Using Extreme Programming in Practice:
Developing DrJava

Charlie Reis
CSE 403 Guest Lecture
## My Background

<table>
<thead>
<tr>
<th>Year</th>
<th>Experience</th>
</tr>
</thead>
<tbody>
<tr>
<td>1998-2002</td>
<td>Undergrad at Rice University</td>
</tr>
<tr>
<td>2002-2003</td>
<td>Masters at Rice, Lead DrJava developer</td>
</tr>
<tr>
<td>2003-Present</td>
<td>UW PhD program (wireless networks, web browser security)</td>
</tr>
</tbody>
</table>
Overview

- What is DrJava?
- What is extreme programming (XP), and why use it?
- How well did it actually work for DrJava?
DrJava
DrJava

- Pedagogic Java IDE
- Simple, interactive, intuitive
- Used at dozens of schools around the world
- Freely available
DrJava Development

- **Written by students** at Rice University
  - Graduate, Undergraduate
  - High rate of turnover

- Developers use only Extreme Programming

- Work in DrJava itself
Development Team

Professor

PhD Student

Masters Student

Undergrad

Undergrad

Undergrads

Comp 312
Open Source License

• Make it freely available
• Tool and management support
• Incorporate code from others
• Educational value
• Credibility?
Tools and Management

- Sourceforge.net
  - Free hosting for 100,000+ projects
  - Professional management tools
    - Track features, bugs, tasks, support
- Ant, JUnit, CVS
Project Complexity

• Java with generics (e.g., `List<String>`)  
• ~400 classes, 50,000 lines of code  
• Complex  
  • Two JVMs, plus multithreaded GUI  
  • RMI, JDI, Custom Classloaders  
  • Backward compatibility
How would you do it?

- Context:
  - Young project, quickly maturing
  - Small team (3 to 15), high turnover
  - Vocal customers (other schools)
  - Source code available to world
Extreme Programming / Agile Methods
XP / Agile Methods

• Set of development practices:
  1. Unit tests / test first
  2. Pair programming
  3. Continuous refactoring
  4. Incremental development
  5. Onsite customer
1. Unit Tests / Test First

- Safeguard for code quality
- *Can't commit without testing*
- Improves design of program APIs
- *Think about how to use before building*
- "Executable documentation"
- *Knowledge transfer*
public void testIndentBraces() {
    String text =
        "{
        "class Foo
    +
    "extends F {
        "int i;  
    +
    "}\\n    +
    "}
    
    String indented =
        "{
        "class Foo
    +  // After open brace
    "    extends F {
        "int i;  
    +  // Not new statement
    "}\\n    +  // After open brace
    "}\\n    +  // Close brace
    "}
    
    doc.insertString(0, text, null);
    _assertContents(text, doc);
    doc.indentLines(0, doc.getLength());
    _assertContents(indented, doc);
}
2. Pair Programming

- Two people coding at one machine
- “Isn’t that a waste of effort?”
- Knowledge transfer
- Higher quality code
- Stay on task
2. Pair Programming
...for DrJava

- Getting up to speed
  - New DrJava developers paired with old
- Course projects
  - Student pairs in Comp 312
3. Continuous Refactoring

- Change the code as requirements change
- Collective ownership: sense of pride
- Always safe, thanks to unit tests
3. Continuous Refactoring

...for DrJava

- Anyone could change any DrJava module
- Many parts changed over time
  - Overhaul of indent logic
  - New approach for configuring options
4. Incremental Development

- Release early and often!
- Evolve in manageable amounts
- Keep product stable and usable
- React quickly to feedback
4. Incremental Development

...for DrJava

- Small changes (with tests)
- Development releases
  - Frequent, for new features and bug fixes
- Stable releases
  - Periodically, after a short beta period
5. Onsite Customer

- Important to work alongside customer
- Constant feedback on desired features
- No surprises when product delivered
5. Onsite Customer

...for DrJava

- Special case: developers were customers
- Local feedback from:
  - Intro courses
  - TeachJava workshop for teachers
- Many offsite customers
XP In Practice
Typical Dev Activity

• Prioritize bug reports
• Write test to exhibit bug
• Pair program to fix bug
• "Commit" (update, compile, test, commit)
• Release
Releases

- **Theory**: repository can always be released
- **Practice**: not exactly...

- Development releases (weekly/monthly)
- Stable releases (a few each year)
Life Cycle

- Peak development in spring and summer (Comp 312, summer interns)
- 3-4 large features, many small fixes
- Masters Theses
- Maintenance in "off-season"
Lessons Learned

- Unit tests are essential to stability
- Work incrementally
- XP is effective for high turnover
- Much to be gained from open source, even without many external developers
Difficulties

• Hard to test (and design) GUls
• Hard to enforce good test coverage
• Concurrency can be a mess
• Java isn't *really* platform independent...
• Tough to keep documentation up to date
• Maintenance/support is a full time job
Closing Thoughts

- XP / Agile Methods can help a project mature quickly
- Can maintain high quality as project changes
- Very satisfying to work on a project with real users
More Info

- [http://drjava.org](http://drjava.org)
- creis@cs.washington.edu