Agile Software Development

Agile Software Development

- **Agile software development** is a group of software development methods based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams.
  - Methods
  - Iterative
  - incremental

- It promotes **adaptive** planning, **evolutionary** development and delivery, a time-boxed **iterative** approach, and encourages rapid and flexible response to **change**.

- It is a conceptual framework that promotes **foreseen interactions** throughout the development cycle.

- The *Agile Manifesto*[introduced the term in 2001

- Let's take this definition apart.

Iterative and Incremental Development

Iterative and Incremental development is at the heart of a cyclic software development process developed in response to the weaknesses of the waterfall model.

> It starts with an initial planning and ends with deployment with the cyclic interactions in between.

Iterative and incremental development are essential parts of the SCRUM, Extreme Programming and generally the various agile software development frameworks.

It follows a similar process to the “plan-do-check-act” cycle of **business process improvement**.

Time-Boxed Approach

- In time management, a **time box** allots a **fixed** period of time for an activity.

- **Timeboxing** plans activity by allocating time boxes.

Introductory Thoughts

- Fears regarding software development led to a number of pioneers / industry experts to develop the **Agile Manifesto** based up some firm values and principles.

- Practitioners had become afraid that repeated software failures could not be stopped without some kind of **guiding process** to guide development activities.

Common Fears

- Practitioners were afraid that
  - The project will produce the wrong product
  - The project will produce a product of inferior quality
  - The project will be late
  - We'll have to work 80 hour weeks
  - We'll have to break commitments
  - We won't be having fun.

- Goal: outline values and principles to allow software teams to
  - **develop quickly** and
  - **respond to change.**

Manifesto for Agile Software Development

- "We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to the value:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

That is, **while there is value** in the items on the **right**, we value the items on the **left** more."

Let's look at these values to discern exactly what is meant.

Value 1: Individuals and Interactions over Processes and Tools

- **Strong players**: a must, but can fail if don't work together.
- **Strong player:** not necessarily an 'ace;' work well with others!
  - Communication and interacting is **more important** than raw talent.
- '**Right' tools** are vital to smooth functioning of a team.
- **Start small**. Find a free tool and use until you can demo you've outgrown it. Don't assume bigger is better. Start with white board; flat files before going to a huge database.
- **Building a team** more important than **building environment.**
  - Some managers build the environment and expect the team to fall together.
  - Doesn't work.
  - Let the team build the environment on the **basis of need**.

Value 2: Working Software over Comprehensive Documentation

- **Code** – not ideal medium for communicating rationale and system structure.
  - Team needs to produce human readable documents describing system and design decision rationale.
- **Too much documentation is worse than too little**.
  - Take time; more to keep in sync with code; Not kept in sync? it is a lie and misleading.
- **Short rationale and structure document**.
  - Keep this in sync; Only highest level structure in the system kept.

Value 2: Working Software over Comprehensive Documentation

- **How to train newbees** if short & sweet?
  - Work closely with them.
  - Transfer knowledge by sitting with them; make part of team via close training and interaction
- **Two essentials** for transferring info to new team members:
  - **Code** is the only unambiguous source of information.
  - **Team** holds every-changing roadmap of systems in their heads; cannot put on paper.
  - **Best way** to transfer info- **interact with them**.
- **Rule**: Produce no document unless need is immediate and significant.

Value 3: Customer Collaboration over Contract Negotiation (1 of 2)

- Not possible to describe software requirements up front and leave someone else to develop it within cost and on time.

- Customers cannot just cite needs and go away

- Successful projects require **customer feedback on a regular and frequent basis** – and not dependent upon a contract or SOW.

Value 3: Customer Collaboration over Contract Negotiation (2 of 2)

- **Best contracts are NOT** those specifying requirements, schedule and cost.
  - Become meaningless shortly.
- **Far** better are **contracts that govern the way the development team and customer will work together**.
- Key is intense collaboration with customer and a contract that governed collaboration rather than details of scope and schedule
  - Details ideally **not** specified in contract.
  - Rather contracts could pay when a block passed customer's acceptance tests.
  - With frequent deliverables and feedback, acceptance tests never an issue.

