Software Development Methodologies

There is a demand for increasingly complex systems in the IT industry. These systems are very nearly impossible to build without a solid plan. Since it doesn't make sense to devise a project-specific plan from scratch for each project, we have methodologies!

These methodologies, when charted on a spectrum, reveals a range from **predictive, plan-driven** schemes like the Waterfall methodology, to **adaptive, change-driven** schemes like the Agile methodology.

The Waterfall Methodology

The Waterfall methodology is loosely analogous to manufacturing workflows. This is in the sense that it's highly **sequential**, where it's policy of no backtracking is perhaps its most distinctive characteristic. Projects developed with this methodology therefore share several key characteristics.

1. (very) **specific & stable project requirements**. It's pretty self-explanatory why.
2. **A need for documentation**. This is made possible by the structure of waterfall methodology's sequential design.
3. **Strict timeline.** Product Owners (POs) can get decent estimates- the waterfall methodology is plan-driven, after all.
4. **Uninvolved PO**. The PO's involvement is only needed for project requirements specification. POs that don't want to or cannot get involved with the development of their product would therefore opt for this methodology.
5. (relatively) **low importance for speed**. Waterfall is a meticulous process.

So maybe you have a project you want to embark on. But, which software development methodology?! 😟 Here are some advantages & disadvantages of the waterfall methodology. Maybe they'll help.

Advantages

- The methodology's emphasis on meticulous documentation means **future improvements can be made easily**.

- The PO **gets what they asked for**. Details like size, cost, & timeline are always well-defined.
- **Low dependence on development team**. This goes back to waterfall's thing for documentation. Another developer can easily pick up from where one left off.

Disadvantages

- **No backtracking**!
- **No changing of requirements**, either!
- **Everything hinges on the project's initial requirements**. It's do or die.
- **Late-stage Testing**. We all know how errors can cascade. 😕

The Agile Methodology

In stark contrast to the waterfall methodology, the agile methodology makes use of an incremental approach. A project typically begins with a simple set of requirements that are formed with the knowledge that they can be changed. The project is developed in small modules, where testing & deployment happen regularly.

The same way that waterfall projects share a set of key characteristics, agile projects share several hallmarks.

1. **Rapid production is important**. Agile is great at producing a market-ready product quickly.
2. PO wants **flexibility** or the project's nature means requirements are susceptible to change mid-development.
3. There is **no strict picture of the final product**. Agile's incremental approach rocks here.
4. **Motivated & skilled developers**.

Advantages

- **Changing requirements are expected**.
- The resulting systems are **up-to-date**.
- **The development process produces the envisioned product** because the PO is expected to give regular feedback.
- Regular testing == **bugs are found & fixed early on**!
- Builds systems that **can be deployed whenever**. These projects are thus more likely to make good on the intended launch date.

Disadvantages

- An incompetent project manager can lead to a late & over budget product.
- The **final product could be completely different than what was initially intended**.

The Agile Manifesto

The baseline of The Agile Methodology. Consists of 4 values & 12 principles.

The 4 Values of Agile

- **Individuals & Interactions** (over Processes & Tools)
- **Working Software** (over Comprehensive Documentation)
- **Customer Collaboration** (over Contract Negotiation)
- **Change-driven** (over Plan-driven)

The 12 Guiding Principles of Agile

1. The Priority- **Customer Satisfaction**. This is accomplished with early & continuous delivery of software.
2. **Welcome** changing requirements.
3. Software should be **delivered frequently**. This opens up a continuous flow of user/ client feedback.
4. There should be **daily collaboration** between the PO & the dev team. This means frequent demos!
5. Build a team of **competent & motivated** developers; provide them with support & trust them to get the job done.
6. Nothing beats **F2F conversation**.
7. Progress is measured by **working software**.
8. **Sustainable development**! Agile teams should be able to maintain their pace indefinitely.
9. Applaud **technical excellence** & **good design**.
10. Aim to **do as little work as possible**.
11. Give the team **autonomy**. Self-organisation wins the battle.
12. **Frequent retrospectives** manifest as productivity boosts!

