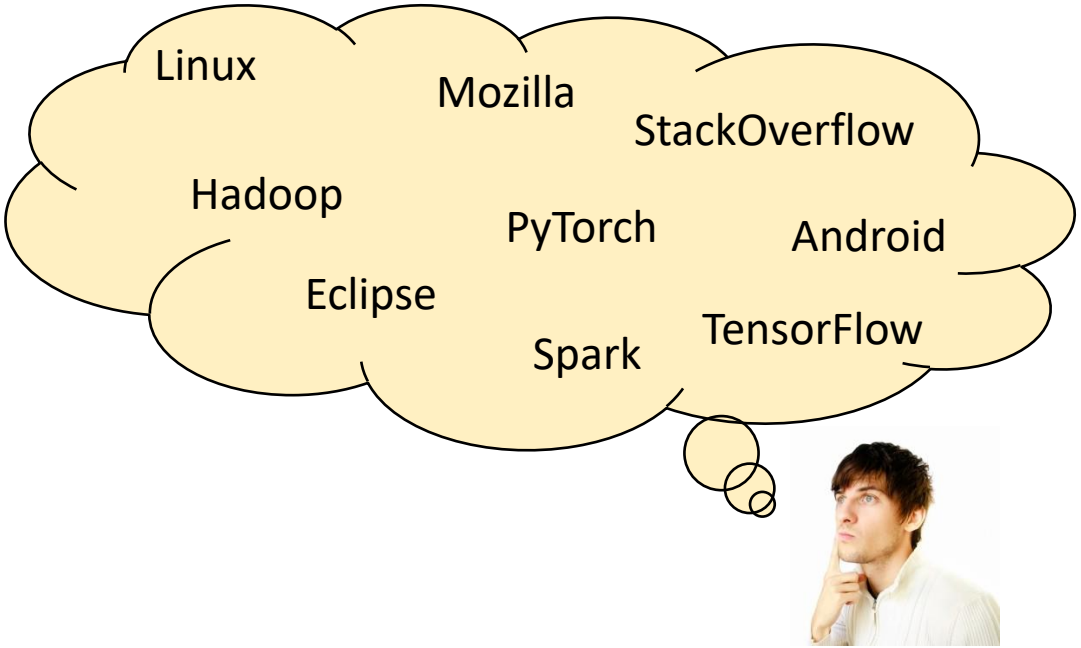
COMP5111 – Fundamentals of Software Testing and Analysis
Mining of Public Software Repositories

Shing-Chi Cheung

Department of Computer Science & Engineering

HKUST

Rich Pools of Programming Wisdom



How to:

- **use library APIs**
- **construct unit tests**
- **mock test dependencies**
- **resolve build failures**
- **fix bugs**
- **implement functions**
- **write comments**

A Recent Effort Made by Mozilla

Ubisoft And Mozilla Announce A.I. Coding Assistant Clever-Commit

FEBRUARY 12, 2019 11:05AM PT

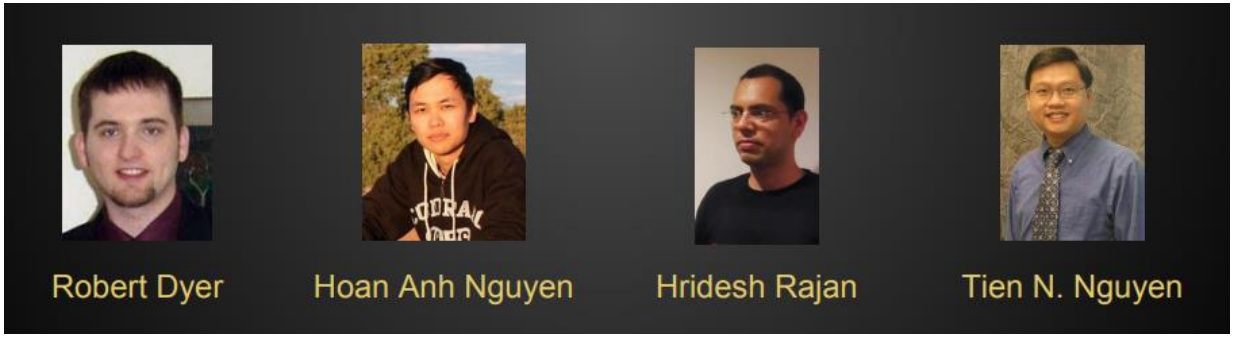
Video game publisher Ubisoft is working with Mozilla to develop an artificial intelligence coding assistant called Clever-Commit, …
Clever-Commit reportedly helps programmers **evaluate** whether or not a **code change** will introduce a new bug **by learning from past bugs and fixes**.
The prototype was tested using data collected during game development, Ubisoft said, and it's already contributing to some major AAA games. The publisher is also working on integrating it into other brands.

https://variety.com/2019/gaming/news/ubisoft-and-mozilla-announce-clever-commit-1203137446/

boa

Mining Ultra-Large-Scale Software Repositories

Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N. Nguyen, "Boa: A Language and Infrastructure for Analyzing Ultra-Large-Scale Software Repositories", In the proceedings of the 35th International Conference on Software Engineering (ICSE 2013), May 22, 2013. San Francisco, CA



https://www.youtube.com/watch?v=KZi_KdWv8ns (49:36)

Mining Wisdom from Public Software Repositories

Codes, issues, commits, tests and APIs encapsulate developers' wisdom.

- Apache: 318 projects
- SourceForge: 430,000+ projects
- GitHub:
  - 25M+ active projects
  - 1B+ commits per year
  - 12.5M+ issues filed per year

Example

- Q: How many code revisions at GitHub involve IF-clause?
- Q: Is null-pointer a major issue in these revisions?
- A: 1,340,561 (82.6%) out of the 1,622,375 code revisions of IF-clauses filed at GitHub as at Sept 2015 involve null-pointer checks.

BOA Framework – Code Query Engine

Projects	7,830,023
Code Repositories	380,125
Revisions	23,229,406
Unique Files	146,398,339
File Snapshots	484,947,086
AST Nodes	71,810,106,868

GitHub Sep 2015

Projects	699,331
Code Repositories	494,158
Revisions	15,063,073
Unique Files	69,863,970
File Snapshots	147,074,540
AST Nodes	18,651,043,238

SourceForge Sep 2013

http://boa.cs.iastate.edu/stats/

BOA – A Query Engine on Code Repositories

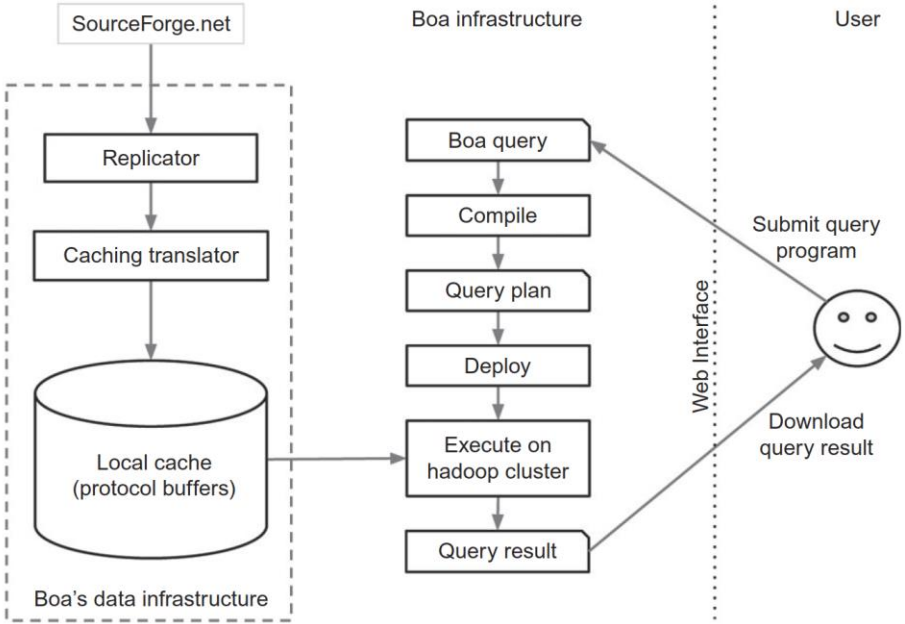


FIGURE 20.1

Overview of Boa's architecture. Data is downloaded, translated, and cached on Boa's servers. Users interact with a Web-based interface to submit queries, which are compiled into Hadoop distributed programs, run on a cluster, and the results are given via the Web interface.

