you have impediments(roadblocks) ?.
- They go around and see how everyone is doing. They make a list of impediments so that they can **make visible impediments** or can work with a team member to go out there and

resolve that impediment.
- In an environment where people didn't want to answer those three questions. They do did something called appreciative inquiry. So the question instead of asking 'what did you do yesterday?' what we said is 'what are you going to focus on today? Are we in track to meet our sprint goal?'
- Will facilitate sprint meetings. And do whatever Team needs. Scrum master **facilitate Sprint review.The sprint review is one of the most important ceremonies in Scrum where the team gathers to review completed work and determine whether additional changes are needed.** The official Scrum Guide describes it as a working session and makes the point that the "Scrum team should avoid limiting it to a presentation."

The Development Team (Dev Team)

The Scrum framework says that **everybody else except the product owner and the Scrum Master is a team.** For example, a QA - Quality Assurance Analyst or a business analyst or an architect. All of these roles come under the team. **A team is a professional who can do the work of delivering a potentially releasable increment at the end of the sprint.** The team members are self-organizing,

meaning that they don't need a manager to organize their work. They will actually talk among each other and figure out which I am to work on next. Obviously looking at the priority provided by the product owner.

They **talk to the product owner in almost a daily basis** just to understand the priority and sequencing of the work. And there are also cross-functional. **Cross-functional means they have all the skill sets that are required to develop that product or the feature for that team.**

- In an Agile team, no matter what your title is you're just part of the team and you want to You are a self-organizing team. But we both want to help each other out.
- So when you are a developer, part of the Agile team. You're not saying that you can only do this. You still have to learn other components of what the team was developing.

Scrum ceremonies

Daily stand up or The Daily Scrum meeting

LET ME START WITH WHAT A DAILY STAND UP IS NOT. THIS IS NOT A PROJECT STATUS MEETING WHERE A MANAGER OR A SCRUM MASTER IS THERE TO COLLECT STATUS. This is a quick 15 minutes meeting to do inspect and adapt, which the core of Scrum Framework where everybody comes in and everybody answer the three question. **What did you do yesterday to achieve the sprint goal? What are you going to do today that will help us achieve the sprint goal? And do you have any impediments that might hinder or become an obstacle towards us achieving the sprint goal?** Usually, the team will go one by one and answer these three questions and walk out of them with a very clear plan of attack for the day.

This meeting is held at the same time and same place just to reduce any complexities.This meeting is held at the same time and same place just to reduce any complexities. The Scrum Master facilitates this daily stand up meeting and is responsible for helping the team understand the value of the daily stand up. The team is responsible for having this daily Scrum meeting. The product manager and stakeholders are allowed to come and join the meeting but they are not allowed to interrupt the team. Scrum Master is

responsible for protecting the team from external distraction during the stand up meeting.

In summary, this is the meeting to inspect how the team is doing towards achieving the goal for the sprint and identifying and taking an action item for resolving any impediment or the roadblock.

The reason why this is called a stand-up meeting is that when people are standing up it's harder for them to go on and on because everybody else is also waiting for you to finish.

Sprint planning

So what happens in sprint planning, just like it sounds it is a sprint. **Let's say for this example this is a two week sprint. We are planning this two weeks time-box. We were planning on what goes in used two weeks where we can take in the work and we get this work done done, that this work is shippable to the customer, the customer can use it and give us feedback.**

So when you're planning it, we're not just planning for developing it, we're planning for developing and then testing and kind of making sure this product is ready to be shipped to the customer. Scrum master facilities this meeting meaning he will get the counter invite to everybody and he'll reserve the room and make

sure this time is blocked in everybody's calendar.

Product owner is one of the main players in this meeting. Product owner comes in and he understands the vision of the product. He knows the needs and the challenges of the product. And he's coming back to the team and he's saying that hey I know these items are the **most important items for the stakeholder and this is the priority.** And once the team looks at the priority, they can now say that okay, well based on this priority these are the items that we can, we feel very confident that we can deliver. But this one or two items we're not very sure about or we need to do more research. So then it's **pretty much a back and forth discussion between the team and a product owner.**

Maybe they're negotiating on what should be the scope of the sprint. And after the discussion, let's say it is a two week sprint the planning will be for two hours. **And in these two hours, they create a clear concrete plan on how they're going to work and how they're going to create this potential potentially shippable item in this two weeks iteration.** They will also have already learned about the risk. the dependencies that the work in the sprint might have and they would, they actually had resolved these challenges, risks and impediments for the team to start out to work. The other thing that **they walk out is a very clear**

picture or a goal for the iteration or the sprint.

They know exactly why they are doing this work and it's not necessarily just individual people working on one or two stories, it's for one specific goal that they have. One last thing to remember is a **sprint backlog** comes out of the sprint planning meeting. **Sprint backlog is the amount of work that the team member agreed to work on on that sprint.**

For example, we took a two weeks iteration, let's say they have about twelve user stories that they said if we can get this done in the next two weeks, that is the sprint backlog. And let's just take an example, there are a few scenarios of what happens in the sprint planning meeting. When I was a scrum master, the team had come up with the sprint goal and the team have already committed to a certain amount of user stories and director of a product came back and said oh, we need these two things done right away. So what would you do when you are running scrum and somebody comes back and wants to add more scope to your sprint? One of the things that I did is actually went back to the product owner and I also communicated with the director saying that hey, this is great that this is important. Would you like to talk to our product owner who does the prioritizing of the work? And based on the urgency and the value that it brings to the

organization we can prioritize and put that in the next iteration.

There was another time where this was peak season for this team where they will get a lot of production support or they would have to support the customer base where we knew that, you know, some of the tickets were common. Team members would have to support these users. And one of the things that we did in the sprint planning meeting is we said hey this is already happening. How can we allocate some time, some buffer time so we can also focus on supporting our user base because this is a very peak one month of the year where we need to be there for our customers. So we allocated five story points per sprint for supporting the customer in that sprint.

Sprint planning

The sprint review is held at the last day of the sprint. And this is the event where the team comes together and Scrum Master facilitates this. And invites the Product Owner and the key stakeholders who might be involved in that Sprint work. **Everybody is looking at based on what we did, What are some things that really got done? and what are some things didn't get done? They're also looking at the overall Product Backlog. As the team was working on the Sprint, did there anything else change in the product backlog? that might inform a different priority**

for the next sprint.

So this is a very informal meeting. This is not a project status meeting. This is for the team to showcase on what are some of the progress that they've made. The other thing that's discussed in this meeting is what are potentially uses of this product increment that a team has just developed? What are some other products or competitive companies with similar features are doing? So it's a really good discussion to kind of really look at, what's already done? and how should they evolve the product backlog. And it kind of gives you some key takeaways for the team and the Product Owner. So a Sprint Review is a meeting where you're just revealing what the team did and you want to watch out for some of these things.

If you're working with this new team, chances are at some sprint, the team will say we've got all these things done. We have nothing to demo and we just had got some stories done and we don't have a fully functioning product. Is it important for the team to do Sprint reviews, still? I, as a Scrum Master, would highly recommend teams still do a review even though if it's not fully functional a demo of a product. It's important for stakeholders and the product owner to be there and give some feedback on the work that a team did.

If the team knows that the Sprint review is going

to happen no matter what. The team's focus automatically will be on, Okay, we are already going to have this review. Maybe we should have some kind of increment in the product that we can demo to our stakeholders. And whenever we have time we want to at least have some really good question to ask him. How we are evolving this product? **Another thing that actually happened when I was working as a Scrum Master is a Product Owner or the key stakeholders not being in the Sprint Review where you lose a lot of the value of that meeting when you don't have anybody who can look at the product except the people who built it. And at that point I, as a Scrum Master, had a conversation with the stakeholders and the product owner on being a little bit more available and treating the Sprint Review as a priority.**

Sprint retrospective

Just like it sounds like, it's we're looking at a sprint and we're looking at the retrospective. Retrospective meaning what did the team do well, what didn't go well in this last sprint and what should be improved on. So that's basically what we're trying to capture in this retrospective meeting and this is a two hour meeting for a four weeks sprint. And it could be a one hour meeting for the two week sprint. So scrum master is the one facilitator who sets the stage for the team.

Again creates the safety container where everybody feels empowered to talk about not only the good things, but also some of the bad things or some of the challenges that are happening within the team member or some conflicts that team members have.

Now this next step is gathering data. Like okay, what happened in this last iteration of the sprint? What are some of the challenges that you saw? And everybody will collect data on what happened, you know, what can we learn from this last sprint. And after everybody collects, **these are some of the things that happened in the last sprint, then it's time for the team to generate some insights, like why did this happen, what happened, let's learn more on like the depths of challenges or the problem that the team had. And after they have a good amount of insights, they actually decide on how they want to go about and improve that. So this is an opportunity for the team to have a learning action plan.**

So despite of what happened in the sprint, the team members walk out with a clear plan and what they're going to do and how they're going to improve. **And sometimes the team will vote on what are the three main things that they want to implement or they want to be mindful of in the next sprint.** Retrospective, you're bringing in every two weeks or three weeks, and sometimes

just asking those three question: what went well this sprint? What didn't go well? And what should we improve? Those things can kind of get old and I think it's important for teams to kind of change things up a little bit. I've done things like taking team out on lunch and doing a lunch retro, where it's a little bit light touch, more for conversational retro. And we come up with a few, just a few items on how we want to improve.

One of my favorite ways to facilitate a retro is to do a sailboat retro. So you just draw a sailboat and you draw an anchor and anchor meaning what are some of the impediments. And you draw some wind on the top saying that the winds are some of the things that actually help us and you actually draw a shark in the bottom and the shark would actually imply those are some of the risk that the team might have. And you actually draw out an island asking teams, OK, where is that island? Where do we want to go in the next few months? How do we want to be known in the organization? And what we do and what kind of value we add to the organization?

Another example I'll give you for a retrospective is this is a meeting where we are not there as a team to blame on each other, sometimes things go wrong. If you're working on a complex problem you have to experiment. And whenever you experiment sometimes things go wrong. And one

of the important things as a scrum master, that I did, is I actually created that safe space for everybody to come back and just say what did we really learn, not to blame the people, but what did we learn from what happened and what can we do to improve so that these things don't happen and we make that person recognize the mistake that he made. But it's nothing personal, it's something that we all learn as a team and we move on.

Post Script

Backlog grooming or Backlog refinement

This is one of the things that happened within a sprint, it's called backlog grooming or backlog refinement. So once the sprint is live, you also want to prep for your next sprint. So those stories are already grooming. You're making sure the conversations are happening, the confirmations are happening, the user stories are there so that we could bring it while the sprint is happening, we look at our next dprint and we say OK, these are some of the things that we need to talk about. And you know it's almost like a workshop and you're kind of looking at what kind of problems we need to solve, what are the different potential solutions and you are adding your details to the stories, you know, in the backlog grooming session. Pretty much you're grooming your backlog, you're adding more details to your backlog user stories and getting them ready for the next sprint. Yeah. As a business analyst that's a very interesting point, when you're working as a business analyst or other roles I'm saying business analysts in a very loose term. Yeah. But when you're working in that, you are for

half of that sprint pretty much working in that spirit, and for half of that sprint you're kind of working out of that sprint because you're looking ahead to that, you're going to prep everything, so those user stories are ready, so they're ready to be pulled out. So if you are, most of the time those business analysts, the product owner will get that in there, then it's up to the team to iterate on that and keep checking back with that product owner to make sure that it's in line with their vision when they kind of created that initial user story. So that was always, that was actually always a tough thing as a business analyst is balancing between the current and the next and making sure that you're getting everything done.

SPRINT PLANNING CHECKLIST

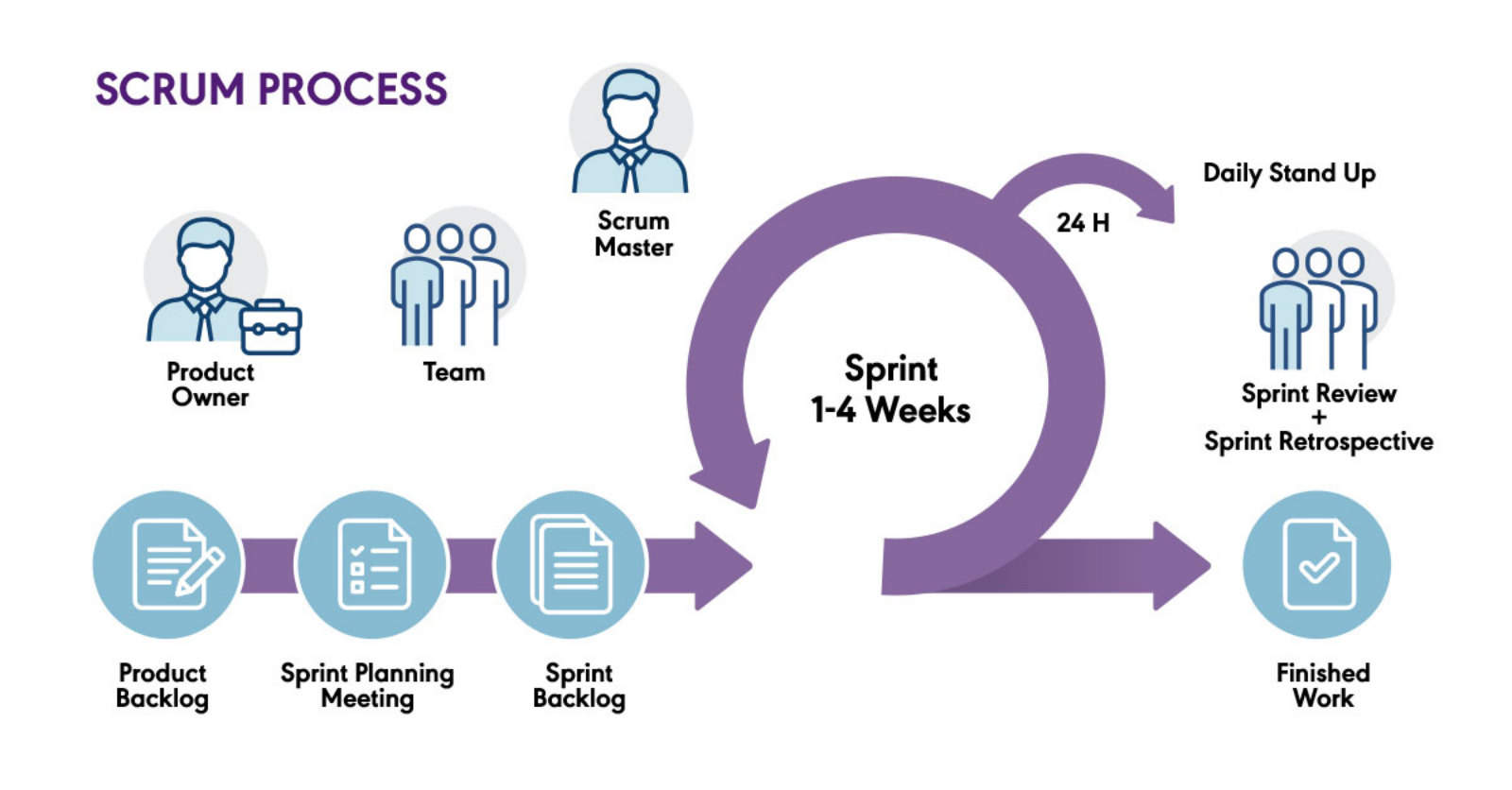
What:

- [ ] Do user stories all contain who, what and why?
- [ ] Are we clear on the acceptance criteria of each story?
- [ ] Have you split large stories into small ones and estimated them?
- [ ] Do you have enough stories for team members to work on for the sprint?
- [ ] Is the priority on the stories clear and understood by everyone?
- [ ] Are the stories on the sprint backlog independent (not dependent upon other stories)?
- [ ] Does the whole team agree on the sprint commitment?
- [ ] Has the team considered PTO (paid time off) and holidays of team members while determining the sprint commitment?
- [ ] Does the team have a Sprint Goal?