Value 4: Responding to Change over Following a Plan

- **Our plans and the ability to respond to changes is critical!**
- **Course of a project cannot be predicted far into the future.**
  - **Too many variables; not many good ways at estimating cost.**
- **Tempting** to create a PERT or Ghant chart for whole project.
  - This does Not give novice managers control.
  - Can track individual tasks, compare to actual dates w/planned dates and react to discrepancies.
  - But the structure of the chart will degrade
  - **As developers gain knowledge of the system and as customer gains knowledge about their needs, some tasks will become unnecessary; others will be discovered and will be added to 'the list.'**
  - **In short, the plan will undergo changes in *shape*, not just dates.**

Value 4: Responding to Change over Following a Plan

- **Better planning strategy – make detailed plans for the next few weeks, very rough plans for the next few months, and extremely crude plans beyond that.**

- Need to know what we will be working on the next few weeks; roughly for the next few months; a vague idea what system will do after a year.
- **Only invest in a detailed plan for immediate tasks**; once plan is made, difficult to change due to momentum and commitment.
  - But rest of plan remains flexible. The lower resolution parts of the plan can be changed with relative ease.

Agile Principles (12)

- The following principles are those that differentiate agile processes from others.

Principle 1: Our Highest Priority is to **Satisfy the Customer** through **Early** and **Continuous Delivery** of Valuable Software

- Number of practices have significant impact upon quality of final system:
- 1. Strong **correlation** between **quality** and **early delivery of a partially functioning system.**
  - The less functional the initial delivery, the higher the quality of the final delivery.
- 2. Another strong **correlation** exists between **final quality** and **frequently deliveries of increasing functionality**.
  - The more frequent the deliveries, the higher the final quality.

- **Agile processes deliver early and often**.
  - Rudimentary system **first** followed by systems of **increasing functionality** every few weeks.
  - Customers may use these systems in production, or
  - May choose to review existing functionality and report on changes to be made.
  - Regardless, they must provide meaningful **feedback**.

Principle 2: Welcome Changing Requirements, even late in Development. Agile Processes harness change for the Customer's Competitive Advantage.

- This is a statement of **attitude**.
- Participants in an agile process are **not afraid** of change.
  - Requirement changes are good;
  - Mean team has learned more about what it will take to satisfy the market.
- Agile teams work to keep the **software structure flexible**, so requirement change impact is minimal.
- Moreso, the **principles of object oriented** design help us to maintain this kind of flexibility.

Principle 3: Deliver Working Software Frequently

(From a couple of weeks to a couple of months with a preference to the shorter time scale.

- We deliver working software.
  - **Deliver early and often**.
  - **Be not content** with delivering bundles of **documents, or plans**.
  - Don't count those as true deliverables.
- The **goal** of delivering software that satisfies the customer's needs.

Principle 4: Business People and Developers Must **Work Together Daily** throughout the Project.

- For agile projects, there must be **significant** and **frequent interaction** between the
  - customers,
  - developers, and
  - stakeholders.

An agile project must be **continuously guided**.

Principle 5: Build Projects around Motivated Individuals. (Give them the environment and support they need, and trust them to get the job done.)

- **An agile project has people the most important factor of success.**
  - All other factors, process, environment, management, etc., are considered to be second order effects, and are subject to change if they are having an adverse effect upon the people.

- **Example:** if the office environment is an obstacle to the team, **change the office environment**.
- If certain process steps are obstacles to the team, **change the process steps.**

Principle 6: The Most Efficient and Effective Method of Conveying Information to and within a Development Team is **face-to-face Communications**.

