

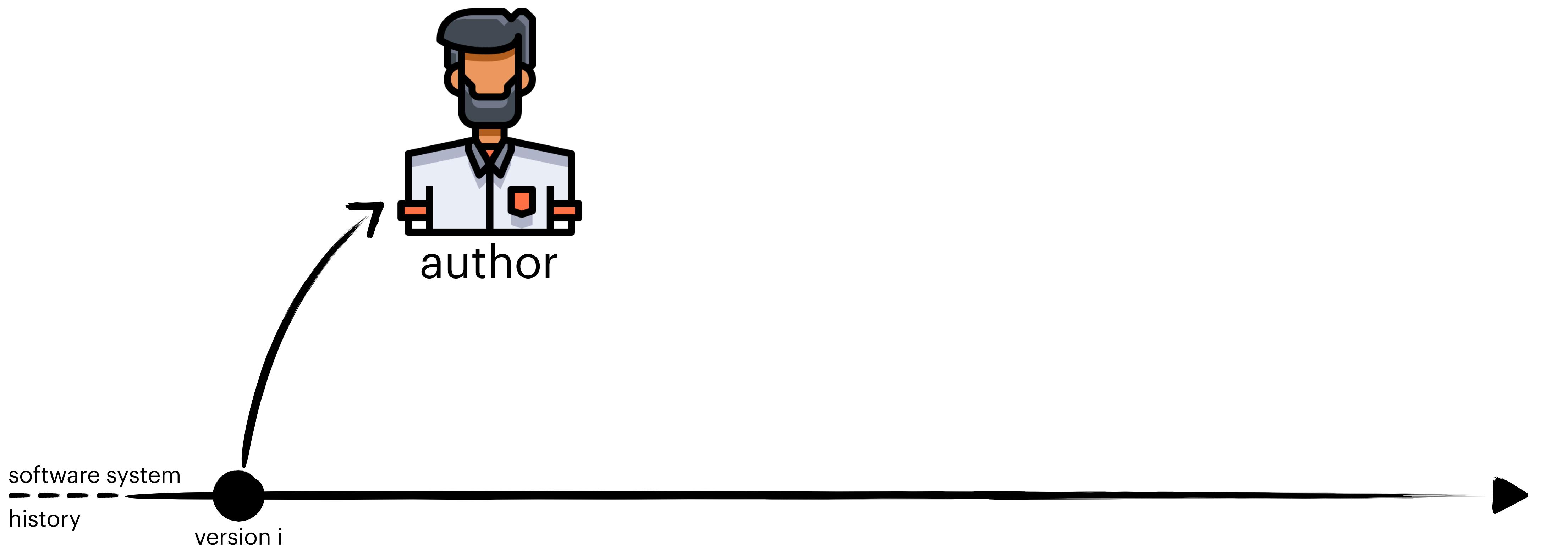
Code review effectiveness

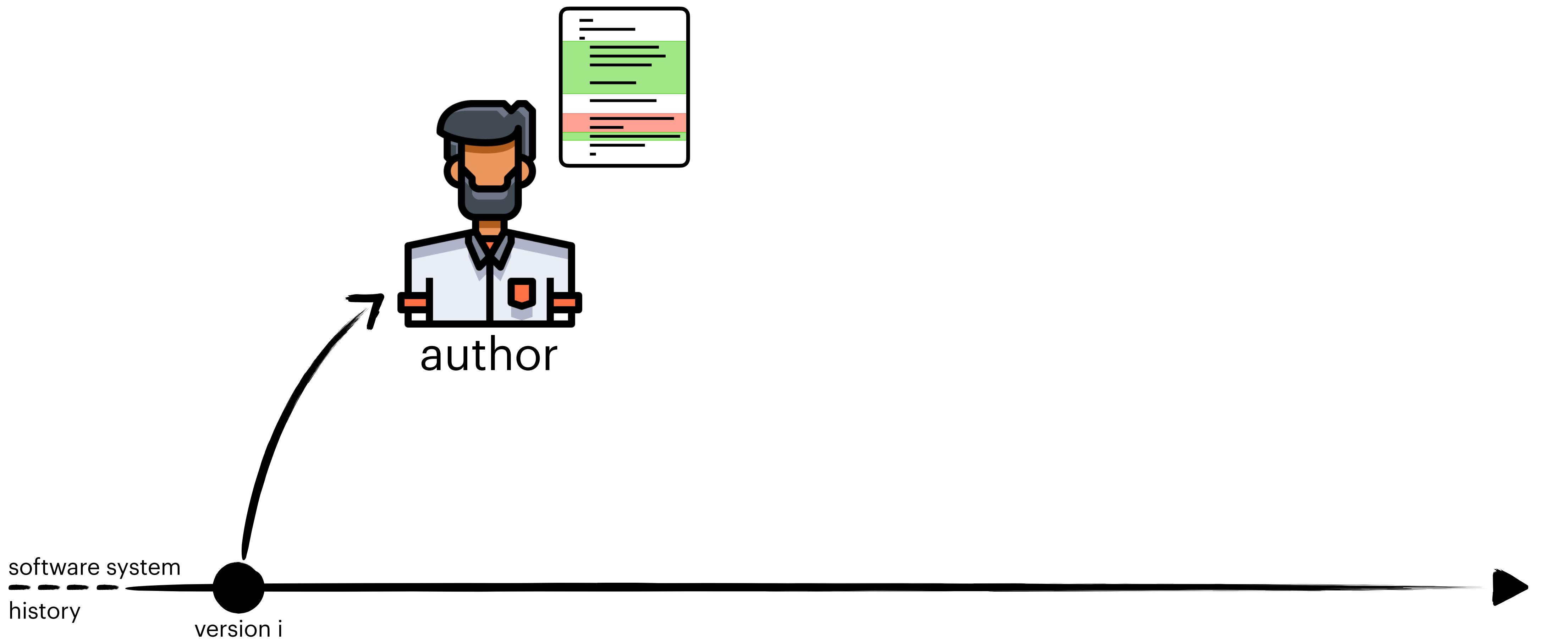
How code review works (and doesn't) in the real world.

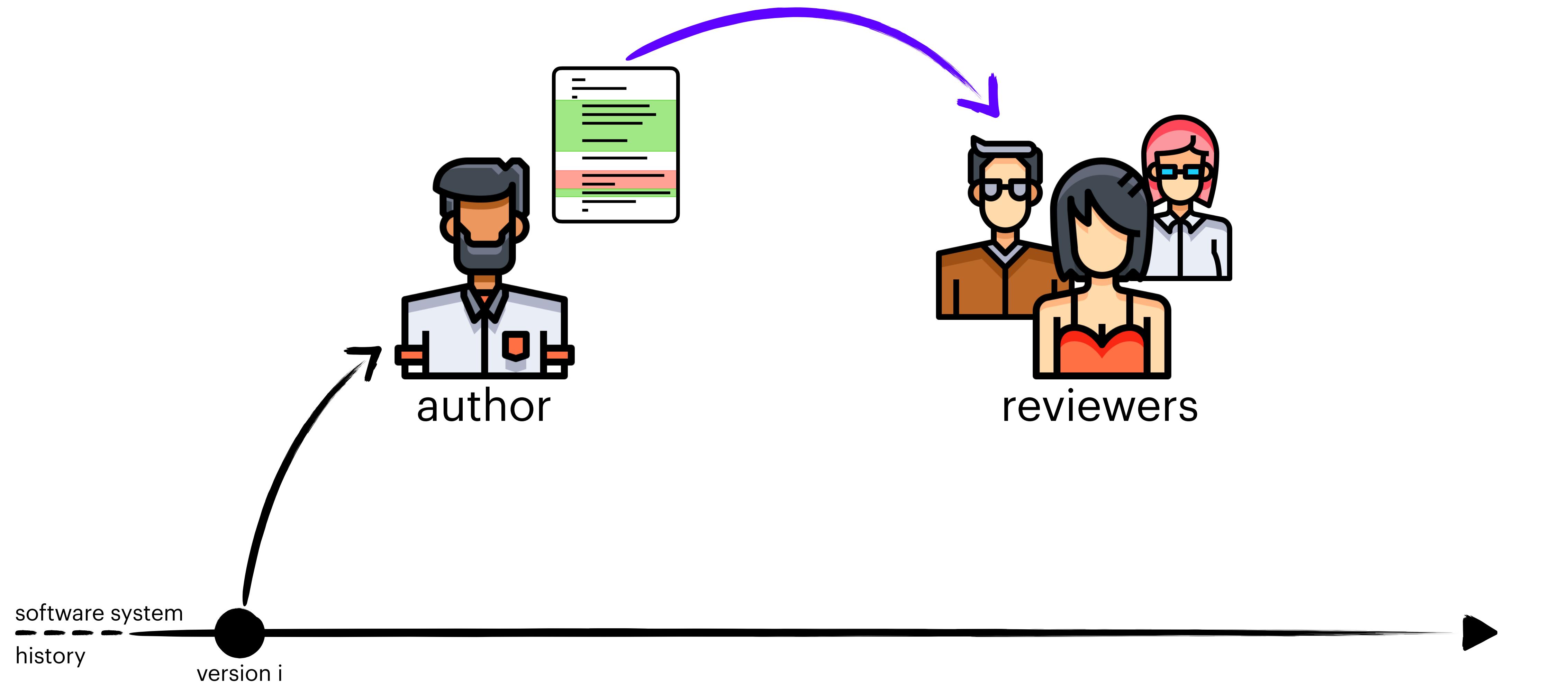
Alberto Bacchelli
zest

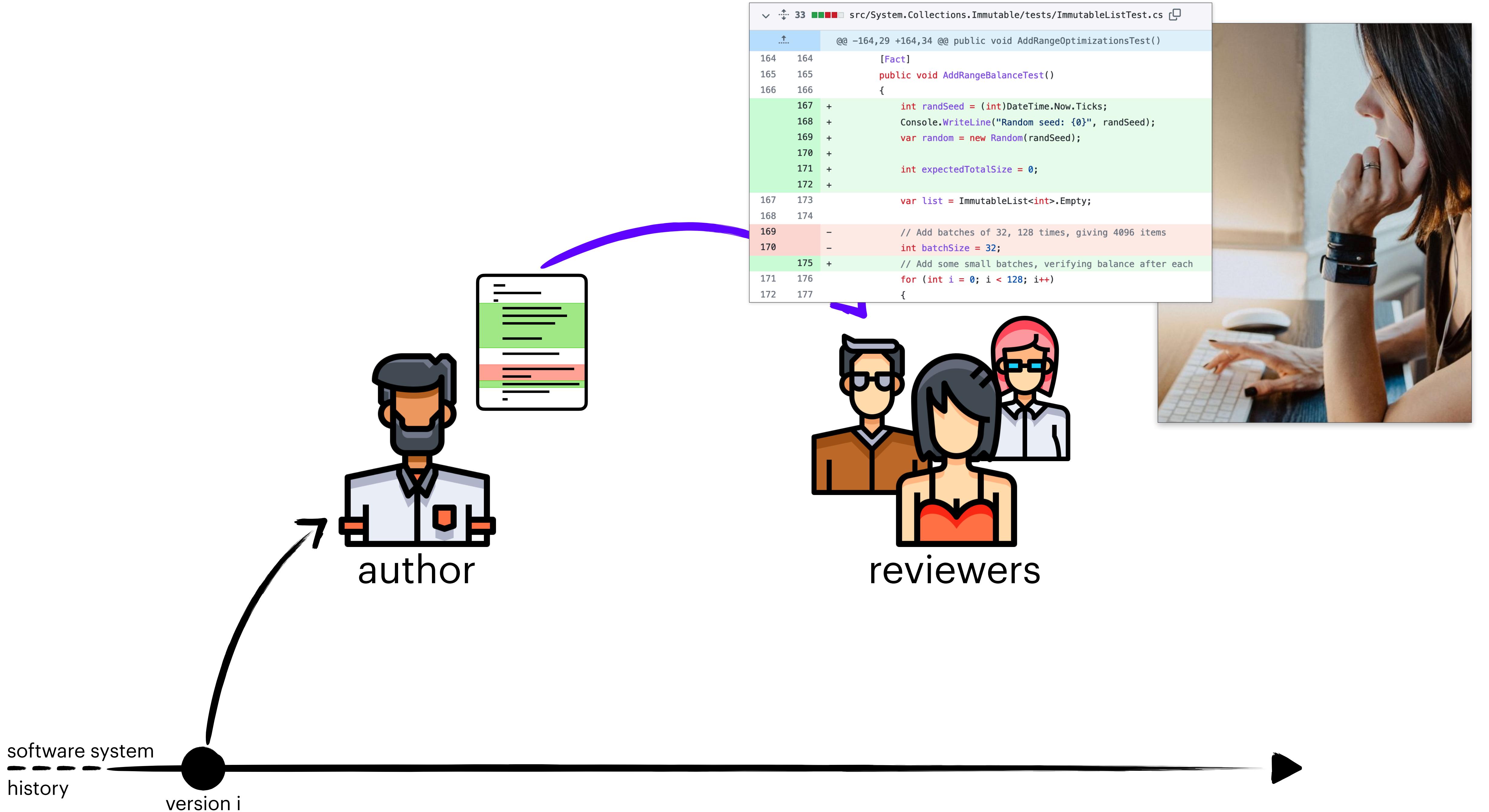


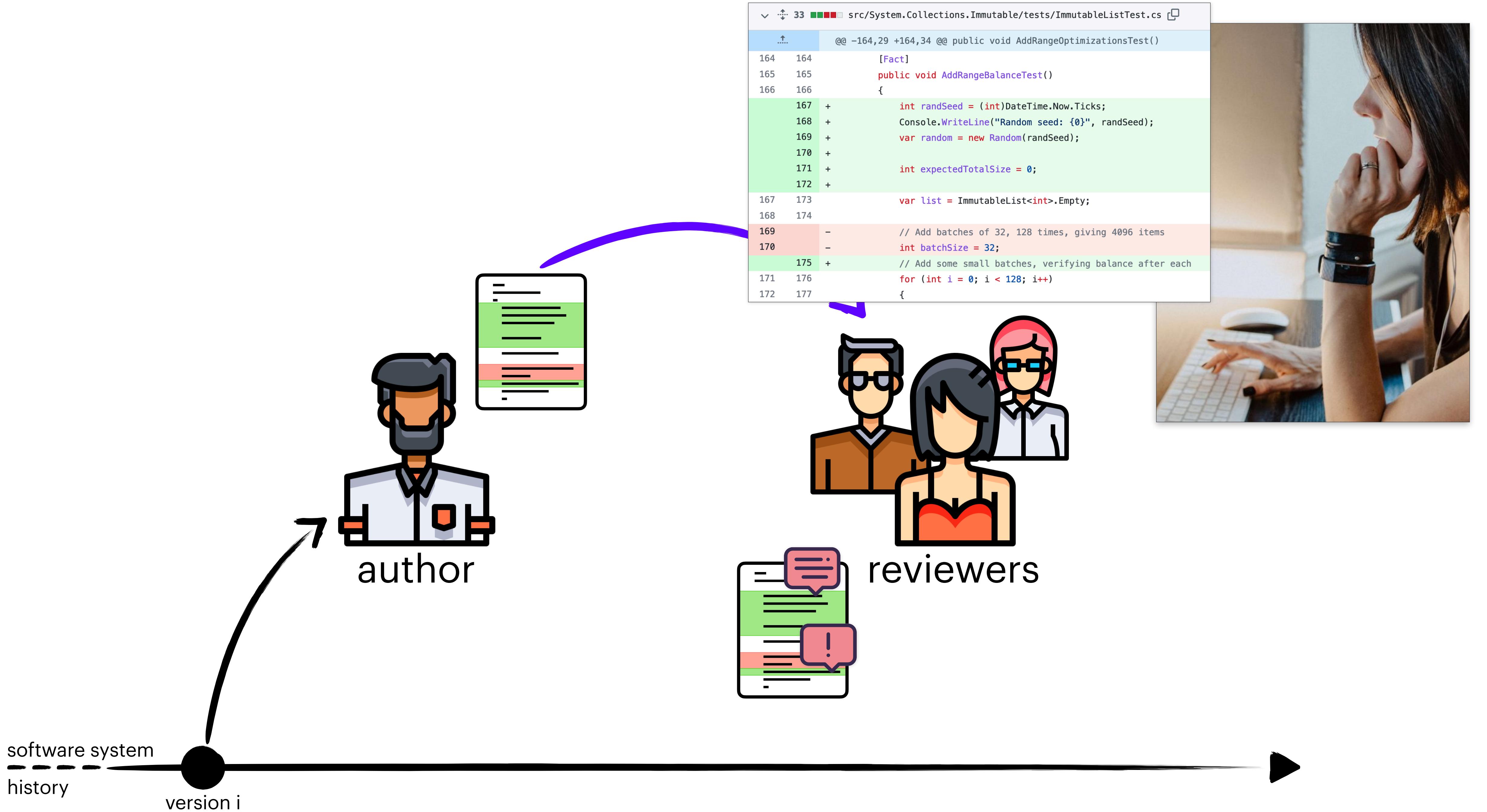
University of
Zurich^{UZH}

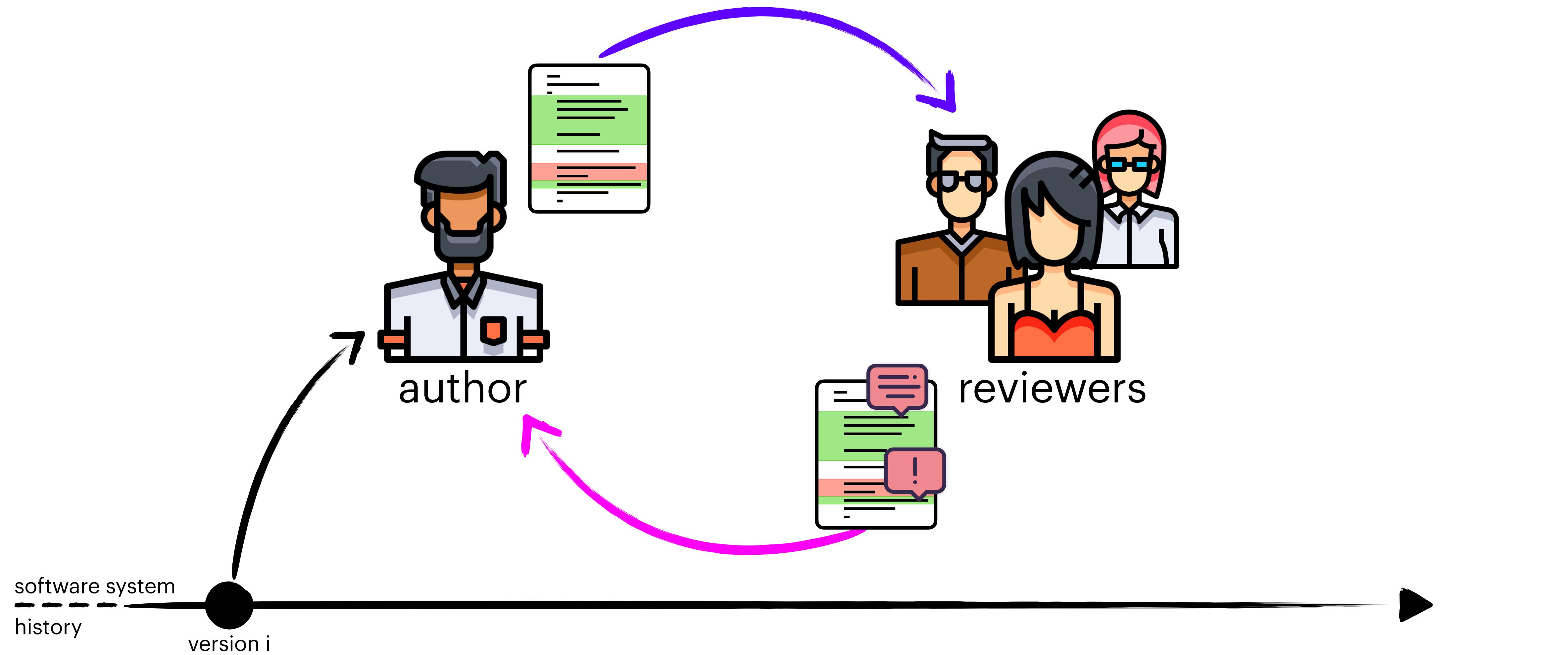


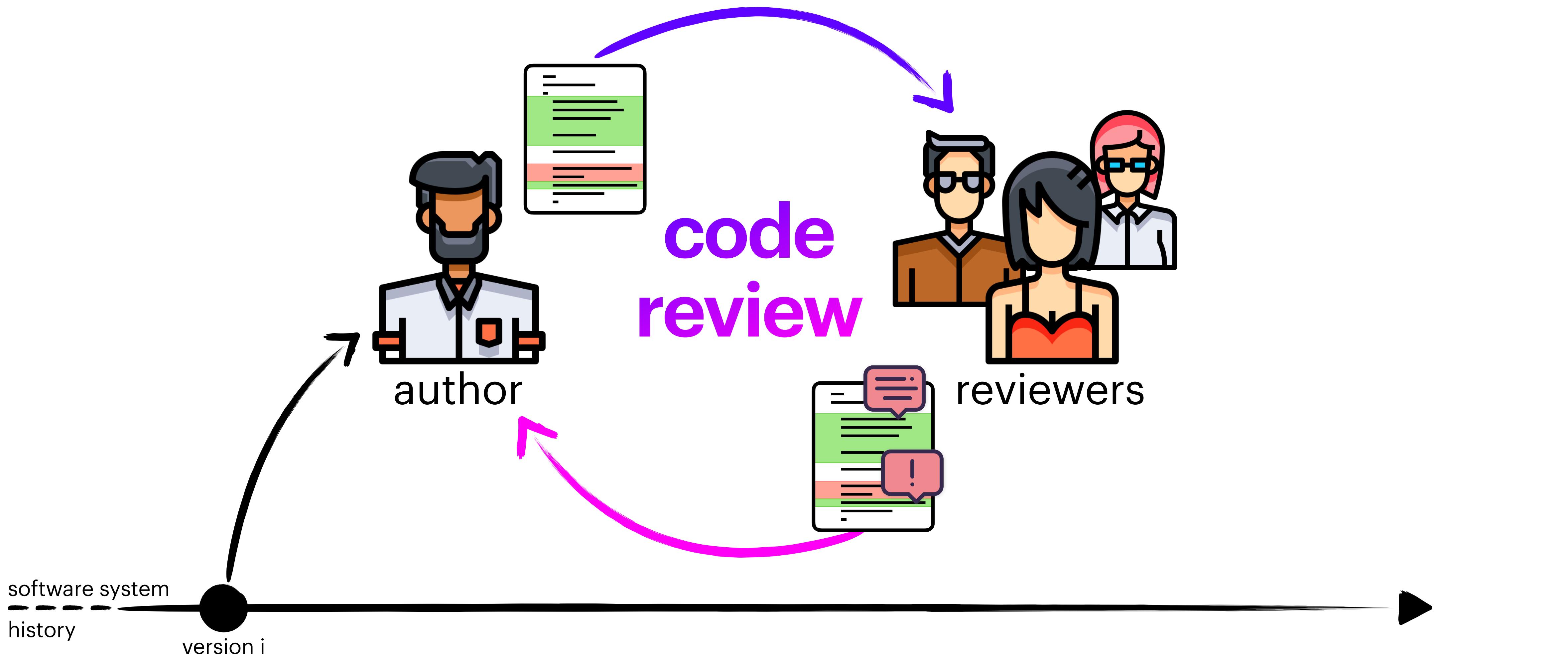


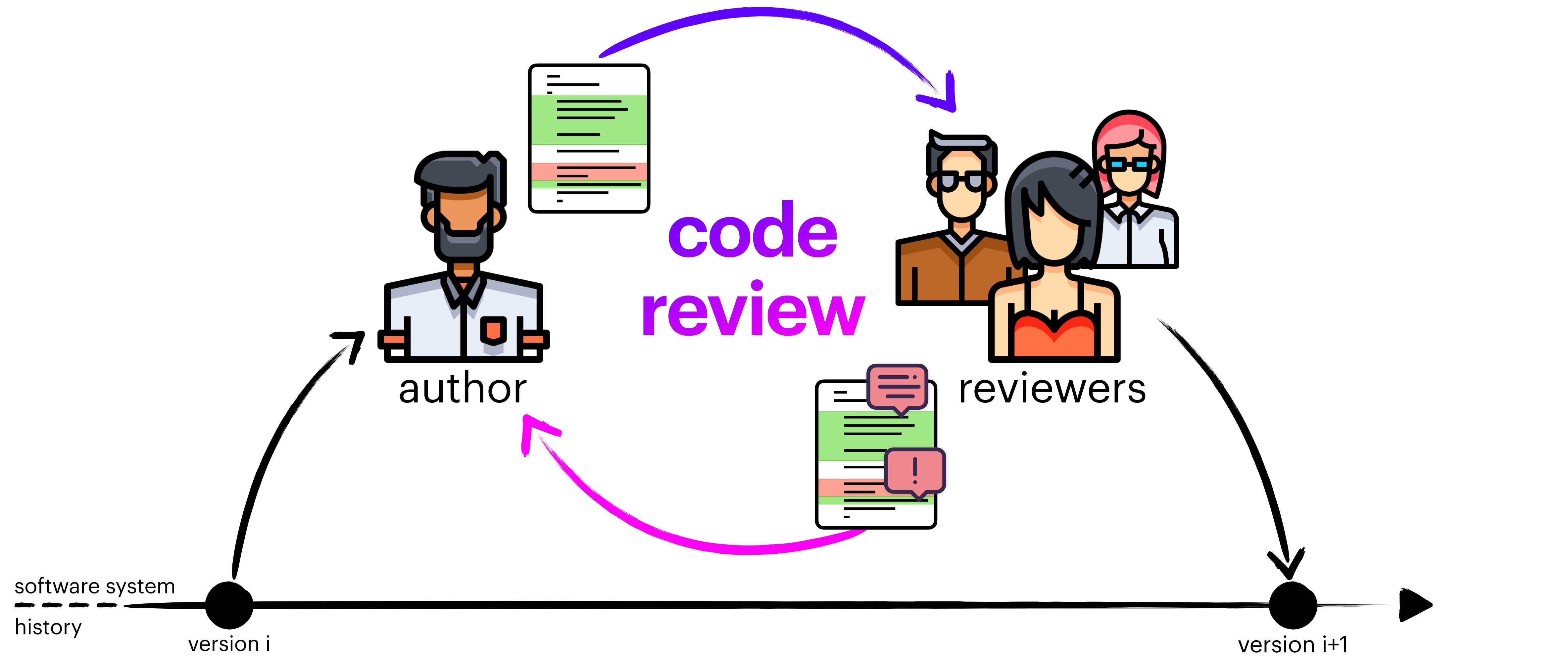


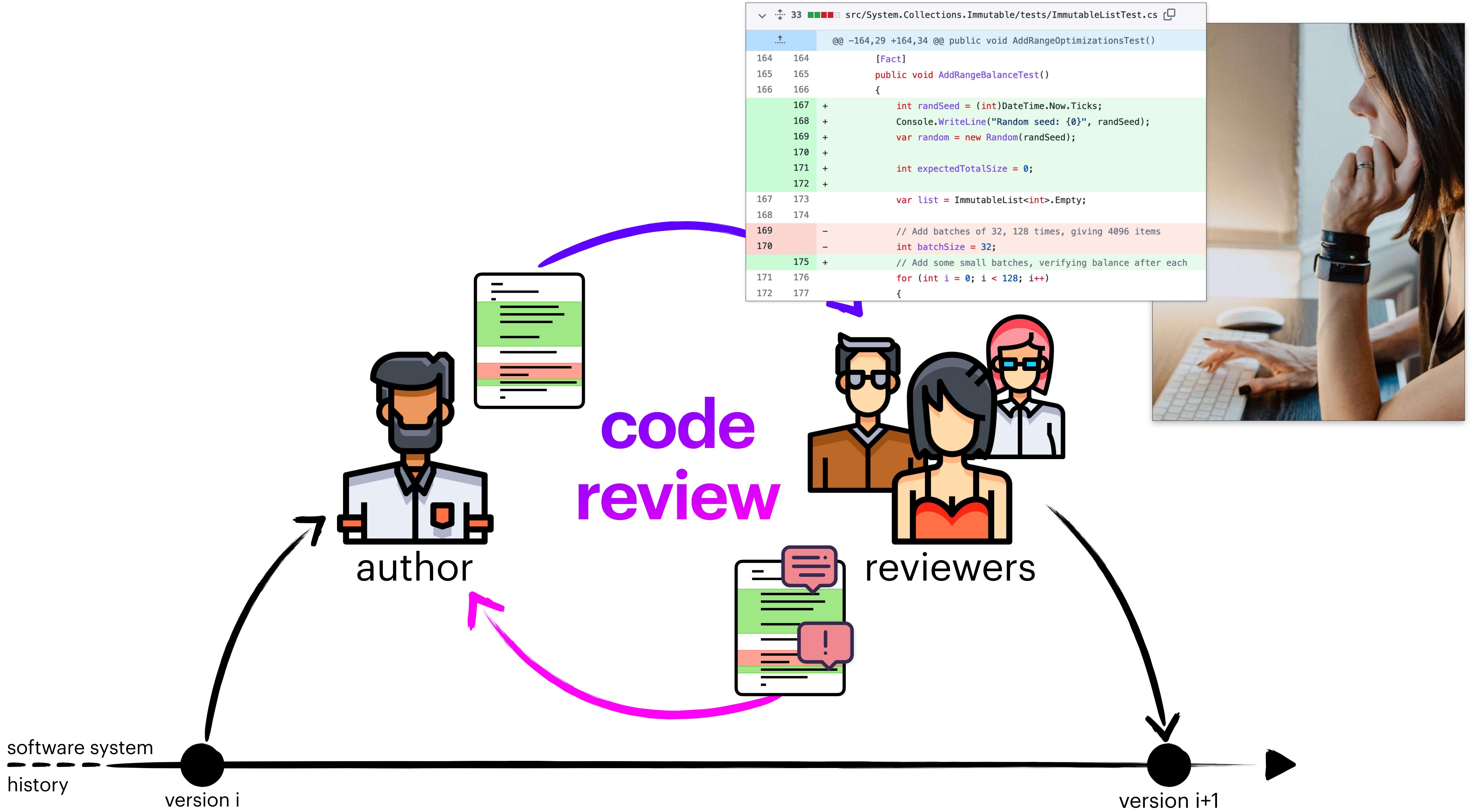












code review

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 two side-by-side code diff panels from GitHub's code review interface.