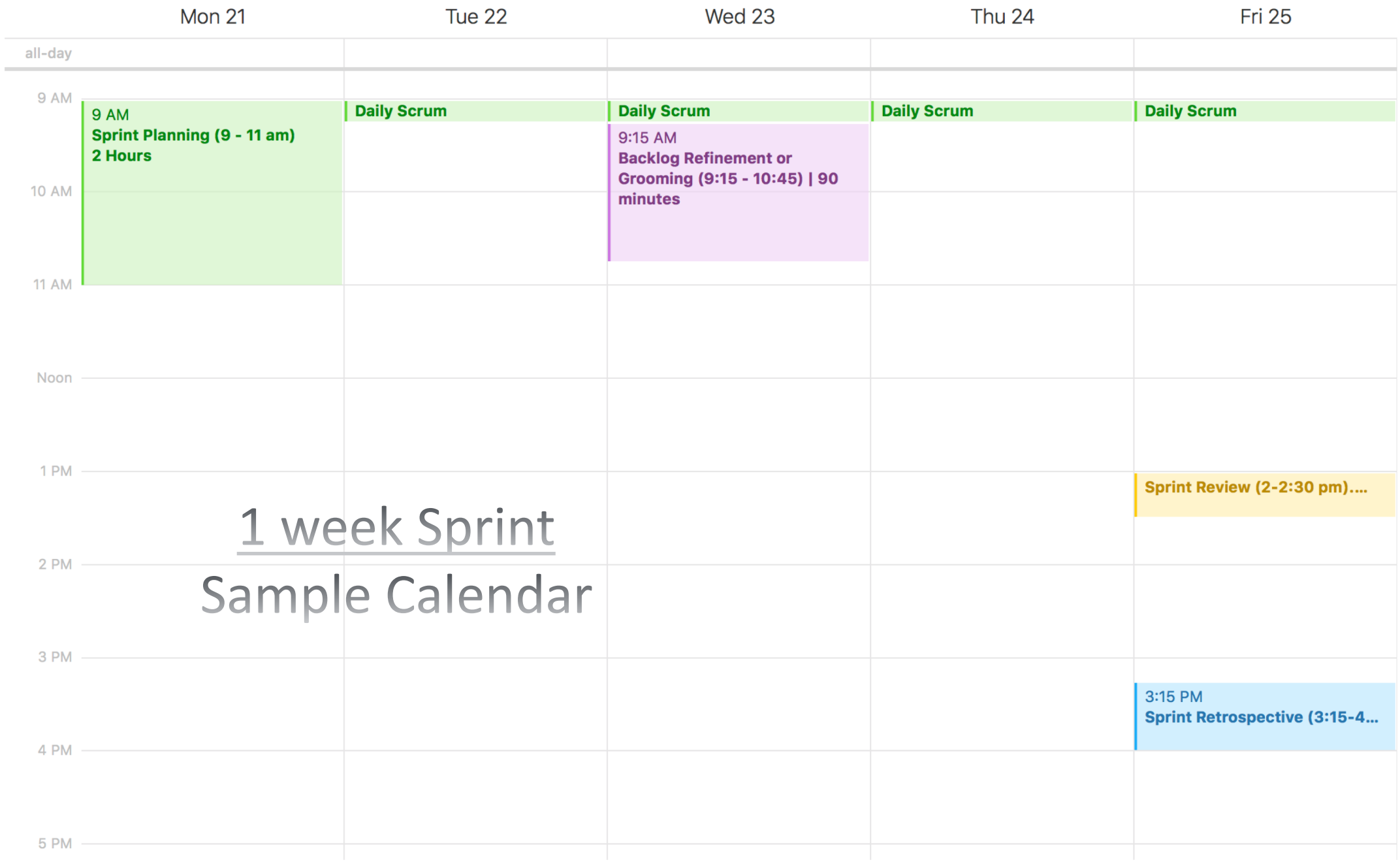
How:

- [ ] Does the team have a shared understanding of Definition of Done?
- [ ] Are there any impediments the team can proactively work on that can help the team to achieve flow?
- [ ] Does every member understand what needs to be done on their respective stories to meet the acceptance criteria?
- [ ] Can team members agree to work on a single story at a time (to avoid context switching)?
- [ ] Are there any knowledge silos or gaps where a task can be done by only one person?
    - [ ] If so, does the team have a plan to share that knowledge with everyone so that everyone in the team can work on that task.
- [ ] Have you put the (most urgent) retrospective action item from the last sprint in this sprint?

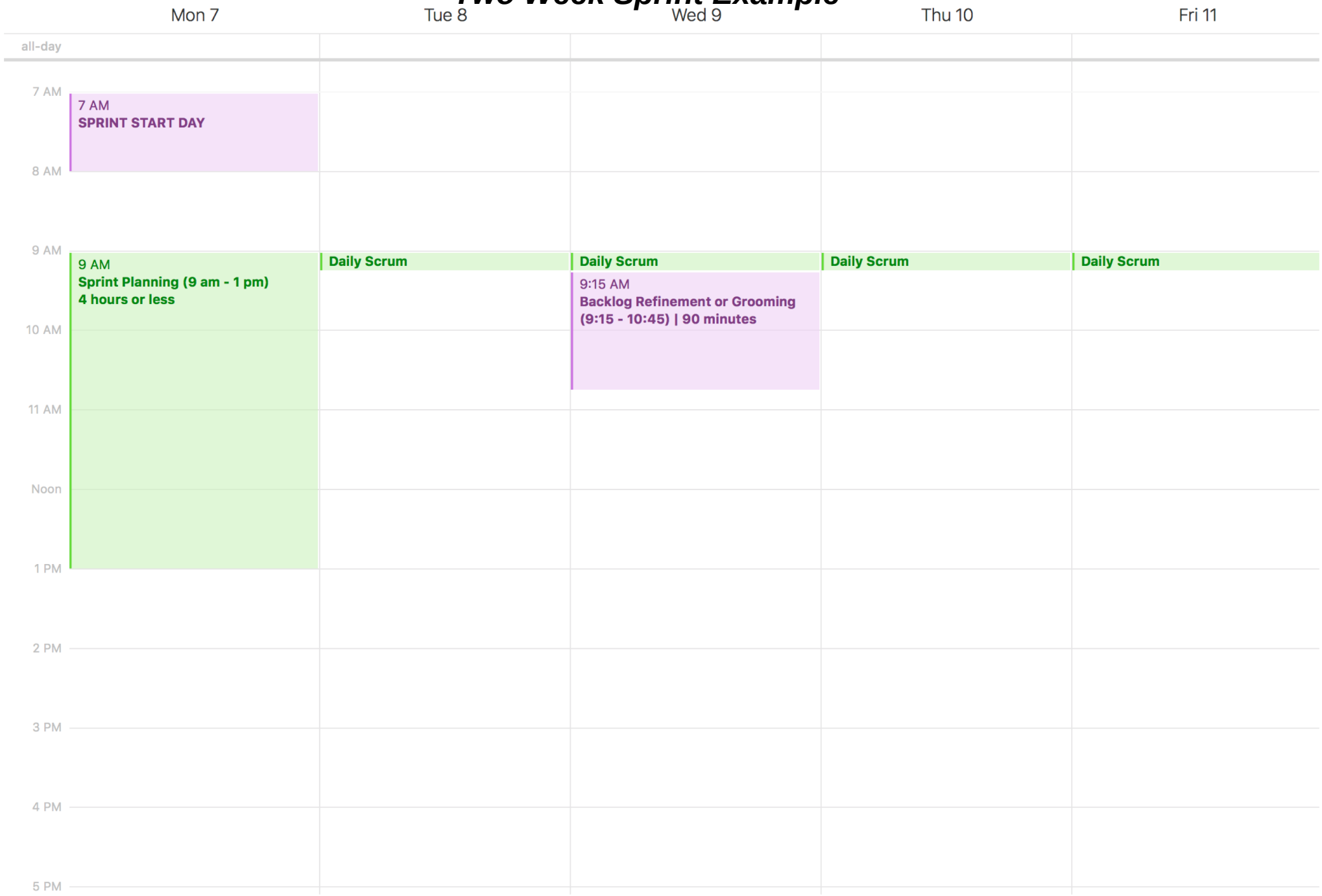
SCRUM PROCESS

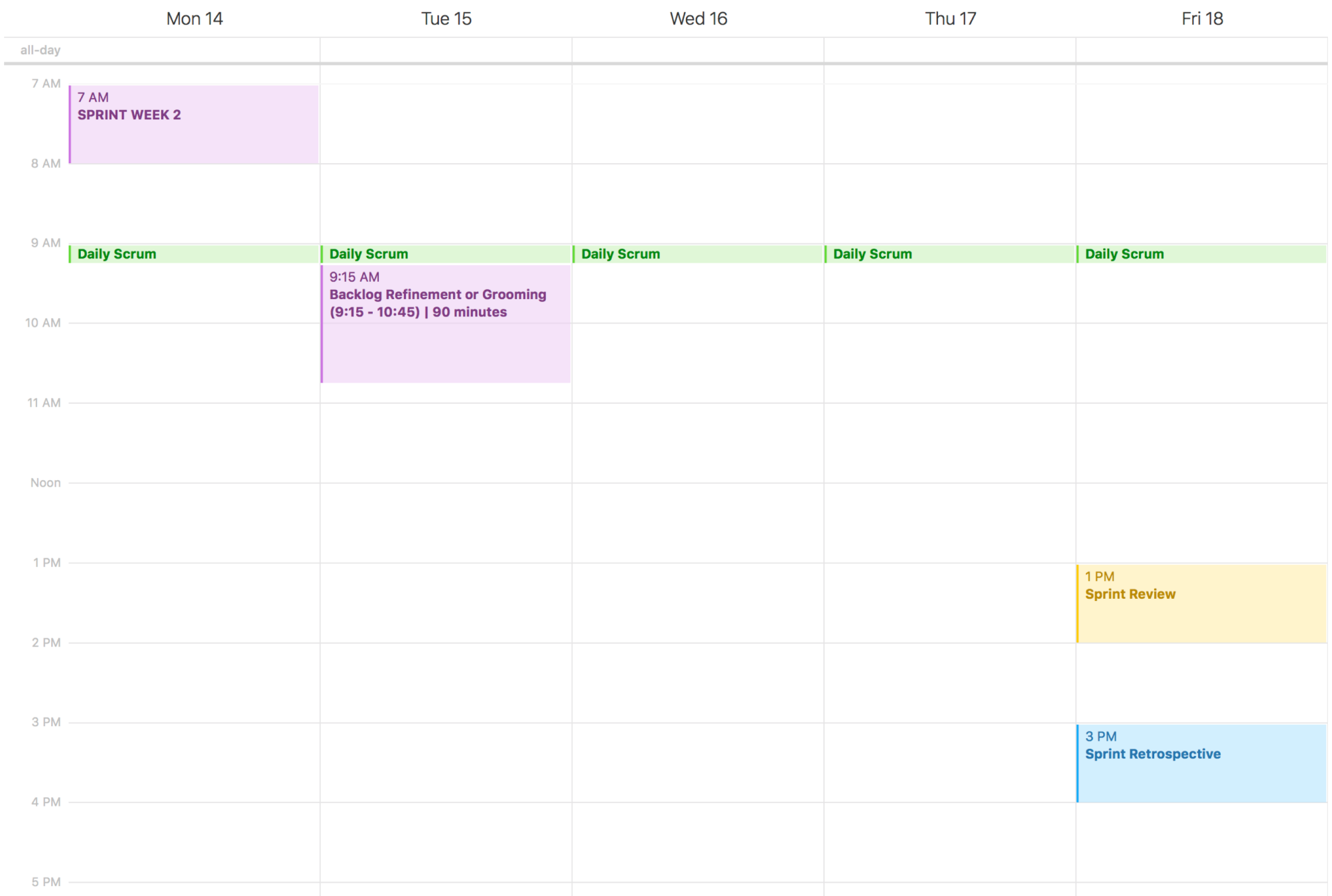


One Week Sprint Example



Two Week Sprint Example




Mon 14
Tue 15
Wed 16
Thu 17
Fri 18
all-day
7 AM
8 AM
9 AM
10 AM
11 AM
Noon
1 PM
2 PM
3 PM
4 PM
5 PM
7 AM
SPRINT WEEK 2
Daily Scrum
Daily Scrum
Daily Scrum
Daily Scrum
Daily Scrum
9:15 AM
Backlog Refinement or Grooming
(9:15 - 10:45) | 90 minutes
1 PM
Sprint Review
3 PM
Sprint Retrospective

Key Scrum Terms and Artifacts

User stories

What are user stories ? User story is a simple description of a product feature that is written from end users point of view. That means whenever we are writing this product increment, we are thinking about the customer.

Now let's look at how do we capture this user story ? and what is captured in the user stories ?

- **The first thing that you want to look at is the who. Who is the user ? What kind of customers are they? This helps developers to really think about and empathize with the customer who's actually going to be using this software.**
- **When we deliver this product increment, what will we empower our customers to do ? What is their goal ?**
- **And then the last thing that we want to capture in these user stories is why. What kind of value will it bring? If you're able to deliver this user story to the customer.**

Let's look at the format that most of the companies use, as a user or type of a user. I want a goal, I want to solve a certain kind of problem or that is more for what. The third piece of the story is so that I can achieve some value or why. So the who, what, and why can be captured by

using this format of the user story.

Now let's look at how do we write a good user story. Is the user story just a requirement document? or something that replaces the requirement document? **The user story is not a document, a user story is more of a conversation. It's a lightweight description of what the user might want in this software.**

There is a really neat way of thinking about a user story and this is an acronym called **Three C's. 3C stands for card, conversation, and confirmation.**

- **If you look at the first step, Card. This really means that Product Owner after he talks to the customer or the stakeholders come back to the team with a (3 / 5) three by five card that captures the who, what, and why. And this is just a high-level need of the user. Remember, how there is not a lot of description added or a lot of time spent in documentation. This is just a three by five card that this only has a necessary piece of information and that's brought to the team where the conversation actually happened.**
- **So that's the second piece, Conversation. Team members and Product Owner discuss about this card. Okay. Who is this user story for ? You say, this is a first time patient. Great. What did they want ? What kind of problem are we solving ? So that's what we discussed, the**

who, what, and why. This really helps developers to understand how they want to build this product increment.

- **The third step is Confirmation. The team will then go talk to the product owner. The development team is confirming that what they have understood is the same as what product owner or customers have in mind, just to avoid any confusion.**

So by just going through these three things, Card, Conversation, and Confirmation. You can write a quality user story.

User Stories: Acceptance Criteria

Acceptance criteria are simple notes or conditions added to the user story that tells you what the user story must do so that it can satisfy the need of the customer that is defined by the product owner. ***It should be clear, concise and product owner should be able to verify it.***

We talked about the 3 C's. ***Acceptance criteria shows up in the confirmation phase,*** where our team members and product owner are collaborating with discussing and coming to an agreement on the acceptance criteria of the user story. **Again these conditions of acceptance statements are known as acceptance criteria.** Acceptance criteria enriches the user story by making it testable and also it ensures the story by making a demo to the

stakeholders or the product owner or the external customers. ***The product owner writes the initial acceptance criteria and brings them to the team and they decide and come to an common agreement on what acceptance criteria should be part of that user story.***

So let's look at an example. Using our very simple user story format we're going to define who, what and why. As a new student, I want to be able to create a new account on theagilecoach.com so that I can enroll into a course. This is a very simple story. How can we add more details and how can we make the story more testable and clear? Well, we can add three acceptance criteria on this that defines the entry method. Create an account through Linked-In. create an account through Google and create an account through Facebook. These three conditions are very specific. These are three logging and account creation portals and it's really easy for developers to understand they need to integrate with these three platforms and allow the new students to log in and create accounts from these three social media platforms.

Lastly you want to remember these three things:

1. ***Acceptance criteria should be testable with clearly defined pass/fail results.***
2. ***The acceptance criteria should be clear and concise, the developer should be able to understand it. No poetic language or***

philosophical statements just to avoid any confusions.

3. *And acceptance criteria is established with shared understanding between the team and the product owner.*

One might ask a question how many acceptance criteria per user story. This is totally up to teams. I recommend teams, right, anywhere between three to seven acceptance criteria and if it's more than that, they can always split that user story into two.

Writing Great User Stories

Let's look at a technique called invest and we're going to look at this acronym to write quality user stories. Usually most of the people when*they think of a user story, they just think of this simple format as a user I want this goal so that I can achieve this.* You can compare this to an analogy of an iceberg, that is just the tip of the iceberg. Most of the value comes when you can pay attention to invest acronym of the story. What do I mean by that? Let's look at these guidelines on writing great user stories.

INVEST

1. *The first one is independent. Independent just means that when you're writing a user story you don't have any dependency with any other user story and a team can spin it and get it to done without having to depend on any other user*

stories.

2. *The second one is negotiable. Whenever the product owner or the product manager is coming to the team and he's bringing the card, based on his understanding of the needs and the goals of the users, this user story should be negotiable. Meaning once after talking to the team, if the product owner has a better idea this user story should be flexible so that it can change and adapt to the needs of the user.*
3. *The third one is valuable, that this user story has some concrete value to the users.*
4. *The 4th one is estimatable, this just means if you have a user story, you should have some idea on how much complexity this user story has.*
5. *The fifth thing is small. It should be small enough for team members to finish this in a few days.*
6. *And the last thing is testable. Whenever you're writing a user story you should be able to test that user story.*

And if you pay attention to these six things you will be able to come up with great user stories.

