Limitations of plan-based

Waterfall model (and others) are heavy top-down approach. Do all the planning, then all the work, then all the testing. This is expensive, slow, inflexible.

Many limitations

- Depends on a lot of documentation to be successful
- Lots of waiting times between different teams
- Expensive to fix buggy code
- Barely any freedom for developers
- Hard to manage change during a project

The agile manifesto

- Individuals and interactions over process and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan.

Original agile principles

- Highest priority is to satisfy the customer trough **early and continuous delivery**
- Welcome changing requirements, even late in development. Agile processes **harness change for the customer's competitive advantage**
- Deliver working software frequently, from a couple of weeks to a couple of months, with a **preference to the shorter time scale**
- **Business people and developers must work together daily** trough the project.
- Build projects around motivated individuals. Give them the **environment and support** they need and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is **face-to-face conversation**
- **Working software is primary measure of progress**
- Agile process promote **sustainable development**. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- **Continuous attention** to technical excellence and good design enhances agility.
- Simplicity – the art of **maximizing the amount of work not done** is essential
- The best architectures, requirements and designs **emerge from self-organizing teams.**
- At regular intervals, **the team reflects on how to become for effective**, then tunes and adjusts its behaviour accordingly

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

Overall methodology & each stage

The overall approach

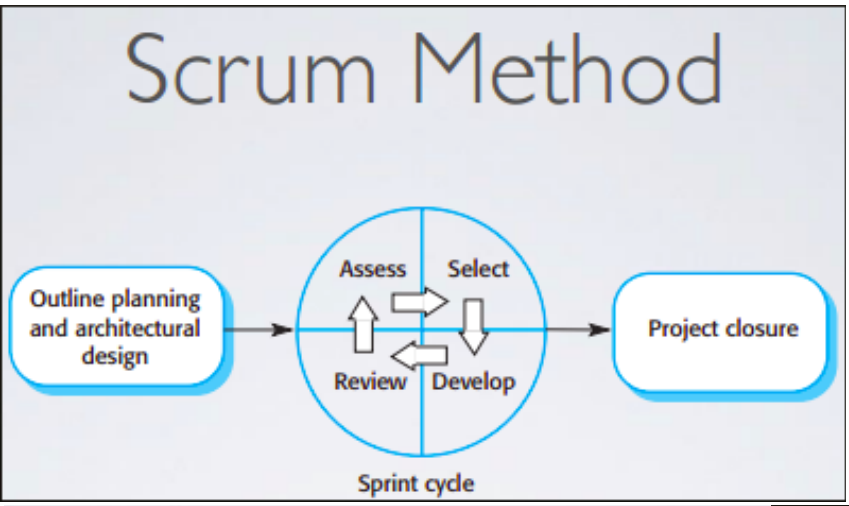
- The spiral model is already an adaptive approach
- But the agile principles apply to **adapt your approach**

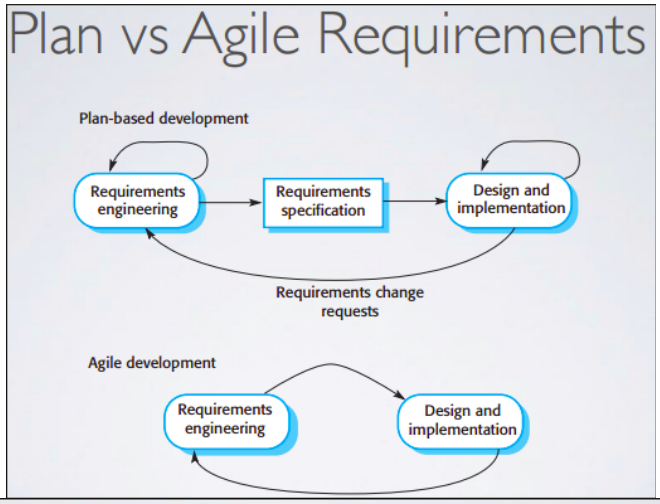
Each stage and activity

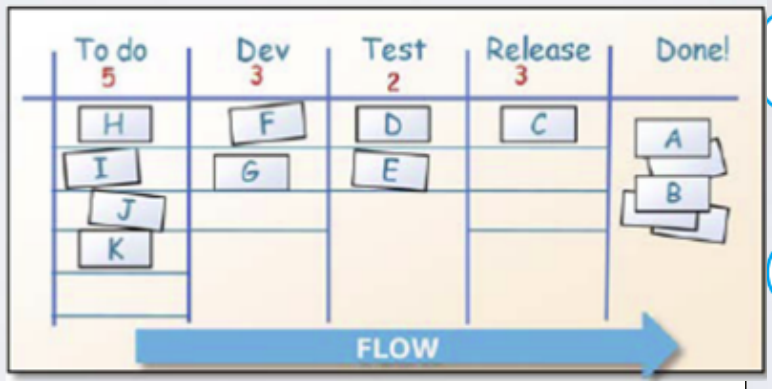
- agile requirements : **user stories** not requirement documents
- agile design : **prototypes** not specification documents
- agile implementation : test driven paired development
- agile testing : **testing is not a phase**, Agile team tests continuously and continuous testing is the only way to ensure continuous progress.

