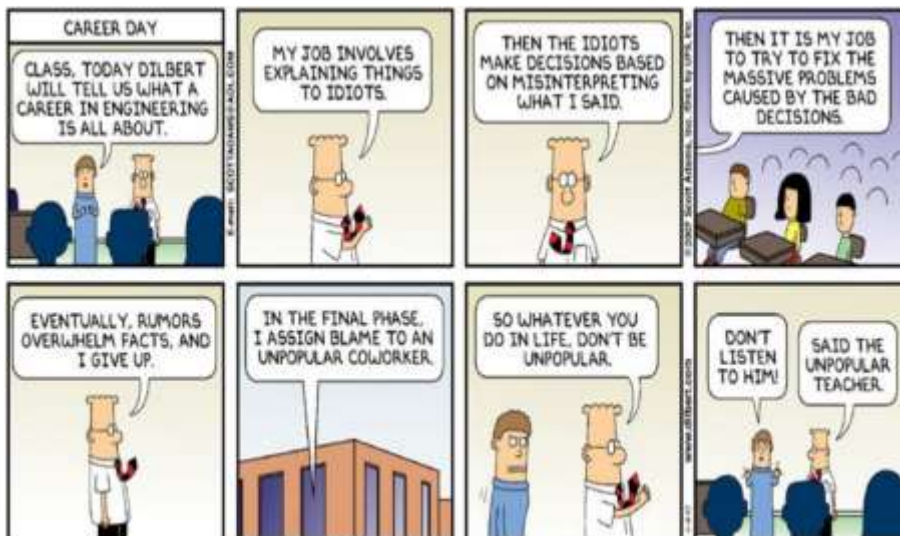


Extreme Programming XP

Another Agile Methodology



XP

- *Extreme Programming is a discipline of software development with values of simplicity, communication, feedback and courage. We focus on the roles of customer, manager, and programmer and accord key rights and responsibilities to those in those roles*

■ [Kent Beck, Ward Cunningham, Martin Fowler, Ron Jeffries]

XP Timeline

- **1986-1996**, Beck and Cunningham developed a large set of best practices that were succinctly captured in:-
 - "Pattern Languages of Program Design 2", Vlissides, Coplien, Kerth, Addison and Wesley, Reading MA, 1996 (Ch. 23)
- **1989-1992**
 - William F. Opdyke, "Refactoring Object-Oriented Frameworks". PhD Thesis, University of Illinois at Urbana-Champaign. Also available as Technical Report UIUCDCS-R-92-1759, Department of Computer Science, University of Illinois at Urbana-Champaign.
- **1996**
 - "Smalltalk Best Practices Patterns", Kent Beck
- **1999**
 - Refactoring: Improving the Design of Existing Code by Martin Fowler, Kent Beck (Contributor), John Brant (Contributor), William Opdyke, Don Roberts, Addison and Wesley, Reading MA, 1999.
- **1997**
 - "JUnit", Erich Gamma and Kent Beck, OOPSLA, Atlanta 1997

XP Roles

- ❑ **Customer role**
 - *Defines,*
 - What will deliver business value (scope)
 - What to do and what to defer (Priority)
 - Useful deliverables per release and schedule
 - Tests to show that the system does what it needs to
- ❑ **Programmer role**
 - *Performs,*
 - Analyze, Design, Test, Code, and Integrate the system
 - *Estimates,*
 - Difficulty of all stories
 - *Defines,*
 - Detailed schedule
- ❑ **Manager role**
 - *Liaisons between,*
 - Customer and developers
 - *Establishes,*
 - A smoothly operating team

XP Values

- ❑ **Communication**
 - ❑ Somebody **not** talking to somebody else about something important
 - ❑ XP employs many practices that can't be done without communicating
- ❑ **Simplicity**
 - ❑ "What is the simplest thing that could possibly work?"
 - ❑ It is better to do a simple thing today and pay a little more tomorrow to change it if it **needs it**, than to do a more complicated thing today that may never be used anyways
- ❑ **Feedback**
 - ❑ Optimism is an occupational hazard of programming. Feedback is the treatment.
 - ❑ State updates through periodic (system generated) feedbacks
- ❑ **Courage**
 - ❑ *Radical surgery on the code*
 - ❑ *Try out "crazy ideas"*

Principles of XP

- ❑ Rapid Feedback
 - Action – Response delay is critical
- ❑ Assume Simplicity
 - “Hey, this is ridiculously simple!”
- ❑ Incremental Change
 - Build for NOW
- ❑ Embracing Change
- ❑ Quality Work
 - Build Clean
 - Refactor continuously

Programmer Rights


- You have the right to know what is needed, with clear declarations of priority.
- You have the right to produce quality work at all times.
- You have the right to ask for and receive help from peers, superiors, and customers.
- You have the right to make and update your own estimates.
- You have the right to accept your responsibilities instead of having them assigned to you.