These sample user stories will help you grasp the concepts and put into perspective the lessons taught in the user story lectures

SAMPLE USER STORIES

User Story 1

As a **startup founder**,
I want to **be able to create an account on xlr8 website**,
so that **I can join the xlr8 community of investors, mentors and angel investors**

Acceptance Criteria

1. Able to create an account manually (filling out the sign-up form)
2. Able to create an account via Facebook
3. Able to create an account via Google
4. Able to create an account via LinkedIn

User Story 2

As a **startup founder**,
I want to **be able to log into the xlr8 website**,
so that **I can access the xlr8 content and community**

Acceptance Criteria

1. Able to log in with username and password
2. Log in via Facebook
3. Log in via Google
4. Log in via LinkedIn

Story Description or Notes:

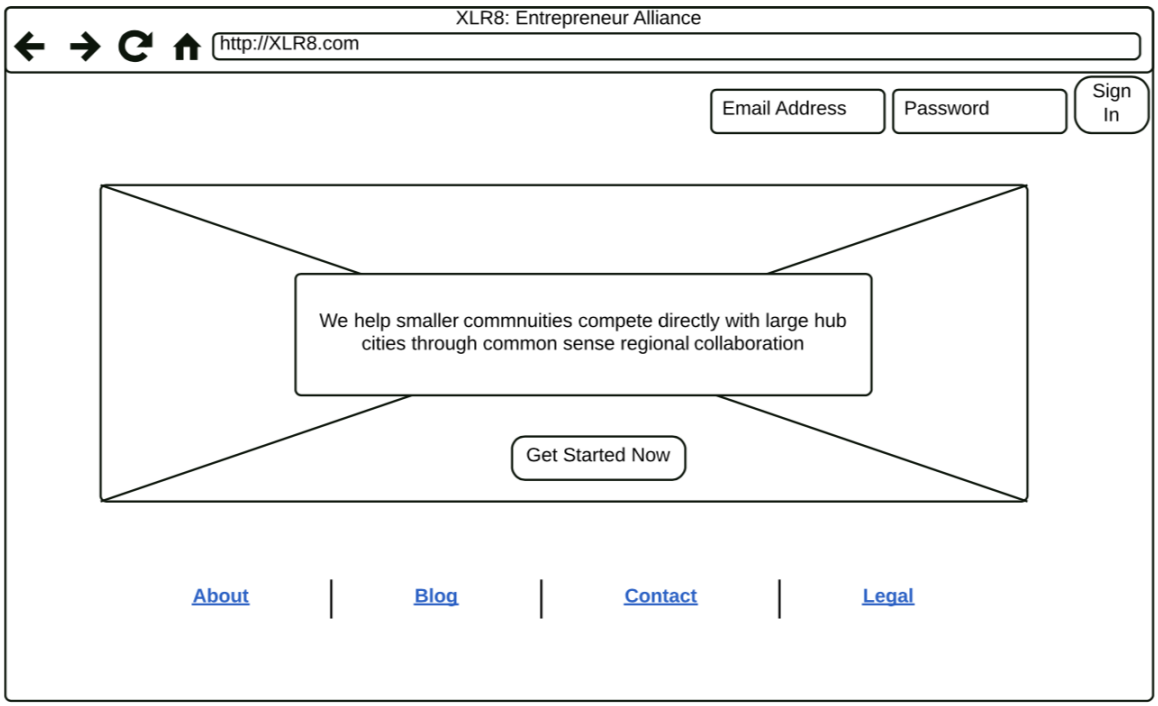
- Standard login Information (email address and password)
- Background Image of a bridge or something that ties in nicely with XLR8 branding
- Small text blurb that informs new users of the purpose behind XLR8
    - In the future - this text blurb will be accompanied by a demo video

User Story 3

As a **new user**,
I want to **be able to go to the landing page of the xlr8 website**,
so that **I can learn more about xlr8 and community**

Acceptance Criteria

1. Page should have an *About Us* tagline
2. *Get Started Now!* button that prompts new users to the Registration Form
3. Links at the bottom of the page that offer additional information on XLR8
    - About - a mission statement and some info on founders/partners
    - Blog - an attached blog that documents some of the cool things XLR8 does outside the scope of the web portal
    - Contact - Contact information for XLR8 headquarters/ Customer Service
    - Legal - Terms & Conditions

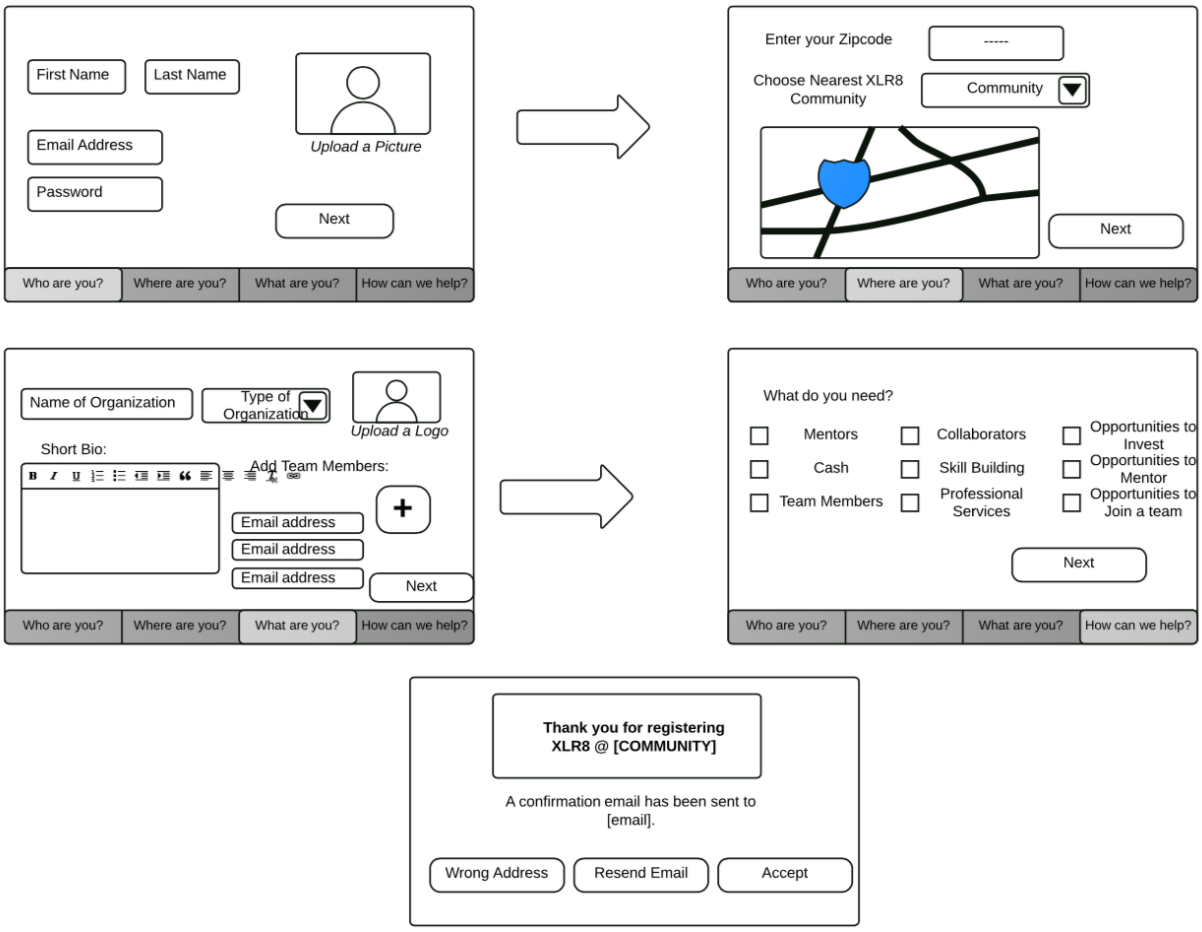


User Story 4

As a **potential xlr8 mentor or member**,
I can **fill out an application to become a mentor**
so that **I can help startup founders who have good business plan and product**

Acceptance Criteria

1. Application should be more than 4 form boxes – who are you, where are you, what are you, and how can you contribute.
2. Application questions should have some mandatory elements
3. Application process should be completed within 15 minutes

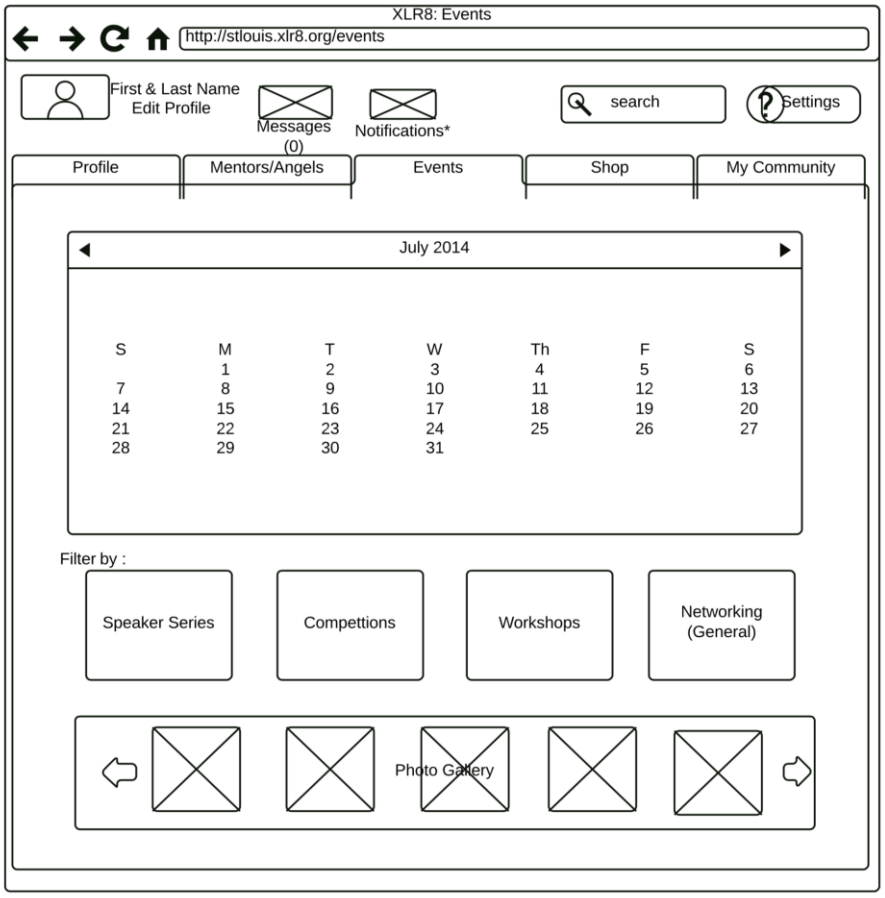


User Story 5

As a **site visitor**,
I want to **be able to view the monthly view of all the upcoming events in my area**,
so that **I can connect to other members and attend the events that can help me or my startup**

Acceptance Criteria

1. Event page should have a month view of the event calendar
2. Ability to filter events by speakers, competitions, workshops, and networking
3. The past event pictures should be in the sliding mode on the bottom of the events page.

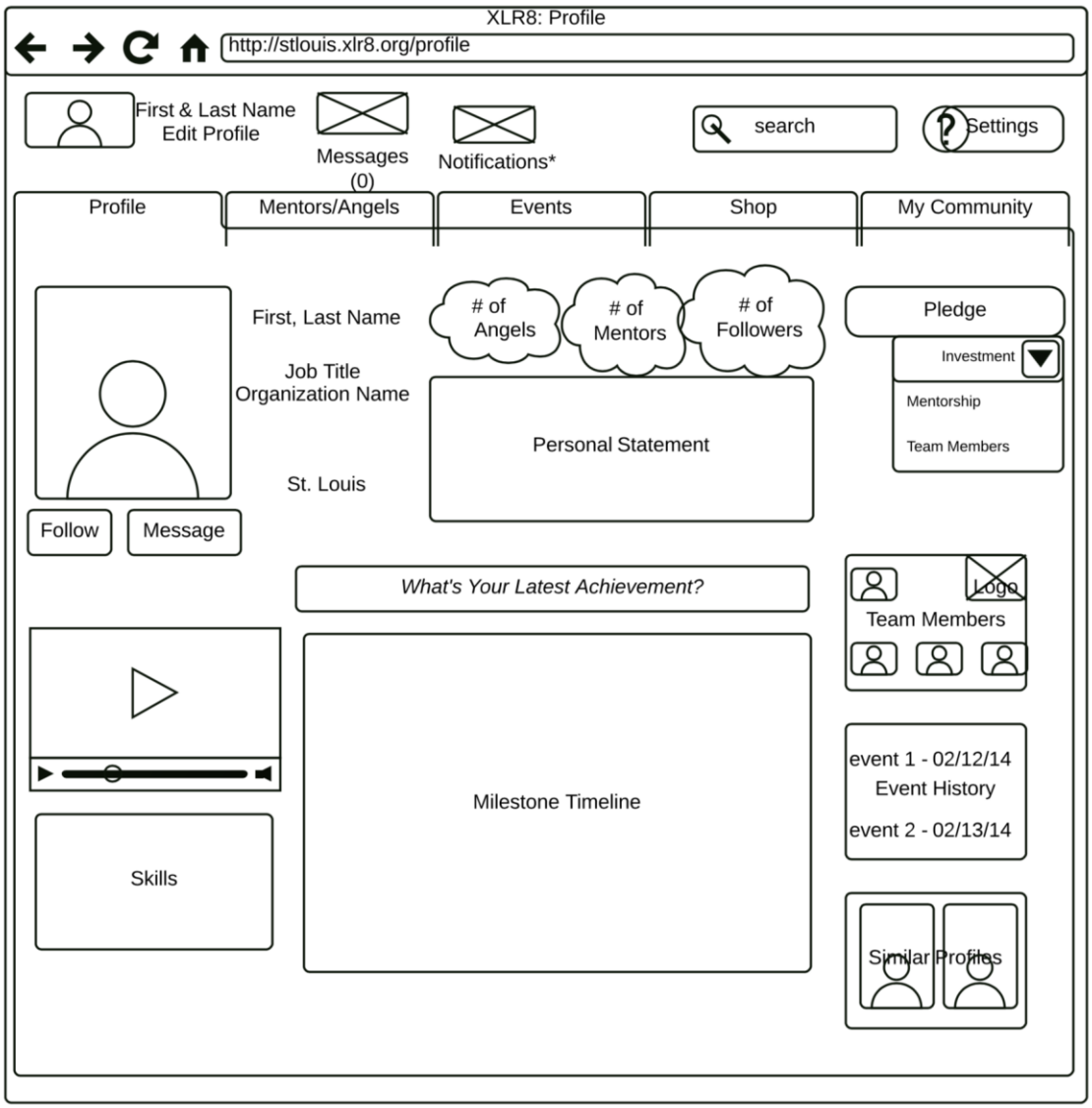


User Story 6

As an **XLR8 Mentor**,
I want **my profile page to include additional details about me**
so that **the startup founders can learn about me and decide if I'm the right mentor for them**.

Acceptance Criteria

1. Profile page should have my details (viewable on computer and mobile)
2. Mobile page should have a learn more button
3. Profile page should have a section to upload a picture



User Story 7

As a **website visitor**,
I want **to be able to view the site in google all major browsers**,
so that **I can use the browser that I have installed on my computer**.

Acceptance Criteria

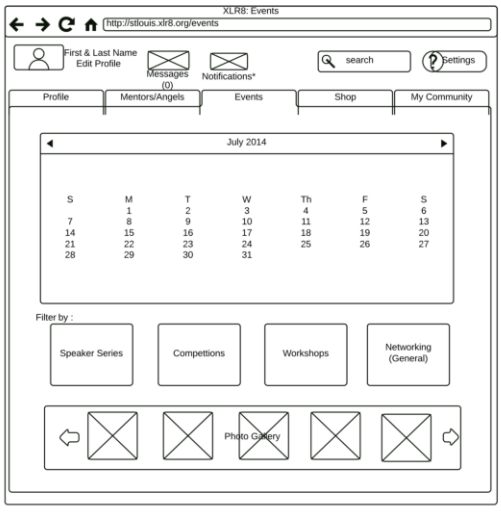
1. Users should be open the site in Google Chrome browser
2. Users should be open the site in Safari browser
3. Users should be open the site in Internet Explorer browser
4. Users should be open the site in Firefox Mozilla browser
5. Users should be open the site in Opera browser

User Story 8

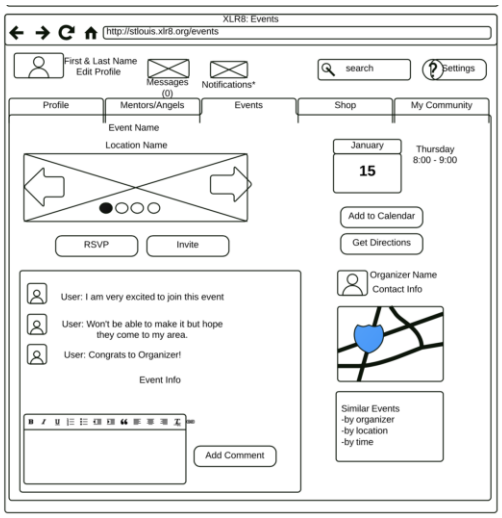
As an **event creator**,
I **want to be able to create an event**
So that **other users can see our events**

Acceptance Criteria:

1. Able to create an event
2. Able to enter event information
3. Event can be submitted for approval
4. System sets the status for event to pending
5. When events are approved, the system displays note “Thanks for entering an event, this is going to be awesome. Your event will post in 2 days”.