A Boa query counting the number of Java projects:

```
count: output sum of int;
p: Project = input;

exists (i: int; input.programming_languages[i] == "Java")
        count << 1;
```

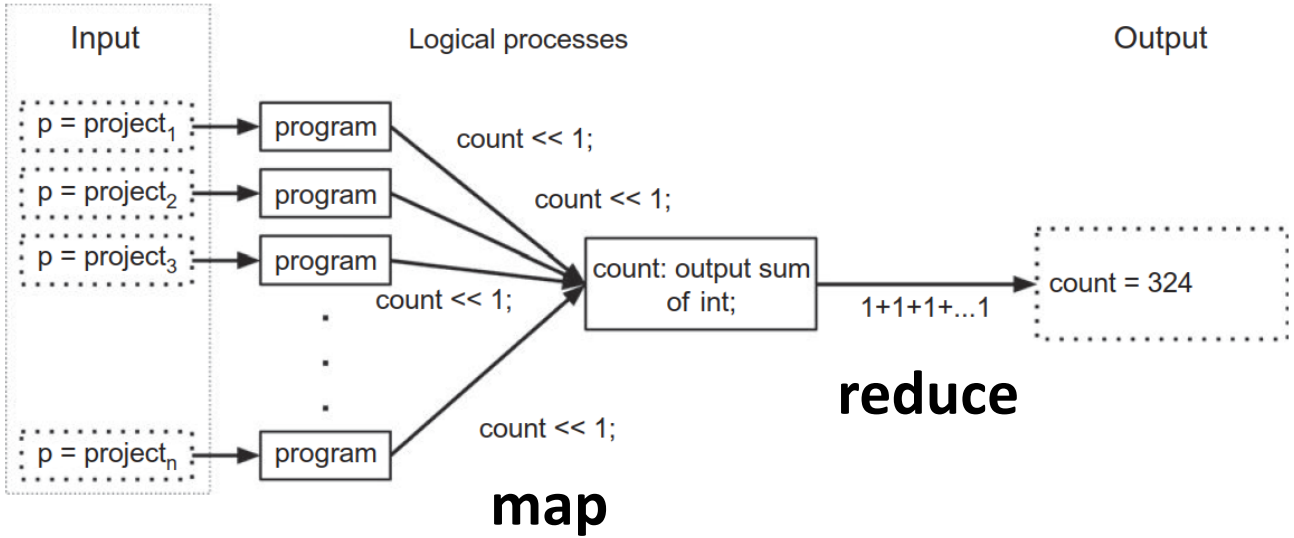
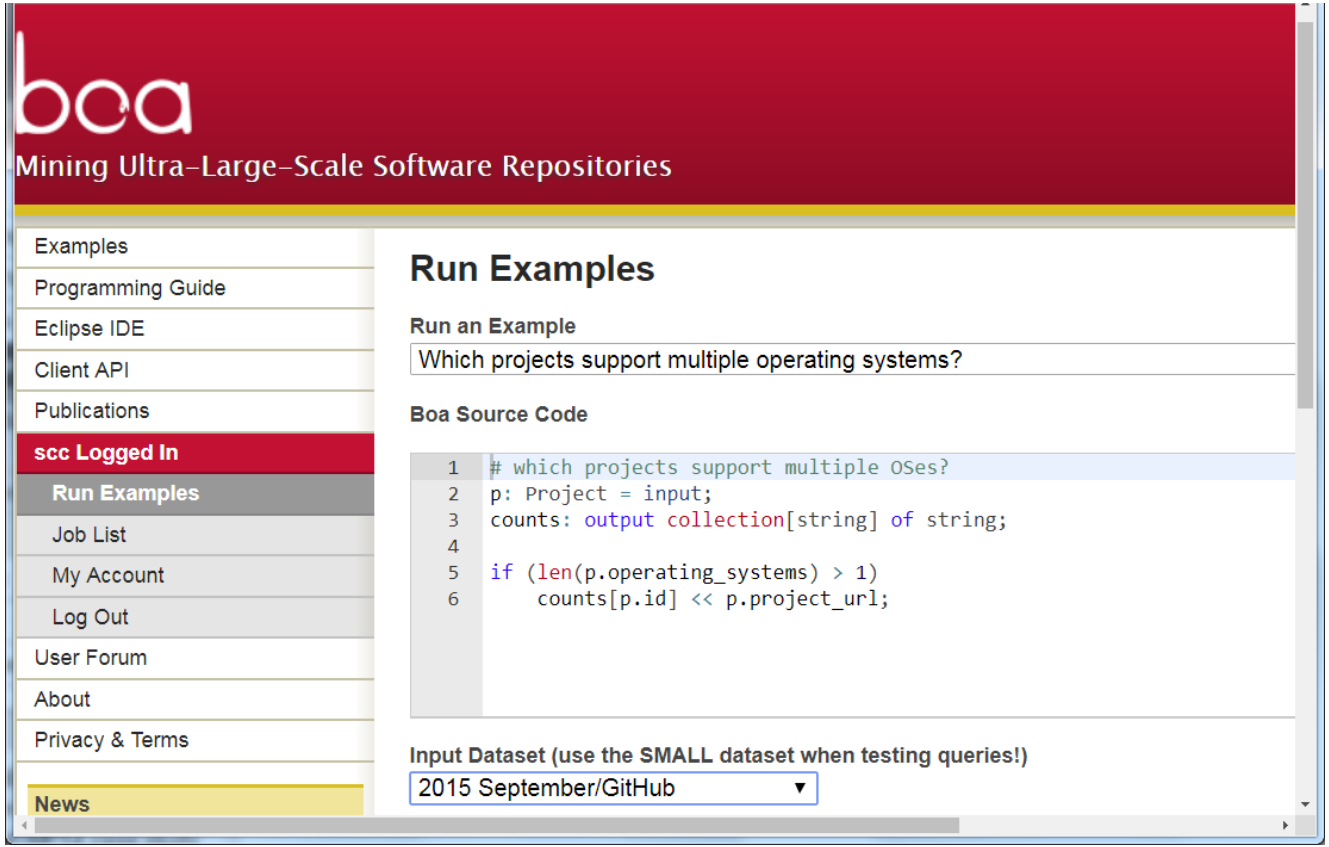


FIGURE 20.3

The semantic model of a Boa program. Inputs are single projects. Each input has its own process and analyzes one project. Outputs are also their own processes and receive data, aggregate the data, and generate the resulting output.



http://boa.cs.iastate.edu/boa/?q=boa/run

Indexed Project Information

- Project attributes
  - id, name, created_date, description, developers, maintainers, OS, programming_languages, project_url, topics
- CodeRepository
  - url, kind, revisions
- Revision
  - id, log, committer, commit_date, files
- ChangedFile
  - name, (file) kind, change (kind)

Indexed Source Code Information

- Source file – indexed as an Abstract Syntax Tree to facilitate richer queries.
- Namespace
- Declarations
  - name, parents, type, modifiers (e.g., visibility, static/dynamic, final, public/private/protected), generic parameters, fields, methods, nested declarations

Indexed Source Code Information

```
for (int i = 0; i < 5; i++) { … }
```

indexed as 

Statement {

kind = StatementKind.FOR,

initializations = [/* int i = 0 */],

updates = [/* i < 5 */],

expression = /* i++ */,

statements = … }

So, what can we mine?

Two demo applications:

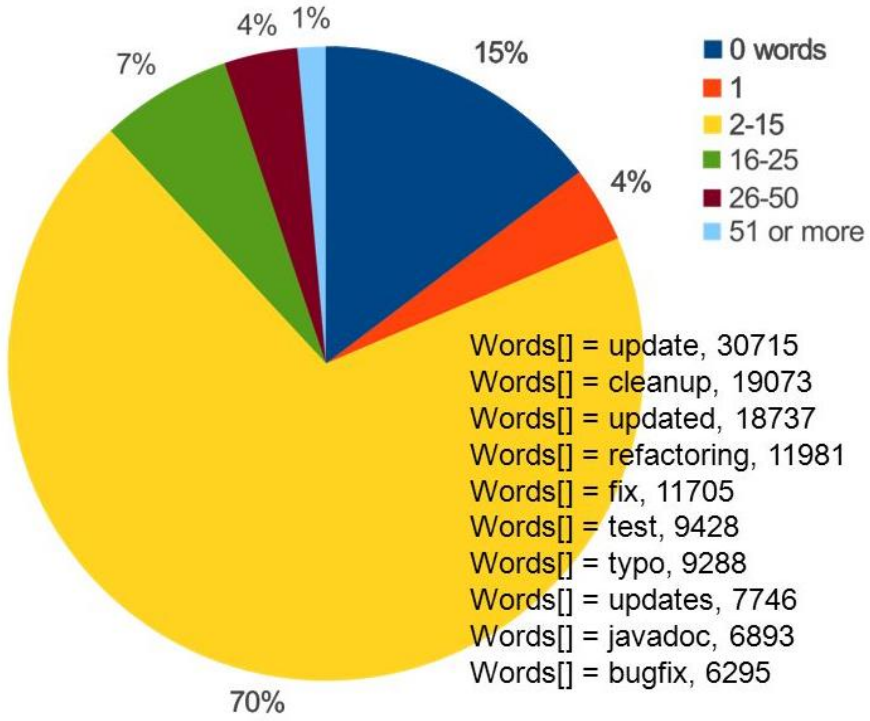
- **Developers' practice**
- **Program code**

adapted from Mining Programs by Andreas Zeller

Mining Developers' Practice

- What are the common objectives of code commits?
- Will each commit be often followed by a unique review?
- How often are patch commits rejected in review?
- What are the major reasons that a commit is rejected?