Agile Framework- Scrum

In this module, we study Scrum- an Agile framework. Scrum is .. lightweight! It's made of **sprints** & maintains light overhead by maximising the time spent doing work. Good for complex projects, as it'll break it down into smaller, more manageable blocks.

A Sprint

The building blocks of the Scrum framework. Are typically short blocks that span 2 or 4 weeks. Its stages-

Sprint Planning

2 hrs / sprint week. i.e. a 4 week sprint will have a 8 hr sprint planning meeting (SPM). An SPM has 2 goals-

1. **Product Backlog Review**- (conducted with the PO) To ensure the product backlog remains aligned with the PO's vision of the product. The team looks at & re-evaluates the stories' content, size & their priorities.
2. **Sprint Backlog**- Decide the tasks to be completed in the coming sprint.

Basically, the SPM ensures that the team is ready to jump right into the next sprint.

Daily stand-ups

~15 minutes, every day. The goal: update the team. During the meeting, everyone answers 3 questions-

1. What have you accomplished since the last stand-up?
2. What are you planning on accomplishing by the next stand-up?
3. Any issues?

Note: the PO's presence is optional for these meetings!

Sprint Review

4 hrs/ 4 week sprint. A product-oriented meeting where the team showcases their accomplishments of the last sprint to the entire team + the PO. This serves as a way to update everyone on the project's progress (esp. relevant for larger projects) & the PO gives feedback regarding the direction of the project & communicates that to the team.

Note: the showcase is a *fun* thing. Meaning- snacks & light-heartedness!

The team also looks back on the last sprint, to draw their progress charts, raise any issues (tech-wise) & provide constructive feedback all around.

Sprint Retrospective

A team-oriented meeting where the team looks at the team dynamic & tries to find areas for improvement. The duration for this stage varies wildly, depending on the project & what's best for it. Scrum as a framework recommends a 3-hour maximum retrospective for a 4-week sprint. However, some teams implement a version as short as 10-15 minutes, where the sprint retrospective meeting becomes more of a huddled discussion.

Like mentioned above, sprint retrospectives' objectives remain the same, no matter the duration. Here are the key topics that retrospectives revolve around.

- What worked well?
- What can be improved?
- What are some goals that the team wants to achieve in the next sprint?

Do note that for this last point, it isn't simply a case of identifying big-picture goals. Individual team members are expected to make actionable commitments. For example, **have better time management** in the next sprint? No bueno. Team members should come forward with commitments like **update my to-do list & review my tasks' priorities twice a day**.

Benefits

- ⬆️ quality of deliverables
- Responds well to changing requirements
- Provides better estimates, quicker
- Provides better control over the project schedule

Non-Benefits

- No exact deadlines

Closing notes for the Scrum framework

After covering so much content related to the scrum framework, you may be inclined to take the points made in this document as the hard & fast truth. However, keep in mind that scrum is really nothing more than a set of guidelines that gives teams a nudge in the right direction. At the end of the day, teams are expected to adjust the scrum process to fit their team's workflow & the project's goals.

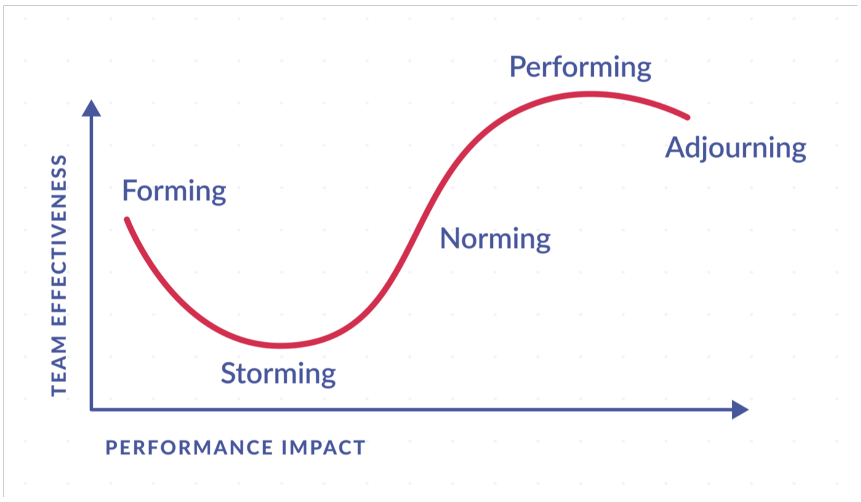
If you're interested in reading more about scrum to become a true **professional** 👨🏽‍💻, you can check out scrum.org, the Scrum Alliance page or the Scrum Guides.