Event Page

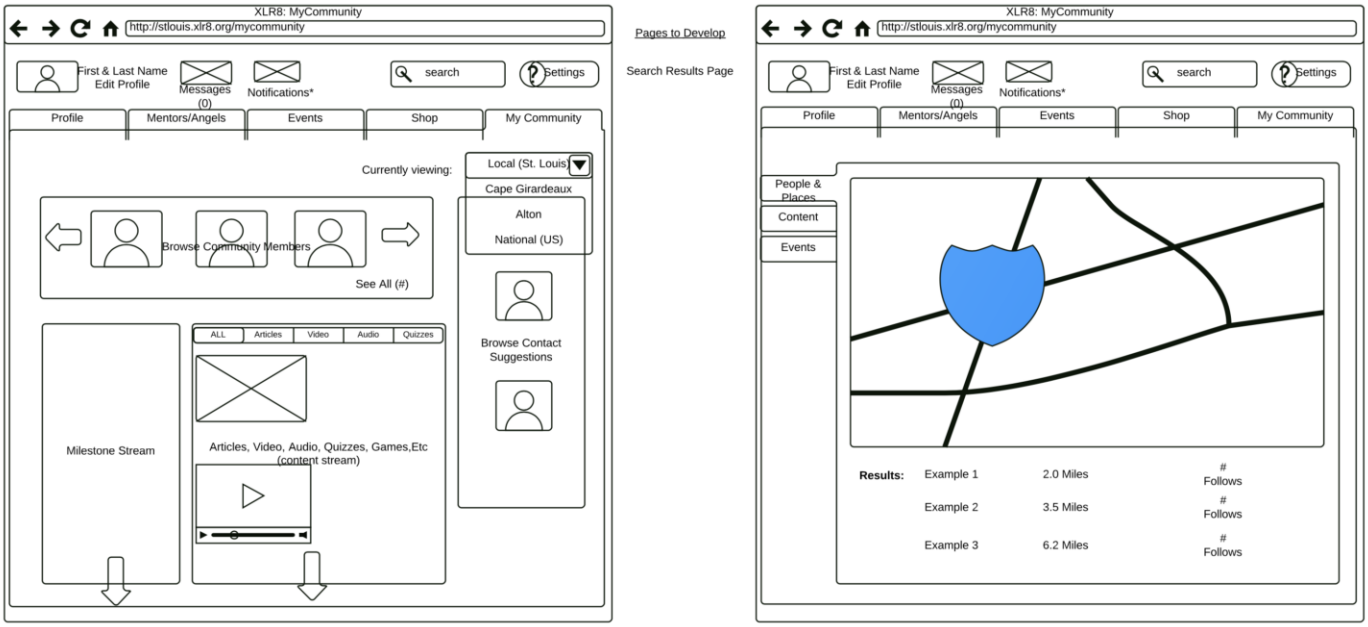


User Story 9

As a community manager (logged in),
I want **to see a "my community" tab in the website**
so that **I can easily access my community**

Acceptance Criteria

1. My community tab should be on the leftmost (last) side of the toolbar
2. My community tab should have sliding list of community members
3. My community tab should show what city is selected for the community
4. My community should have articles, videos, quizzes, and audio content

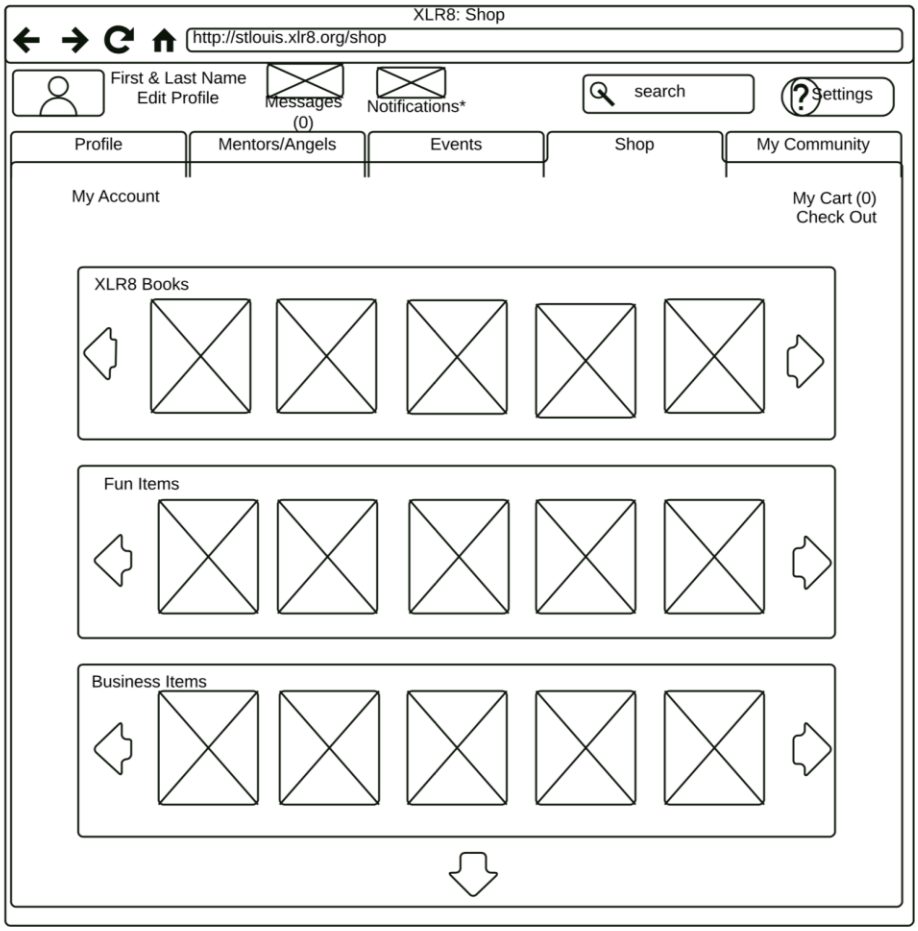


User Story 10

As a **community site admin**,
I want **a shop tab within the xlr8 website**
so that **I can show off our xlr8 community products**.

Acceptance Criteria

1. Shop tab on the community site should have sliding list of books
2. Shop tab on the community site should have sliding list of fun items
3. Shop tab on the community site should have sliding list of business items



User Story 11

As an xlr8 member,
I want a search bar to be available within the website,
so that I can quickly search for and find what I need on the website.

User Story 12

As a community manager,
I want to provide a site tour to pop up when new user logs in,
so that they can navigate the site better.

User Story 13

As a user,
I want to see a progress bar as I create an account,
so that I can see how close I am to finish the registration process.

User Story 14

As a community manager (logged in),
I want a community filter within the community tab,
so that I can filter what information I want to see on my community page.

User Story 15

As a community site member,
I want to the ability post in a forum,
so that I can post questions and discuss topics with the mentors and community managers.

User Story 16

As an xlr8 member,
I want to be able to view my shopping cart,
so that I can see my items that I plan to purchase

User Story 17

As an xlr8 user,
I want an event tab within the xlr8 page,
so that I can access events within the community.

Document provided courtesy of www.TheBAGuide.com
And www.TheAgileCoach.com

User Story 18

As a community member,
I want a toolbar to be present as I navigate the website,
so that I can easily access my account and other website content.

User Story 19

As a community administrator,
I want to get notification when an event is created,
so that I can approve the events.

User Story 20

As a xlr8 website visitor**,**
I want to be able to read Frequently Asked Questions (FAQs),
so that I can learn how to navigate the site and leverage the xlr8 community.

User Story 21

As a community manager/event creator,
I want to be able to edit my event,
so that I can keep the event details up-to-date and accurate.

User Story 22

As an xlr8 user (any user),
I want a profile tab,
so that I can easily access and edit my information from anywhere on the site.

User Story 23

As a xlr8 account member,
I want my payment options on the account page,
so that I can update my payment information.

User Story 24

As an event manager,
I want a create an event button in the event tab,
so that I can create an event.

Document provided courtesy of www.TheBAGuide.com
And www.TheAgileCoach.com

User Story 25

As a startup founder,
I want a notification page,
so that I can see my notifications regarding my account.

User Story 26

As an event manager,
I want an event tab,
so that I can see and view all my past and future events.

User Story 27

As an xlr8 user,
I want a mentors tab,
so that I can see all the profiles of those who have pledged themselves as mentors.

User Story 28

As an xlr8 member,
I want to view mentor of the month within the mentor tab,
so I can view the top-rated mentors.

User Story 29

As a potential user interesting in mentoring,
I want an option to sign up as a mentor,
so that I can be made available to the xlr8 user that has questions.

User Story 30

As an angel investor,
I want to be able to register within the xlr8 website,
so that I can learn more about startup companies that I want to invest in.

User Story 31

As a community member,
I can leave a comment about the articles or videos shared on the website,
so that I can share my input with the rest of the community.

Document provided courtesy of www.TheBAGuide.com
And www.TheAgileCoach.com

User Story 32

As a xlr8 site visitor,
I want the privacy policy available to me,
so that I read and understand the privacy policy.

User Story 33

As a site admin,
I can post information in a community manager only page,
so that only community managers can see the information.

User Story 34

As a site admin,
I can create content for the entrepreneurship and startups section,
so that there is initial content for the beginning users to get a reference for kind of the content we want on the site.

User Story 35

As a site visitor (not logged in),
I want to be able to read some of the xlr8 original articles,
so that I can learn more about the community before I decide to create an account in the site.

Product Backlog and Sprint Backlog

So let's look at the product backlog. ***The product backlog is a list of user stories collected by the product owner by talking to the users and stakeholders. And one of the most important thing about the product backlog is you prioritize, the highest priority items are on the top and the least priority are in the bottom. And this is an ever increasing list because the wishes and the needs of the user are infinite. Right. So it's up to the product owner. On the regular basis the product owner will update this and put the highest priority items on the top.***

And this is where the scrum team can pull the story from the top because they know they're doing the highest priority item. So the product backlog should reflect the product vision. What are they trying to build ? ***The product backlog should have a pretty good sequence on how the product should be built. Sure, all the needs of the business, the huge long laundry list of the needs, the wants, the shoulds, the coulds. And then you can imagine that the team then selects ones that are on the top.***

And so we kind of talked about the product backlog. So let's transition that into like a ***sprint backlog. Yes, that is the second artifact.*** So we the product owner's list of user stories that is called the product backlog, that the product owner

owns, ***In one of the scrum ceremonies which is sprint planning. This is what the product owner comes with, he comes with the product backlog's highest priority items and he says: these are the highest priority items.***

Scrum team, let's discuss these, let's just look at the user stories. How many of these can get done in this iteration or the sprint? And if it took two weeks iteration, the team is thinking okay, we have two weeks and we have to whatever story that we say yes to, we have to get this to done. You have to fully design it, build it, test it, get ready and it should work. And he has accepted criteria too. So for them to understand what's the scope of the user story,

at the end of the sprint they will demo this work. It's will almost, surely do that for the user base or the stakeholders. That is your sprint backlog. So those are the committed user stories that will be completed in that sprint based on what the team says. Yes, this is what we can do. And who owns the sprint backlog? That would be the team.

Working Agreement

This is outside of a scrum framework, but working agreement is used by a lot of team members and just like it sounds like, working agreement is when you work in a team, what are some of the rules. what's your expectations, what are the expectations. Those

are some of the things that need to be there, explicitly there, where everybody can get access to it, so we all know that this is a multiplayer game. ***There are some agreements there so everybody can play together and nobody is mad or nobody's unheard so we can make that very explicit for the team.*** Yeah, I think a lot of that came up because we're collaborating as a team.

There's no manager, project manager dictating 'hey, you have to get your stuff done, why don't you get your stuff done, hey, you have to get your stuff done'. It's not that, it's a collaboration, so everybody needs to say yes, I agree to what the team is saying and I can do my part to make that happen. The working agreement, you're going to find it in a lot of different organizations, it's not required by any means and it's not even part of scrum. But a lot of companies and a lot of organizations have picked that up to help them solve some of those problems before they become problems, truly get ahead of everything.

How to Create an Agile Working Agreement (with Sample)

A functional working agreement is critical for a self-organizing, productive agile team. This collaborative, living document helps teams build trust, establish flow, and easily resolve conflicts.

Here, we lay out how to create a working agreement and provide a sample working agreement that we created for our remote team.

What is a Working Agreement?

A working agreement in agile is a set of guidelines created by the team to establish expectations the team has for one another. The working agreement has many names, including the “Code of Conduct,” “Ways of Working,” “Team Norms,” and “Team Constitution.”

The Working Agreement serves as a reminder of agreed-upon norms for team members to collaborate and communicate. It is usually created when a new team is formed or when onboarding new team members.

Characteristics of an Effective Working Agreement

- **Public and visible.** The working agreement should always be visible to the team so that it can be referred to easily.

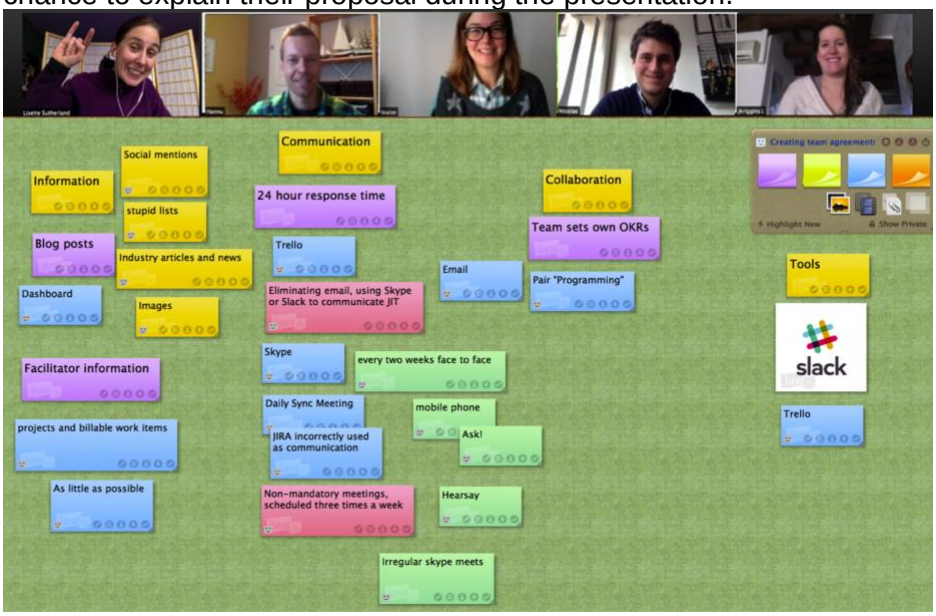
- **Malleable.** The working agreement should be reviewed frequently and can be updated as new cases come up. For instance, during COVID-19 when we had to adjust to a remote working arrangement, we modified our working agreement to accommodate new remote working rules.
- **Concise.** The number of items in the working agreement should be limited to only the most important so that the team can recall it easily.
- **Collaborative.** Each team member should participate in creating and accepting the items in the working agreement.
- **Enforced.** If an item in the working agreement is violated, the team should hold each other accountable and point it out.

Creating an Agile Working Agreement

The Working Agreement is formed during a meeting with the team, Scrum Master, and Product Owner. To get the team onboard with the working agreement and drive home its importance, it helps to break it into pre-set categories. For instance, when we were establishing our remote working agreement, we broke it down into the following categories: Communication, Sprint and Meeting Cadences, Timeboxes, Team Values, Artifacts, and Future topics to discuss.

At first, team members will write on post-it notes some ideas for what they think should go in the working agreement. The scrum master can present some prompts to help facilitate ideas, such as “How should we resolve conflicts?” and “What guidelines would help establish expectations for each other when there’s an approaching deadline?”

The team will have some time to write down their ideas. The scrum master should instruct the team to write proposals that are self-explanatory at first sight. Each team member will have a chance to explain their proposal during the presentation.



Source: Lisette Sutherland, Collaboration Superpowers

Next comes the voting and discussion. Each team member will take turns to present and explain one of their ideas. The rest of the team will have a chance to ask questions about the proposal.

The team will then vote all at the same time using the hand signals below:

Thumb up means: Yes, I like it AND I believe I will be capable of honouring it.

Thumb down means: No, I believe this would be bad for us as a team – or – No, I don't believe I'm capable of honouring this at all.

Thumb sideways means: Sure, if there are no thumbs down, I will accept this being in our Working Agreement. I would however like to share a thought or concern.

Closed fist means: This specific principle doesn't apply to me or my work, so I abstain. I will let the others decide. I'm okay with their decision and will respect it.

(from Crisp's Blog)

If there are no thumbs down for that proposal, the scrum master can move the post-it to the Working Agreement section of the board.

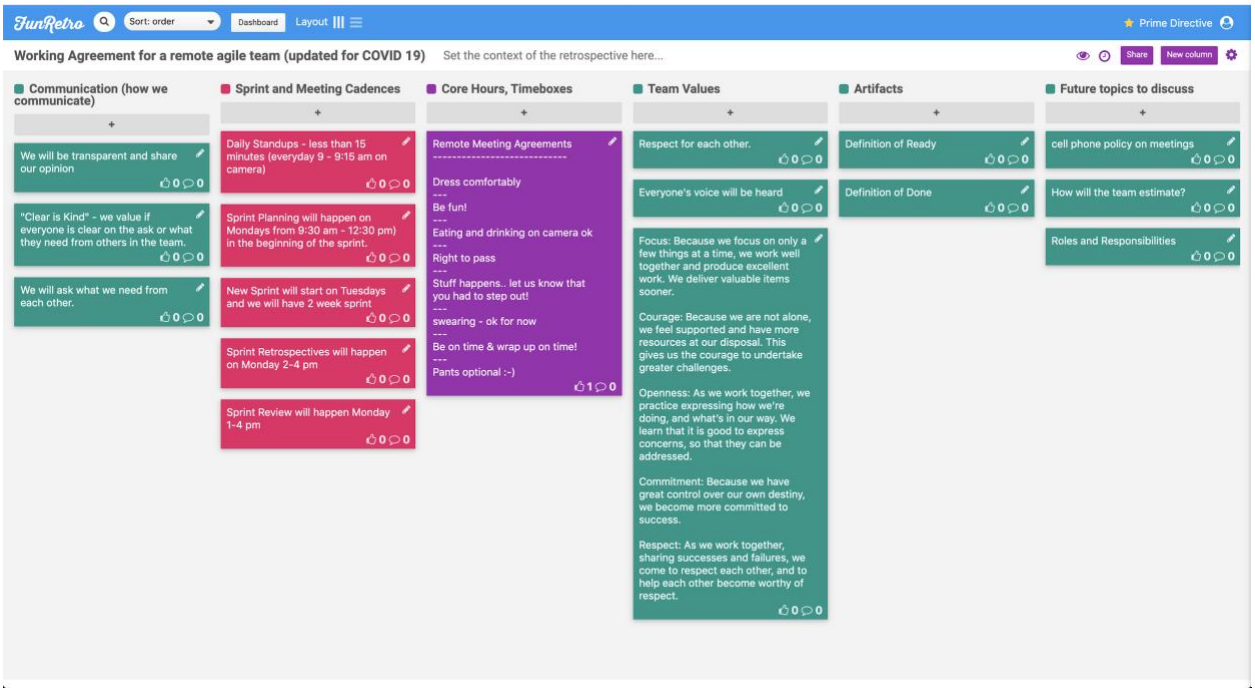
If there is a thumb down, then the team should have a discussion to see if there is a way to tweak the proposal so that everyone is happy with it. However, if the discussion goes on for longer than the timebox allows, the proposal should be put in the Maybe Later section to be discussed at a future Retrospective.

