

# Code review effectiveness

How code review works (and doesn't) in the real world.

**Alberto Bacchelli**

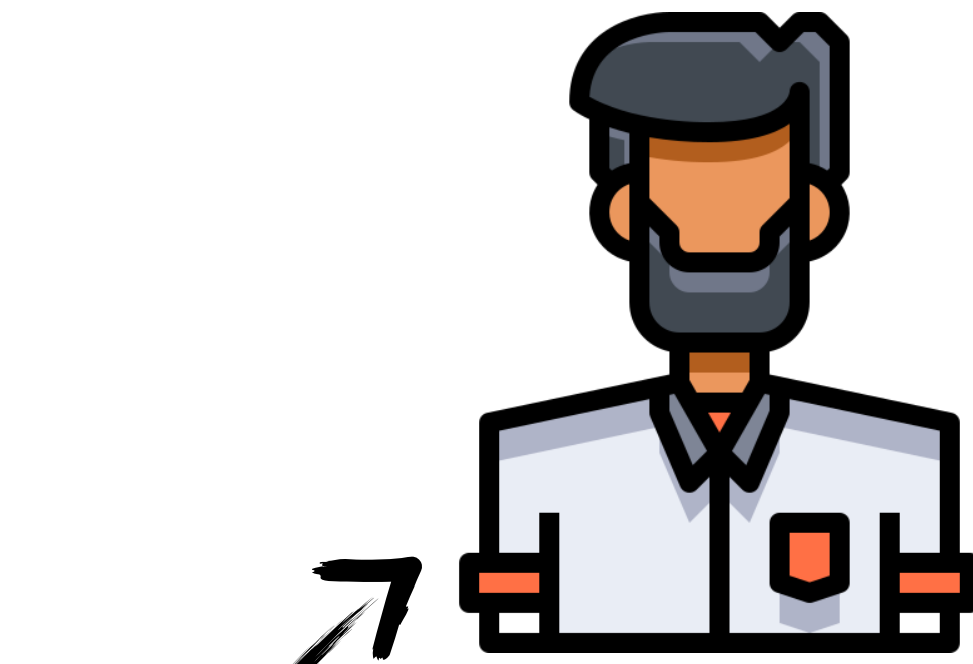
**zest**



University of  
Zurich <sup>UZH</sup>

software system  
history

version i

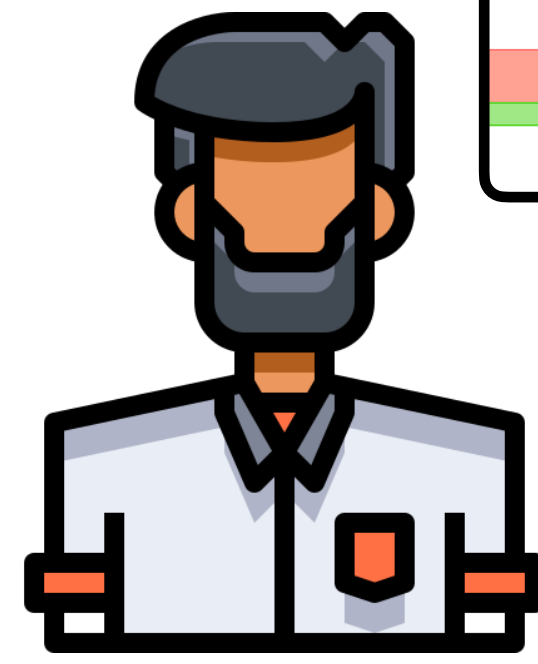
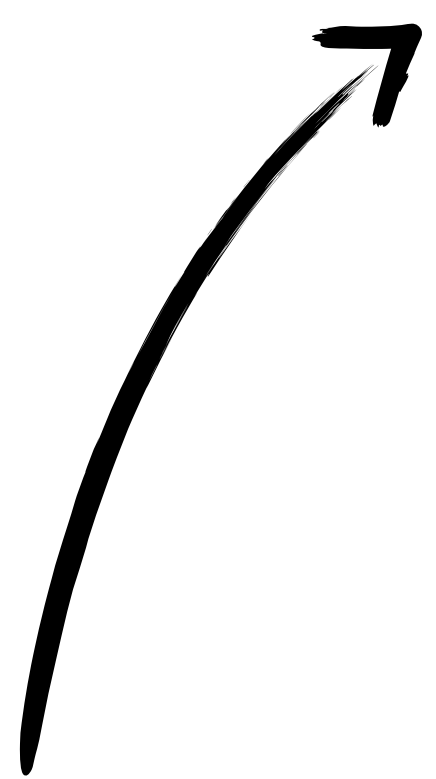


author



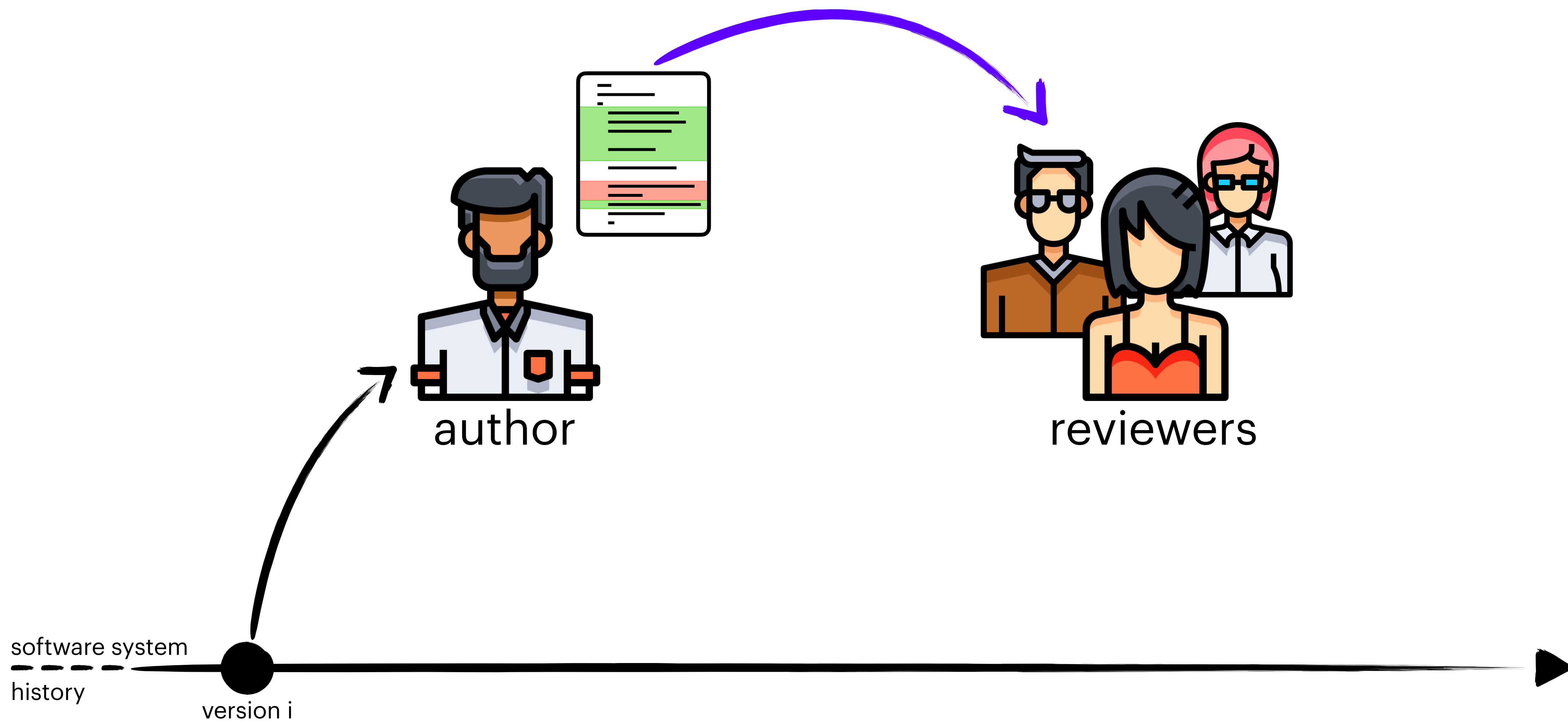
software system  
history

version i



author



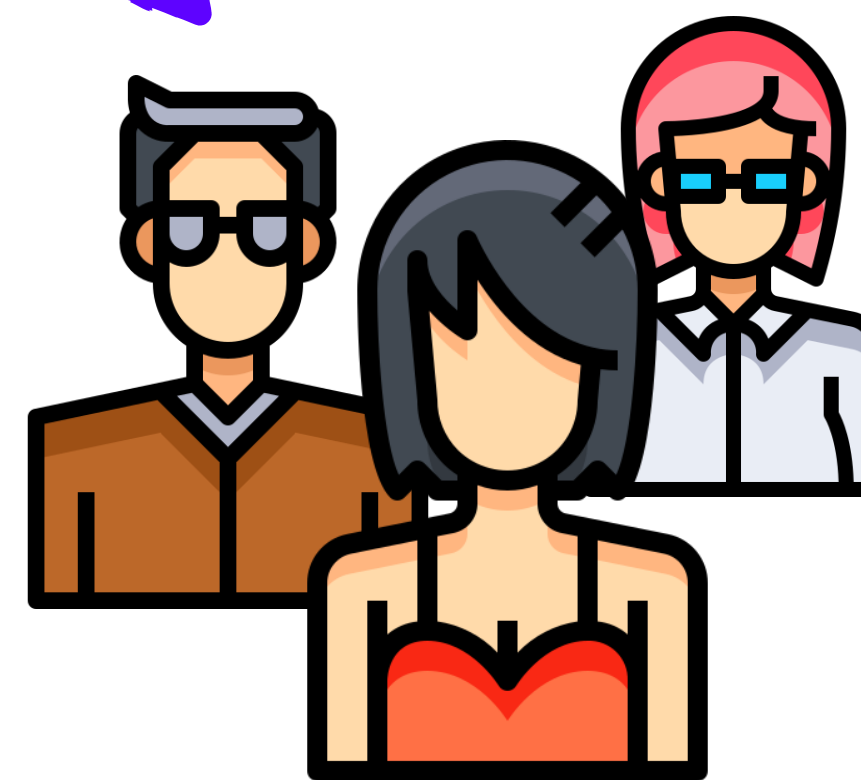


software system  
history

version i

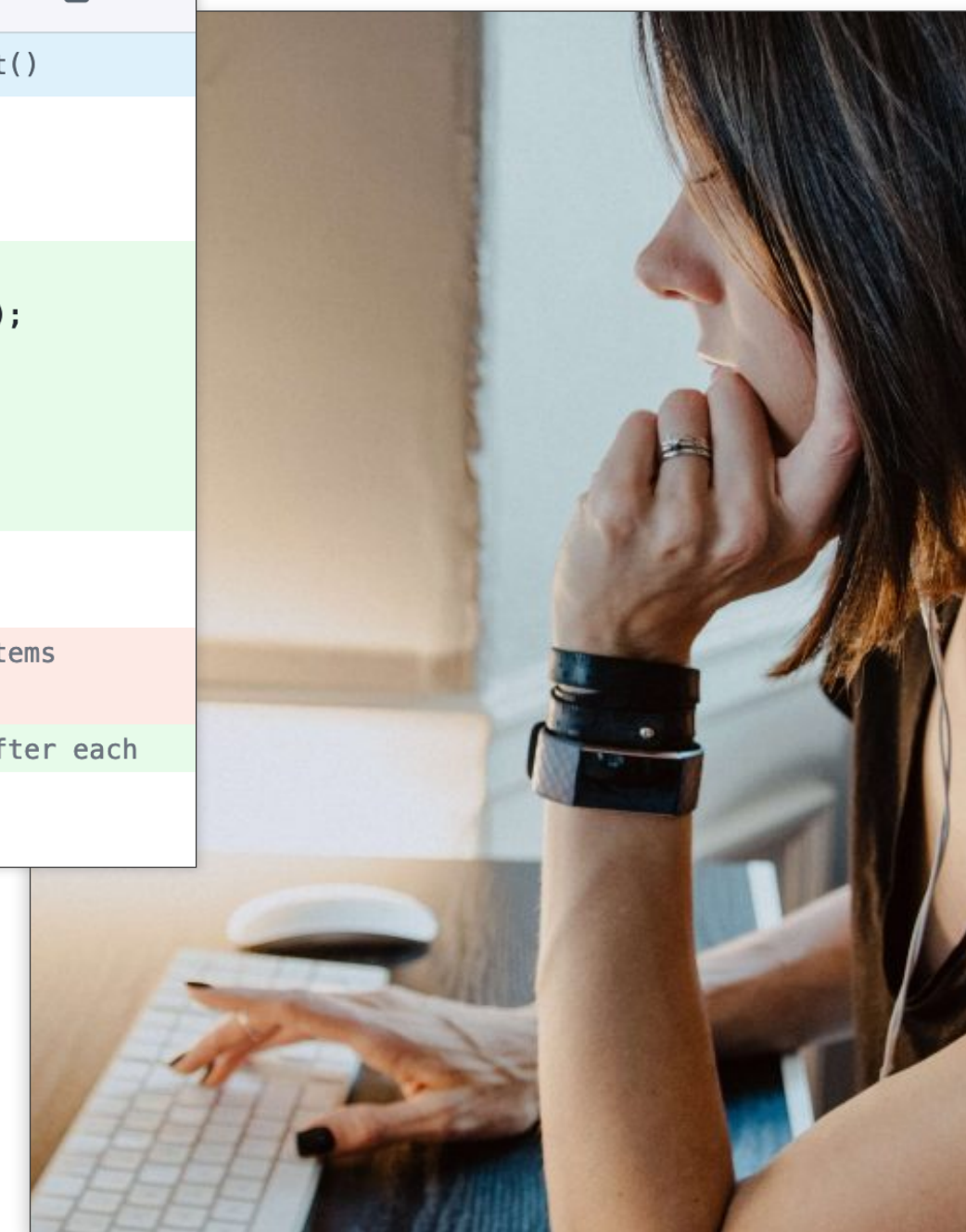


author



reviewers

```
src/System.Collections.Immutable/tests/ImmutableListTest.cs
@@ -164,29 +164,34 @@ public void AddRangeOptimizationsTest()
164 164 [Fact]
165 165 public void AddRangeBalanceTest()
166 166 {
167 + int randSeed = (int)DateTime.Now.Ticks;
168 + Console.WriteLine("Random seed: {0}", randSeed);
169 + var random = new Random(randSeed);
170 +
171 + int expectedTotalSize = 0;
172 +
167 173 var list = ImmutableList<int>.Empty;
168 174
169 - // Add batches of 32, 128 times, giving 4096 items
170 - int batchSize = 32;
175 + // Add some small batches, verifying balance after each
171 176 for (int i = 0; i < 128; i++)
172 177 {
```

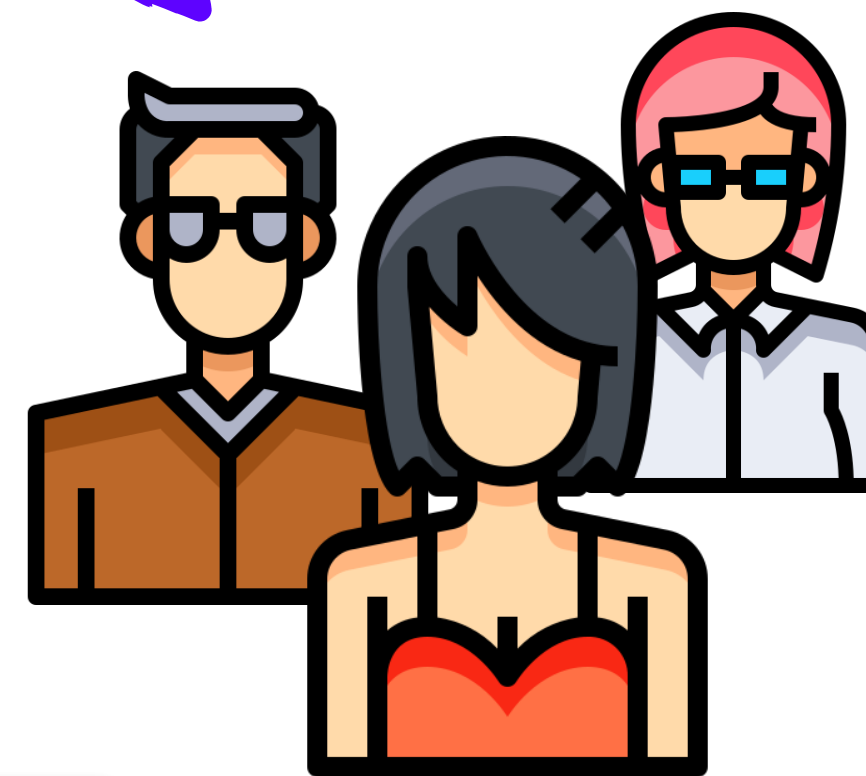


software system  
history

version i

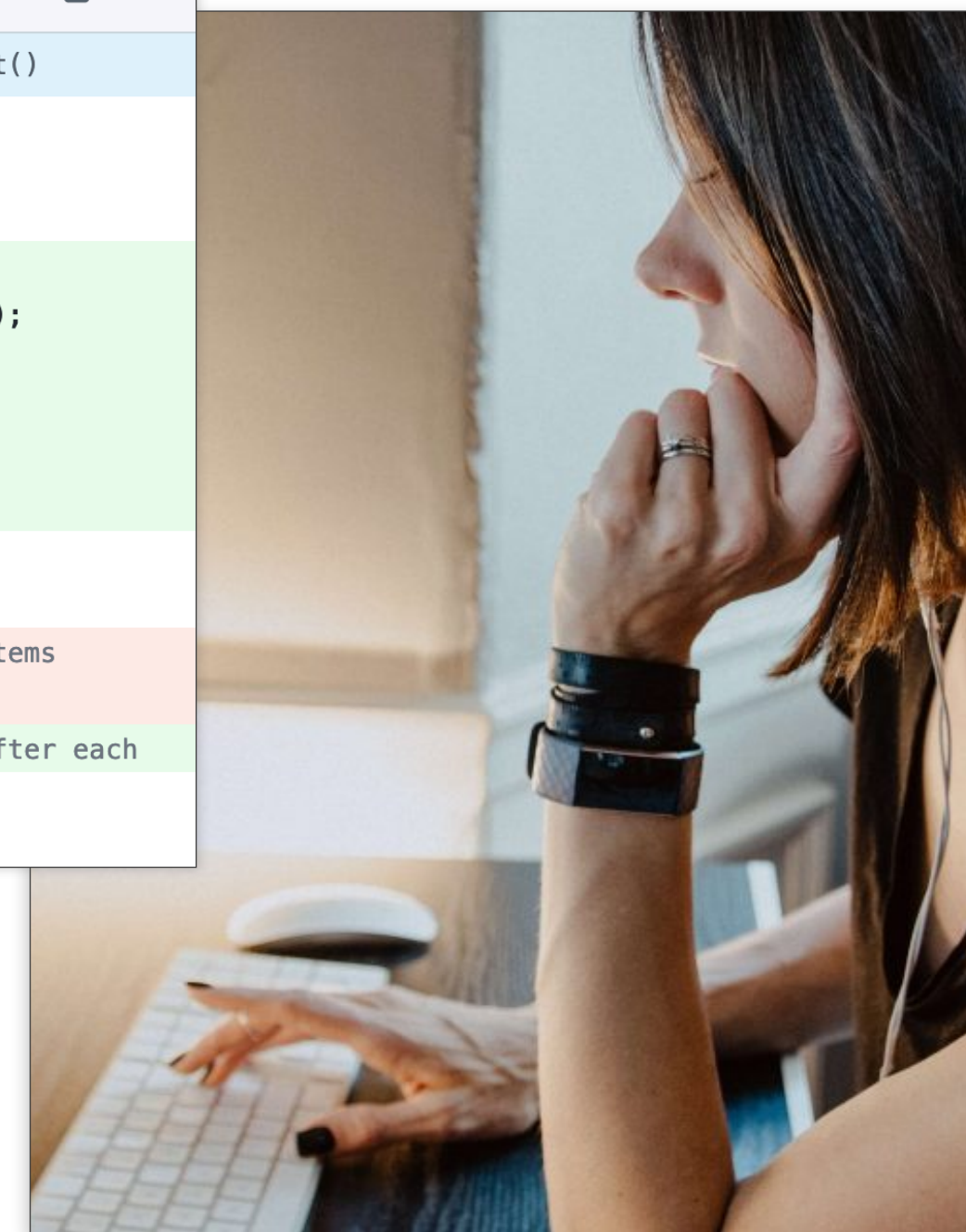


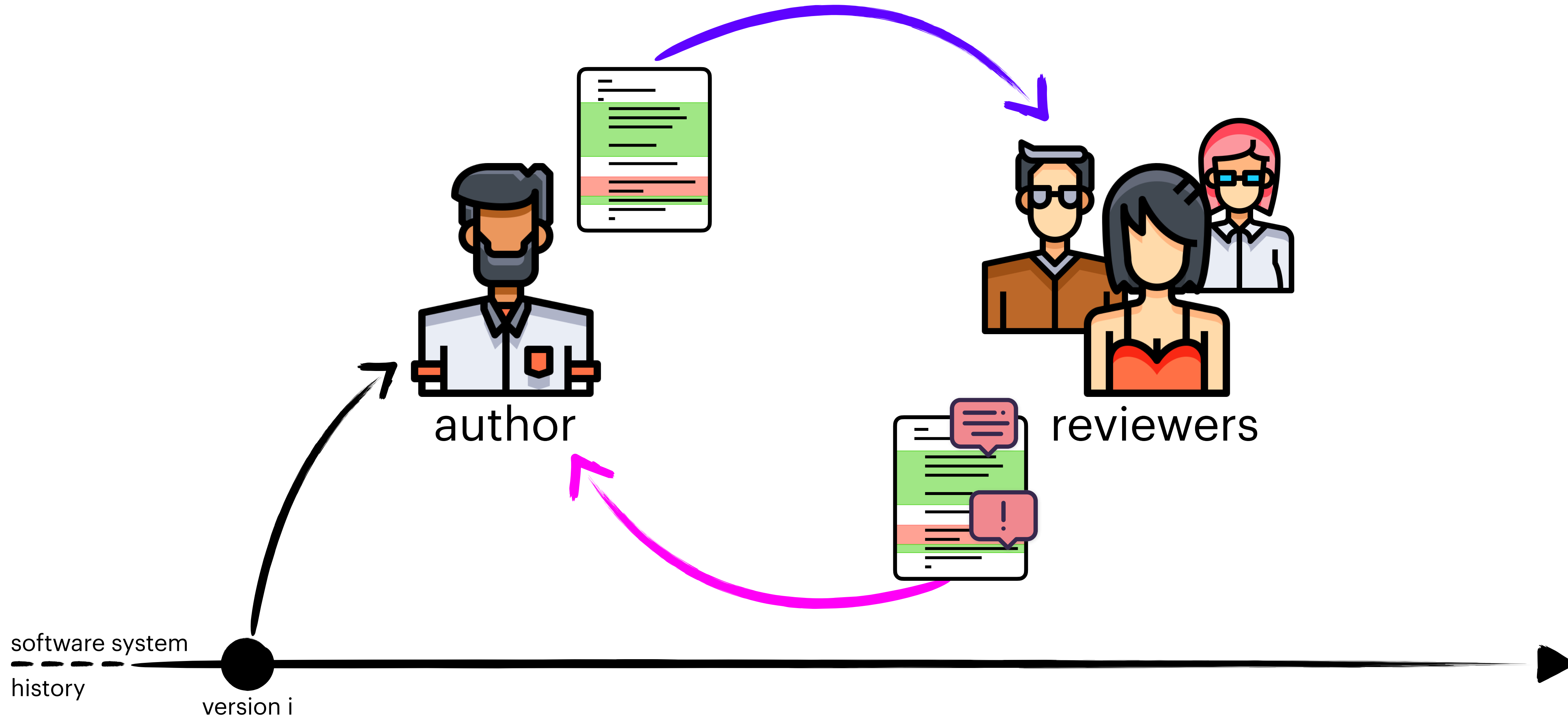
author

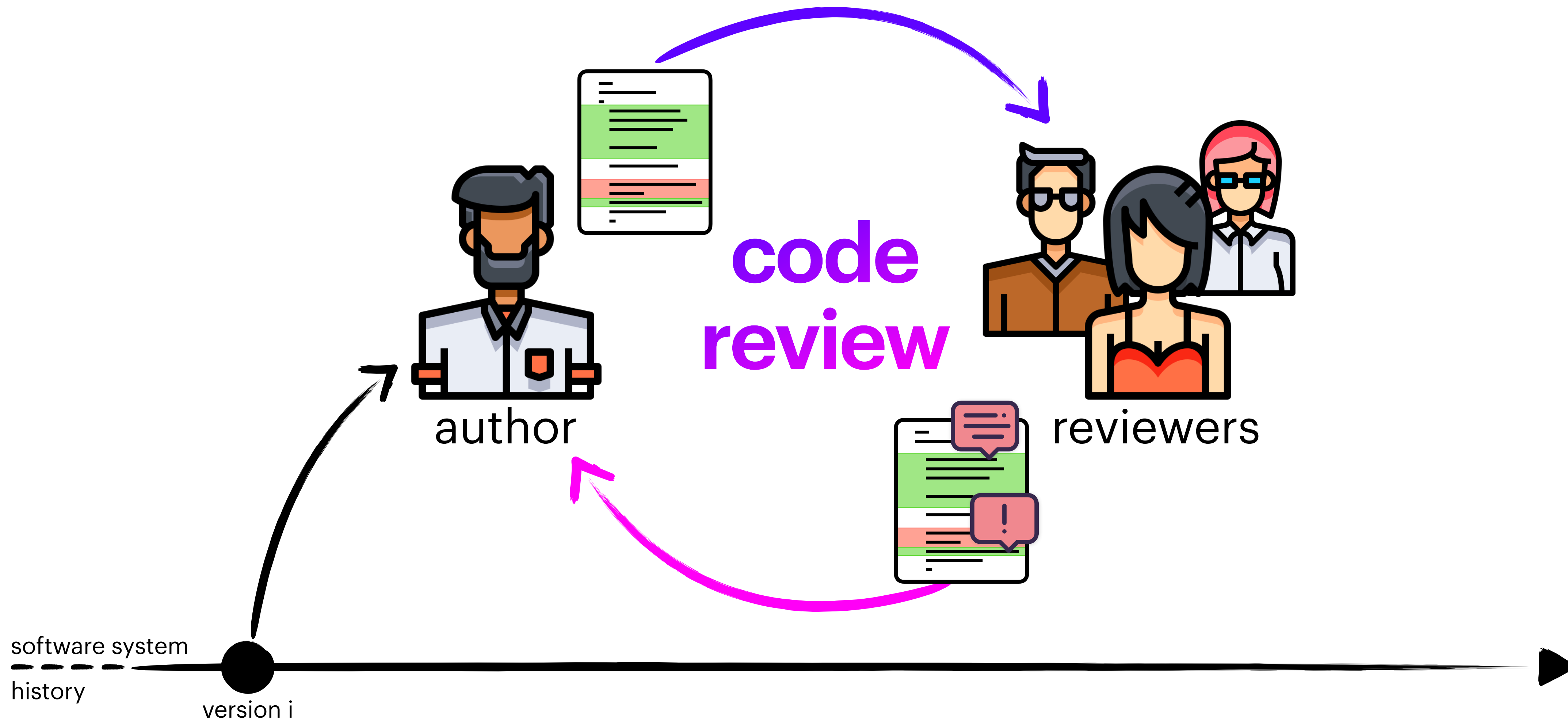


reviewers

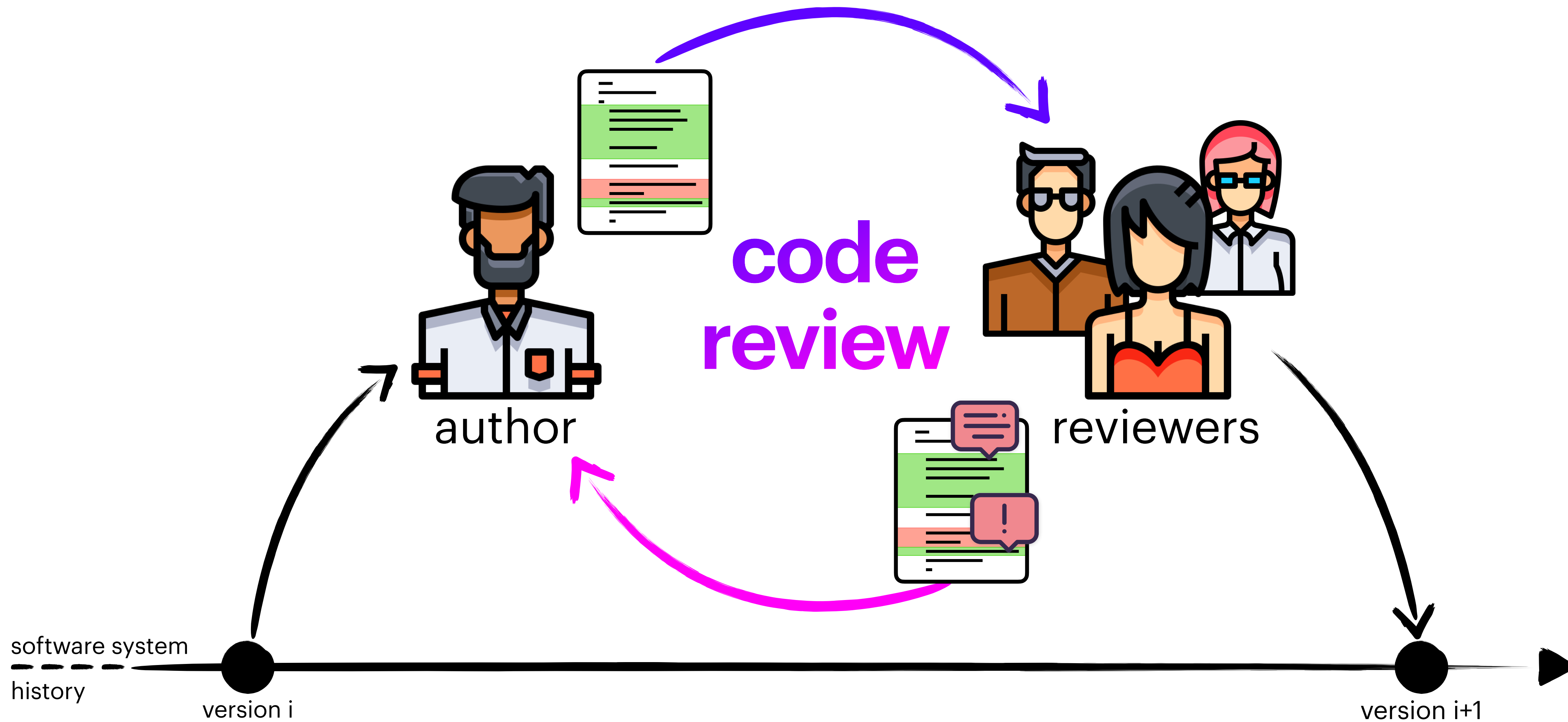
```
src/System.Collections.Immutable/tests/ImmutableListTest.cs
@@ -164,29 +164,34 @@ public void AddRangeOptimizationsTest()
164 164 [Fact]
165 165 public void AddRangeBalanceTest()
166 166 {
167 + int randSeed = (int)DateTime.Now.Ticks;
168 + Console.WriteLine("Random seed: {0}", randSeed);
169 + var random = new Random(randSeed);
170 +
171 + int expectedTotalSize = 0;
172 +
173 + var list = ImmutableList<int>.Empty;
174 +
169 - // Add batches of 32, 128 times, giving 4096 items
170 - int batchSize = 32;
175 + // Add some small batches, verifying balance after each
171 176 for (int i = 0; i < 128; i++)
172 177 {
```





