AGILE & SCRUM & Kanban

* **AGILE** Development: <https://www.zhihu.com/question/19645396>
  + what is agile?
    - it is a set of values and principles;
    - it is a collection of **BELIEFS** that team can use to make **DECISIONS** software development
    - **NOT**: methodology; way of developing software; framework/process
  + **Values: Agile Manifesto (Left side > Right side)**
    - individual & interactions > processes & tools
    - working software > comprehensive documentation
    - customer collaboration > contract negotiation
    - responding to change > following a plan
  + **12 principles (how to make the best decision for the situation; to support the 4 values)**

<https://www.cgi.com/us/en-us/life-sciences/blog/12-principles-of-agile-methodologies>

1. **Satisfy the Customer:** Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. **welcome change:** Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage. It's hard not to get a wave of despair when thinking about change requests but change is good if you can react to it fast enough. Change means you are getting closer to client needs and that's a good thing.
3. **Deliver Frequently:** Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale. The sooner you deliver incremental software, the faster the feedback and faster you can identify a wrong turn or a miscommunication with the client. Would you rather find out earlier when you can do something about it or at the end when a complete rework is required?
4. **work together:** Business people and developers must work together daily throughout the project. It makes sense for the customer to become part of the team. After all, both the developers and the customers have the same goal; to deliver valuable software.
5. **build project:** Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done. Agile projects emphasize self-organizing teams who instinctively can manage both themselves and the work. The micromanagement of projects is no longer required or beneficial.
6. **face to face time:** best way to make sure everyone is on the same page! The most efficient and effective method of conveying information to and within a development team is face-to-face conversation. Co-location is the ideal. Osmotic communication - where you absorb some part of the conversation around whether you are part of the actual conversation or not - is a very real benefit of co-location. Unfortunately, development teams are often distributed. If the teams are not co-located, every effort must be made to communicate often and to increase the use of technical communication techniques.
7. **measure of process:** Working software is the primary measure of progress. When you focus on following the plan you typically get too involved in updating documentation - taking focus off the objective of the project. When you make working software the primary measure of progress you promote it to the primary focus of the project.
8. **sustainable development:** Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely. Think Work/Life Balance. Remember the end of the project dash? Everyone worked almost around the clock to get the project finished. Never mind the impact on quality, how about the impact on the team? Agile strives to maintain a consistent level of activity which translates to consistent velocity. One important result is a better ability to forecast.
9. **continuous attention:** Continuous attention to technical excellence and good design enhances agility. While an elegant design is meaningful even more valuable is a solution that will span the test of time. Perhaps even more importantly a solution that can be updated to keep it current. What good is an elegant design if it cannot maintain its value through update and maintenance cycles.
10. **keep it simple:** Simplicity! The art of maximizing the amount of work not necessary - essential. Of course, the most reliable features are the ones that are not yet built...they can't fail. But that aside, nearly 30% of the functionality we build is seldom or never used. Agile is ruthless about cutting functionality that does not lend value.
11. **organized teams:** The best architectures, requirements, and designs emerge from self-organizing teams. Self-organizing teams that are cross functional as well. Who better to recognize issues before they become real impediments; of course, the people closest to the solution.
12. **reflect for effectiveness:** At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly. We've all been on projects that end with an AAR, After Action Review. These reviews help the next client but not the one you just finished. Agile defines several ceremonies and important among those is the Retrospective. Generally held at the end of each Sprint/Iteration it is a way for teams to catch and improve behaviors before they have a huge, detrimental impact on the project.
    * Conclusion:
      + what is agile? set of values and principles
      + how does a team become agile? **make their decisions** basing on those values and principles

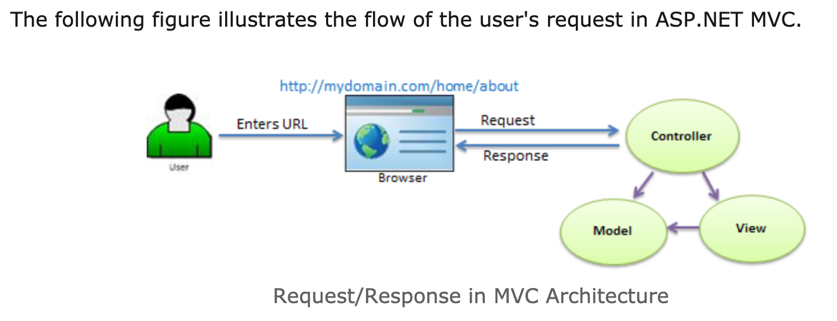
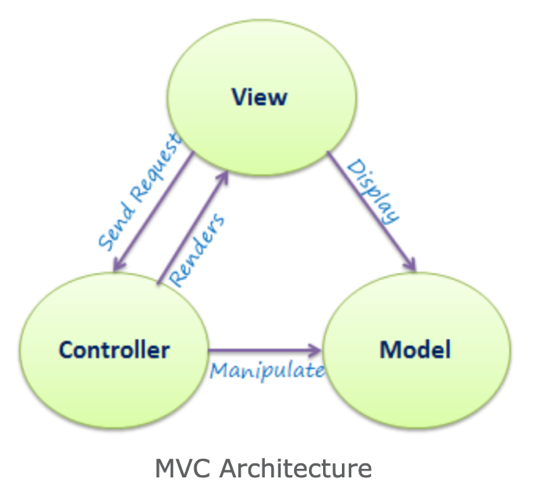
**SCRUM Development**: corporate pattern; the way to arrange & complete set of tasks

* <https://medium.com/doflowy/%E4%BB%80%E9%BA%BC%E6%98%AFscrum-%E4%B8%8D%E6%98%AF%E5%B7%A5%E7%A8%8B%E5%B8%AB%E4%B9%9F%E8%83%BD%E6%87%82%E7%9A%84scrum%E5%85%A5%E9%96%80%E6%95%99%E5%AD%B8-1cc6683575f8>



**MVC** architecture (Model, View, Controller)

* Model: represents shape of the data and business logic; maintains data of the application
* View: user interface; display data to user and allow them to modify
* controller: handles user request



**TDD** (test driven development): an AGILE requirement/design

* TDD is a software development process that relies on the repetition of a very short development cycle: requirements are turned into very specific test cases, then the software is improved so that the tests pass. This is opposed to software development that allows software to be added that is not proven to meet requirements.
* ATDD: acceptance test driven development (general sense)
* UTDD: unit test driven development (narrow sense)

**BDD** (behavior driven development): an AGILE requirement/design

**Spring** framework for JEE (Java Enterprise Edition)

* most popular app development framework for enterprise Java
* used to create high performing, easily testable, reusable code
* lightweight: ??????????? provides some tools only ?????????

**RESTful** API (Representational State Transfer): <https://www.jianshu.com/p/7f8810c28ab6>

**6 restrictions**

* uniform interface: decoupling & simplify the structure; every part work as an individual
  + resource-based
  + manipulation of resources through representations
  + self-descriptive message
* stateless (key)
* cacheable
* client-server
* layered system
* code on demand

**REST is design pattern NOT standard**

**RESTful** Web Service: <https://www.cnblogs.com/JoannaQ/p/3247077.html>

* use XML form for info transformation: allow web service work for any platform; any language
* core techniques:
  + SOAP: simple object access protocol
  + WSDL,
  + UDDI