The team will continue to present, vote and discuss until the timebox is up. Once the working agreement is complete, the scrum master can read each of the items and prompt the team to establish a home for the working agreement, such as a physical or digital Kanban board.

Sample Working Agreement for a Remote Agile Team (Updated for COVID-19)

The beauty of agile is the ability to adapt to unexpected situations. That was the case with Coronavirus, which forced our team to work from home. Remote working brought up new scenarios that we had to accommodate into our working agreement, which is why we drafted up an amended Remote Working Agreement for COVID-19.

While most of the items like our team values and communication rules stayed the same, we had to make adjustments to the working agreement like core hours and timeboxes. Below is our FunRetro Kanban board with our special remote working agreement. Feel free to take a look and adopt any items for your own team's working agreement!



Another Sample Working Agreement

Working Agreement: Definition of ready

Definition of ready is one of the most important part of the working agreement. Setting up the definition of ready is Scrum Master's one of the crucial responsibilities as well. Usually, in teams, definition already applies to the user story.

Again, what is a user story ? A user story is a very simple brief statement, describing the feature or the work that has to be done by the developer from an end users perspective.

So how does a team member know when the user story is ready? Most of the time, what we see is a definition ready has, that the user story actually has who, what, and why. Who is the customer? Who is going to use it? all the end users ? What did they want ? and why do we need it ? What is the reason why we're working on this story? So those three things are crucial in a user story.

The other thing that's very crucial in a user story, is the ***acceptance criteria.*** So what are the criteria's for this user story to be done ? That needs to be there before a developer can pick up the story and start working at this. Overall, the definition already means the user story is actionable. The user story has what needs to be done and the amount of work and the complexity of the work. Obviously, having the acceptance criteria. So when you read the user story. It should be clear,

concise, and actionable.

Working Agreement: Definition of Done

Let's look at what you mean by definition of done. This is an agreement document that team creates so that it's accessible by everybody and it is a common understanding that everybody has at that point of time and is it ever evolving document. What we mean by that is when a team starts with a checklist of items, when these items in the checklist are done we know this story is done.

Let's look at a unit of work, take and user story. How do we know when we are done with the user story? One of the things that I've used in my team is starting with a very simple document. For one of the teams that I used to coach it was as simple as this, ***that user story meets the acceptance criteria provided by the product owner. So whatever product owner said, those conditions of acceptance, that piece of work actually includes those. The other thing is it's properly tested. The other checklist in the definition of done document is there is code review or peer review in that story or that piece of code.***

And once we have all that we need two more things we need a lightweight documentation on what was being done in that user story and the last thing is the story must be given to a product owner for review. And if we finish all those things

we will consider this story or piece of work is done.

Let's say for example, if there is a new team member who joins a team, it is so easy for the team member to look at the working agreement and part of that is definition of done and they can look at it and they can say, they can know exactly where the team's mindset and agreements are in place so they know what to do. Following the working agreement they would complete the story, they will make sure the story has all the acceptance criteria in place, they will test those stories or they will actually give it to another team member.

There will be code review. There will be a lightweight update on conference or Jira on how the story was implemented and the last thing is they will pass that story to the product owner and explain the product owner here this story is done. And when a new team member agrees to this definition done then everybody is in the same page and it can really help team gain productivity versus getting into the argument: oh, this is not actually done or this is almost done and I just need to implement this, or I just need to test it in this browser. Those things can actually take a lot of time and you know there's a lot of hand-offs that can happen within the user story whenever you don't have an explicit definition of done.

One of the things that you want to make sure is you want to make this document visual and you want to put in a place where everybody can access it. For example I had this document in a team room, in a poster and with some of the developers actually printed this definition of done and actually posted in their cubicle so that it almost reminds them what our definition of done means.

Product Increment

According to the scrum guide, a product increment is a potentially shippable, vertical slice of a solution that's created as part of a sprint time-box.

That definition has a lot of terms and it can be very confusing. So what I want to do is actually use an illustration to explain what a product increment is.

Let's first start with your traditional methodology, such as a waterfall methodology. On the screen you'll see a little chart from left to right is time. So as time goes on we move towards the right. The way traditional methodologies have been broken up, is you have your various stages. So you have your analysis stage, once you're complete with your analysis, you're done with your analysis, you move on to your requirements stage, then your design stage and on throughout the project. You don't go back to a previous stage, you continue moving forward.

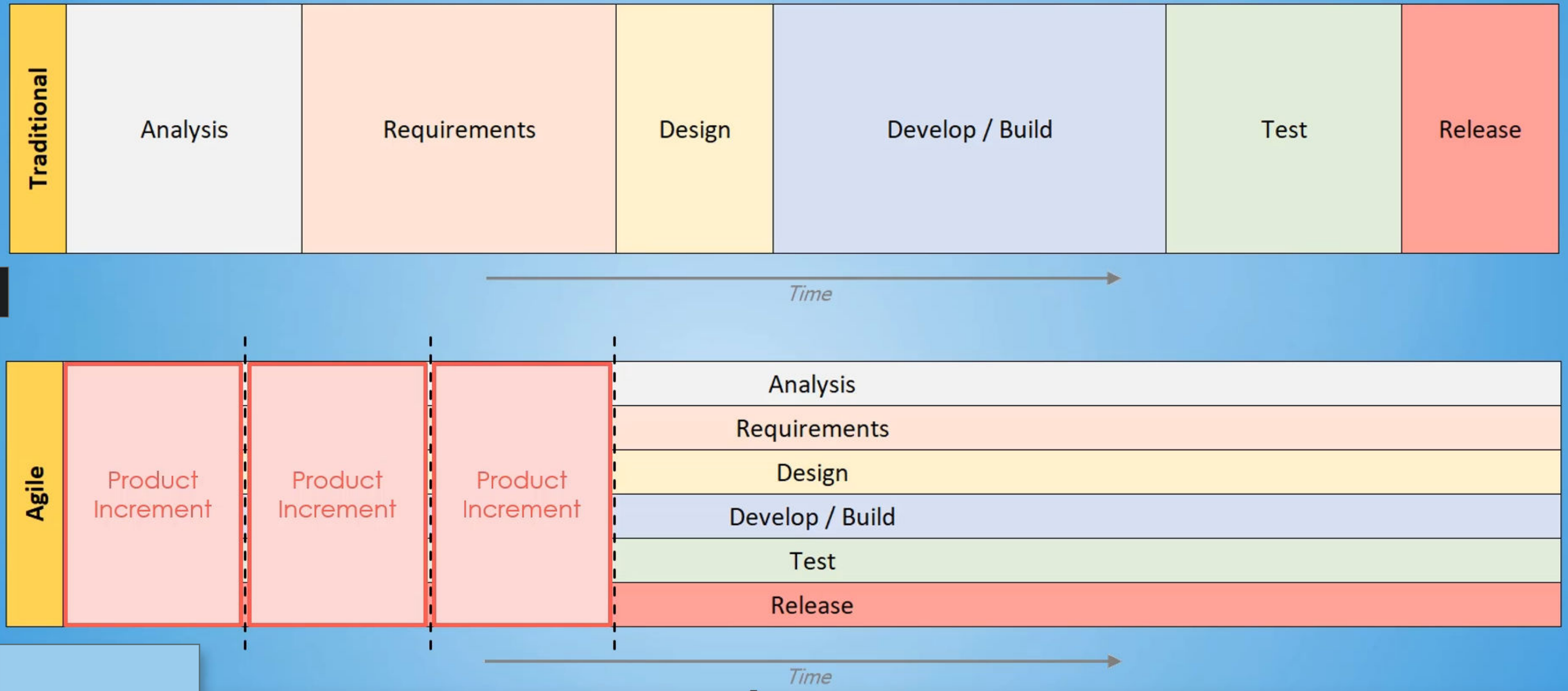
The challenge with this is you don't actually release anything of use or value to the business until the end of the project, until that full cycle is done. So let's say pretty early on in the project, the business wants to stop the project or once again some value from it. I'm sorry you can't. We've completed our analysis and started our requirements, but there's really nothing to deliver to you to help you out. So that is a traditional methodology.

Now let's take a look at an agile methodology. It takes these stages and it still uses them, but it flips them on their head. So instead of going from stage to stage to stage throughout time, the stages are listed vertically. So when you now have a slice and you do a vertical slice, that's where that term comes from, so you do a vertical slice on that product now you have this product increment. So at a certain point of time which is a sprint or a time-box or within that 1 to 4 week time frame, you are creating a potentially shippable product increment. Now you do that over and over on every single sprint and you're creating these product increments.

To explain this with another type of example, everybody's had some type of application software either on their phone or on some type of desktop computer that once every couple weeks, once a month that you get some type of update it's now version 1.4.6 and then it's 1.4.7 little bit later and a couple of weeks, 1.4.8, what they're doing is

they're shipping new product increments the shipping to new features and changes and updates that they've made to the application. And they're doing that over and over and over. So you can start to realize that value and that product increments it's only going to contain a couple of features, a couple of new things, a couple of changes, but when they do that over and over and over and they continue to deliver that to you, you're able to continue to realize that value rather than waiting for all of their features as part of this project to be complete.

So product increment is a vertical slice of a potentially shippable product that's developed as part of a sprint time-box.


Traditional
Analysis
Requirements
Design
Develop / Build
Test
Release
Time
Agile
Product Increment
Product Increment
Product Increment
Analysis
Requirements
Design
Develop / Build
Test
Release
Time

Estimating in agile

Introduction to Estimating

So estimating, is the concept of roughly calculating. In this particular case, we're roughly calculating how long it's going to take us to complete a particular user story, how long it's gonna take for us to get from beginning to done done. Now that roughly calculating thing ? I don't know, calculating to me sounds a lot like you have all the information and you're just kind of putting it together or figuring things out- like a math problem. But to me, estimation is nothing like a calculation, it's more of a guess, but it's not just a wild guess. It's actually an educated guess.

You're using some information to help you come up with your best educated guess on how long this is going to take. Now the information you use, there's really three key pieces to it.

1. Number one- which is the most important- is the elicited information. So this is the information you've gathered from stakeholders. This is their various needs. This is their wants. This is ultimately what put together the core of that particular user story. So that obviously based on what they tell you there will dictate how long it's going to take you to actually solve that with some type of solution.

2. Secondly, you want to utilize your past experiences. So this could be experiences on other projects or other user stories that are similar to this- where now you can utilize the lessons learned from that past user story- to actually help you create your best estimate for this user story.
3. And finally, you can also use various documentation from your organization. So your organization may have some standards documentation for estimating. So some organizations are very mature in how they do it. Some organizations have some repetitive tasks and so they have some various documents that will help their team members to put together an accurate, or as accurate as possible, estimate.

So now that you understand what estimating is in general let's move on and talk about estimating specifically for Agile.

Estimating in Agile

Where I want to start is actually not looking at agile, I want to talk about estimating for non agile projects and these will help to give you some context prior to jumping into the agile side. So for non agile projects, is this more waterfall or iterative projects, you are going to get absolute estimates.

What I mean by that is you've completed requirements and you've handed that those requirements for that particular item or task or feature over to a developer. And now that developer's are going to developer, I guess I should say developer or solution or because it may not be like a software developer. But anyways you handed over to the person that's going to be completing that particular feature. And they are going to read through everything and from a non agile standpoint they're are going to deliver back a number. So this is going to usually signify the number of days for the duration of that particular feature. I feel this particular feature is going to take me three days to complete. That would be an absolute estimate, you're giving a set number of days, in this case, for how long it's going to take to complete that particular feature.

Agile is a little bit different. Agile looks at things relative to the other features, items, tasks that were estimated as part of that project. So agile looks at it and says: ***Is this a small or is this a medium or is this a large item? We don't care about specifically how long we feel it's going to take us.*** We just need to know if it's small, medium or large, what bucket does it go into. And now when we say a relative estimate the reason they say that is it's relative to the other items that have already been estimated as part of that project.

So let's break that down a little further with an example. On the screen you'll see a fly, a frog and an elephant. The fly is small, the frog is medium and the elephant is large. So as we're working through if we were trying to categorize or bucket a whole bunch of animals we would use these as our relatives sizing, if we find a rhinoceros is going to be a large is going to go with that elephant. If we find a butterfly, maybe that goes with the frog because it's the same size of frog or maybe it's a really small butterfly and goes with the fly. ***So you can start to do the sizing or the estimating based on relativity to the other things that were already estimated.***

Now, the thing to recognize with the relative estimate is that relativity is really only inside of that project. So those team members are estimating tasks based on other tasks or features that were already estimated as part of that project. And so they're going to look to try to put it in that small, medium or large bucket. But those buckets don't necessarily apply to every other project. Let's look at another one. So here we've got a ladybug, an elephant and a tree. The Ladybug here represents the small, the elephant is now a medium, in this particular project ,and the tree is a large. So as you can see even though that elephant was a large estimated item as part of the previous project, that exact same item or a

requirement or feature could be estimated as a medium as part of a different project. It's just, it's all relative to the other items, the other tasks, the other features that are being estimated out. So it's a really important concept to grasp.

So I hope that makes sense in regards to a relative estimate from an agile standpoint, to an absolute estimate in not agile. Personally from an estimation standpoint agile is so much easier so much easier to do a relative estimate than it is to do an absolute estimate for a couple of reasons. One is you can just kind of place it in a bucket comparing it to other things rather than detailing out in your mind every step that you need to complete to then come up with an absolute estimate. So that's one big thing and the other big thing is the third bullet points that I have listed here. In agile updates usually don't affect a whole lot in the project, they usually have a pretty minor effect. And what I mean by updates is an estimation update. So let's say we put it into a small bucket and it ends up turning into a medium sized feature. OK, well it could hurt us in our sprint a little bit.

We could maybe not get everything done that we wanted to get done, but it's ok, we can bump another thing out to the next sprint. From a non agile standpoint it can have pretty dire consequences for a project. What I mean by that is let's say it was

estimated at two days and then it actually took four days to complete, that two extra days has to come from somewhere and I guarantee on the plan there were other features and items and tasks that were lined up to be completed after that. Well now that that first one got pushed back has to cascade across all of those other tasks, moving them back or trying to find a way to make up that additional two days that was unaccounted for because of the incorrect or invalid estimate.

Estimating in Agile: Why Estimate?

Let's talk a little bit about why it's important.

1. Number one and this is the most obvious one is an estimate is providing you an estimated duration for completion. About how long do you think it's going to take you to complete this feature based on the requirements you were given? From an agile standpoint, they'll put it into a bucket in the examples we gave earlier small, medium, large. But that could be different buckets based on various techniques, . But that's the most obvious one, you are giving them the requirement. They're telling us an estimated duration. Now we can use that for our planning and moving forward on that particular feature or deciding if, you know what, let's go ahead and table that, it's going to take a little bit longer than what we're expecting.

2. The second thing that estimating does for you is it helps to drive out clarification questions. When you're asking somebody to commit something. You're asking them to commit an estimate for a particular task they're going to spend a little bit extra time making sure that they read the feature, they read all those requirements, they read the acceptance criteria and they have a good understanding of how they would go to accomplish this particular feature. Now with doing that, they're going to also drive out additional clarification questions or unknowns that weren't defined there. So by asking for an estimate, not only are you going to potentially get a duration back, you also are going to get clarifying questions of things that you need to dig in further to and make sure that you have defined as part of the requirement prior to that requirement actually being able to be completed by that developer or the person creating that solution. So that's number two it helps to drive out clarifying questions.
3. Number three is it highlights complex or high risk tasks. As you're working in projects more and more you get a sense of how long things take. When you're writing your user stories or features. You have an idea. This is a small, medium or large type of task. Now that's going to be really useful when you hand that to somebody that's going to be creating it and

then they give you an estimate back and they say OK, I think this is a this is a medium requirement or medium feature you may say wait a minute I was thinking it was a small. So let's talk through a little bit. What are some of the complexities? What are some of the risk items that made you consider it a medium? So it helps to bring conversation and highlight some of those complexities and risky issues that you may not have been aware of or accounted for. Now by highlighting and by knowing that information you can make adjustments, maybe you tweak the requirements a little bit, maybe tweak that feature to make it less complex by taking very little away from it or changing it very little. Maybe you look to mitigate or remove those protect potential risks that were identified as part of that feature. And so then it can be reduced in size. Ultimately, the goal isn't to reduce the size, but it's to make sure that it's in line with expectations and with what would want to be done for the value that feature provides.

So those are the big reasons why we estimate in agile. Provides that estimated duration, drives out additional clarification questions and can help to highlight or point out areas of complexity or high risk that were previously unidentified accounted for.

Common Agile estimation techniques

Big/Uncertain/Small also known as the BUS, we have T-shirt sizes and we have the Fibonacci sequence.

Big/Uncertain/Small also known as the BUS

We're going to start with the most simple one the BUS. The Big/Uncertain/Small estimation technique has three buckets. You have a bucket for big user stories, you have a bucket for uncertain user stories and you have a bucket for small user stories. All of the features, all the items that are being estimated get placed into one of those three buckets. Ideally, you want as many features in the small bucket as possible.

Those features are small and bite size and can be done in a relatively short amount of time. Now with any of the stories that are in the big bucket, the first thing you should try to do is break them up. Is there a way that we can take that existing user story and break it up into multiple user stories, ultimately, parse out that user story and then have those be estimated at small that way their bite size they're easy to complete and provide value. Not all big user stories can be broken up and still maintain the same amount of value delivered to the stakeholders or the users in this case.

So some of them will stay big. But you do want to

make an attempt to try to break up as many of those as possible. Any stories that are part of the uncertain bucket that means that the requirements, the user stories are too vague, maybe the concepts behind it are bleeding edge and so it's really difficult to estimate how long it's going to take. These ones most definitely need to be broken up, defined and redefined and then reestimated to hopefully get them into either the big or the small buckets. So BUS is a fairly simple agile estimation technique. The most often I see this utilized by companies that are just switching to agile from more predictive methodologies such as waterfall.

