Teaching Future Software Developers

Václav Rajlich
Department of Computer Science
Wayne State University
Detroit, Michigan 48202, U.S.A
“... many graduates have essentially no education or training in software development outside their hobbyist activities...

For many, "programming" has become a strange combination of unprincipled hacking and invoking other people's libraries.”

(Comm. ACM 2010)
Expectations

• Employers and graduate schools expect CS graduates
  – To have software development skills
  – To be able to work on software projects
• Question: How to meet that expectation
Contents of this webcast

• Design of 1\textsuperscript{st} Software Engineering Course (1SEC)

• Code development basics (craftsmanship)

• Experience: 1SEC at Wayne State University

• Summary and conclusions
Current curricula

• General unease with the way SE has been taught

• Missing link
  – Expecting students to learn code development on their own
Pyramid of SE skills

- introductory programming
- technologies portfolio
- code development
- advanced practices
Introductory programming

• First step towards developer skills
• Taught in programming courses
  – “Data structures”
• Objective
  – Confidence in programming abilities
  – Belonging
• Result
  – Ability to develop small programs
Learning code development

- Requires two more steps:
  - Technologies for development
  - Processes and techniques of development
    - Being able to do (not just “know about”)
    - Bloom taxonomy level 3 or higher
- Requires (at least) one course
Teaching Software Development

• Introduce a new course?

• Reorganize 1\textsuperscript{st} Software Engineering Course (1SEC)?
  – 1SEC is already present in many programs
  – What should be the contents?

• Assumption: Just one course available
Technologies portfolio

- Core language (C++, Java, …)
- IDE (Visual studio, Eclipse, …)
- GUI (Swing, MFC, …)
- Database (SQL, Hibernate, …)
- Testing (Abbot, JUnit, …)
- Modeling (UML, XML, …)
- Intra-team communication (Wiki, …)
- Version control (CVS, Subversion, …)
- …
1SEC deadly sin #1

• Too much time devoted to technologies
• Not enough time left for code development
• Portfolio is a very specialized asset
  – Technologies change at a fast pace
  – Avoid fascination with the new technologies
  – Teach only the necessary minimum
Code development

• Development of a realistic code
  – Realistic size and complexity
    • 40 .. 400 KLOC
  – Realistic coverage of the domain
  – Realistic quality
    • Moderate technical debt
    • 1 .. 3 faults/KLOC
Old mainstream: waterfall
Waterfall is no longer mainstream

- **Vatility** of requirements, technology, knowledge
Another reason for the shift

• The accumulated technical debt of waterfall is invisible to the outside stakeholders
  – Low quality requirements
  – Suboptimal design
  – Low quality of the early code

• The things come to their head at the very end of the implementation
  – Too late
Paradigm shift

• From waterfall to evolutionary development
  – Agile ⊆ Iterative ⊆ Evolutionary development
• Happened in industry on large scale about 10 years ago
  – Only 18% of projects are using waterfall
• Academia is behind industry!
  – The shift insufficiently reflected in teaching, curricula, textbooks
Mainstream practices today

• Evolutionary, iterative, and agile code development
  – Adding new functionality to an existing software – software change

• Can be practiced on realistic code
  – Small change in a big code
Teach evolutionary development

• Adding one feature at a time
  – “Software change”
  – All stakeholders can check the result
  – Different from “maintenance”!
Deadly sin #2

• Teach practices out of mainstream

• Old waterfall
  – Realistic code not attainable in classroom
  – Does not deal with volatility, accumulated technical debt
  – Minor practice today

© 2015 Václav Rajlich
Addressing only top students

• They may be able to figure out software change on their own

• Teaching software change is important for the middle students
  – Cannot figure it out on their own
  – Their future is at stake
Deadly sin #3

- Skip code development basics and teach more advanced practices instead
  - Middle students are not ready for them
  - Teaching advanced stuff will be superficial
- Examples of advanced practices/roles not suitable for 1SEC
  - Continuous integration
  - Manager role
  - Evolution of decayed code
  - ...
1SEC as a survey course

• 60+ years of software
• 40+ years of software engineering
• HUGE amount of material
  – Google scholar: “software engineering” returns 1,320,000 publications
• What merits a slot in 1SEC?
Survey topics