Top Panel: Diff for file `authentication.ts`. The changes are as follows:

```
@@ -7,7 +7,7 @@ const GITHUB_AUTH_PROVIDER_ID = 'github';
 7   7 // https://docs.github.com/apps/building-oauth-apps/understanding-scopes-for-oauth-apps
 8   8 const SCOPES = ['repo'];
 9   9
10  - /**
11  + /**
12  * Handles authentication to GitHub, using the VS Code [authentication API](https://code.visualstudio.com/api/references/authentication-api)
13  */
14  export class Credentials {
15  @@@ -18,6 +18,15 @@ export class Credentials {
16  18 // eslint-disable-next-line @typescript-eslint/no-empty-function
17  19 private constructor() {}
18  20
19  + /**
20  * Initializes an instance of credentials with an octokit instance.
21  */
22  + *
23  * Do not call this method until you know you actually need an instance of credentials.
24  * since calling this method will require the user to log in.
25  */
26  +
27  * @param context The extension context.
28  * @returns An instance of credentials.
29  */
30 static async initialize(context: vscode.ExtensionContext): Promise<Credentials> {
31   const c = new Credentials();
32   c.registerListeners(context);
33 }
```

Bottom Panel: Diff for file `query-history.ts`. The changes are as follows:

```
@@ -36,6 +36,8 @@ import { QueryStatus } from './query-status';
36   36 import { slurpQueryHistory, splatQueryHistory } from './query-serialization';
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-actions-api-client';
41   41 /**
42   * query-history.ts
43  */
44  @@ -316,7 +318,7 @@ export class QueryHistoryManager extends DisposableObject {
45  316   318   private qs: QueryServerClient,
46  317   319   private dbm: DatabaseManager,
47  318   320   private queryStorageDir: string,
48  319  -   ctx: ExtensionContext,
49  321  +   private ctx: ExtensionContext,
50  322   private queryHistoryConfigListener: QueryHistoryConfig,
51  323   private doCompareCallback: (
```

The screenshot shows the Crucible code review interface for a Java project.

Left Panel: A Java code diff for file `TrackedBranchesSearchCriteriaBuilder.java`. The changes are as follows:

```
@@ -32,36 +32,36 @@
32   }
33   }
34   }
35   public static TrackedBranchesSearchCriteria withReview(String reviewPermaId) {
36     return builder().reviewPermaId(reviewPermaId).build();
```

Right Panel: A comment thread between Piotr Swiecicki and Cezary Zawadka.

- Piotr Swiecicki: providing nullEmpty reviewPermaId results in criteria matching all reviews, I'd rather expect precondition failure in such circumstances
Add to favourites · 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 perma id means we should return empty list if there is no review with the perma id - 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 code review interface for a Python file.

Top Panel: Summary of changes for commit `+ Fix regressions from merges and unit test updates.` by Christian Hammond.

Bottom Panel: Detailed code diff for file `reviewboard/diffviewer/parser.py`. The changes are as follows:

```
@@ -20,266 +20,1304 @@
267 #: Version Added:
268 #: 4.0.6
269 #: 4.0.6
270 #: Type:
271 #: int
272 #: str
273 old_unix_mode = TypedProperty(six.text_type)
274 #: The new UNIX mode for the file.
275 #: Version Added:
276 #: 4.0.6
277 #: Type:
278 #: int
279 #: str
280 new_unix_mode = TypedProperty(six.text_type)
281 #: The parsed original name of the file.
282 #: Version Added:
283 #: 4.0.6
284 #: Deprecation:
285 #: 4.0.6
286 #: 4.0.6
287 #: 4.0.6
288 #: def parse_diff(self):
289 symlink_target = file_meta.get('symlink_target')
290 if isinstance(symlink_target, dict):
291   if isinstance(symlink_target, dict):
```

code review

The screenshot shows a GitHub code review interface for a pull request titled "Handle cancelling of remote queries". The top navigation bar includes links for Conversation (14), Commits (2), Checks (22), and Files changed (5). The main area displays two diff panels.

File: 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 - /**
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  21 + /**
22  22 * Initializes an instance of credentials with an octokit in
23  23 *
24  24 * Do not call this method until you know you actually need
25  25 * since calling this method will require the user to log in
26  26 *
27  27 * @param context The extension context.
28  28 * @returns An instance of credentials.
29  29 */
30  30 static async initialize(context: vscode.ExtensionContext): P
31  31     const c = new Credentials();
32  32     c.registerListeners(context);

```

File: 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  39 + import { Credentials } from './authentication';
40  40 + import { cancelRemoteQuery } from './remote-queries/gh-actions
```

code review

The screenshot shows a GitHub code review interface for a pull request titled "Handle cancelling of remote queries". The top navigation bar includes links for Conversation (14), Commits (2), Checks (22), and Files changed (5). The main area displays two diff panels:

- File: extensions/ql-vscode/src/authentication.ts**
This panel shows changes to the `authentication.ts` file. A new class `Credentials` is added, which has a private constructor and a static method `initialize` that creates an instance of `Credentials` and registers listeners. The code includes JSDoc comments explaining the purpose of the class and its methods.
- File: extensions/ql-vscode/src/query-history.ts**
This panel shows changes to the `query-history.ts` file. It imports `QueryStatus` from `./query-status` and adds imports for `slurpQueryHistory`, `splatQueryHistory` from `./query-history`, `fs-extra`, and `CliVersionConstraint` from `./cli`. It also adds imports for `Credentials` from `./authentication` and `cancelRemoteQuery` from `./remote-queries/gh-actions`.

The sidebar on the left shows the project structure with files like `authentication.ts`, `query-history.ts`, and `remote-queries`.

Beta Give feedback

code review

reviewers' behavior

The screenshot shows a GitHub code review interface for a pull request. At the top, there are tabs for Conversation (14), Commits (2), Checks (22), and Files changed (5). Below these are filters for Changes from all commits, File filter, and Conversations.

The main area displays a file diff for `authentication.ts`. The left sidebar shows the file structure:

- `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]

The right pane shows the diff for `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 - /**
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  21 + /**
22  22 * Initializes an instance of credentials with an octokit in
23  23 *
24  24 * Do not call this method until you know you actually need
25  25 * since calling this method will require the user to log in
26  26 *
27  27 * @param context The extension context.
28  28 * @returns An instance of credentials.
29  29 */
30  30 static async initialize(context: vscode.ExtensionContext): P
31  31     const c = new Credentials();
32  32     c.registerListeners(context);

```

A "Give feedback" button is located at the bottom left of the diff area. Below the main diff, another commit for `query-history.ts` is partially visible:

```
@@ -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  39 + import { Credentials } from './authentication';
40  40 + import { cancelRemoteQuery } from './remote-queries/gh-actions
```

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]

The screenshot shows a GitHub code review interface for a pull request titled "Handle cancelling of remote queries". The top navigation bar includes links for Conversation (14), Commits (2), Checks (22), and Files changed (5). The main area displays two files with their diff history:

- authentication.ts**: A TypeScript file with 11 changes. The diff shows the addition of `GITHUB_AUTH_PROVIDER_ID` and `SCOPES` constants, and the export of the `Credentials` class. The class has a private constructor and a static `initialize` method that creates an instance and registers listeners.
- query-history.ts**: A TypeScript file with 25 changes. The diff shows imports for `QueryStatus`, `slurpQueryHistory`, `splatQueryHistory`, `fs-extra`, `CliVersionConstraint`, `Credentials`, and `cancelRemoteQuery`.

code review

test files

The screenshot shows a GitHub code review interface for a pull request titled "Handle cancelling of remote queries". The top navigation bar includes links for Conversation (14), Commits (2), Checks (22), and Files changed (5). The main area displays two diff panels.

File 1: 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  21 + /**
22  22 * Initializes an instance of credentials with an octokit in
23  23 *
24  24 * Do not call this method until you know you actually need
25  25 * since calling this method will require the user to log in
26  26 *
27  27 * @param context The extension context.
28  28 * @returns An instance of credentials.
29  29 */
30  30 static async initialize(context: vscode.ExtensionContext): P
31  31     const c = new Credentials();
32  32     c.registerListeners(context);
33  33 }
```

File 2: 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  39 + import { Credentials } from './authentication';
40  40 + import { cancelRemoteQuery } from './remote-queries/gh-actions'
```

code review

test files

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

The screenshot shows a GitHub pull request interface. At the top, there are tabs for Conversation (14), Commits (2), Checks (22), and Files changed (5). Below this, a sidebar lists the files affected by the pull request, including `authentication.ts`, `query-history.ts`, and several test files like `gh-actions-api-client.ts`. The main area displays the diff for `authentication.ts`. The diff shows changes from line 7 to 23. Lines 10 and 11 are highlighted in red, indicating they have been deleted. Lines 12 and 13 are highlighted in green, indicating they have been added. The code snippet includes imports for `GITHUB_AUTH_PROVIDER_ID` and `SCOPES`, and a class definition for `Credentials` with a constructor and a static method `initialize`.

```
@@ -7,7 +7,7 @@ const GITHUB_AUTH_PROVIDER_ID = 'github';
// https://docs.github.com/apps/building-oauth-apps/understanding OAuth scopes
const SCOPES = ['repo'];

- /**
+ */

 * Handles authentication to GitHub, using the VS Code [authen
+
* @param context The extension context.
* @returns An instance of credentials.
*/
export class Credentials {

@@ -18,6 +18,15 @@ export class Credentials {
// eslint-disable-next-line @typescript-eslint/no-empty-func
private constructor() { }

+ /**
+ * Initializes an instance of credentials with an octokit instance.
+ *
+ * Do not call this method until you know you actually need it,
+ * since calling this method will require the user to log in.
+ */
+ * @param context The extension context.
+ * @returns An instance of credentials.
*/
static async initialize(context: vscode.ExtensionContext): Promise<Credentials> {
const c = new Credentials();
c.registerListeners(context);
}

@@ -36,6 +36,8 @@ import { QueryStatus } from './query-status'
import { slurpQueryHistory, splatQueryHistory } from './query-hist
import * as fs from 'fs-extra';
import { CliVersionConstraint } from './cli';

+ import { Credentials } from './authentication';
+ import { cancelRemoteQuery } from './remote-queries/gh-actions

```

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 shows a GitHub pull request interface. At the top, there are tabs for Conversation (14), Commits (2), Checks (22), and Files changed (5). Below this, a sidebar lists 'Changes from all commits' and 'File filter'. The main area displays two files with diff highlights:

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 - /**
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  21 + /**
22  22 * Initializes an instance of credentials with an octokit in
23  23 *
24  24 * Do not call this method until you know you actually need
25  25 * since calling this method will require the user to log in
26  26 *
27  27 * @param context The extension context.
28  28 * @returns An instance of credentials.
29  29 */
30  30 static async initialize(context: vscode.ExtensionContext): P
31  31     const c = new Credentials();
32  32     c.registerListeners(context);
33  33 }
```

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  39 + import { Credentials } from './authentication';
40  40 + import { cancelRemoteQuery } from './remote-queries/gh-actions'
```

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 GitHub pull request interface. At the top, there are tabs for Conversation (14), Commits (2), Checks (22), and Files changed (5). Below this, a sidebar lists 'Changes from all commits' and 'File filter'. The main area displays two files with diff highlights:

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 - /**
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  21 + /**
22  22 * Initializes an instance of credentials with an octokit in
23  23 *
24  24 * Do not call this method until you know you actually need
25  25 * since calling this method will require the user to log in
26  26 *
27  27 * @param context The extension context.
28  28 * @returns An instance of credentials.
29  29 */
30  30 static async initialize(context: vscode.ExtensionContext): P
31  31     const c = new Credentials();
32  32     c.registerListeners(context);
33  33 }
```