Group Development

It's easy to overlook the importance of group development. After all, is it not sufficient to get a group of talented workers together? **No!** A group of talented people != a good team. An effective, high-functioning team has to be cultivated into formation!

The Stages in Group Development

Breaking this group development process into distinct stages aims to create high-performing teams. These different stages are significant for their differing levels of productivity, as is visualised below.



1. Forming

Teammates become acquainted with each other. A *first date*, if you will. Conversation typically revolves around easy topics like **skills & background**, **the project's goals & further details**, etc.

A lack of productivity at this stage is somewhat expected, since tasks are people-oriented, & team members are focused on getting to know each other rather than on getting work done.

Of course, nobody wants a team to fail in its wee stages. To avoid this, remember that **Communication is key!** Talk about everything. Talk about skills, interests, & project goals. Make sure that everyone's on the same page. Additionally, you're going to want to remove as much uncertainty as possible. So- **define everyone's role clearly**. This ensures that all job roles are covered & that boundaries are clear to everyone.

2. Storming

At this point, there is an increased order of communication, though team spirit remains lacking. The leader's word will not be followed. Personalities start to clash, members disagree, & leads to a generally bad time.

Some teams try to avoid this stage by, well, avoiding it. This is bad, though! This only encourages the problem(s) to grow until they explode! Survive this stage by keeping these few things in mind-

- Again, communication is key! Hold a meeting, talk things out.
- Mediation (either internal or external) is typically a good strategy as well.
- Keep your eyes on the prize. Don't let your team kill itself over trivialities.

3. Norming

Team spirit is established! Teammates recognise other teammates' strengths & are able to work together relatively well. This is not to say there are no conflicts in this stage at all. People clash. They just get better at getting over the clashes.

Keep things up by-

- Showing appreciation for your teammates! This sparks motivation & encourages closer relationships to form.
- Again, keep your eyes on the prize. Remind your teammates of the importance of the project.
- Use past successes to motivate your teammates into continually moving forward,

4. Performing

Teammates work in an open, trusting atmosphere with high productivity, like a well-oiled machine! Everyone is motivated & confident about their skills & value to the team.

This is the era of Highest Productivity, the era that all teams strive to attain.

Do note that some teams never make it this far. Some teams get caught up in the storming stage & find themselves unable to overcome their conflicts. 😦 However, the best teams can maintain these levels of productivity. Here are some tips for becoming one of these teams-

- Delegation is super impt!
- Indirect management is the way to go!

5. Adjourning

Assessment + recognition of teammates' contributions. Then, the group disbands. 😕

Common Values found in Successful Teams

The same way we're able to identify the stages of group development, it follows that we can abstract the values that successful teams have in common.

1. **Psychological Safety**- a trust to share honest opinions.
2. **Dependability**- a trust that teammates will get what needs to be done, done.
3. **Structure & Clarity**- having clear roles & goals.
4. **Meaning**- team members sincerely enjoy what they are working on.
5. **Impact**- team members believe that they are working to creating a positive change in the world.