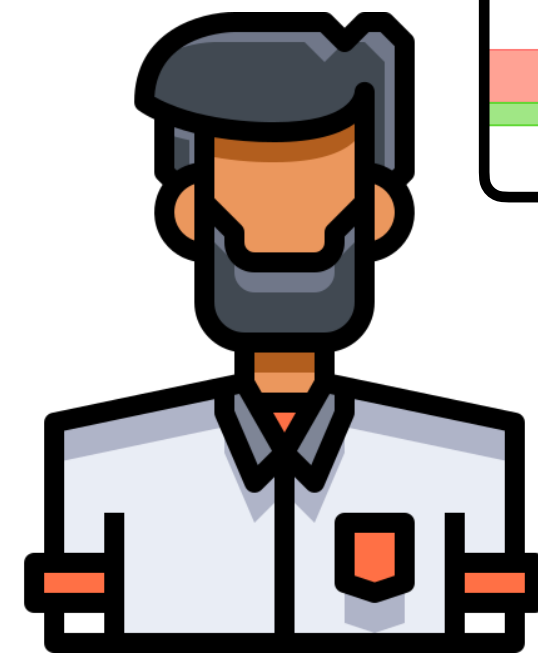






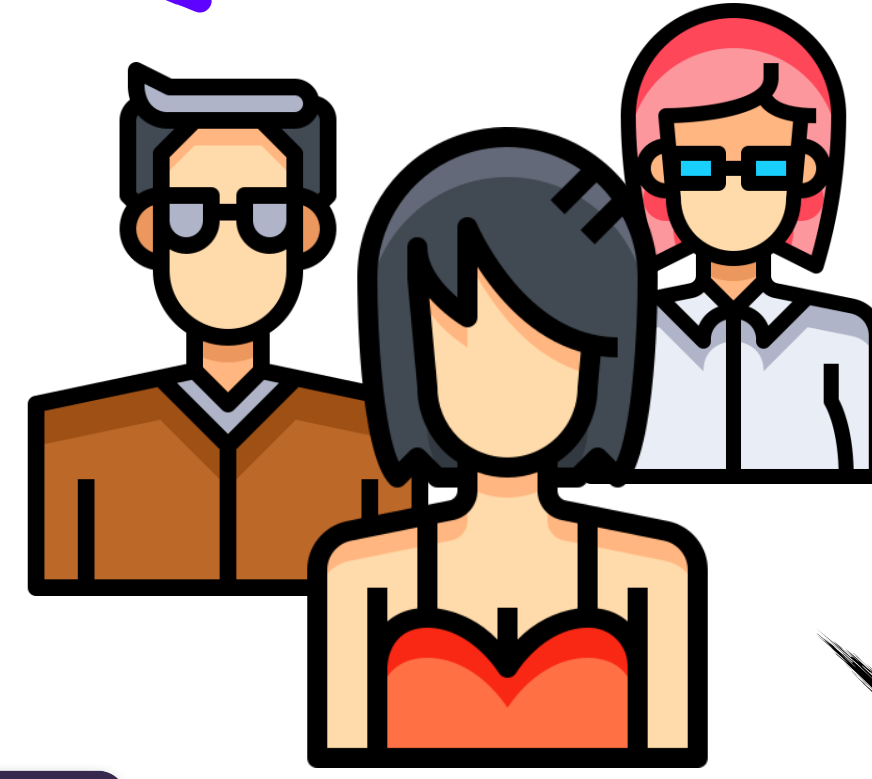
software system  
history

version i



author

# code review

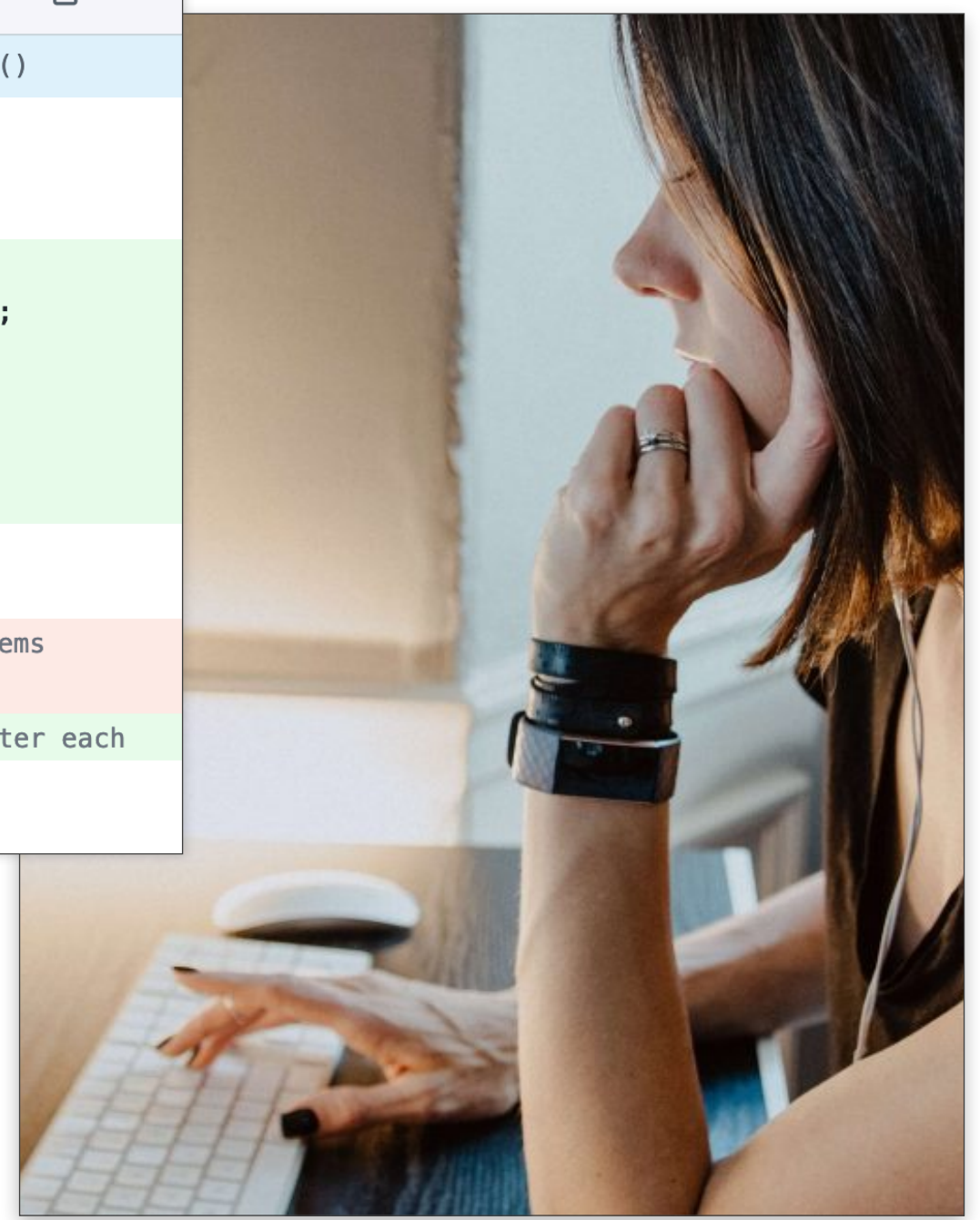


reviewers



version i+1

```
src/System.Collections.Immutable/tests/ImmutableListTest.cs
@@ -164,29 +164,34 @@ public void AddRangeOptimizationsTest()
164 164 [Fact]
165 165 public void AddRangeBalanceTest()
166 166 {
167 + int randSeed = (int)DateTime.Now.Ticks;
168 + Console.WriteLine("Random seed: {0}", randSeed);
169 + var random = new Random(randSeed);
170 +
171 + int expectedTotalSize = 0;
172 +
173 + var list = ImmutableList<int>.Empty;
174 +
169 - // Add batches of 32, 128 times, giving 4096 items
170 - int batchSize = 32;
175 + // Add some small batches, verifying balance after each
171 176 for (int i = 0; i < 128; i++)
172 177 {
```



# code review

## some facts

- About 70% of developers spend 2 to 8 hours a week reviewing code.  
[Stack Overflow Dev Survey 2019]
- In 2021, 170M pull requests have been merged in GitHub.  
[The 2021 State of the Octoverse]
- Most code changes at Microsoft and Google are reviewed.



# code review tools

The screenshot shows the GitHub VS Code interface. The top navigation bar includes 'Conversation 14', 'Commits 2', 'Checks 22', and 'Files changed 5'. A search bar for 'Filter changed files' is present. The main area displays two files with their changes:

- extensions/ql-vscode/src/authentication.ts** (11 lines changed):
  - Line 7: `const GITHUB_AUTH_PROVIDER_ID = 'github';`
  - Line 8: `// https://docs.github.com/apps/building-oauth-apps/understanding-scopes-for-oauth-apps`
  - Line 9: `const SCOPES = ['repo'];`
  - Line 10: `- /**`
  - Line 11: `+ /**`
  - Line 12: `* Handles authentication to GitHub, using the VS Code [authentication API](https://code.visual`
  - Line 13: `*/`
  - Line 18: `export class Credentials {`
  - Line 19: `// eslint-disable-next-line @typescript-eslint/no-empty-function`
  - Line 20: `private constructor() {}`
  - Line 21: `+ /**`
  - Line 22: `+ * Initializes an instance of credentials with an octokit instance.`
  - Line 23: `+ *`
  - Line 24: `+ * Do not call this method until you know you actually need an instance of credentials.`
  - Line 25: `+ * since calling this method will require the user to log in.`
  - Line 26: `+ *`
  - Line 27: `+ * @param context The extension context.`
  - Line 28: `+ * @returns An instance of credentials.`
  - Line 29: `+ */`
  - Line 30: `static async initialize(context: vscode.ExtensionContext): Promise<Credentials> {`
  - Line 31: `const c = new Credentials();`
  - Line 32: `c.registerListeners(context);`
- extensions/ql-vscode/src/query-history.ts** (25 lines changed):
  - Line 36: `import { slurpQueryHistory, splatQueryHistory } from './query-serialization';`
  - Line 37: `import * as fs from 'fs-extra';`
  - Line 38: `import { CliVersionConstraint } from './cli';`
  - Line 39: `+ import { Credentials } from './authentication';`
  - Line 40: `+ import { cancelRemoteQuery } from './remote-queries/gh-actions-api-client';`
  - Line 41: `/**`
  - Line 42: `* query-history.ts`
  - Line 43: `*/`
  - Line 316: `private qs: QueryServerClient,`
  - Line 317: `private dbm: DatabaseManager,`
  - Line 318: `private queryStorageDir: string,`
  - Line 319: `ctx: ExtensionContext,`
  - Line 320: `private ctx: ExtensionContext,`
  - Line 321: `private queryHistoryConfigListener: QueryHistoryConfig,`
  - Line 322: `private doCompareCallback: (`

The screenshot shows the Crucible code review interface. The top navigation bar includes 'CR-FE-8851 (109)', 'Prefs', 'Filter', and '1h 55min'. The main area displays a code review for a file named `TrackedBranchesSearchCriteria.java`. The code is shown in a diff view, with changes highlighted in green. The review comments are visible below the code:

- Piotr Swiecicki**: providing null/empty reviewPermaid results in criteria matching all reviews, I'd rather expect precondition failure in such circumstances. Create issue · 27 Aug 14
- Cezary Zawadka**: Null is ok as we search for all tracked branches regardless review - auto update feature. However non null permaid id means we should return empty list if there is no review with the permaid - changed to be handled at ReviewPropertiesManager level. Create issue · 29 Aug 14
- Piotr Swiecicki**: fine for reviewPermaid property, but if client calls withReview method to build criteria, I'd assume he wanted to filter by particular perm id and was not expecting to pass null. Create issue · 29 Aug 14
- Cezary Zawadka**: My mistake - for withReview() precondition makes sense. Create issue · 29 Aug 14

The screenshot shows the ReviewBoard interface. The top navigation bar includes 'Fix regressions from merges an...', 'https://reviews.reviewboard.org/r/12144/diff/1/', and '1h 55min'. The main area displays a commit summary and a diff view for a file named `reviewboard/diffviewer/parser.py`. The commit summary is:

- First Last Summary**: `+ Fix regressions from merges and unit test updates.` by **Christian Hammond**

The diff view shows the following changes:

- Line 267: `#: #:`
- Line 268: `#: Version Added: 4.0.6`
- Line 269: `#: 4.0.6`
- Line 270: `#: #:`
- Line 271: `#: Type:`
- Line 272: `#: int`
- Line 273: `old_unix_mode = TypedProperty(six.text_type)`
- Line 274: `#: The new UNIX mode for the file.`
- Line 275: `#: #:`
- Line 276: `#: #:`
- Line 277: `#: Version Added: 4.0.6`
- Line 278: `#: 4.0.6`
- Line 279: `#: #:`
- Line 280: `#: Type:`
- Line 281: `#: int`
- Line 282: `new_unix_mode = TypedProperty(six.text_type)`
- Line 283: `#: The parsed original name of the file.`
- Line 284: `#: #:`
- Line 285: `#: #:`
- Line 286: `#: Deprecated:`
- Line 287: `#: 4.0:`

# code review

The screenshot displays a GitHub pull request interface. At the top, there are navigation tabs for 'Conversation' (14), 'Commits' (2), 'Checks' (22), and 'Files changed' (5). Below this, a search bar 'Filter changed files' is present. The left sidebar shows a file tree with folders 'extensions/ql-vscode/src', 'remote-queries', and 'vscode-tests/no-workspace/r...'. The main area shows two code diff views. The first view is for 'extensions/ql-vscode/src/authentication.ts' (11 lines changed), showing a diff between lines 7-9 and 10-20. The second view is for 'extensions/ql-vscode/src/query-history.ts' (25 lines changed), showing a diff between lines 36-38 and 39-40. The code snippets include TypeScript constants, class definitions, and imports.

Handle cancelling of remote que X

https://github.com/github/vscode-codeq Work 110%

Conversation 14 Commits 2 Checks 22 Files changed 5

Changes from all commits File filter Conversations 0 / 5 files

Filter changed files

extensions/ql-vscode/src

- authentication.ts
- query-history.ts

remote-queries

- gh-actions-api-client.ts
- remote-queries-manage...

vscode-tests/no-workspace/r...

- gh-actions-api-client.tes...

Beta Give feedback

11 extensions/ql-vscode/src/authentication.ts

```
@@ -7,7 +7,7 @@ const GITHUB_AUTH_PROVIDER_ID = 'github';
7 7 // https://docs.github.com/apps/building-oauth-apps/understand
8 8 const SCOPES = ['repo'];
9 9
10 - /**
10 + /**
11 11 * Handles authentication to GitHub, using the VS Code [authen
12 12 */
13 13 export class Credentials {
@@ -18,6 +18,15 @@ export class Credentials {
18 18 // eslint-disable-next-line @typescript-eslint/no-empty-func
19 19 private constructor() { }
20 20
21 + /**
22 + * Initializes an instance of credentials with an octokit in
23 + *
24 + * Do not call this method until you know you actually need
25 + * since calling this method will require the user to log in
26 + *
27 + * @param context The extension context.
28 + * @returns An instance of credentials.
29 + */
21 30 static async initialize(context: vscode.ExtensionContext): P
22 31 const c = new Credentials();
23 32 c.registerListeners(context);
```

25 extensions/ql-vscode/src/query-history.ts

```
@@ -36,6 +36,8 @@ import { QueryStatus } from './query-status'
36 36 import { slurpQueryHistory, splatQueryHistory } from './query-
37 37 import * as fs from 'fs-extra';
38 38 import { CliVersionConstraint } from './cli';
39 + import { Credentials } from './authentication';
40 + import { cancelRemoteQuery } from './remote-queries/gh-action
```

# code review

The screenshot displays a GitHub pull request interface. At the top, there are navigation tabs for 'Conversation' (14), 'Commits' (2), 'Checks' (22), and 'Files changed' (5). Below this, a search bar labeled 'Filter changed files' is present. The left sidebar shows a file tree with folders 'extensions/ql-vscode/src', 'remote-queries', and 'vscode-tests/no-workspace/r...'. The main area shows two code diff views. The first view is for 'extensions/ql-vscode/src/authentication.ts' (11 lines changed), showing a diff between lines 7-9 and 10-20. The second view is for 'extensions/ql-vscode/src/query-history.ts' (25 lines changed), showing a diff between lines 36-38 and 39-40. The code snippets include TypeScript imports, constants, class definitions, and static methods.

Handle cancelling of remote que X

https://github.com/github/vscode-codeq Work 110%

Conversation 14 Commits 2 Checks 22 Files changed 5

Changes from all commits File filter Conversations 0 / 5 files

Filter changed files

extensions/ql-vscode/src

- authentication.ts
- query-history.ts

remote-queries

- gh-actions-api-client.ts
- remote-queries-manage...

vscode-tests/no-workspace/r...

- gh-actions-api-client.tes...

Beta Give feedback

11 extensions/ql-vscode/src/authentication.ts

```
@@ -7,7 +7,7 @@ const GITHUB_AUTH_PROVIDER_ID = 'github';
7 7 // https://docs.github.com/apps/building-oauth-apps/understand
8 8 const SCOPES = ['repo'];
9 9
10 - /**
10 + /**
11 11 * Handles authentication to GitHub, using the VS Code [authen
12 12 */
13 13 export class Credentials {
@@ -18,6 +18,15 @@ export class Credentials {
18 18 // eslint-disable-next-line @typescript-eslint/no-empty-func
19 19 private constructor() { }
20 20
21 + /**
22 + * Initializes an instance of credentials with an octokit in
23 + *
24 + * Do not call this method until you know you actually need
25 + * since calling this method will require the user to log in
26 + *
27 + * @param context The extension context.
28 + * @returns An instance of credentials.
29 + */
21 30 static async initialize(context: vscode.ExtensionContext): P
22 31 const c = new Credentials();
23 32 c.registerListeners(context);
```

25 extensions/ql-vscode/src/query-history.ts

```
@@ -36,6 +36,8 @@ import { QueryStatus } from './query-status'
36 36 import { slurpQueryHistory, splatQueryHistory } from './query-
37 37 import * as fs from 'fs-extra';
38 38 import { CliVersionConstraint } from './cli';
39 + import { Credentials } from './authentication';
40 + import { cancelRemoteQuery } from './remote-queries/gh-action
```

# code review

## reviewers' behavior

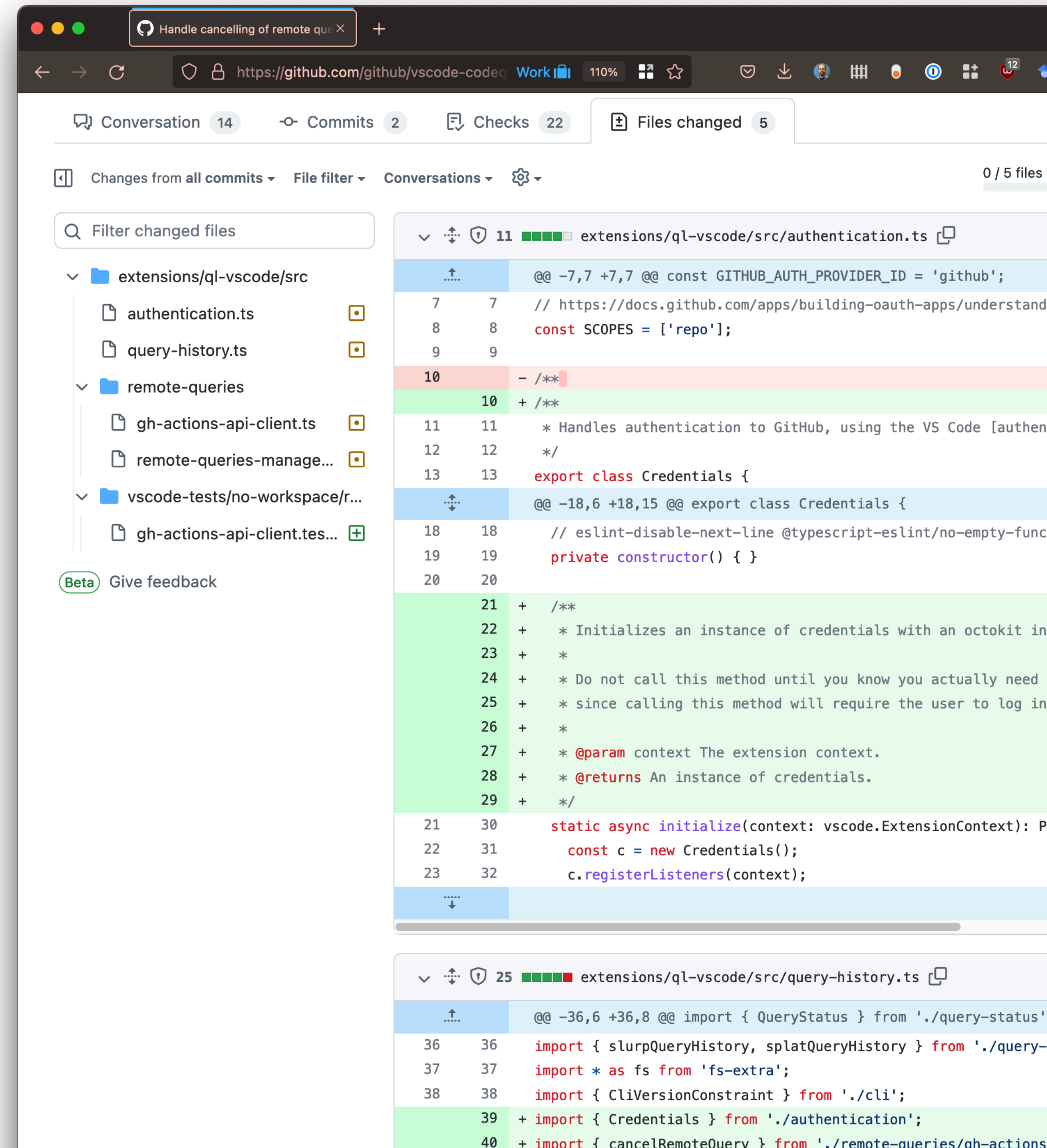
The screenshot displays a GitHub pull request interface. At the top, there are navigation tabs for 'Conversation' (14), 'Commits' (2), 'Checks' (22), and 'Files changed' (5). Below this, a search bar for 'Filter changed files' is present. The left sidebar shows a file tree with folders 'extensions/ql-vscode/src', 'remote-queries', and 'vscode-tests/no-workspace/r...'. The main area shows two code diff views. The first view is for 'extensions/ql-vscode/src/authentication.ts' (11 lines changed). It shows a diff between two versions of the file, with line numbers 7 through 23. The second view is for 'extensions/ql-vscode/src/query-history.ts' (25 lines changed), showing a diff between two versions with line numbers 36 through 40. The code snippets include TypeScript syntax such as 'const GITHUB\_AUTH\_PROVIDER\_ID = 'github'' and 'export class Credentials {'. The interface also includes a 'Beta Give feedback' button at the bottom left of the sidebar.

# code review

## reviewers' behavior

- In more than 50% of the reviews, the reviewer started with the file presented first.
- In almost 40% of the navigations, the reviewer went to the next file in order.

[Baum, Schneider, Bacchelli - ICSME 2017]





# code review

## test files

The screenshot shows a GitHub pull request interface for the repository `github/vscode-codequery`. The browser address bar shows `https://github.com/github/vscode-codequery`. The interface includes navigation tabs for Conversation (14), Commits (2), Checks (22), and Files changed (5). A search bar for "Filter changed files" is present. The file explorer on the left shows a tree view with folders `extensions/ql-vscode/src` and `vscode-tests/no-workspace/r...`. The `authentication.ts` file in the `src` folder is highlighted with a blue box, and the `gh-actions-api-client.test.ts` file in the `vscode-tests` folder is highlighted with a pink box. The main content area displays the diff for `extensions/ql-vscode/src/authentication.ts`, showing changes between lines 7 and 23. The diff includes a new constant `SCOPES`, a new JSDoc comment, and a new `initialize` method. Below this, the diff for `extensions/ql-vscode/src/query-history.ts` is partially visible, showing new imports for `Credentials` and `cancelRemoteQuery`.

Handle cancelling of remote que X

Conversation 14 Commits 2 Checks 22 Files changed 5

Changes from all commits File filter Conversations 0 / 5 files

Filter changed files

extensions/ql-vscode/src

- authentication.ts
- query-history.ts
- remote-queries
  - gh-actions-api-client.ts
  - remote-queries-manage...
- vscode-tests/no-workspace/r...
  - gh-actions-api-client.tes...

Beta Give feedback

11 extensions/ql-vscode/src/authentication.ts

```
@@ -7,7 +7,7 @@ const GITHUB_AUTH_PROVIDER_ID = 'github';
7 7 // https://docs.github.com/apps/building-oauth-apps/understand
8 8 const SCOPES = ['repo'];
9 9
10 - /**
10 + /**
11 11 * Handles authentication to GitHub, using the VS Code [authen
12 12 */
13 13 export class Credentials {
@@ -18,6 +18,15 @@ export class Credentials {
18 18 // eslint-disable-next-line @typescript-eslint/no-empty-func
19 19 private constructor() { }
20 20
21 + /**
22 + * Initializes an instance of credentials with an octokit in
23 + *
24 + * Do not call this method until you know you actually need
25 + * since calling this method will require the user to log in
26 + *
27 + * @param context The extension context.
28 + * @returns An instance of credentials.
29 + */
21 30 static async initialize(context: vscode.ExtensionContext): P
22 31 const c = new Credentials();
23 32 c.registerListeners(context);
```

25 extensions/ql-vscode/src/query-history.ts

```
@@ -36,6 +36,8 @@ import { QueryStatus } from './query-status'
36 36 import { slurpQueryHistory, splatQueryHistory } from './query-
37 37 import * as fs from 'fs-extra';
38 38 import { CliVersionConstraint } from './cli';
39 + import { Credentials } from './authentication';
40 + import { cancelRemoteQuery } from './remote-queries/gh-action
```

# code review

## test files

- Developers see test code as less important than production code.