query-history.ts

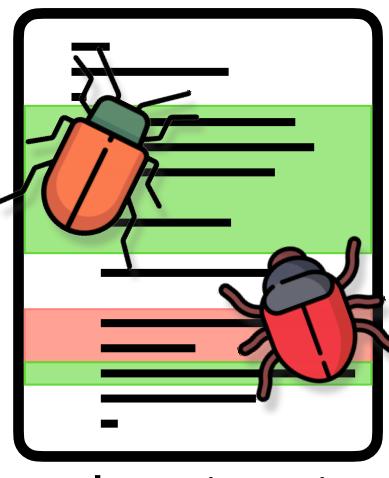
```
@@ -36,6 +36,8 @@ import { QueryStatus } from './query-status'
36  36 import { slurpQueryHistory, splatQueryHistory } from './query-
37  37 import * as fs from 'fs-extra';
38  38 import { CliVersionConstraint } from './cli';
39  39 + import { Credentials } from './authentication';
40  40 + import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

code review

test order experiment

code review

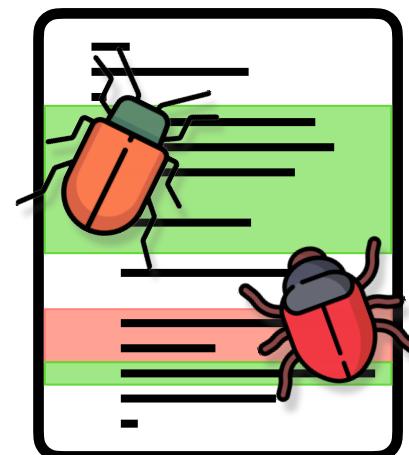
test order experiment



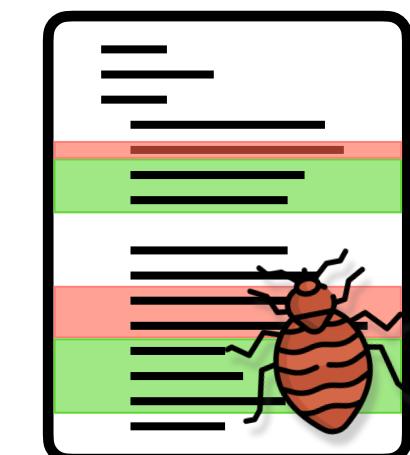
production.java

code review

test order experiment



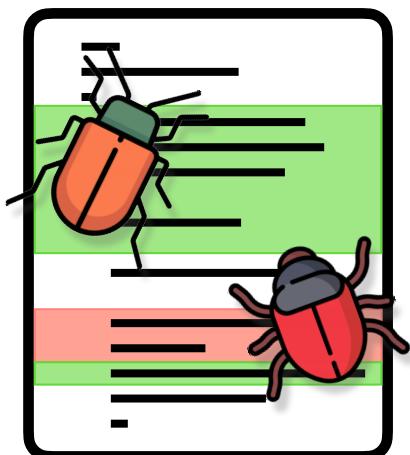
production.java



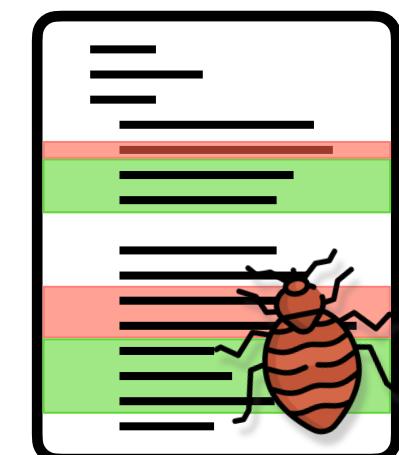
test.java

code review

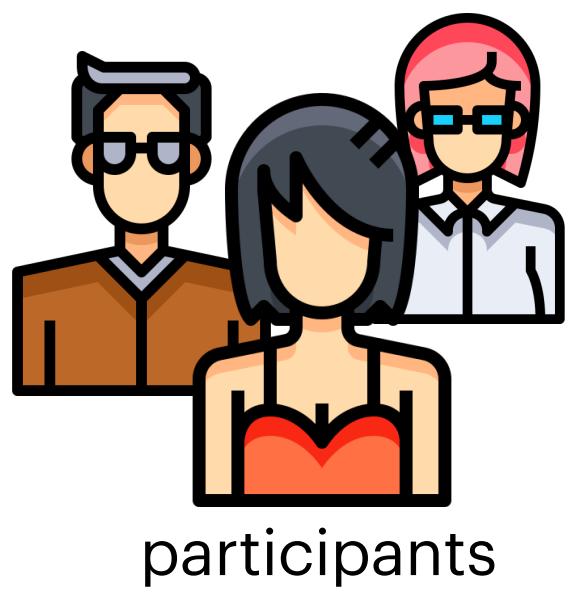
test order experiment



production.java



test.java



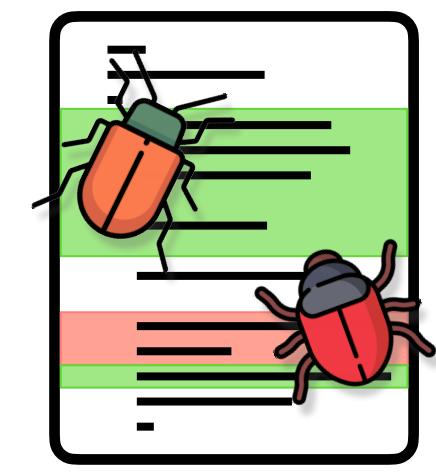
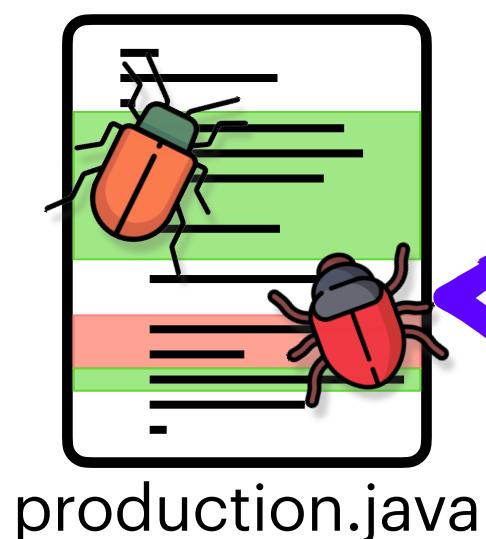
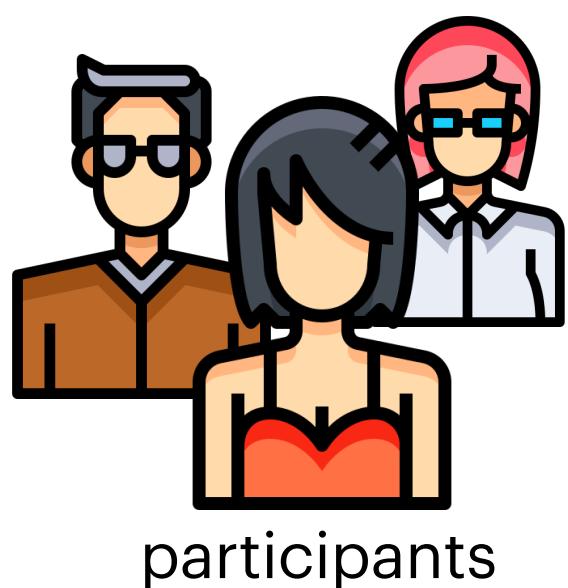
participants



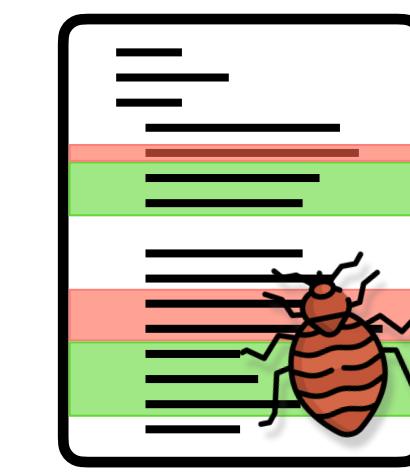
participants

code review

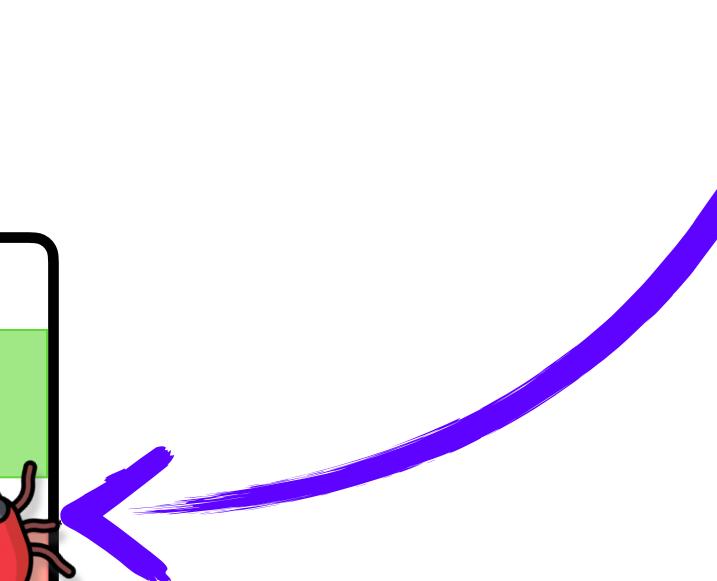
test order experiment



production.java

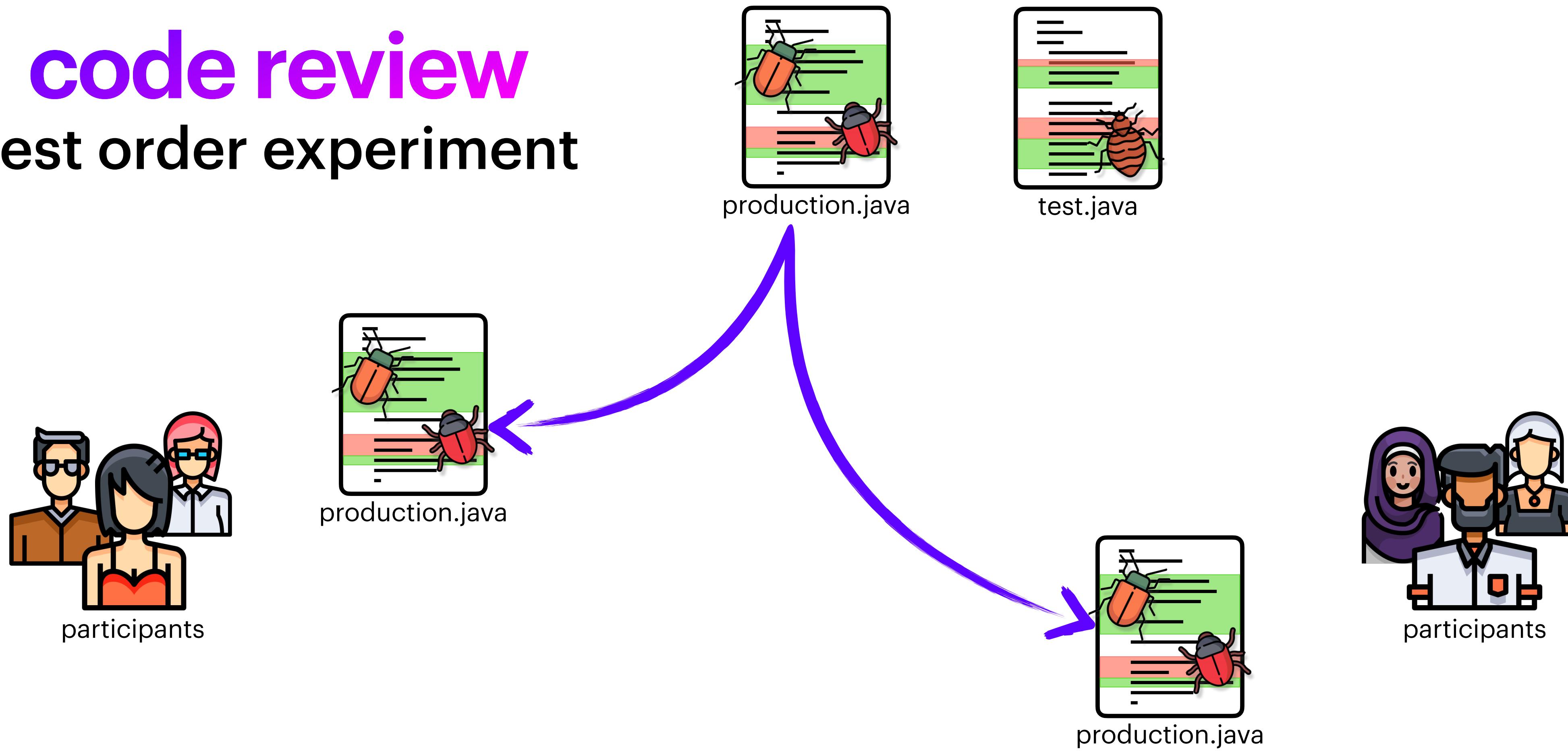


test.java



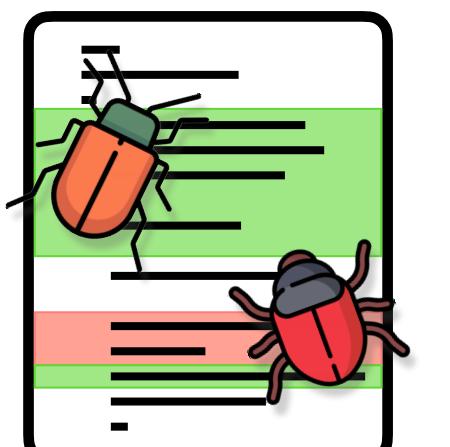
code review

test order experiment

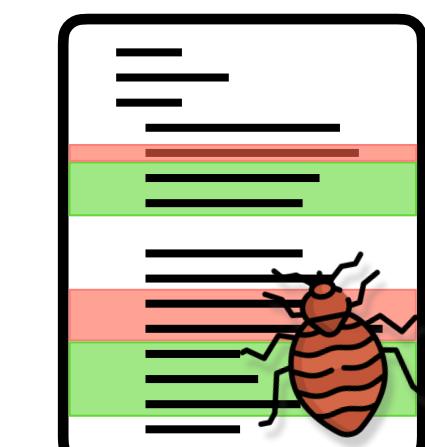


code review

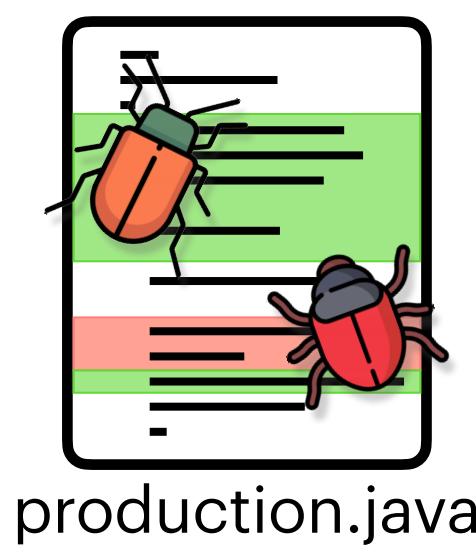
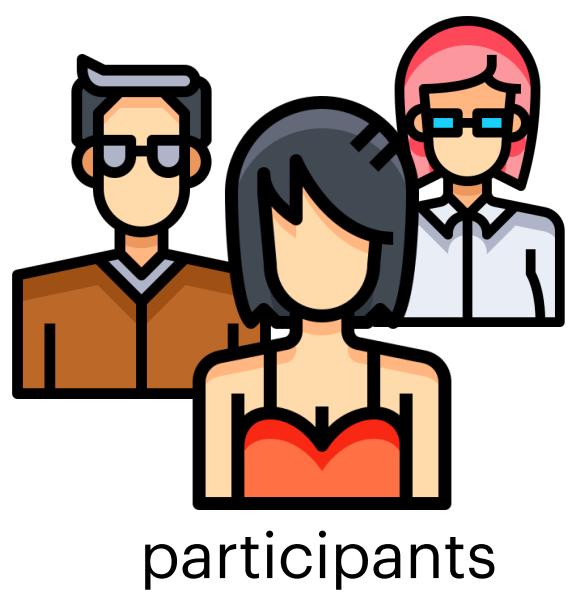
test order experiment



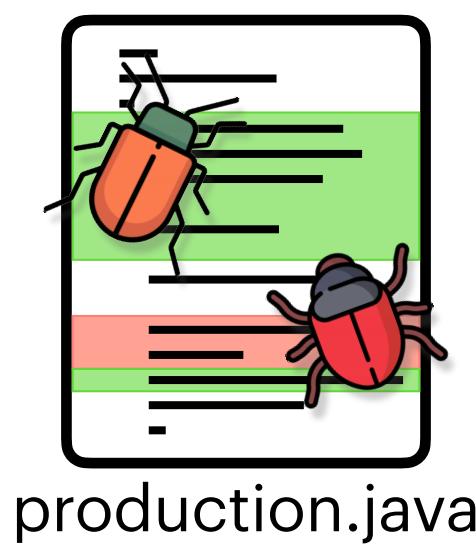
production.java



test.java



production.java



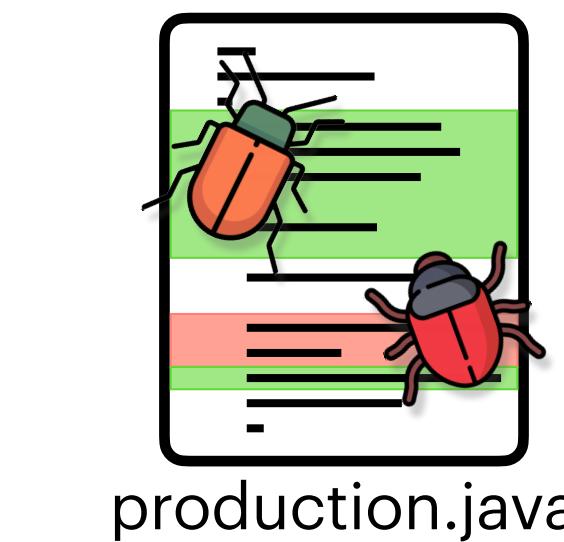
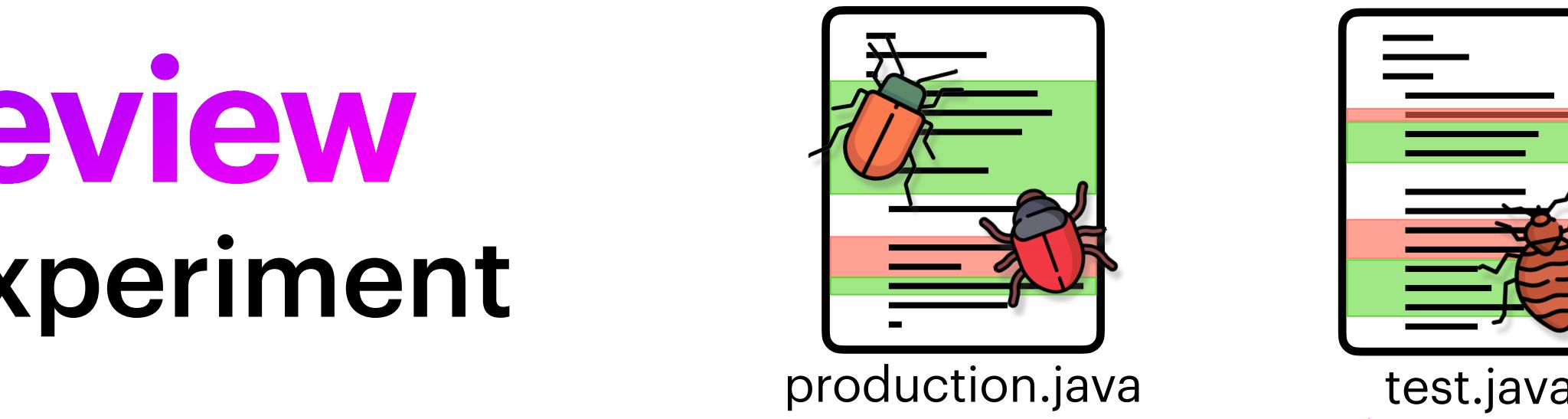
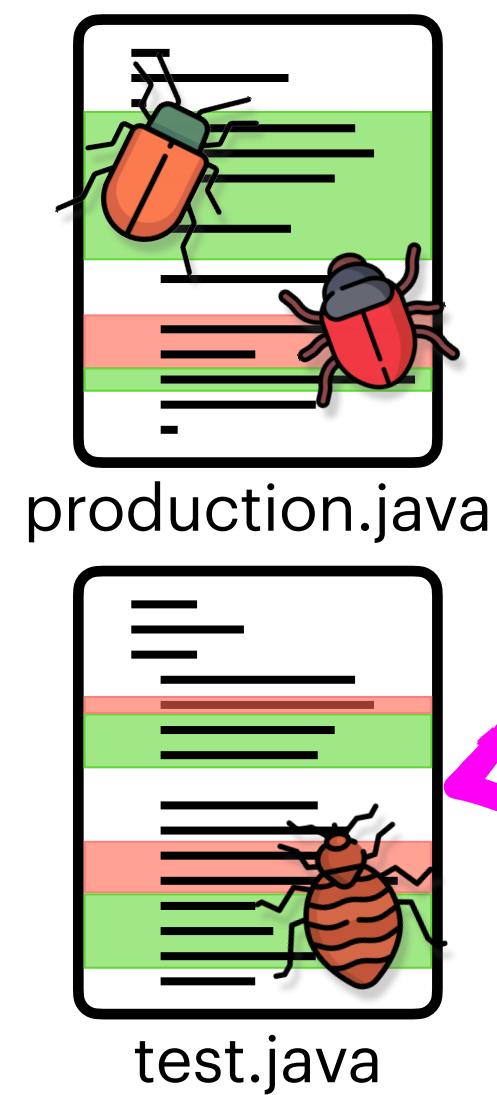
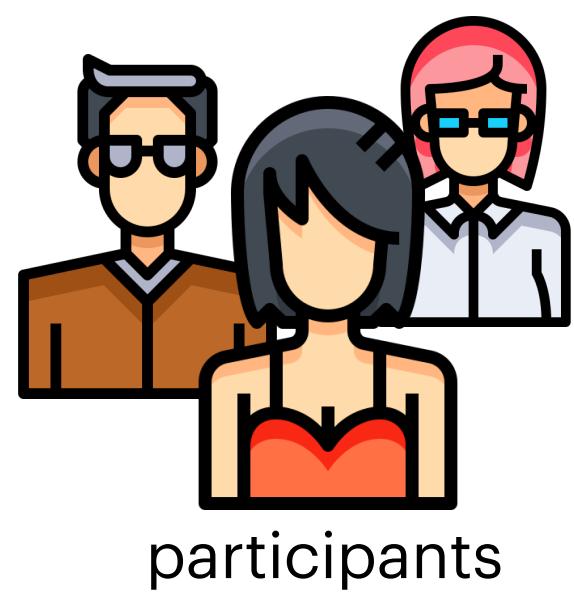
production.java



participants

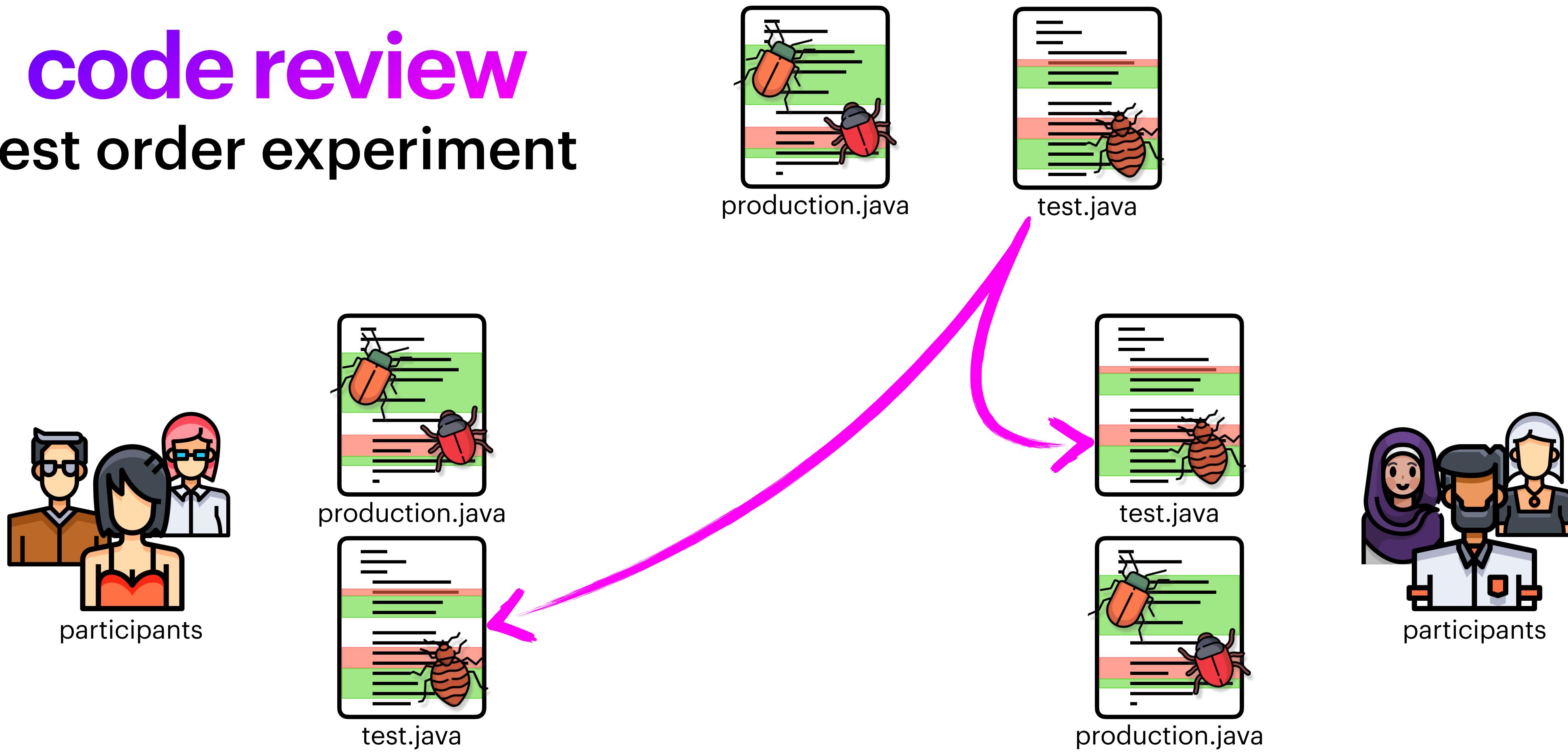
code review

test order experiment



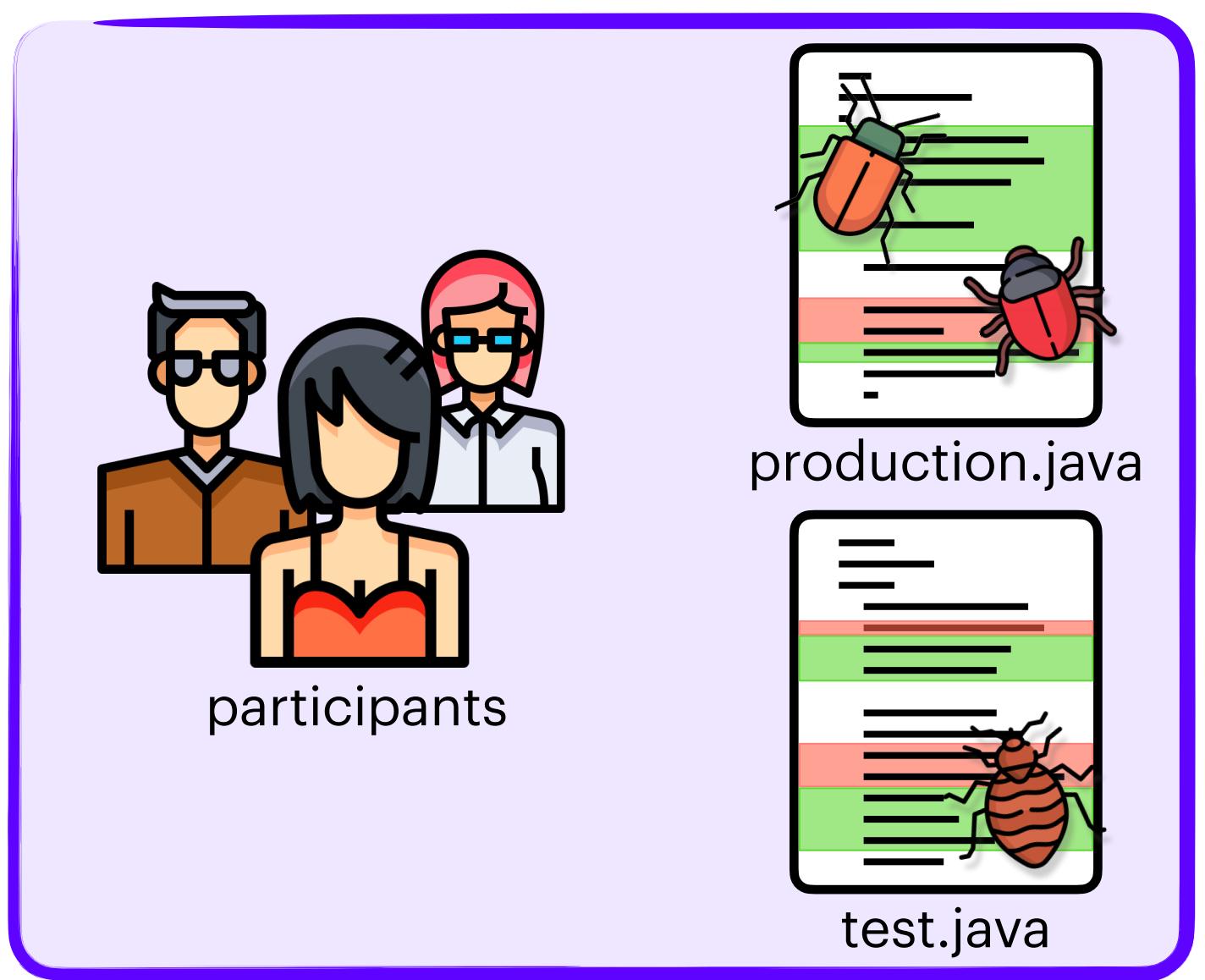
code review

test order experiment

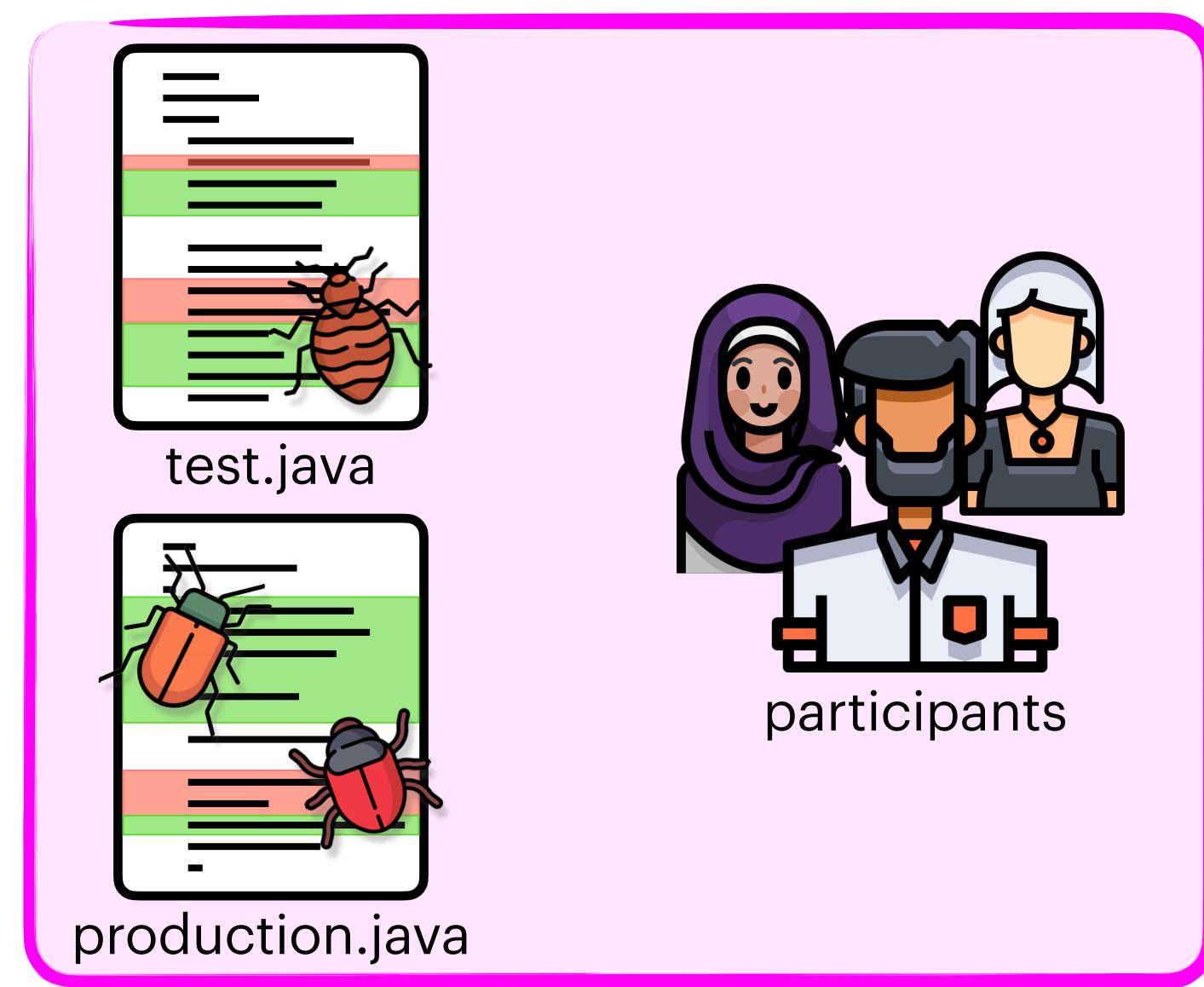


code review

test order experiment



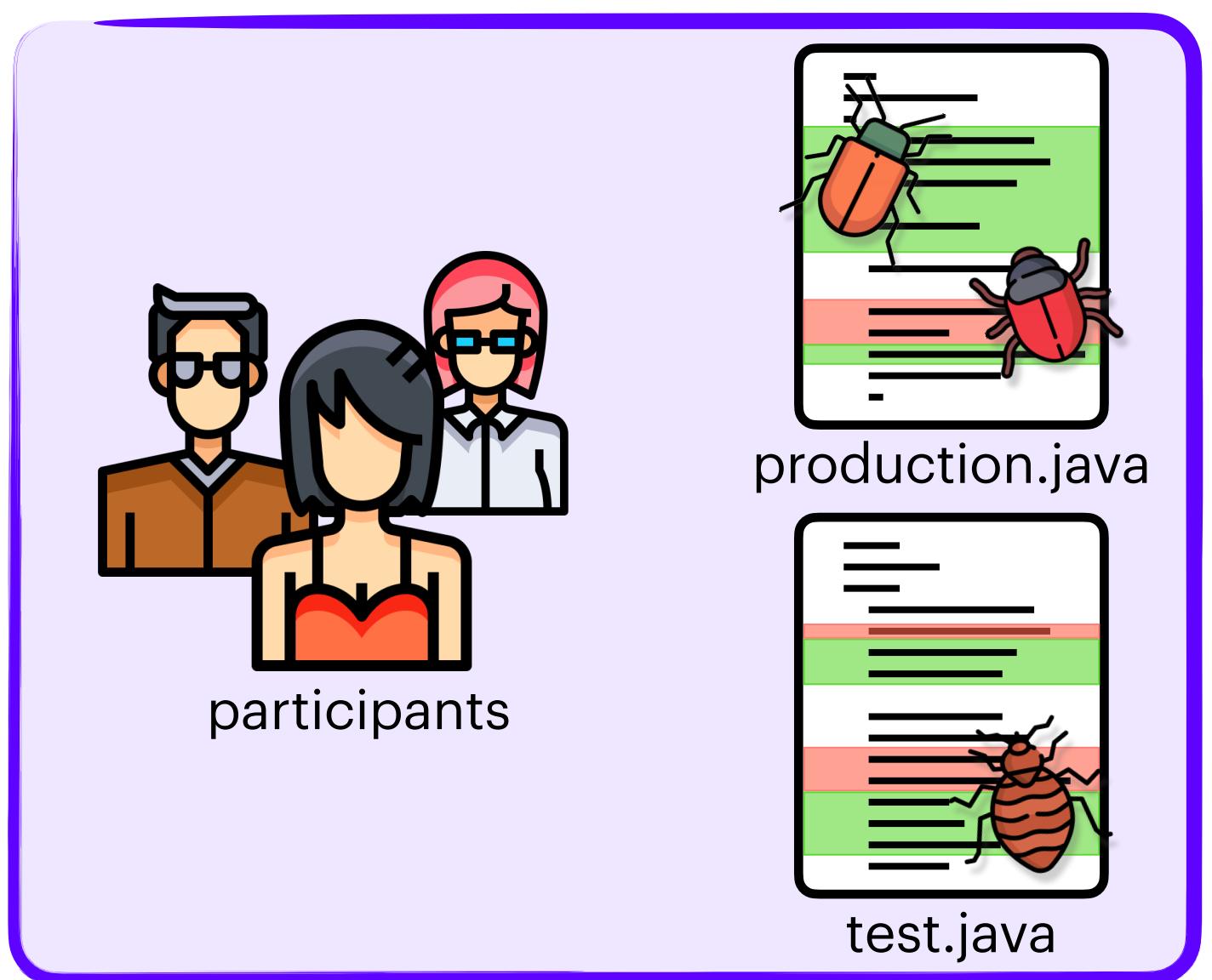
**production code first
order**



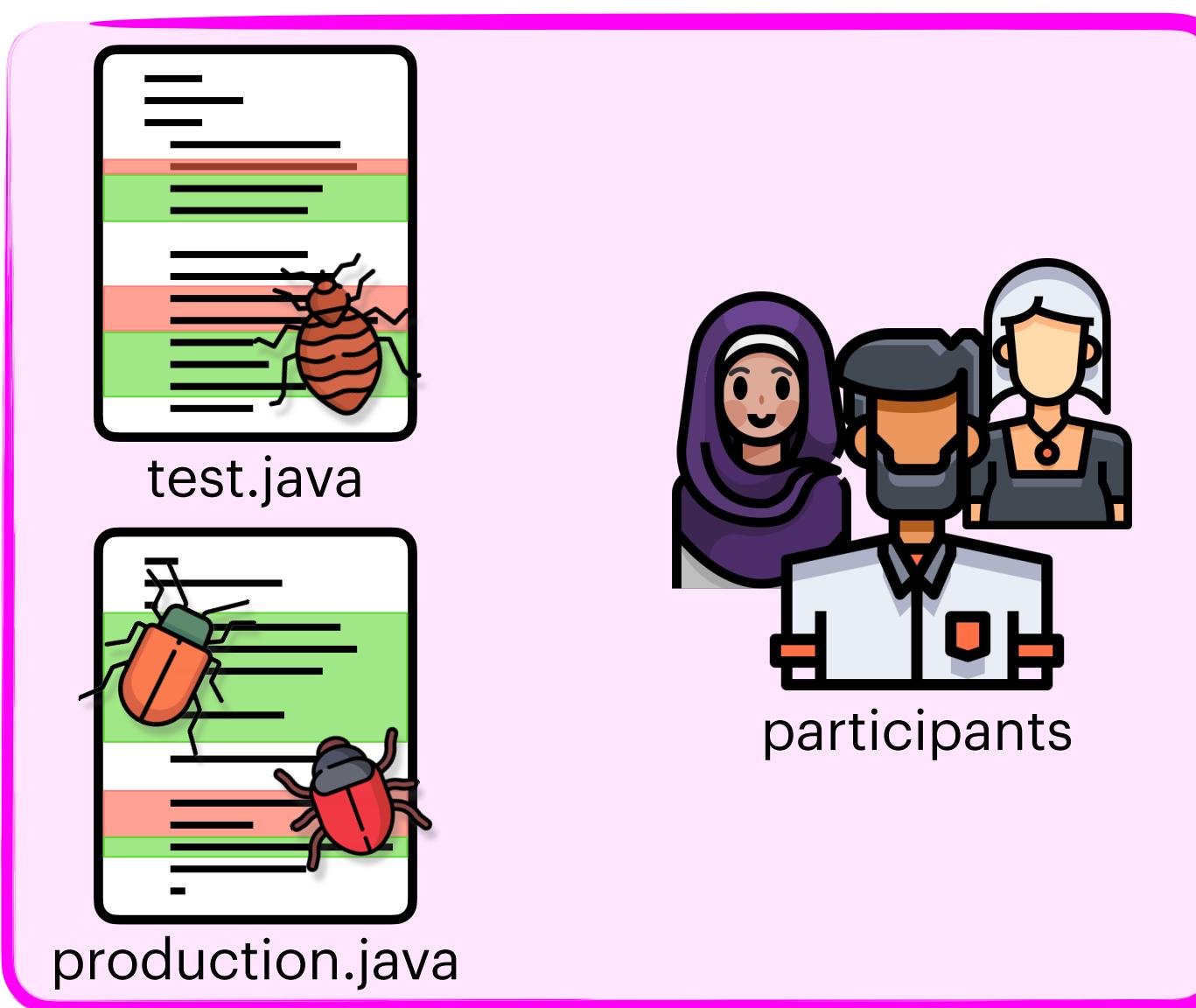
**test code first
order**

code review

test order experiment



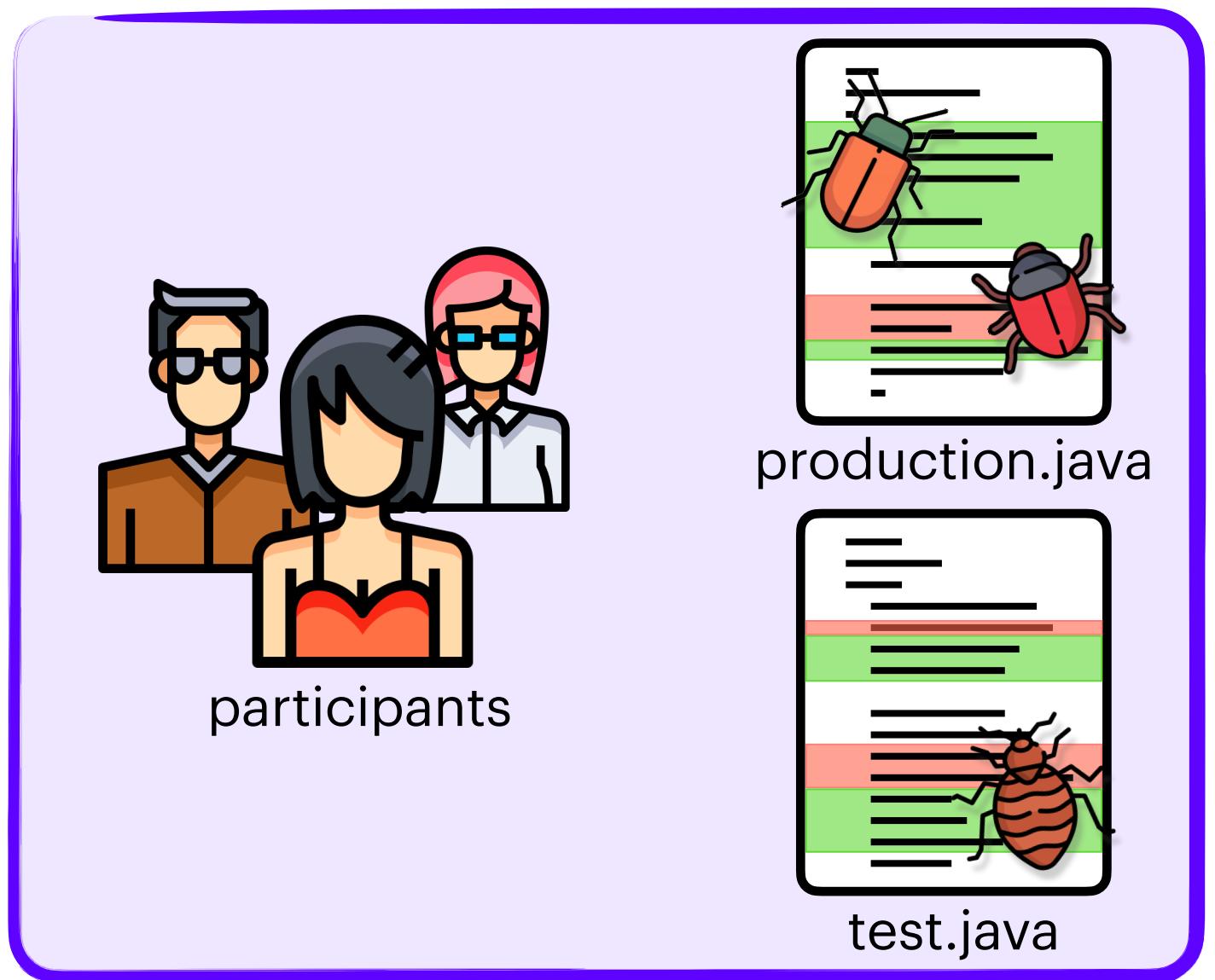
same likelihood
of finding the
production bugs



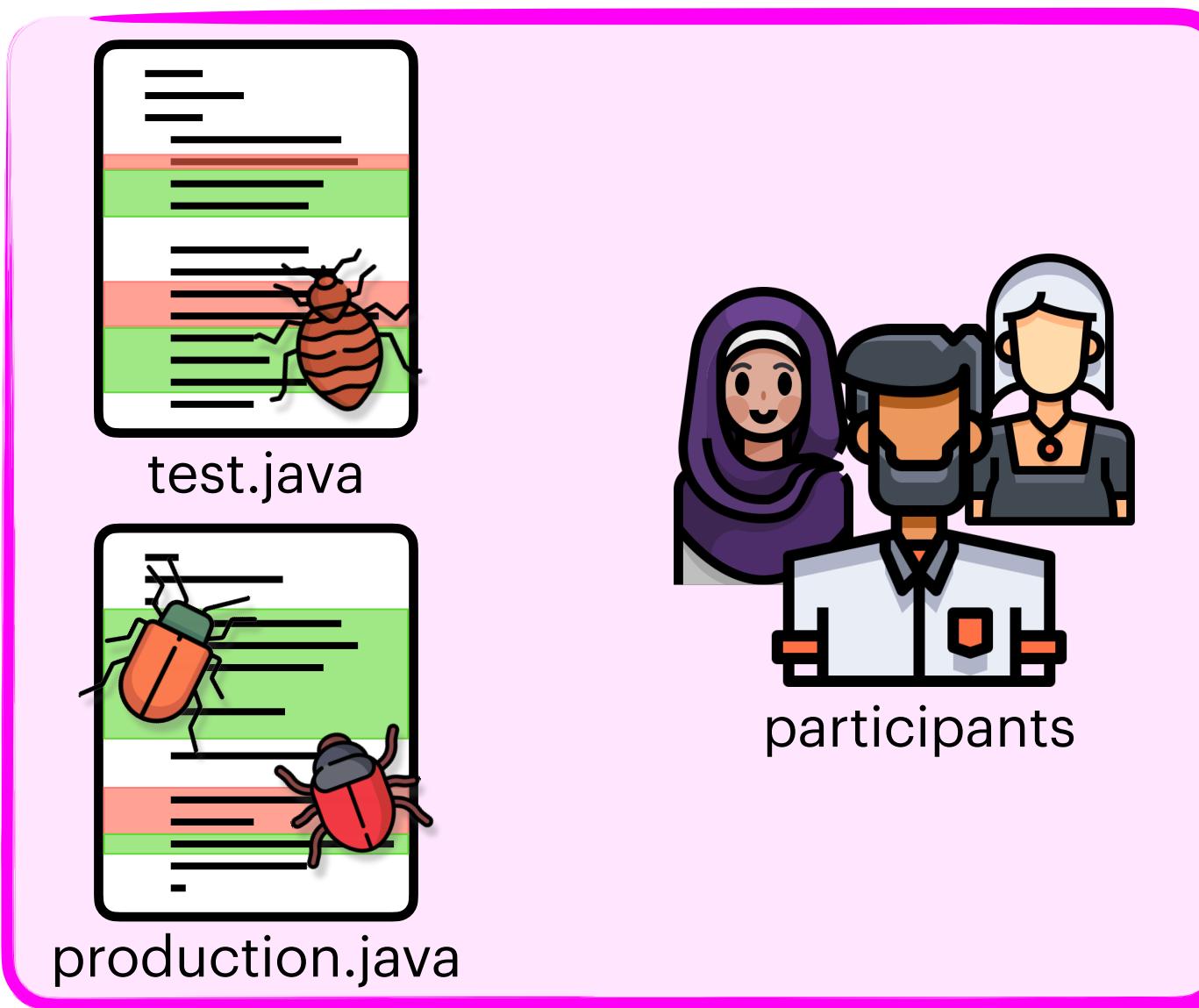
code review

test order experiment

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



**same likelihood
of finding the
production bugs**



code review

test files

The screenshot shows a GitHub code review interface for a pull request titled "Handle cancelling of remote queries". The top navigation bar includes links for Conversation (14), Commits (2), Checks (22), and Files changed (5). The main area displays two diff panels.

File 1: 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  21 + /**
22  22 * Initializes an instance of credentials with an octokit in
23  23 *
24  24 * Do not call this method until you know you actually need
25  25 * since calling this method will require the user to log in
26  26 *
27  27 * @param context The extension context.
28  28 * @returns An instance of credentials.
29  29 */
30  30 static async initialize(context: vscode.ExtensionContext): P
31  31     const c = new Credentials();
32  32     c.registerListeners(context);
33  33 }
```