Programming View of XP

- ❑ **Programming:** Simple design, testing/coding, refactoring, coding standards.
- ❑ **Team practices:** Collective ownership, continuous integration, metaphor, coding standards, 40-hour week, pair programming, small releases
- ❑ **Processes:** On-site customer, testing, small releases, planning game.

Domain Modeling in XP

- XP uses incremental, test-first programming

Find 

Author	Title	Year

Model Design

- Identify the domain objects
 - *Document, Query, Search_logic, Result*
- Test-first
 - *"find out what is there rather than what is **not** there"*

```
class TestCase
{
    public void testDocument()
    {
        Document d = new Document("a", "t", "y");
        assertEquals("a", d.getAuthor());
        assertEquals("t", d.getTitle());
        assertEquals("y", d.getYear());
    }
    public void testResult()
    {
        Result r = new Result();
        assertEquals(0, r.getCount());
    }
    public void testQuery()
    {
        Query q = new Query("test");
        assertEquals("test", q.getValue());
    }
    public void testSearchLogic() {}
}
```

Model Testing

Code Cycle

1. Write one test
2. Compile the test. **It should fail to compile!**
3. Implement just enough to compile
4. Run the test and see it **fail**
5. Implement just enough to make the test pass
6. Run the test and see it **pass**
7. Refactor code for clarity

Code Refactoring

- *In search of simplicity ...*
 - Clearer code, better design, better quality
 - *The design of the code “stinks” when*
 - Classes that are too long
 - Methods that are too long
 - More number of global variables
 - Duplicate code
 - Catch and/or throw clauses are not included as prescribed
 - Over-usage of primitive types than domain-specific types
 - Useless comments
 - Hard-coded values

Refactoring, example in Java

Original	Refactored
<pre>String a, b, c; : return a + b + c;</pre>	<pre>String a, b, c; : StringBuffer sb = new StringBuffer(a); sb.append(b); sb.append(c); return sb.toString();</pre>

XP Team Practices

- Code ownership/visibility practices
- Orphan code


```
/*
 * @author kbm gxs, kbm -at- gxs.com
 * @version $Id$
 */
```
- XP uses **collective ownership**
- Integration practices
 - Continuous integration
- Workspace
 - Release schedule
- Smaller cycles
 - Coding Standard

Pair Programming

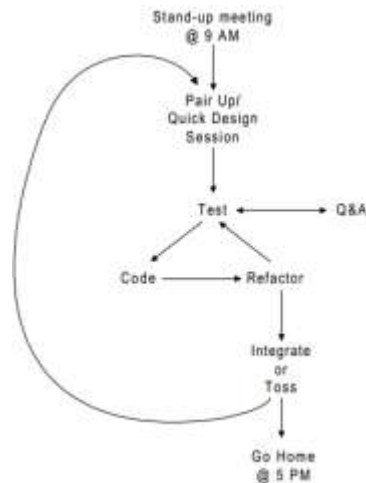


Help each other succeed

- ☐ Two people working at the same Keyboard! Weird?
- ☐ Driver-Navigator analogy
- ☐ Extra brain-power
- ☐ One thinks tactically while the other thinks strategically
- ☐ Initial dismay
- ☐ Switch roles frequently
- ☐ Fewer interruptions and distractions
- ☐ *Technical debt* decreases

Isn't it wasteful to have two people do the work of one?

Programmer's Day in XP

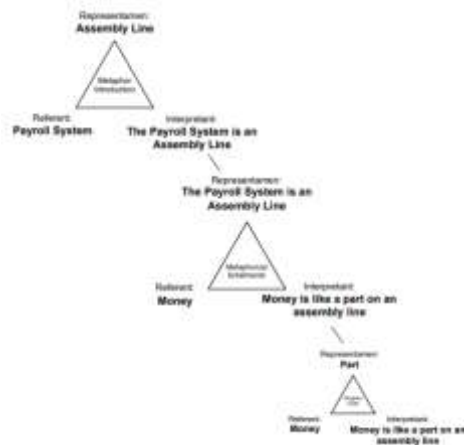


Metaphorical XP



- A metaphor is meant to be agreed upon by **all members** of a project as a means of guiding the structure of the architecture
- “You don’t know anything” syndrome
- A shared story
- Wiki → *Pieces of papers that everyone can write on*
- System Metaphor : *Can they replace the system architecture?*

Payroll System is an Assembly Line



Payroll System is an Assembly Line



$$(BP+DA+HRA+OT) - (IT+Loan+PF+Premium)$$

- While metaphors are a powerful communication tool, the agile community as a whole cannot continue to ignore the existing wealth of information on software architecture.