Conflict Resolution

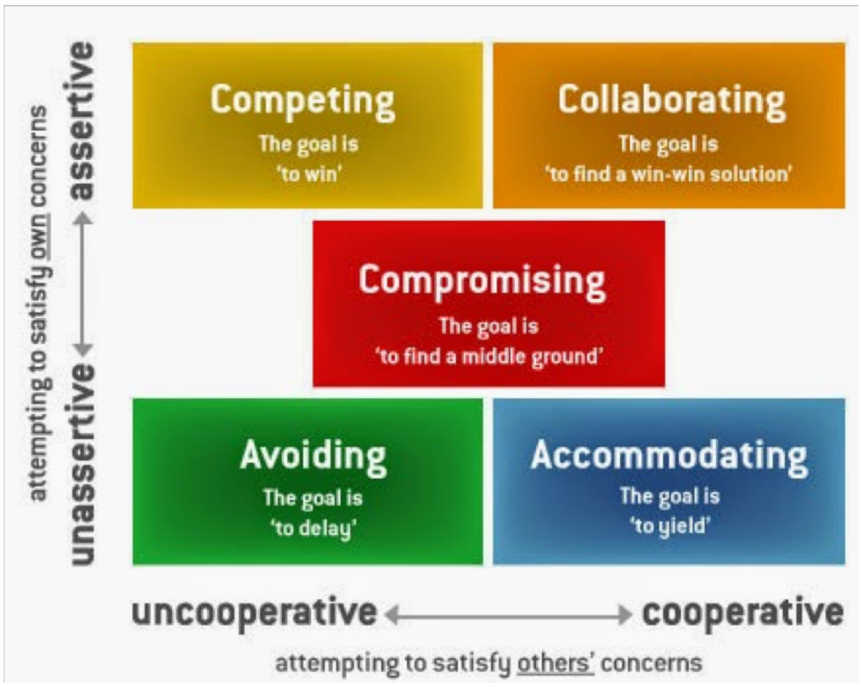
Here are some examples of how conflict can arise within a group. Note, however, that this list is **by no means exhaustive**.

- Unhealthy competition between teammates.
- Misunderstanding of job roles or requirements.
- Interdependence i.e. one teammate relies on another to get their work done.
- A subpar feedback system lacking frequent & genuine feedback.
- Disagreements in strategy or execution.
- Lack of focus on the goals that really matter.
- Poor delegation.

- Unproductive culture e.g. teams where it's commonplace to push the blame of missing a deadline to another teammate.

Conflict Resolution Styles

As people have differing personalities, it thus follows that people have different ways of handling conflict. Here's a widely accepted generalisation of that concept. For a quick overview of the different conflict resolution styles, check out the diagram below.



Competing

These people prioritise, well, winning. They're highly assertive & uncooperative. It's very typically characteristic of these people to be largely goal-oriented, with low priority on relationships. Their assertiveness can, at times, cross the boundary into aggressiveness. This means they are often perceived as intimidating individuals. Their need to win means that there is, more often than not, someone on the losing end.

👍🏻 / 👎🏻 This conflict resolution style is good when the person's decision is, in fact, the best decision. On the other hand, this conflict resolution style commonly breeds hostility & resentment.

Collaborating

These people aim to make all situations into a win-win situation. They're highly assertive, but also highly cooperative. This style shows a value for goals as well as relationships.

👍🏻 / 👎🏻 This conflict resolution is great because at the end of the day everyone gets what they want & theres no hostility. The only downside is that this method takes a lot of time & effort.

Compromising

Not dissimilar to the collaborative style just covered, people that take to compromise to resolve conflicts are ..moderate. They're moderately assertive, & moderately cooperative. These people value both goals & relationships. However, the fact that collaborative resolvers are steadfast in trying to find the best possible solution means they'd give up on some of their own values if others are willing to do the same, so that both parties can meet in the middle.

👍🏻 / 👎🏻 Relationships are maintained & conflicts are resolved. However, compromise can mean that the final outcome may be less than ideal. Also, this opens up avenues for manipulation.

Avoiding

These people simply don't want to deal with conflict of any kind. They're very unassertive & uncooperative. In the face of conflict, they give up their goals easily & remain largely passive. This means that they often end up in lose-lose situations.

👍🏻 / 👎🏻 No harm done to relationships, since they refuse to partake in conflict resolution. However, the conflicts remain unresolved & thus they tend to be perceived as pushovers.