File 2: 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  39 + import { Credentials } from './authentication';
40  40 + import { cancelRemoteQuery } from './remote-queries/gh-actions'
```

code review

production files

The screenshot shows a GitHub pull request interface with the following details:

- Title:** Handle cancelling of remote queries
- Conversation:** 14
- Commits:** 2
- Checks:** 22
- Files changed:** 5
- Changes from all commits:** 0 / 5 files
- File filter:** Conversations
- Filter changed files:** authentication.ts, query-history.ts

The main pane displays the diff for `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 - /**
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  21 + /**
22  22 * Initializes an instance of credentials with an octokit in
23  23 *
24  24 * Do not call this method until you know you actually need
25  25 * since calling this method will require the user to log in
26  26 *
27  27 * @param context The extension context.
28  28 * @returns An instance of credentials.
29  29 */
30  30 static async initialize(context: vscode.ExtensionContext): P
31  31     const c = new Credentials();
32  32     c.registerListeners(context);

```

The sidebar shows the file structure and other 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

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

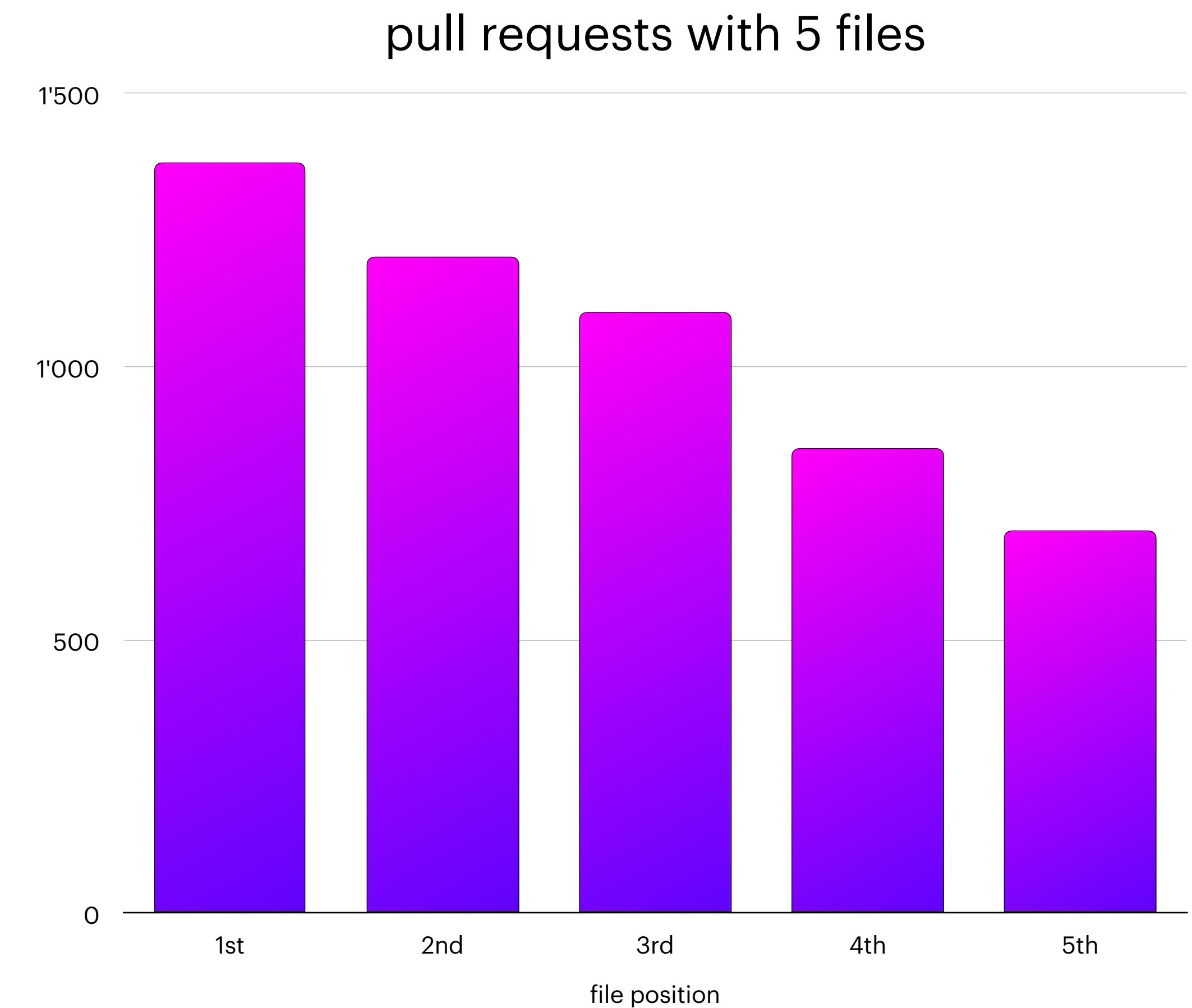
cumulative number of
review comments
by file position

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

code review production files

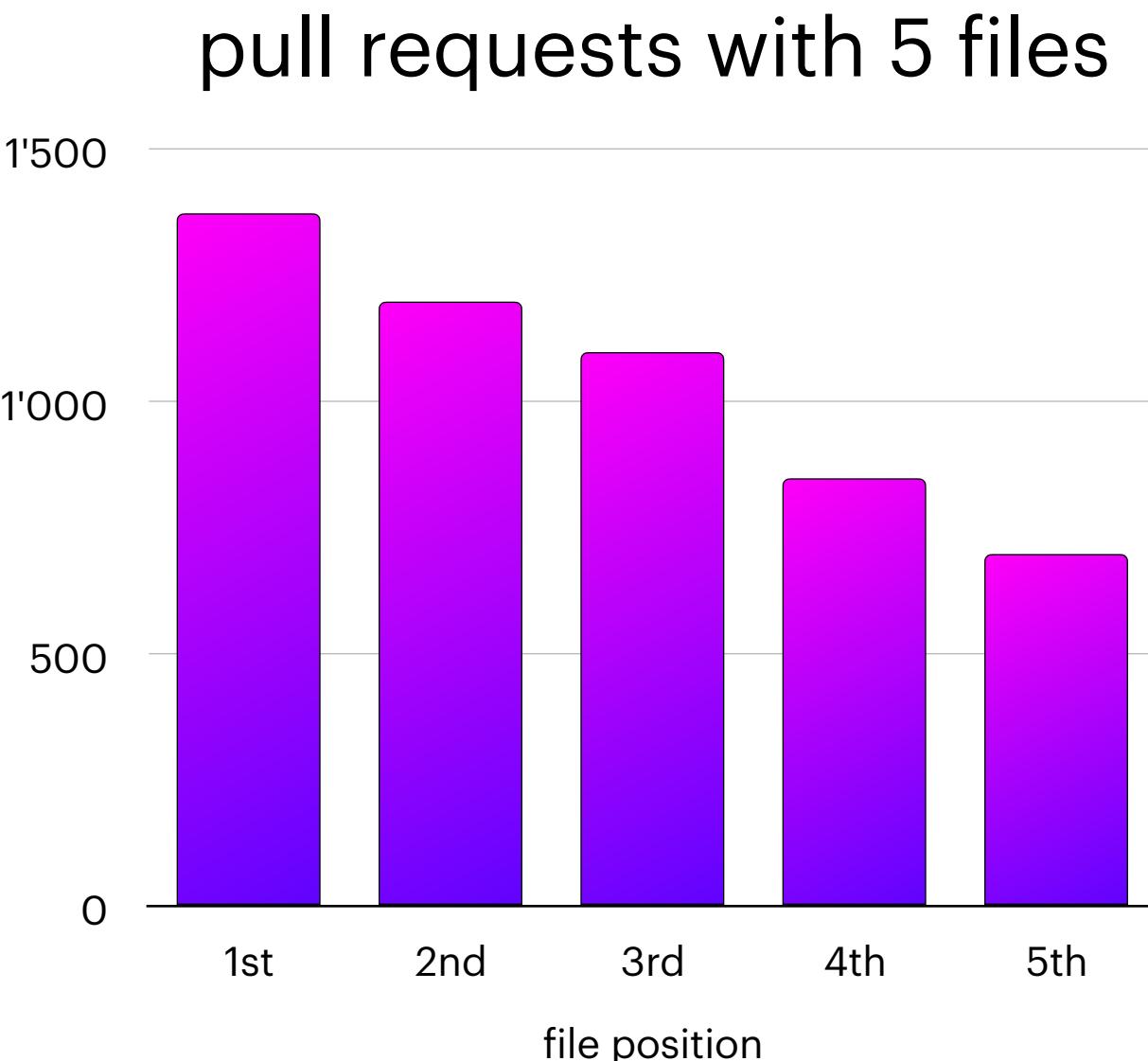
- We analyzed review comments for 200k pull requests from some 140 popular projects in GitHub...
- ... and found something remarkable.

cumulative number of
review comments
by file position

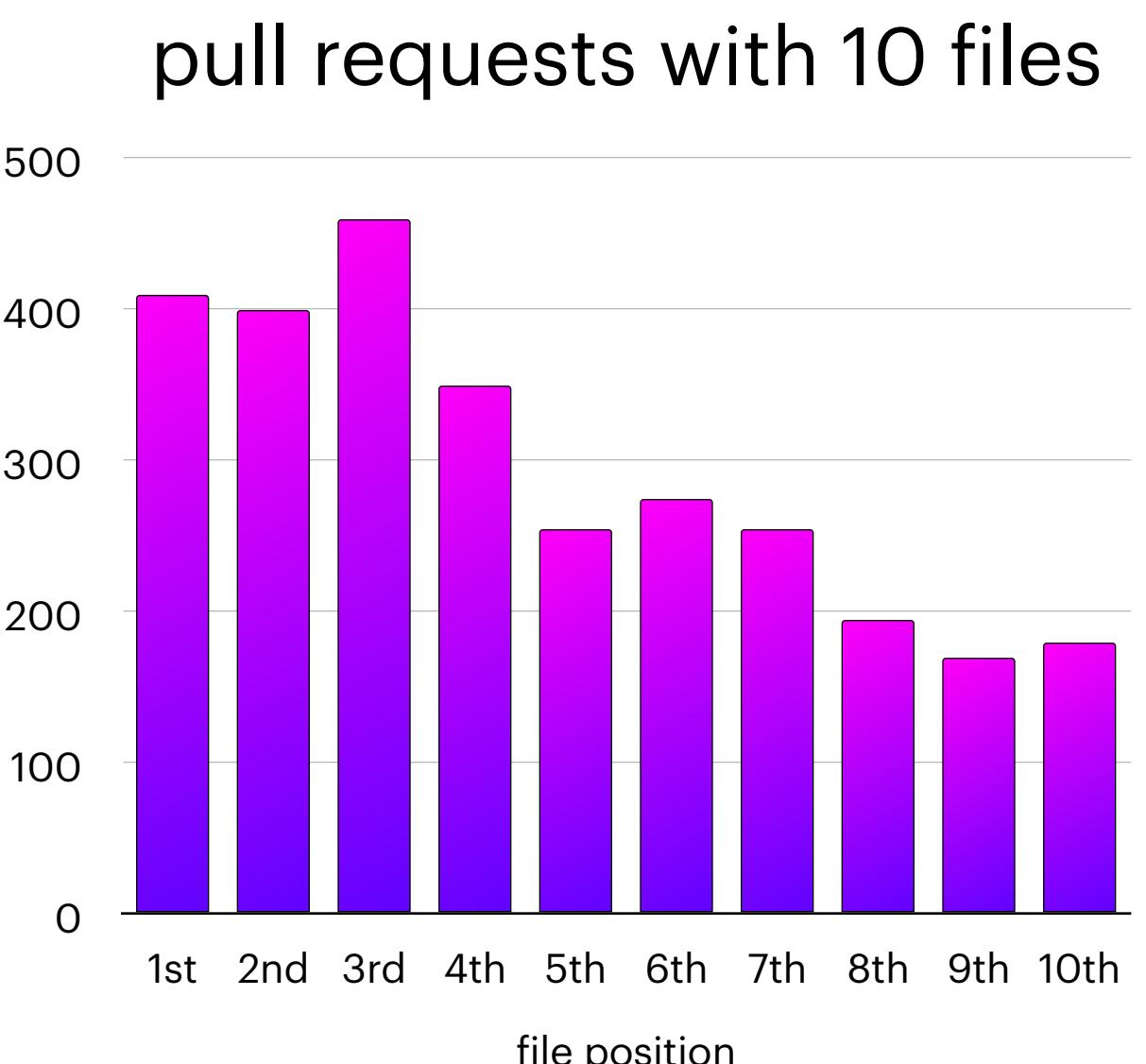
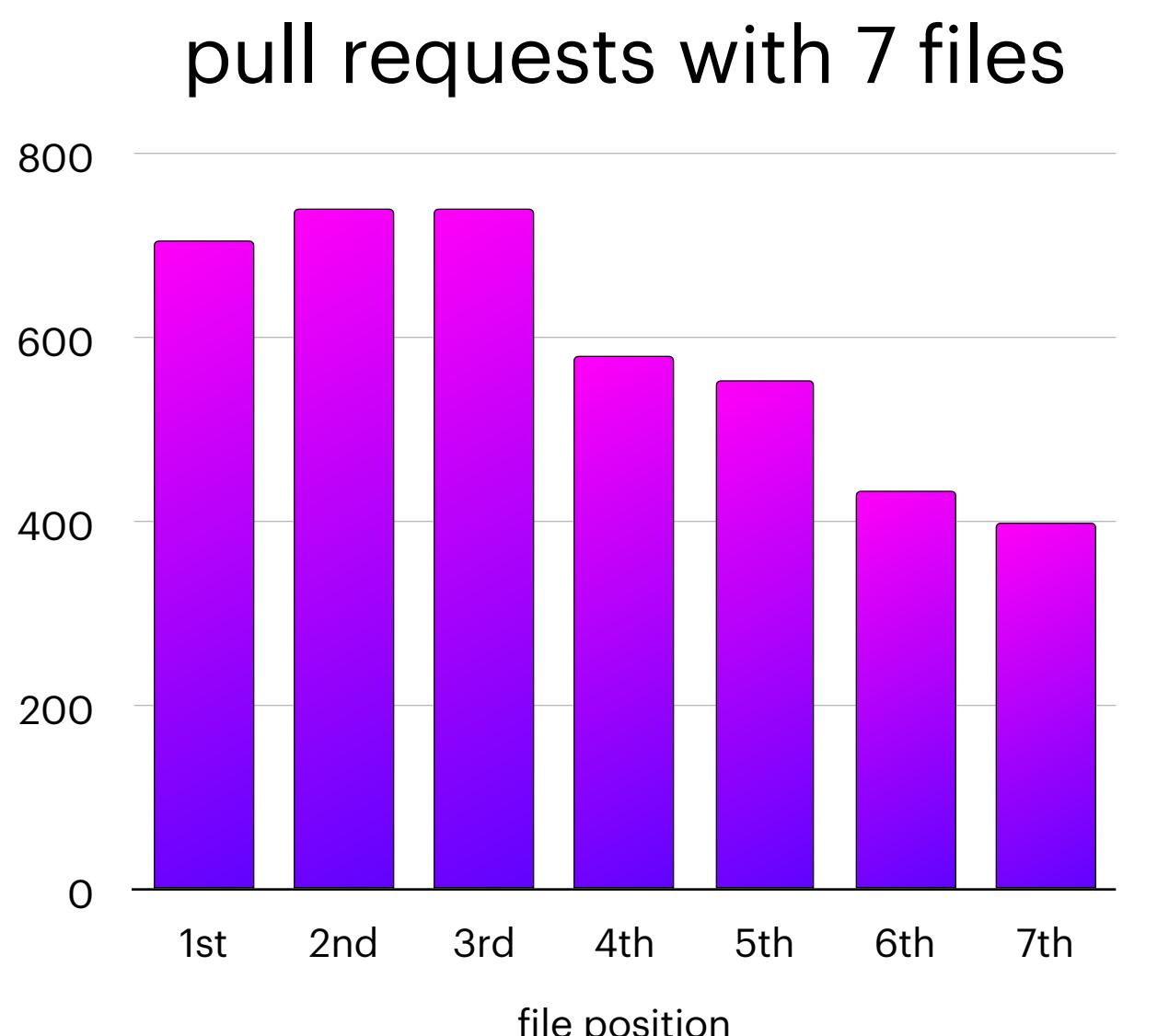