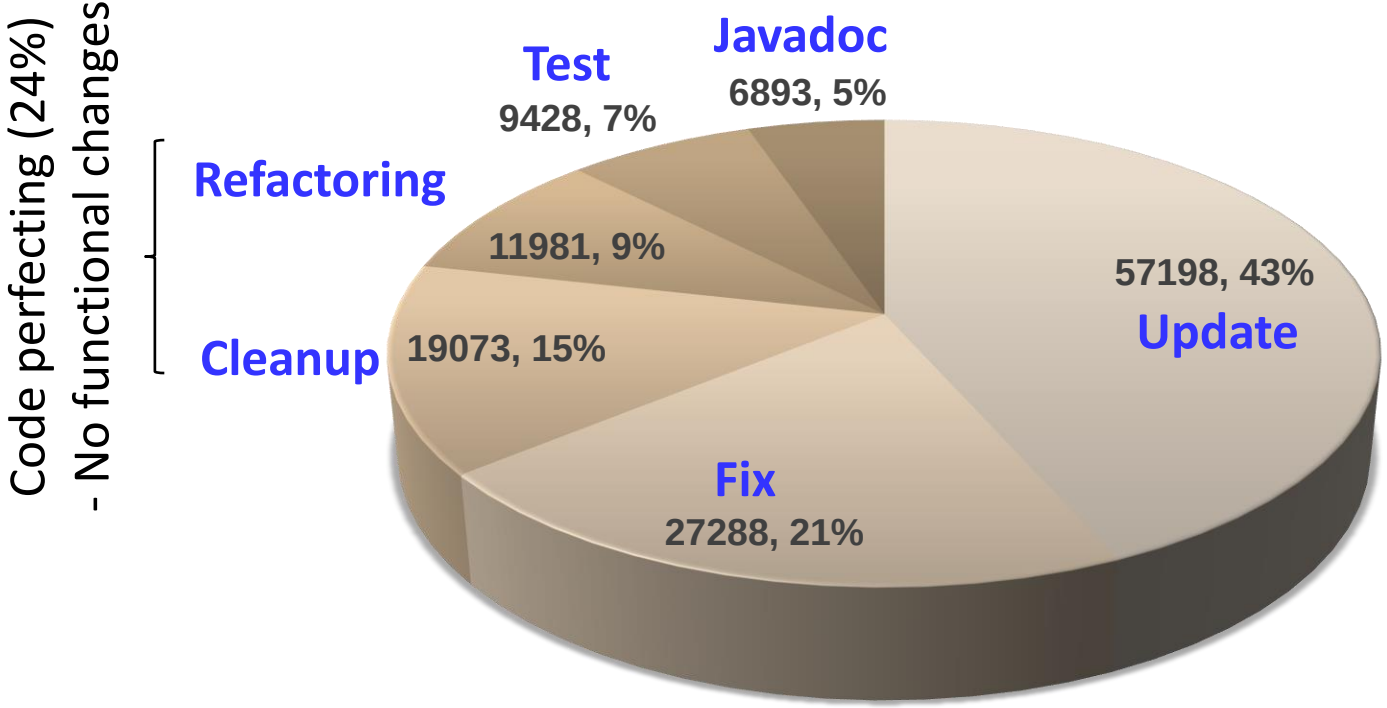
Understanding Code Review Practice



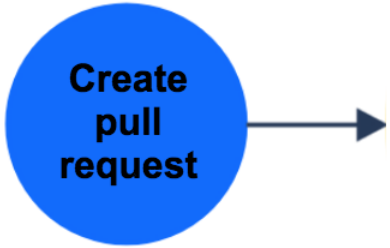
How many words are in commit messages?

What are the most common objectives of code commits?

Code Commit Objectives



Code Review Process

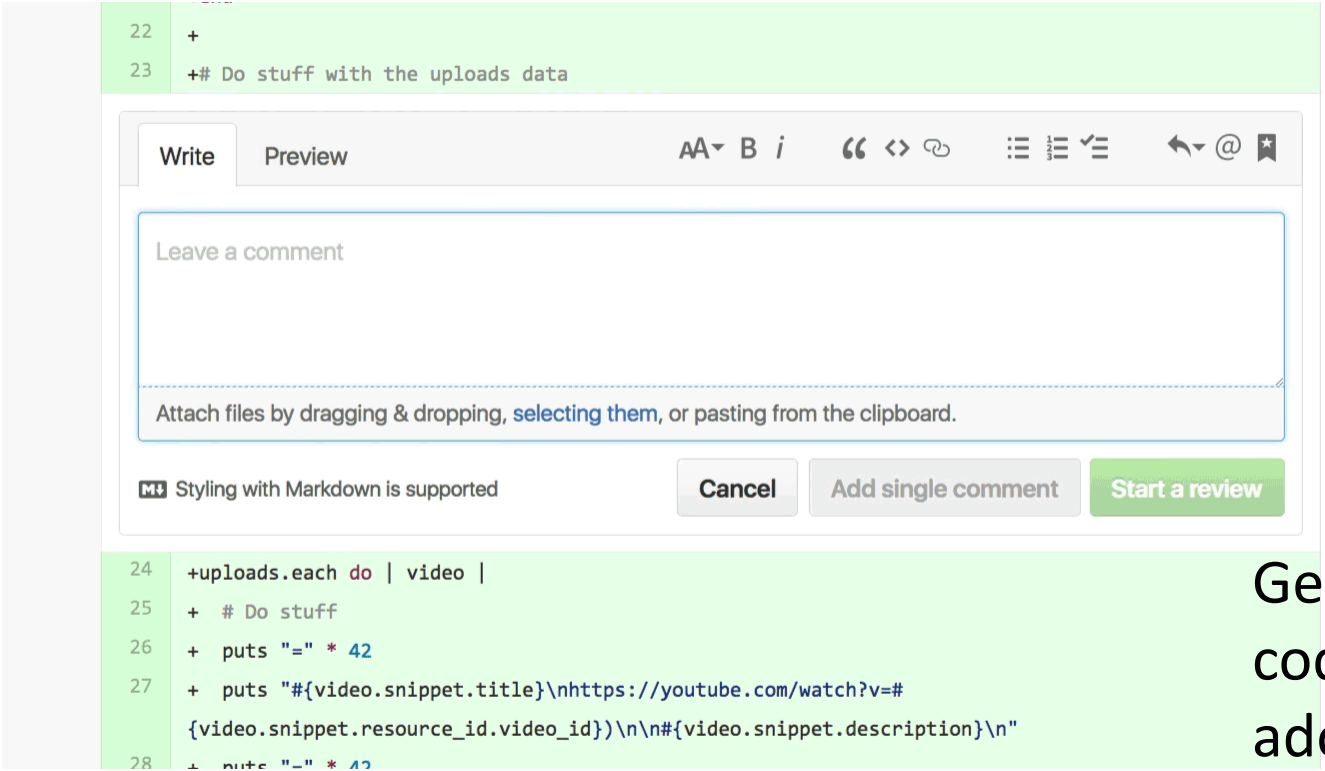


1. Authors (developers) create pull requests to review their code commits

A pull request identifies:

- **which files have been changed**
- **who should review the code commit**

A snapshot of code review

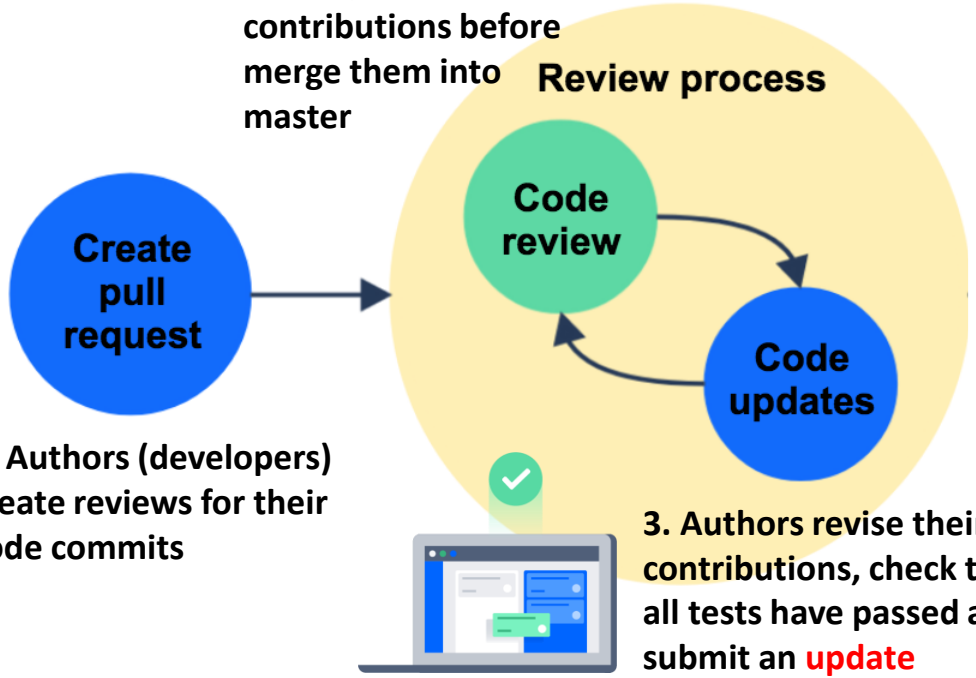


Gerrit – a Git-specific code review system adopted by Google

Code Review Process

2a. Reviewers (look through the diffs to compare the changes with existing code

2. Reviewers (senior developers) review contributions before merge them into master

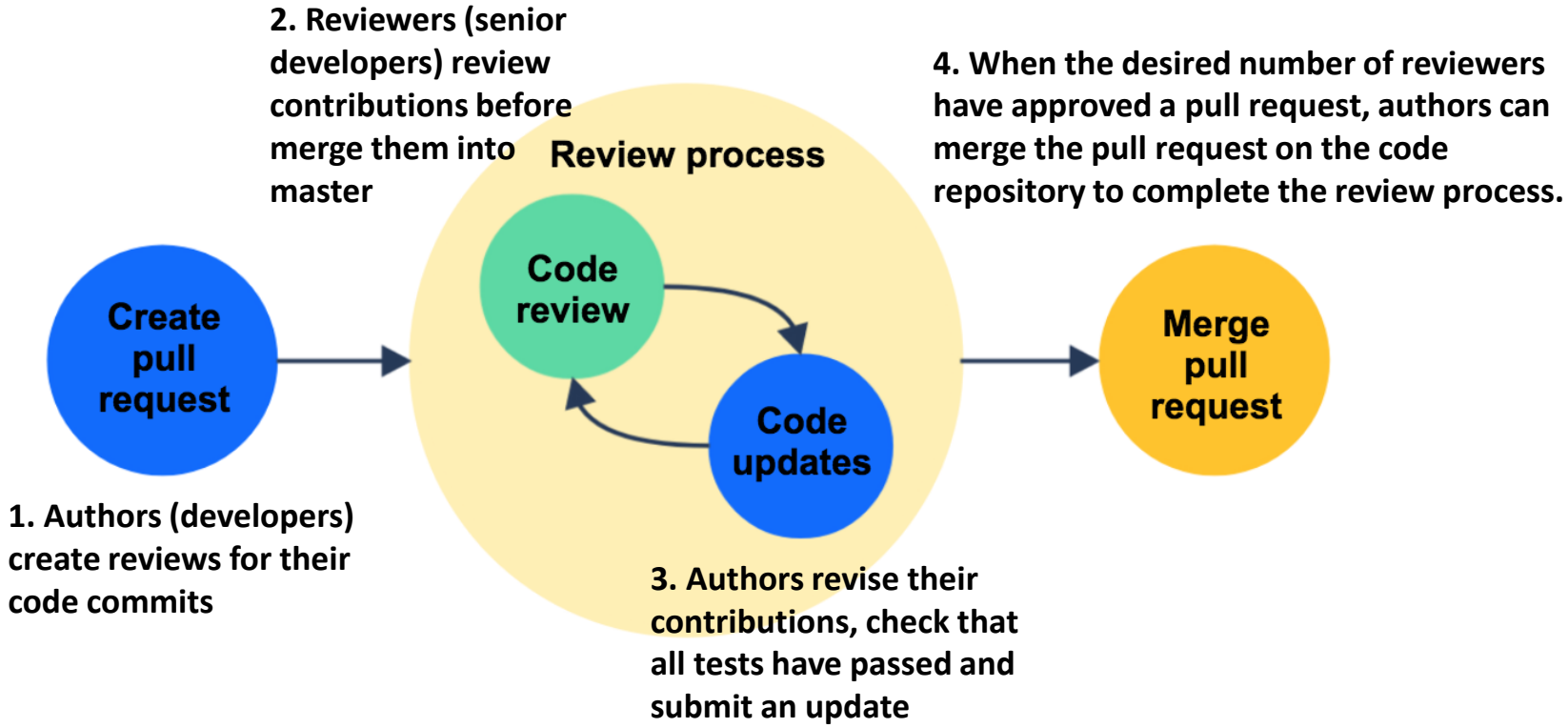


1. Authors (developers) create reviews for their code commits

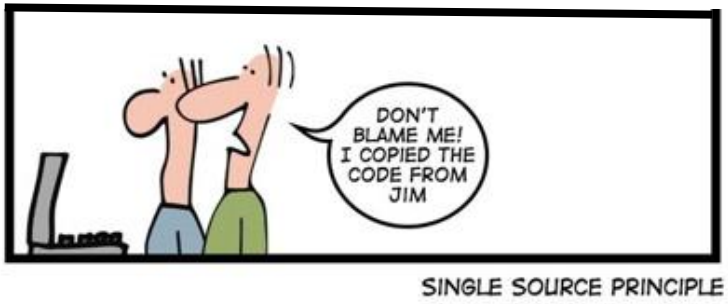
3. Authors revise their contributions, check that all tests have passed and submit an update

2b. Reviewers comment on changes and authors reply to comments, starting a discussion

Code Review Process



Funny Dialogs in Code Review



Features commonly used for Code Review Mining

- Code features
  - #commits, #line_changed, #file_changed, #comments, #participants
- Project features
  - team_size, test coverage, #total commits
- Developer features
  - proportion of approved commits, reputation

Finding 1:

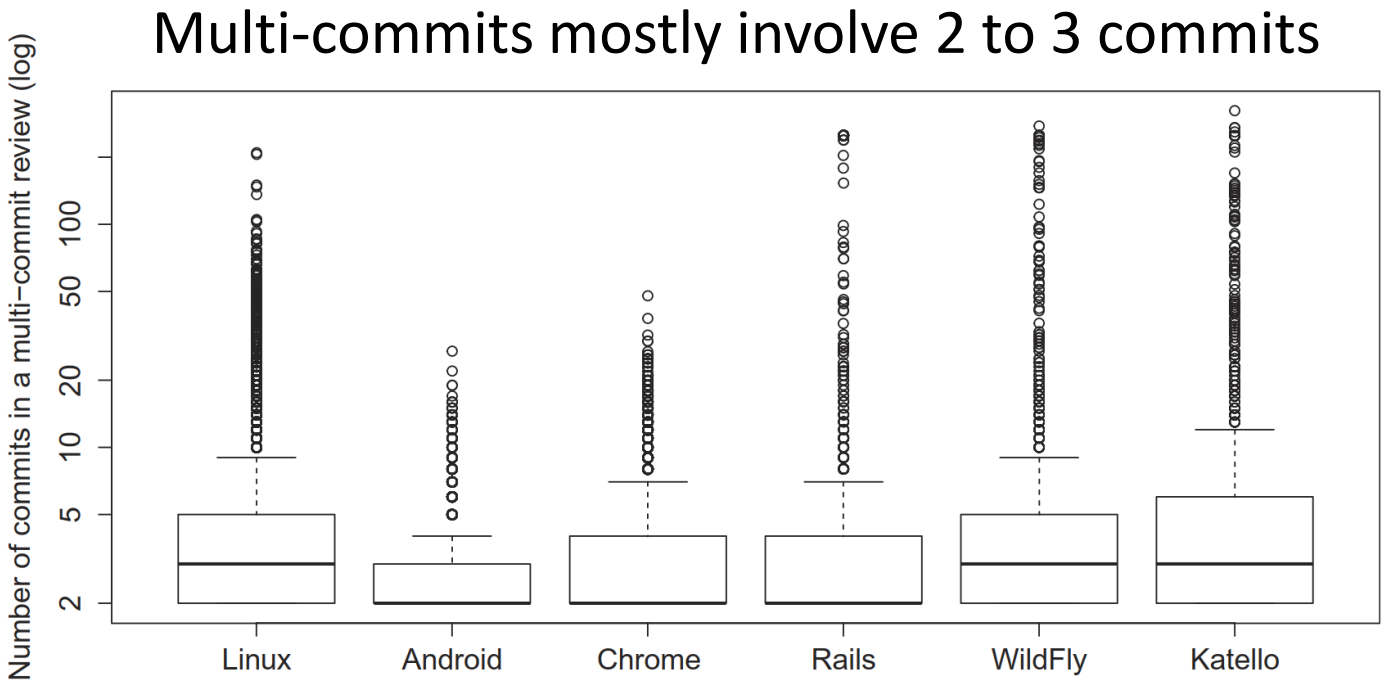
Table 9.3 Number of Reviews in our Dataset and Number of Commits per Review

Project	Reviews	Single Commit (%)	Multicommit (%)
Linux Kernel	20,200	70	30
Android	16,400	66	34
Chrome	38,700	37	63
Rails	7300	71	29
Katello	2600	62	38
WildFly	5000	60	40

Review made for multiple commits: often involves a feature that is broken into multiple changes (i.e., multiple commits).

P. Rigby, et al. A Mixed Methods Approach to Mining Code Review Data: Examples and a Study of Multicommit Reviews and Pull Requests. In Ch 9 of book: Analyzing Software Data, Publisher: Morgan Kaufmann, Editors: Christian Bird and Tim Menzies and Thomas Zimmermann, 2015.

Size of Multicommits



Finding 2: Quality of Patch Commits

- A study found that 50% to 75% of patch commits in Apache, Python and PostgreSQL were rejected.
- Another study found that 61% to 76% of patch commits for Firefox and Mozilla were rejected.

- C. Bird, A. Gourley, P. Devanbu. Detecting Patch Submission and Acceptance in OSS Project, MSR07.
- M. Nurolhzade, SM Nasehi, et al. The Role of Patch Review in Software Evolution: An Analysis of the Mozilla Firefox. IWPSE09.
- K. Jeong, T. Zimmermann. Improving Code Review by Predicting Reviewers and Acceptance of Patches. Technical Memorandum ROSAEC-2009-06, Seoul National University.

