Software Development Tools Lecture 2

COMP220/COMP285

Sebastian Coope

Programming Methodologies

These slides are mainly based on "*Java Tools for Extreme Programming*" – R.Hightower & N.Lesiecki. Wiley, 2002

Topics

- Two kinds of **programming methodologies**
  - ***traditional***
  - ***agile***

  We will concentrate on
- **eXtreme Programming (XP)** methodology
  - example of an ***agile methodology*** of most interest to us

Software Development Methodologies

Software Development Methodology is

a collection of procedures, techniques, principles and tools that help developers to build computer system

Software development methodologies

There are two main approaches to development methodologies:

- Traditional ***monumental*** or ***waterfall*** methodologies
- ***Agile*** or ***lightweight*** methodologies

Traditional methodologies

- **Rigid:**
  - first a *complete functional specification,*
  - then software *development process* with several *distinct waterfall-like phases*
- **Problems:**
  - difficult to adapt to *changing customer requirements*
  - design *errors* are
    - - *hard to detect* and
    - - *expensive to correct*

Waterfall issues

- What is customer doesn’t like the end product
- What if requirements start to change?
- What if project runs out of time/money?
- How is risk managed?
- How is QA managed (at the end !!)

Agile methodologies

Agility in a software development means

- *adaptability*
- ability to *respond quickly to change* in environment
- *eliminate surprises* from changed requirements
- Risk reduction
- Less chance of validation errors

Agile methodologies

- emphasizes an ***iterative*** process:
  - *build* some well-defined set of features
  - *repeat* with another set of features, etc.
- value ***customer involvement*** (quick feedback)
- ***code-centric***, i.e.
  - recognize the value in *documentation* and *modelling*
  - but realize that it is *not as important as the software itself*

Self documenting code

- Using long meaningful names
  - accountBalance
  - accountBalanceInPence
- Comments
  - What to change to change code behaviour
    - static final int RETRY_LIMIT=3; // Change this value if you want to change the maximum number of times an incorrect PIN can be entered
  - TODO
    - Any areas that can be improved or require completion
    - TODO … check for stolen cards and credit risk

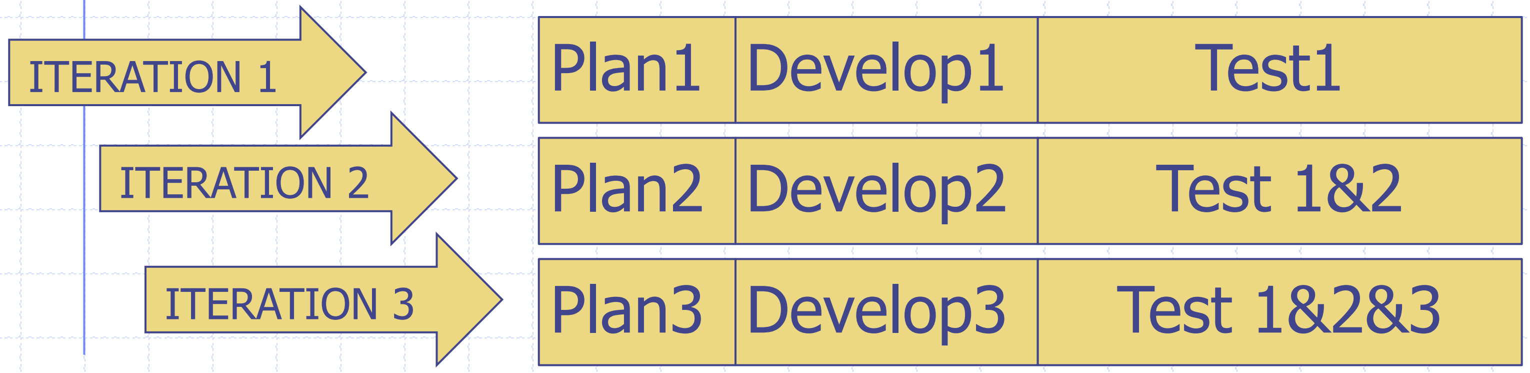
Testing in agile methodologies

- Software development is
  - a ***mix*** *of* ***art*** *and* ***engineering***.
- The only way to *validate* software is *through* ***testing***
- All agile methodologies ***emphasize testing***
- ***Testing can be***
  - ***Functional (specific yes or no tests based on functional specification)***
  - ***Non-functional (stress testing, usability, security testing etc.)***

SCRUM

- Agile approach
- Each iteration of software development called a sprint
- Each sprint delivers working code or partial product
- Each phase requires a set of tests
- Testing is integrated

SCRUM


ITERATION 1
ITERATION 2
ITERATION 3
Plan1
Develop1
Test1
Plan2
Develop2
Test 1&2
Plan3
Develop3
Test 1&2&3

SCRUM phases

- Specification at start
- Then each development phase can be
  - Specification , Design, Coding
- Each iteration tests
  - New functions
  - All old functions (regressive)
- Testing is extensive, must not be burdensome

Testing-driven development

- *Put* ***testing*** ***first*** in the development process
- ***Before implementing*** a piece of code such as a **Java** *method*, start ***writing down a test*** which this *method* should pass.
- ***Test is like a goal*** which you want ***to achieve***
- ***First state a goal, then do steps*** to that goal
- Goals may be quite ***small***, ***intermediate,*** or ***final***
- ***Test-driven*** style of programming!