code review production files

- 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**

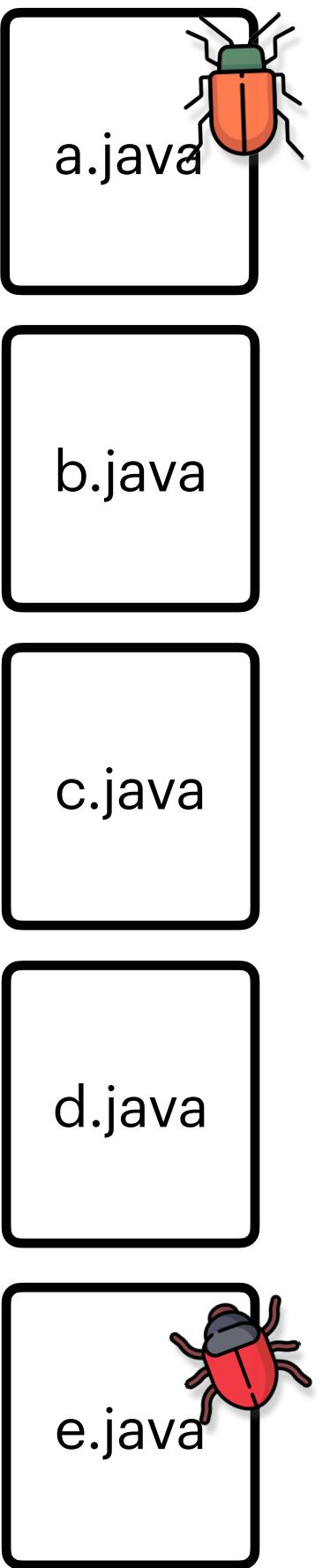


code review

production files experiment

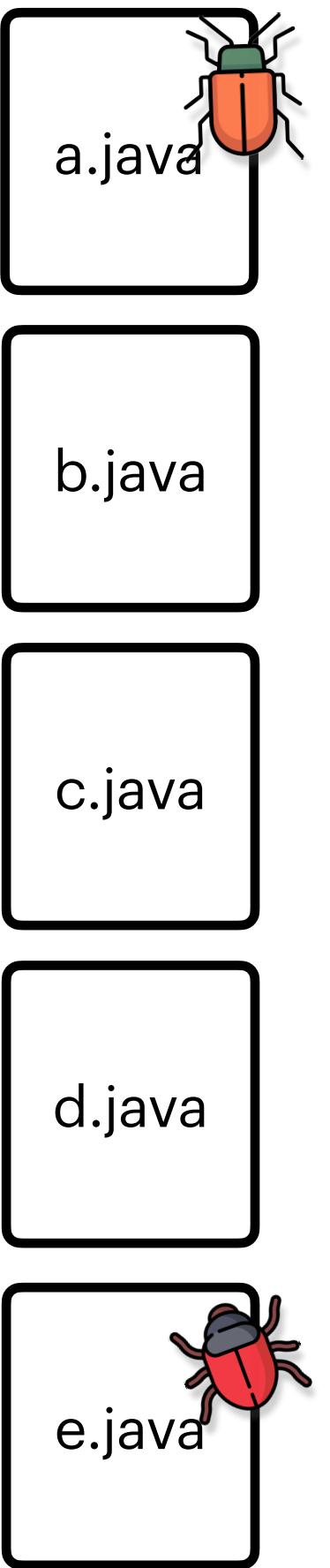
code review

production files experiment



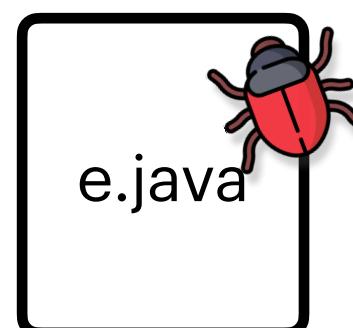
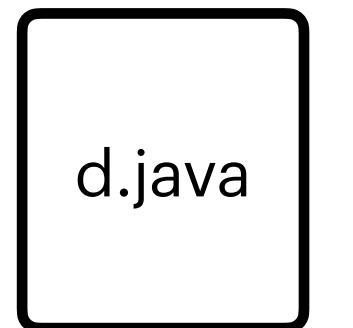
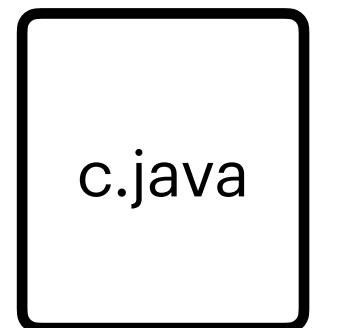
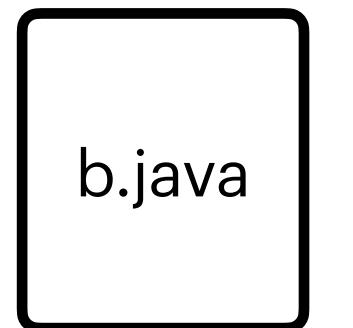
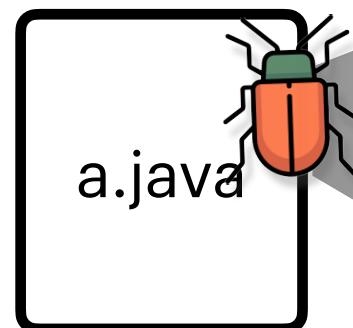
code review

production files experiment



code review

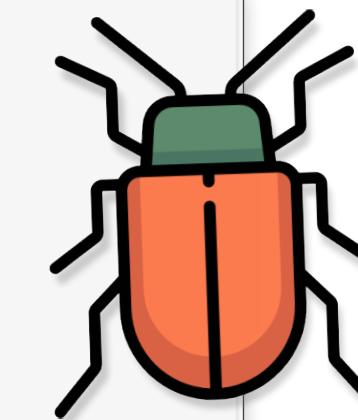
production files experiment



```
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;
```

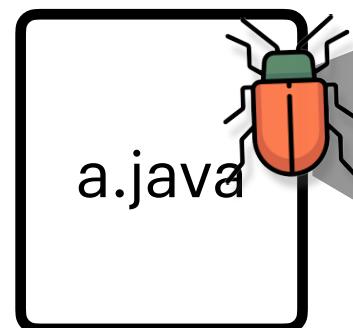
!! MB: Missing Break defect: Here a break statement is missing. In this way, when the country is UK, the execution will fall through the default case and a wrong tax of $1.27 * 2$ will be applied.

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

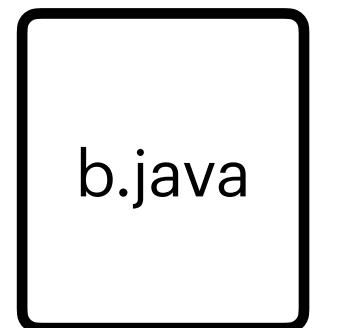


code review

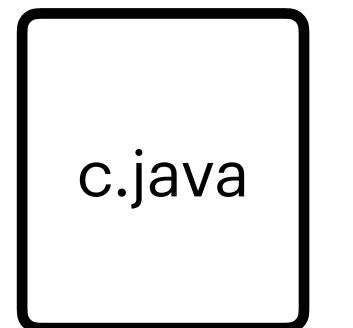
production files experiment



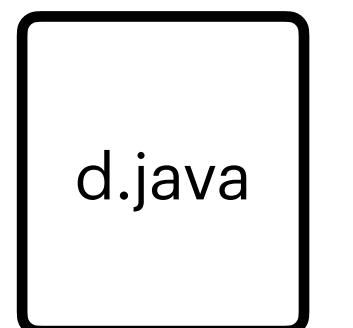
a.java



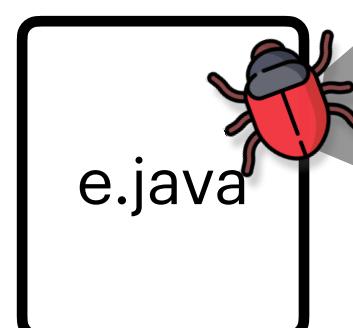
b.java



c.java



d.java

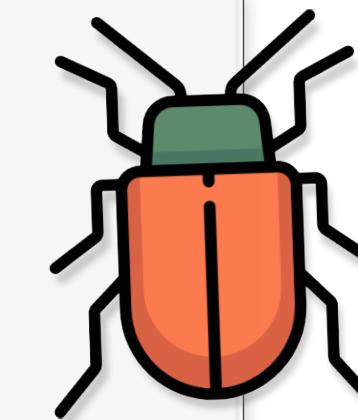


e.java

```
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.

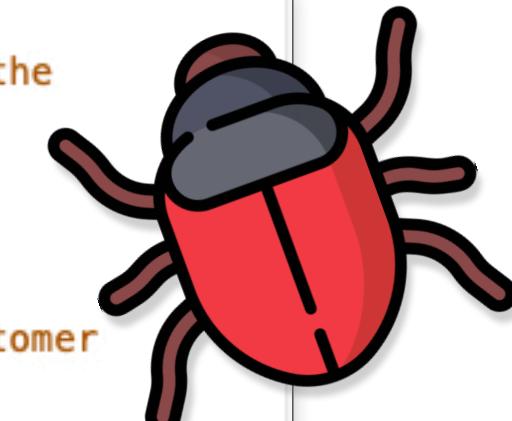
```
38     default:  
39         shippingCost = shippingCost * 2;  
40     }  
41     return shippingCost;
```



```
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) {  
19         discountRate = 0.1;
```

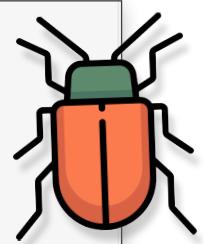
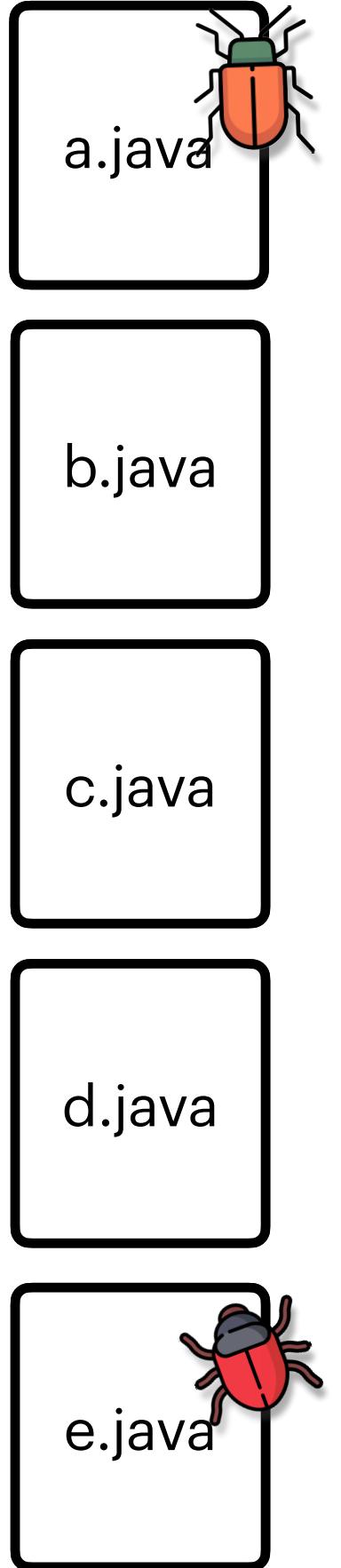
!! CC: Corner Case defect: Here the if statement is missing a check for the condition where customer.membershipLevel == 2. According to the Javadoc of the function, customers with membership level equal to 2 should receive a 10% discount

```
20         }  
21         if(membershipLevel >= 5) {  
22             discountRate = 0.25;  
23         }  
24     }  
25     return discountRate;  
26 }
```



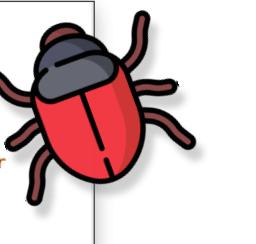
code review

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;
```

① 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.

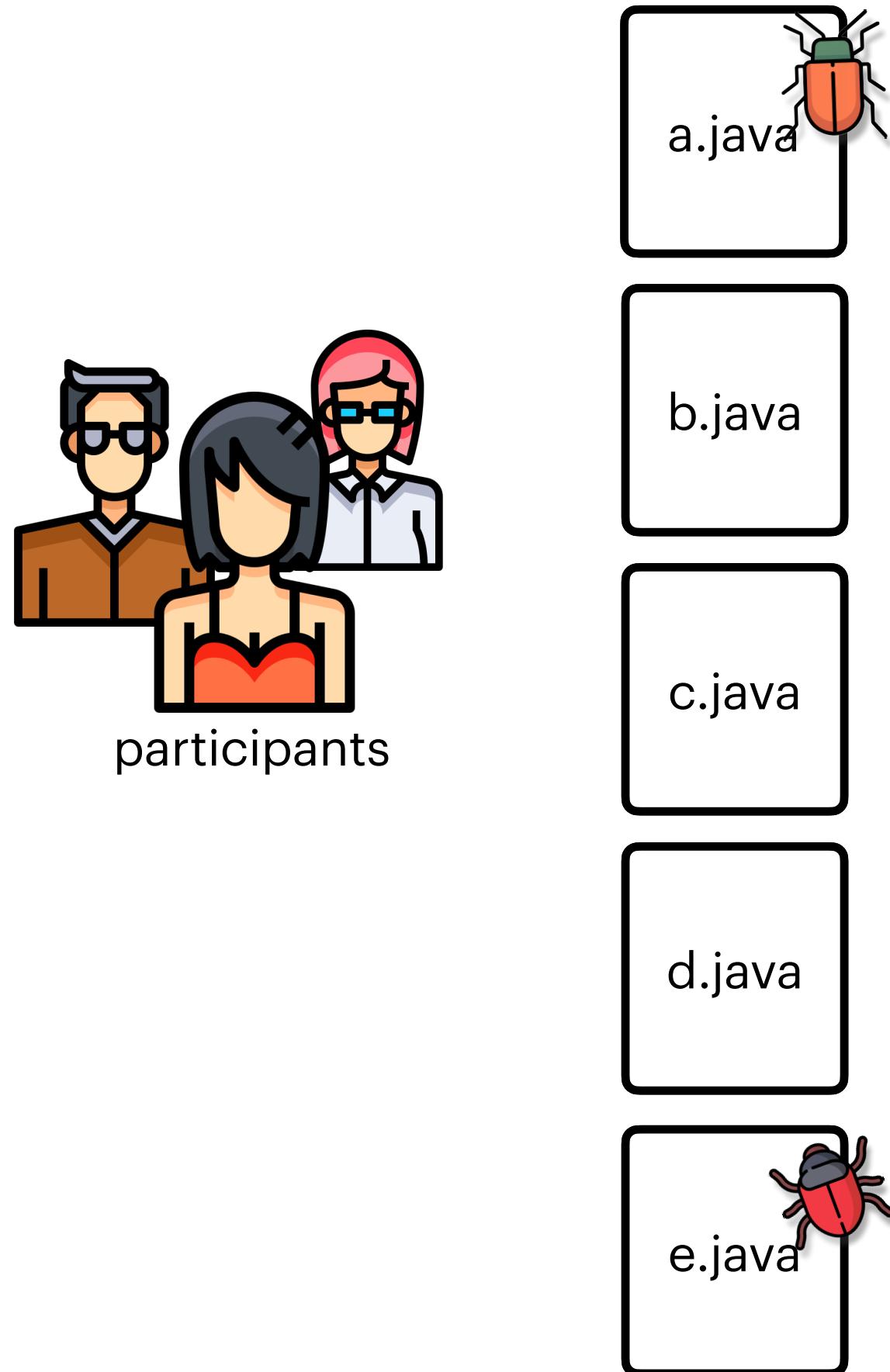


```
7  /**  
8   * Returns the discount rate based on the membership level of the  
9   * customer.  
10  * Customers at level 1 do not receive any discount.  
11  * Customers at level 2 to 4 receive a 10% discount.  
12  * Customers from level 5 included receive a 25% discount.  
13  * @param membershipLevel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

① 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

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;
```

① 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  */  
15  public double getSaleDiscountRate(int membershipLevel){  
16      double discountRate = 0;  
17      if(membershipLevel > 2 && membershipLevel < 5) {  
18          discountRate = 0.1;  
19      }  
20      if(membershipLevel >= 5) {  
21          discountRate = 0.25;  
22      }  
23      return discountRate;  
24 }
```

① 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

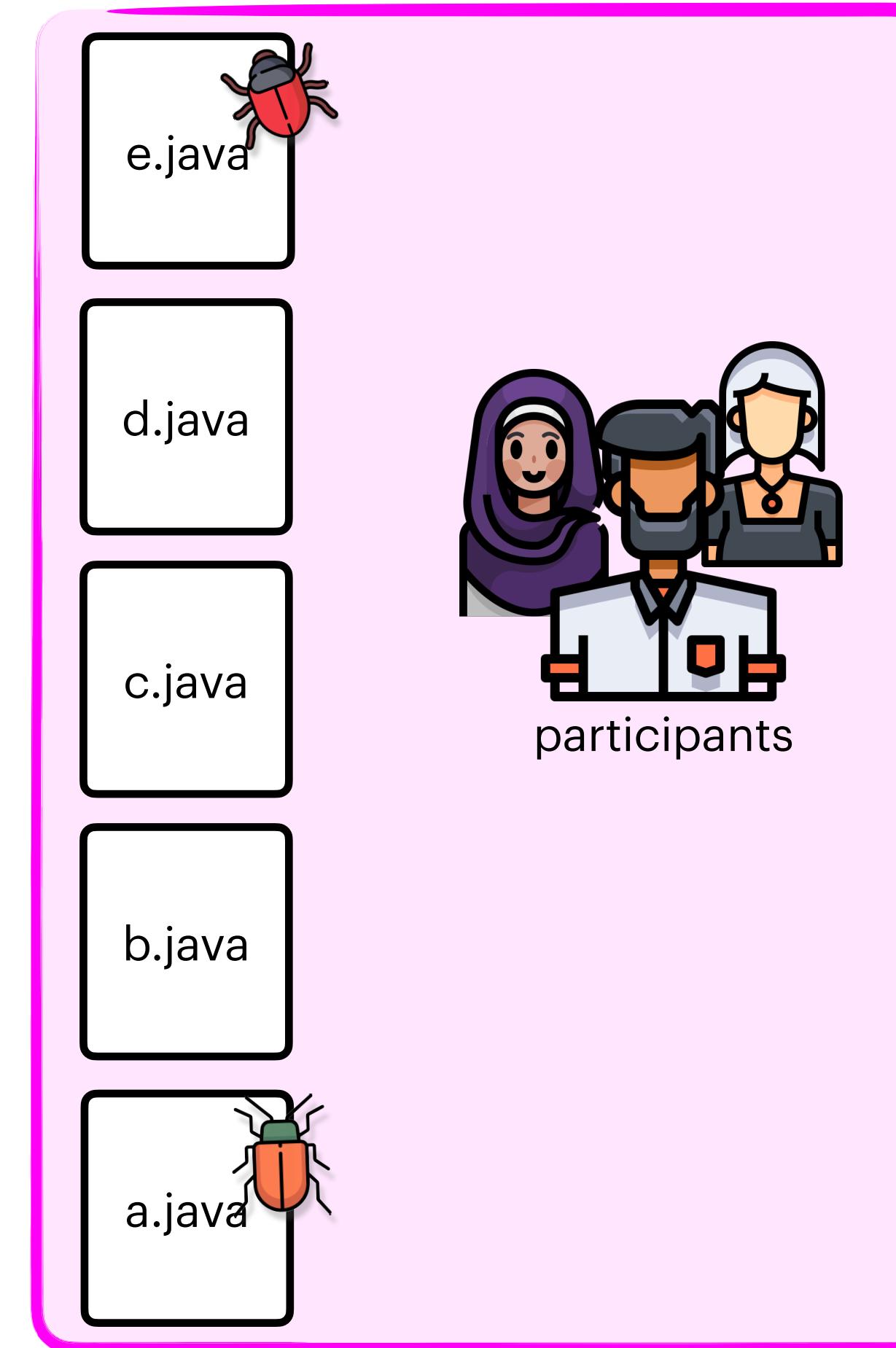
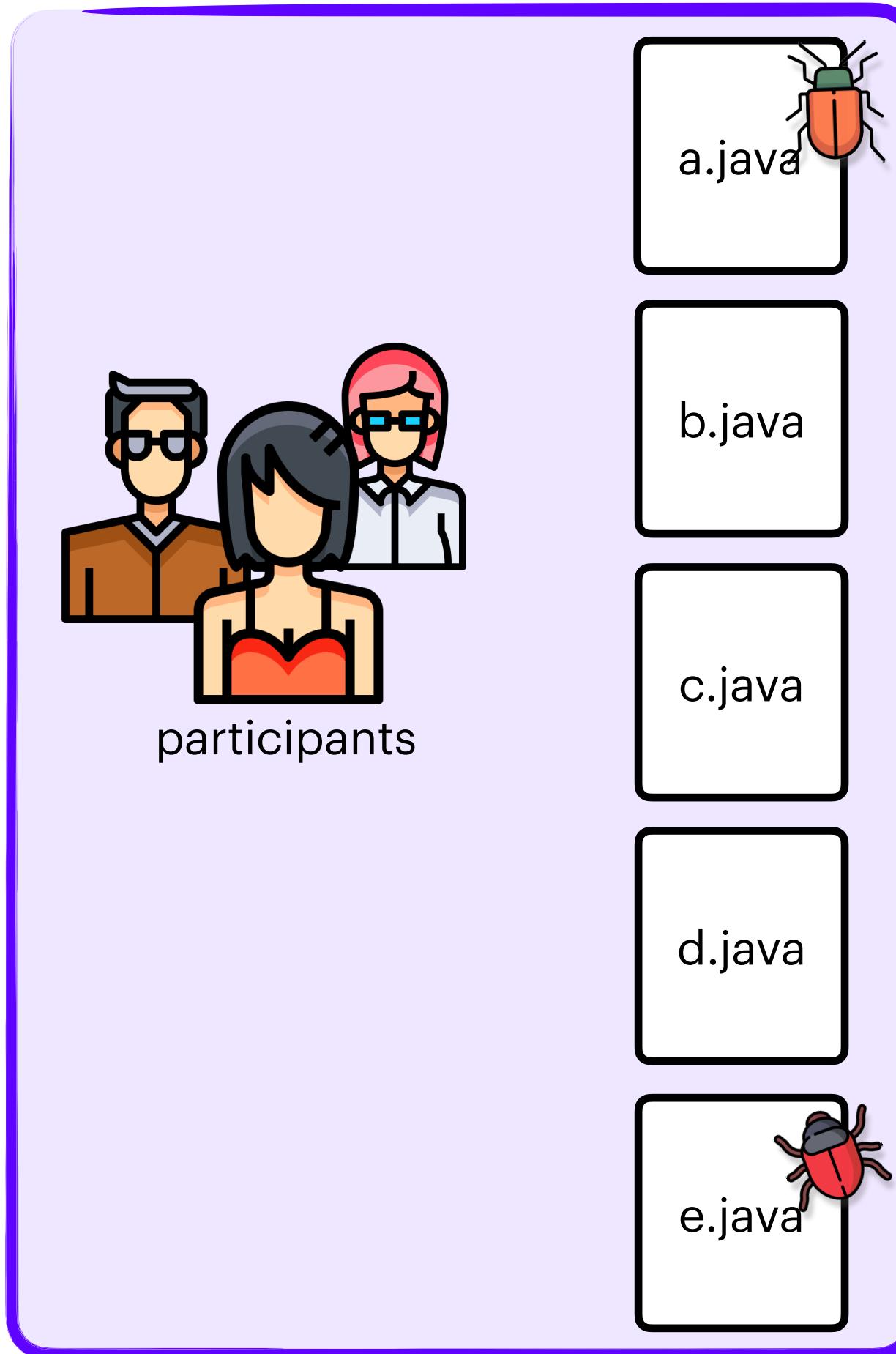
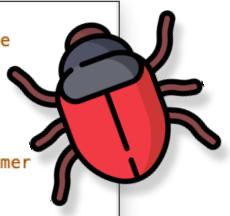
code review

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;  
  
① 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 from 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  */  
15  public double getSaleDiscountRate(int membershipLevel){  
16      double discountRate = 0;  
17      if(membershipLevel > 2 && membershipLevel < 5) {  
18          discountRate = 0.1;  
19      }  
20      if(membershipLevel >= 5) {  
21          discountRate = 0.25;  
22      }  
23      return discountRate;  
24 }
```



code review

production files experiment

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;
```

① 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

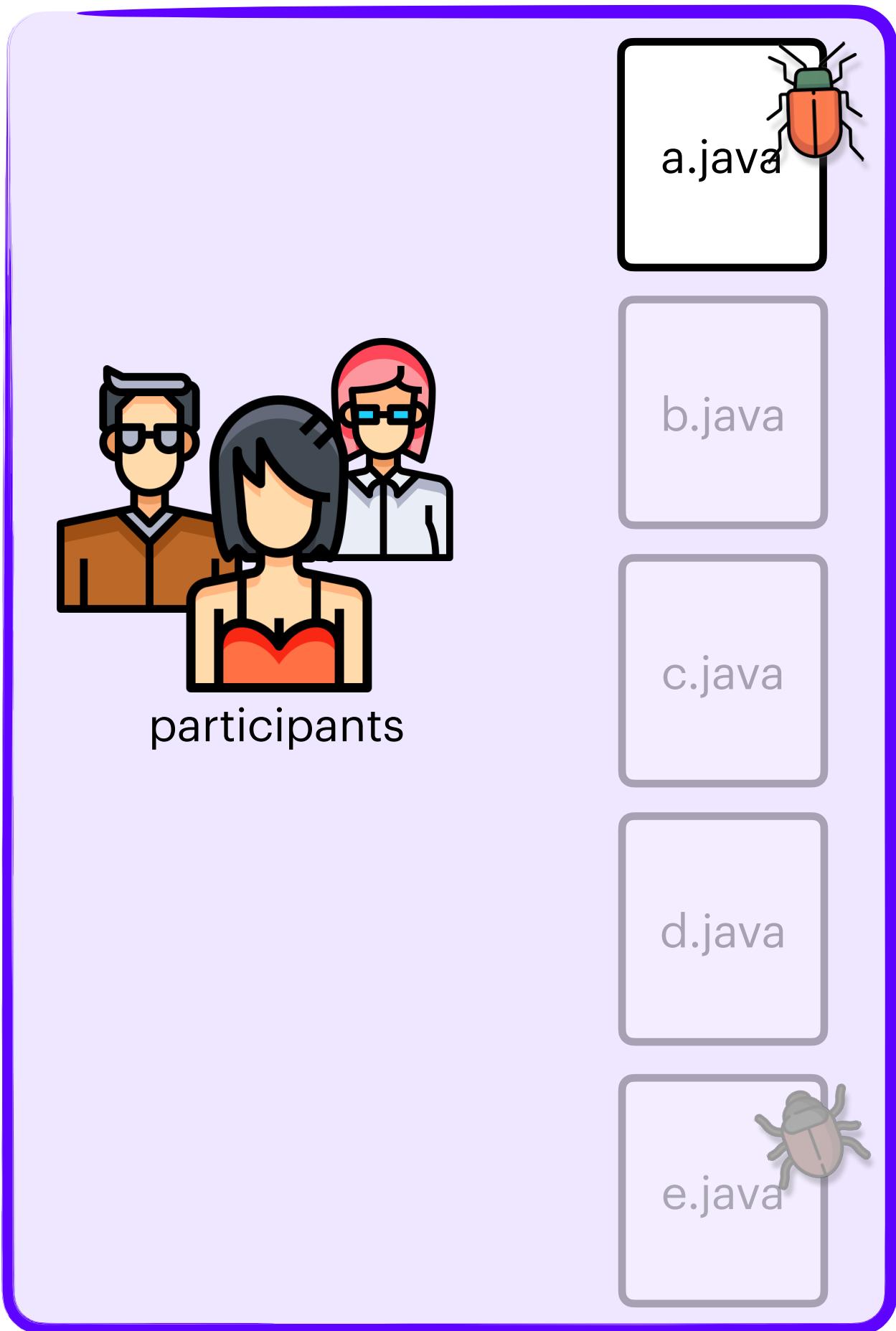
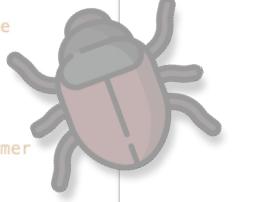
```
7  /**  
8   * Returns the discount rate based on the membership level of the  
9   * customer.  
10  * Customers at level 1 do not receive any discount.  
11  * Customers at level 2 to 4 receive a 10% discount.  
12  * Customers from level 5 included receive a 25% discount.  
13  * @param membershipLevel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

② 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.