Major Rejection Reasons in Code Review

Table 9.4 Reasons for Rejecting Code Under Review

Reason	Description	Percentage
obsolete	The pull request is no longer relevant, as the project has progressed	4
conflict	The feature is currently being implemented by other pull requests or in another branch	5
superseded	A new pull request solves the problem better	18
duplicate	The functionality was in the project prior to the submission of the pull request	2
superfluous	The pull request does not solve an existing problem or add a feature needed by the project	6
deferred	The proposed change is delayed for further investigation in the future	8
process	The pull request does not follow the correct project conventions for sending and handling pull requests	9
tests	Tests failed to run.	1
incorrect implementation	The implementation of the feature is incorrect, is missing, or does not follow project standards.	13
merged	The pull request was identified as merged by the human examiner	19
unknown	The pull request could not be classified owing to lacking information	15

RQ1: Can code review be automated?

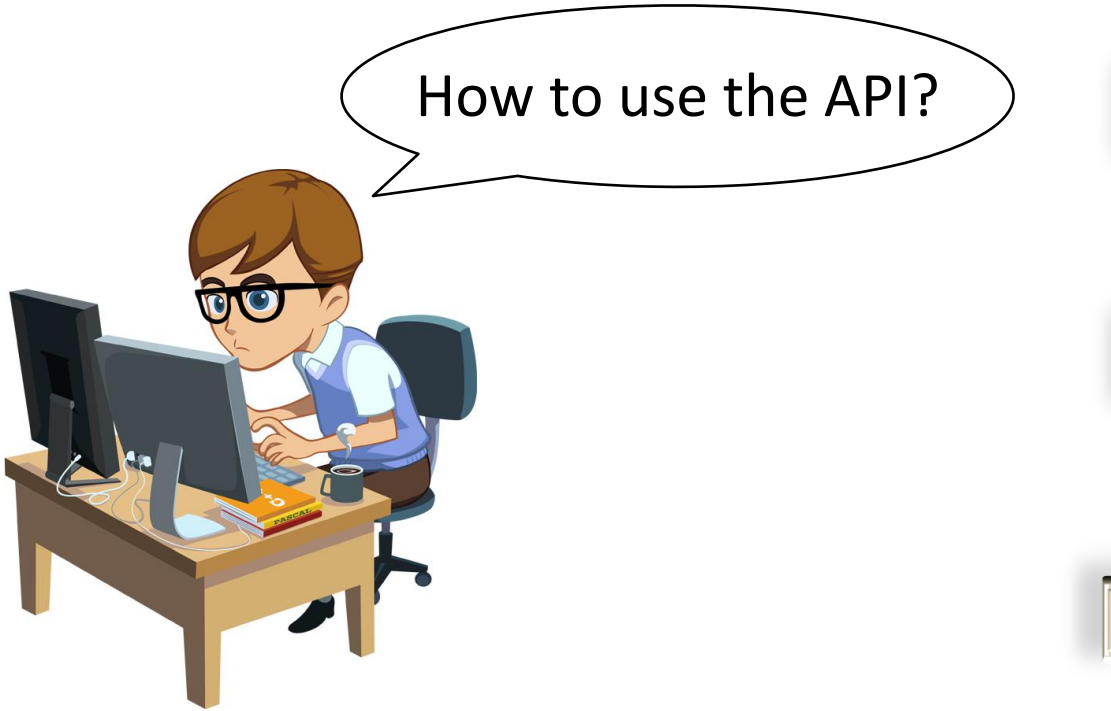
RQ2: Are commits with some of these properties more error prone?

Mining Program Code

Two illustrations:

1. API usage
2. What invariants are preserved by an API

Using unfamiliar API is common …



Usage of Library APIs (OP-Miner)

```
public Stack createStack() {
  Random r = new Random();
  int n = r.nextInt();
  Stack s = new Stack();
  int i = 0;
  while (i < n) {
    s.push(rand(r));
    i++;
  }
  s.push(-1);
  return s;
}
```

Objective: To mine from a method how the APIs of Stack and Random should be used

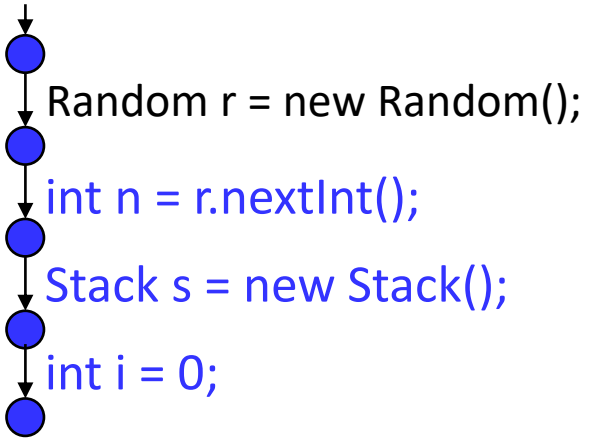
A. Wasylkowski, A. Zeller. Mining Operational Preconditions. Technical Report, Saarland University, Dec. 2008.

Usage of Library APIs

```
public Stack createStack() {
  Random r = new Random();
  int n = r.nextInt();
  Stack s = new Stack();
  int i = 0;
  while (i < n) {
    s.push(rand(r));
    i++;
  }
  s.push(-1);
  return s;
}
```

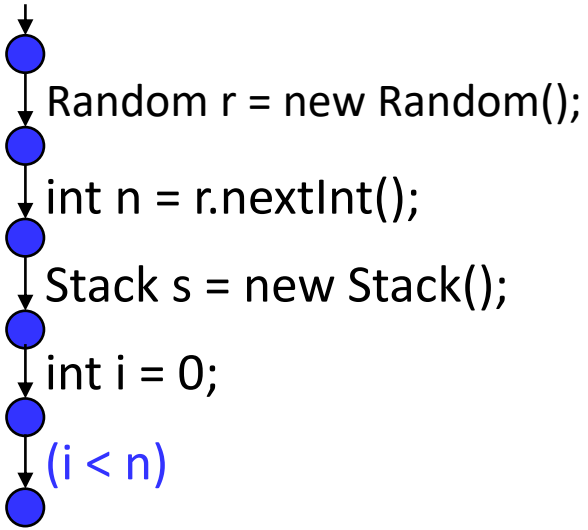
Usage of Library APIs

```
public Stack createStack() {
  Random r = new Random();
  int n = r.nextInt();
  Stack s = new Stack();
  int i = 0;
  while (i < n) {
    s.push(rand(r));
    i++;
  }
  s.push(-1);
  return s;
}
```



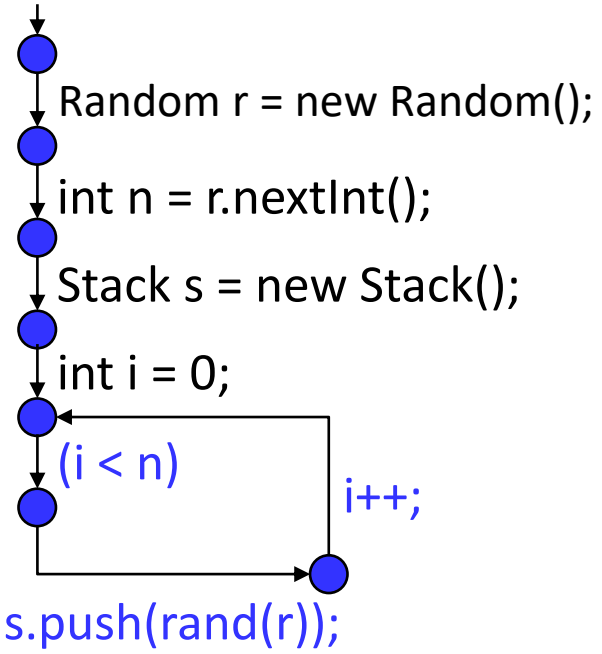
Usage of Library APIs

```
public Stack createStack() {
  Random r = new Random();
  int n = r.nextInt();
  Stack s = new Stack();
  int i = 0;
  while (i < n) {
    s.push(rand(r));
    i++;
  }
  s.push(-1);
  return s;
}
```



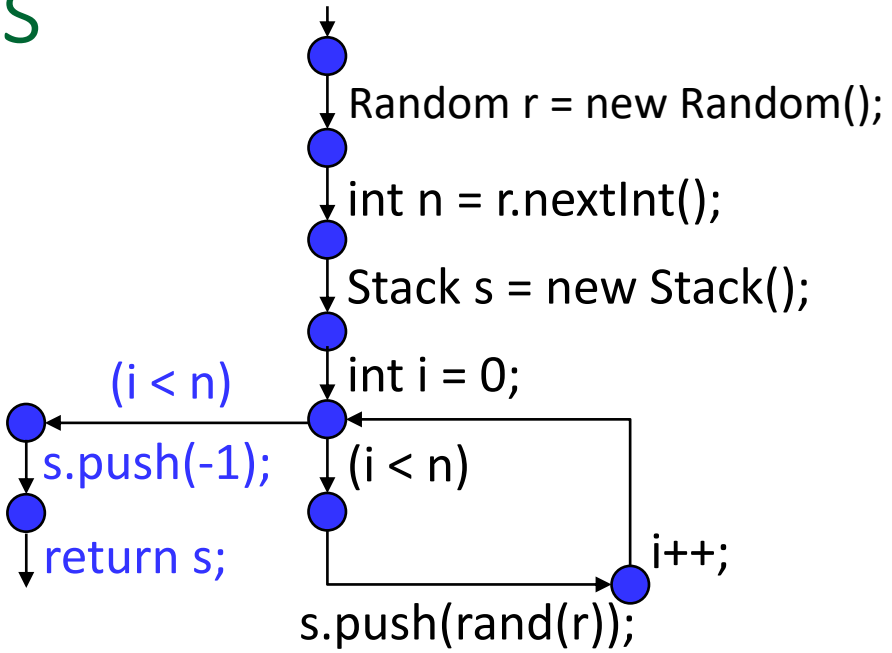
Usage of Library APIs

```
public Stack createStack() {
  Random r = new Random();
  int n = r.nextInt();
  Stack s = new Stack();
  int i = 0;
  while (i < n) {
    s.push(rand(r));
    i++;
  }
  s.push(-1);
  return s;
}
```