T-shirt sizes.

This one has multiple buckets, you actually usually have five buckets as part of this you have an extra small, a small, a medium, a large and an extra large. So those are the five buckets that the features can be estimated into and they go along with relative t-shirt sizes. The reason I think T-shirts are used is because it's very global. Everybody understands what a size of a shirt is regardless of what language they're in, what country they work for or what project methodology they've used in the past. And so this one has also become a fairly popular estimation technique for organizations switching to agile.

But some of them just stay on these T-shirts

sizes, they don't need it more complex than that. So each user story obviously is compared to each other based on how long it's going to take, it's placed into the one of five buckets extra small, small, medium, large and extra large.

The large and extra large user stories are the ones that you should look at and see if you can break them up if possible into smaller user stories, ones that would fit into that extra small, small or medium sized buckets. It's not always going to be possible, but highly recommended that you don't have many if any large or extra large stories as you move forward to begin some of your sprint planning. So that's the T-shirt sizes estimation technique.

Fibonacci sequence

The Fibonacci sequence has the team grouping together their user stories into buckets. They're determining the groups by pulling together the user stories of approximately the same size of effort to complete. Okay, so we have these buckets. Let's call them A through G. So we have Bucket A, Bucket B, Bucket C, D, E, F, and G. The "A" bucket represents the smallest effort user stories. The "G" bucket represents the largest effort user stories. So the team grabs a user story, analyzes and discusses the details, and then determines the appropriate bucket it would fit into. But with our current naming convention, it's

hard to know the real difference between a lot of those middle buckets.

Like what's the real difference between Bucket C and D? Or a buckets D and E? It's hard. So what we'll do instead, is use a sequence of numbers, called the Fibonacci sequence, to name the buckets. So using the Fibonacci sequence, the first bucket is going to be 1. The second bucket will be 2. Now, the way the Fibonacci sequence works is you add together the two previous numbers to get the next one. So with that, our first bucket is 1, second bucket is 2, 1 + 2 is 3, so the third bucket is 3. As we continue on following the same pattern, 2 and 3 makes 5. So the next bucket is 5, 3 and 5 is 8, 5 and 8 is 13, and 8 and 13 is 21. So now our bucket groupings are following that Fibonacci sequence. With the bucket being labeled with numbers, it's naturally easier for So the next bucket is 5, 3 and 5 is 8, 5 and 8 is 13, and 8 and 13 is 21. So now our bucket groupings are following that Fibonacci sequence. With the bucket being labeled with numbers, it's naturally easier for the team to group those user stories by similar size, and be able to tell apart those that were estimated at a medium -ish size. Well, I hope that helps to explain it.

So the sequence goes like this. 1, 2, 3, 5, 8, 13, 21 etc.. So those numbers it may seem a little random at first, but the sequence says add the two

numbers prior to the number, and you'll get that number. So 1and 2 is 3, 2 and 5, 2 and 3 is 5, 3 and 5 is 8, 5 and 8 is 13, 13 and 8 is 21. So if we would go to the next one, it'll be 13 and 21 is 34. So as it... as you get larger, the numbers gets larger exponentially a lot faster. Right? And you... you add up a lot quicker. And the concept around that is, the bigger the story, the more complexities, the more uncertainties, and the more risks, that are [going to] be around in completing that estimate in that... the kind of estimated or relative time frame that's being expected. So, the numbers, like I mentioned for the Fibonacci sequence, are kind of abstract, but they're used to help provide some more details to the team as to what they can commit to for that Sprint. So these numbers are called story points. It's how many points that story will been... will need, in order to be completed.

As in a lot of the other techniques, any user stories that are larger, really should be broken down. In this case, 13 and 21, for sure should be broken down into smaller user stories that can be estimated and put into smaller buckets. Take less time to complete to deliver that value. As well, a lot of times the 8, is kind of looked at in and determined, hey can this be broken up at all? Because the smaller the stories are, and the smaller story points are assigned, the easier that

task is going to be completed, and the more likely you're going to be able to get it done within the estimated timeframe that you're expecting.

Other Agile Roles

Project Sponsor

Their main objective is to approve that project and the budget. They're the one that's championing this whole thing. They're the one that has the idea of the big picture and they are saying that this solution is going to provide value to the business or the organization. And we definitely want to invest our time and money to move forward and achieve that solution. With that, the Project Sponsors are helping to shape that project need and the outcomes.

1. Approves project and budget.
2. Provides the big picture view.
3. Helps shape the project needs and outcome.
4. Ensures project stays aligned to company ojectives.
5. Participates in Sprint Reviews(and other feedback demo).
6. Recognizes the team for quality work
7. Encourages corss-team/ cross-department collaboration.
8. Provides perspective on product impact and use.

Business Management or Business Executives

This is the Business Management or Business Executives, within the organization. One of their responsibilities, that's extremely important is just making sure they have an open mind. And they're

enabling access to their various business resources. Whether that be their management team, their supervisors, their subject matter experts, their users or other business resources that are necessary to help make sure that, that solution meets those business needs.

1. Enables access to necessary business resources.
2. Provides the needs of management.
3. Ensures user story results are creating value for the business.
4. Escalates resolution of identified impediments(as needed).
5. Participates in Sprint Reviews.
6. Encourages corss-team/ cross-department collaboration.
7. Acknowledges team successes and provides support.

Technology Leaders

These are the various IT managers, directors or executives of technology. The number one thing, that technology leaders need to make sure they're doing. If they don't do anything else. This is number one and that is hiring the right team members with the right skill-sets in Agile. Every team member has to be able to produce, on their own. They need to be able to solve little problems that come up, hiccups that come up. And they need to be able to collaborate and work with other team members, in order to help the team create that solution and solve that business

problem.

1. Hire Team members with right skillset.
2. Gives up control - empowers, trusts, and support team.
3. Enables access to necessary technology resources.
4. Escalates resolution of identified impediments(as needed).
5. Participates in Sprint Reviews.
6. Encourages corss-team/ cross-department collaboration.
7. Acknowledges team successes and resolves team conflicts.
8. Ensures solution is aligned with organizational standards.

Agile Detractors - Leadership

These are things that negatively affect the team, as well as, the Agile project.

1. Demands mid-sprint changes.
2. Hijiacks ceremonies to dicuss 'high-priority' topics.
3. Not available for demos . Then requests changes at the sprint Reviews.
4. Discourages communication with other teams.
5. Removes or changes available resources min-project.
6. Doesn't value agile mindset.

Subject Matter Expert / Senior User

These are the people on the business side, that have

been working in that industry, working in that business for a while. They get it. They know the technology, and or they know the processes, and or they know the industry and the business. And so here are the responsibility of those folks.

1. Provides thier needs and the needs of other users.
2. Hijiacks ceremonies to dicuss 'high-priority' topics.
3. Participates in Sprint Reviews(and other feedback demos).
4. Gains and communicates feedback on product deliverables
5. Provides perspective and context for requirements.
6. Completes requested testing in a timely manner.
7. Ensures user story results are used properly

Business Users

Similar to the Senior Users, the Business Users are helping to provide their needs. They're helping to give context and perspective, as to what they need and what they want to have within the solution. If those Business Users are invited to the feedback demos, go! It's very important if the Agile team is inviting various Business Users to feedback demos. It's for a reason that they have a unique perspective or unique use of that product deliverable. And it's important that they attend and provide their

feedback. Obviously, with the Business User, you're going to attend, watch or read training on the deliverables.

1. Provides their needs (differentiates wants and needs).
2. Provides perspective and context for requirements.
3. Attends feedback demo they are invited to.
4. Attends, watches, or reads training on deliverables
5. Utilizes deliverable and gives feedback.
6. Completes requested testing in a timely manner

Agile Coach

The Agile coach is an experienced Agile practitioner. This could be a consultant, it could be somebody that works within the organization. They're there to help train, guide and support the full Agile for the organization. So they're experienced in the Agile ways. They understand the ceremonies. They understand the terms. They understand the roles and responsibilities. And so they're helping everyone, all those various roles know, what their roles and responsibilities are within the various aspects of the project. But beyond that, they're also ensuring that the ceremonies that are being done, are done correctly according to the guidelines that the Agile values. And principles are being upheld within the organization.

So a lot of times, an Agile coach is brought in. Obviously, as companies are transitioning to Agile. But even after that, a lot of them hang onto an Agile coach. Just to really sit back, see the full picture and help guide the organization forward with Agile. And make sure that everybody knows, what they're doing and point out any type of issues or changes that need to be made, for the organization to be better from an Agile perspective or just better from an overall perspective.

Post Script

Now, one point of clarification, is we've talked about and taught you about these five other Agile roles. The Project Sponsor, the Business Leaders, Technology Leaders, Senior Users, and Business Users. And we've taught you these, based on kind of a Scrum perspective. We taught you about Scrum and we've tied this into Scrum. But I wanted to make mention that these roles are also the same thing for Kanban, for Scrumban, and other Agile frameworks. These roles are still helping out move the Agile team forward. That Project Sponsor, while they may not be working with a person titled Product Owner. They're working to approve the budget and

make sure that the needs and objectives of the organization are being met with this particular project. Same with the Business Leaders. So the roles and responsibilities that we've taught throughout this section, can be applied regardless of the framework. We just applied it to more of a Scrum looking framework. Because that's what's familiar to you. Because that's what's been taught at this point in the course. So I hope that helps to make sense. That those five additional roles will play a part, regardless of your Scrum or another Agile framework.

Working Agreement: Definition of ready

Definition of ready is one of the most important part of the working agreement. Setting up the definition of ready is Scrum Master's one of the crucial responsibilities as well. Usually, in teams, definition already applies to the user story.

Again, what is a user story ? A user story is a very simple brief statement, describing the feature or the work that has to be done by the developer from an end users perspective.

So how does a team member know when the user story is ready? Most of the time, what we see is a definition ready has, that the user story actually has who, what, and why. Who is the customer? Who is going to use it? all the end users ? What did they want ? and why do we need it ? What is the reason why we're working on this story? So those three things are crucial in a user story.

The other thing that's very crucial in a user story, is the ***acceptance criteria.*** So what are the criteria's for this user story to be done ? That needs to be there before a developer can pick up the story and start working at this. Overall, the definition already means the user story is actionable. The user story has what needs to be done and the amount of work and the complexity of the work. Obviously, having the acceptance criteria. So when you read the user story. It should be clear,

concise, and actionable.

Working Agreement: Definition of Done

Let's look at what you mean by definition of done. This is an agreement document that team creates so that it's accessible by everybody and it is a common understanding that everybody has at that point of time and is it ever evolving document. What we mean by that is when a team starts with a checklist of items, when these items in the checklist are done we know this story is done.

Let's look at a unit of work, take and user story. How do we know when we are done with the user story? One of the things that I've used in my team is starting with a very simple document. For one of the teams that I used to coach it was as simple as this, ***that user story meets the acceptance criteria provided by the product owner. So whatever product owner said, those conditions of acceptance, that piece of work actually includes those. The other thing is it's properly tested. The other checklist in the definition of done document is there is code review or peer review in that story or that piece of code.***

And once we have all that we need two more things we need a lightweight documentation on what was being done in that user story and the last thing is the story must be given to a product owner for review. And if we finish all those things

we will consider this story or piece of work is done.

Let's say for example, if there is a new team member who joins a team, it is so easy for the team member to look at the working agreement and part of that is definition of done and they can look at it and they can say, they can know exactly where the team's mindset and agreements are in place so they know what to do. Following the working agreement they would complete the story, they will make sure the story has all the acceptance criteria in place, they will test those stories or they will actually give it to another team member.

There will be code review. There will be a lightweight update on conference or Jira on how the story was implemented and the last thing is they will pass that story to the product owner and explain the product owner here this story is done. And when a new team member agrees to this definition done then everybody is in the same page and it can really help team gain productivity versus getting into the argument: oh, this is not actually done or this is almost done and I just need to implement this, or I just need to test it in this browser. Those things can actually take a lot of time and you know there's a lot of hand-offs that can happen within the user story whenever you don't have an explicit definition of done.

One of the things that you want to make sure is you want to make this document visual and you want to put in a place where everybody can access it. For example I had this document in a team room, in a poster and with some of the developers actually printed this definition of done and actually posted in their cubicle so that it almost reminds them what our definition of done means.

Product Increment

According to the scrum guide, a product increment is a potentially shippable, vertical slice of a solution that's created as part of a sprint time-box.

That definition has a lot of terms and it can be very confusing. So what I want to do is actually use an illustration to explain what a product increment is.

Let's first start with your traditional methodology, such as a waterfall methodology. On the screen you'll see a little chart from left to right is time. So as time goes on we move towards the right. The way traditional methodologies have been broken up, is you have your various stages. So you have your analysis stage, once you're complete with your analysis, you're done with your analysis, you move on to your requirements stage, then your design stage and on throughout the project. You don't go back to a previous stage, you continue moving forward.

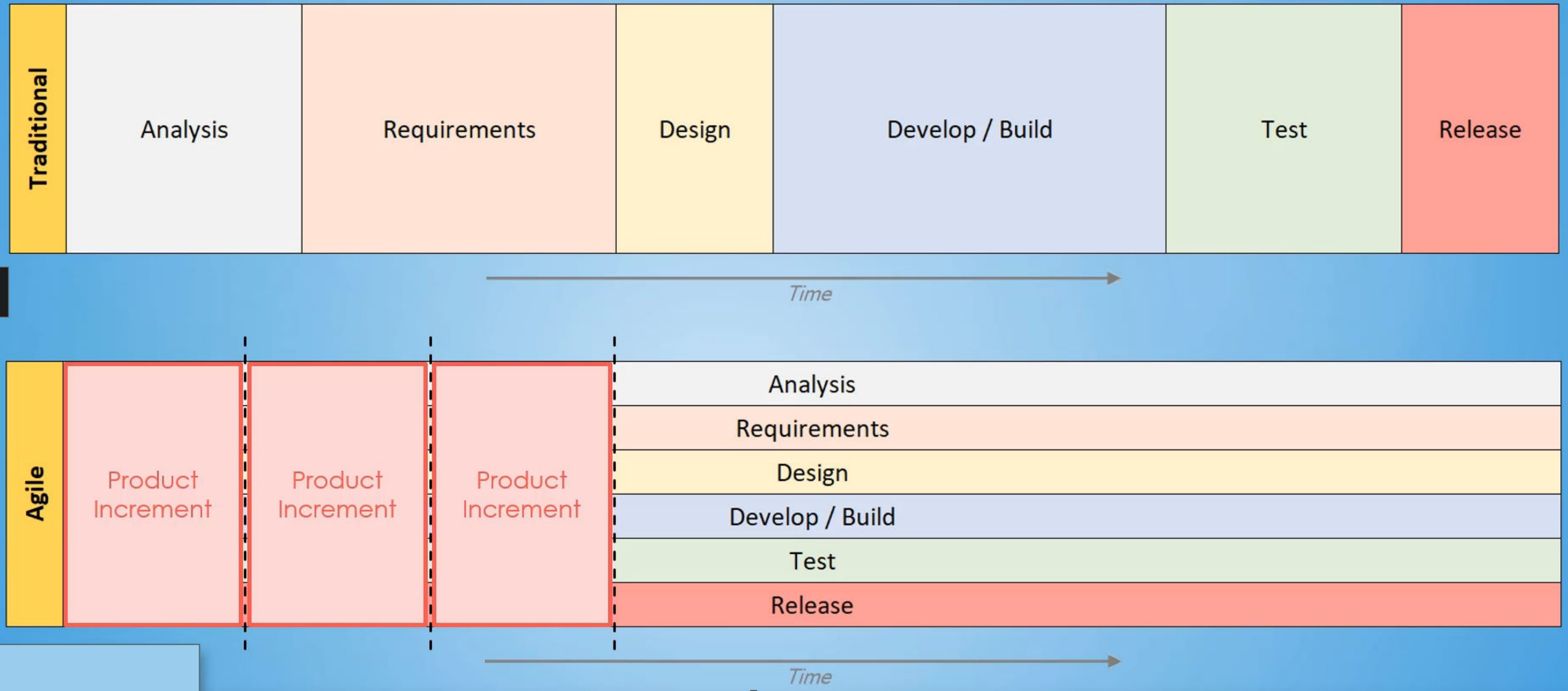
The challenge with this is you don't actually release anything of use or value to the business until the end of the project, until that full cycle is done. So let's say pretty early on in the project, the business wants to stop the project or once again some value from it. I'm sorry you can't. We've completed our analysis and started our requirements, but there's really nothing to deliver to you to help you out. So that is a traditional methodology.

Now let's take a look at an agile methodology. It takes these stages and it still uses them, but it flips them on their head. So instead of going from stage to stage to stage throughout time, the stages are listed vertically. So when you now have a slice and you do a vertical slice, that's where that term comes from, so you do a vertical slice on that product now you have this product increment. So at a certain point of time which is a sprint or a time-box or within that 1 to 4 week time frame, you are creating a potentially shippable product increment. Now you do that over and over on every single sprint and you're creating these product increments.

To explain this with another type of example, everybody's had some type of application software either on their phone or on some type of desktop computer that once every couple weeks, once a month that you get some type of update it's now version 1.4.6 and then it's 1.4.7 little bit later and a couple of weeks, 1.4.8, what they're doing is

they're shipping new product increments the shipping to new features and changes and updates that they've made to the application. And they're doing that over and over and over. So you can start to realize that value and that product increments it's only going to contain a couple of features, a couple of new things, a couple of changes, but when they do that over and over and over and they continue to deliver that to you, you're able to continue to realize that value rather than waiting for all of their features as part of this project to be complete.