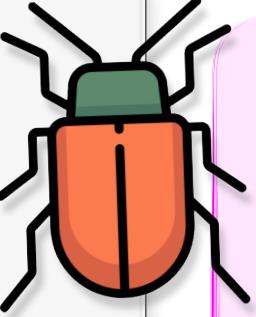
code review

production files experiment

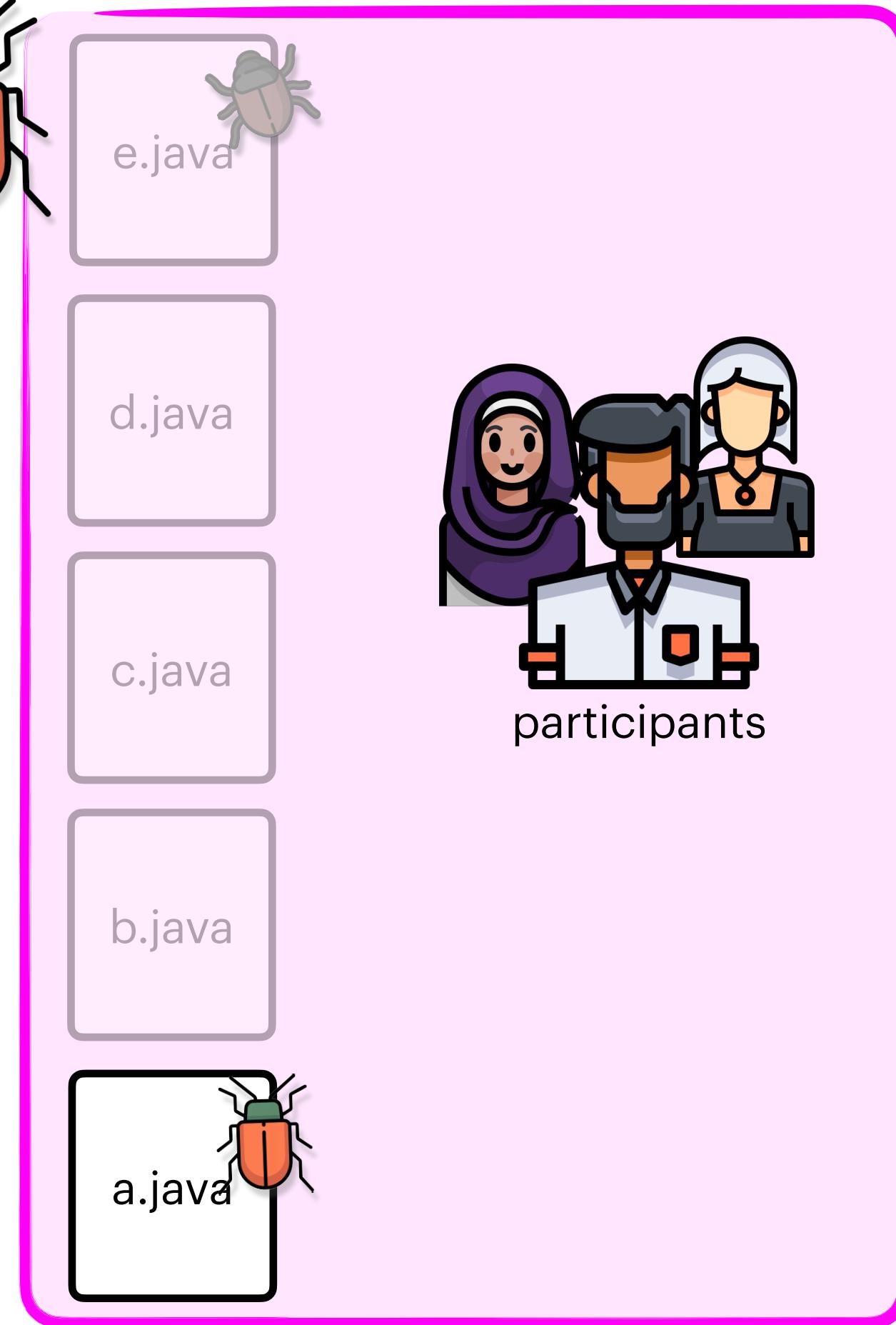
```
7  /**
8   * Returns the discount rate based on the membership level of the
9   * customer.
10  * Customers at level 1 do not receive any discount.
11  * Customers at level 2 to 4 receive a 10% discount.
12  * Customers from level 5 included receive a 25% discount.
13  * @param membershipLevel -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



```
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;
```



① 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.

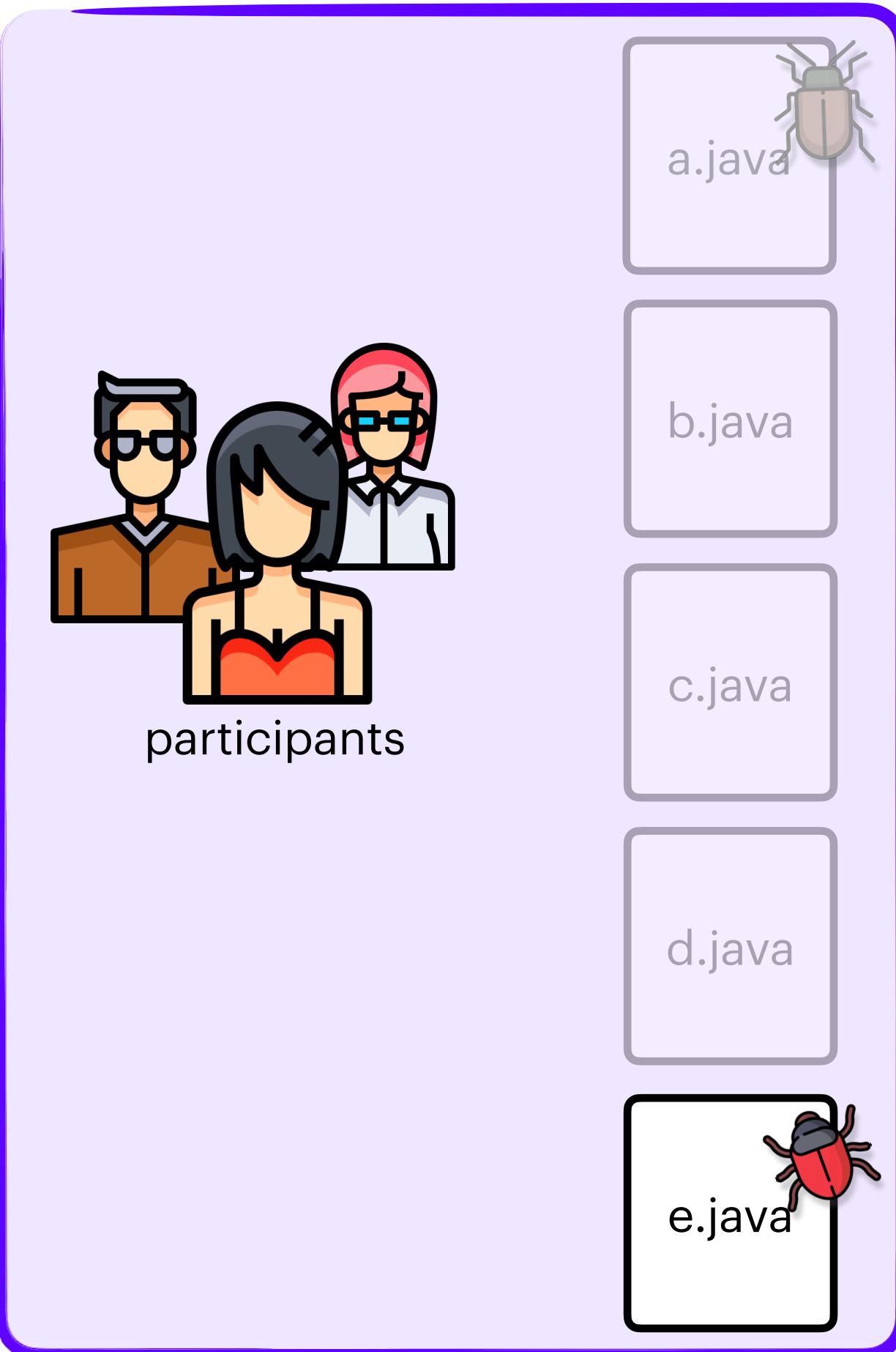
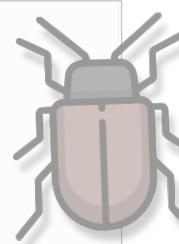


same likelihood of
finding the bug

code review

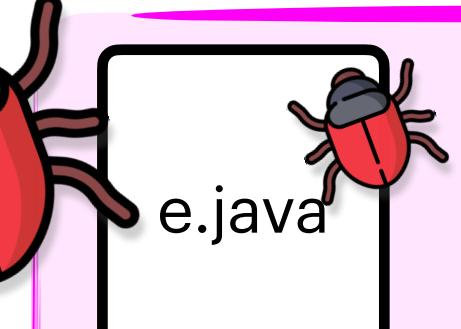
production files experiment

```
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;  
  
① 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;
```

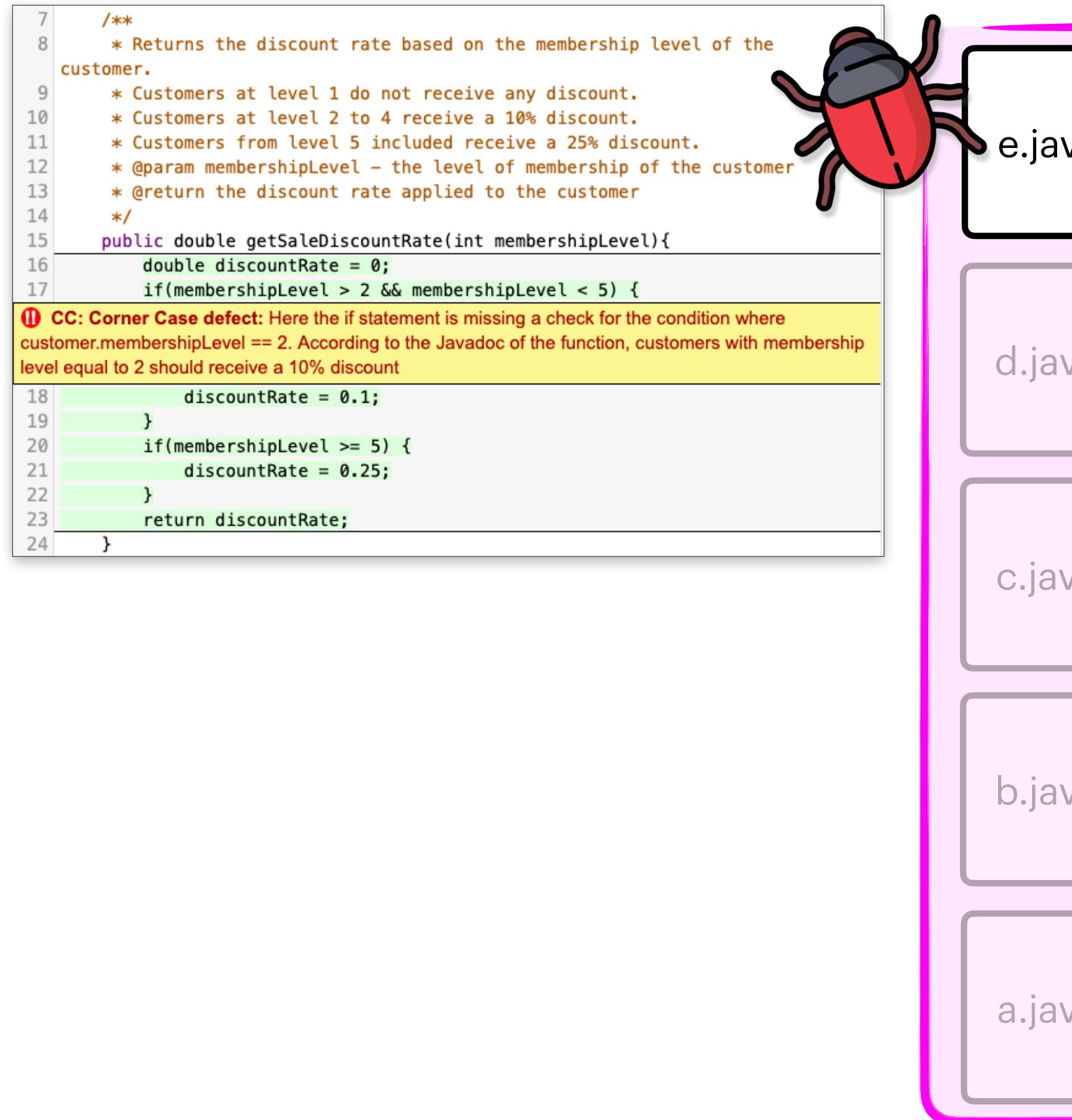


participants

```
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 }
```



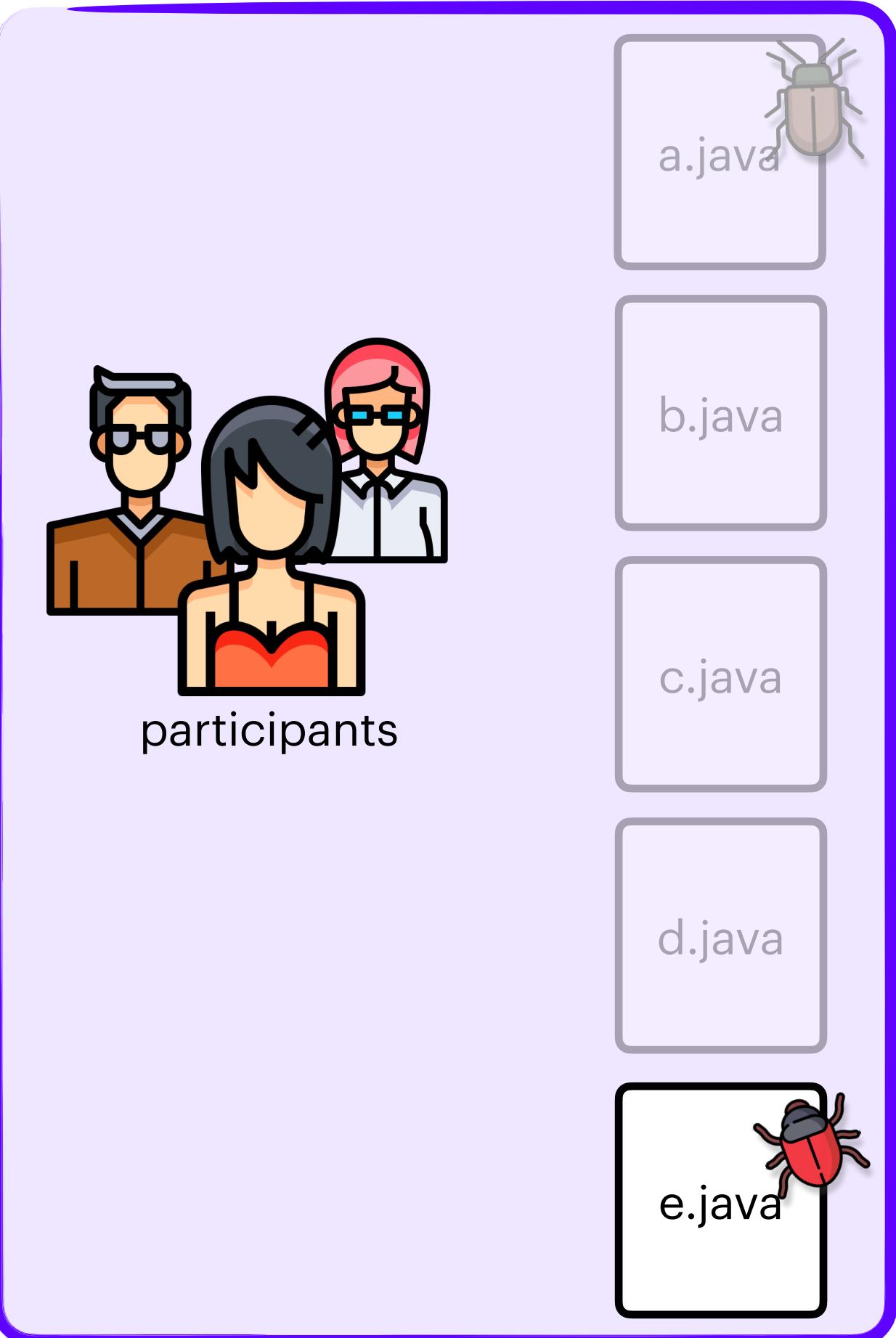
e.java



participants

code review

production files experiment

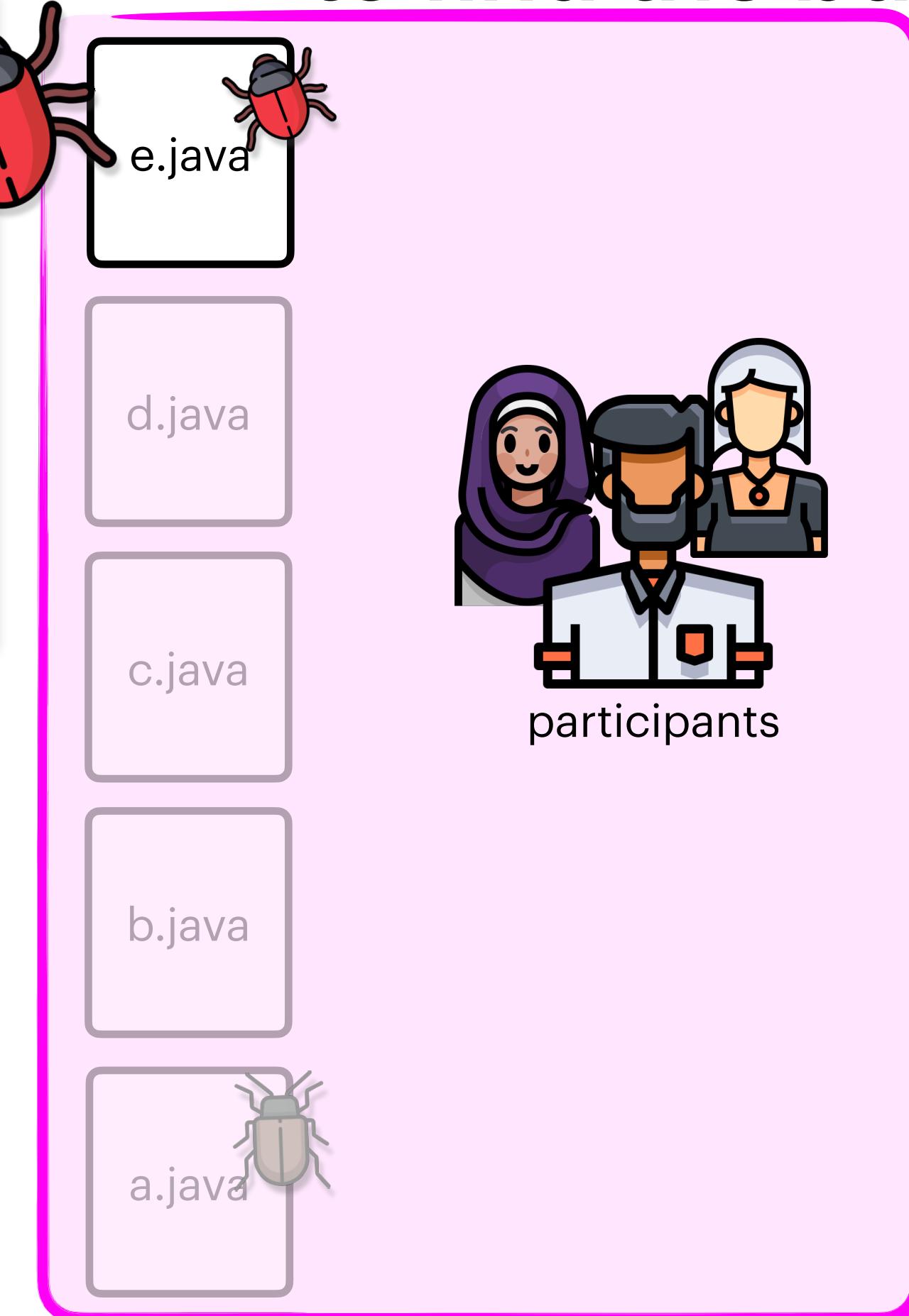


```
7  /**
8  * Returns the discount rate based on the membership level of the
9  * customer.
10 * Customers at level 1 do not receive any discount.
11 * Customers at level 2 to 4 receive a 10% discount.
12 * Customers from level 5 included receive a 25% discount.
13 * @param membershipLevel – the level 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 }
```

① 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