- In agile projects, developers ***talk*** to each other.
  - The **primary mode of communication is conversation**.
  - Documents may be created, but there is no attempt to capture all project information in writing.
- An agile project team **does not demand written** specs, written plans, or written designs.
  - They may create them if they perceive **an immediate and significant need**, but they are not the default.
  - The **default is conversation**.

Principle 7: Working Software is the Primary Measure of Progress

- Agile projects measure their progress by measuring the amount of **working software**.
  - Progress **not measured** by phase we are in, **or**
  - by the volume of produced documentation **or**
  - by the amount of code they have created.
- **Agile teams are 30% done when 30% of the necessary functionality is working.**

Principle 8: Agile Processes promote sustainable developmt The sponsors, developers, and users should be able to **maintain a constant pace** indefinitely.

- An agile project is not run like a 50 yard dash; it is run like a marathon.
  - The team does not take off at full speed and try to maintain that speed for the duration.
  - Rather they run at a fast, but **sustainable**, pace.

- Running too fast leads to burnout, shortcuts, and debacle.
- Agile teams pace themselves.
  - They don't allow themselves to get too tired.
  - They don't borrow tomorrow's energy to get a bit more done today.
  - They work at a **rate** that allows them to maintain the highest quality standards for the duration of the project.

Principle 9: Continuous Attention to Technical Excellence and Good Design enhances Agility.

- **High quality is the key to high speed**.
  - The way to go fast is to **keep the software as clean and robust as possible.**
  - Thus, all agile team-members are **committed** to producing only the **highest quality code** they can.
  - They do not make messes and then tell themselves they'll clean it up when they have more time.
  - **Do it right the <u>first</u> time!**

Principle 10: **Simplicity – the art of maximizing the amount of work not done – is essential.**

- Agile teams take the simplest path that is consistent with their goals.
  - They don't anticipate tomorrow's problems and try to defend against them today.
  - **Rather they do the simplest and highest quality work today, confident that it will be easy to change if and when tomorrows problems arise.**

Principle 11: **T**he Best Architectures, Requirements, and Designs emerge from **Self-Organizing Teams**

- An agile team is a self organizing team.
  - Responsibilities are **not handed to individual team members** from the outside.
  - Responsibilities are **communicated** to the team as a whole, and the **team determines** the best way to fulfill them.
- Agile team members work together on all project aspects.
  - Each is allowed input into the whole.
  - No single team member is responsible for the architecture, or the requirements, or the tests, etc.
  - The team shares those responsibilities and each team member has influence over them.

Principle 12: At regular Intervals, the **Team reflects** on how to become **more** effective, **then tunes and adjusts** its **behavior** accordingly.

- An agile team continually adjusts its organization, rules, conventions, relationships, etc.

- An agile team knows that its environment is continuously changing, and knows that they must change with that environment to remain agile.

Problems with agile methods

- It can be difficult to keep the interest of customers / users who are involved in the process.
- Team members may be unsuited to the intense involvement that characterizes agile methods.
- Prioritizing changes can be difficult where there are multiple stakeholders.
- Maintaining simplicity requires extra work.
- Contracts may be a problem as with other approaches to iterative development.
- Because of their focus on small, tightly-integrated teams, there are problems in scaling agile methods to large systems.
- Less emphasis on documentation - harder to maintain when you get a new team for maintenance

Conclusions

- The professional goal of every software engineer, and every development team, is to deliver the highest possible value to our employers and customers.
  - And yet, our projects fail, or fail to deliver value, at a dismaying rate.
- Though well intentioned, the **upward spiral of process inflation** is culpable for at least some of this failure.
- The principles and values of agile software development were formed as a way
  - to help teams break the cycle of process inflation, and
  - to focus on simple techniques for reaching their goals.

- There are many agile processes to choose from. These include
  - SCRUM,
  - Crystal,
  - Feature Driven Development (FDD),
  - Adaptive Software Development (ADP), and most significantly,
  - Extreme Programming (XP).
  - Others…