So product increment is a vertical slice of a potentially shippable product that's developed as part of a sprint time-box.


Traditional
Analysis
Requirements
Design
Develop / Build
Test
Release
Time
Agile
Product Increment
Product Increment
Product Increment
Analysis
Requirements
Design
Develop / Build
Test
Release
Time

Estimating in agile

Introduction to Estimating

So estimating, is the concept of roughly calculating. In this particular case, we're roughly calculating how long it's going to take us to complete a particular user story, how long it's gonna take for us to get from beginning to done done. Now that roughly calculating thing ? I don't know, calculating to me sounds a lot like you have all the information and you're just kind of putting it together or figuring things out- like a math problem. But to me, estimation is nothing like a calculation, it's more of a guess, but it's not just a wild guess. It's actually an educated guess.

You're using some information to help you come up with your best educated guess on how long this is going to take. Now the information you use, there's really three key pieces to it.

1. Number one- which is the most important- is the elicited information. So this is the information you've gathered from stakeholders. This is their various needs. This is their wants. This is ultimately what put together the core of that particular user story. So that obviously based on what they tell you there will dictate how long it's going to take you to actually solve that with some type of solution.

2. Secondly, you want to utilize your past experiences. So this could be experiences on other projects or other user stories that are similar to this- where now you can utilize the lessons learned from that past user story- to actually help you create your best estimate for this user story.
3. And finally, you can also use various documentation from your organization. So your organization may have some standards documentation for estimating. So some organizations are very mature in how they do it. Some organizations have some repetitive tasks and so they have some various documents that will help their team members to put together an accurate, or as accurate as possible, estimate.

So now that you understand what estimating is in general let's move on and talk about estimating specifically for Agile.

Estimating in Agile

Where I want to start is actually not looking at agile, I want to talk about estimating for non agile projects and these will help to give you some context prior to jumping into the agile side. So for non agile projects, is this more waterfall or iterative projects, you are going to get absolute estimates.

What I mean by that is you've completed requirements and you've handed that those requirements for that particular item or task or feature over to a developer. And now that developer's are going to developer, I guess I should say developer or solution or because it may not be like a software developer. But anyways you handed over to the person that's going to be completing that particular feature. And they are going to read through everything and from a non agile standpoint they're are going to deliver back a number. So this is going to usually signify the number of days for the duration of that particular feature. I feel this particular feature is going to take me three days to complete. That would be an absolute estimate, you're giving a set number of days, in this case, for how long it's going to take to complete that particular feature.

Agile is a little bit different. Agile looks at things relative to the other features, items, tasks that were estimated as part of that project. So agile looks at it and says: ***Is this a small or is this a medium or is this a large item? We don't care about specifically how long we feel it's going to take us.*** We just need to know if it's small, medium or large, what bucket does it go into. And now when we say a relative estimate the reason they say that is it's relative to the other items that have already been estimated as part of that project.

So let's break that down a little further with an example. On the screen you'll see a fly, a frog and an elephant. The fly is small, the frog is medium and the elephant is large. So as we're working through if we were trying to categorize or bucket a whole bunch of animals we would use these as our relatives sizing, if we find a rhinoceros is going to be a large is going to go with that elephant. If we find a butterfly, maybe that goes with the frog because it's the same size of frog or maybe it's a really small butterfly and goes with the fly. ***So you can start to do the sizing or the estimating based on relativity to the other things that were already estimated.***

Now, the thing to recognize with the relative estimate is that relativity is really only inside of that project. So those team members are estimating tasks based on other tasks or features that were already estimated as part of that project. And so they're going to look to try to put it in that small, medium or large bucket. But those buckets don't necessarily apply to every other project. Let's look at another one. So here we've got a ladybug, an elephant and a tree. The Ladybug here represents the small, the elephant is now a medium, in this particular project ,and the tree is a large. So as you can see even though that elephant was a large estimated item as part of the previous project, that exact same item or a

requirement or feature could be estimated as a medium as part of a different project. It's just, it's all relative to the other items, the other tasks, the other features that are being estimated out. So it's a really important concept to grasp.

So I hope that makes sense in regards to a relative estimate from an agile standpoint, to an absolute estimate in not agile. Personally from an estimation standpoint agile is so much easier so much easier to do a relative estimate than it is to do an absolute estimate for a couple of reasons. One is you can just kind of place it in a bucket comparing it to other things rather than detailing out in your mind every step that you need to complete to then come up with an absolute estimate. So that's one big thing and the other big thing is the third bullet points that I have listed here. In agile updates usually don't affect a whole lot in the project, they usually have a pretty minor effect. And what I mean by updates is an estimation update. So let's say we put it into a small bucket and it ends up turning into a medium sized feature. OK, well it could hurt us in our sprint a little bit.

We could maybe not get everything done that we wanted to get done, but it's ok, we can bump another thing out to the next sprint. From a non agile standpoint it can have pretty dire consequences for a project. What I mean by that is let's say it was

estimated at two days and then it actually took four days to complete, that two extra days has to come from somewhere and I guarantee on the plan there were other features and items and tasks that were lined up to be completed after that. Well now that that first one got pushed back has to cascade across all of those other tasks, moving them back or trying to find a way to make up that additional two days that was unaccounted for because of the incorrect or invalid estimate.

Estimating in Agile: Why Estimate?

Let's talk a little bit about why it's important.

1. Number one and this is the most obvious one is an estimate is providing you an estimated duration for completion. About how long do you think it's going to take you to complete this feature based on the requirements you were given? From an agile standpoint, they'll put it into a bucket in the examples we gave earlier small, medium, large. But that could be different buckets based on various techniques, . But that's the most obvious one, you are giving them the requirement. They're telling us an estimated duration. Now we can use that for our planning and moving forward on that particular feature or deciding if, you know what, let's go ahead and table that, it's going to take a little bit longer than what we're expecting.

2. The second thing that estimating does for you is it helps to drive out clarification questions. When you're asking somebody to commit something. You're asking them to commit an estimate for a particular task they're going to spend a little bit extra time making sure that they read the feature, they read all those requirements, they read the acceptance criteria and they have a good understanding of how they would go to accomplish this particular feature. Now with doing that, they're going to also drive out additional clarification questions or unknowns that weren't defined there. So by asking for an estimate, not only are you going to potentially get a duration back, you also are going to get clarifying questions of things that you need to dig in further to and make sure that you have defined as part of the requirement prior to that requirement actually being able to be completed by that developer or the person creating that solution. So that's number two it helps to drive out clarifying questions.
3. Number three is it highlights complex or high risk tasks. As you're working in projects more and more you get a sense of how long things take. When you're writing your user stories or features. You have an idea. This is a small, medium or large type of task. Now that's going to be really useful when you hand that to somebody that's going to be creating it and

then they give you an estimate back and they say OK, I think this is a this is a medium requirement or medium feature you may say wait a minute I was thinking it was a small. So let's talk through a little bit. What are some of the complexities? What are some of the risk items that made you consider it a medium? So it helps to bring conversation and highlight some of those complexities and risky issues that you may not have been aware of or accounted for. Now by highlighting and by knowing that information you can make adjustments, maybe you tweak the requirements a little bit, maybe tweak that feature to make it less complex by taking very little away from it or changing it very little. Maybe you look to mitigate or remove those protect potential risks that were identified as part of that feature. And so then it can be reduced in size. Ultimately, the goal isn't to reduce the size, but it's to make sure that it's in line with expectations and with what would want to be done for the value that feature provides.

So those are the big reasons why we estimate in agile. Provides that estimated duration, drives out additional clarification questions and can help to highlight or point out areas of complexity or high risk that were previously unidentified accounted for.

Common Agile estimation techniques

Big/Uncertain/Small also known as the BUS, we have T-shirt sizes and we have the Fibonacci sequence.

Big/Uncertain/Small also known as the BUS

We're going to start with the most simple one the BUS. The Big/Uncertain/Small estimation technique has three buckets. You have a bucket for big user stories, you have a bucket for uncertain user stories and you have a bucket for small user stories. All of the features, all the items that are being estimated get placed into one of those three buckets. Ideally, you want as many features in the small bucket as possible.

Those features are small and bite size and can be done in a relatively short amount of time. Now with any of the stories that are in the big bucket, the first thing you should try to do is break them up. Is there a way that we can take that existing user story and break it up into multiple user stories, ultimately, parse out that user story and then have those be estimated at small that way their bite size they're easy to complete and provide value. Not all big user stories can be broken up and still maintain the same amount of value delivered to the stakeholders or the users in this case.

So some of them will stay big. But you do want to

make an attempt to try to break up as many of those as possible. Any stories that are part of the uncertain bucket that means that the requirements, the user stories are too vague, maybe the concepts behind it are bleeding edge and so it's really difficult to estimate how long it's going to take. These ones most definitely need to be broken up, defined and redefined and then reestimated to hopefully get them into either the big or the small buckets. So BUS is a fairly simple agile estimation technique. The most often I see this utilized by companies that are just switching to agile from more predictive methodologies such as waterfall.

T-shirt sizes.

This one has multiple buckets, you actually usually have five buckets as part of this you have an extra small, a small, a medium, a large and an extra large. So those are the five buckets that the features can be estimated into and they go along with relative t-shirt sizes. The reason I think T-shirts are used is because it's very global. Everybody understands what a size of a shirt is regardless of what language they're in, what country they work for or what project methodology they've used in the past. And so this one has also become a fairly popular estimation technique for organizations switching to agile.

But some of them just stay on these T-shirts

sizes, they don't need it more complex than that. So each user story obviously is compared to each other based on how long it's going to take, it's placed into the one of five buckets extra small, small, medium, large and extra large.

The large and extra large user stories are the ones that you should look at and see if you can break them up if possible into smaller user stories, ones that would fit into that extra small, small or medium sized buckets. It's not always going to be possible, but highly recommended that you don't have many if any large or extra large stories as you move forward to begin some of your sprint planning. So that's the T-shirt sizes estimation technique.

Fibonacci sequence

The Fibonacci sequence has the team grouping together their user stories into buckets. They're determining the groups by pulling together the user stories of approximately the same size of effort to complete. Okay, so we have these buckets. Let's call them A through G. So we have Bucket A, Bucket B, Bucket C, D, E, F, and G. The "A" bucket represents the smallest effort user stories. The "G" bucket represents the largest effort user stories. So the team grabs a user story, analyzes and discusses the details, and then determines the appropriate bucket it would fit into. But with our current naming convention, it's

hard to know the real difference between a lot of those middle buckets.

Like what's the real difference between Bucket C and D? Or a buckets D and E? It's hard. So what we'll do instead, is use a sequence of numbers, called the Fibonacci sequence, to name the buckets. So using the Fibonacci sequence, the first bucket is going to be 1. The second bucket will be 2. Now, the way the Fibonacci sequence works is you add together the two previous numbers to get the next one. So with that, our first bucket is 1, second bucket is 2, 1 + 2 is 3, so the third bucket is 3. As we continue on following the same pattern, 2 and 3 makes 5. So the next bucket is 5, 3 and 5 is 8, 5 and 8 is 13, and 8 and 13 is 21. So now our bucket groupings are following that Fibonacci sequence. With the bucket being labeled with numbers, it's naturally easier for So the next bucket is 5, 3 and 5 is 8, 5 and 8 is 13, and 8 and 13 is 21. So now our bucket groupings are following that Fibonacci sequence. With the bucket being labeled with numbers, it's naturally easier for the team to group those user stories by similar size, and be able to tell apart those that were estimated at a medium -ish size. Well, I hope that helps to explain it.

So the sequence goes like this. 1, 2, 3, 5, 8, 13, 21 etc.. So those numbers it may seem a little random at first, but the sequence says add the two

numbers prior to the number, and you'll get that number. So 1and 2 is 3, 2 and 5, 2 and 3 is 5, 3 and 5 is 8, 5 and 8 is 13, 13 and 8 is 21. So if we would go to the next one, it'll be 13 and 21 is 34. So as it... as you get larger, the numbers gets larger exponentially a lot faster. Right? And you... you add up a lot quicker. And the concept around that is, the bigger the story, the more complexities, the more uncertainties, and the more risks, that are [going to] be around in completing that estimate in that... the kind of estimated or relative time frame that's being expected. So, the numbers, like I mentioned for the Fibonacci sequence, are kind of abstract, but they're used to help provide some more details to the team as to what they can commit to for that Sprint. So these numbers are called story points. It's how many points that story will been... will need, in order to be completed.

As in a lot of the other techniques, any user stories that are larger, really should be broken down. In this case, 13 and 21, for sure should be broken down into smaller user stories that can be estimated and put into smaller buckets. Take less time to complete to deliver that value. As well, a lot of times the 8, is kind of looked at in and determined, hey can this be broken up at all? Because the smaller the stories are, and the smaller story points are assigned, the easier that

task is going to be completed, and the more likely you're going to be able to get it done within the estimated timeframe that you're expecting.

Other Agile Roles

Project Sponsor

Their main objective is to approve that project and the budget. They're the one that's championing this whole thing. They're the one that has the idea of the big picture and they are saying that this solution is going to provide value to the business or the organization. And we definitely want to invest our time and money to move forward and achieve that solution. With that, the Project Sponsors are helping to shape that project need and the outcomes.

1. Approves project and budget.
2. Provides the big picture view.
3. Helps shape the project needs and outcome.
4. Ensures project stays aligned to company ojectives.
5. Participates in Sprint Reviews(and other feedback demo).
6. Recognizes the team for quality work
7. Encourages corss-team/ cross-department collaboration.
8. Provides perspective on product impact and use.

Business Management or Business Executives

This is the Business Management or Business Executives, within the organization. One of their responsibilities, that's extremely important is just making sure they have an open mind. And they're

enabling access to their various business resources. Whether that be their management team, their supervisors, their subject matter experts, their users or other business resources that are necessary to help make sure that, that solution meets those business needs.

1. Enables access to necessary business resources.
2. Provides the needs of management.
3. Ensures user story results are creating value for the business.
4. Escalates resolution of identified impediments(as needed).
5. Participates in Sprint Reviews.
6. Encourages corss-team/ cross-department collaboration.
7. Acknowledges team successes and provides support.

Technology Leaders

These are the various IT managers, directors or executives of technology. The number one thing, that technology leaders need to make sure they're doing. If they don't do anything else. This is number one and that is hiring the right team members with the right skill-sets in Agile. Every team member has to be able to produce, on their own. They need to be able to solve little problems that come up, hiccups that come up. And they need to be able to collaborate and work with other team members, in order to help the team create that solution and solve that business

problem.

1. Hire Team members with right skillset.
2. Gives up control - empowers, trusts, and support team.
3. Enables access to necessary technology resources.
4. Escalates resolution of identified impediments(as needed).
5. Participates in Sprint Reviews.
6. Encourages corss-team/ cross-department collaboration.
7. Acknowledges team successes and resolves team conflicts.
8. Ensures solution is aligned with organizational standards.

Agile Detractors - Leadership

These are things that negatively affect the team, as well as, the Agile project.

1. Demands mid-sprint changes.
2. Hijiacks ceremonies to dicuss 'high-priority' topics.
3. Not available for demos . Then requests changes at the sprint Reviews.
4. Discourages communication with other teams.
5. Removes or changes available resources min-project.
6. Doesn't value agile mindset.

Subject Matter Expert / Senior User

These are the people on the business side, that have

been working in that industry, working in that business for a while. They get it. They know the technology, and or they know the processes, and or they know the industry and the business. And so here are the responsibility of those folks.

1. Provides thier needs and the needs of other users.
2. Hijiacks ceremonies to dicuss 'high-priority' topics.
3. Participates in Sprint Reviews(and other feedback demos).
4. Gains and communicates feedback on product deliverables
5. Provides perspective and context for requirements.
6. Completes requested testing in a timely manner.
7. Ensures user story results are used properly

Business Users

Similar to the Senior Users, the Business Users are helping to provide their needs. They're helping to give context and perspective, as to what they need and what they want to have within the solution. If those Business Users are invited to the feedback demos, go! It's very important if the Agile team is inviting various Business Users to feedback demos. It's for a reason that they have a unique perspective or unique use of that product deliverable. And it's important that they attend and provide their

feedback. Obviously, with the Business User, you're going to attend, watch or read training on the deliverables.

1. Provides their needs (differentiates wants and needs).
2. Provides perspective and context for requirements.
3. Attends feedback demo they are invited to.
4. Attends, watches, or reads training on deliverables
5. Utilizes deliverable and gives feedback.
6. Completes requested testing in a timely manner

Agile Coach

The Agile coach is an experienced Agile practitioner. This could be a consultant, it could be somebody that works within the organization. They're there to help train, guide and support the full Agile for the organization. So they're experienced in the Agile ways. They understand the ceremonies. They understand the terms. They understand the roles and responsibilities. And so they're helping everyone, all those various roles know, what their roles and responsibilities are within the various aspects of the project. But beyond that, they're also ensuring that the ceremonies that are being done, are done correctly according to the guidelines that the Agile values. And principles are being upheld within the organization.

So a lot of times, an Agile coach is brought in. Obviously, as companies are transitioning to Agile. But even after that, a lot of them hang onto an Agile coach. Just to really sit back, see the full picture and help guide the organization forward with Agile. And make sure that everybody knows, what they're doing and point out any type of issues or changes that need to be made, for the organization to be better from an Agile perspective or just better from an overall perspective.

