SOFTWARE MAINTENANCE AND EVOLUTION
CS563
WEEK 1 - THU

MAPPING THE LANDSCAPE OF REFACTORING RESEARCH

AKA – REFACTORING THE REFACTORING

Danny Dig
YOUR Expectations

- Apply state-of-the-art research to my software projects
- Understand evolution steps in project lifecycle
- Team SW development
  - Understand state of practice in industry on SW Maintenance & Evolution
  - Practice maintenance & evolution of SW projects (scalability)
- Learn about research in SW & TFJ Evolution
- Design & find reusable code
- Using ML in innovative IoT applications
- Build tools to evolve SW with ML components
- Understand SW changes and become comfortable with highly evolving platforms
- Learn about cloud applications
Course Objectives

Software Maintenance and Evolution research & practice
- state of the art under the theme of Mobile and IoT

Develop critical thinking abilities

Practice giving scientific presentations and teaching others

Engage in active learning activities in class, such as discussions

Practice a research or novel-industrial project through all its stages

Have fun learning
Course Syllabus

Research-based course

Complete a research or industrial-novel project of your choice (teams of 2-3 students) [50%]

Read papers, write Critiques. [20%]

Class Discussion [10%]

Papers Presentation [20%]

Put Your Dream to the Test
What is Refactoring?

“A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour” — M. Fowler [1999]

Top-level menu in all modern IDEs

- In 2000, I created the first open-source refactoring tool
Refactoring research growth

2,880 refactoring papers (4,944 authors) since 1990
The Humble Beginnings

First refactoring paper:
- Bill Opdyke and Ralph Johnson [SOPPA’90]: Refactoring, an Aid in designing application frameworks and evolving OO systems

First PhD dissertations:
- Bill Griswold ‘91 at U of Washington
- Bill Opdyke ‘92 at U of Illinois
- Don Roberts ‘99 at U of Illinois

Refactoring research hard to publish in early 90s
- conflated with the compiler community
Most recent Decade of Refactoring Research

2,880 refactoring papers since 1990

2,442 papers between 2005-2016
Corpus of Papers

Work done by Marouane Kessentini and his team at Michigan

Scopus and Web of Science
- "Refactoring" in title, abstract, and keywords
- yielded 3277 papers

Refactoring definition:
- transformation with behavior preservation

Manual validation of ALL papers:
- each paper analyzed title, abstract (and sometimes content)
- 4 grad students who took a graduate class on Softw QA
- Kessentini (faculty) looked at the contentious papers

In the end we removed 397 papers
O1: To Grow, Welcome Outsiders, Champions from Other Communities
O2: To Grow, Expand Focus of Interest (the WHAT)

Expand focus to meet new needs that you can serve
Examples of new Focus on Automation

Refactoring for introducing functional features in OO programs loop iterators \(\rightarrow\) functional streams with lambda [Gyori et al. FSE’13]

Scalability 1: Refactoring to Design Patterns contain hundreds of lower-level refactoring steps [Batory et al. – ICSE’16] - 10x faster than state-of-the-art IDE refactorings

Scalability 2: Ultra-large scale refactoring for codebases of Hundreds of Millions LOC (e.g., Google, Microsoft scale) - whole-program analysis is not feasible - IDE-dependent tools do not fit CI workflows

Advancing the next generation of global, distributed refactoring - MapReduce on the cloud - scalable, safe, useful (developers accept hundreds of patches)
Examples of new Focus on Inferring Refactorings

RefactoringCrawler infers API-level refactorings for API migration  
[Dig et al. – ECOOP’06]

RefFinder – infers the most comprehensive list of refactorings  
[Kim et al. – ICSM’10]

RefactoringMiner: commit-based detection  
[Tsantalis et al. – ICSE’18]

  No similarity thresholds
  High accuracy: 98% precision, 87% recall
  Ultra-fast: 58ms on median per commit
  Better than competitive tools (RefDiff): +22% precision, 7x faster

Largest and least biased refactoring oracle up to date  
3188 true refactoring instances, 538 commits, 185 open-source projects  
http://refactoring.encs.concordia.ca/oracle/
O3: To Grow, Expand the Target Artefacts

Expand target: new refactoring research is about change to the code, models, architecture, DB, UI
O4: To Grow, Expand Objectives (the WHY)

Expand Objectives: new refactoring research is to improve performance, security, migration (beyond internal quality)
Overview of Our Refactorings for Asynchronous Programming for Mobile Apps

Slow operations freeze mobile apps and frustrate users - 75% of performance bugs in Android [Li et al., ICSE’14]

Culprit: long running operations running in the main UI thread
Solution: refactoring for asynchronous execution
Overview of Our Refactorings for Parallelism

Refactorings for **thread-safety**
- make class immutable [ICSE'11]
- convert to Atomic* classes [ICSE'09]
- use concurrent collections [ICSE'09]
- infer region annotations [ASE'09]
- atomic check-then-act operations [ICST’13]

Refactorings for **throughput**
- parallel recursive divide-and-conquer [ICSE'09]
- loop parallelism via ParallelArray [OOPSLA’10]
- loop parallelism via lambda functional operators [FSE’13]

Refactorings for **scalability**
- Atomic*, concurrent collections [ICSE'09]
O5: To Increase Practical Impact, Work with Industry

Industrial collaboration levels:
- surveys with practitioners
- tool validated on industrial codebase
- tool licensed to industry, adopted in products