The screenshot displays a GitHub pull request interface. The top navigation bar shows 'Conversation 14', 'Commits 2', 'Checks 22', and 'Files changed 5'. The main content area is divided into two panels. The left panel shows a file explorer with a search bar 'Filter changed files'. It lists two folders: 'extensions/ql-vscode/src' and 'vscode-tests/no-workspace/r...'. The 'src' folder contains 'authentication.ts', 'query-history.ts', and 'remote-queries'. The 'vscode-tests' folder contains 'gh-actions-api-client.ts'. The right panel shows the diff view for two files. The top file is 'extensions/ql-vscode/src/authentication.ts' with 11 changes. The bottom file is 'extensions/ql-vscode/src/query-history.ts' with 25 changes. The diff view shows line numbers and code changes, including imports, constants, and class definitions.

```
@@ -7,7 +7,7 @@ const GITHUB_AUTH_PROVIDER_ID = 'github';
7 7 // https://docs.github.com/apps/building-oauth-apps/understand
8 8 const SCOPES = ['repo'];
9 9
10 - /**
10 + /**
11 11 * Handles authentication to GitHub, using the VS Code [authen
12 12 */
13 13 export class Credentials {
@@ -18,6 +18,15 @@ export class Credentials {
18 18 // eslint-disable-next-line @typescript-eslint/no-empty-func
19 19 private constructor() { }
20 20
21 + /**
22 + * Initializes an instance of credentials with an octokit in
23 + *
24 + * Do not call this method until you know you actually need
25 + * since calling this method will require the user to log in
26 + *
27 + * @param context The extension context.
28 + * @returns An instance of credentials.
29 + */
21 30 static async initialize(context: vscode.ExtensionContext): P
22 31 const c = new Credentials();
23 32 c.registerListeners(context);
@@ -36,6 +36,8 @@ import { QueryStatus } from './query-status'
36 36 import { slurpQueryHistory, splatQueryHistory } from './query-
37 37 import * as fs from 'fs-extra';
38 38 import { CliVersionConstraint } from './cli';
39 + import { Credentials } from './authentication';
40 + import { cancelRemoteQuery } from './remote-queries/gh-action
```

# code review

## test files

- Developers see test code as less important than production code.
- Test files are almost twice less likely to be discussed during code review when together with production files.

The screenshot displays a GitHub pull request interface. The top navigation bar shows 'Conversation 14', 'Commits 2', 'Checks 22', and 'Files changed 5'. The main content area is divided into a file explorer on the left and a diff view on the right.

**File Explorer:**

- extensions/ql-vscode/src
  - authentication.ts
  - query-history.ts
  - remote-queries
    - gh-actions-api-client.ts
    - remote-queries-manage...
- vscode-tests/no-workspace/r...
  - gh-actions-api-client.tes...

**Diff View:**

The diff view shows changes in two files:

- authentication.ts:** Lines 7-9 show a constant definition for `GITHUB_AUTH_PROVIDER_ID` and `SCOPES`. Line 10 shows a removal of a comment block. Line 11 shows a new comment block. Line 12 shows a closing comment. Line 13 shows the start of a `Credentials` class. Lines 18-20 show the `private constructor()` method. Lines 21-29 show a new comment block for the `initialize` method. Lines 30-32 show the `static async initialize` method implementation.
- query-history.ts:** Lines 36-38 show imports for `slurpQueryHistory`, `splatQueryHistory`, and `CliVersionConstraint`. Lines 39-40 show new imports for `Credentials` and `cancelRemoteQuery`.

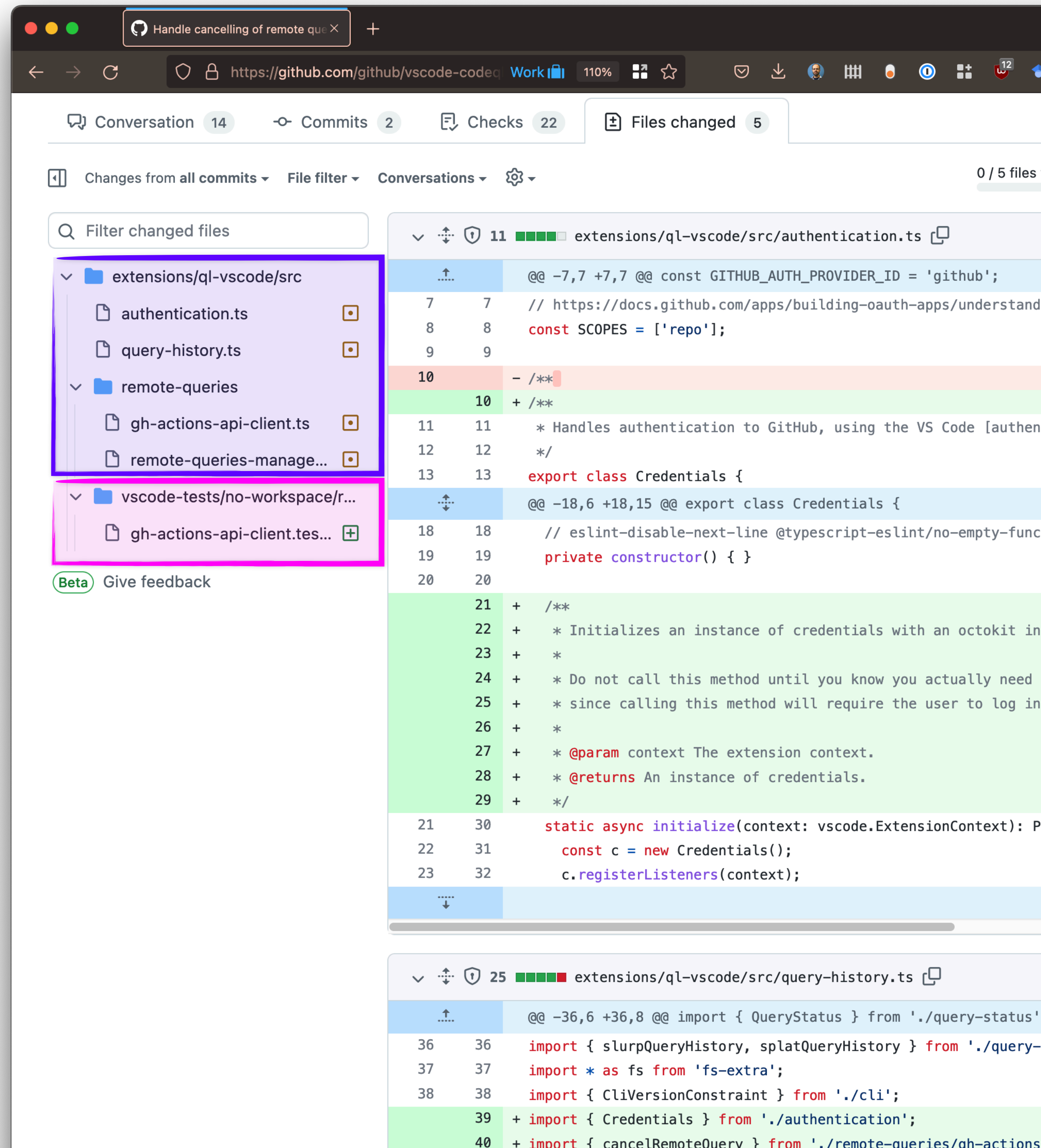
A 'Beta Give feedback' button is visible at the bottom left of the diff view.

# code review

## test files

- Developers see test code as less important than production code.
- Test files are almost twice less likely to be discussed during code review when together with production files.

[Spadini, Aniche, Storey, Bruntink, Bacchelli - ICSE 2018]

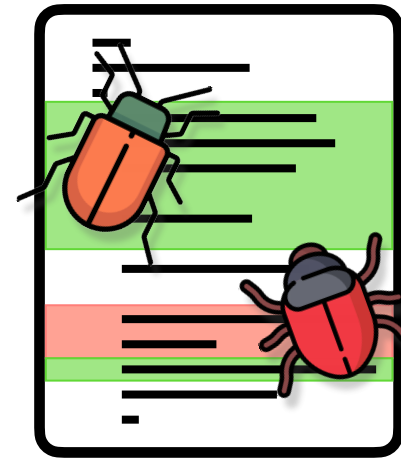


**code review**

**test order experiment**

# code review

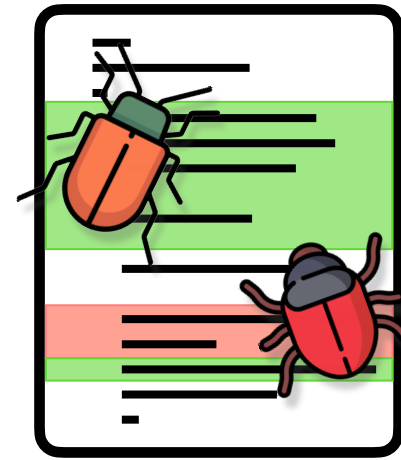
## test order experiment



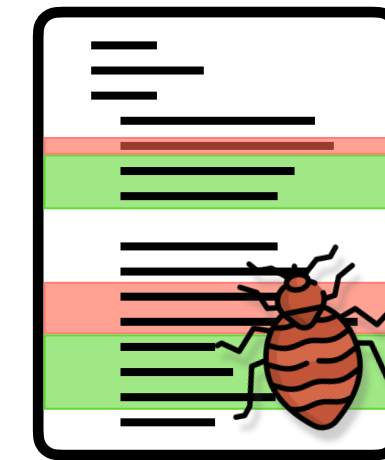
production.java

# code review

## test order experiment



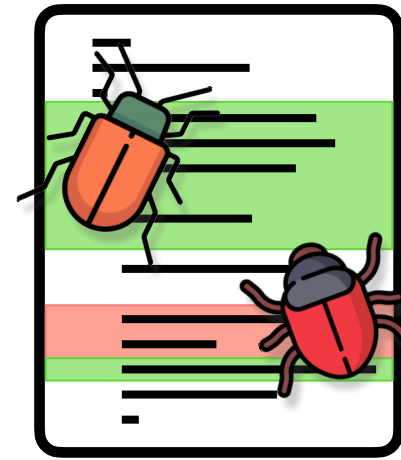
production.java



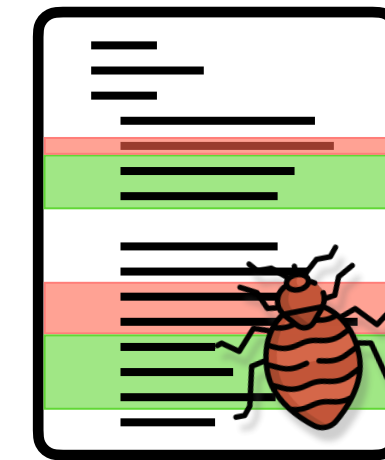
test.java

# code review

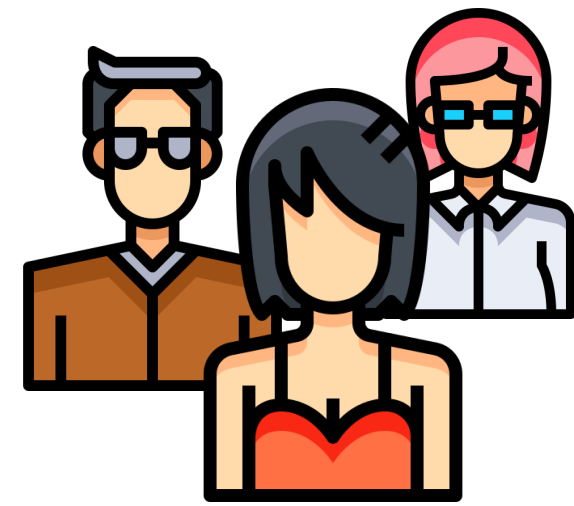
## test order experiment



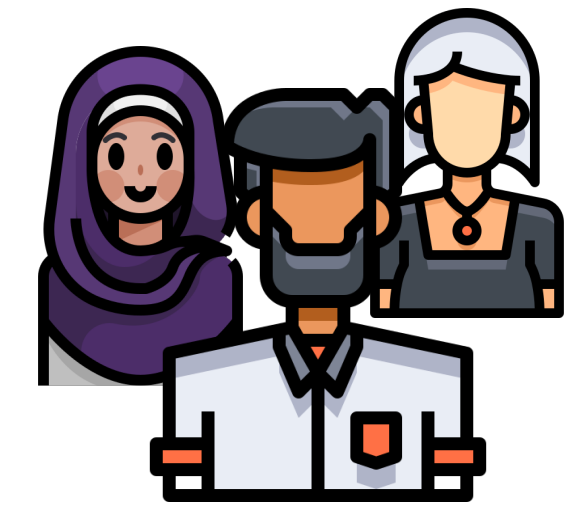
production.java



test.java



participants

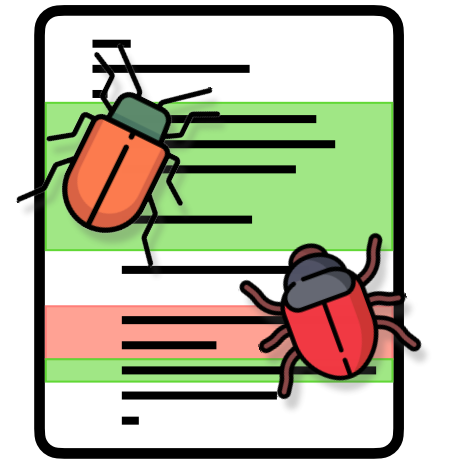


participants

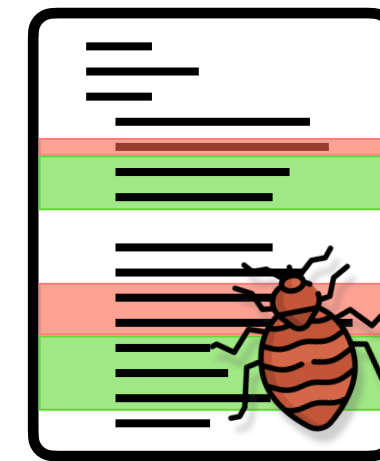


# code review

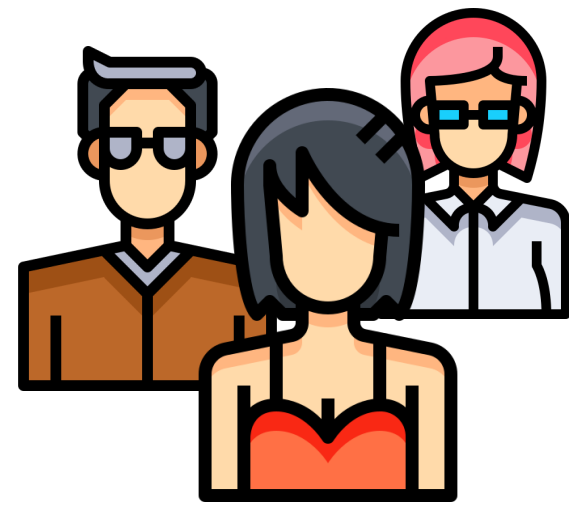
## test order experiment



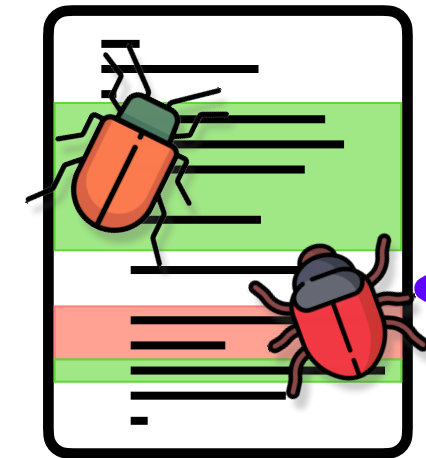
production.java



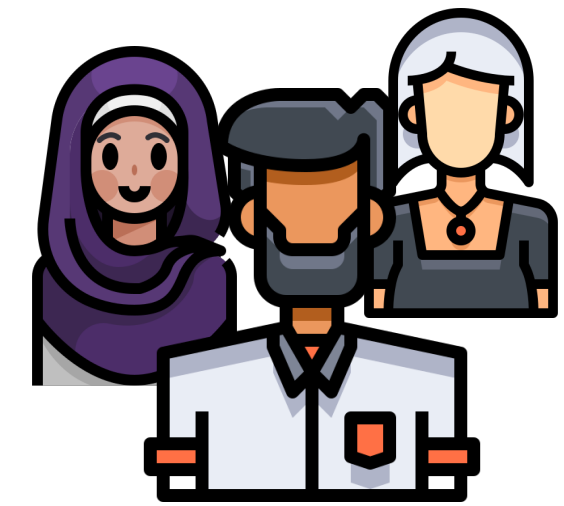
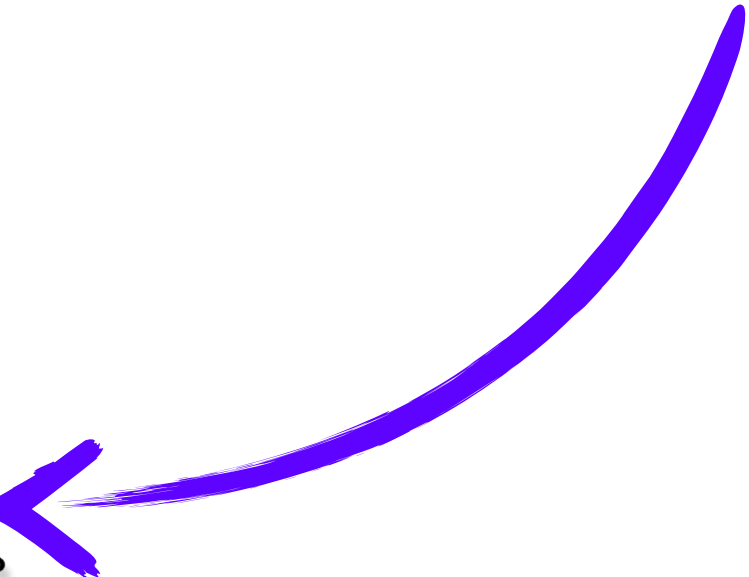
test.java



participants



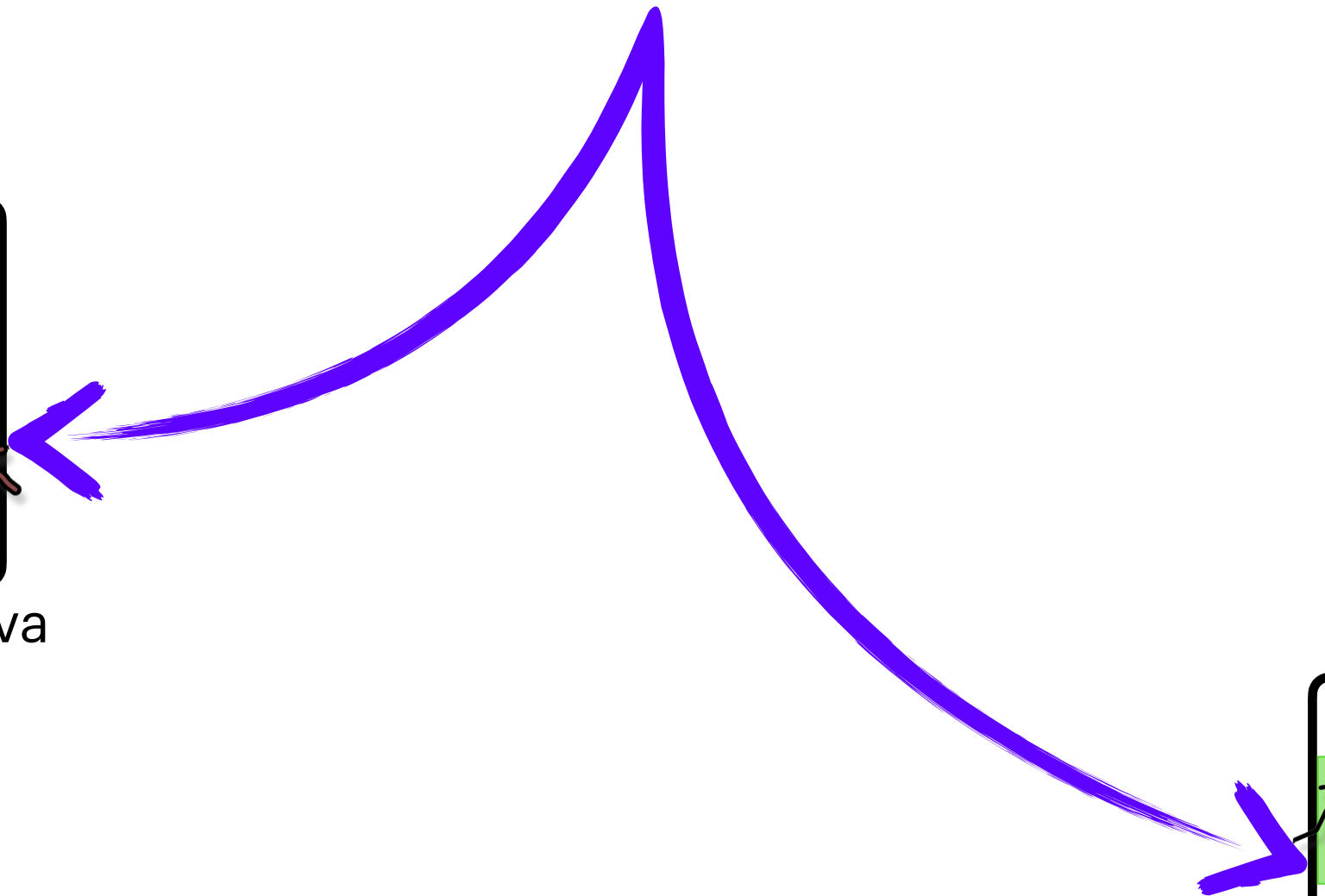
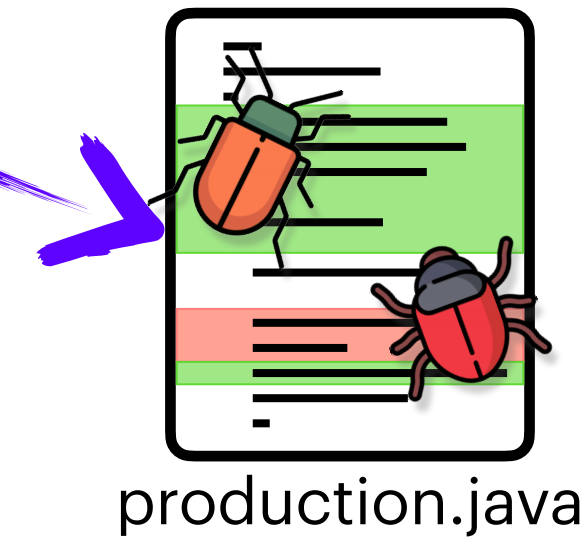
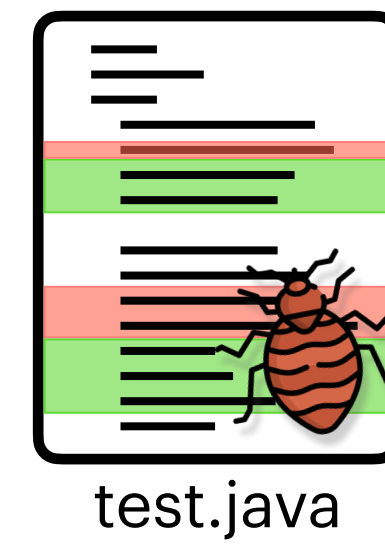
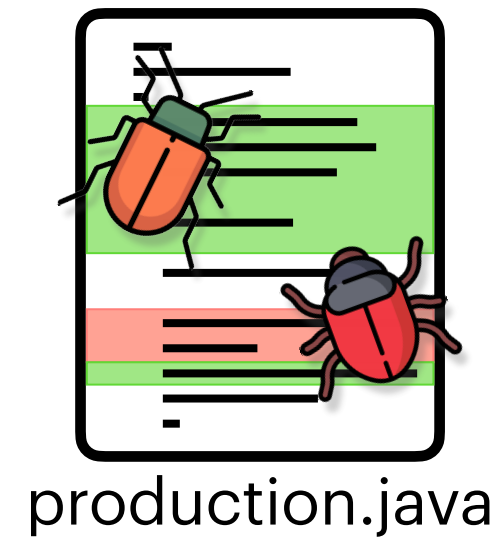
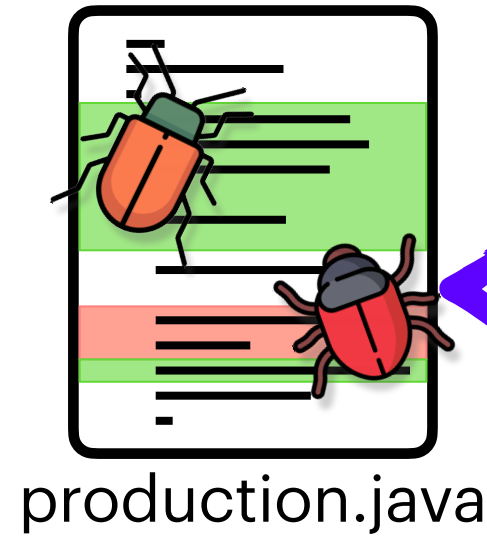
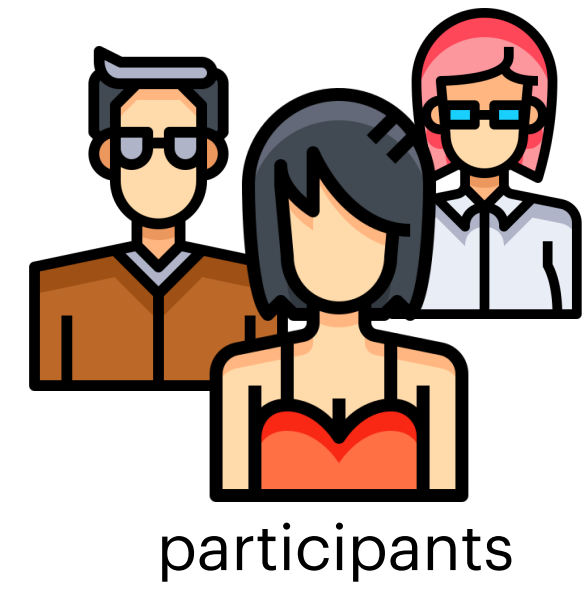
production.java