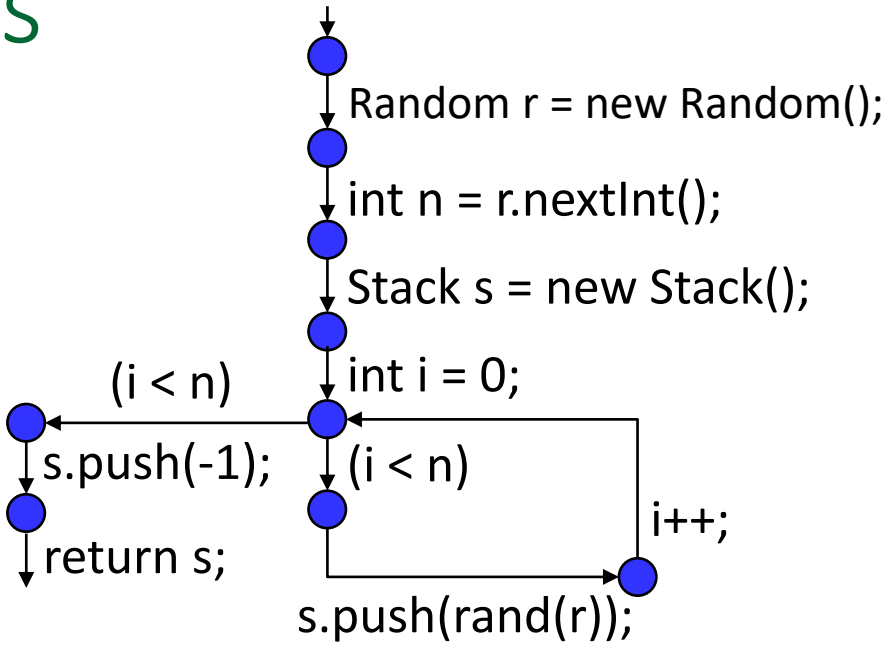
Usage of Library APIs

```
public Stack createStack() {
  Random r = new Random();
  int n = r.nextInt();
  Stack s = new Stack();
  int i = 0;
  while (i < n) {
    s.push(rand(r));
    i++;
  }
  s.push(-1);
  return s;
}
```



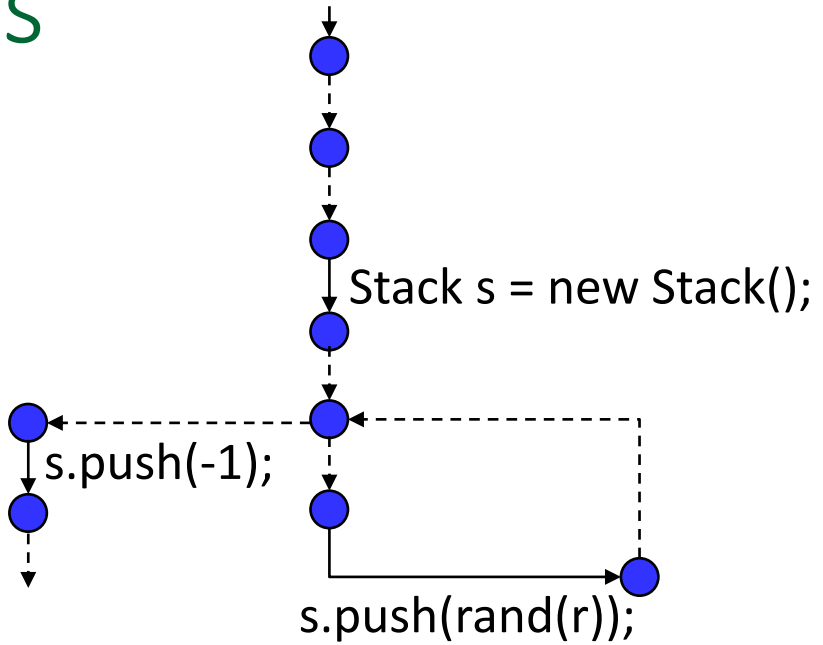
Usage of Library APIs

```
public Stack createStack() {
  Random r = new Random();
  int n = r.nextInt();
  Stack s = new Stack();
  int i = 0;
  while (i < n) {
    s.push(rand(r));
    i++;
  }
  s.push(-1);
  return s;
}
```

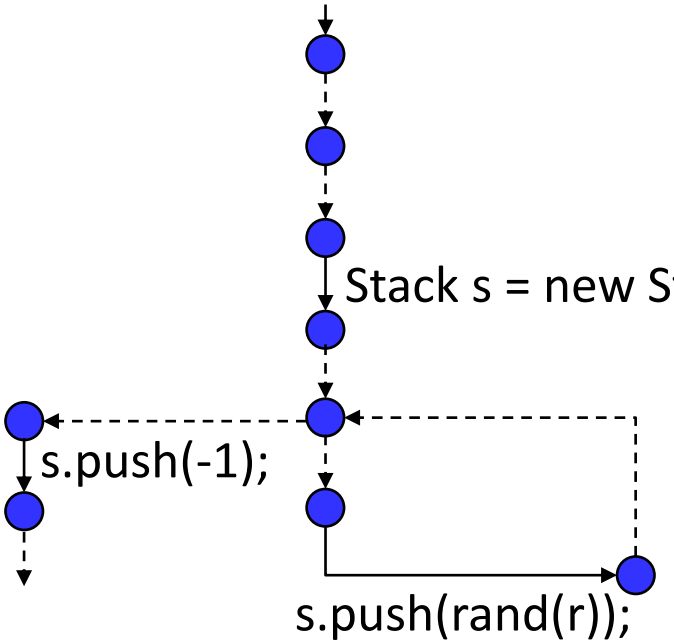


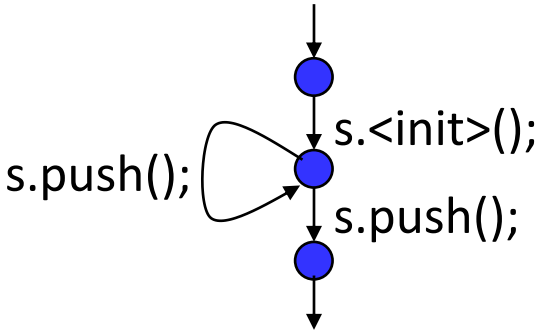
Usage of Library APIs

```
public Stack createStack() {
  Random r = new Random();
  int n = r.nextInt();
  Stack s = new Stack();
  int i = 0;
  while (i < n) {
    s.push(rand(r));
    i++;
  }
  s.push(-1);
  return s;
}
```



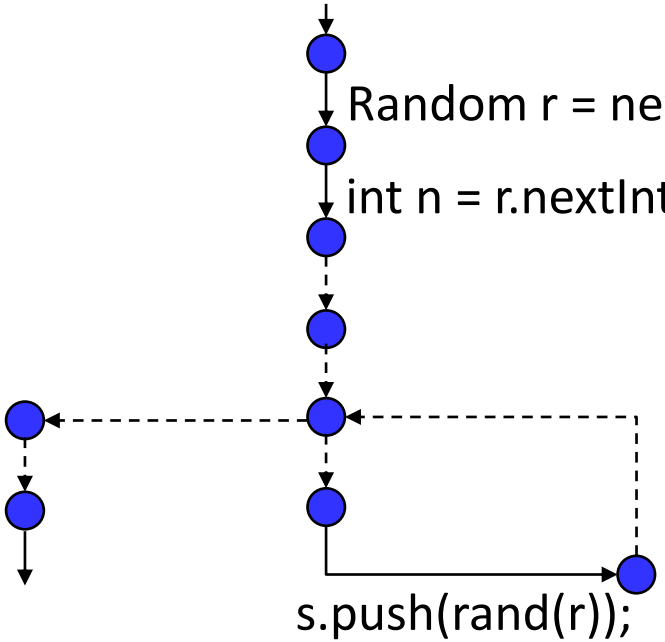
Usage of Library APIs

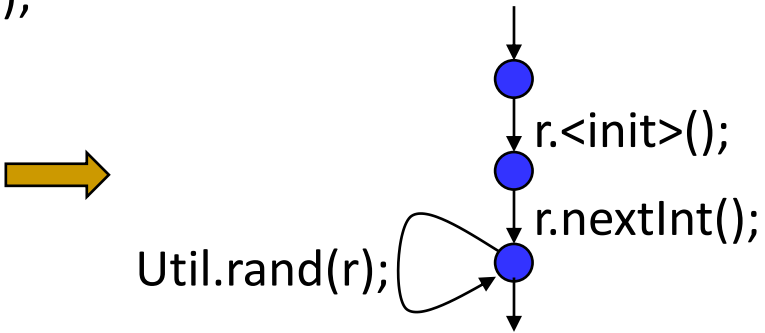




Nodes at both ends of a dashed arrow are merged.

