Agile Manifesto & XP

Chapter 3.1-3.3

CMPT 276

Based on slides from Software Engineering 9th ed, Sommerville.

Topics

1) What is Agile trying to do?
2) How to choose plan-driven vs Agile?
3) What practices go into Agile (XP) development?
4) How to write tests while writing new code?

Rapid software development

- Rapid development and delivery is often the...

  most often requirement

  for a software system.
  - Businesses change fast; practically impossible to have stable software requirements.
  - Software has to evolve quickly to keep up.
- Agile aims at rapid software development:
  - Interleave specification, design and implementation.
  - Incrementally developed with user evaluating each version.

Agile methods

- Inspired by dissatisfaction with... **overheads** in plan-driven software methods.
- Agile Methods
  - Focus on the **code** rather than the **plan and design**
  - Based on iterative approach to software development;
  - Intended to deliver working software quickly and evolve it quickly to meet changing requirements.
- Aim of Agile Methods
  - Reduce overheads in the software process
    e.g... **limit documentation**
  - Respond quickly to changing requirements without excessive rework.

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

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **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.

Agile manifesto

Signed by Kent Beck, Robert Martin, Martin Fowler, and 14 other founders of Agile development.

Scrum & Agile Manifesto Values

- Agile Manifesto Values can be summarized by..

  inspect and adapt

  - Perceive current state of the project;
  - Adjusting the.. **product** and.. **process** to respond.

- In-Class Exercise:
  For each Agile Manifesto Value:
  - Explain how it is “Inspect and Adapt”.
  - Explain how scrum achieves the value.

Inspect & Adapt in Scrum

- Scrum is about learning through inspecting and adapting:

    1. Daily stand-up:.. **for the sprint**
       keep on track

    2. Sprint demo:.. **for the product**
       ensure most valuable features being added

    3. Retrospective:.. **for the team**
       continuous improvement to the team

Principles of agile methods

Principle	Description
	Customer is closely involved throughout development. They provide and prioritize new system requirements and evaluate the iterations of the system.
Incremental delivery	Software developed in increments. Customer specifies requirements to be included in each increment.
	Recognize and exploit development team's skills. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect system requirements to change; so design the system to accommodate these changes.
	Focus on simplicity in both the software being developed and in the... Actively work to eliminate complexity from the system.

Plan-driven vs Agile Methods

Agile Applicability

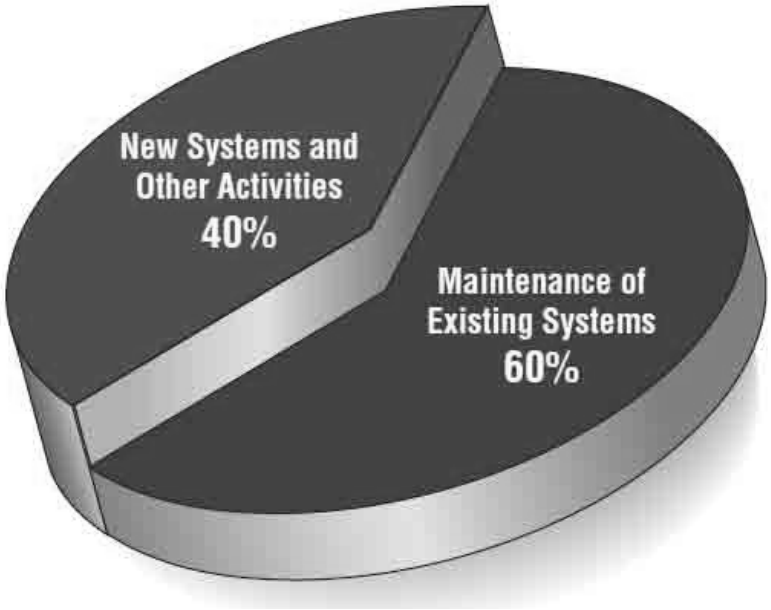
- Agile applicable for:
  - Developing small or medium-sized product
  - Requires
    1. Customer willing to be involved in development process
    2. Few.. **external regulations** affect the software.

- Problems with Agile:
  - May not.. **scale to large systems**
    focus on small, tightly-integrated teams.
  - Intense interactions may not suit development team.
  - Prioritizing change is hard with... **multiple stakeholders**
  - Maintaining simplicity requires extra work.

Agile & Maintenance

- Maintenance:
  - More $ on maintenance than on initial development.
  - To succeed, Agile has to support both!

- Estimated at 60% +/- time spent on maintenance.



Agile & Maintenance cont.

- Lacking Formal Documentation
  - *Good* documentation supports maintenance.
    - BDUF has docs, but.. **docs get out of date fast**
    - .. **Agile focuses on clear code**
      expressing the design which is better maintained.
  - Agile can create required documents as needed.
- Changing needs in Maintenance
  - Agile good at accommodating change.
  - Challenging if.. **original developers leave team**
    - Process is less formal, so may be poorly understood by new group of devs.

Choosing agile vs plan-driven

- Most projects use elements of both plan-driven and agile processes depending on:
  - Need detailed specification and design before moving to implementation?