participants

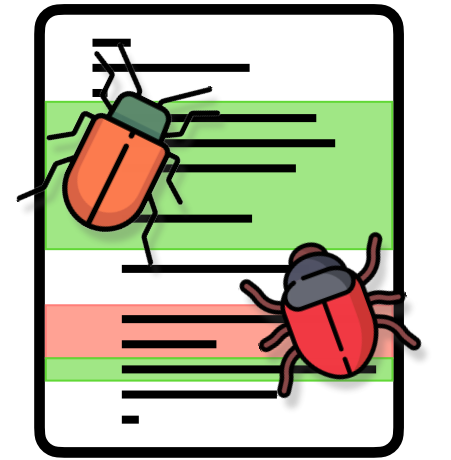
# code review

## test order experiment

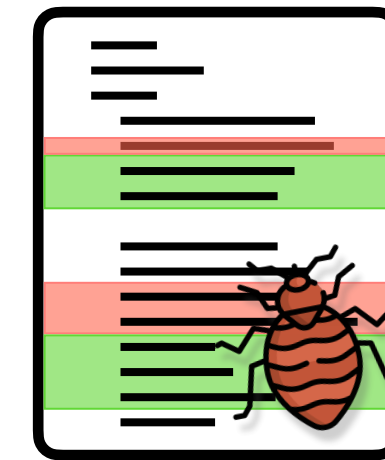


# code review

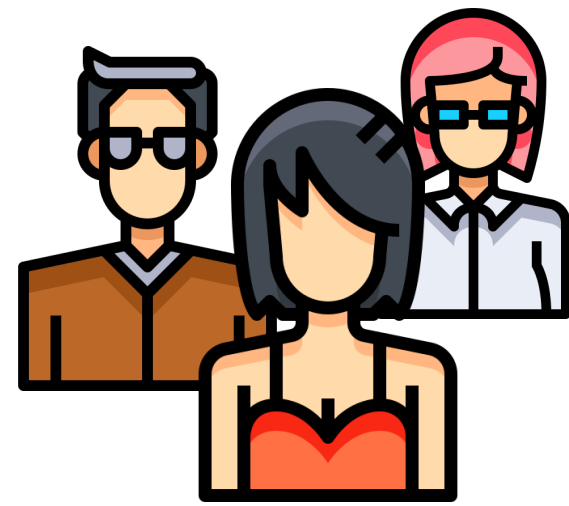
## test order experiment



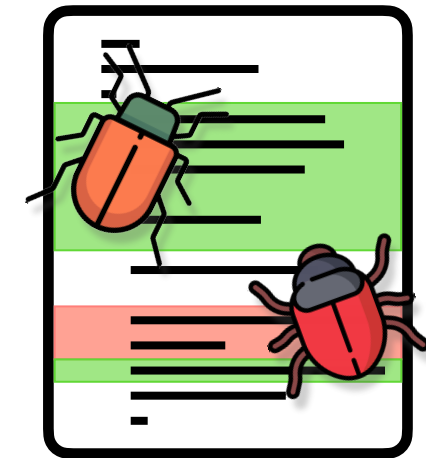
production.java



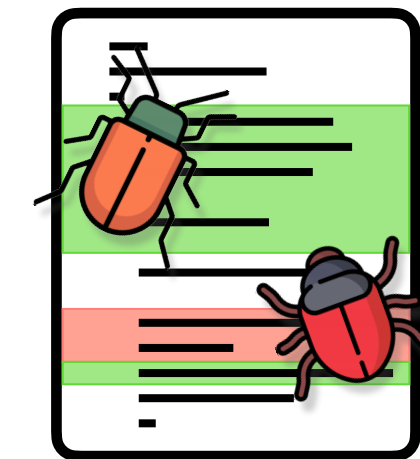
test.java



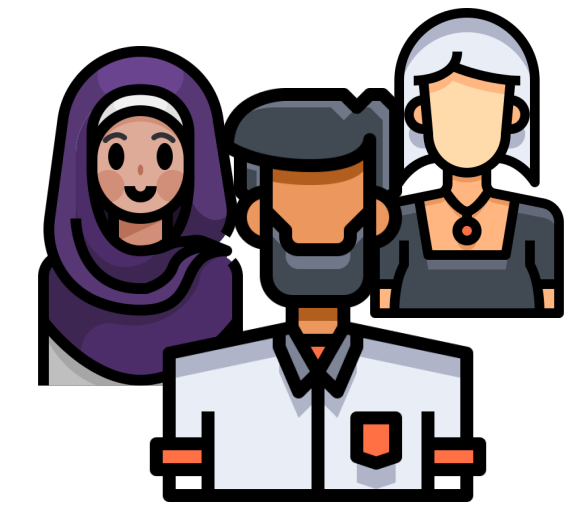
participants



production.java



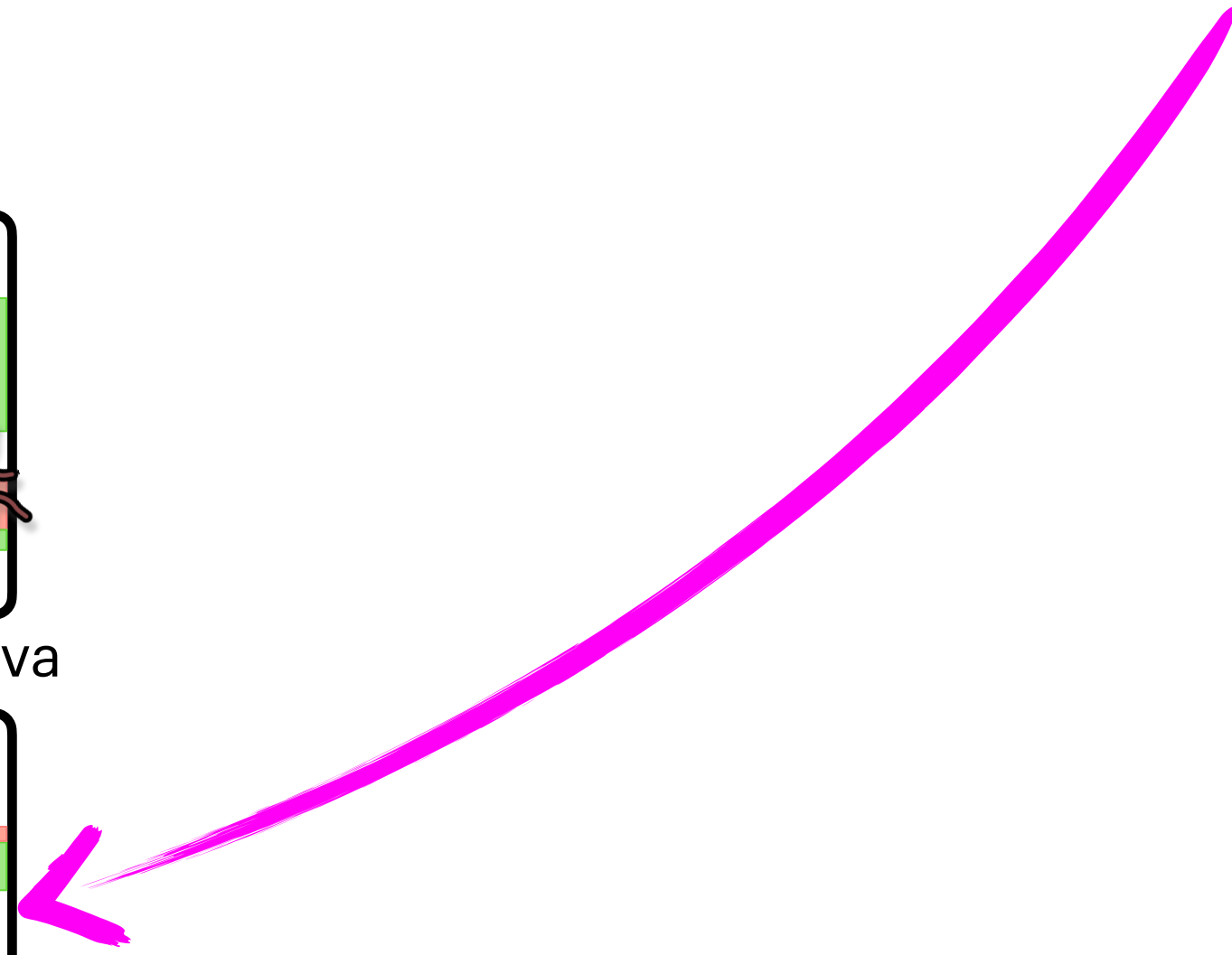
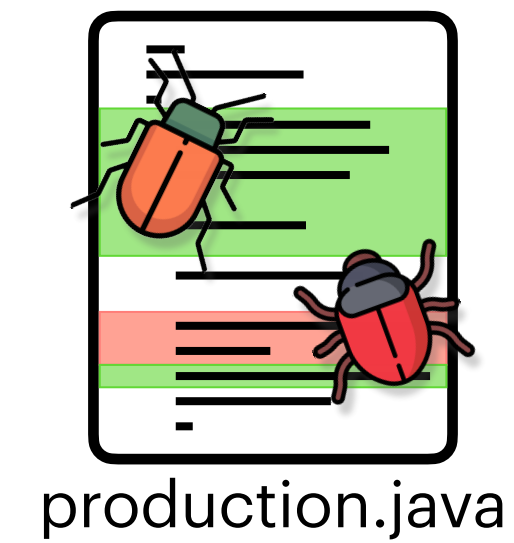
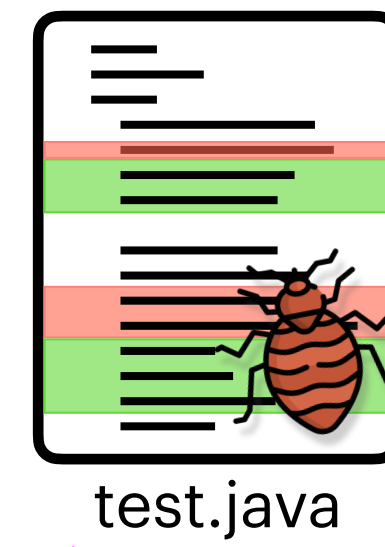
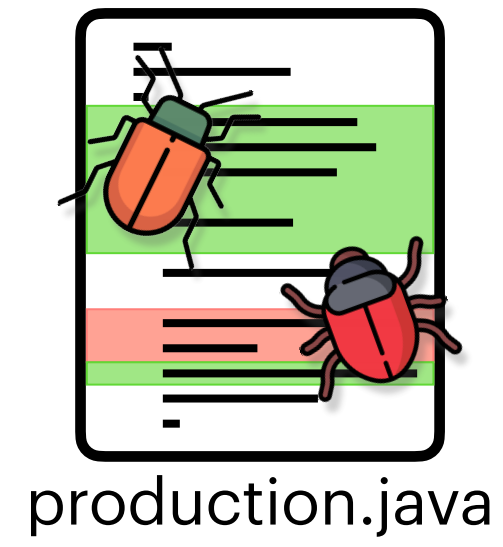
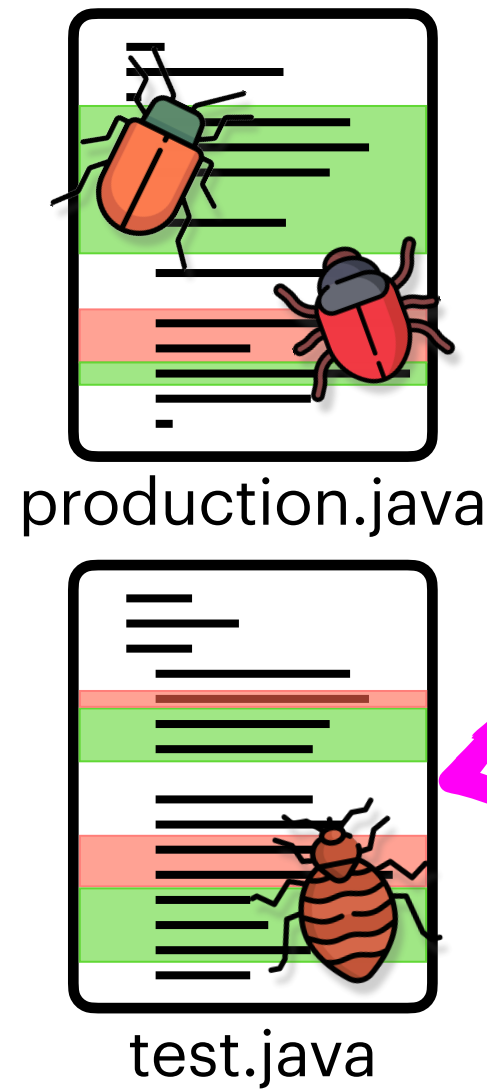
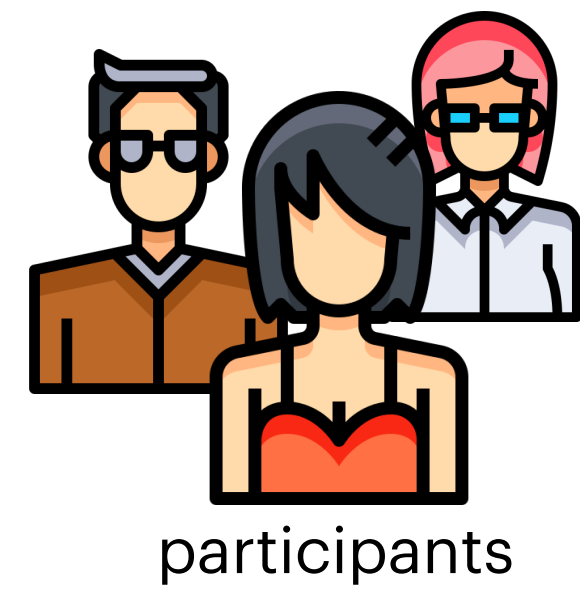
production.java



participants

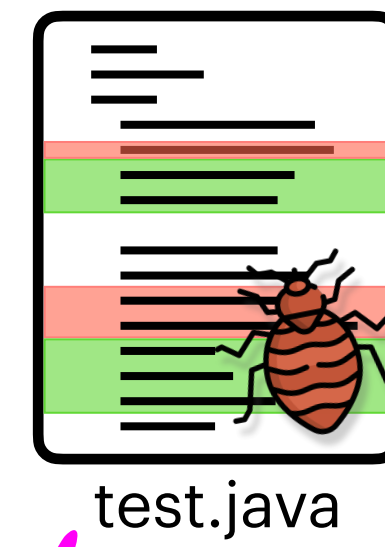
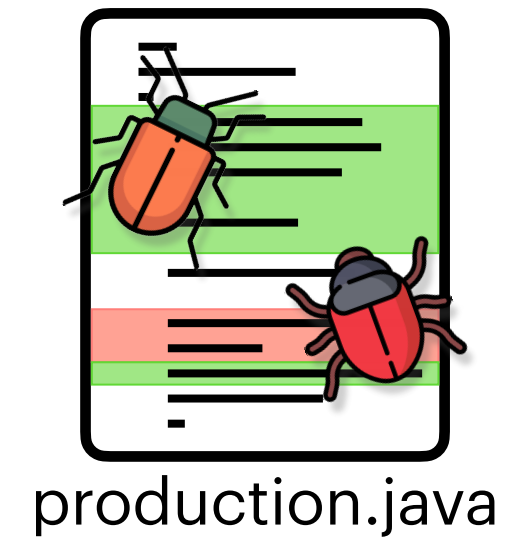
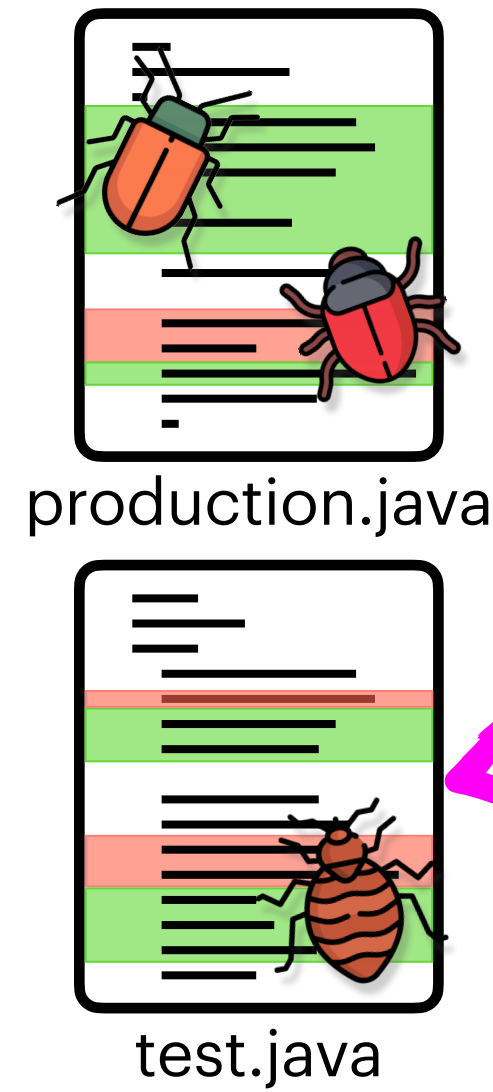
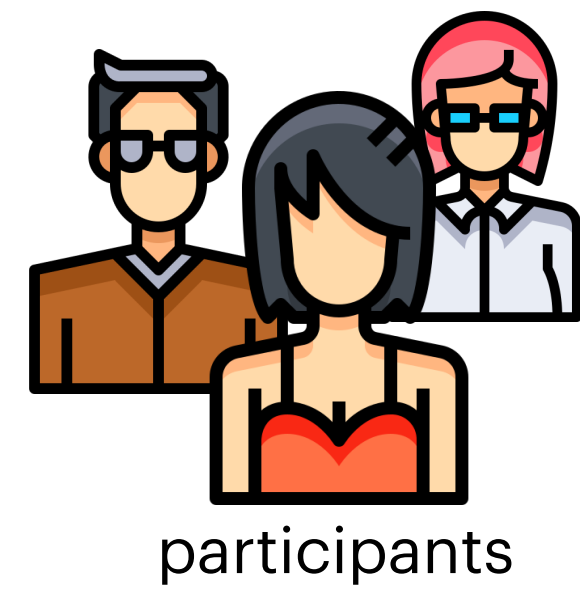
# code review