Scrum vs Kanban vs XP(Extreme Programming)s

- All three are designed to uphold the Agile principles, just have different focuses
- Scrum is about people and working software
- Kanban is about responding to change
- XP is about software quality and team effectiveness







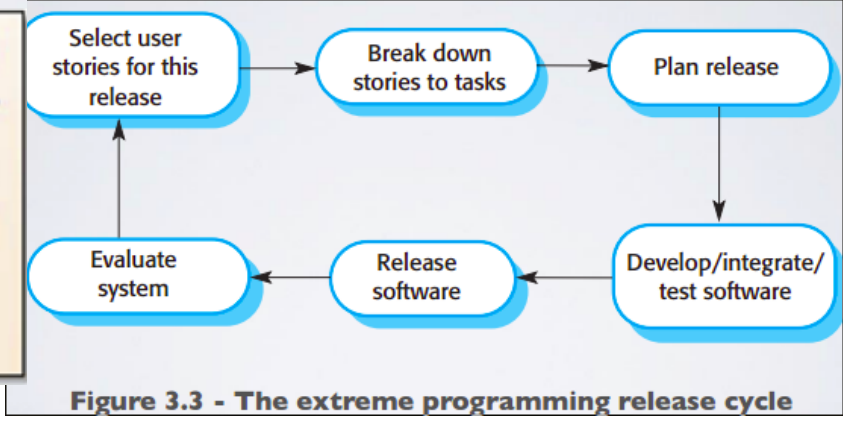


Figure 3.3 - The extreme programming release cycle

Extreme programming

XP is a fast ‘extreme approach to iterative development’

- New versions may be build everyday, or more often
- Increments are delivered to customers around every 2 weeks
- All tests are applied before a build is accepted
- Builds are only released when they pass all tests
- It takes many of the “faster techniques” that we have discussed from every SE stage.

Extreme programming principles

- Customer involvement means full-time customer engagement with the team.
- Incremental development is supported through small, frequent system releases.
- People not process through pair programming, collective ownership and a process that avoids long working hours.
- Change supported through regular system releases.
- Maintaining simplicity through constant re-factoring of code

Extreme programming requirements

- User stories
  - created with the customer
  - prioritised with the customer
  - selected for current iteration
  - broken down into tasks
- Design
  - should be simple
  - is fast (rapid throw-away prototyping)
  - focusing on selected user stories rather than predicting future versions
  - are discussed with the customers as early deliverables
- Implementation
  - done in pared-programming
  - shared code responsibility
  - is done to resolve User stories
  - progress is measured by % of user stories/tasks that work
  - Builds are frequent/daily and are tested to meet User Stories
  - Refractoring is always performed if better code is conceived
- Testing
  - Test-driven development
  - Incremental tests from user stories
  - acceptance tests designed with customers
  - all tests automatically run for every build

XP focuses on

- iterative development – stages of functionality
- percentage of user stories working and delivered
- fast lightweight activities at each SE stage
- close face-to-face collaboration
- close customer involvement

Scrum method

1. Plan and outline what you will achieve
2. Set a series of `sprint cycles` doing the work
3. Document the current phase

Sprint cycles

- A sprint is fixed time period to achieve the current plan
- Sprint cycles are intermittent points. What will be achieved by the next release point
- Sprint involve continuous group communication (daily meetings)
- Sprints are managed by a Scrum Master

Kanban method

- Rather than what will we achieve in this cycle, Kanban is what shall we build next (passing jobs on). The work is driven by an updating priority list
- You use a kanban board to manage current work
- All features to build are kept in a list
  - items and their priority can be updated at any time
- When 1 is completed, it's integrated into product and released, then a new one is picked
- Team effectiveness is managed by limiting the numbers in columns of the board
- Release cycles is a measure of time taken to build features rather than a prescription of iterations in Scrum

Combining methods

- Clearly these techniques aren't mutually exclusive
  - you can do paired programming in Scrum
  - you can use TDD in Kanban,Scrum, or XP
- Naturally these mixtures already exist
  - Scrumban
- Companies tend to pick a technique that works
  - based on team size, skill set, project clarity, etc.

Comparisons

More formal	**More agile**
Waterfall, V-Model	Spiral, Iterative Models
Requirements Tables	User stories, personas
Detailed UML/Specs	Rapid prototyping
Separate Teams	Scrum/Kaban/XP
Integration testing phase	TDD,CI, Paired coding
Formal user testin	Frequent client interaction

Agile methods

Especially valuable for :

- Small or medium sized projects for release/sale
- custom system development with an involved customer

But

- It’s not always easy to have an involved customer long term
- Team members don’t always have the right personalities
- It’s hard to prioritise change and too much can still be bad
- Simplification takes time and can be forgotten
- Companies can see agile methods as high risk
- Harder to plan/budget/contract for an agile project

Good methods for being Agile

- Agile methods are practised techniques for being agile
- There are guidelines for ‘How you should do ‘ agile
- The point of Agile is to **be agile**

When agile or plan	
Is it important to have a very detailed specification and design before moving to implementation?	Plan-driven
Is an incremental delivery strategy, where you deliver the software to customers and get rapid feedback from them, realistic?	Agile
Is the software small enough to be built by a small co-located team? Or can be broken down so?	Agile
Does the system require lots of careful analysis before the implementation can begin?	Plan-driven
What is the expected system lifetime? Will it be revised or update regularly, needing documentation?	Plan-driven
Are your tools/IDEs good for Agile development (keeping track of change)?	Agile
Do you need clear documentation to communicate between project partners, or remote colleagues?	Plan-driven
Is the current culture, and all other projects, very process oriented?	Plan-driven
Does your team have all the right requirements, design, testing, skill-sets in it?	Agile
Is your project or system subject to external regulation or formal (e.g. safety critical) standards?	Plan-driven

Following a strict Agile method misses the point about being Agile

Lean SE

- Eliminate waste – surplus tasks, admin, processes
- Amplify learning – get answers as soon as possible
- Decide as late as possible – wait until needed
- Deliver as fast as possible – work done with this in mind
- Empower the team – developers know the best way to dev
- Build integrity in – maintain quality, including refractoring
- See the whole – avoid doing just your bit