Usage of Library APIs





Food for thoughts: Suggest a possible application.

Recall - Generalizing the Concept

A test requirement is more suspicious if it participates in more failing than passing runs. The test requirement can be:

- statement // *adopted so far in prior discussion*
- statement sequence // *e.g., f.open() … f.close() … f.read()*
- branch // *i.e., a specific evaluation of a predicate*
- prime path
- def-use pair
- mutant

Why Method Invariants?

- Invariants are some properties that always hold in method execution
  - Implicit assumptions on parameters
    - Improve method robustness
    - Reduce irrelevant generated tests
  - Intention of the method
    - Validate consistency with comments and API documentation
    - Identify implementation substitute
    - Identify fix ingredients in automated program repair

Invariant Pattern:

A == B

Variables:

s size(b[ ])

sum(b[ ])

n orig(n)

return…

Why Method Invariants?

- Other invariant patterns
  - A >= B, A > B, A <= B, A < B, A != B
  - …
- May collect common patterns from documentation and comments in open-source projects and public forums
- Invariant variables
  - Parameters, local variables or return values
  - Predicates formed by invariant variables

Invariant Pattern:

A == B

Variables:

s size(b[ ])

sum(b[ ])

n orig(n)

return…

Mining API invariants (Daikon)

```
public int libapi(int[ ] b, int n) {
  int s = 0;
  int i = 0;
  while (i != n) {
    s = s + b[i];
    i = i + 1;
  }
  return s;
}
```

from various tests elsewhere:

run_1
run_2
…
run_n

e.g.,
…
o.libapi([1,2,3], 3);
…

Pattern: **A == B**

Variables:

s **size(b[ ])**
sum(b[ ])
n **orig(n)**
return…

"The Daikon system for dynamic detection of likely invariants" by Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, and Chen Xiao. Science of Computer Programming, vol. 69, no. 1--3, Dec. 2007, pp. 35-45.

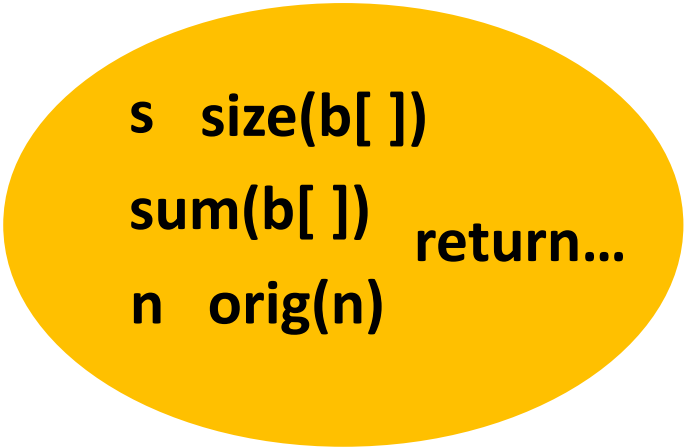
Mining API Invariants

A separate table for each invariant pattern

==	s	n	size(b[])	sum(b[])	orig(n)	ret
s		×			×	
n	×			×		×
size(b[])						
sum(b[])		×				
orig(n)	×					×
ret		×			×	

run 1

A == B **Pattern**



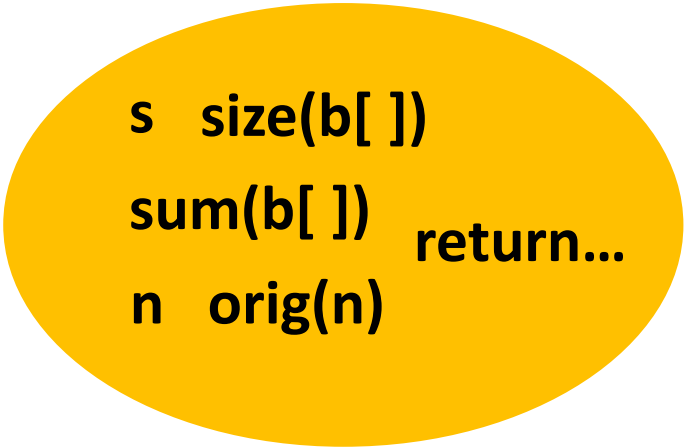
Variables

Mining API Invariants

==	s	n	size(b[])	sum(b[])	orig(n)	ret
s		×	×		×	
n	×			×		×
size(b[])	×			×		×
sum(b[])		×	×		×	
orig(n)	×	×		×		×
ret		×	×		×	

run 2

A == B **Pattern**



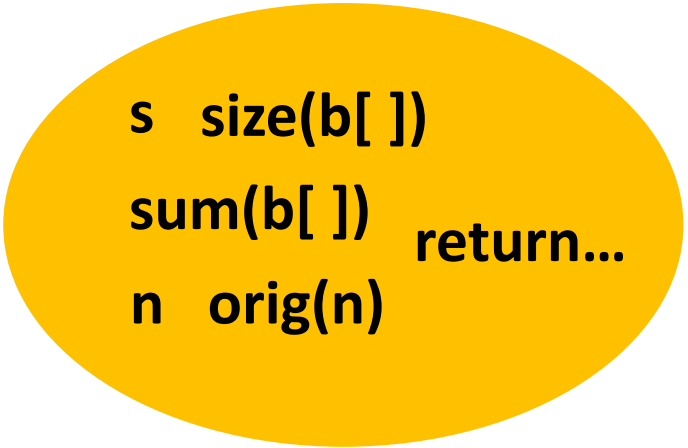
Variables

Mining API Invariants

==	s	n	size(b[])	sum(b[])	orig(n)	ret
s		×	×		×	
n	×			×		×
size(b[])	×			×		×
sum(b[])		×	×		×	
orig(n)	×	×		×		×
ret		×	×		×	

run 3

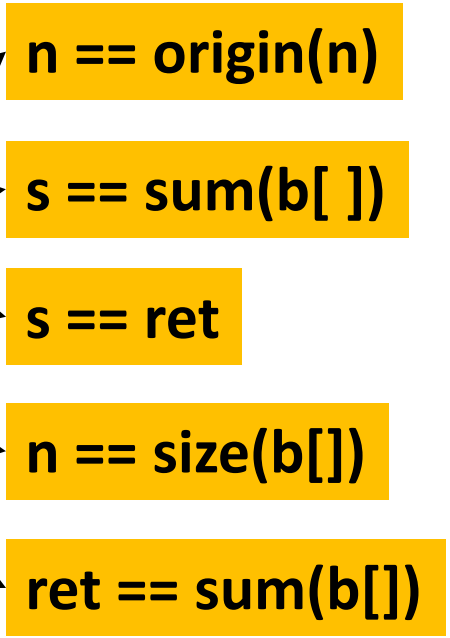
A == B **Pattern**



Variables

Mining API Invariants

==	s	n	size(b[])	sum(b[])	orig(n)	ret
s		×	×		×	
n	×			×		×
size(b[])	×			×		×
sum(b[])		×	×		×	
orig(n)	×	×		×		×
ret		×	×		×	



Mining API invariants

```
public int libapi(int[ ] b, int n) {
  int s = 0;
  int i = 0;
  while (i != n) {
    s = s + b[i];
    i = i + 1;
  }
  return s;
}
```

n == origin(n)

s == sum(b[ ])

s == ret

n == size(b[])

ret == sum(b[])

What if tests are randomly generated?

Mining API invariants

public int libapi(int[ ] b, int n) { … }

- ✓ obj.libapi(new int[0], 0)
- ✘ obj.libapi(null, 0)
- ✘ obj.libapi(arr, -1)
- ✓ obj.libapi(new int[] {0, 1}, 2)

n == origin(n)

s == sum(b[ ])

s == ret

n == size(b[])

ret == sum(b[])

- replace x=libapi(new int[] {0, 1}, 2); with x=0+1;

Mining API invariants

public int libapi(int[ ] b, int n) { … }

- ✓ obj.libapi(new int[0], 0)
- ✘ obj.libapi(null, 0)
- ✘ obj.libapi(arr, -1)
- ✓ obj.libapi(new int[] {0, 1}, 2)

Lessons learnt?