## test order experiment



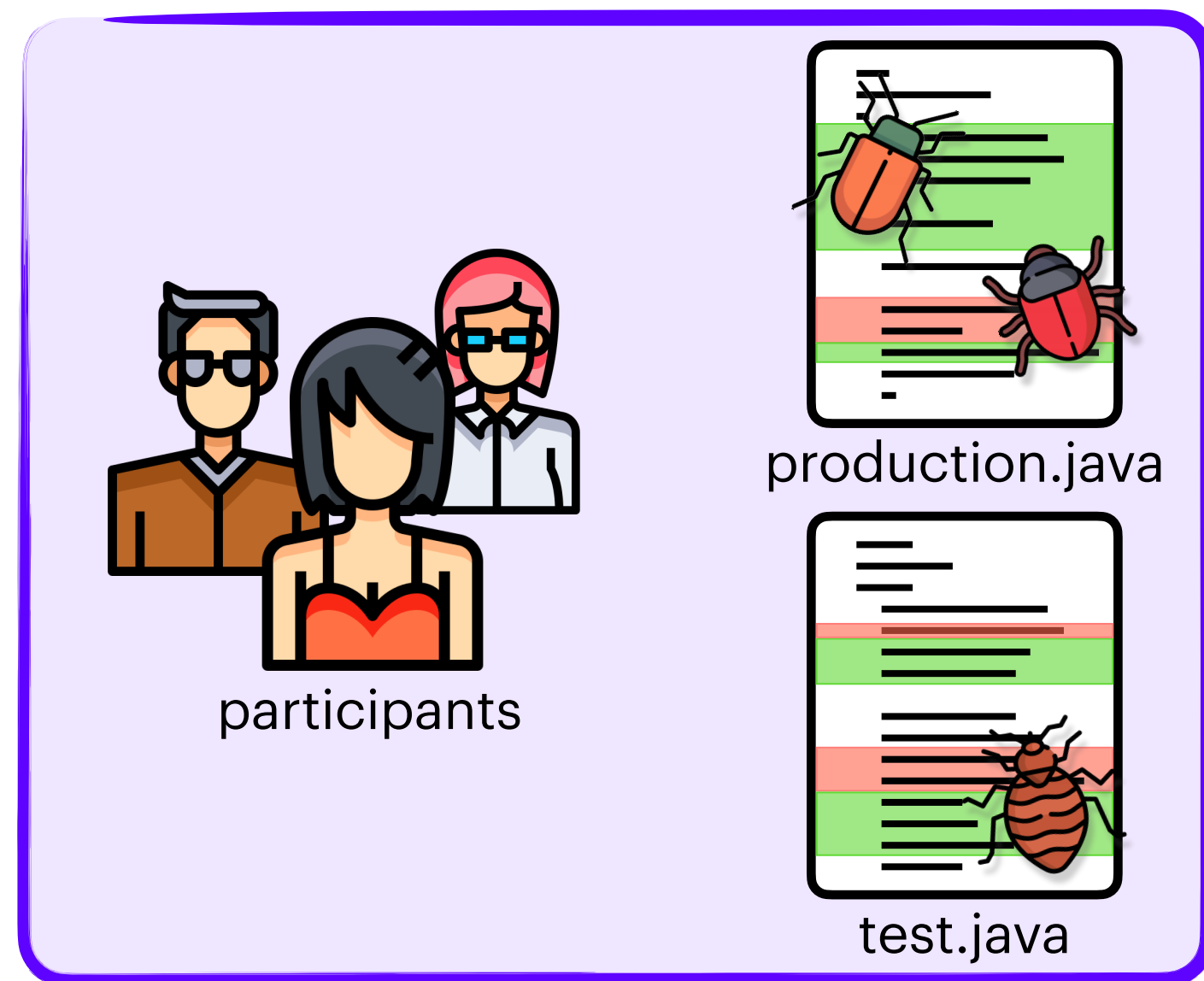
# code review

## test order experiment

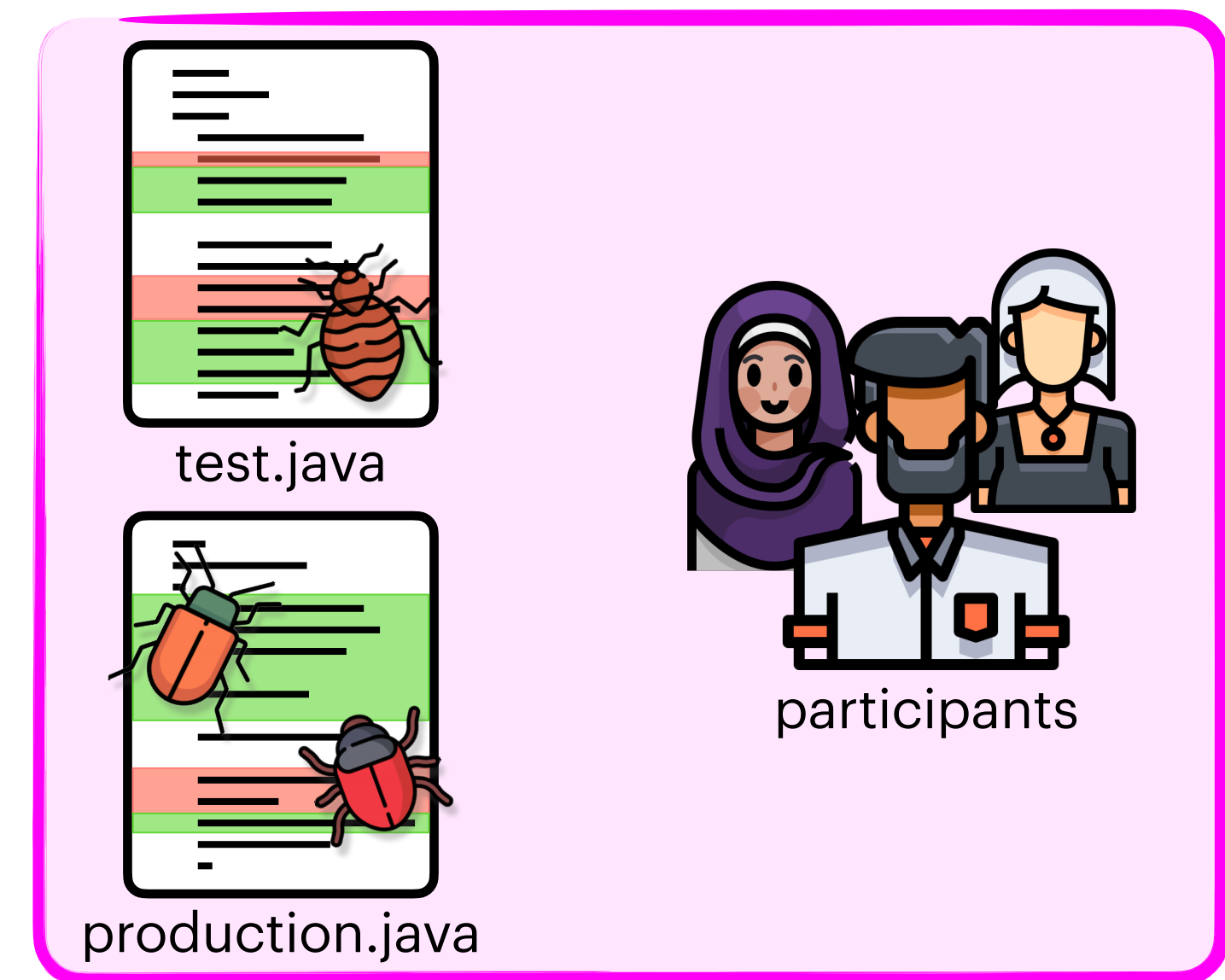


# code review

## test order experiment



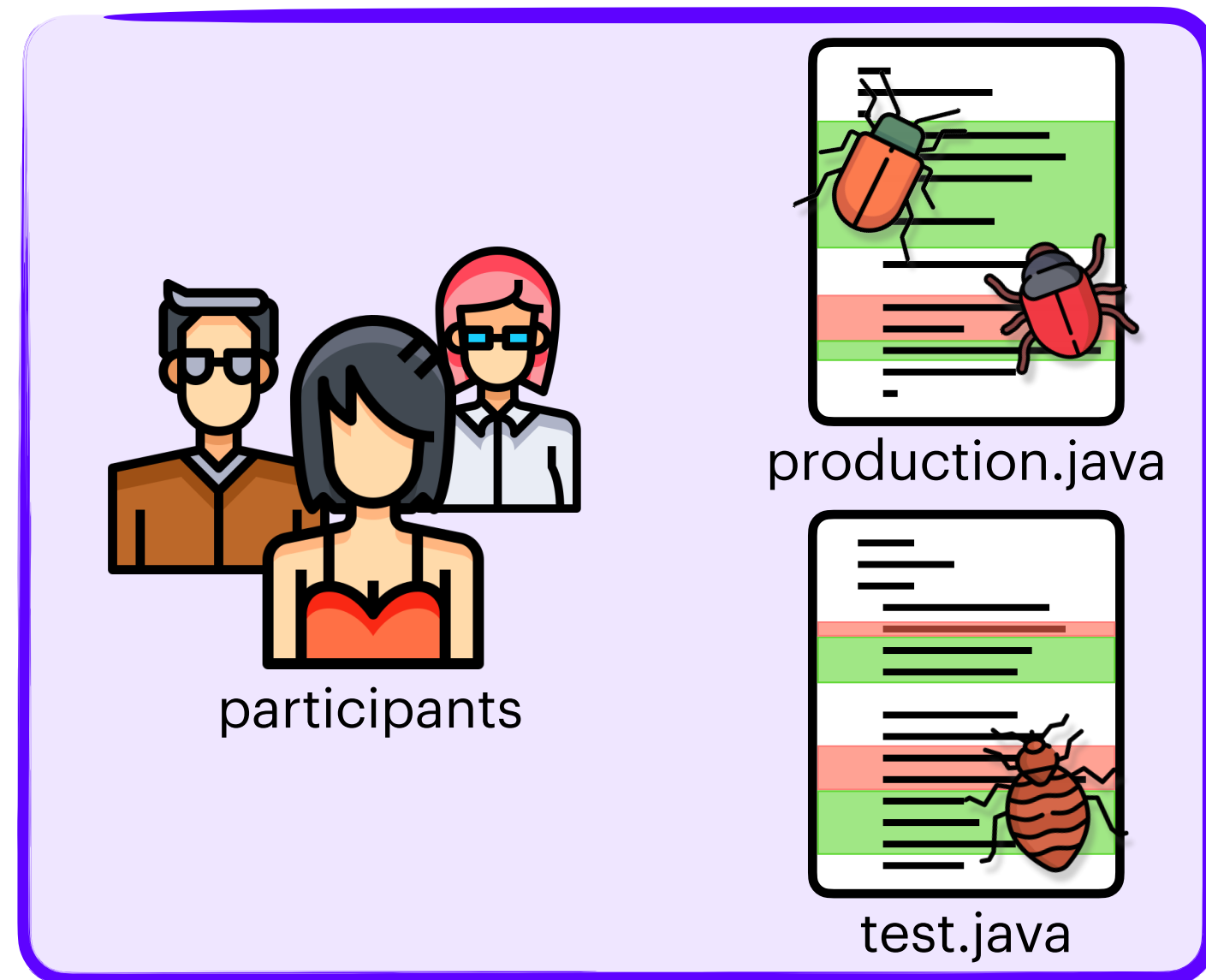
**production code first  
order**



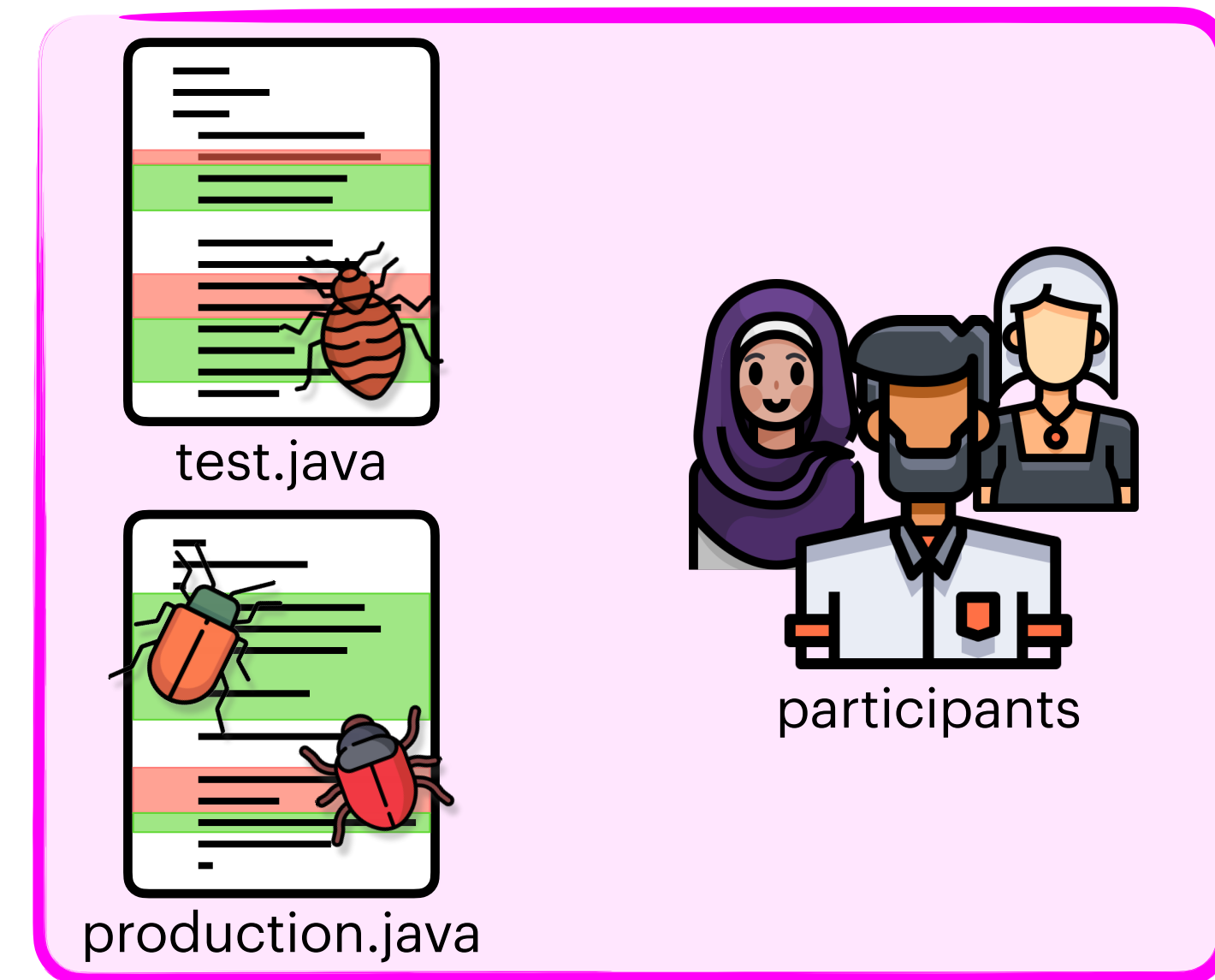
**test code first  
order**

# code review

## test order experiment

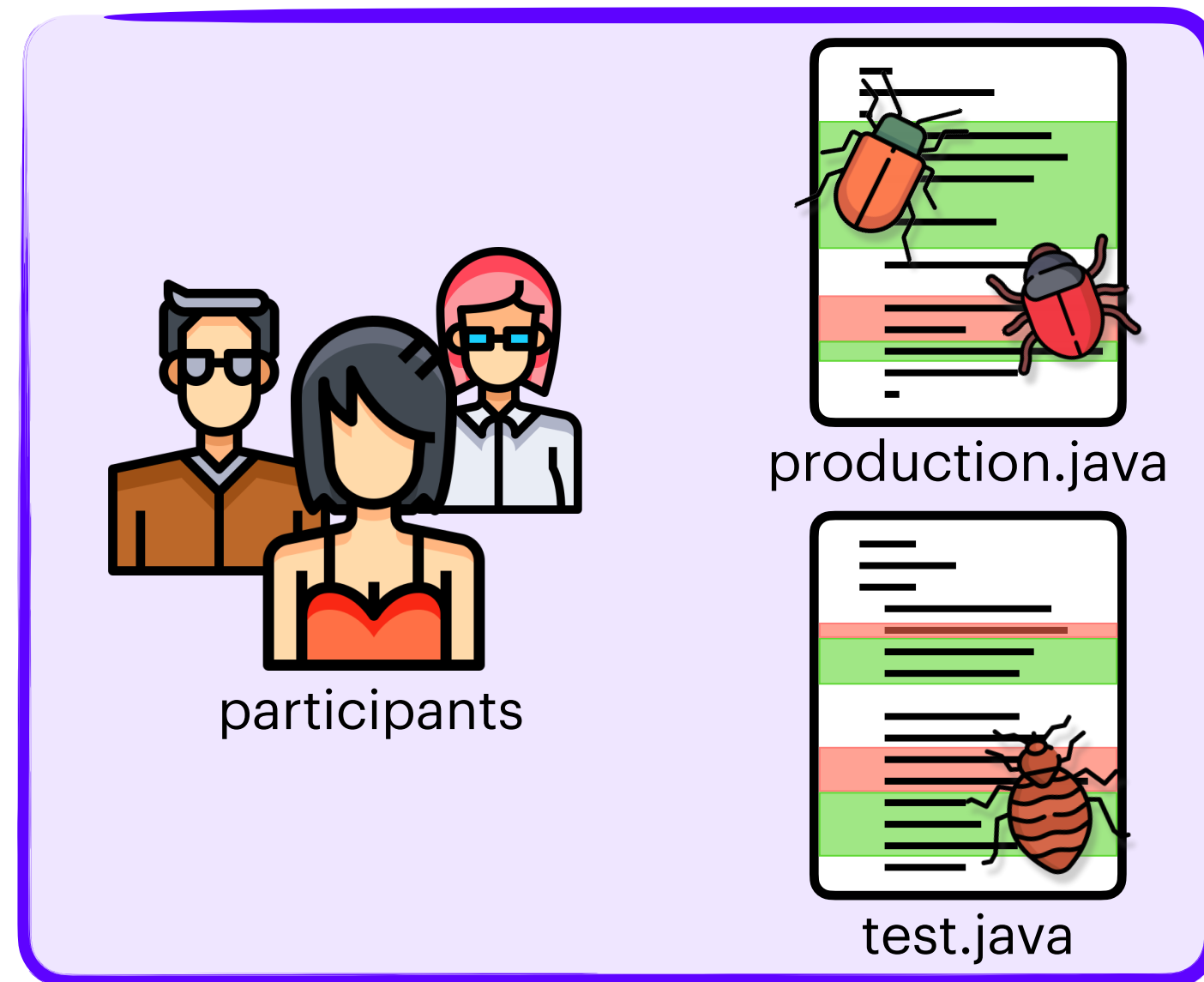


same likelihood  
of finding the  
production bugs



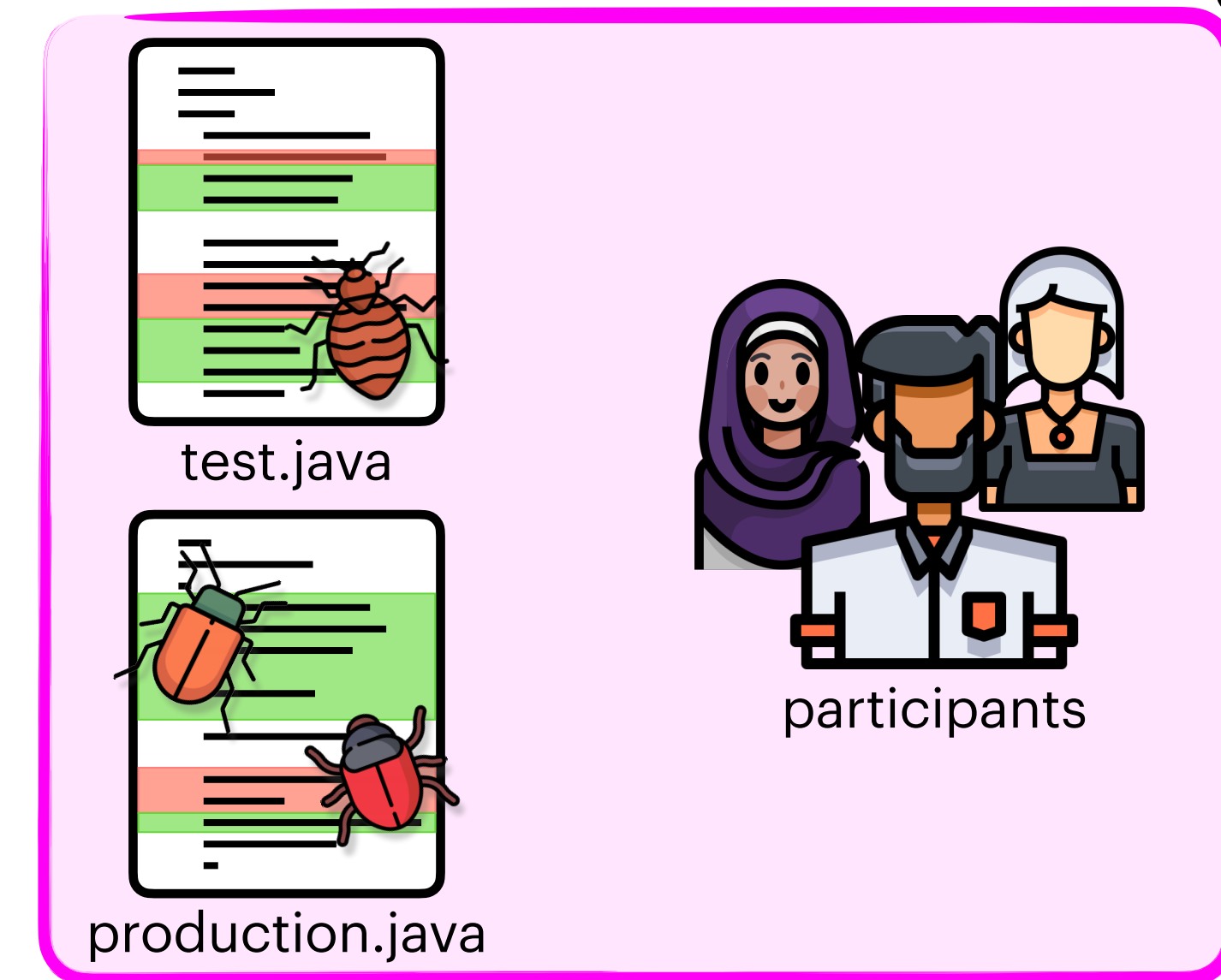
# code review

## test order experiment



same likelihood  
of finding the  
production bugs

**250% more likely**  
to find the test bug





# code review

## test files

The screenshot displays a GitHub pull request interface. The browser address bar shows the URL `https://github.com/github/vscode-codeq`. The repository name is `Work` and the page is zoomed in at 110%. The interface includes navigation tabs for Conversation (14), Commits (2), Checks (22), and Files changed (5). A search bar for "Filter changed files" is present. The file explorer on the left shows the directory structure: `extensions/ql-vscode/src` (containing `authentication.ts`, `query-history.ts`, and `remote-queries` subdirectory) and `vscode-tests/no-workspace/r...` (containing `gh-actions-api-client.tes...`). The main area shows a diff view for `extensions/ql-vscode/src/authentication.ts` (11 lines changed) and `extensions/ql-vscode/src/query-history.ts` (25 lines changed). The diff for `authentication.ts` shows changes to `const GITHUB_AUTH_PROVIDER_ID`, `const SCOPES`, and the `Credentials` class definition. The diff for `query-history.ts` shows changes to imports and the `initialize` method.

```
@@ -7,7 +7,7 @@ const GITHUB_AUTH_PROVIDER_ID = 'github';
7 7 // https://docs.github.com/apps/building-oauth-apps/understand
8 8 const SCOPES = ['repo'];
9 9
10 - /**
10 + /**
11 11 * Handles authentication to GitHub, using the VS Code [authen
12 12 */
13 13 export class Credentials {
@@ -18,6 +18,15 @@ export class Credentials {
18 18 // eslint-disable-next-line @typescript-eslint/no-empty-func
19 19 private constructor() { }
20 20
21 + /**
22 + * Initializes an instance of credentials with an octokit in
23 + *
24 + * Do not call this method until you know you actually need
25 + * since calling this method will require the user to log in
26 + *
27 + * @param context The extension context.
28 + * @returns An instance of credentials.
29 + */
21 30 static async initialize(context: vscode.ExtensionContext): P
22 31 const c = new Credentials();
23 32 c.registerListeners(context);
@@ -36,6 +36,8 @@ import { QueryStatus } from './query-status'
36 36 import { slurpQueryHistory, splatQueryHistory } from './query-
37 37 import * as fs from 'fs-extra';
38 38 import { CliVersionConstraint } from './cli';
39 + import { Credentials } from './authentication';
40 + import { cancelRemoteQuery } from './remote-queries/gh-action
```

# code review

## production files

The screenshot displays a GitHub pull request interface. At the top, there are navigation tabs for 'Conversation' (14), 'Commits' (2), 'Checks' (22), and 'Files changed' (5). Below these, there are filters for 'Changes from all commits', 'File filter', 'Conversations', and a settings icon. A search bar labeled 'Filter changed files' is present. The file explorer on the left shows a tree view with a purple highlight around the 'extensions/ql-vscode/src' directory, containing files like 'authentication.ts', 'query-history.ts', and 'remote-queries'. The main area shows a diff view for 'extensions/ql-vscode/src/authentication.ts' (11 lines changed) and 'extensions/ql-vscode/src/query-history.ts' (25 lines changed). The diff view shows line numbers and code changes, with some lines highlighted in green (added) and some in red (removed). The code is in TypeScript and includes comments and class definitions.

```
@@ -7,7 +7,7 @@ const GITHUB_AUTH_PROVIDER_ID = 'github';
7 7 // https://docs.github.com/apps/building-oauth-apps/understand
8 8 const SCOPES = ['repo'];
9 9
10 - /**
10 + /**
11 11 * Handles authentication to GitHub, using the VS Code [authen
12 12 */
13 13 export class Credentials {
@@ -18,6 +18,15 @@ export class Credentials {
18 18 // eslint-disable-next-line @typescript-eslint/no-empty-func
19 19 private constructor() { }
20 20
21 + /**
22 + * Initializes an instance of credentials with an octokit in
23 + *
24 + * Do not call this method until you know you actually need
25 + * since calling this method will require the user to log in
26 + *
27 + * @param context The extension context.
28 + * @returns An instance of credentials.
29 + */
21 30 static async initialize(context: vscode.ExtensionContext): P
22 31 const c = new Credentials();
23 32 c.registerListeners(context);
@@ -36,6 +36,8 @@ import { QueryStatus } from './query-status'
36 36 import { slurpQueryHistory, splatQueryHistory } from './query-
37 37 import * as fs from 'fs-extra';
38 38 import { CliVersionConstraint } from './cli';
39 + import { Credentials } from './authentication';
40 + import { cancelRemoteQuery } from './remote-queries/gh-action
```

**code review**  
**production files**

# code review

## production files

- We analyzed review comments for 200k pull requests from some 140 popular projects in GitHub...

# code review

## production files

- We analyzed review comments for 200k pull requests from some 140 popular projects in GitHub...
- ... and found something remarkable.

# code review

## production files

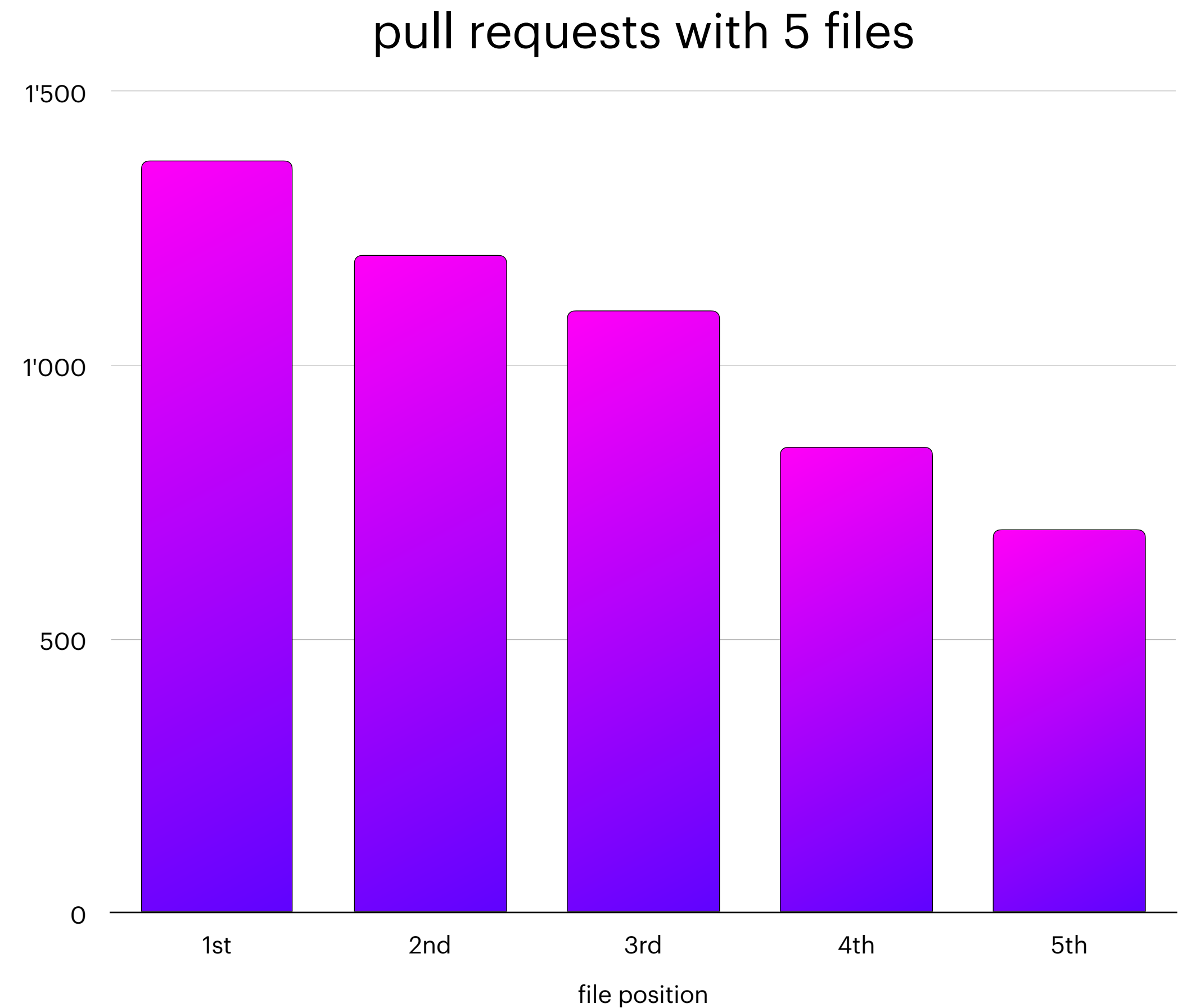
- We analyzed review comments for 200k pull requests from some 140 popular projects in GitHub...
- ... and found something remarkable.

cumulative number of  
review comments  
by file position

# code review production files

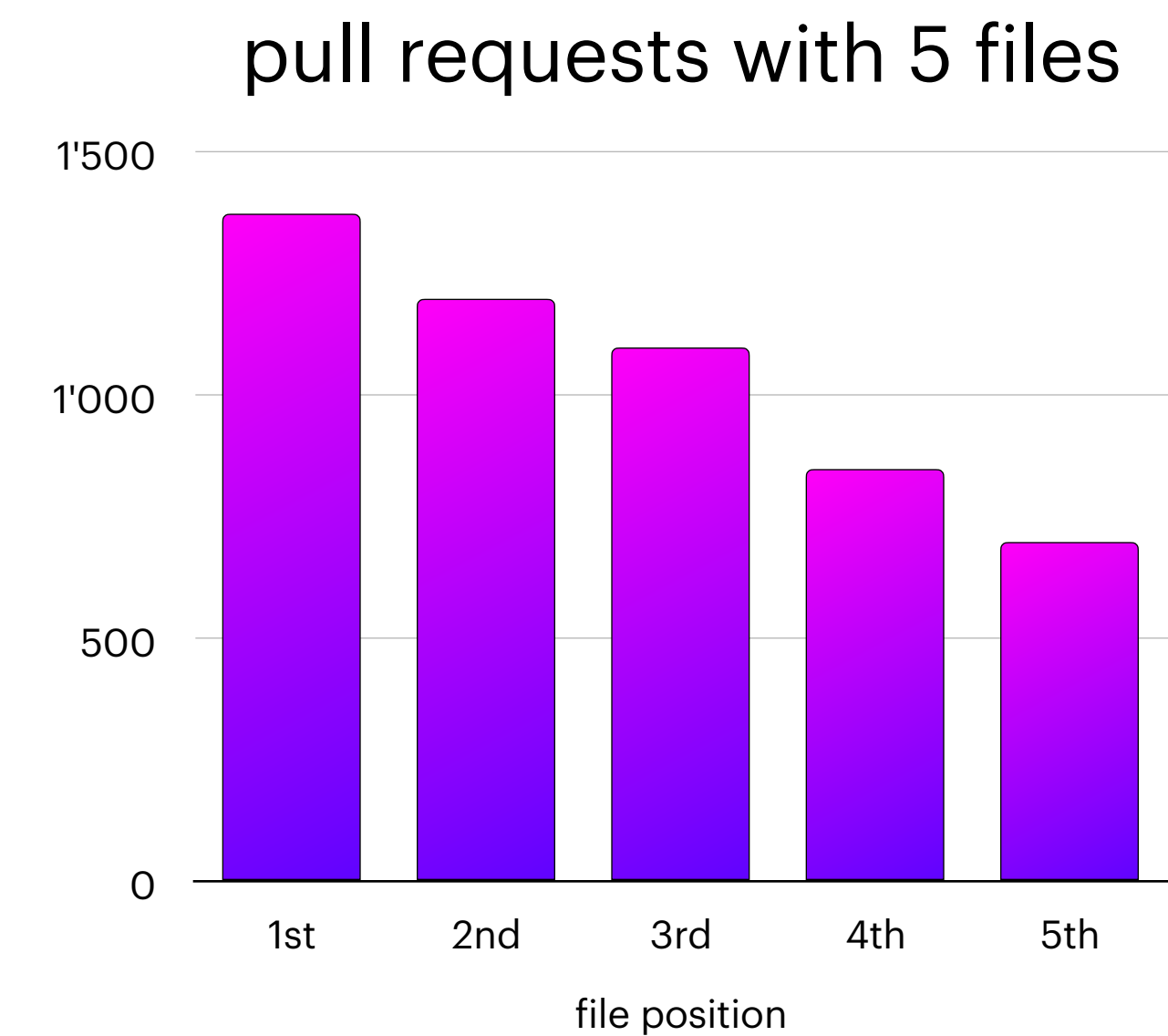
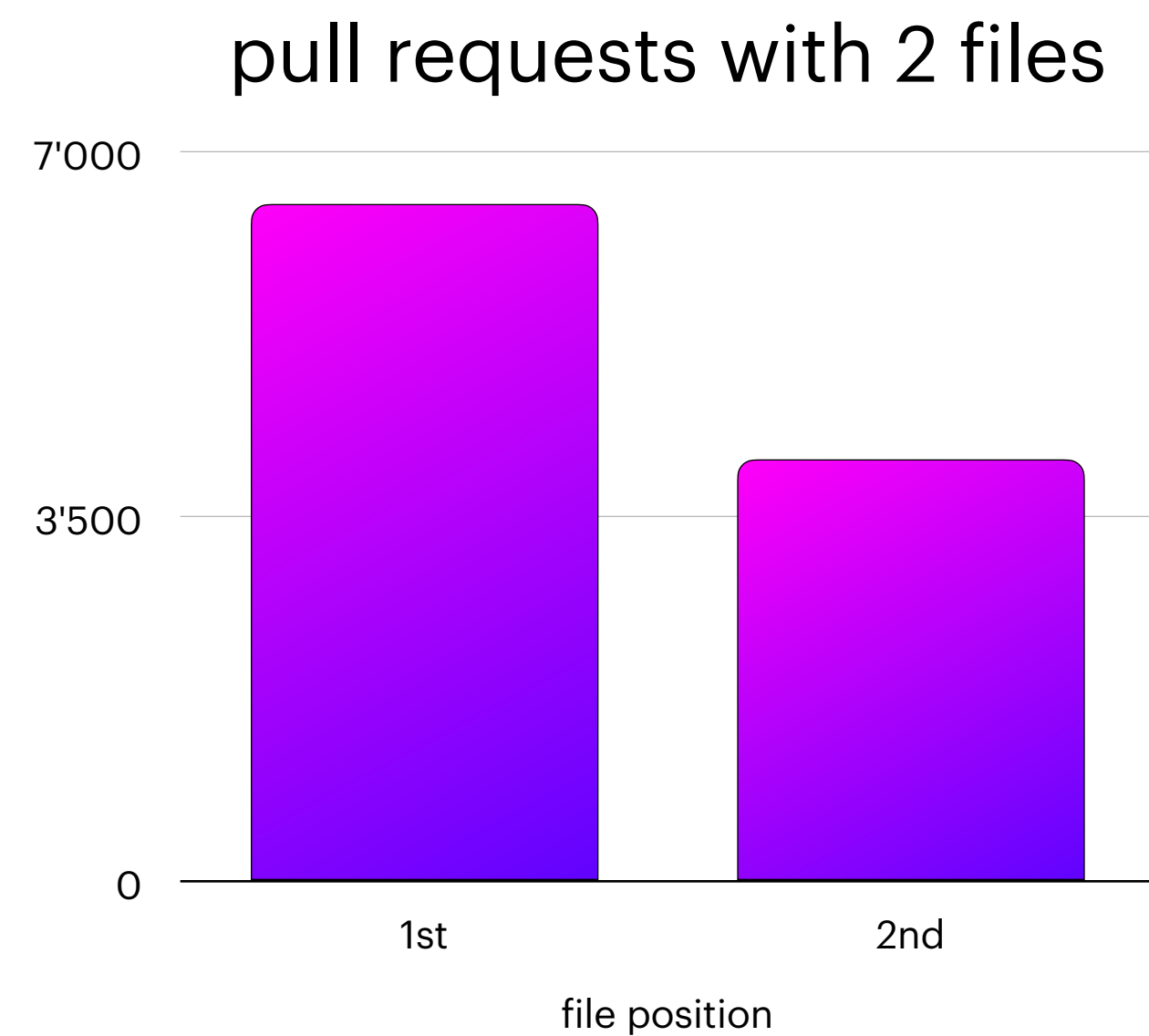
- We analyzed review comments for 200k pull requests from some 140 popular projects in GitHub...
- ... and found something remarkable.

## cumulative number of review comments by file position

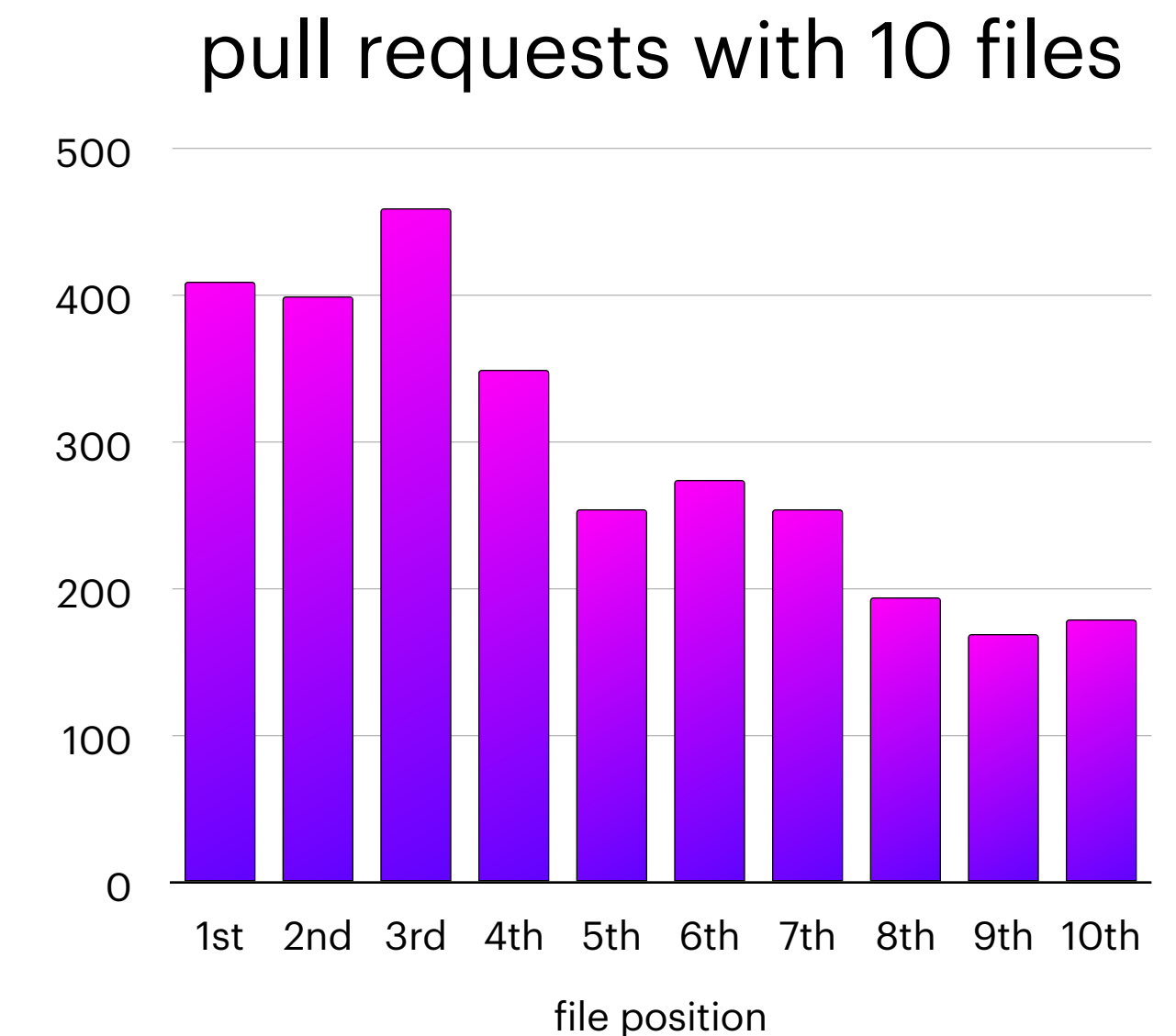
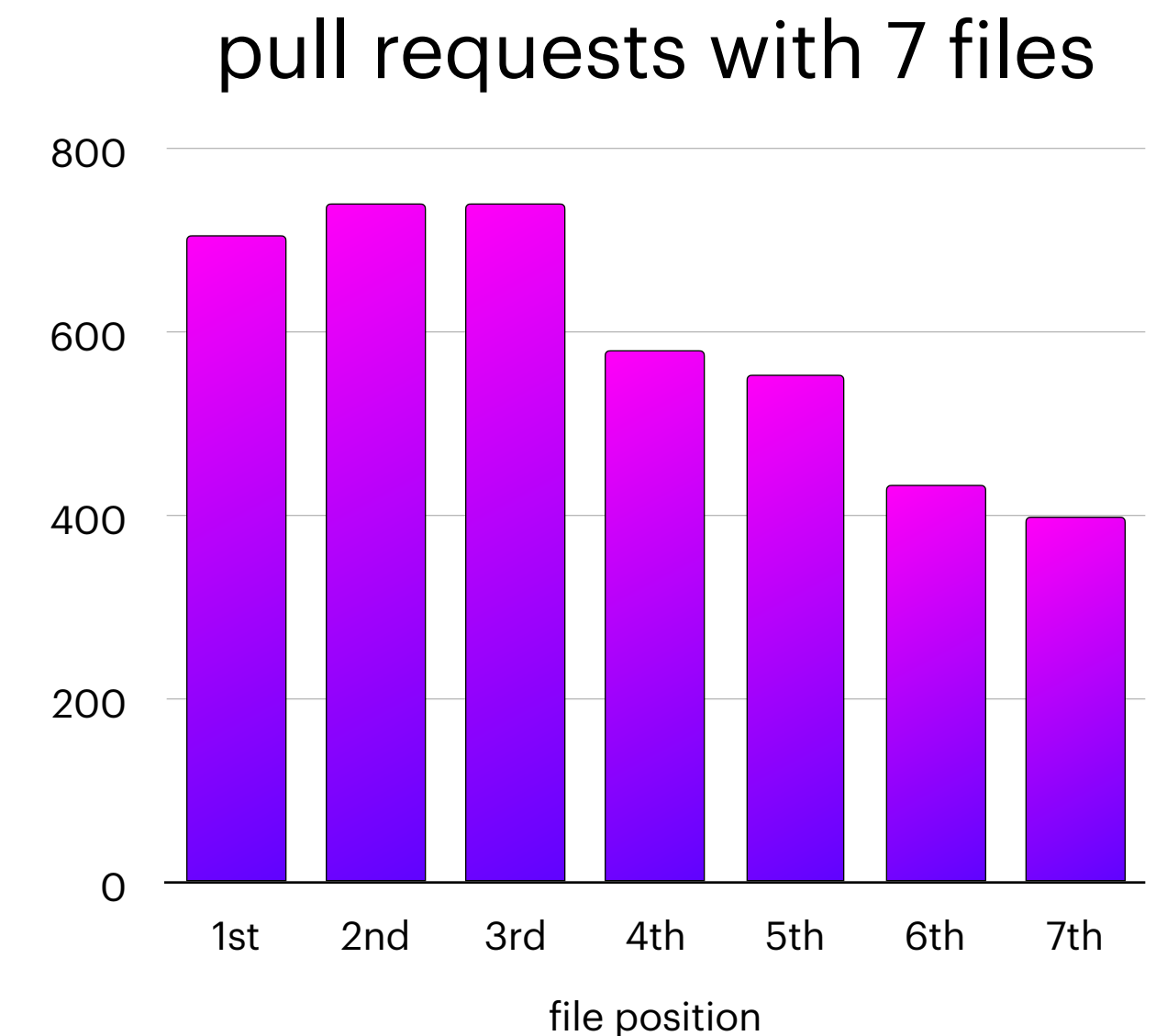


# code review production files

- We analyzed review comments for 200k pull requests from some 140 popular projects in GitHub...
- ... and found something remarkable.