Post Script

Now, one point of clarification, is we've talked about and taught you about these five other Agile roles. The Project Sponsor, the Business Leaders, Technology Leaders, Senior Users, and Business Users. And we've taught you these, based on kind of a Scrum perspective. We taught you about Scrum and we've tied this into Scrum. But I wanted to make mention that these roles are also the same thing for Kanban, for Scrumban, and other Agile frameworks. These roles are still helping out move the Agile team forward. That Project Sponsor, while they may not be working with a person titled Product Owner. They're working to approve the budget and

make sure that the needs and objectives of the organization are being met with this particular project. Same with the Business Leaders. So the roles and responsibilities that we've taught throughout this section, can be applied regardless of the framework. We just applied it to more of a Scrum looking framework. Because that's what's familiar to you. Because that's what's been taught at this point in the course. So I hope that helps to make sense. That those five additional roles will play a part, regardless of your Scrum or another Agile framework.

Digging Deeper

Team Velocity

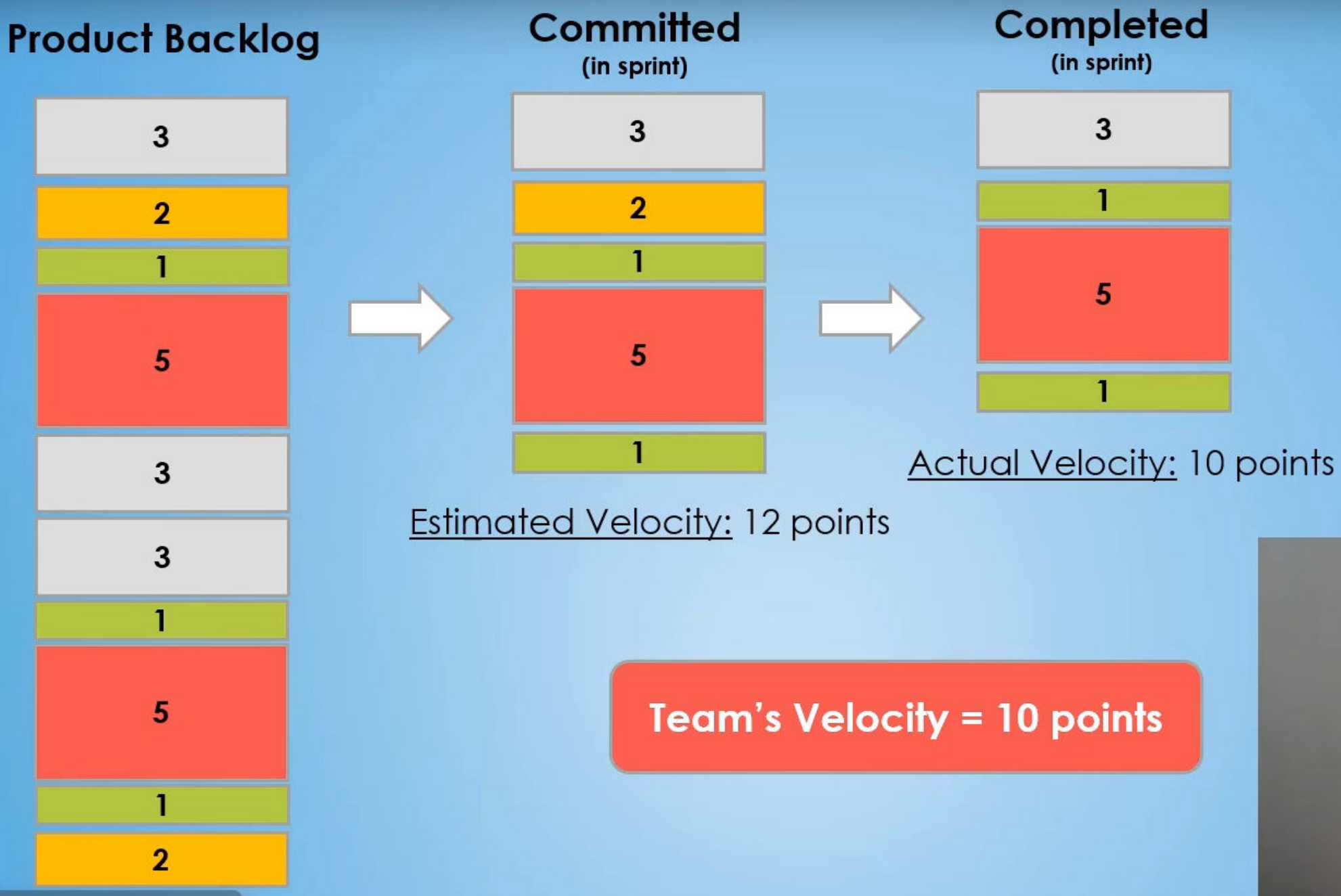
Team Velocity. Team Velocity in short, is how many points of stories can the team, this project team, complete in a given Sprint. So, ultimately, what's the average number of story points that can be completed every Sprint?

Now let's break that down a little bit, with an example. So imagine, here we have our Product Backlog. We've got a number of stories out there. Every rectangle is a user story. The numbers inside of it is the estimate that was applied to it, utilizing the Fibonacci Sequence. One (1), being the smallest user story. In our Product Backlog, 5 being the largest user story. So there's our full Product Backlog. Now we have a pretty small team. And so that team has gone through, looked at the priority, understood what the points are and they have committed to twelve total story points.

They've committed to twelve (12) story points, to be completed as part of that Sprint. So that becomes their Sprint Backlog. So that, team's estimated velocity is twelve (12) points. Now throughout that Sprint, the team has actually only been able to complete ten (10) of those story points. So two (2) of them are going to roll over to the next Sprint. So the actual velocity of the team

is ten (10) points. Now, over time, over Sprints, you're able to look back and understand.

Okay, how many points were we actually be able to complete in that particular Sprint? And then if you look, over Sprint over Sprint, you are able to start to understand your Team's Velocity. This is going to help you, not only understand how long it's going to take you to get through the full backlog potentially. But it's also going to help you as you're committing every Sprint, to not overcommit. If you know that you guys can complete ten (10) points, every single Sprint. Well, then commit to ten (10) points and work through them, get them done. ***So that Team Velocity is going to really help in your Sprint commitment planning, as well as, the commitment of the overall project timeline.***


Product Backlog
3
2
1
5
3
3
1
5
1
2
Committed
(in sprint)
3
2
1
5
1
Completed
(in sprint)
3
1
5
1
Estimated Velocity: 12 points
Actual Velocity: 10 points
Team's Velocity = 10 points

Burndown Chart

A Burndown chart is a visual representation of work being completed, throughout the various Sprints of the project. Now the design of this particular chart will be different, depending on the tool you utilize to create it. But ultimately, ***they show the same sort of data. On the vertical axis, we have our work, our story points. On the horizontal axis, we have our time, our Sprints. And in the middle, we have two lines. The orange line, is representative of what we want from a trending standpoint.***

For us, to achieve our hundred and twenty story points, in our ten Sprints. That orange line is, taking into that account. This is where we need to be, in order to complete that at the end of this project. The blue line is the actual. This is the work that's left remaining after that Sprint is completed. So as we look at Sprint Zero, we're at a hundred and twenty story points. Obviously, we haven't done any work yet. So hundred and twenty points, as part of the project and a hundred twenty story points that need to be completed yet. As we move into Sprint one, you can see that we fell behind a little bit.

We were slightly behind there, because our number of points that we completed, was actually less than what our trend was showing. Now this happens a lot on teams, especially new teams. When you're just forming your team together,

you've got to go through, you've got to make sure everybody knows their role and kind of get into a good fluid rhythm, before the scrum really takes off. And before you can really achieve your full Team Velocity, in every single Sprint.

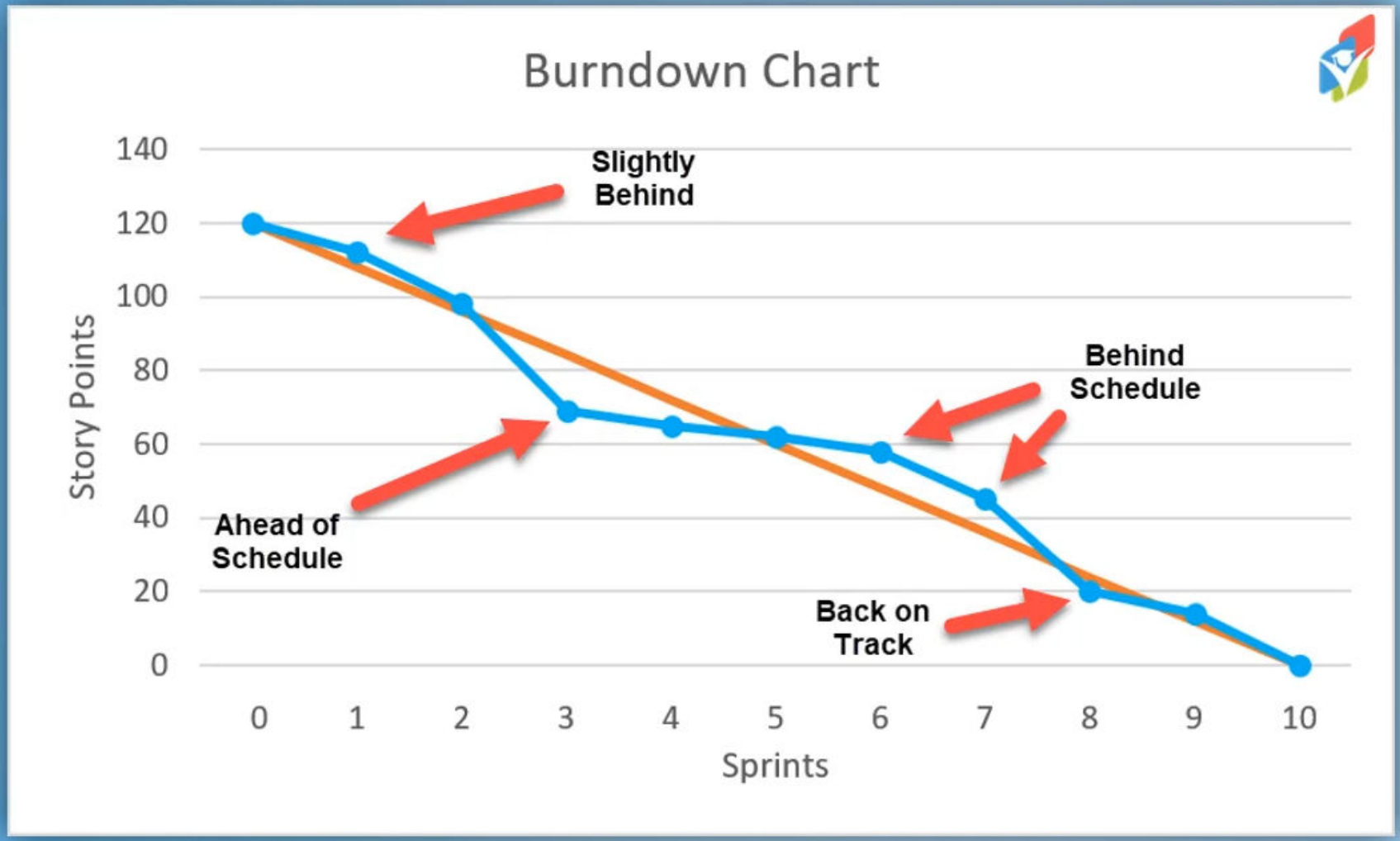
So this happens a lot. That first Sprint, you tend to fall a little behind on that trend line, just because of that. Then as we move into our second Sprint, you can see we caught up a little bit. We are able to complete some additional story points and kind of catch up towards that trend. As we look to Sprint three, we actually got ahead of schedule. We were able to complete more story points, than what we needed to in order to achieve those four hundred and twenty story points, at the end of the project.

In this case, 10 Sprints. Sprint four. We completed a little bit but we didn't make much progress. Sprint five, same thing. Sprint six, same thing and we actually fell behind. Because through sprints three, four, five, and six. You can see that the line doesn't really change much. We didn't really get many story points completed. As we look at Sprint six and seven, were now behind schedule and we're getting towards the end of the project. We've gone above the trend line and so we need to make some adjustments, to make sure that we can get completed with this particular project, in our 10 Sprint timeline.

Or look ahead to potentially adding sprints, if we need to, in order to get all of our work done. In Sprint eight, we get back on track.We get our story points completed and now we're ahead of the trend line, heading into the final two Sprints. We're able to knock it out, stay on task, on target, and complete our project on time. So, as you can tell, this Burndown chart is very helpful to a project team. So they understand where they're at, in the grand scheme of things. In a Scrum project, you're so focused on that particular Sprint. You're so narrowly focused, working hard to get those requirements and design, development or solution created, tested, and ready to be delivered, at the end of that Sprint.

But you're not really focusing on the big picture. So it's really easy to get lost in where the project is overall. The Burndown chart was created to help give a visual representation to where you're at. That way you can make adjustments. You know that you're behind or you're ahead. You might be able to add some additional, nice to have features, and stay on the same task. Or maybe you can move up your delivery date and deliver that overall project value sooner.

Burndown Chart



Burn Up Chart

Now one of the big challenges with the burndown chart is it doesn't do a good job to show changes in the project. We're working in an agile environment in the burndown chart is kind of assuming that the amount of work that was estimated at the beginning is the amount of work that needs to be completed for the project. And we know that's not going to be true, it's going to change. We can't guess at the beginning 120 story points based on our estimates of our user stories and assume that no user stories will going to be added or removed or estimates are going to be changes. That's just unrealistic.

And so, there's an alternative tool that some organizations utilize. And the alternative tool's called a burn up chart. This is very similar to a burndown chart in the vertical axis you have your story points. your horizontal axis you have your sprints and you have two lines. The details that are shown within those lines are slightly different. The orange line in our example is showing the total number of story points that need to be completed for that project to be considered done. The blue line is showing the total number of story points that have actually been completed by the team. By putting both of these on the chart, now you're able to compare and see are you really actually making progress and the team actually make good progress

or did something change overall.

So let's compare that to our burndown chart. And actually these two different charts that we created for this example are the same project. So you wouldn't know that by looking at it, but we're going to be able to make some very key discoveries with the burn up chart that we would never be able to see with the burndown chart. First of both of the examples show that there's a 120 story points to be completed for the project as of sprint zero. So before the project started hundred twenty story points.

Now as we move forward if we look at the burndown chart between sprint 2 and 3 it looks like the team completed a heck of a lot of work. We took a great decrease in the amount of work that needs to be done yet for this particular project. So it looks like the team did a great job just to get a lot knocked out. But when we look at the burn up chart it's actually because work was removed. The team did make progress. We can see that, the completed story points did go up in Sprint 3 on our burn up chart, but it did not go up as drastically as what the burndown chart is making you believe it did. And the biggest reason is because work was removed. We removed like 25 points or so from the overall project and that's not the only thing.

Let's look ahead to sprint 3 and 4 so comparing for looking at the burndown chart it looks like the team didn't really complete much, it's stayed pretty steady. There weren't many points that were completed, but when we look at the burn up chart we can see all those work added. We actually added 35 story points as part of that sprint. So likely a new feature was recommended as part of the the project. And so in sprint 3 they removed a feature that they deemed not as important, in sprint 4 they added a feature that they wanted as part of the project or maybe it was a redesigned look at that same feature. Whatever the case the burn up chart helps to show that work was actually added there and that wasn't that the team just didn't get done with their work.

Looking back at the burndown chart between sprints 6 and 7 again it looks like we made pretty decent progress. We made some some good progress on that but it doesn't take into account that there were some adjusted estimates, we actually reduced our total number of story points for the project from 125 to 120. So again something the burndown chart doesn't show. Between sprint 7 and 8 looks like the team did a great job catching up. And in fact they did an awesome job. There was actually five or or 10 story points that were added as part of that sprint that were able to get knocked out, the burn up chart

helps to show that, it helps to show that there was work added and we still were able to make progress up to it.

So as you can see the burn up chart, while very similar and showing the same type of data it shows it in a different way to help point out some various changes to the project that the burndown chart may not be showing you.

Something remember

Well really it comes down to usually the scrum master. The scrum master's usually the one that's put in charge of making sure that these are created and shared with the team every single sprint. I've actually been part of quite a few teams that will create these and they'll put them up inside of the rooms that the project team members are in. It's a constant reminder of where we need to be and where we're at. And it can help to keep the team focused on driving things forward.

Small List of Major one line defination

1. ***Sprint*** Is a time-box - a specific set of time
2. ***Courage Value***: Do the right thing and work on tough problems
3. ***Definition of Ready*** User story contains enough details to be worked
4. ***Product Increment*** Vertical slice of a potentially shippable product
5. ***Burndown Chart*** Visual aid that shows trending of user stories completed

6. ***Sprint Retrospective*** Share lessons learned from the previous iteration
7. ***Scrum Master*** Servant leader that removes roadblocks
8. ***Focus Value:***Complete the right work based on sprint goals
9. ***Sprint Review*** Meeting to demo progress made during the iteration
10. ***User Story*** Simple description of a feature from a user's perspective
11. ***Sprint Backlog*** Committed user stories for upcoming iteration
12. ***Product Owner*** Tasked with understanding the product vision
13. ***Sprint planning*** Discussion and commitment for the next iteration's work
14. ***Respect*** Value: Treat everyone as a capable team member
15. ***Product Backlog*** Prioritized list of user stories that grows and changes
16. ***Definition of Done*** Common understanding of when something is complete
17. ***Daily Standup*** Discussion of yesterday's progress, today's plan, and issues
18. ***Working Agreement*** Team-created list of rules, expectations, and procedures
19. ***Openness Value:*** Discuss the work and challenges of the project
20. ***Team Velocity*** Number of story points that are completed per iteration
21. ***CommitmentValue:*** Give 100% effort to achieve project team's goals
22. ***Work in Progress*** Work that is currently being completed

Burn Up Chart