We cannot create knowledge from code generated without knowledge. But we can extract developers' knowledge via their program code written

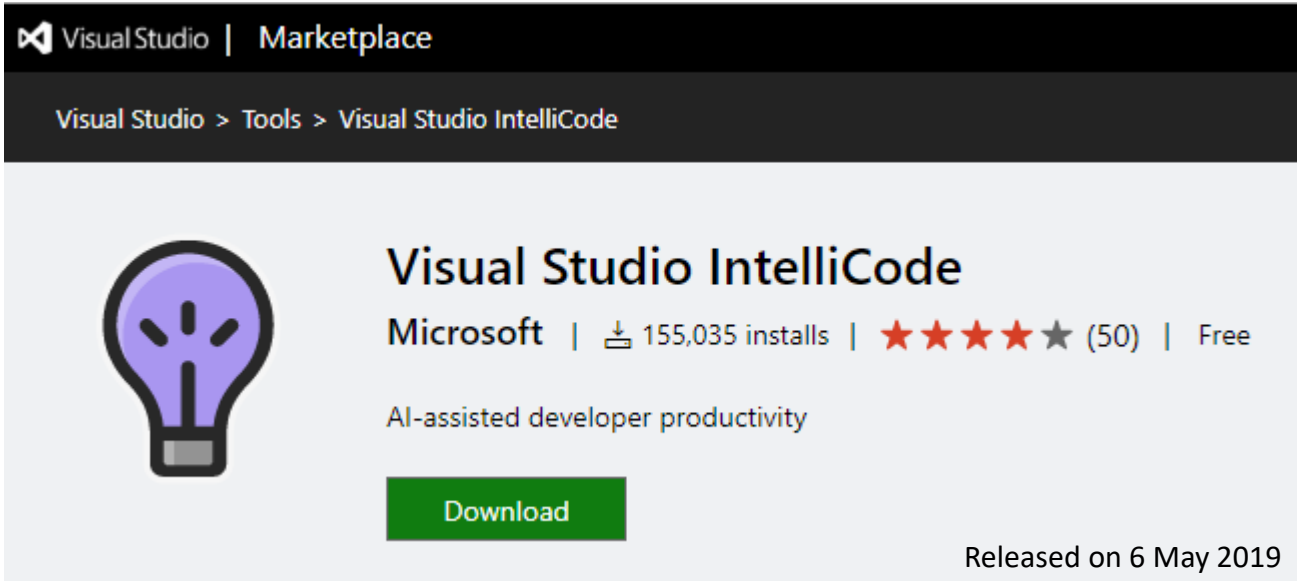
- replace x=libapi(new int[] {0, 1}, 2); with x=0+1;

Upcoming Trend

Increasing use of software analysis, data mining and machine learning to extract coding wisdom from public software repositories.

Emerging industrial deployment on code completion

Recent Effort by Microsoft



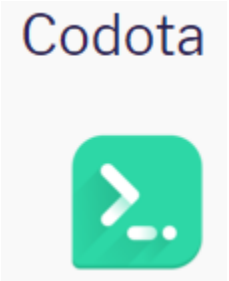
Visual Studio IntelliCode (IntelliCode) is a set of AI-assisted capabilities that improve developer productivity with features like contextual IntelliSense, argument completion, code formatting, and style rule inference.

https://marketplace.visualstudio.com/items?itemName=VisualStudioExptTeam.VSIntelliCode

Codota – Advanced Java Code Completion

Codota completes lines of code based on millions of open source Java programs and your context helping you code faster with fewer errors.
The following features are available with the new version of Codota:

- Developed by an Israeli company
- Full line AI autocomplete
- In line and relevant code examples
- Code suggestions based on your own coding practices
- Available as plugin at Eclipse and IntelliJ marketplaces
- Homepage: **https://www.codota.com/**



Amazon CodeGuru

- A commercial code reviewing cloud service from Amazon since 2020
- To identify code that deviates from common practices
- To identify computationally expensive lines
- Support Java/Scala projects on GitHub and BitBucket
- Triggered by pull requests

https://www.youtube.com/watch?v=oUeSW1STqKY (2:20)

Automated Code Synthesis/Completion

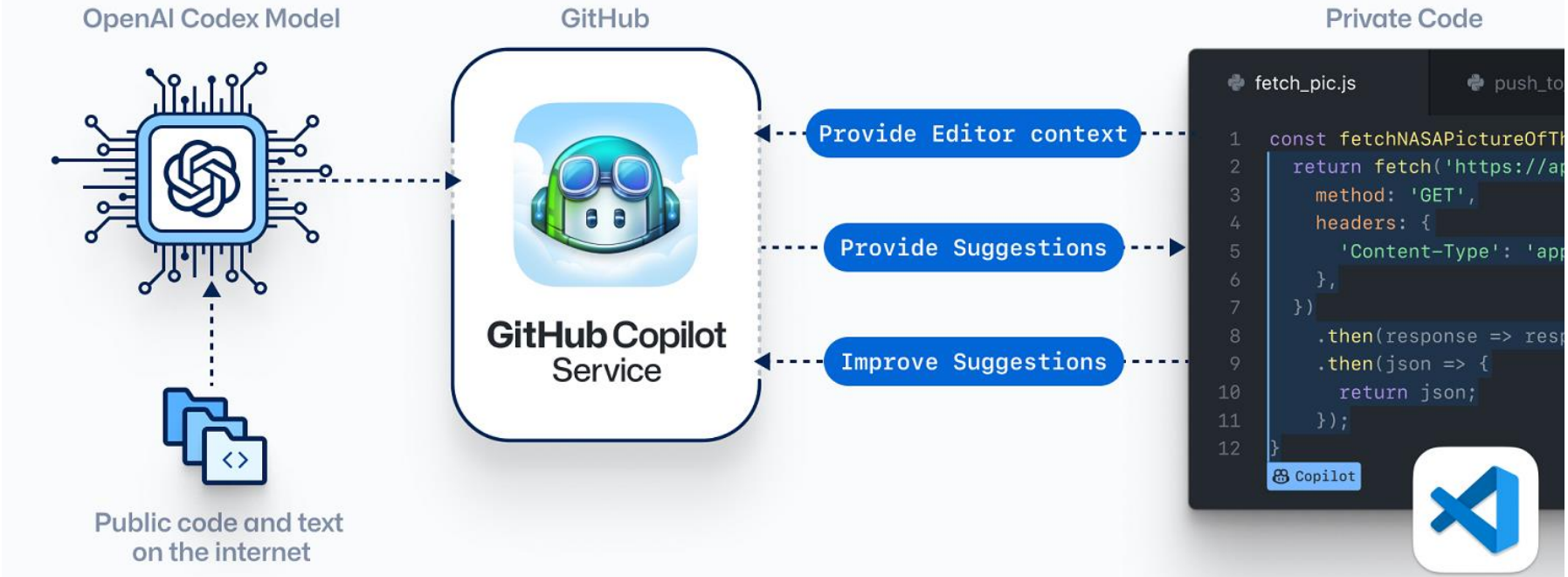
GitHub Copilot

- Launched on 30 June 2021
- Powered by OpenAI Codex, jointly developed by OpenAI and Microsoft
- Usage scenario 1
  - Input: A short comment about what a function intends to do in English
  - Output: Suggestion of implementation code in several supported languages
- Usage scenario 2
  - Input: code written by a programmer
  - Output: Suggestion of further lines or alternative code
- https://copilot.github.com/

Automated Code Synthesis/Completion

- Supports Java, C, C++, C#, Python, JavaScript, TypeScript, Ruby, and Go
- 30% of newly written code for some programming languages is being suggested by Copilot
- Evaluation based on generated Python functions
  - 43% correct on the first attempt
  - 57% correct within 10 attempts
- Trained over terabytes of open-source code on GitHub
- Closed-source

GitHub Copilot – AI Pair Programmer



https://www.youtube.com/watch?v=x_Yw2f161CU **(9:51)**

Weaknesses

- Suggested code may contain bugs
  - 40% of GitHub Copilot generated code can be insecure (https://analyticsindiamag.com/upto-40-percent-of-github-copilot-generated-code-may-be-insecure/)
  - Training dataset can contain insecure coding patterns, bugs, deprecated API references and out-of-date practices
- May sometimes produce undesirable outputs, including biased, discriminatory, abusive, or offensive outputs
- 0.1% chance suggests code the same as that in the training dataset
  → copyright infringement
- Empirical evaluation of Copilot's code (https://arxiv.org/pdf/2108.09293.pdf)
- **GPT Code Clippy** – a community effort to create an open-source version of Copilot based on GPT-Codex (https://github.com/CodedotAl/gpt-code-clippy)
- Very unfriendly to environment

Epilogue

- The data availability is more important than the choice of machine models
  - The choice of model parameter values are more important than the choice of model architectures
  - However, training these parameters even we have enough data is expensive
    - Model used by Copilot: GPT-2 (1.5 Billion); GPT-3 (175 Billion); GPT-4 (100 Trillion)
- Since program code is a kind of languages, the use of transformers is getting popular
- Pre-processing is important
  - The code inputted for prediction is critical in automated code completion
    - Including all code can include a lot of noises
    - Including only the last few lines can exclude important information

Epilogue

- Explainability is a key research direction
  - Instead of requiring the ML model to be interpretable, it can be more useful to devise explainability of each suggested result in the specific problem domain
    - API recommendation -> Relating it to API knowledge graph
    - Autocode completion -> Relating it to dataflow / control flow
    - Automated program repair -> Similarity with some previous commits or StackOverflow posts
    - Code summarization -> Related StackOverflow posts
    - …
- It is a fast-expanding topic in AIOps

More Resources

- Mining Software Repositories (Nov 2020)
  https://www.youtube.com/watch?v=KZi_KdWv8ns
- Top 3 Source Code Repository Hosts
  https://www.youtube.com/watch?v=AuLmDhY3Kjc