```
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;
```

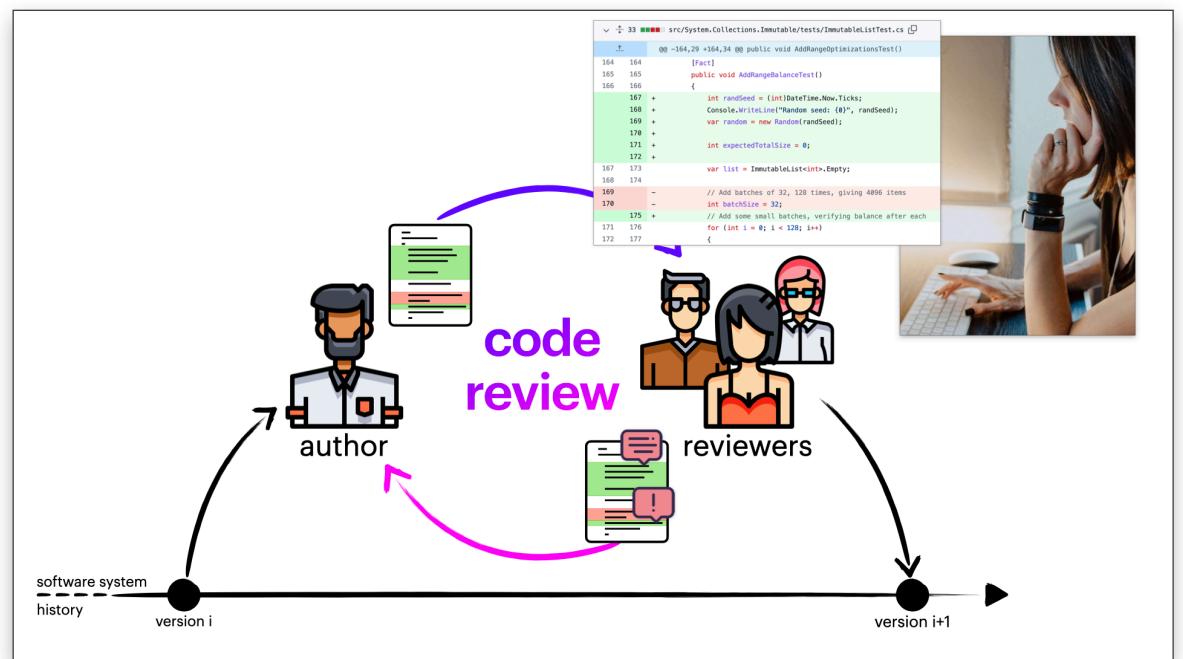
② MB: Missing Break defect: Here a break statement is missing. In this way, when the country is UK, the execution will fall through the default case and a wrong tax of 1.27 * 2 will be applied.



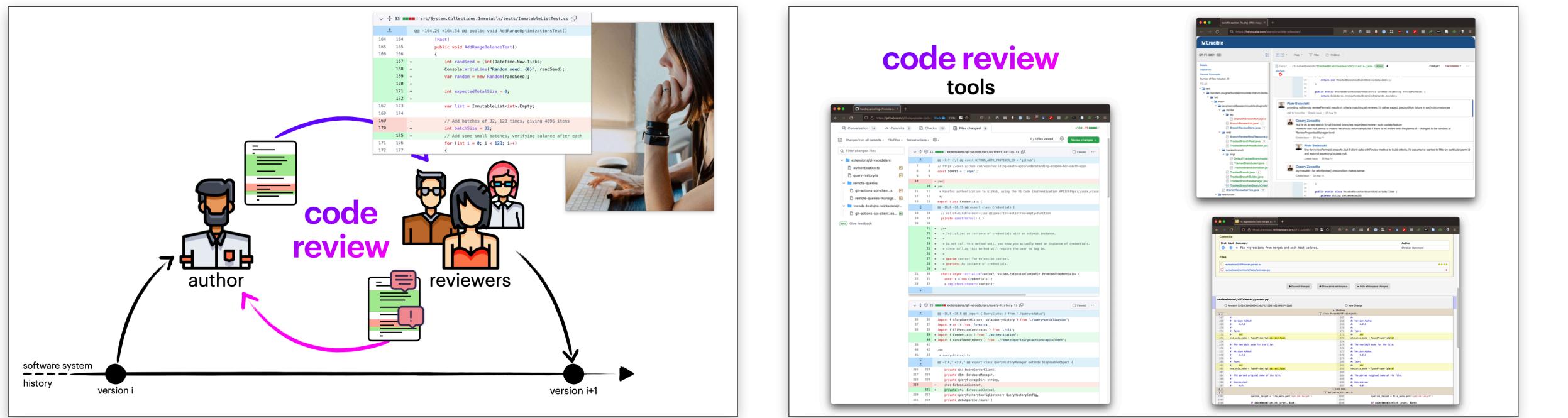
175% more likely
to find the bug

review

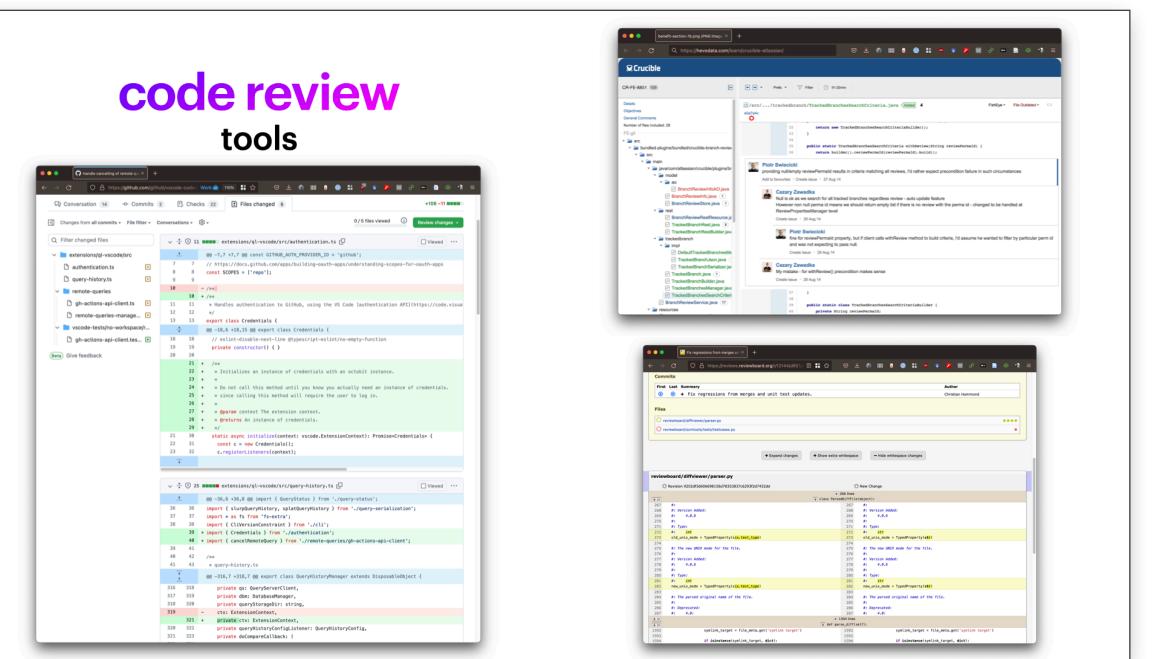
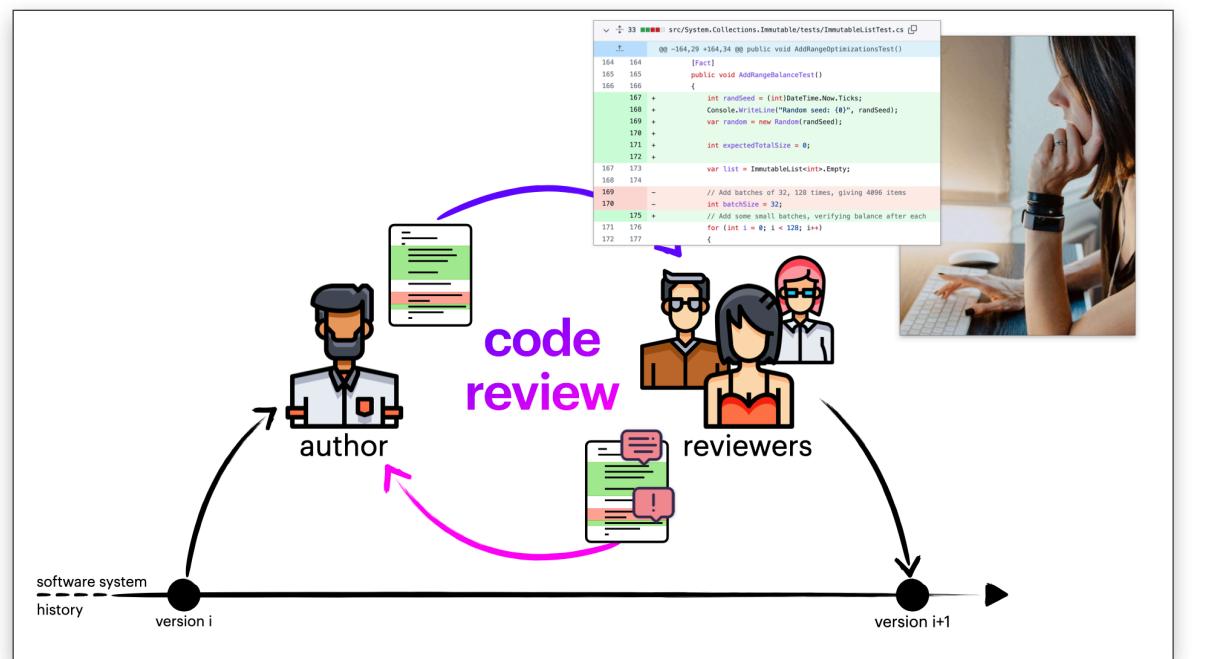
review



review



review

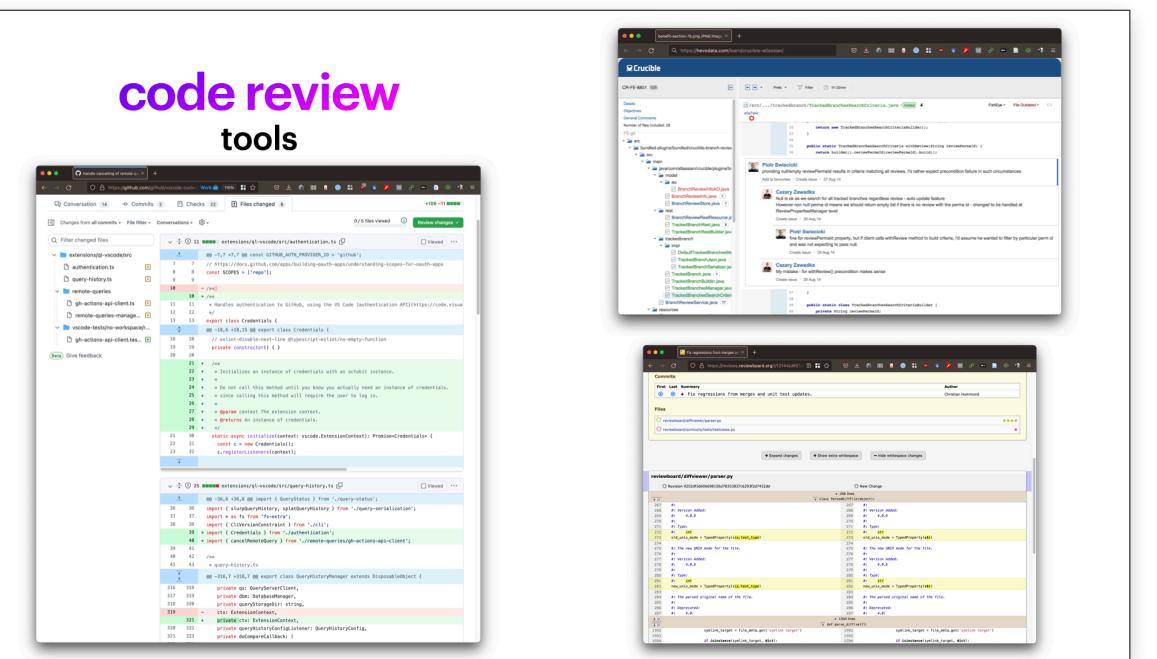
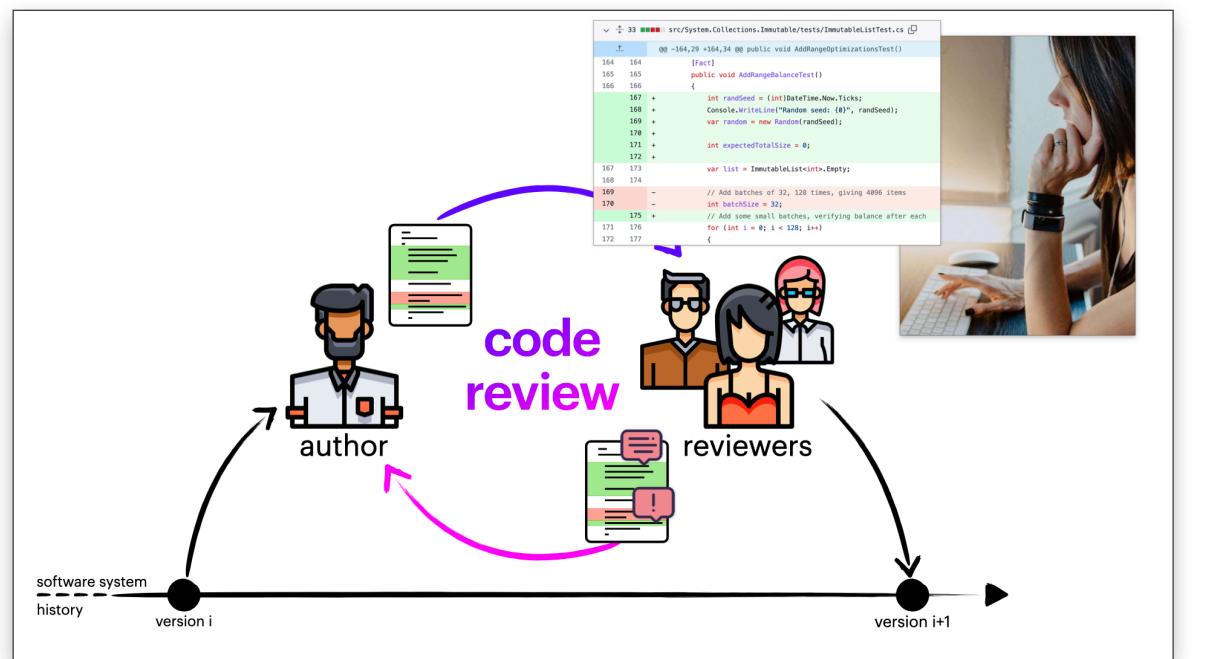


code review test files

- 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]

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]

code review test order experiment

production code first order
participants | production.java

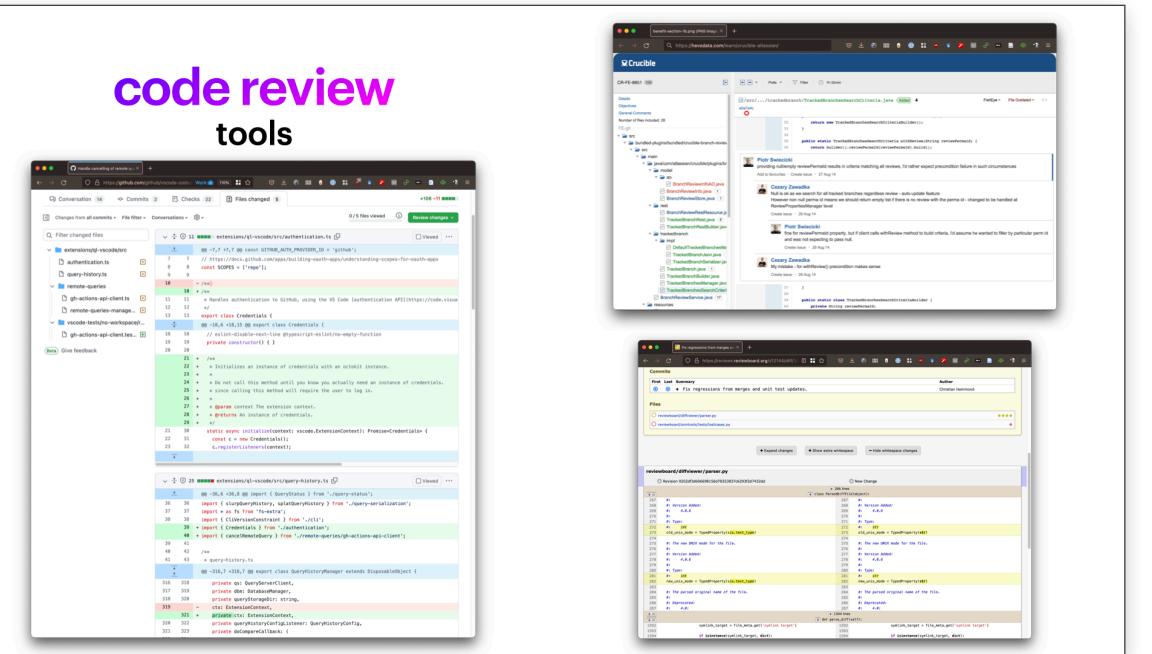
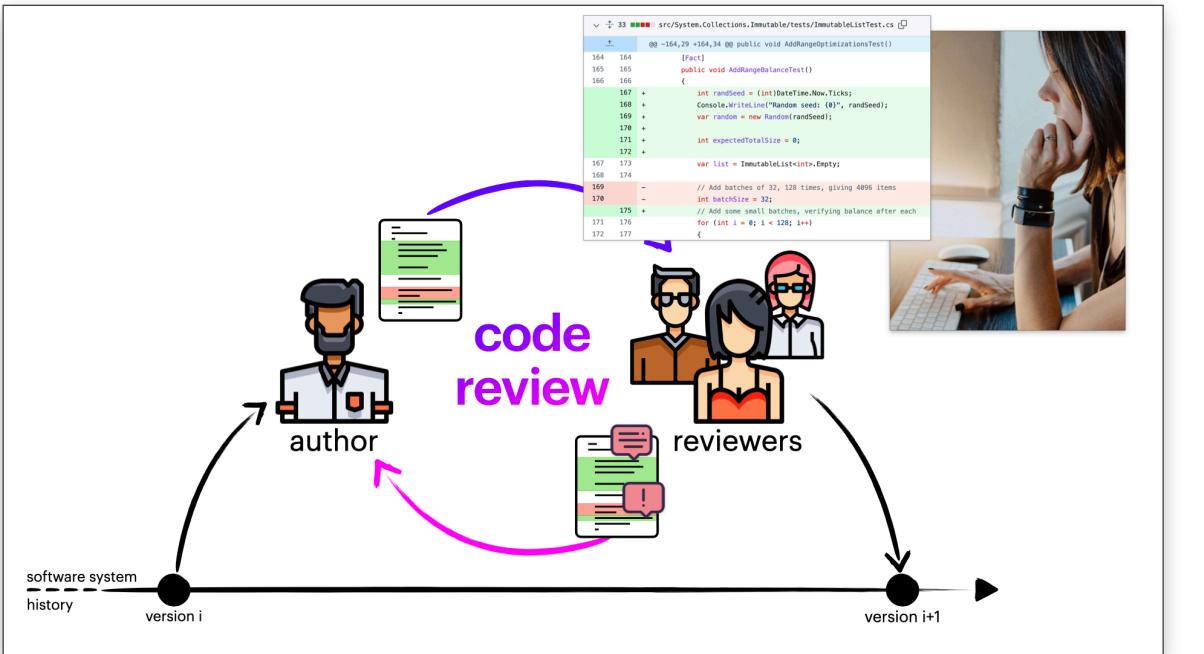
test code first order
participants | test.java | production.java

same likelihood of finding the production bugs

250% more likely to find the test bug

A screenshot of a GitHub repository showing a comparison between production code first and test code first orders. The production code first order shows participants reviewing production.java. The test code first order shows participants reviewing test.java and production.java. The text indicates that the test code first order results in 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]

code review test order experiment

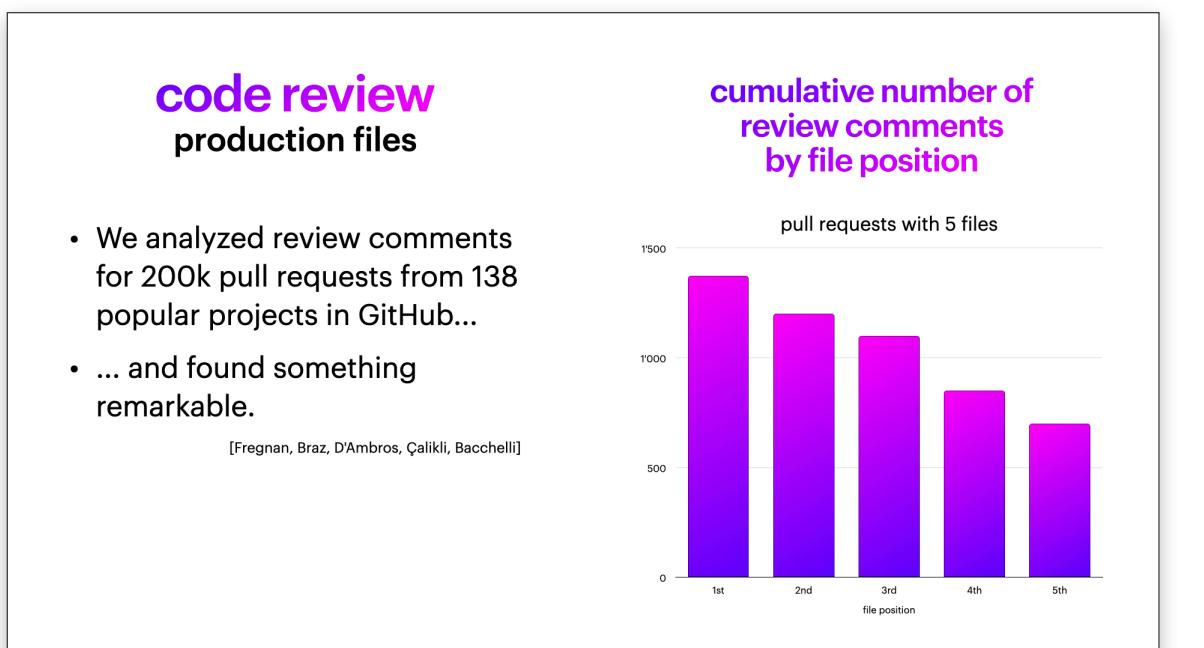
production code first order

test code first order

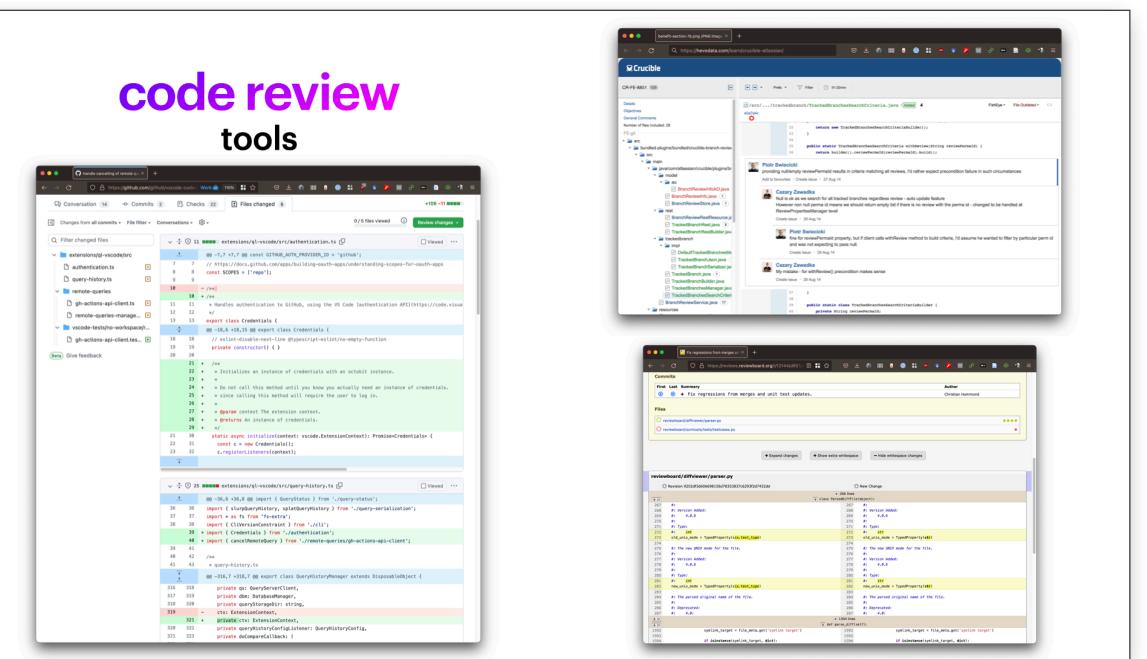
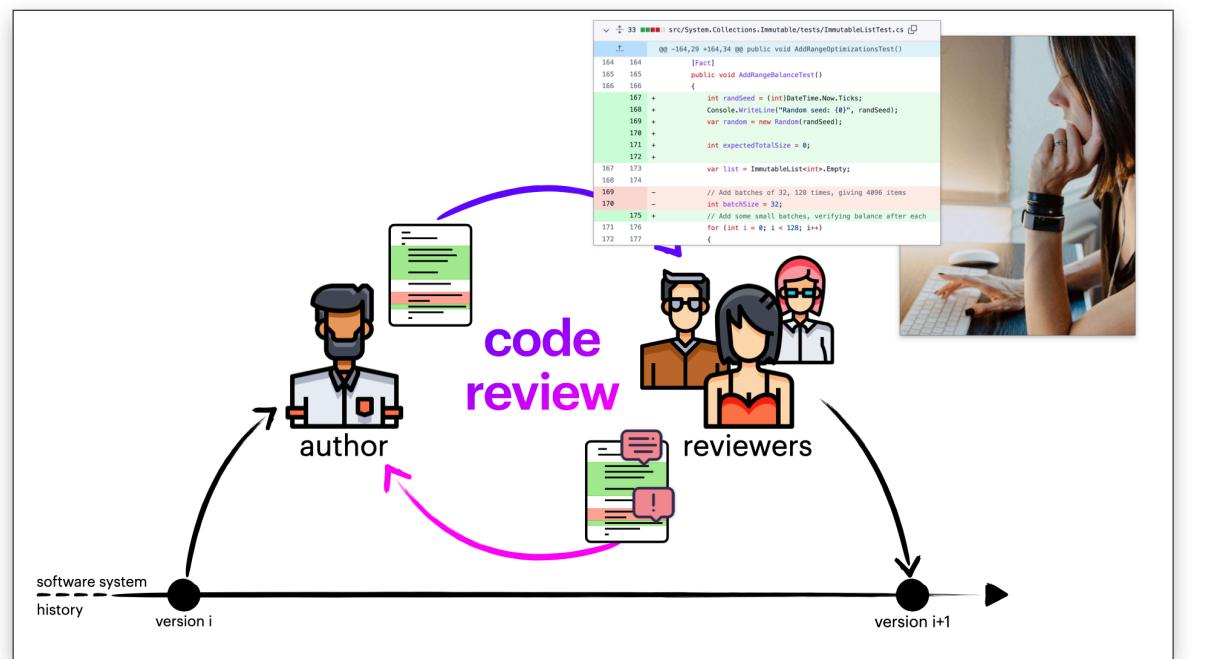
same likelihood of finding the production bugs

250% more likely to find the test bug

A GitHub pull request interface showing code changes. It highlights that test files are less likely to be discussed during code review when they are grouped with production files.



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]

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

A GitHub repository interface showing a file structure with both production and test files. A callout highlights the difference in likelihood of finding bugs between the two orders.

code review production files

cumulative number of review comments by file position

pull requests with 5 files

A bar chart titled "pull requests with 5 files" showing the cumulative number of review comments across five file positions. The y-axis ranges from 0 to 1500, and the x-axis shows the file positions from 1st to 5th.

file position	cumulative review comments
1st	~1300
2nd	~1100
3rd	~950
4th	~800
5th	~650

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

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

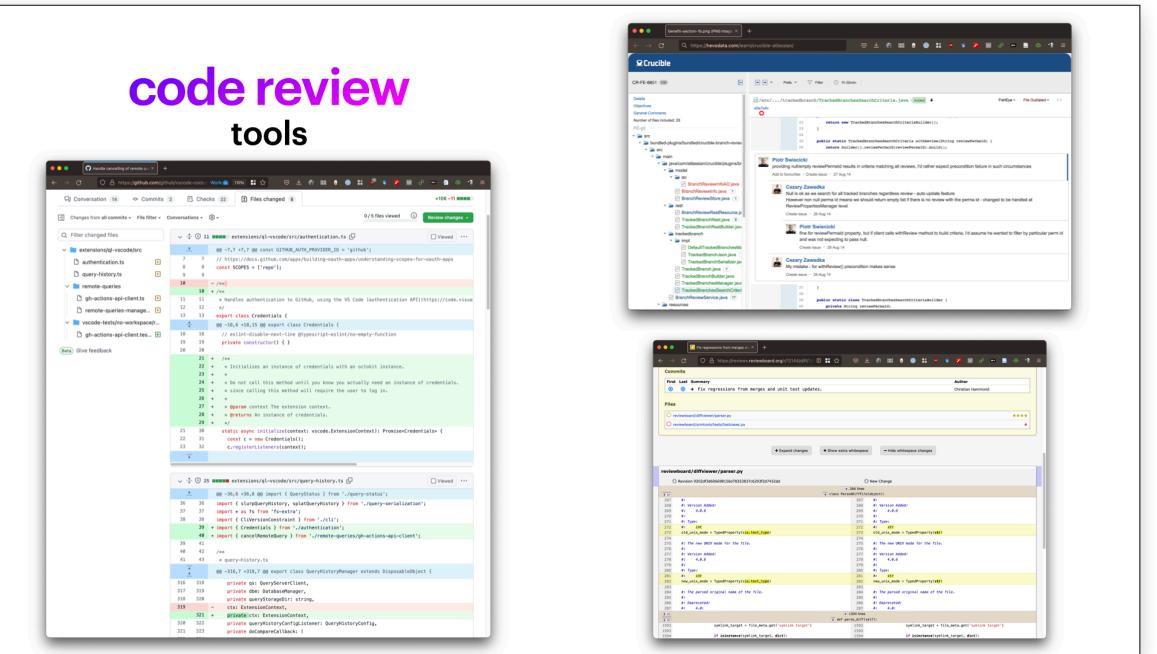
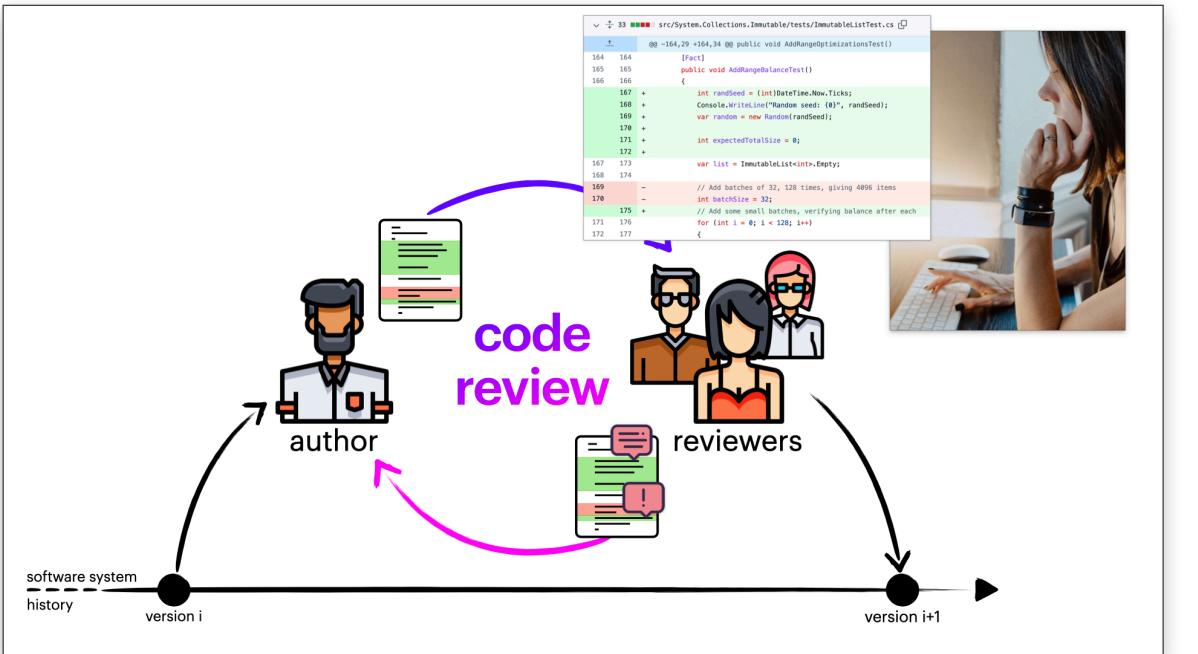
code review production files order experiment

A diagram showing five Java files (a.java, b.java, c.java, d.java, e.java) arranged in a circle. Some files have small red bugs. Participants are shown reviewing files in different orders. A callout states: "175% more likely to find the bug when in the 1st file shown".

Code snippet from a.java:

```
public class CustomerDiscountCalculator {  
    // ...  
    public double calculateDiscountRate(MemberLevel membershipLevel) {  
        double discountRate = 0.0;  
        if (membershipLevel == MemberLevel.GOLD) {  
            discountRate = 0.15;  
        } else if (membershipLevel == MemberLevel.SILVER) {  
            discountRate = 0.1;  
        } else if (membershipLevel == MemberLevel.BRONZE) {  
            discountRate = 0.05;  
        } else if (membershipLevel == MemberLevel.FREE) {  
            discountRate = 0.0;  
        }  
        return discountRate;  
    }  
}
```

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]

code review test order experiment

production code first order vs **test code first order**

same likelihood of finding the production bugs

250% more likely to find the test bug

A screenshot of a GitHub repository showing a comparison between production code first and test code first orders.

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	cumulative review comments
1st	~1300
2nd	~1050
3rd	~850
4th	~650
5th	~500

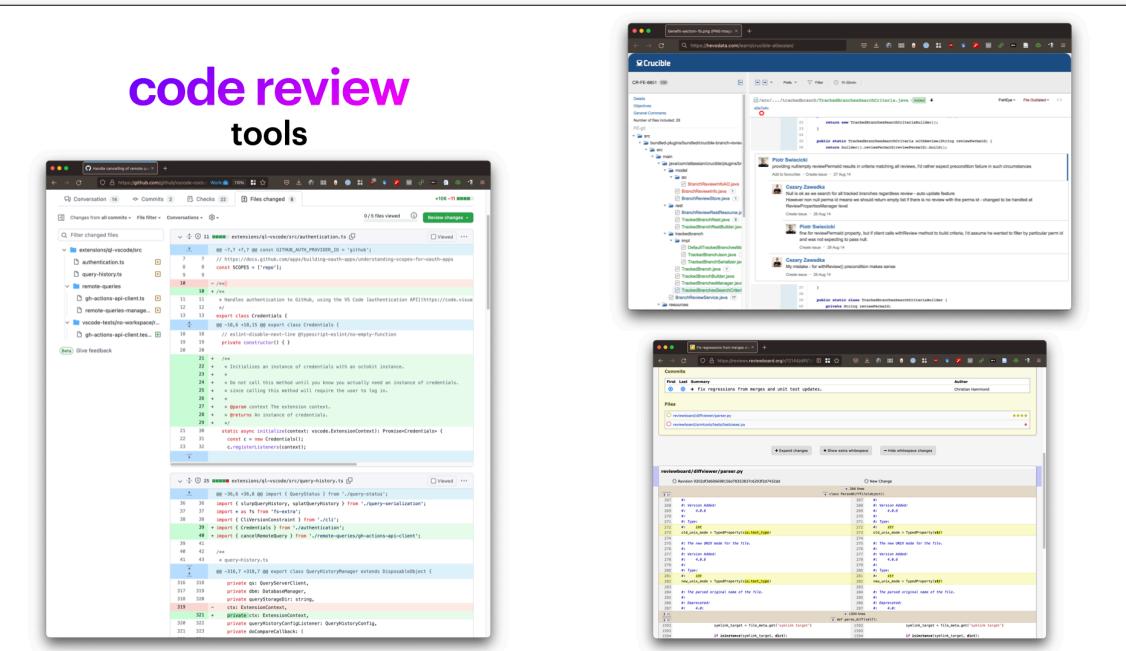
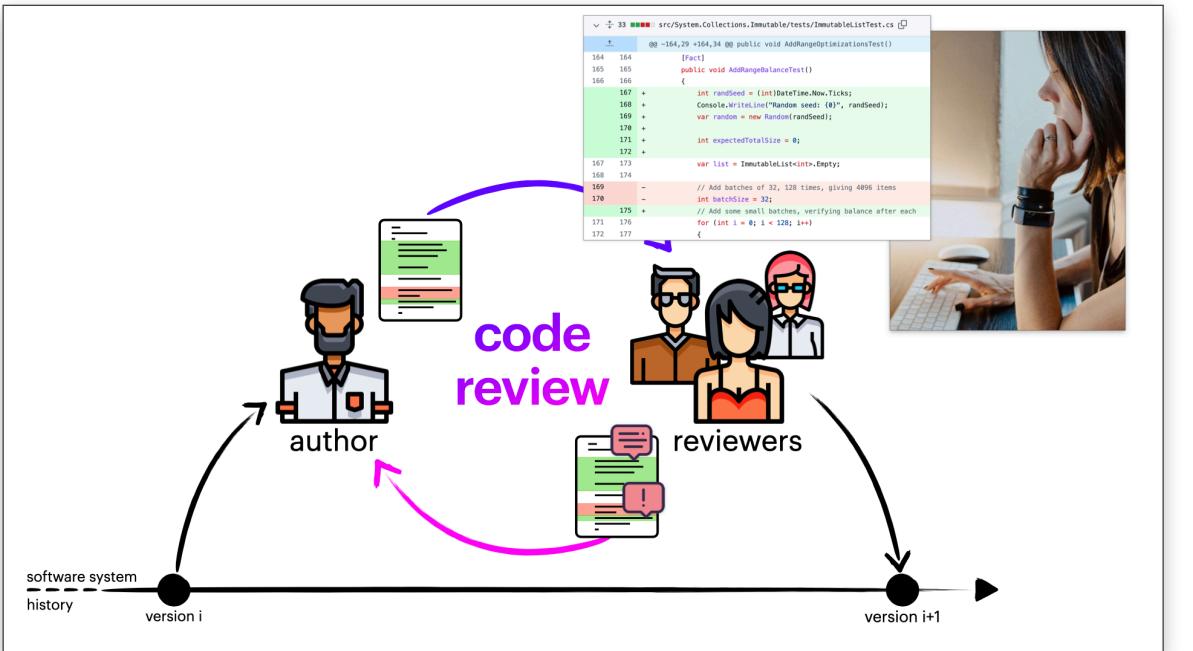
code review production files order experiment

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

A diagram showing five Java files (a.java, b.java, c.java, d.java, e.java) in different orders. A red bug icon is placed on file `e.java` in the top row. In the bottom row, the files are arranged such that `e.java` is the first file. A note states: "175% more likely to find the bug when in the 1st file shown".

takeaways

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]

code review test order experiment

production code first order

This diagram shows two participants reviewing production code files (`production.java` and `test.java`). It includes a snippet of Java code for a `Credentials` class and a note that the same likelihood of finding production bugs is achieved.

test code first order

This diagram shows two participants reviewing test code files (`test.java` and `production.java`). It includes a snippet of Java code for a `queryHistory` method and a note that it is 250% more likely to find test 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

A bar chart titled "pull requests with 5 files" showing the cumulative number of review comments by file position. The x-axis is labeled "file position" and has categories "1st", "2nd", "3rd", "4th", and "5th". The y-axis ranges from 0 to 1500. The bars show a decreasing trend: 1st (~1300), 2nd (~1000), 3rd (~800), 4th (~600), and 5th (~500).

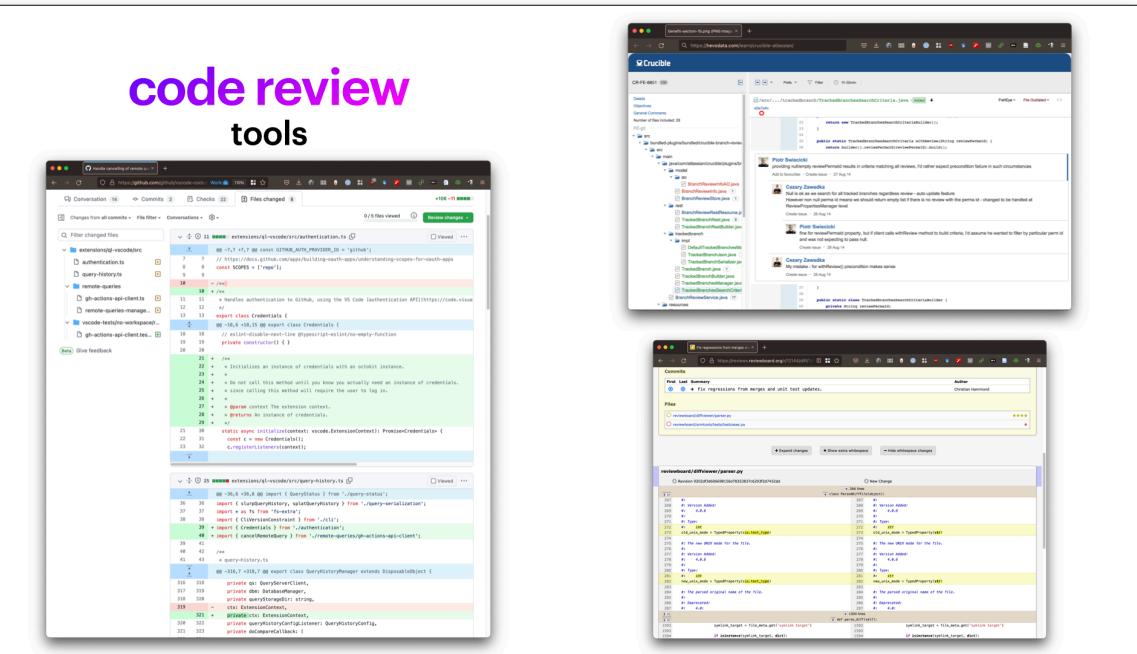
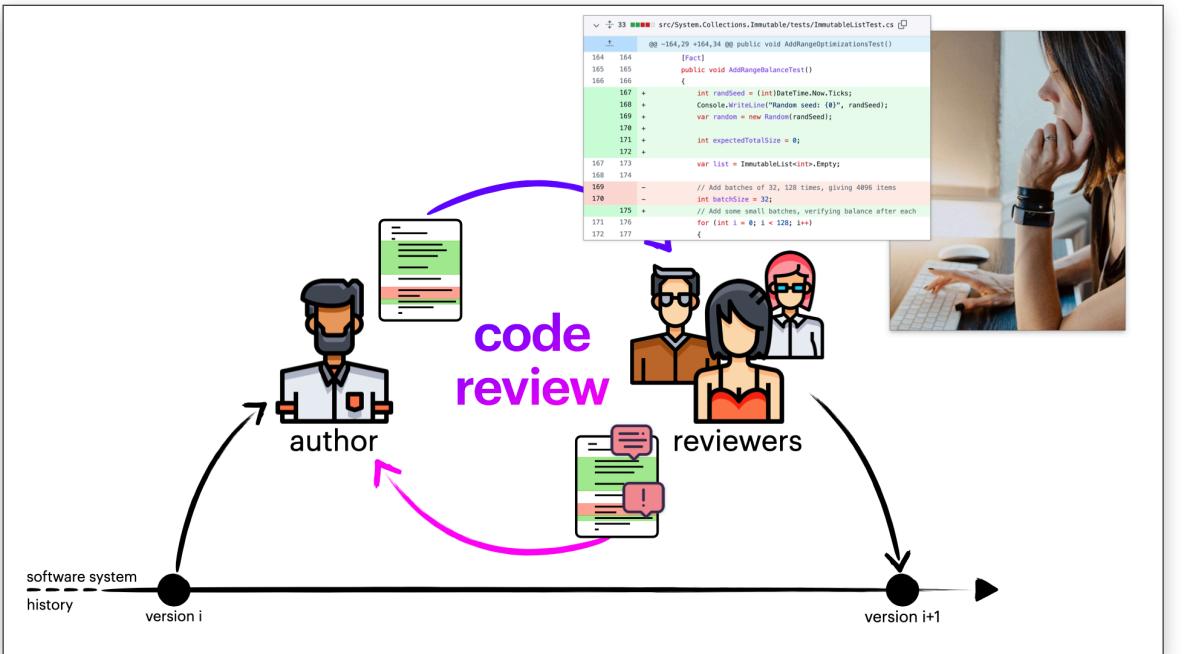
code review production files order experiment

This diagram illustrates an experiment where participants review files in a different order than they were written. It shows five files: `a.java`, `b.java`, `c.java`, `d.java`, and `e.java`. The order of review is `e.java`, `a.java`, `b.java`, `c.java`, and `d.java`. A note states that it is 175% more likely to find the bug when in the 1st file shown.

takeaways

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

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]

code review test order experiment

production code first order vs **test code first order**

same likelihood of finding the production bugs

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]

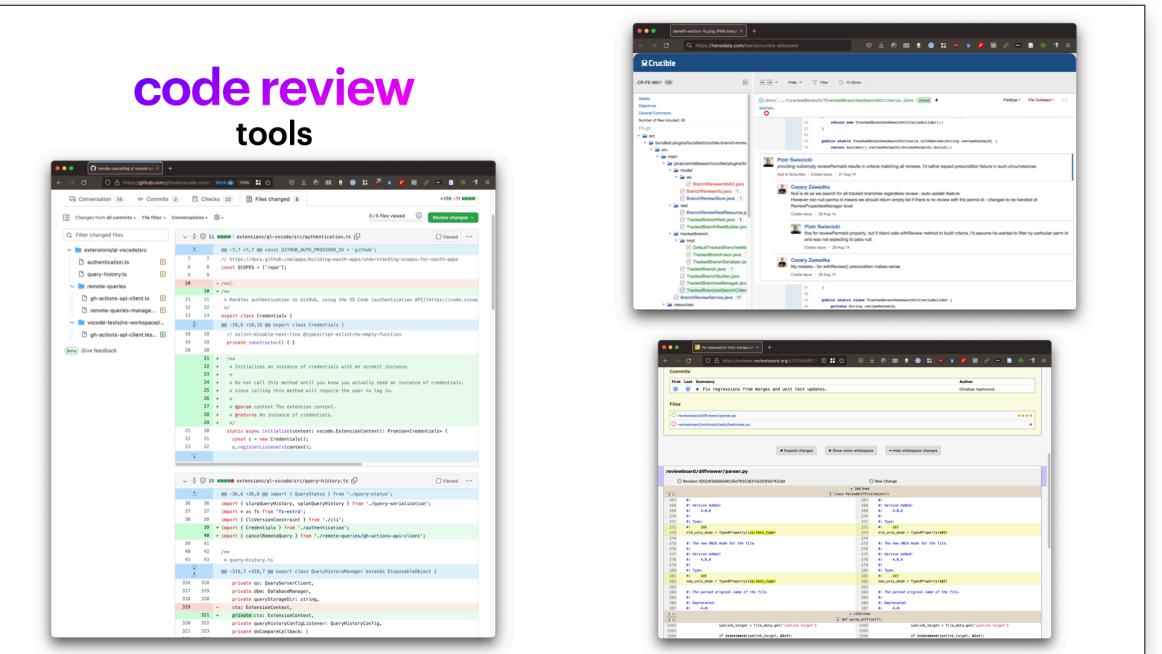
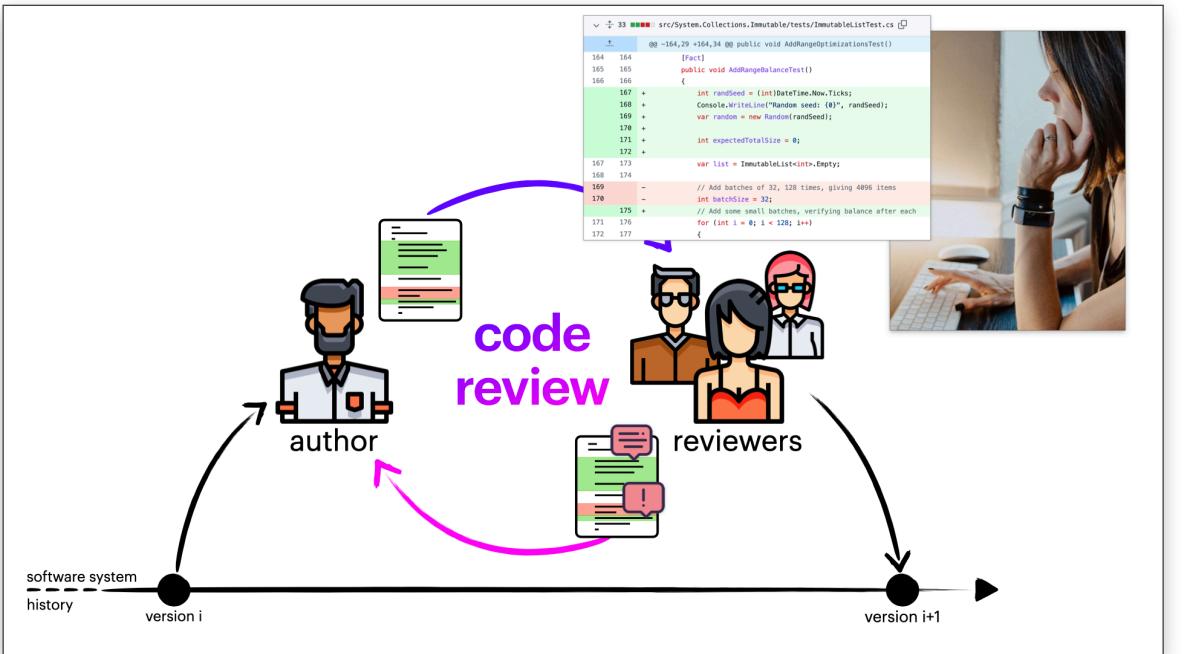
code review production files order experiment

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

takeaways

- 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



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 vs **test code first order**

same likelihood of finding the production bugs
250% more likely to find the test bug

A GitHub pull request interface showing code changes. It highlights that test files are less likely to be discussed during review when they are grouped with production files.

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, Çalaklı, Bacchelli]

cumulative number of review comments by file position

pull requests with 5 files

file position	review comments
1st	~1300
2nd	~1000
3rd	~800
4th	~600
5th	~500

code review production files order experiment

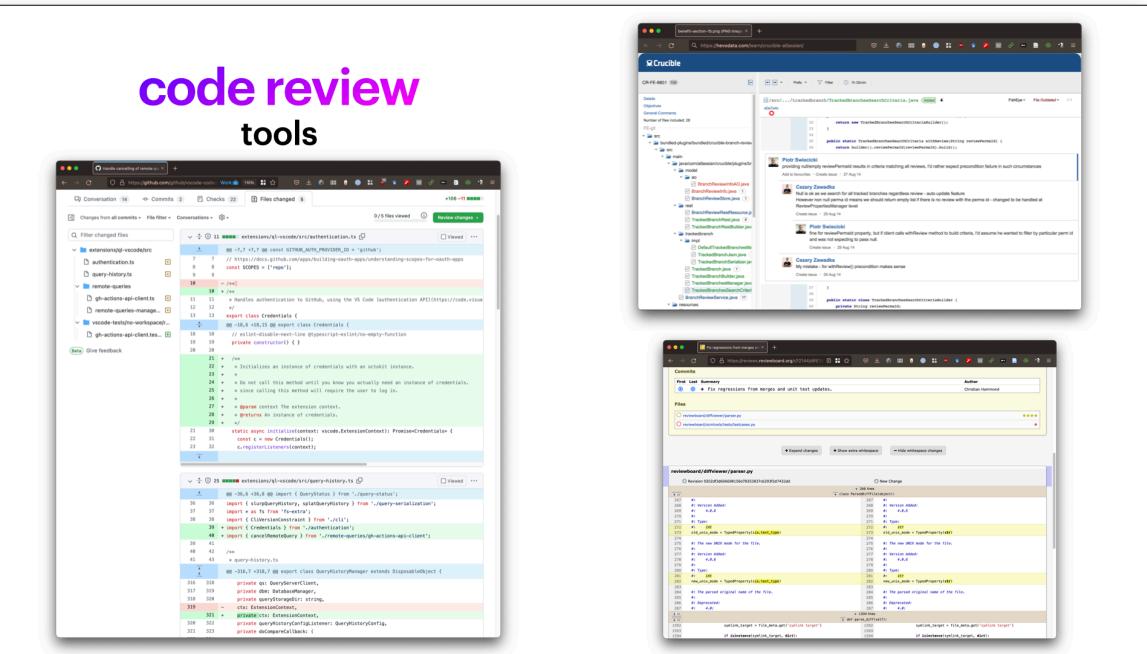
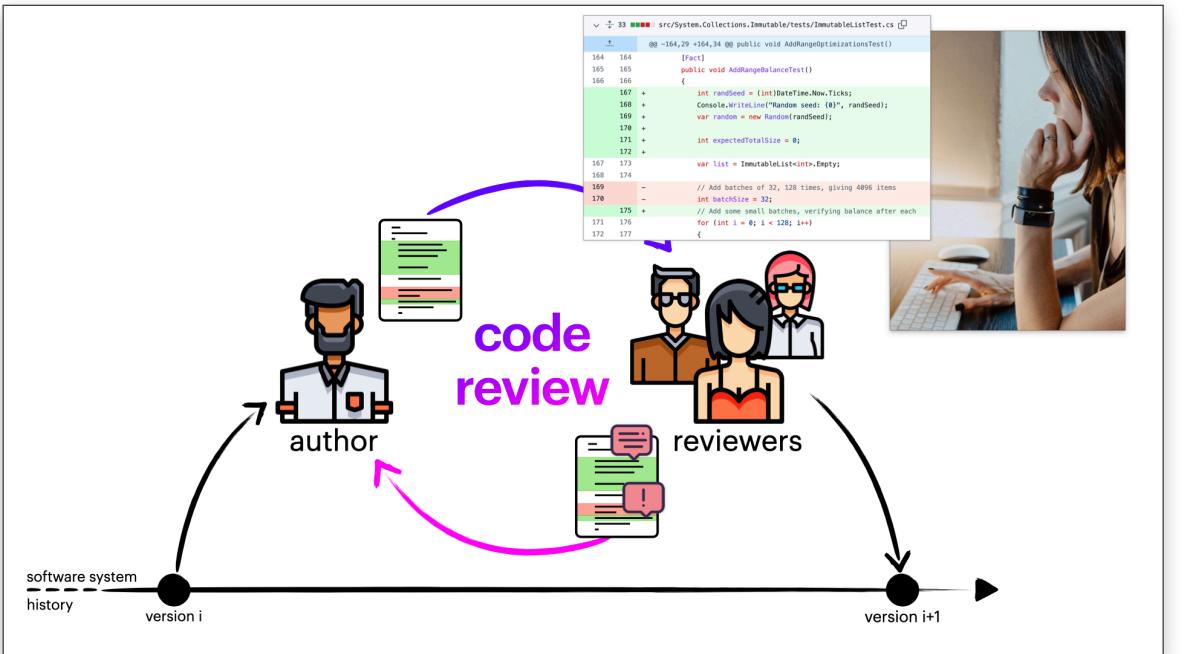
175% more likely to find the bug when in the 1st file shown

A diagram showing five Java files (a.java, b.java, c.java, d.java, e.java) in different orders. Participants are shown reviewing them. A red bug icon is placed on file 'e.java'. A callout bubble says '175% more likely to find the bug when in the 1st file shown'.

takeaways

- 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.

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]

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 illustrates two experimental orders for reviewing code changes. In the 'production code first order', participants review production.java and test.java in that sequence. In the 'test code first order', they review test.java and production.java. The results show that the test code first order is 250% more likely to find the test bug compared to the production code first order.

code review production files

cumulative number of review comments by file position

pull requests with 5 files

A bar chart titled 'pull requests with 5 files' showing the cumulative number of review comments by file position. The x-axis is 'file position' (1st, 2nd, 3rd, 4th, 5th) and the y-axis is 'comments' (0, 500, 1000, 1500). The data shows a decreasing trend:

file position	comments
1st	~1300
2nd	~900
3rd	~700
4th	~500
5th	~400

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

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

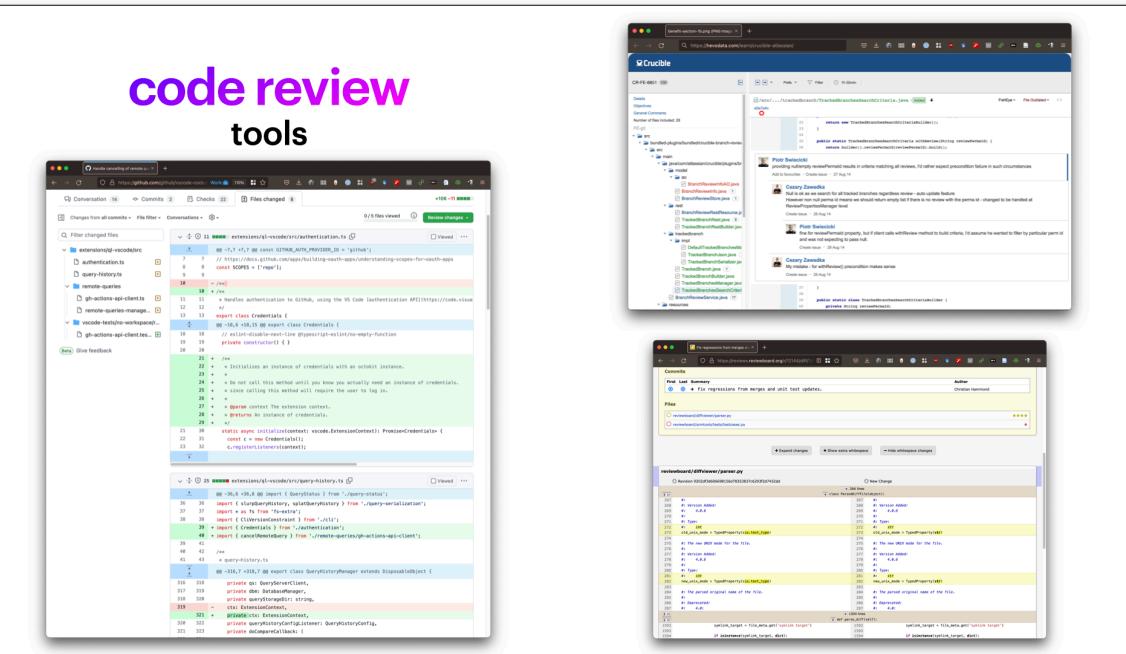
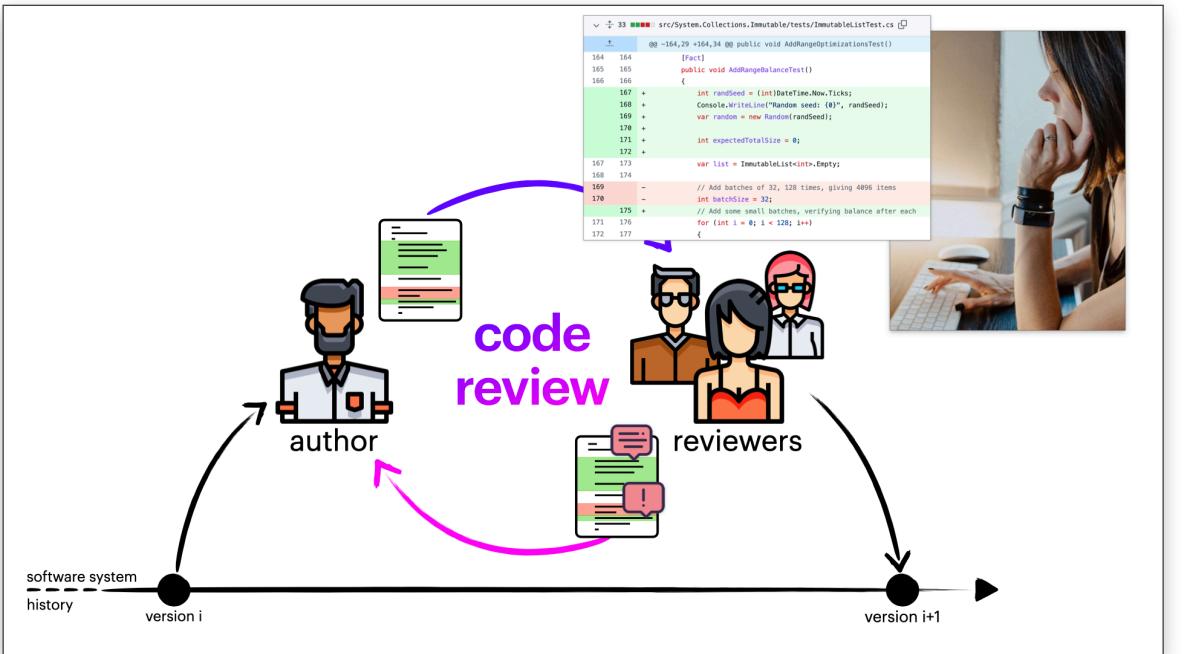
code review production files order experiment

The diagram shows five Java files (a.java, b.java, c.java, d.java, e.java) arranged in a circle. Participants review them in a specific order. A red bug icon is shown near file 'e.java'. A callout box states: '175% more likely to find the bug when in the 1st file shown'.

takeaways

- 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



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 vs test code first order

same likelihood of finding the production bugs
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]

code review production files order experiment

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

takeaways

- 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.

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

