Mining Evolutionary Data

• Version control data (e.g. commits history) enriches our vision of **software evolution**
  – Metadata about the software change: comments, user-ids, timestamps
  – Differences between the versions: addition, deletion or modification
  – Analysis of different software versions (snapshots)

• Add the **time dimension** to codebase understanding
Mining Software Repositories

- Describe a broad class of investigations into the examination of software repositories data
- Empirical and systematic investigations
- Identify uncovered information, relationships or trends

- Data can be automatically **gathered** using collecting tools
  - VCS APIs (JGit, repodriller, pydriller…)
  - IDE APIs (plugins)
  - Supporting Tools (issue trackers, coverage reports, CI tools…)
  - Knowledge platforms and developers communities: (Stack Overflow, GitHub)

- Collected data can be **analysed** to reason about software evolution
  - Change history
  - Static code analysis
  - Software process insights

- Ease code comprehension by data **visualisation**
Example of analysis questions

- Which IDE commands do developers use?
- How are test cases executed?
- Does refactoring lead to more failed tests?
- How do developers navigate the code base?
- What kind of changes do developers revert?

<table>
<thead>
<tr>
<th>Evolutionary task category</th>
<th>Approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evolutionary couplings/patterns</td>
<td>Bieman et al. [79], Canfora and Cerulo [56,57], Fischer et al. [23,27], Gall et al. [20,26,76], Hassan and Holt [69], Kagdi et al. [36], Shirabad et al. [37–39], Williams and Hollingsworth [48], Zimmermann et al. [15,33], Ying et al. [34]</td>
</tr>
<tr>
<td>Change classification/representation</td>
<td>Antoniol et al. [62], German [80], Hindle and German [29], Holt and Pak [75], Kim et al. [30], Mockus and Votta [77], Nikora and Munson [73]</td>
</tr>
<tr>
<td>Change comprehension</td>
<td>Beyer and Noack [81], Burch et al. [82], Chen et al. [22], Chen et al. [83], Cubranic et al. [60,61], Gall et al. [76], Görg and Weiβgerber [78], Hindle and German [29], Holt and Pak [75], Kim et al. [84], Purushothaman and Perry [67,68], Claudio [85], Robles et al. [44], Van Rysselberghoe and Demeyer [53], Venolia [86]</td>
</tr>
<tr>
<td>Defect classification and analysis</td>
<td>Anvik et al. [63], German [21], Livshits and Zimmermann [35], Menzies et al. [52], Nagappan et al. [87], Ostrand and Weyuker [43], Sandusky et al. [42], Sliwerski et al. [28], Williams and Hollingsworth [45,46]</td>
</tr>
<tr>
<td>Source code differencing</td>
<td>Maletic and Collard [31], Neamtiu et al. [71], Raghavan et al. [70], Sager et al. [88]</td>
</tr>
<tr>
<td>Origin analysis and refactoring</td>
<td>Dig et al. [89,90], Godfrey et al. [72,91], Görg and Weiβgerber [49,78], Henkel and Diwan [92], Kimand Notkin [55], Ratzinger et al. [54], Tu and Godfrey [74], Weiβgerber and Diehl [93], Zou and Godfrey [24]</td>
</tr>
<tr>
<td>Software reuse</td>
<td>Selby [47], Van Rysselberghoe and Demeyer [32], Xie and Pei [94]</td>
</tr>
<tr>
<td>Development process and communication</td>
<td>Dinh-Trong and Bieman [40], El-Ramly and Stroulia [95], Hayes et al. [59], Huang and Liu [64], Mockus et al. [96], Ohba and Gondow [58], Ohira et al. [65,66], Ying et al. [50]</td>
</tr>
<tr>
<td>Contribution analysis</td>
<td>Koch and Schneider [97], Mockus et al. [96], Robles et al. [41,98]</td>
</tr>
<tr>
<td>Evolution metrics</td>
<td>Capiluppi et al. [51], Godfrey et al. [72,91], Menzies et al. [52], Nagappan et al. [87], Nikora and Munson [73], Tu and Godfrey [74]</td>
</tr>
</tbody>
</table>
Use case: Reduce code complexity

- Software maintenance is both difficult and expensive: we need to reduce complexity

- **Metrics** can help in identifying complex code, that is hard to understand and tricky to modify ([last snapshot](#))

- “If no one needs to read or modify a particular part of the code, does it really make a difference whether it’s complex?” [Adam Tornhill, 2015]
YES! IT’S A TIMEBOMB READY TO EXPLODE, BUT...

• We can prioritize software-design improvements involving system parts that developers will most likely work with again in the future

• We need to understand how the developers work on the code

• We must look at how the system evolved (not just the last snapshot)
  – Treating version-control data as our evidence on developers behaviour
Code as a Crime Scene

• Geographical profiling has its scientific basis in statistics and environmental psychology

• The locations where crimes occur are very rarely random:
  – Typically, the location of the next crime is in the opposite direction from the first scene

• Geographic profiling can help track down Jack the Ripper

[A. Tornhill: Your Code As a Crime Scene]
Whitechapel Murders

Aaron Kosminski (1865-1919)

James Maybrick (1838-1889)
Geographical Profiling with Dragnet

Uses the locations of a series of crimes to prioritise the surrounding areas in relation to the likely location of the offender's base, known as hotspot (red area)

Investigators can focus their efforts on the smaller area instead of patrolling the entire city!

Explore the geography of code

• What if we could devise techniques that let us identify **hotspots** in **large software systems**?