• Advanced practices
• Usability
• Promising new technologies
• Management practices
• Professional ethics !!
• . . .
Deadly sin #4

- Survey occupies too much of the course
- Insufficient time left for code development
- Graduates have encyclopedic but superficial knowledge of SE
  - Do not have basic skills
  - Many textbooks are promoting this approach
1SEC project

• Opportunity to practice the code development skills learned in the lectures
Deadly sin #5

• Unrealistic expectations
  – Academic version of death march project
  – Trying to reach realistic code from scratch
• Wrong introduction to the profession
• Unfamiliar project domain can be a contributing factor
Deadly sin #6

• False remedy for excessively demanding projects
• The project aims at superficially impressing casual observer
• Sound engineering practices are missing
  – Code quality is often ignored
• More PR than substance
Deadly sin #7

• Students are graded based on the work of the others
  – They work in teams
  – Individual contribution to the team effort is invisible to the instructor
  – Instructor gives a single grade to the whole team
    • Or relies on the team member survey
  – Stressful for the best students
Contents of this webcast

• 1\textsuperscript{st} Software Engineering Course (1SEC)
  Design

• Code development basics (craftsmanship)

• 1SEC at Wayne State University

• Summary and conclusions
Basic task of code development

• Software change
  – Add/modify a functionality of existing code

• Basic task of evolutionary development

• Phased Model of Software Change
  – Software change divided into phases
Phased Model of Software Change (PMSC)

Team interactions
Comprehension of the old code
Implementation
Change initiation

• Analyzing requirements or bug reports
• Selecting a change request
Old code comprehension

• Concept (feature) location
  – Search through the code
  – Find the location to be changed
  – Key to the work in realistic software
  – Techniques: grep, dependency search, other search techniques, dynamic analysis, ...
  – Research ~15 years old

• Impact analysis
  – Determining the full extent of the change
Refactoring

- Prefactoring: Prepare code for the change
- Postfactoring: Clean-up after the change
- Numerous refactorings
  - Function extraction
  - Base class extraction
  - Component class extraction
  - Renaming
  - . . .
Actualization

- Incorporation
  - Polymorphism
  - New component
  - New composite
  - Component/composite replacement

- Change propagation
  - Making secondary changes in interacting entities
Verification

• Testing
  – Unit
  – Functional
  – Structural (coverage)

• Inspections
Conclusion

• Commit the updated code
• Resolve conflicts
• New baseline
• . . .
Example: Point of Sale

• Supports small store
• The system supports cash-only payments.
• Change request
  – Add a support for credit card payments
Concept location

Cashiers

CashierRecord

Session

Store

Sale
+payment: Payment

Inventory

Item

Price

SaleLineItem

Payment
+amount: double
+getAmount(): double

PromoPrice
Prefactoring

Cashiers

CashierRecord

Session

Store

Sale

+payment: Payment

Inventory

Item

Price

SaleLineItem

Abstract Payment

+amount: double

+getAmount(): double

Payment

+change: double

PromoPrice

© 2015 Václav Rajlich
Cashiers

Store

Inventory

CashierRecord

Sale

Item

Session

Price

SaleLineItem

PromoPrice

Abstract Payment
+amount: double
+getAmount(): double

Payment
+change: double

Credit
+cardNumber: String

Actualization
Postfactoring
JRipples tool

• Eclipse plugin
• Impact analysis
Contents of this webcast

• 1st Software Engineering Course (1SEC) Design

• Code development basics (craftsmanship)

• 1SEC at Wayne State University

• Summary and conclusions
# Technologies portfolio

<table>
<thead>
<tr>
<th></th>
<th>Prerequisite courses</th>
<th>1SEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming language</td>
<td>C++</td>
<td>C++</td>
</tr>
<tr>
<td>IDE</td>
<td>Visual studio basics</td>
<td>Visual studio</td>
</tr>
<tr>
<td>GUI</td>
<td>Qt</td>
<td></td>
</tr>
<tr>
<td>Build system</td>
<td>CMake</td>
<td></td>
</tr>
<tr>
<td>Testing tools</td>
<td>Unit testing in Visual studio</td>
<td></td>
</tr>
<tr>
<td>Modelling</td>
<td>StarUML</td>
<td></td>
</tr>
<tr>
<td>Version control client</td>
<td>Tortoise SVN</td>
<td></td>
</tr>
<tr>
<td>Version control</td>
<td>SVN</td>
<td></td>
</tr>
</tbody>
</table>
# Course schedule