Accommodating

These people are ..afraid to get on people's wrong side. They are often highly cooperative to whatever view is presented to them & are very unassertive. They prioritise relationships over goals. Put in the middle of a conflict, they can relinquish their goals if others are then able to achieve their goals. This typically manifests as a win-lose situation where the resolver is on the losing end.

👍🏻 / 👎🏻 Relationships are kept in top-notch conditions, though it leaves them highly susceptible to be taken advantage of.

misc. Jargon

Some terms you are likely to come across as you traipse through the world of software development.

Inception Deck

Your project, in a nutshell. It's not a pitch, in that it should cover more than just the flashy bits. It's really an attempt to iron out all possible kinks before even encountering them. This means deciphering details like..

- **Problem Statement**- A single line that conveys the project's mission.
- **The Elevator Pitch**- A crash course on the solution that covers details like..
  - Target Audience
  - Product Name
  - Main Features
  - Benefits
  - Innovation
- **The NOT List**- Clarifies the project's scope. What features can be expected to be included? What features are straight no-gos?
- **The Solution**- Great for setting expectations straight. This is a good time to express concerns regarding possible risks & the planned tech stack!
- **(rough) Timeline**- Sets expectations in terms of time, manpower, & budget straight.
- **Priorities**- In a pinch, how will the team side? Refers to things like scope, budget, time, & quality.

Product Backlog

A list (ordered by priority) of the features to be built into the final product. The act of maintaining & ordering this list falls on the PO. Do also note that this list is not set in stone after a one-time discussion. This list is fluid & expected to be kept as up-to-date as possible.

Definition of Done (DoD)

It's important to ensure that the entire team has the same understanding of **done** to prevent any unwanted miscommunications etc.

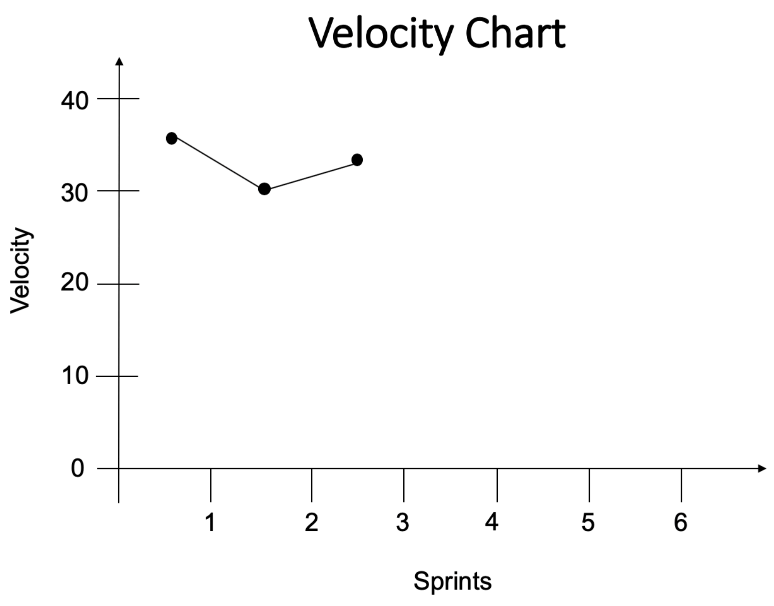
Note also that the DoD varies not only between teams, but also between different levels (e.g. feature, sprint, or the release). The level of complete-ness to signify a completed feature may not suffice to signify a completed release.

Velocity

How fast the team works. Its productivity, if you will. In science, **velocity = distance / speed**. In agile, **distance** refers to story points & **speed** is measured in terms of sprints.