## cumulative number of review comments by file position



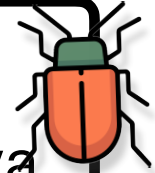


**code review**

**production files experiment**

# code review


production files experiment

a.java 

b.java

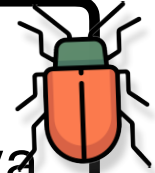
c.java

d.java

e.java 

# code review


production files experiment

a.java 

b.java

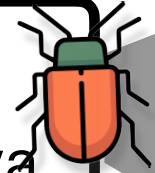
c.java

d.java

e.java 

# code review


production files experiment

a.java 

b.java

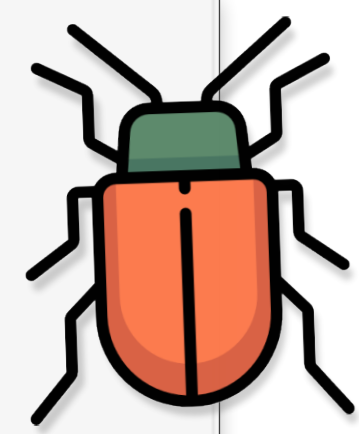
c.java

d.java

e.java 


```
26     switch (destinationAddress.getCountry()) {
27         case "USA":
28             shippingCost = shippingCost * 1.2;
29             break;
30         case "Canada":
31             shippingCost = shippingCost * 1.18;
32             break;
33         case "Mexico":
34             shippingCost = shippingCost * 1.35;
35             break;
36         case "UK":
37             shippingCost = shippingCost * 1.27;
38         default:
39             shippingCost = shippingCost * 2;
40     }
41     return shippingCost;
```

**⚠ MB: Missing Break defect:** Here a break statement is missing. In this way, when the country is UK, the execution will fall through the default case and a wrong tax of 1.27 \* 2 will be applied.



# code review


## production files experiment

a.java 

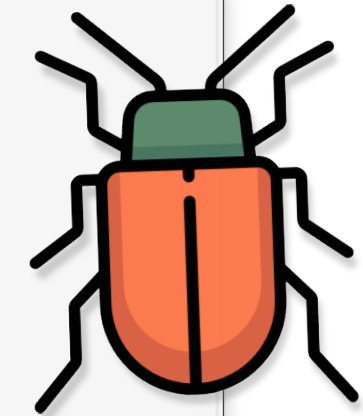
b.java

c.java

d.java

e.java 

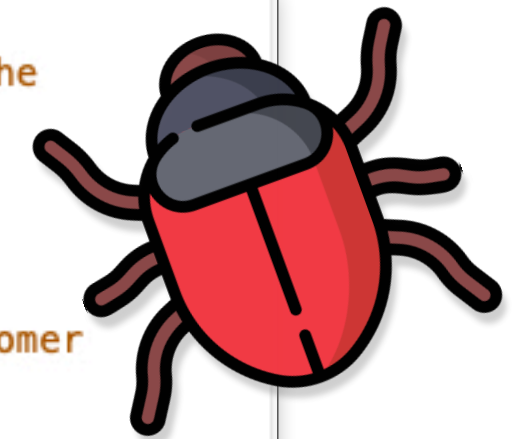
```
26     switch (destinationAddress.getCountry()) {
27         case "USA":
28             shippingCost = shippingCost * 1.2;
29             break;
30         case "Canada":
31             shippingCost = shippingCost * 1.18;
32             break;
33         case "Mexico":
34             shippingCost = shippingCost * 1.35;
35             break;
36         case "UK":
37             shippingCost = shippingCost * 1.27;
```



**❗ MB: Missing Break defect:** Here a break statement is missing. In this way, when the country is UK, the execution will fall through the default case and a wrong tax of 1.27 \* 2 will be applied.

```
38         default:
39             shippingCost = shippingCost * 2;
40     }
41     return shippingCost;
```

```
7     /**
8      * Returns the discount rate based on the membership level of the
9      * customer.
10     * Customers at level 1 do not receive any discount.
11     * Customers at level 2 to 4 receive a 10% discount.
12     * Customers from level 5 included receive a 25% discount.
13     * @param membershipLevel - the level of membership of the customer
14     * @return the discount rate applied to the customer
15     */
16     public double getSaleDiscountRate(int membershipLevel){
17         double discountRate = 0;
18         if(membershipLevel > 2 && membershipLevel < 5) {
```



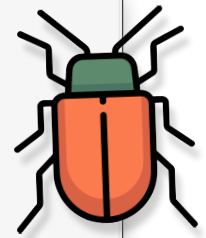
**❗ CC: Corner Case defect:** Here the if statement is missing a check for the condition where customer.membershipLevel == 2. According to the Javadoc of the function, customers with membership level equal to 2 should receive a 10% discount

```
19         }
20         if(membershipLevel >= 5) {
21             discountRate = 0.25;
22         }
23         return discountRate;
24     }
```

# code review

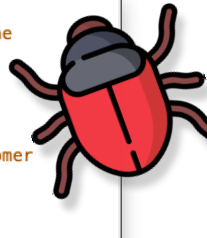
## production files experiment

```
26 switch (destinationAddress.getCountry()) {
27     case "USA":
28         shippingCost = shippingCost * 1.2;
29         break;
30     case "Canada":
31         shippingCost = shippingCost * 1.18;
32         break;
33     case "Mexico":
34         shippingCost = shippingCost * 1.35;
35         break;
36     case "UK":
37         shippingCost = shippingCost * 1.27;
38     default:
39         shippingCost = shippingCost * 2;
40 }
41 return shippingCost;
```

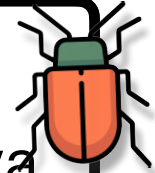


**MB: Missing Break defect:** Here a break statement is missing. In this way, when the country is UK, the execution will fall through the default case and a wrong tax of 1.27 \* 2 will be applied.

```
7 /**
8  * Returns the discount rate based on the membership level of the
9  * customer.
10 * Customers at level 1 do not receive any discount.
11 * Customers at level 2 to 4 receive a 10% discount.
12 * Customers from level 5 included receive a 25% discount.
13 * @param membershipLevel - the level of membership of the customer
14 * @return the discount rate applied to the customer
15 */
16 public double getSaleDiscountRate(int membershipLevel){
17     double discountRate = 0;
18     if(membershipLevel > 2 && membershipLevel < 5) {
19         discountRate = 0.1;
20     }
21     if(membershipLevel >= 5) {
22         discountRate = 0.25;
23     }
24     return discountRate;
25 }
```




**CC: Corner Case defect:** Here the if statement is missing a check for the condition where customer.membershipLevel == 2. According to the Javadoc of the function, customers with membership level equal to 2 should receive a 10% discount.

a.java 

b.java

c.java

d.java

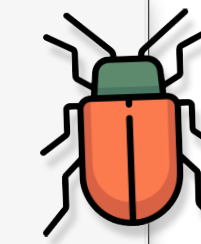
e.java 

# code review

## production files experiment

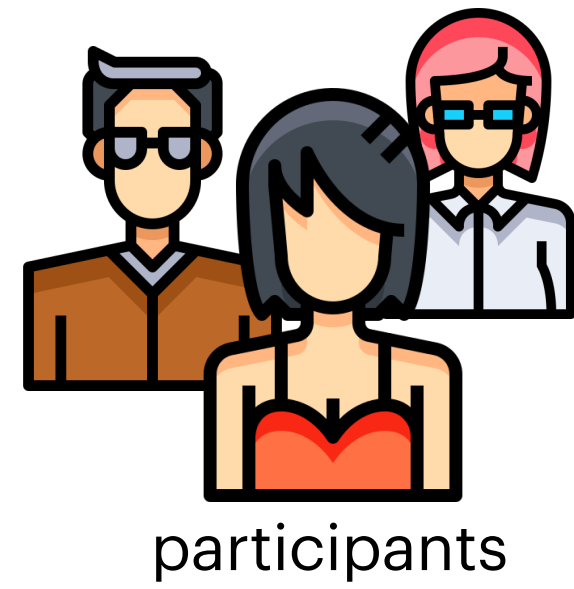
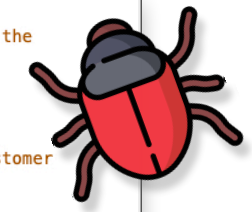
```
26 switch (destinationAddress.getCountry()) {
27     case "USA":
28         shippingCost = shippingCost * 1.2;
29         break;
30     case "Canada":
31         shippingCost = shippingCost * 1.18;
32         break;
33     case "Mexico":
34         shippingCost = shippingCost * 1.35;
35         break;
36     case "UK":
37         shippingCost = shippingCost * 1.27;
38     default:
39         shippingCost = shippingCost * 2;
40 }
41 return shippingCost;
```

**MB: Missing Break defect:** Here a break statement is missing. In this way, when the country is UK, the execution will fall through the default case and a wrong tax of 1.27 \* 2 will be applied.

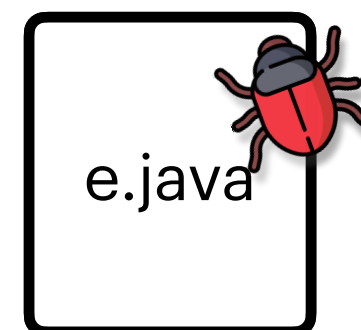
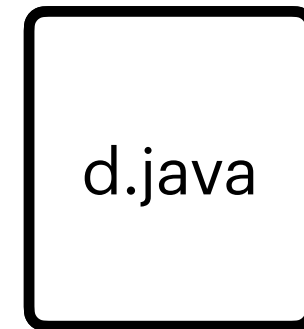
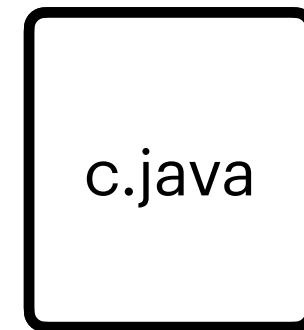
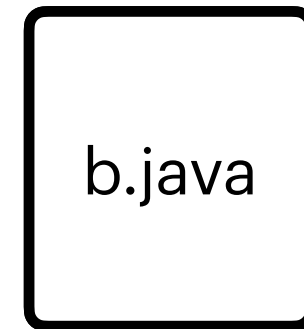
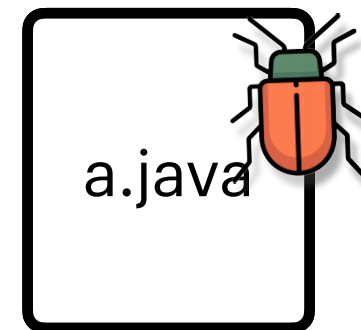


```
7 /**
8  * Returns the discount rate based on the membership level of the
9  * customer.
10 * Customers at level 1 do not receive any discount.
11 * Customers at level 2 to 4 receive a 10% discount.
12 * Customers from level 5 included receive a 25% discount.
13 * @param membershipLevel - the level of membership of the customer
14 * @return the discount rate applied to the customer
15 */
16 public double getSaleDiscountRate(int membershipLevel){
17     double discountRate = 0;
18     if(membershipLevel > 2 && membershipLevel < 5) {
19         discountRate = 0.1;
20     }
21     if(membershipLevel >= 5) {
22         discountRate = 0.25;
23     }
24     return discountRate;
25 }
```

**CC: Corner Case defect:** Here the if statement is missing a check for the condition where customer.membershipLevel == 2. According to the Javadoc of the function, customers with membership level equal to 2 should receive a 10% discount.



participants



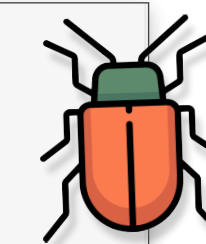
participants

# code review

## production files experiment

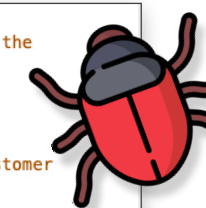
```
26 switch (destinationAddress.getCountry()) {
27     case "USA":
28         shippingCost = shippingCost * 1.2;
29         break;
30     case "Canada":
31         shippingCost = shippingCost * 1.18;
32         break;
33     case "Mexico":
34         shippingCost = shippingCost * 1.35;
35         break;
36     case "UK":
37         shippingCost = shippingCost * 1.27;
38     default:
39         shippingCost = shippingCost * 2;
40 }
41 return shippingCost;
```

**MB: Missing Break defect:** Here a break statement is missing. In this way, when the country is UK, the execution will fall through the default case and a wrong tax of 1.27 \* 2 will be applied.




```
7 /**
8  * Returns the discount rate based on the membership level of the
9  * customer.
10 * Customers at level 1 do not receive any discount.
11 * Customers at level 2 to 4 receive a 10% discount.
12 * Customers from level 5 included receive a 25% discount.
13 * @param membershipLevel - the level of membership of the customer
14 * @return the discount rate applied to the customer
15 */
16 public double getSaleDiscountRate(int membershipLevel){
17     double discountRate = 0;
18     if(membershipLevel > 2 && membershipLevel < 5) {
19         discountRate = 0.1;
20     }
21     if(membershipLevel >= 5) {
22         discountRate = 0.25;
23     }
24     return discountRate;
25 }
```

**CC: Corner Case defect:** Here the if statement is missing a check for the condition where customer.membershipLevel == 2. According to the Javadoc of the function, customers with membership level equal to 2 should receive a 10% discount.




participants


a.java 

b.java

c.java

d.java

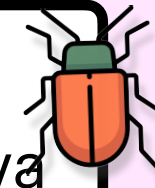
e.java 

e.java 

d.java

c.java

b.java

a.java 

participants

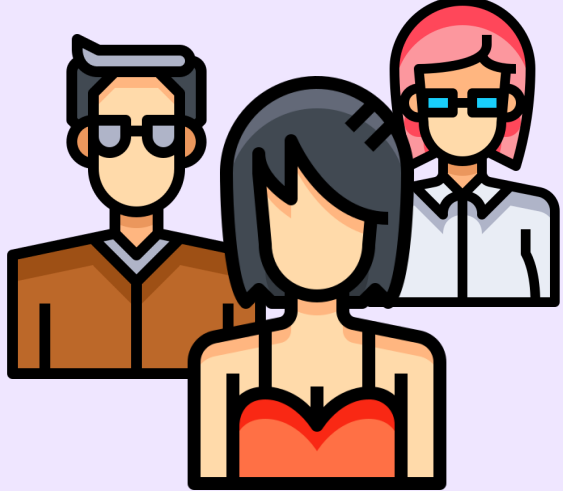


# code review

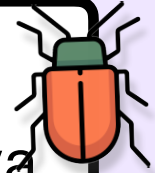
## production files experiment

```
7  /**  
8   * Returns the discount rate based on the membership level of the  
9   * customer.  
10  * Customers at level 1 do not receive any discount.  
11  * Customers at level 2 to 4 receive a 10% discount.  
12  * Customers from level 5 included receive a 25% discount.  
13  * @param membershipLevel - the level of membership of the customer  
14  * @return the discount rate applied to the customer  
15  */  
16  public double getSaleDiscountRate(int membershipLevel){  
17      double discountRate = 0;  
18      if(membershipLevel > 2 && membershipLevel < 5) {  
19          discountRate = 0.1;  
20      }  
21      if(membershipLevel >= 5) {  
22          discountRate = 0.25;  
23      }  
24      return discountRate;  
25  }
```

**CC: Corner Case defect:** Here the if statement is missing a check for the condition where customer.membershipLevel == 2. According to the Javadoc of the function, customers with membership level equal to 2 should receive a 10% discount




participants

a.java 

b.java

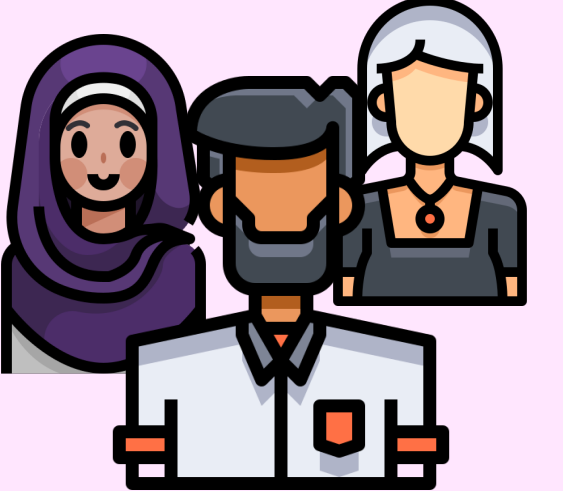
c.java

d.java


e.java 

```
26  switch (destinationAddress.getCountry()) {  
27      case "USA":  
28          shippingCost = shippingCost * 1.2;  
29          break;  
30      case "Canada":  
31          shippingCost = shippingCost * 1.18;  
32          break;  
33      case "Mexico":  
34          shippingCost = shippingCost * 1.35;  
35          break;  
36      case "UK":  
37          shippingCost = shippingCost * 1.27;  
38      default:  
39          shippingCost = shippingCost * 2;  
40  }  
41  return shippingCost;
```

**MB: Missing Break defect:** Here a break statement is missing. In this way, when the country is UK, the execution will fall through the default case and a wrong tax of 1.27 \* 2 will be applied.




participants

e.java 

d.java

c.java

b.java

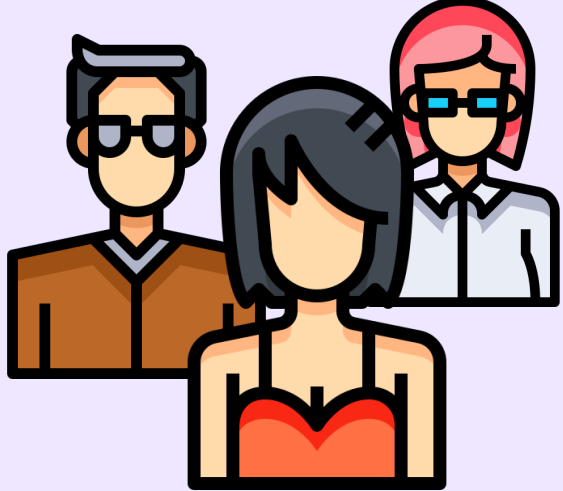
a.java 

# code review

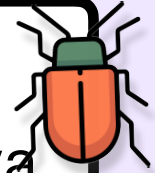
## production files experiment

```
7  /**  
8   * Returns the discount rate based on the membership level of the  
9   * customer.  
10  * Customers at level 1 do not receive any discount.  
11  * Customers at level 2 to 4 receive a 10% discount.  
12  * Customers from level 5 included receive a 25% discount.  
13  * @param membershipLevel - the level of membership of the customer  
14  * @return the discount rate applied to the customer  
15  */  
16  public double getSaleDiscountRate(int membershipLevel){  
17      double discountRate = 0;  
18      if(membershipLevel > 2 && membershipLevel < 5) {  
19          discountRate = 0.1;  
20      }  
21      if(membershipLevel >= 5) {  
22          discountRate = 0.25;  
23      }  
24      return discountRate;  
25  }
```

**CC: Corner Case defect:** Here the if statement is missing a check for the condition where customer.membershipLevel == 2. According to the Javadoc of the function, customers with membership level equal to 2 should receive a 10% discount




participants

a.java 

b.java

c.java


d.java

e.java 

```
26  switch (destinationAddress.getCountry()) {  
27      case "USA":  
28          shippingCost = shippingCost * 1.2;  
29          break;  
30      case "Canada":  
31          shippingCost = shippingCost * 1.18;  
32          break;  
33      case "Mexico":  
34          shippingCost = shippingCost * 1.35;  
35          break;  
36      case "UK":  
37          shippingCost = shippingCost * 1.27;  
38      default:  
39          shippingCost = shippingCost * 2;  
40  }  
41  return shippingCost;
```

**MB: Missing Break defect:** Here a break statement is missing. In this way, when the country is UK, the execution will fall through the default case and a wrong tax of 1.27 \* 2 will be applied.


same likelihood of finding the bug

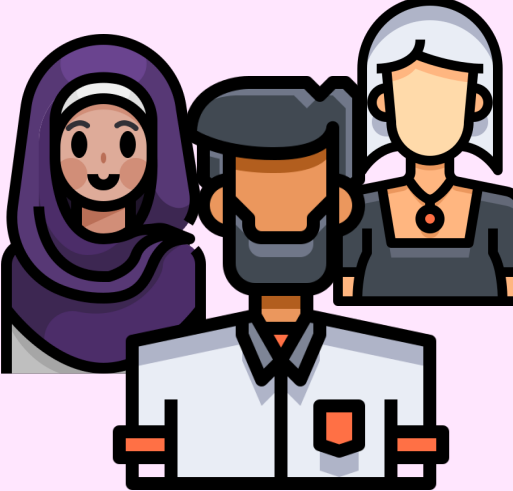
e.java 

d.java

c.java

b.java

a.java 



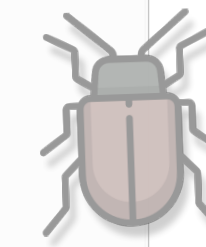
participants

# code review

## production files experiment

```
26 switch (destinationAddress.getCountry()) {
27     case "USA":
28         shippingCost = shippingCost * 1.2;
29         break;
30     case "Canada":
31         shippingCost = shippingCost * 1.18;
32         break;
33     case "Mexico":
34         shippingCost = shippingCost * 1.35;
35         break;
36     case "UK":
37         shippingCost = shippingCost * 1.27;
38     default:
39         shippingCost = shippingCost * 2;
40 }
41 return shippingCost;
```

**MB: Missing Break defect:** Here a break statement is missing. In this way, when the country is UK, the execution will fall through the default case and a wrong tax of 1.27 \* 2 will be applied.



participants

a.java 

b.java

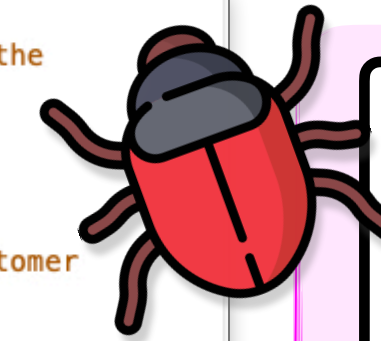
c.java


d.java

e.java 

```
7 /**
8  * Returns the discount rate based on the membership level of the
9  * customer.
10 * Customers at level 1 do not receive any discount.
11 * Customers at level 2 to 4 receive a 10% discount.
12 * Customers from level 5 included receive a 25% discount.
13 * @param membershipLevel - the level of membership of the customer
14 * @return the discount rate applied to the customer
15 */
16 public double getSaleDiscountRate(int membershipLevel){
17     double discountRate = 0;
18     if(membershipLevel > 2 && membershipLevel < 5) {
19         discountRate = 0.1;
20     }
21     if(membershipLevel >= 5) {
22         discountRate = 0.25;
23     }
24     return discountRate;
25 }
```

**CC: Corner Case defect:** Here the if statement is missing a check for the condition where customer.membershipLevel == 2. According to the Javadoc of the function, customers with membership level equal to 2 should receive a 10% discount




e.java 

d.java

c.java

b.java

a.java 

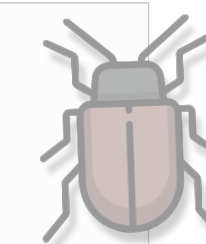
participants

# code review

production files experiment

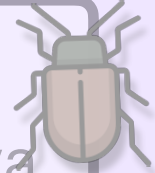
```
26 switch (destinationAddress.getCountry()) {
27     case "USA":
28         shippingCost = shippingCost * 1.2;
29         break;
30     case "Canada":
31         shippingCost = shippingCost * 1.18;
32         break;
33     case "Mexico":
34         shippingCost = shippingCost * 1.35;
35         break;
36     case "UK":
37         shippingCost = shippingCost * 1.27;
38     default:
39         shippingCost = shippingCost * 2;
40 }
41 return shippingCost;
```

**MB: Missing Break defect:** Here a break statement is missing. In this way, when the country is UK, the execution will fall through the default case and a wrong tax of 1.27 \* 2 will be applied.



