7: Software Development Paradigms

Monday 6th of November, 2017

Intro

A paradigm is a way of thinking, or a lens through which to view the world. It represents a set of assumptions.

For example, in programming there are functional (e.g. Haskell) and object-oriented (e.g. Java) paradigms.

In software development, there are different paradigms:

- Rational Model (Waterfall)
    - Software development is best seen as a sequence of steps.
    - Requirements are assumed not to change.
- Empirical Model (iterative and incremental development)
    - Start small, add features iteratively and incrementally
    - Assumes requirements will change

The Traditional Approach

Software development is easy on your own. When scaling up, issues arise:

- Need for communication
    - Who knows what, and who needs to know what?
- Need for co-ordination
    - Who does what, and when?
- Need for systematic approach

Waterfall

The original waterfall model was presented as a bad approach, with proposed adjustments to make it more suitable for large-scale project.

Basically, the adjustments proposed to do it twice – try and do the whole approach once, very quickly, and then use what you learned to repeat the approach and do it better.

Document Heavy

The waterfall model is document-heavy – documents are produced at every step.

Why Was It Used?

Initially, computers were expensive and scarce resources – it made sense to carefully plan the access by programmers.

The later in the lifecycle that defects were detected, the more expensive it was to fix them.

V-Model

The v-model is derived from the waterfall model. There's an increased focus on verification and validation, and it's shaped like a V.

Problems

1. Requirements change
    - due to the long time between requirements and delivery, customers change their minds
    - This causes rework.
2. No possibility to get early feedback from customers
    - Delivered solution may not be satisfactory.
    - Want running code as soon as possible.
3. Hard to completely specify a problem.
4. Testing is left till last, which causes delays in delivery when unexpected defects are found
    - Fixing defects late in the process causes a lot of rework upstream.
5. Developer Frustration

    - Working on a project for a long time without visible progress (runnable software) decreases developers' job satisfaction.

6. Document-heavy
    - Conventional approaches produce a lot of documentation that is not necessarily useful (and goes out of date quickly)

These problems apply to everything derived from the waterfall model.

Rapid Application Development

This is one of the earliest agile methods. It's not radically new, but leverages currently available tools and techniques.

Mostly it's management techniques [...]

Principles

1. Active User Involvement
    - The gap between developers and users causes delays in conventional approaches.
    - Users know business needs best.
    - Users are heterogeneous – different background and expertise.
    - Get users involved and committed.
2. Small Empowered Teams
    - Creative dynamic through skilled and highly-motivated teams.
    - Eliminate bureaucratic and delayed communication and decision making.
    - Small teams and fewer communication channels.
    - Roles rather than individuals. Assign roles to different people. [...]
3. Frequency Delivery of Working Products
    - Time-boxing: 2-6 months
        - if you deliver working products often, it's more likely that the next one will work
    - Simplified project management
    - Focus on product (working software) rather than process
4. Iterative Development and Prototyping

- Initial delivery of sub-systems, followed by refinement.
- Systems evolve and are never complete.
  - New requirements emerge, can't know them all in advance.
- Requirements specification is a learning process.
- Prototyping: working model of the system.
- Recognition is easier than recall for users, so it's good to get feedback from them often to point you in the right direction.

5. CASE Tools
   - CASE: Computer-assisted systems engineering
   - Need to eliminate or automate routine, time-consuming tasks
     - e.g. code generation, version control, bug tracking, documentation, testing, continuous integration, project management

Emergence of Agile Methods

The Agile Manifesto

Individuals and interactions over processes and tools.

Working software over comprehensive documentation.

Customer collaboration over contract negotiation.

Responding to change over following a plan.

Principles

[...]

Agile Methods

Already existing methods were labelled "agile" – Extreme Programming, Scrum, etc.

Scrum

Scrum is a particular agile method. It was retrospectively labelled as agile when that phrase was coined.

Overview

Traditional approaches are kind of like a relay race – different stages completed in order, possibly by different people.

Instead, scrum takes a more holistic/rugby-like approach, where the team [...].

Empirical Approach

Relies on what you know.

Requires:

- Transparency: process must be visible
- Inspection: make sure everything's still ok
- Adaptation: adjust when needed

Scrum Roles

Product Owner

Responsible for maximising value of the product.

Plays the role of the customer.

Sole person responsible for maintaining, prioritising, and communicating product backlog.

Product owner must be an individual.

Scrum Master

Responsible for enacting scrum values.

Removes impediments to get work done.

Ensures that the team is functional and productive.

Shields the team from external interference.

More like a coach than a boss/project manager. Team is self-organising and self-managing.

Scrum/Development Team

Typically 5-9 people.

Teams are self-organising and cross-functional. Team has all the necessary skills and isn't dependent on anyone else. Every member of the team is on an equal footing.