Simple Design and Code

- “Do the simplest thing that can possibly work”
 - Complexity to achieve something is seldom a good idea
- Program for the features you need today, not the ones you think you will want tomorrow
 - **YAGNI** -- “You ain’t gonna need it”
- Follow the **DRY** principle: “Don’t repeat yourself”
 - Don’t leave redundant copies of identical /very similar code/design

Workspace



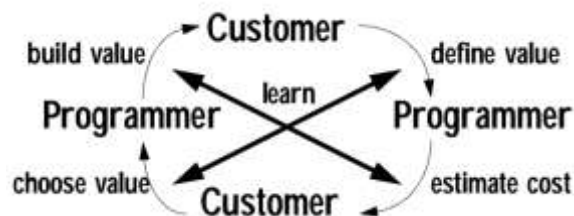
Customer Defines, Programmer Builds

- *An XP project succeeds when the customers select business value to be implemented, based on the team's measured ability to deliver functionality over time*



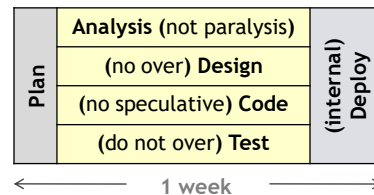
Behind all ...It's Learning

- How **valuable** the proposed features really are?
- How **difficult** the chosen features really are?
- How **long** it really takes to build the features we need?



Driving XP – No hard-cut Phases

- XP team includes *on-site customers*
 - Planning games
 - Every day stand-ups to touch base
- Customer tests (exemplified requirements)
 - Programmers' intent matches Customers' expectations?*
 - Quality-concerned pair programming
 - Energized work* + Balancing slacks
- TDD (*first level of defense*)
 - Frequent code integration
 - Exploratory testing
 - Automated regression testing
- Release to internal stakeholders
 - Only the "Done Dones" please*



Rights and Responsibilities

- **Customer/Manager**
 - You have the right to an overall plan, to know what can be accomplished, when, and at what cost
 - You have the right to get the most possible value out of every programming week
 - You have the right to see progress in a running system proven to work by passing repeatable tests that you specify
 - *Time-boxed iterations, CI, UAT*
 - You have the right to change your mind, to substitute functionality, and to change priorities without paying exorbitant costs
 - *Refactoring*
 - You have the right to be informed of schedule changes, in time to choose how to reduce scope to restore the original date. You can cancel at any time and be left with a useful working system reflecting investment to date

Programmer

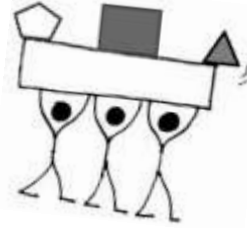
- You have the right to know what is needed, with clear declarations of priority
 - User Stories, Customer defined Release plans and Iteration plans
- You have the right to produce quality work at all times
 - Unit Testing, Pair programming, Refactoring
- You have the right to ask for and receive help from peers, superiors, and customers
 - On-site customer
- You have the right to make and update your own estimates
 - Story estimation done by programmers
- You have the right to accept your responsibilities instead of having them assigned to you
 - Programmers choose the items during Iteration planning

Programming in XP

- You have a partner – *Pair Programming*
 - The pairs vary from time-to-time
- Work on one task at a time
 - Test first, then fix code.
 - Reflect your intentions in the code and let algorithms only follow
 - Take collective ownership
 - Approved coding standards
 - Refactoring – *Try something simpler?*
 - You (as a pair) release *100% working code and test*
 - Commit your release candidates into SVN
 - Trigger CI (*wait for the next cycle*)

Collective Ownership to Code

- Collective Ownership to Code
 - Let's fix it!
- Own everything
- And ... let others also
- You wish you could find ...
 - Simple designs
 - Simple code
 - Simple test
 - Simple everything
- Simplicity is more complicated!



"Not Invented Here!" Syndrome



- The tendency of both individual developers and entire organizations to reject suitable external solutions to software development problems in favor of internally developed solutions.

So, Don't Do

- Don't try to design the whole system before you start implementing
- Do design all the time
- Don't try to freeze requirements before you start implementing
- Do focus on communication
- Do not segregate the team
- Do not delink any programmer from design activity
- Do not build for tomorrow
 - Do not code for tomorrow
 - Do build perfectly for today

Technical Debt