175% more likely to find the bug


participants

a.java 

b.java

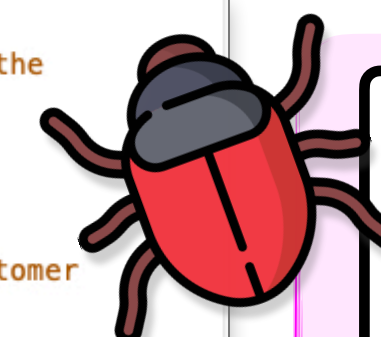
c.java


d.java

e.java 

```
7 /**
8  * Returns the discount rate based on the membership level of the
9  * customer.
10 * Customers at level 1 do not receive any discount.
11 * Customers at level 2 to 4 receive a 10% discount.
12 * Customers from level 5 included receive a 25% discount.
13 * @param membershipLevel - the level of membership of the customer
14 * @return the discount rate applied to the customer
15 */
16 public double getSaleDiscountRate(int membershipLevel){
17     double discountRate = 0;
18     if(membershipLevel > 2 && membershipLevel < 5) {
19         discountRate = 0.1;
20     }
21     if(membershipLevel >= 5) {
22         discountRate = 0.25;
23     }
24     return discountRate;
25 }
```

**CC: Corner Case defect:** Here the if statement is missing a check for the condition where customer.membershipLevel == 2. According to the Javadoc of the function, customers with membership level equal to 2 should receive a 10% discount

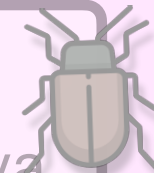


e.java 

d.java

c.java

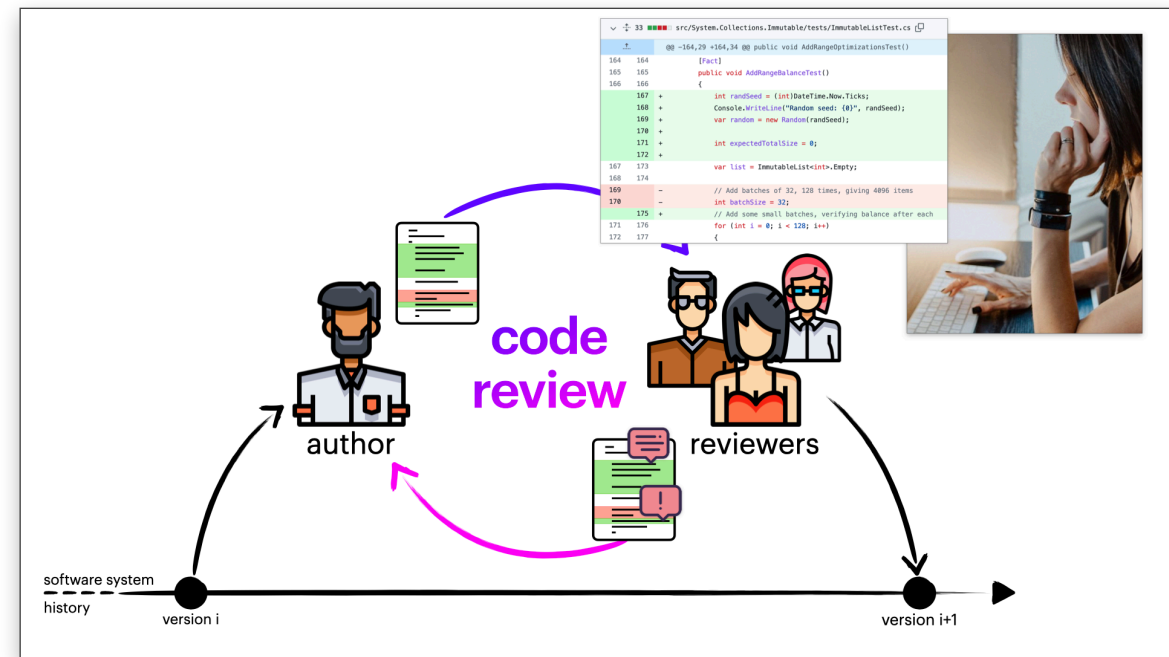
b.java

a.java 

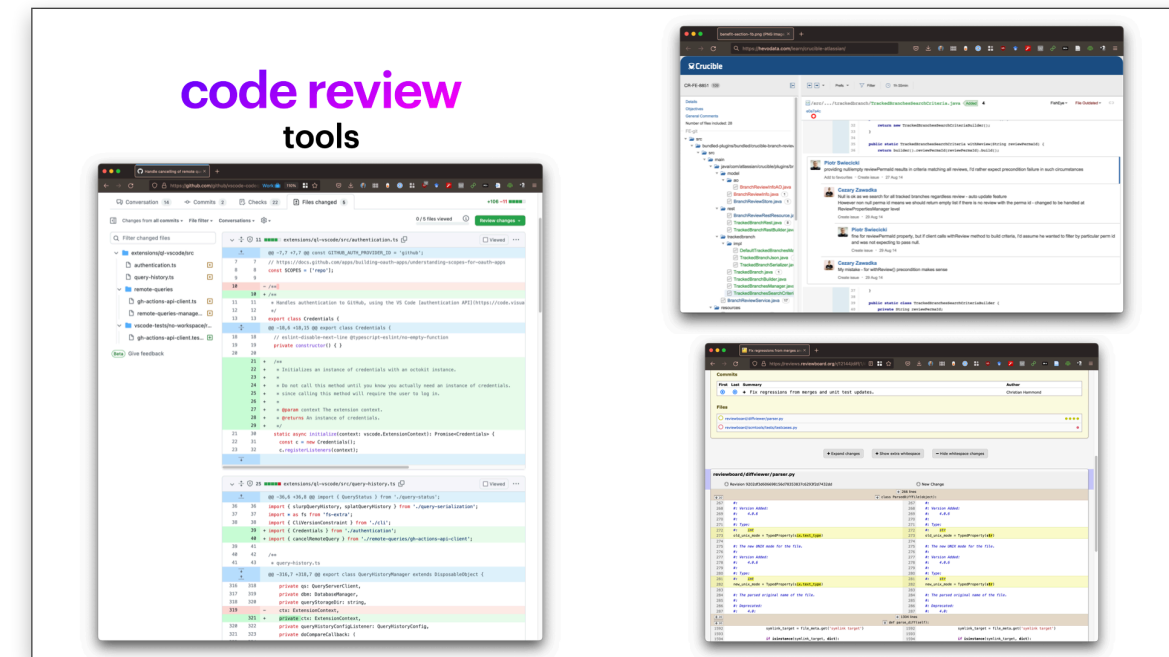
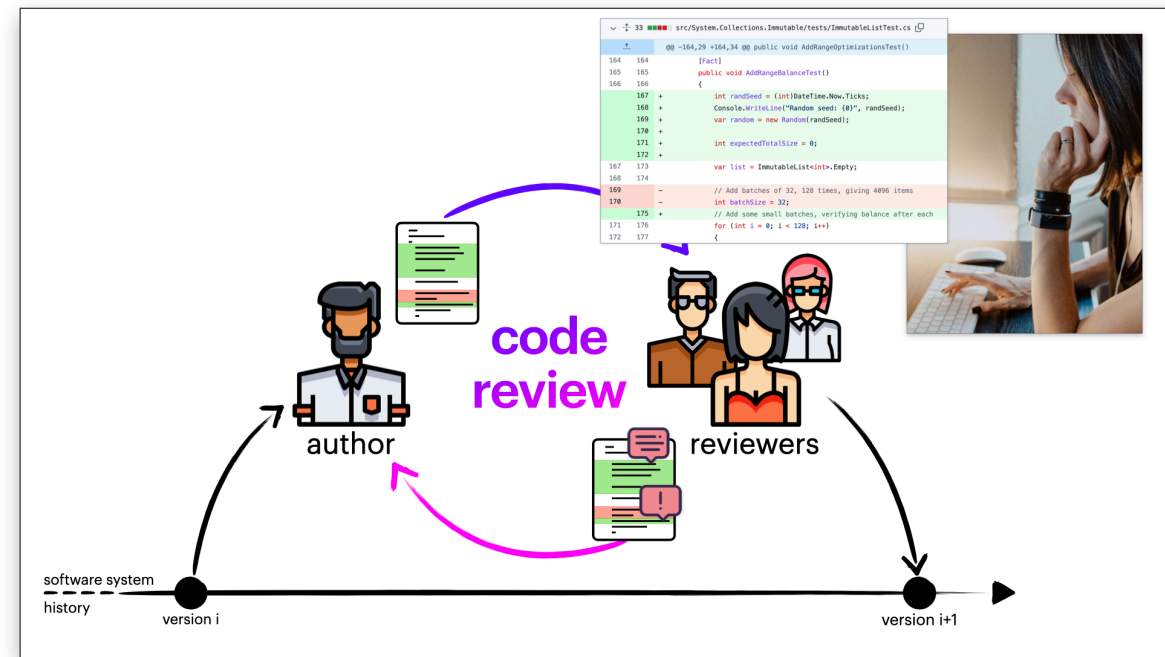
participants

**review**

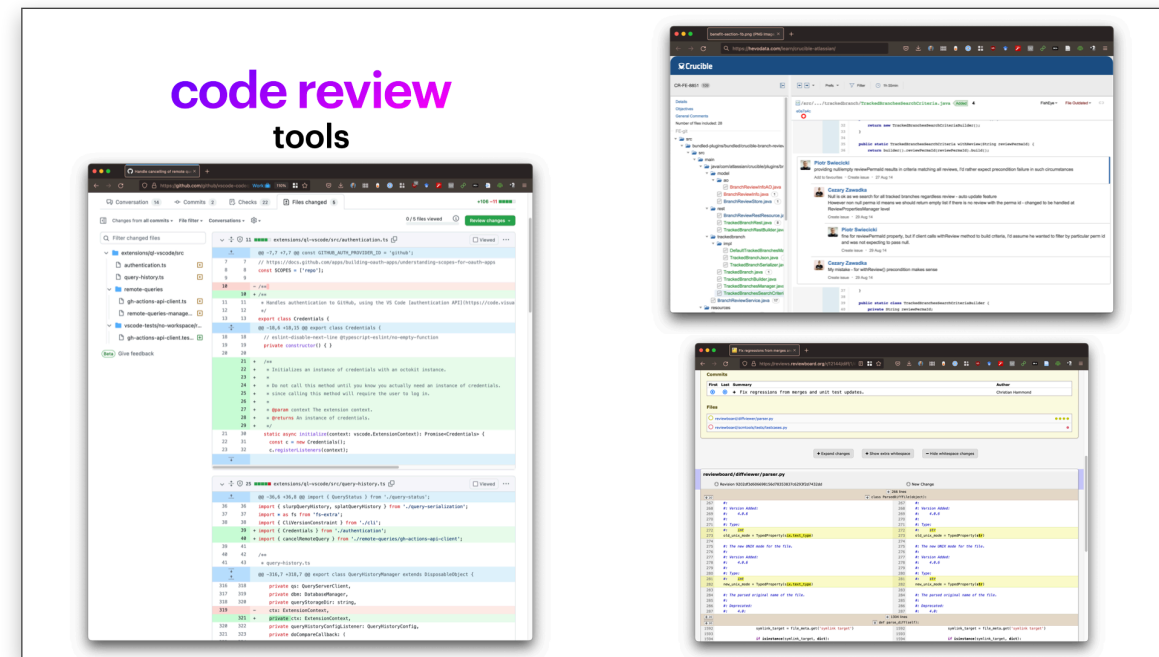
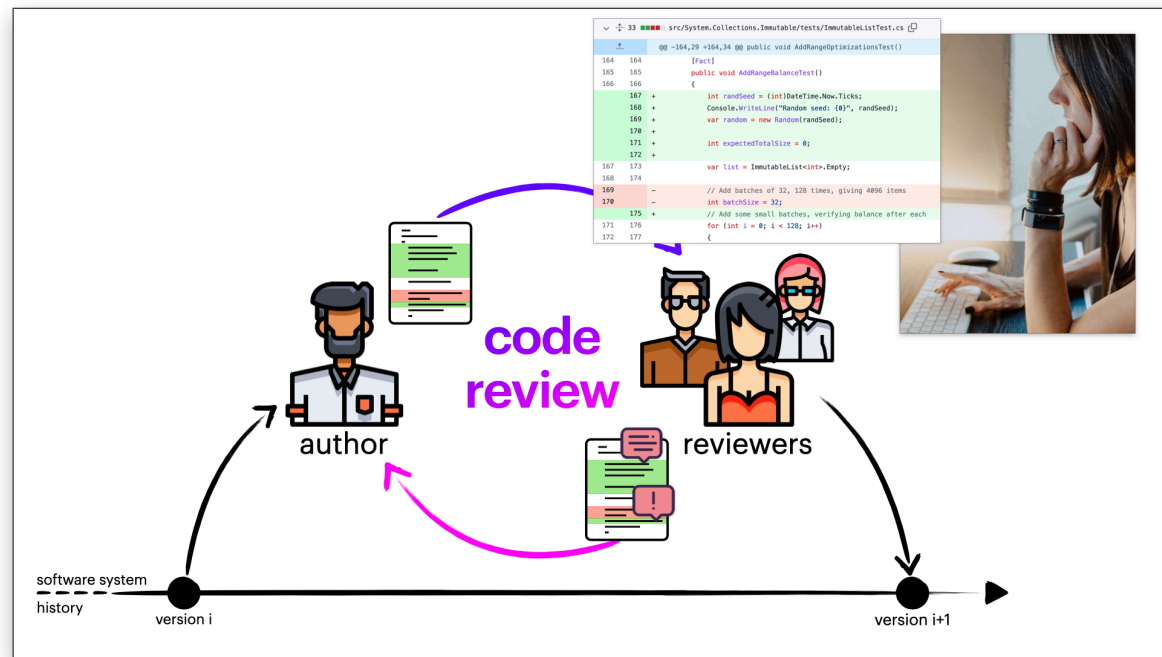
# review



# review



# review



## code review test files

- Developers see test code as less important than production code.
- Test files are almost twice less likely to be discussed during code review when together with production files.

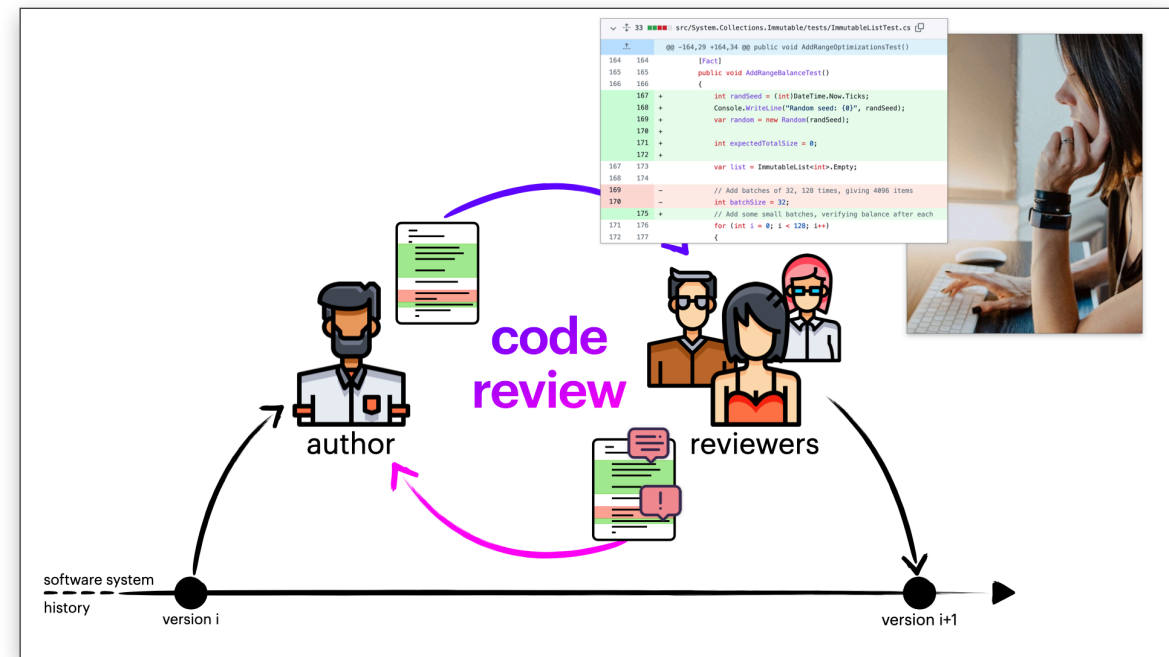
[Spadini, Aniche, Storey, Bruntink, Bacchelli - ICSE 2018]

The screenshot shows a code review tool interface with a list of files. The 'test' files are highlighted in pink, and the 'production' files are highlighted in blue. The 'test' files are significantly fewer in number than the 'production' files.





# review



### code review tools

Three screenshots of code review tools: GitHub Pull Requests, Gerrit, and Phabricator.

### code review test files

- Developers see test code as less important than production code.
- Test files are almost twice less likely to be discussed during code review when together with production files.

[Spadini, Aniche, Storey, Bruntink, Bacchelli - ICSE 2018]

A screenshot of a code review interface showing a list of files. Test files are highlighted in pink, illustrating the focus on production code during reviews.

### code review test order experiment

production code first order

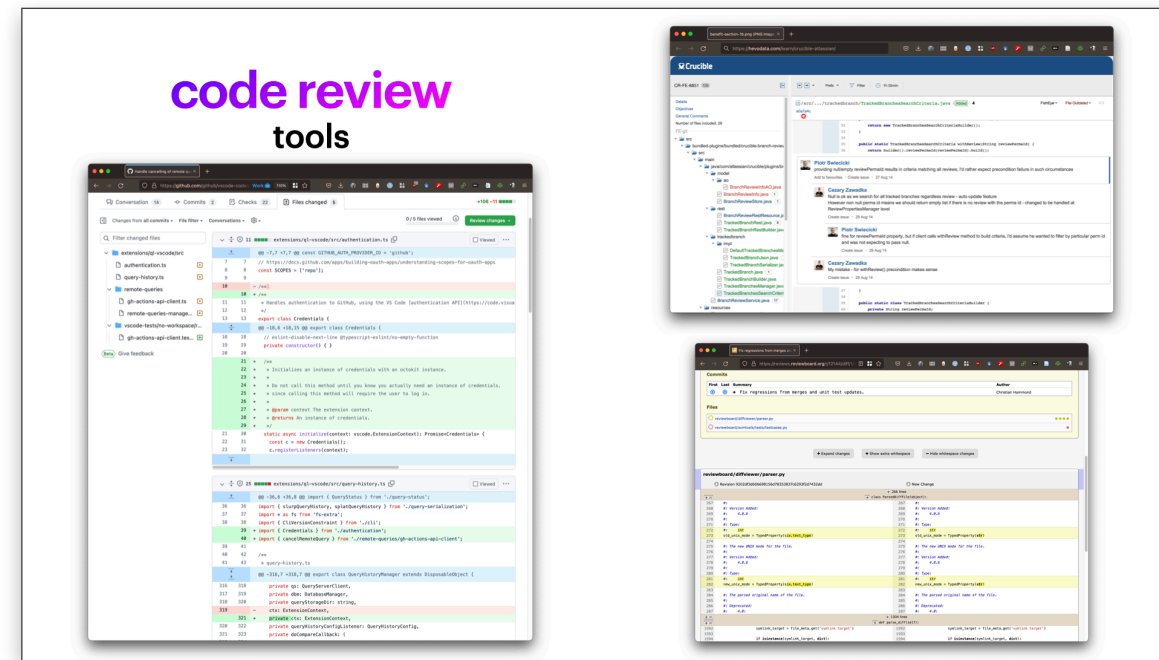
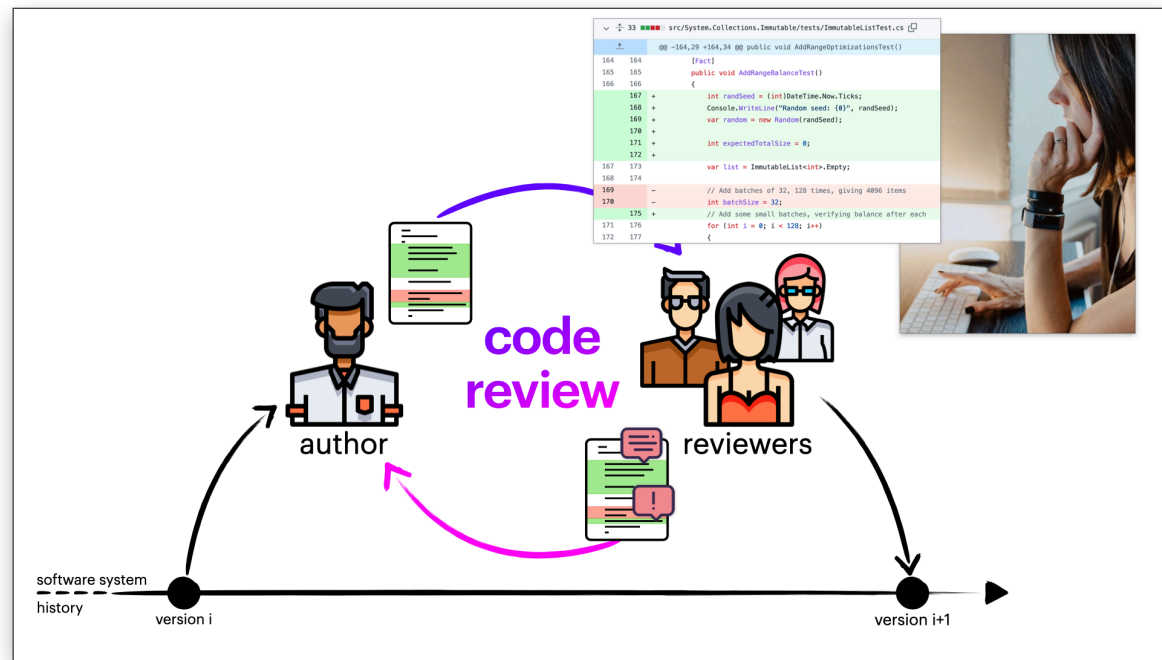
test code first order

same likelihood of finding the production bugs

250% more likely to find the test bug

The diagram compares two code review orders. In the 'production code first' order, participants review production.java and then test.java. In the 'test code first' order, participants review test.java and then production.java. The experiment shows that the 'test code first' order is 250% more likely to find the test bug.

# review



### code review test files

- Developers see test code as less important than production code.
- Test files are almost twice less likely to be discussed during code review when together with production files.

[Spadini, Aniche, Storey, Bruntink, Bacchelli - ICSE 2018]

The screenshot shows a code review interface with a list of files. Test files are highlighted in pink, and production files are highlighted in blue.

### code review test order experiment

production code first order

test code first order

same likelihood of finding the production bugs

250% more likely to find the test bug

The diagram compares two code review orders: 'production code first' and 'test code first'. The 'test code first' order is shown to be 250% more likely to find the test bug.

### code review production files

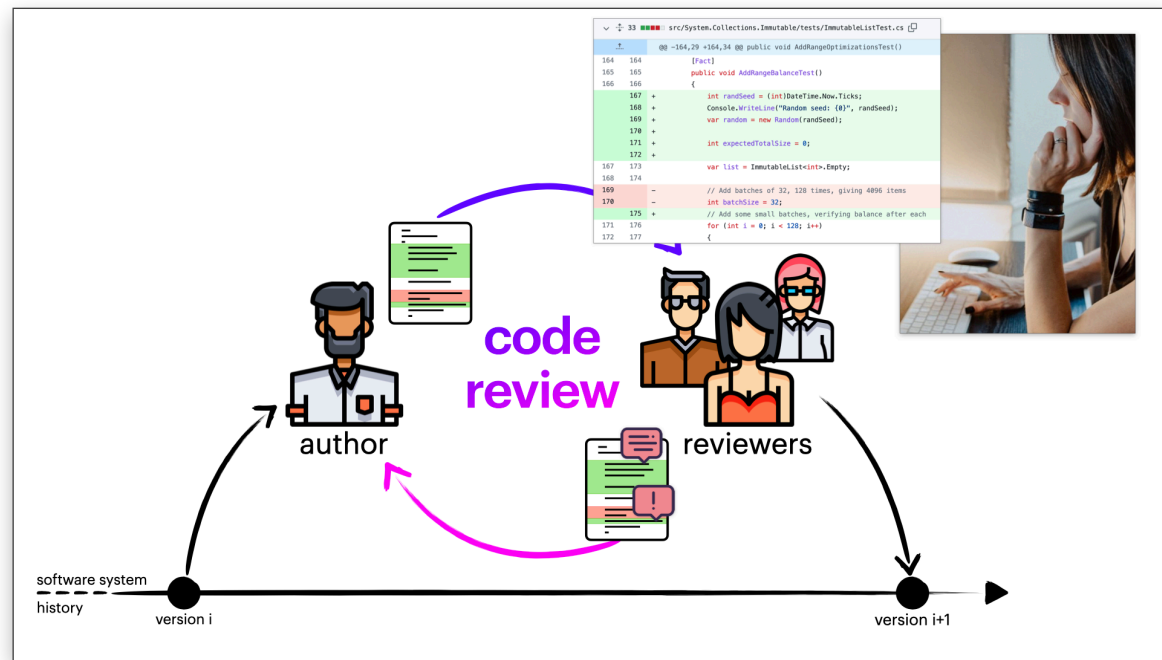
- We analyzed review comments for 200k pull requests from 138 popular projects in GitHub...
- ... and found something remarkable.

[Fregnan, Braz, D'Ambros, Çalikli, Bacchelli]

The bar chart shows the cumulative number of review comments by file position for pull requests with 5 files. The number of comments decreases as the file position increases.

file position	number of comments
1st	~1300
2nd	~1100
3rd	~900
4th	~700
5th	~500

# review



### code review tools

### code review test files

- Developers see test code as less important than production code.
- Test files are almost twice less likely to be discussed during code review when together with production files.

[Spadini, Aniche, Storey, Bruntink, Bacchelli - ICSE 2018]

### code review test order experiment

production code first order

test code first order

participants

production.java

test.java

participants

production.java

test.java

250% more likely to find the test bug

### code review production files

- We analyzed review comments for 200k pull requests from 138 popular projects in GitHub...
- ... and found something remarkable.

[Fregnan, Braz, D'Ambros, Çalikli, Bacchelli]

### cumulative number of review comments by file position

pull requests with 5 files

file position	number of comments
1st	~1400
2nd	~1200
3rd	~1000
4th	~800
5th	~600

### code review production files order experiment

participants

a.java

b.java

c.java

d.java

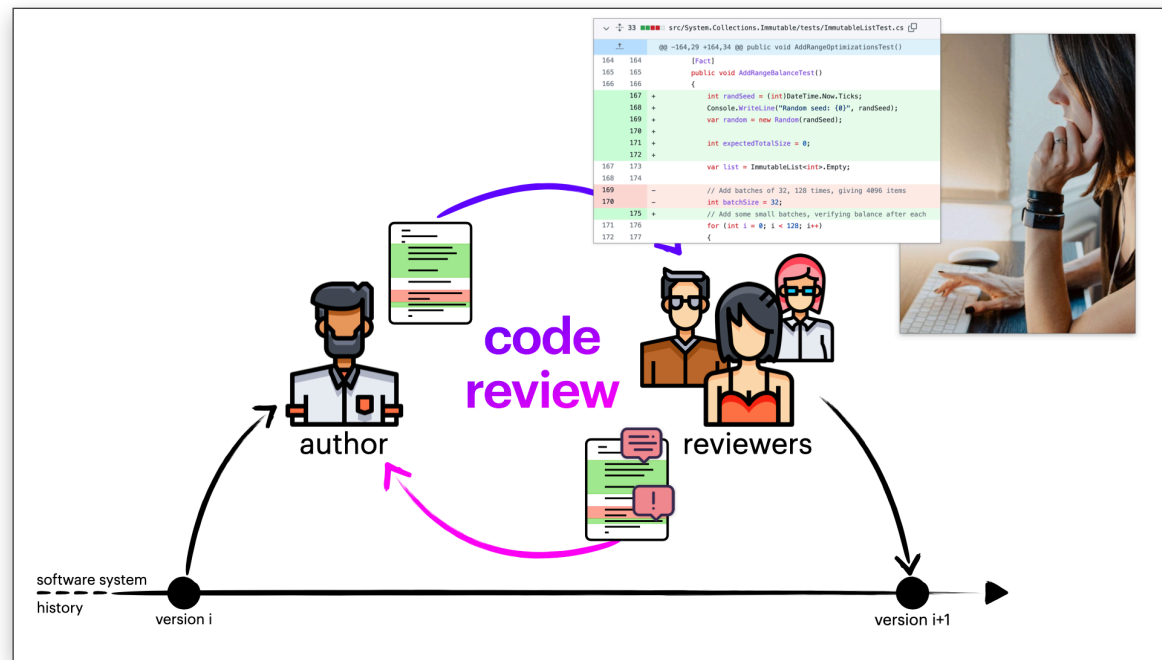
e.java

participants

175% more likely to find the bug when in the 1st file shown

# review

# takeaways



### code review tools

### code review test files

- Developers see test code as less important than production code.
- Test files are almost twice less likely to be discussed during code review when together with production files.

[Spadini, Aniche, Storey, Bruntink, Bacchelli - ICSE 2018]

### code review test order experiment

**production code first order**

participants

production.java

test.java

**test code first order**

participants

test.java

production.java

**250% more likely to find the test bug**

same likelihood of finding the production bugs

### code review production files

- We analyzed review comments for 200k pull requests from 138 popular projects in GitHub...
- ... and found something remarkable.