• Geographical profiling would give us a **prioritized lists** of sections that need refactoring

• **Code Hot Spot**: complex code that is frequently changed
Hotspot analysis

Complexity + Efforts

[http://www.inf.usi.ch/phd/wettel/codecity.html]
Distributed components

Service 1

↓

Service 2

↓

Service 3

↓

Service 4

↑

Service 5

Change coupling

X : commit 1
X : commit 2
X : commit 3
Change coupling (co-change)
MSR 2019 Mining Challenge

- Mining SOTorrent, a dataset providing the version history of Stack Overflow posts:
  - textual discussion
  - code blocks
  - linked references to GitHub files
- The overall goal is to study the origin, evolution, and usage of Stack Overflow code snippets

[https://arxiv.org/pdf/1809.02814.pdf]
Here is my go at it (no loops and handles both SI units and binary units):

```java
public static String humanReadableByteCount(long bytes, boolean si) {
    int unit = si ? 1000 : 1024;
    if (bytes <= unit) return bytes + (" B");
    int exp = (int) (Math.log((double)bytes) / Math.log(unit));
    String pre = (si ? "kB" : "KiB") + (exp == 0 ? (si ? " B" : "B")
    return String.format("%.
3.0f", bytes / Math.pow(unit, exp), pre);
}
```

Example output:

<table>
<thead>
<tr>
<th>SI</th>
<th>0 B</th>
<th>0 B</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>27 B</td>
<td>27 B</td>
</tr>
<tr>
<td>999</td>
<td>999 B</td>
<td>999 B</td>
</tr>
<tr>
<td>1000</td>
<td>1.0 kB</td>
<td>1000 B</td>
</tr>
<tr>
<td>1023</td>
<td>1.0 kB</td>
<td>1023 B</td>
</tr>
<tr>
<td>1024</td>
<td>1.0 kB</td>
<td>1.0 KiB</td>
</tr>
<tr>
<td>1728</td>
<td>1.7 kB</td>
<td>1.7 KiB</td>
</tr>
<tr>
<td>10992</td>
<td>10.6 kB</td>
<td>10.0 KiB</td>
</tr>
<tr>
<td>7677888</td>
<td>7.1 MB</td>
<td>6.8 MiB</td>
</tr>
<tr>
<td>45298432</td>
<td>453.0 MB</td>
<td>432.0 MiB</td>
</tr>
<tr>
<td>2895162948</td>
<td>29.0 GB</td>
<td>27.0 GiB</td>
</tr>
<tr>
<td>185542567192</td>
<td>1.9 TB</td>
<td>1.7 TiB</td>
</tr>
</tbody>
</table>
| 9223372036854775807 | 9.2 EB| 8.0 EiB (Long.MAX_VALUE)

Related article: **Java: Formatting byte size to human readable format**

FileUtils.byteCountToDisplaySize(long size) would work if your project can depend on org.apache.commons.io.

JavaDoc for this method
Example of research questions

• How many clones of code snippets exist inside Stack Overflow?
• How are code snippets on Stack Overflow maintained?
• Does the evolution of Stack Overflow code snippets follow patterns?
• Do these patterns differ between programming languages?

• How can we detect buggy versions of Stack Overflow code snippets and find them in GitHub projects?
• How frequently are code snippets copied from external sources into Stack Overflow and then co-evolve there?
• How do snippets copied from Stack Overflow to GitHub co-evolve?

• Can we reliably predict bug-fixing edits to code on Stack Overflow?
• Can we reliably predict popularity of Stack Overflow code snippets on GitHub?
RepoDriller & PyDriller

- **RepoDriller**: a Java framework for mining software repositories
  - Extract information from Git repositories
    - commits, branches, tags
    - developers info
    - modifications and diffs
    - source codes
- Quickly export CSV files
- Integration with Static Analysis tools and Code Parsers (Eclipse JDT, Java Parser)
- **PyDriller**: Python version of RepoDriller

[https://github.com/mauricioaniche/repodriller]
[https://github.com/ishepard/pydriller]
import org.repodriller.*;

public class S1 implements Study {
    public static void main(String[] args) {
        new RepoDriller().start(new S1());
    }

    public void execute() {
        String gitUrl = "https://github.com/mauricioaniche/repodriller.git";
        String repoDir = "exp/repodriller";
        String cvsFile = "exp/devs.cvs";

        GitRemoteRepository.hostedOn(gitUrl).inTempDir(repoDir).buildAsSCMRepository();

        new RepositoryMining().in(GitRepository.singleProject(repoDir)).through(Commits.all()).process(new DevelopersVisitor(), new CSVFile(cvsFile)).filters(new OnlyModificationsWithFileTypes(Arrays.asList(".java", ".xml")), new OnlyInBranches(Arrays.asList("master")), new OnlyNoMerge()).mine();
    }
}
import org.repodriller.*
public class DevelopersVisitor implements CommitVisitor {

    public void process(SCMRepository repo, Commit commit, PersistenceMechanism writer) {
        writer.write(
            commit.getHash(),
            commit.getCommitter().getName());
    }
}
import os
from git import Repo
from pydriller import RepositoryMining

git_url = 'https://github.com/ishepard/pydriller'
repo_dir = 'exp/pydriller'

if not os.path.exists(repo_dir):
    Repo.clone_from(git_url, repo_dir)

for commit in RepositoryMining(repo_dir).traverse_commits():
    print('Hash {}, author {}'.format(commit.hash, commit.author.name))
References

• A. Tornhill: Your Code As a Crime Scene