Why write test first

- Test is based on the specification and not the code, not assumptions based on source code
- If testing is done second, testing might be skipped
- Makes the developer analyse the requirements
  - Requirements might be wrong or ambiguous
- Produces more testable code
- Keeps the code simpler/shorter (only target is to pass the test)
  - Stops the code being over-engineered
  - But note simple goal .. conflicts with non-functional code requirements, code quality

eXtreme Programming

Most general features of XP:

- one of the most unique and controversial approaches
- *agile* or *lightweight* methodology
- *human-centric* development philosophy

Overview of the **XP** methodolog

- ***focuses on coding*** as the main task
- regards ***continuous*** (*) and ***automated testing*** as central to the activity of software development
- ***refactoring*** (*) is a core XP practice
- ***continuous integration*** (*)
- one of XP's radical ideas is that

  design should evolve and grow

  through the project

Continuous testing validates that the software works and meets the customer's requirements

Refactoring: changing existing code for simplicity, clarity and/or feature addition

Continuous integration means building copy of the system so far several times per day

Some Essential of **12 Practices** of **XP**

1. Testing

- key practice to XP
- how will you *know if a feature works* if you do not test?
- how will you *know if a feature* *<u>still</u>* *works* after you re-factor, unless you re-test?
- should be *automated*
  - so you can have the *confidence* and *courage* to change the code and re-factor it without breaking the system!

Some Essential of **12 Practices of XP**

1. Testing (cont.)

- the *opposite of waterfall* development
- keeps *code fluid*
- **JUnit** and its “friends” (versions or analogues of **JUnit**) will help to

  automate testing
- *everything* that can potentially break *must have a test*

Some Essential of **12 Practices of XP**

2. Continuous integration

- a crucial concept
- means *building and testing a complete copy of the system several times per day*, including all the latest changes
- *why wait until the end of a project* to see if all the pieces of the system will work together?
- *the longer* integration bugs survive, *the harder* they are to exterminate

Some Essential of **12 Practices of XP**

2. Continuous integration (cont.)

- benefits from *using good software tools*
- **Ant** (integrated with **JUnit**) can help to ***automate***

  the build,

  distribution, and

  deploy processes
- see the paper by *Fowler* (and *Foemmel*) in www.martinfowler.com/articles/continuousIntegration.html

Some Essential of **12 Practices of XP**

3. Refactoring

- a technique for
  - *restructuring the internal structure of code*
  - *without changing* its external *behaviour*
  - or with adding *new features*
- enables developers to
  - *add features while keeping the code simple*
- each *refactoring transformation*
  - *does little,*
  - so, it is *less likely to go wrong,*
  - but a *sequence of transformations can produce a significant restructuring*
- the *design is improved* through the refactoring

Some Essential of **12 Practices of XP**

3. Refactoring (cont.)

- ***relies on testing*** which validates that the code is still functioning
- testing makes *refactoring possible*
- *automated* unit-level tests will give you
  - the *courage* to re-factor and
  - keep the code *simple* and *expressive*

Further Practices of **XP**

4. ***Planning* game** (to discuss *scope of the current iteration, priority of features* , etc.)
5. **40-hour work week**
6. ***Small* releases** (*feedback, testing, cont. integration*)
7. ***Simple* design** (*keeping also the code simple*)
8. **Pair programming** (improves *communication* and mutual understanding among team members, *learning*)
9. **Collective ownership** (*no crucial dependence* on one developer)
10. **On-Site customer** (quick feedback, etc.)
11. **Metaphor (***common language* for developers and customer**)**
12. **Coding standards** (*understand one another's code*)

(See more detail in the **XP Book**)

XP and SCRUM

- Can and do work well together
- XP
  - More about programming/testing practise and small scale organisation.. TDD, re-factoring, continuous integration
- SCRUM
  - Project organisation and development life-cycle

Some other principles

- KISS (General engineering)
  - Keep it Simple Stupid
- YAGNI (XP)
  - **You ain’t going to need it**
  - **So don’t**
    - **Add functions not in spec**
    - **Add too much future proofing**

Problems with XP/Agile

- YAGNI/KISS
  - Might discourages code flexibility
    - Image today we have English as locale next year we want Mandarin, Spanish and Mexican
    - Answer .. Put flexibility into requirements!
  - Might discourage re-use
- Hard to develop a complete schedule
  - Too elastic?
    - Timebox?

Summary (XP)

- **XP** is *lightweight* methodology that focused on *coding as a main task*.
- **XP** *encourages* *full integration daily* (**Ant**)
- **XP** is a *test-driven* methodology (**JUnit**, etc.)

XP - Conclusion

- You can adopt in your practice the whole or only a part of **XP *methodology*** (considered here only fragmentary)… think of Group Software Project
- Anyway, you will probably ***benefit from*** the related ***software development tools*** and ***techniques*** we will consider in the rest of this course

- Time-to time we will need to return to some of these methodological questions

Our aims in this course

- To explore **XP** ***methodology***
  - by providing an insight into the ***tools*** for *building*, *testing*, and *deploying* code
  - by demonstrating how to ***use all these tools together***