Team decides how to turn backlog items into product increments.

Members should be full-time, and membership should only change between sprints.

Scrum Ceremonies / Events

The Sprint

A time-boxed period of 1-4 weeks (fixed deadline).

During a sprint, no changes are allowed.

Scope may be re-negotiated with the product owner – if the dev team feels there is too much or too little work.

The sprint must result in a shippable product (though this won't necessarily be deployed).

Sprints may be cancelled when Sprint goals become obsolete (e.g. changing business focus). This is rare.

Sprint Planning

Two main questions:

1. What to deliver in this sprint?
2. How to achieve this?

Which items in the product backlog are delivered in this sprint? (Decided just by the team.)

For a 4-week sprint, there's a maximum of 8 hours planning.

Once committed to a plan, no changes are allowed unless the sprint is cancelled.

Daily Scrum

15 minute time-boxed event, held at the same time and same place every day. Meant to be quick, no-fuss.

Aim to synchronise activities, and create a plan for the next 24 hours.

Three key questions:

1. What have I done towards the goal?
2. What will I do today towards the goal?
3. What's blocking me from reaching the goal?

Scrum master ensures the daily scrum takes place.

It's not meant for problem-solving – issues get recorded and resolved after the meeting.

Sprint Review

Happens after the sprint is done. Maximum 4-hour time-boxed meeting for the team and product owner to inspect the software product and adapt in the next sprint.

Results should be accepted only if they fill the definition of done (e.g. code is integrated, fully tested, documented, and potentially shippable). Partially done work or defective work shouldn't be taken into account – gives a false sense of progress. Need to define for a project what counts as done.

Sprint Retrospective

After sprint review, focus on inspecting and adapting the process, rather than the product.

Key questions:

- What's working well?
- What could work better?

3-hour time-boxed meeting.

Should be evidence-based retrospectives – hard data and underlying causes.

Scrum Artefacts

Product Backlog

Evolving, prioritised queue of everything needed in the product. The single definitive source for features, functions, requirements, enhancements, and fixes.

The product owner is responsible for maintaining, ordering, clarifying, and communicating this queue.

Backlog items get refined/clarified over time.

Sprint Backlog

Items from a product backlog selected for a sprint, plus the plan for implementation – a design that is discussed and agree upon by all.

Potentially Shippable Product / Product Increment

Every sprint ends with a potentially shippable product, which may or may not be released.

It's the sum of all product backlog items completed during a sprint, plus all previous increments.

Must be usable by a customer.

Scrum in Action

Before the Sprint

A project is a series of sprints, where each sprint is 1-4 weeks (varies by project).

Each sprint leads to a potentially shippable product, and has a sprint goal which indicates the overall objective.

Start of a Project

Vision statement: a description of the project goals – this helps to keep the team focused on what it important for the organisation.

Product roadmap: an initial visual timeline of major product features, created by the product owner.

[...]

Product Backlog

All features are collected and sorted based on their priority.

During development, new features, feature changes, and defects may emerge, and will be added to the backlog.

Once the product backlog has sufficient features, sprints can start – you don't need to wait for the complete set.

Product Backlog Items can be represented in different ways. In agile development, it's common to define them as user stories. However, there may be other forms, for example bugs may link to an issue tracker. User stories may also be from the perspectives of developers, as well as end-users.

Effort Estimation

You need to estimate the effort involved in backlog items. This is commonly done using a relative indication called story points. Assign fibonacci numbers: Feature A has 5, Feature B has 2, Feature C has 3.

Planning Poker

After discussing a particular backlog item, each estimator picks a card representing the number of effort points. If there's consensus, then that value is used. If there isn't, the highest and lowest scorers justify, and then the process is repeated until consensus is reached.

Velocity

For each sprint, keep track of how many story points you got done. Then you can choose how many story points should be included in the backlog based on your velocity.

Sprint Planning

Select a number of backlog items to create a sprint backlog – keep 10% of time as a buffer.

Stories are broken down into tasks with time estimates, and each task should take no longer than 8 hours.

During the Sprint

Daily scrum to discuss activities and synchronise.

Work isn't assigned – team members pick tasks (agile methods emphasise self-organising teams).

Progress is updated daily.

Definition of Done

There needs to be an agreed definition of done – using a checklist is an unambiguous way to make sure that a task is done.

The DoD may vary per project or per sprint.

Burndown Chart

Indicates the number of effort hours remaining for the current sprint.

Summary

Scrum is intended for:

- small projects

- co-located teams
- non-critical systems

However, Scrum has been successfully adopted outside of these restricted domains by tailoring the method:

- distributed scrum
- scrum of scrums
- additional roles, ceremonies, and artefacts