**Agile (and Scrum)**

Agile is a term to describe the opposite of Waterfall.

Waterfall is the process of:

* doing a ton of research to build a complete set of requirements upfront for a product
* doing all the work in one loooong stretch
* reviewing the work only once everything is complete

Agile is the aim of:

* working iteratively ("sprints" in Scrum)
* having frequent opportunities to review progress
* having frequent opportunities to change direction

The point of all of this is flexibility and speed. If I follow Agile, I can get a barebones website out faster, and then I can add smaller features. I can decide to put more time into more popular features by reacting to how well they are used in the moment.

If I follow Waterfall, I can't publish anything until it's all complete; this also means that I have to decide which features to prioritise at the very beginning before I have ANY users or feedback.

**Scrum**

In Scrum (a popular Agile methodology) you work in "Sprints" - usually two week iterations.

At the beginning you have Planning; you decide how much work you can do in the Sprint, and roughly break it down into smaller tasks to work on.

You work on your tasks (called Stories in Scrum) during the Sprint, working down in priority order as a team to get as many completed tasks as possible in the time available.

At the end, you have two meetings:

* Sprint Review, where you show and tell what you've done
* Sprint Retrospective, where your team discusses how the last sprint went and how they could improve.

You generally work with a Product Owner, who gives you the list of tasks (they're the single source of truth). They talk to the clients/stakeholders so you don't have to.

You sometimes also work with a Scrum Master who will facilitate all your meetings, including a daily check-in (stand up), and will help make sure you have what you need to do your job.

This is the most popular way of working in most tech companies; however, most companies won't use ALL these people and meetings.

Scrum is designed to be applied to teams of 7 plus or minus two (doesn't work very well on teams smaller than 5 people, but there are still some merits to keeping some of the processes/practices).

**Other Agile practices/methodologies**

List of buzzwords:

* Kanban
  + another Agile methodology
  + Sensei instead of a Scrum Master
  + You don't work in sprints, you just have a long "backlog" (task list) that you pick from
  + You should still have regular processes to retrospect
  + More responsive to change because you're not tied to two week/sprint buckets of time
* TDD - Test Driven Development
  + the practice of writing a test case, and then writing code that satisfies it
  + then add another test for the next requirement, build the code
  + etc
  + practice it on <https://cyber-dojo.org/creator/home>
* Pair-working/mob working
  + working in pairs or groups to solve problems and have multiple eyes on it
* Rubber ducking
  + asking a rubber duck instead of a person to fix your code
  + [https://rubberduckdebugging.com](https://rubberduckdebugging.com/) for info on this
* Lean
  + a set of practices particularly aimed at startups
* XP / extreme programming
  + another set of Agile practices/opinions

This isn't all there is to know about Agile, but it's a good taster of what you might want to read up on and be aware of when you go into your first role.