Plan	Agile

  - Is an incremental delivery strategy with rapid feedback realistic?

Plan	Agile

  - Is it a small-medium size system being developed?
    - Agile most effective with small team who can...

Plan	Agile

  - Does it require a lot of analysis before implementation?
    - e.g. real-time system with complex timing requirements.

Plan	Agile

Choosing agile vs plan-driven

- Is it expected to be long lived where system maintainers need design docs?
- Are good tools available to support development?
- Is team spread out/outsourced and needs design docs to communicate?
- Does team have a plan-based development culture (engineering)?
- Is detailed documentation required for external regulatory approval?

Plan	Agile

Plan	Agile

Plan	Agile

Plan	Agile

Plan	Agile

Extreme Programming (XP)
Another Agile Process

Created by Kent Beck (1999)

Extreme Programming (XP)

- Extreme Programming (XP)
  takes best software dev. practices to an extreme
  - Short iterations;
    deliver working software often (<2 weeks)
  - Frequent communication with customer
  - All tests automated and pass for each change
  - Based on 12 XP Practices
- Extreme Example
  - Normal: Code reviews are good.
  - Extreme: ..

Select XP Practices

Practice	Description
small releases	Start small: First develop minimal useful set of functions which deliver business value. Release often: Releases are frequent and incrementally add functionality.
simple design	Design is only done to support current requirements, not any possible future ones.
test-driven development	Automated unit tests are written for a class before the class is written.
refactoring	Code kept simple and maintainable by continuous refactoring by all developers.

Select XP Practices cont.

Practice	Description
pair programming	Developers work in pairs, always checking each other's work and providing support.
collective ownership	All developers work on all parts of the code. Shared responsibility for the code, and no one developer has all knowledge about an area.
continuous integration	Changes integrated into system as soon as the are completed.
sustainable pace	Large amounts of overtime are discouraged: would compromise productivity and code quality.
on-site customer	Customer representative (user) is a full time member of the development team: brings the team requirements and priorities.

Pair programming

- Developers work in pairs..

  sitting together to develop code

  - Pairs change so everyone works together.

- Fosters **common ownership** of code and spreads knowledge across the team.
  - Reduce problem when key developers leave.
  - No one person blamed for bugs.

- Informal review process:

  each line of code instantly reviewed

- Encourages refactoring:
  - Whole team gets benefit of clean code.

- Productivity with P.P. ≈ two people working independently.

XP and change

- Conventional wisdom:.. **design for change**
  - It is worth spending time and effort anticipating changes as this reduces costs later in the life cycle.

- XP's view:

  changes cannot be reliably anticipated
  - XP uses refactoring to constantly improve code.
  - This makes changes easier when they have to be implemented.

Refactoring

- Developers look for possible code improvements and make these improvements..
  - This improves the understandability of the code and reduces the need for documentation.
- Changes are easier to make because..
  - However, some changes require architecture refactoring and this is much more expensive.

Examples of refactoring

- Refactoring Examples
  - **renaming method** to make its purpose clearer.
  - **extract method** to make a long function shorter or reduce duplicate code.
  - **extract class** to split a class which does 2 things into two classes.
- Single Responsibility Principle (SRP)
  - **each class should do one thing**
  - It should have only one reason to change.

XP Testing:
Test driven development

Overview of testing in XP

- Testing is Central to XP
  -
- Features of XP Testing
  - Test-driven development (TDD).
  - User involvement in test development and validation.

Test automation

-

  before the task is implemented.
  - Automated testing framework (JUnit) runs stand-alone tests which simulate input and check result.
- Tests run whenever new functionality added.
  - Tests can be run quickly and easily.
  - Catch problems immediately.
- Automated verification
  - Gives developers the confidence and security of knowing nothing broke.
  -

Test-driven development

- Test-driven development (TDD):
- Idea:
  - Write a test that fails, add code to make it pass.

Process

1. Find some small functionality to implement
   - 2-3 lines of code.
2. Write the JUnit test for it.
3. Run all tests
   - Will give you...
4. Write code to fix the bug.
5. Run all tests, hopefully gives you a...
   - Refactor as required.
6. Goto 1.

TDD & Pair Programming Game

- Take turns while pairing. Valid moves are:
  - If project in “Red”:
    - Make it “Green”:..
  - If project in “Green”:
    - ..
    - ..
    - ..
- TDD Pair Programming Game Demo w/ Student
  - In Android Studio, create Tail.java; code & JUnit: public static int lengthDescendingTail(int[] data);

Benefits of TDD

- ..
  - Each line of code written to satisfy a test, so all lines are tested.
- ..
  - Run old tests after new changes to prove you have not broken previous features.
  - A regression test suite is continually developed as a program is developed.
- ..
  - When a test fails, it should be obvious where the problem lies (the new stuff!).
- ..
  - The tests themselves are a form of documentation that describe what the code should be doing.

Summary

- Agile: incremental development focused on
  - rapid development,
  - frequent releases,
  - reducing process overheads,
  - producing high-quality code.
- Agile vs plan-driven depends on type of software
- XP Practices: good practices done to the extreme.
  - TDD to ensure well tested code