[Fregnan, Braz, D'Ambros, Çalikli, Bacchelli]

### cumulative number of review comments by file position

pull requests with 5 files

file position	number of comments
1st	~1400
2nd	~1200
3rd	~1100
4th	~850
5th	~700

### code review production files order experiment

participants

a.java

b.java

c.java

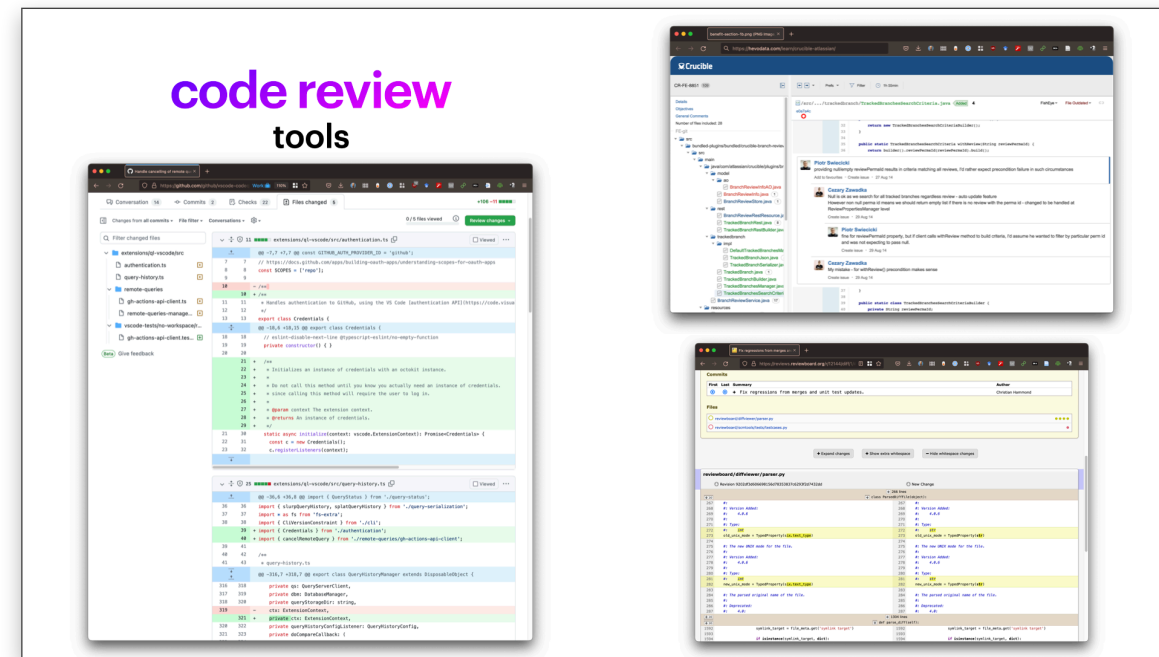
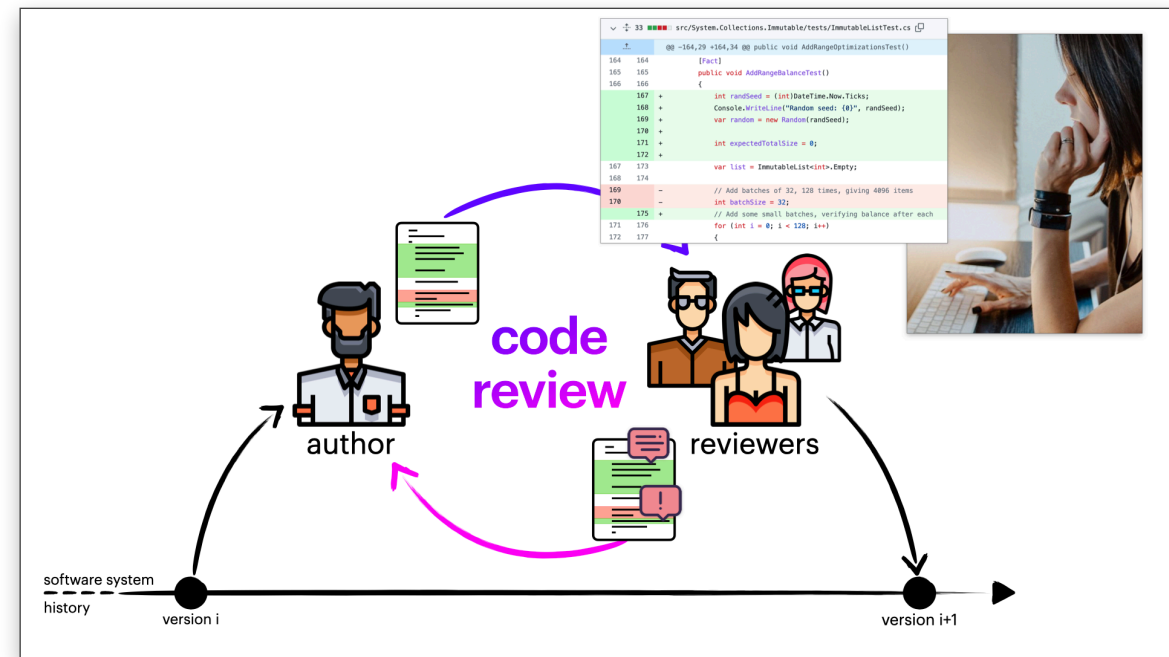
d.java

e.java

**175% more likely to find the bug when in the 1st file shown**

# review

# takeaways



- **reviewers:** be aware of this effect and decide where to start your review in a principled way.

### code review test files

- Developers see test code as less important than production code.
- Test files are almost twice less likely to be discussed during code review when together with production files.

[Spadini, Aniche, Storey, Bruntink, Bacchelli - ICSE 2018]

### code review test order experiment

production code first order

test code first order

participants

production.java

test.java

participants

production.java

test.java

250% more likely to find the test bug

### code review production files

- We analyzed review comments for 200k pull requests from 138 popular projects in GitHub...
- ... and found something remarkable.

[Fregnan, Braz, D'Ambros, Çalikli, Bacchelli]

file position	number of comments
1st	~1400
2nd	~1200
3rd	~1000
4th	~800
5th	~600

### code review production files order experiment

participants

production.java

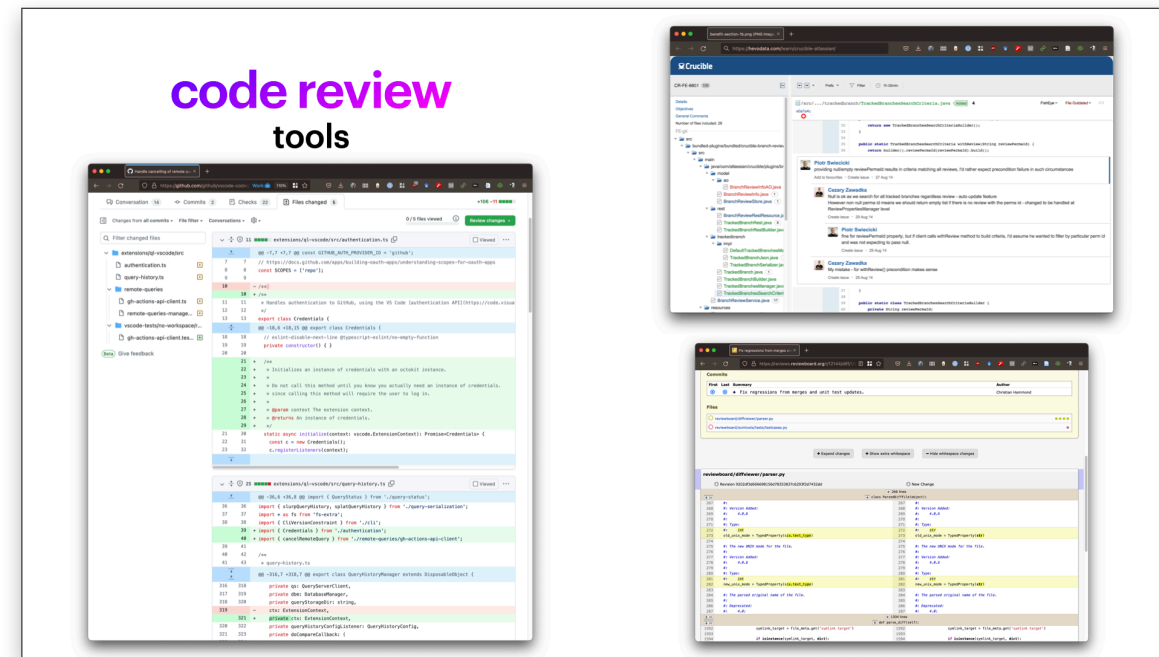
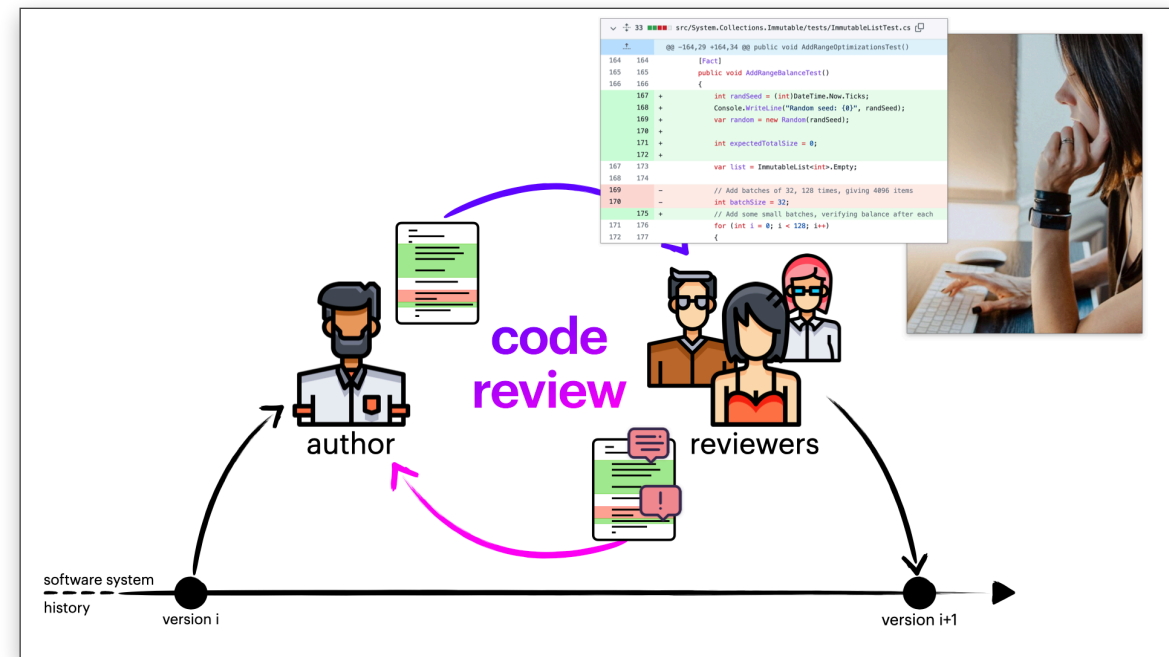
test.java

participants

175% more likely to find the bug when in the 1st file shown

# review

# takeaways



### code review test files

- Developers see test code as less important than production code.
- Test files are almost twice less likely to be discussed during code review when together with production files.

[Spadini, Aniche, Storey, Bruntink, Bacchelli - ICSE 2018]

### code review test order experiment

**production code first order**

participants

production.java

test.java

same likelihood of finding the production bugs

**test code first order**

participants

test.java

production.java

250% more likely to find the test bug

### code review production files

- We analyzed review comments for 200k pull requests from 138 popular projects in GitHub...
- ... and found something remarkable.

[Fregnan, Braz, D'Ambros, Çalikli, Bacchelli]

file position	number of comments
1st	~1300
2nd	~1100
3rd	~900
4th	~700
5th	~500

### code review production files order experiment

participants

a.java

b.java

c.java

d.java

e.java

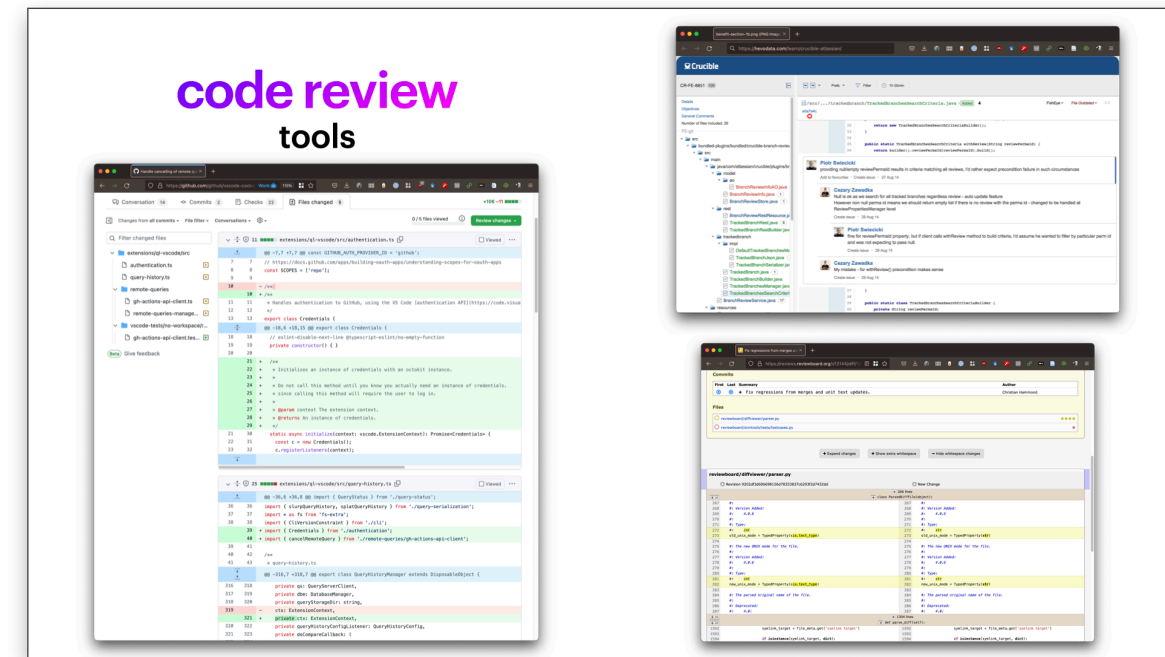
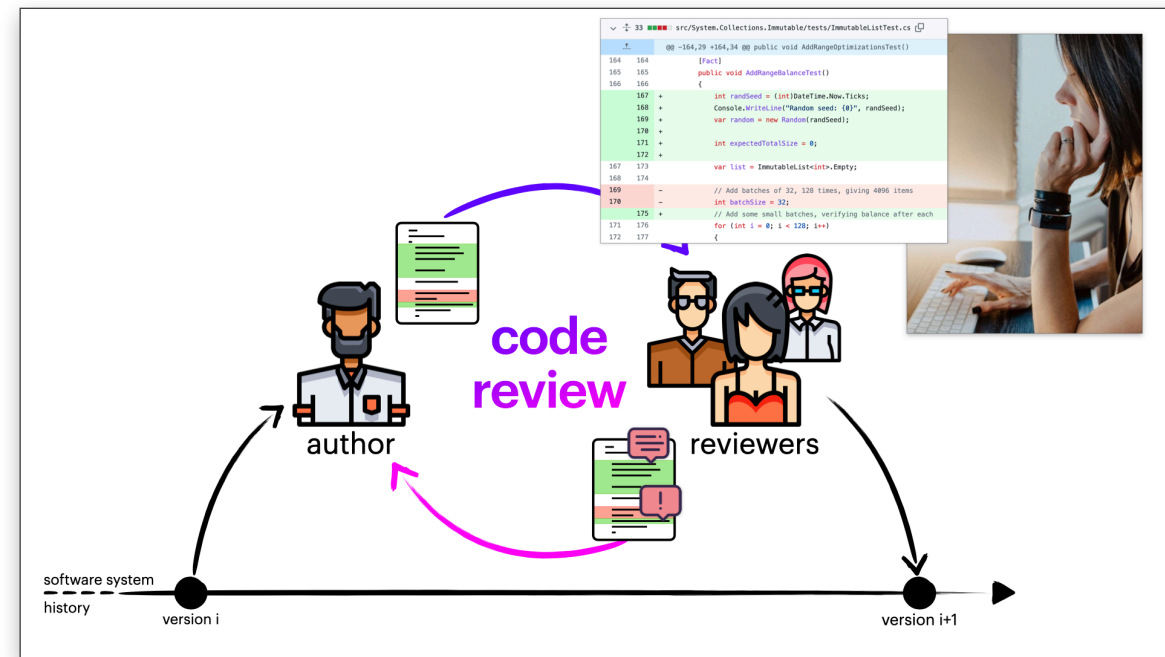
175% more likely to find the bug when in the 1st file shown

participants

- **reviewers:** be aware of this effect and decide where to start your review in a principled way.
- **authors:** guide the reviewers towards the most challenging part of your change.

# review

# takeaways



- **reviewers:** be aware of this effect and decide where to start your review in a principled way.

### code review test files

- Developers see test code as less important than production code.
- Test files are almost twice less likely to be discussed during code review when together with production files.

[Spadini, Aniche, Storey, Bruntink, Bacchelli - ICSE 2018]

code review test files

### code review test order experiment

production code first order

test code first order

same likelihood of finding the production bugs

250% more likely to find the test bug

- **authors:** guide the reviewers towards the most challenging part of your change.
- **tool builders:** empower users to choose how to order their changes and more.

### code review production files

- We analyzed review comments for 200k pull requests from 138 popular projects in GitHub...
- ... and found something remarkable.

[Fregnan, Braz, D'Ambros, Çalikli, Bacchelli]

### cumulative number of review comments by file position

pull requests with 5 files

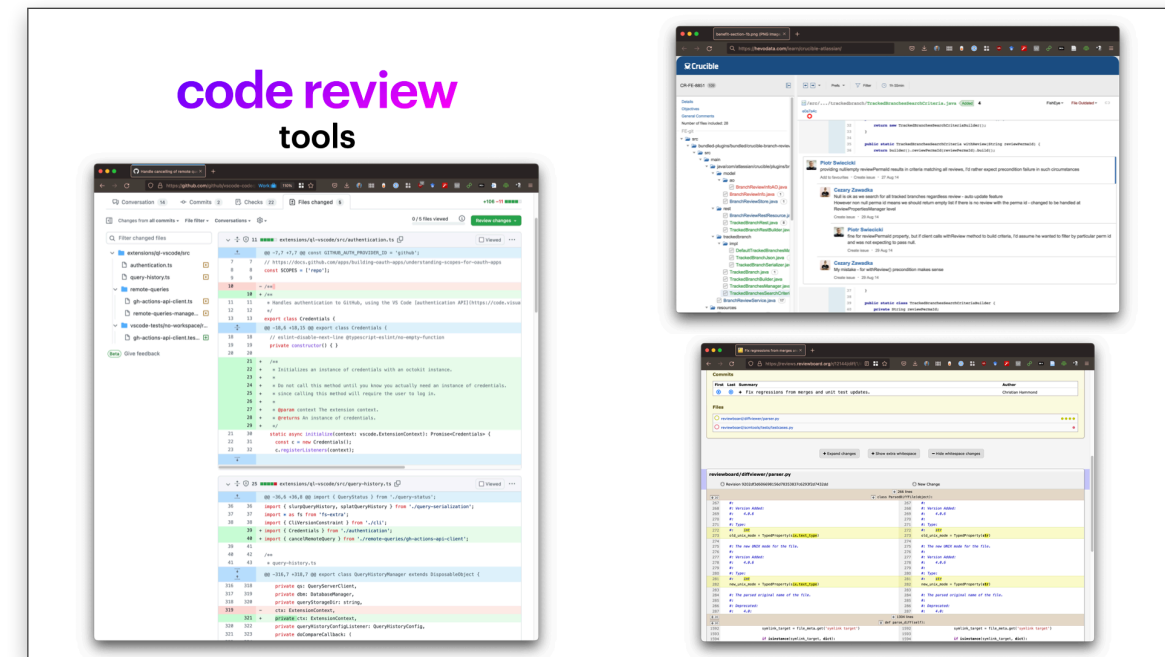
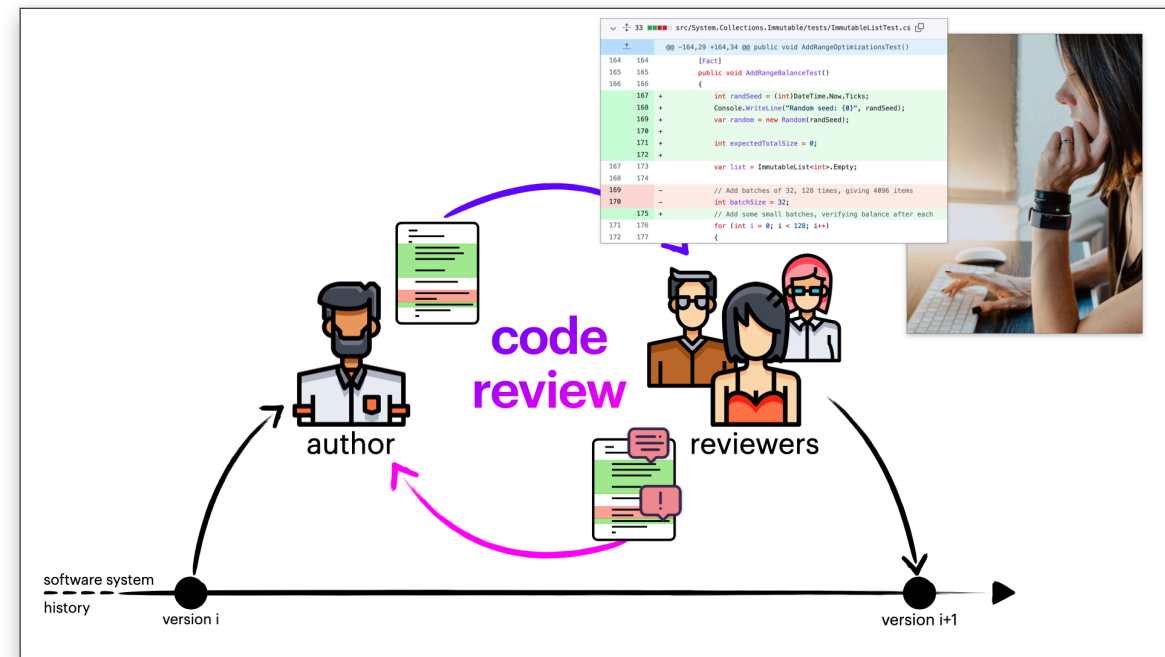
file position	number of comments
1st	~1400
2nd	~1200
3rd	~1000
4th	~800
5th	~600

### code review production files order experiment

175% more likely to find the bug when in the 1st file shown

# review

# takeaways



### code review test files

- Developers see test code as less important than production code.
- Test files are almost twice less likely to be discussed during code review when together with production files.

[Spadini, Aniche, Storey, Bruntink, Bacchelli - ICSE 2018]

The screenshot shows a code editor interface with a list of files. Test files are highlighted in pink, indicating their relative importance or frequency of discussion during code reviews.

### code review test order experiment

production code first order

test code first order

same likelihood of finding the production bugs

250% more likely to find the test bug

This diagram compares two code review orders. In the 'production code first' order, participants review production.java and then test.java. In the 'test code first' order, participants review test.java and then production.java. The results show that the 'test code first' order is 250% more likely to find test bugs, while both orders have the same likelihood of finding production bugs.

### code review production files

- We analyzed review comments for 200k pull requests from 138 popular projects in GitHub...
- ... and found something remarkable.

[Fregnan, Braz, D'Ambros, Çalikli, Bacchelli]

The bar chart shows the cumulative number of review comments for pull requests with 5 files. The x-axis represents the file position (1st to 5th), and the y-axis represents the number of comments. The number of comments decreases as the file position increases.

file position	number of comments
1st	1400
2nd	1200
3rd	1000
4th	800
5th	600

### code review production files order experiment

175% more likely to find the bug when in the 1st file shown

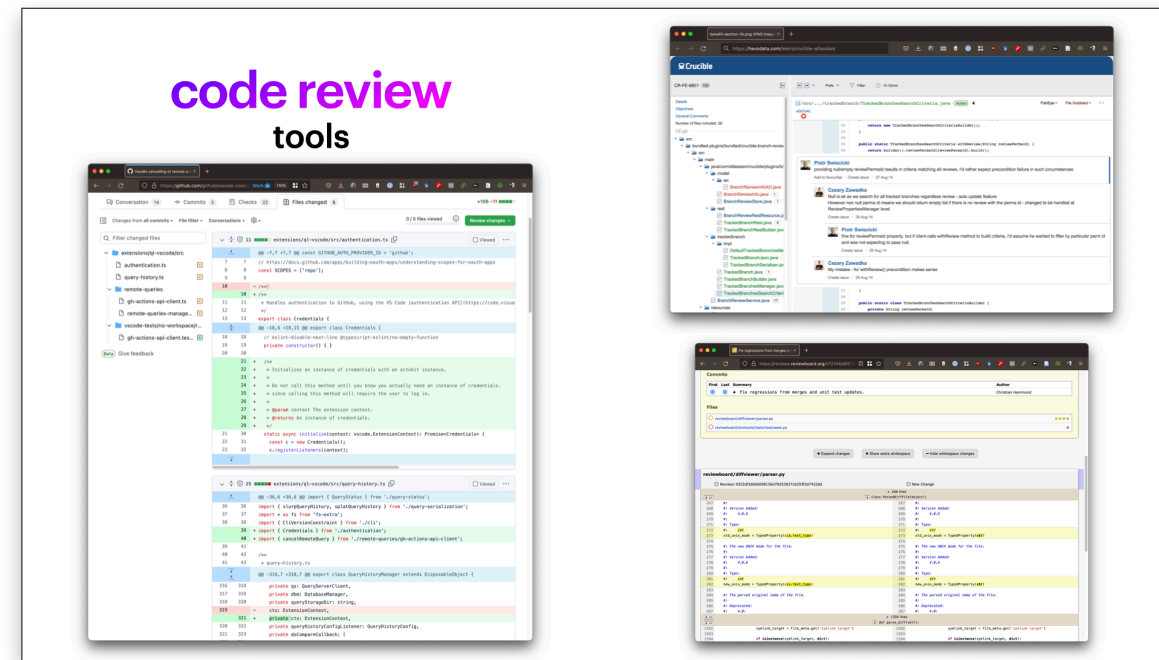
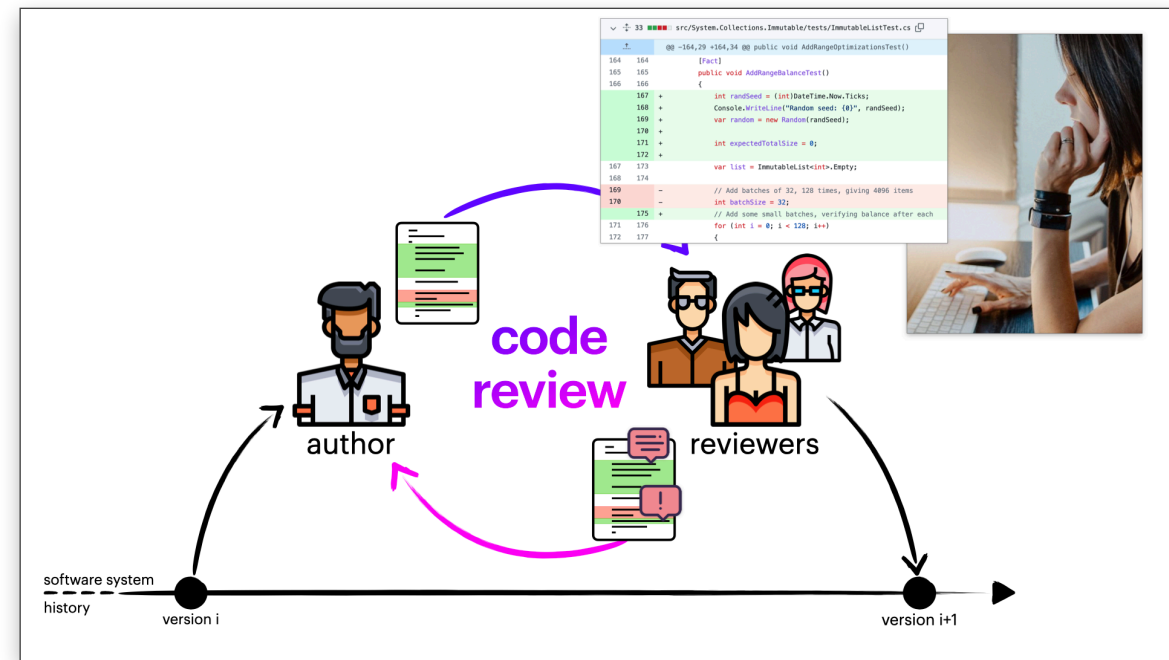
This diagram illustrates the 'production files order experiment'. It shows a sequence of files (a.java through e.java) being reviewed. The results indicate that participants are 175% more likely to find a bug when it is located in the first file shown.

- **reviewers:** be aware of this effect and decide where to start your review in a principled way.
- **authors:** guide the reviewers towards the most challenging part of your change.
- **tool builders:** empower users to choose how to order their changes and more.
- **everybody:** do not forget the power of the default settings.



# review

# takeaways



### code review test files

- Developers see test code as less important than production code.
- Test files are almost twice less likely to be discussed during code review when together with production files.

[Spadini, Aniche, Storey, Bruntink, Bacchelli - ICSE 2018]

### code review test order experiment

**production code first order**

participants

production.java

test.java

**test code first order**

participants

test.java

production.java

250% more likely to find the test bug

same likelihood of finding the production bugs

### code review production files

- We analyzed review comments for 200k pull requests from 138 popular projects in GitHub...
- ... and found something remarkable.

[Fregnan, Braz, D'Ambros, Çalikli, Bacchelli]

### cumulative number of review comments by file position

pull requests with 5 files

file position	number of comments
1st	~1400
2nd	~1200
3rd	~1000
4th	~800
5th	~600

### code review production files order experiment

participants

a.java

b.java

c.java

d.java

e.java

175% more likely to find the bug when in the 1st file shown

- **reviewers:** be aware of this effect and decide where to start your review in a principled way.
- **authors:** guide the reviewers towards the most challenging part of your change.
- **tool builders:** empower users to choose how to order their changes and more.
- **everybody:** do not forget the power of the default settings.

**Alberto Bacchelli**  
ASSOCIATE PROFESSOR  
HEAD OF ZEST

**zest**  
University of Zurich

Binzmühlestrasse 14, 8050 Zurich, Switzerland  
**ADDRESS**

zest@ifi.uzh.ch  
**EMAIL**

@ZESTuzh  
**TWITTER**

http://zest.ifi.uzh.ch  
**URL**

Credits: most presentation icons created by [justicon](#) & [freepik](#) @ [Flaticon](#)