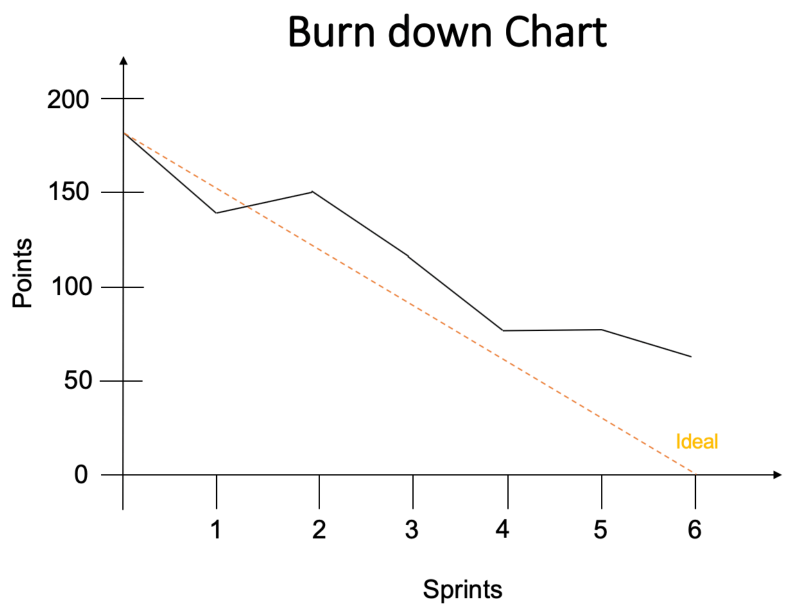
Therefore, **velocity = story points / number of sprints**. Velocity can be calculated before a project begins, using the project's total story points & the expected duration of the project to get an estimate of how fast the team has to work. Velocity can also be calculated after sprints, to get a gauge of whether the team is on track to completing the project as expected.

Velocity can be charted as a line graph of velocity against sprints, to provide a visualisation of the team's productivity. Check it out!



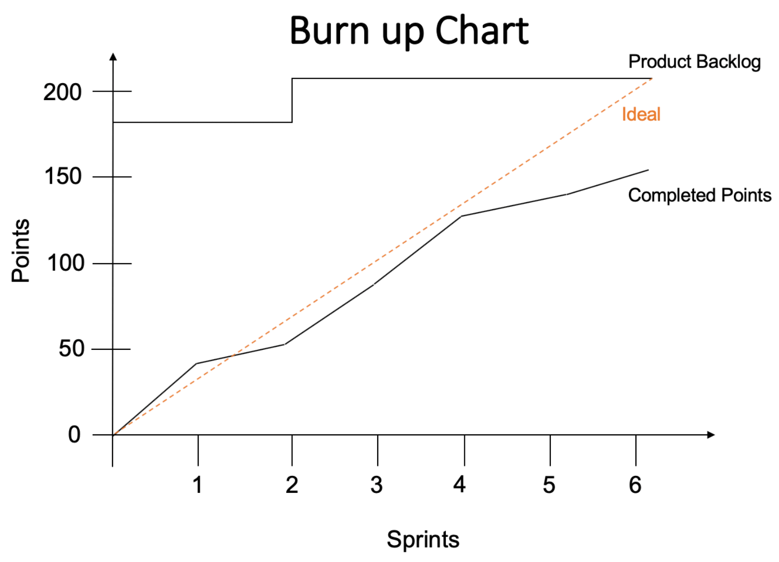
Burn down Chart

Provides a visualisation of the team's progress towards the project's completion. Is typically a line graph, of story points against sprints. Check it out!



Burn up Chart

Provides a visualisation of the team's progress throughout the project. Is typically a line graph, of story points against sprints. These are typically preferred for projects with changing total story points. Check it out!



The INVEST Framework

A quick guide to- **is this user story any good?** Try to keep to this framework, & you should be good for the most part.

- **I**ndependent
- **N**egotiable
- **V**aluable
- **E**stimable
- **S**mall
- **T**estable

Triangulation

An estimation technique commonly used for user story sizing. Get yourself some sample users stories that you've already implemented or are confident in your estimation of. Then, the team is able to size all other user stories in relation to these.

Simple, & logical! What more could you ask for? 😧

Planning Poker

Like triangulation, planning poker is an estimation technique used for user story sizing.

In planning poker, each member of the team first sizes all user stories (typically using the Fibonacci numbers with arbitrary units). Then, the entire team's estimates are compared & if everyone's estimates are similar, the team sticks with it. Otherwise, the team discusses the matter until a consensus is reached.