Open-source data
Industrial data
Big Growth of the Field: Expanding Definition

“A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour” – M. Fowler ‘99

Expanded Focus, Objectives, Techniques

“Automation/insight/testing/prioritization of changes to the artifacts of software to improve non-functional requirements and without changing its proper, intended behaviour” – D. Dig ‘19

Communities that thrive are going to be more accepting of new ideas
Big Growth Enabled by Community Engineering

Industry champions: Martin Fowler, Kent Beck, Ward Cunningham, Joshua Kerievsky, Michael Feathers, Uncle Bob

Complementary skills: tool builders, paper writers, curators

Mindset for industrial collaboration and adoption

Shared platform:
  - Eclipse (Erich Gamma + Frank Tip), analysis frameworks

Community infrastructure: 7 Refactoring Workshops, Dagstuhl
  - first workshop in 2007, 50+ participants, 32 posters
  - invited all major IDE providers
  - growing new leaders
Reflections and Lessons I am Learning
On Aug 5, 2015 ...

A life of significance: intentionally serve others
L1: Work in Your Strength Zone but Reinvent Yourself

Mobile
- add async
- fix async
- privacy

Parallelism & Concurrency
- make thread-safe
- improve throughput
- improve scalability

Library migration
- upgrade APIs

IoT and ML
- from deterministic to probabilistic

Principles for changing between different programming models
L2: Find Your Dream and then Live It

- Automating: ship with official NetBeans IDE, Visual Studio
  - hundreds of accepted patches
  - first open-source refactoring

- Refactoring: Eclipse
  - used at Google, IBM
  - dozen labs

- Inferring: founded Workshop on Refactoring Tools, HotSwUp, Dagstuhl S.

- Understanding: shaped APIs in Java and .NET official concurrency libraries

- Testing: Oracle
  - learnparallelism.net
  - 150,000+ visitors
## L3: Proactively Look for Opportunities, but Be Flexible

<table>
<thead>
<tr>
<th>Expected Company</th>
<th>Actual Company</th>
<th>Expected Target</th>
<th>Actual Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td>ORACLE®</td>
<td>Lambda Expressions</td>
<td>Lambda Expressions</td>
</tr>
<tr>
<td>Google</td>
<td>Google</td>
<td>Async Programming</td>
<td>Type migration at scale</td>
</tr>
</tbody>
</table>
Do you have a plan for your personal growth?
How do you get better at what you do?
How do you improve your relationships?
How do you hire great students?
How do you mentor and grow them into tomorrow’s tech leaders?
How do you prioritize the important over the urgent?
My Most Important Investment

Malinda Dilhara (PhD)
Ameya Ketkar (PhD)
Tom Dickens (PhD)
Michael Hilton (PhD’17, now at CMU)
Semih Okur (PhD’16, now at Microsoft)
Yu Lin (PhD’15, now at Google)
Stas Negara (PhD ‘13, now at Google)
Shane McKane (MS’17, now at Intel)
Mihai Codoban (MS ‘15, now at Microsoft)
Kendall Bailey (MS ‘15, now at Intel)
Cosmin Radoi (MS ‘13, now PhD student UIUC)
Sandro Badame (MS ‘12, now at Google)
Fredrik Kjolstad (MS 2011, now PhD student MIT)
Binh Le (MS 2009, SW developer)
Can Comertoglu (MS 2009, now at Microsoft)

Jacob Lewis (Summer’16 – ‘17)
Jonathan Harijanto (Summer’16 –’17)
Lily Mast (Summer’15)
Elias Rademacher (Summer’15 - current)
Nicholas Nelson (Summer 2014-15)
Sean McDonald (Summer’14 –Fall’15)
Hugh McDonald (Summer’14 – Fall’15)
Alexandria Shearer (Summer’12)
Kyle Doren (Summer’12)
Lyle Franklin (UIUC, Summer’12)
Alex Gyori (UIUC, Summer’12)
Yuwei Chen (UIUC, Spring 2012)
Anda Bereckzy (UIUC, Fall’11-Spring’12)
Alex Sikora (UIUC, Fall’11)
Jack Ma (UIUC, Summer’11)
Lorand Szacaks (UIUC, Summer’11)
Caius Brindescu (UIUC, Summer’11)
Mihai Codoban (UIUC, Summer ‘11)
Mihai Tarce (UIUC, Summer’09)
Cosmin Radoi (UIUC, Summer’09)
John Marrero (MIT, Spring’08 – Summer’08)
Call to Action

Big growth enabled by “refactoring” the refactoring
Change is the only guaranteed constant

L1: work in your strength zone, but reinvent yourself
L2: find your dream and then live it
L3: proactively look for opportunities, be flexible
L4: to grow others, first grow yourself

If you want to go fast in life, go alone. If you want to go far, go with others.

Contact me at digd@eecs.oregonstate.edu
- you have codebases where you retrofit ML computations
- you have questions about the practice of refactoring
- finding your own peer-best-practices, like-minded group
- feedback on today’s presentation: valuable or contrarian points
Questions from Hilton’s paper

This was a class project

Miguel: “The sample set is too small and too limited to convince me”

Braden: refactoring in environments that allow aliasing

Dan, Braden: Why touchDevelop?

Malinda: what about false positives? How they check type correctness?

Vijay: other tools there? CloudCone?
New Ideas from Hilton’s paper

This was a class project

Miguel, Dan: “generalize beyond touchDevelop”

Braden: standardize the data type across providers

Malinda: predict and resolve latency

Vijay: expand to other areas: energy consumption, tolerance for network shortages