<table>
<thead>
<tr>
<th>Week</th>
<th>Lectures</th>
<th>Lab</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>syllabus, introduction</td>
<td>syllabus, project tools</td>
</tr>
<tr>
<td>2</td>
<td>life span models, technologies</td>
<td>SVN, Merge and Diff, Wiki</td>
</tr>
<tr>
<td>3</td>
<td>software change initiation</td>
<td>GUI technologies: QT, Cmake</td>
</tr>
<tr>
<td>4</td>
<td>concept location</td>
<td>teams formation, assign change 1</td>
</tr>
<tr>
<td>5</td>
<td>impact analysis</td>
<td>groups meetings + Q&amp;A</td>
</tr>
<tr>
<td>6</td>
<td>actualization</td>
<td>change 1 due + team presentation</td>
</tr>
<tr>
<td>7</td>
<td>refactoring</td>
<td>refactoring - in class exercise</td>
</tr>
<tr>
<td>8</td>
<td>verification</td>
<td>groups meetings + Q&amp;A</td>
</tr>
<tr>
<td>9</td>
<td>verification, cont.</td>
<td>change 2 due + team presentation</td>
</tr>
<tr>
<td>10</td>
<td>software change conclusion</td>
<td>unit testing - in class exercise</td>
</tr>
<tr>
<td>11</td>
<td>evolutionary software development</td>
<td>groups meetings + Q&amp;A</td>
</tr>
<tr>
<td>12</td>
<td>initial development</td>
<td>groups meetings + Q&amp;A</td>
</tr>
<tr>
<td>13</td>
<td>final stages of software lifespan</td>
<td>change 3 due + team presentation</td>
</tr>
<tr>
<td>14</td>
<td>professional ethics</td>
<td>extra credit due</td>
</tr>
</tbody>
</table>

© 2015 Václav Rajlich
Course Projects

- Running in parallel with the lecture
- Open source
- Use students’ technologies portfolio
- Simple and familiar domain
  - Editors, simple graphics apps, file systems
Survey part of 1SEC

• Team practices for software evolution
  – Agile
  – Directed

• Other phases of code development
  – Initial development from scratch
  – Final stages of software lifespan
    • Servicing (maintenance)
    • Phase-out, close-down
    • Reengineering

© 2015 Václav Rajlich
Other survey topics

• Software engineering professional ethics
  – Privacy and confidentiality
  – Information diversion

• Management
  – Process management
  – Product management
Which of the following skills have been useful in your project?

<table>
<thead>
<tr>
<th>Skill</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version control</td>
<td>75.0%</td>
</tr>
<tr>
<td>Working in a team</td>
<td>75.0%</td>
</tr>
<tr>
<td>Software change process</td>
<td>43.8%</td>
</tr>
<tr>
<td>Requirements analysis</td>
<td>56.3%</td>
</tr>
<tr>
<td>Concept location</td>
<td>75.0%</td>
</tr>
<tr>
<td>Impact analysis</td>
<td>62.5%</td>
</tr>
<tr>
<td>Refactoring</td>
<td>62.5%</td>
</tr>
<tr>
<td>Unit testing</td>
<td>56.3%</td>
</tr>
<tr>
<td>Functional testing</td>
<td>62.5%</td>
</tr>
<tr>
<td>Regression testing</td>
<td>50.0%</td>
</tr>
</tbody>
</table>
Contents of this webcast

• 1st Software Engineering Course (1SEC) Design
• Code development basics (craftsmanship)
• 1SEC at Wayne State University
• Summary and conclusions
Future work

• Regularly update technologies portfolio

• Develop integrated PMSC tools
  – Should guide students
  – Help grading
Proposed follow-up courses

• Processes and roles
  – Tester
  – Agile developer
  – Initial development (Design and Code)
  – Requirements engineer

• Technologies
  – Cloud
  – SOA
  – End user programming
The goal (repeated)

• To give CS graduates code developer skills
• To change the current situation where the most valuable knowledge is self-taught
  – Creates chasm between classroom and practice
  – There are casualties who do not make it across the chasm
Book on this topic