From Code Review to Real-World Insights:

Dissecting Empirical Research in Software Engineering

Alberto Bacchelli
Faculty of Business, Economics and Informatics

University of Zurich[UZH]

zest

alumni


A. Sawant


D. Spadini


L. Pascarella


V. Kovalenko


F. Palomba


L. Di Geronimo


G. Çalikli

present-day


E. Fregnan


L. Braz


P. Wurzel Gonçalves


N. Singh


F. Sovrano


S. Pirouzkhah


P. Rani

amazing collaborators


M. Storey
Uni. of Victoria


G. Gousios
Endor Labs


C. Bird
Microsoft


M. Castelluccio
Mozilla


M. Aniche
Adyen


S. Proksch
TU Delft


V. Hellendoorn
Google

...

Swiss National Science Foundation

University of Zurich[UZH]

First Come First Served: The Impact of File Position on Code Review

Enrico Fregnan
fregnan,@ifi.uzh.ch
University of Zurich
Switzerland

Larissa Braz
larissa@ifi.uzh.ch
University of Zurich
Switzerland

Marco D'Ambros
marco.dambros@usi.ch
CodeLounge at Software Institute
Università della Svizzera Italiana,
Switzerland

Gül Çalıklı
handangul.calikli@glasgow.ac.uk
University of Glasgow
Scotland

Alberto Bacchelli
bacchelli@ifi.uzh.ch
University of Zurich
Switzerland

ABSTRACT

The most popular code review tools (*e.g.*, Gerrit and GitHub) present the files to review sorted in alphabetical order. Could this choice or, more generally, the relative position in which a file is presented bias the outcome of code reviews? We investigate this hypothesis by triangulating complementary evidence in a two-step study.

First, we observe developers' code review activity. We analyze the review comments pertaining to 219,476 Pull Requests (PRs) from 138 popular Java projects on GitHub. We found files shown earlier in a PR to receive more comments than files shown later, also when controlling for possible confounding factors: *e.g.*, the presence of discussion threads or the lines added in a file. Second, we measure the impact of file position on defect finding in code review. Recruiting 106 participants, we conduct an online controlled experiment in which we measure participants' performance in detecting two unrelated defects seeded into two different files. Participants are assigned to one of two treatments in which the position of the defective files is switched. For one type of defect, participants are not affected by its file's position; for the other, they have 64% lower odds to identify it when its file is last as opposed to first. Overall, our findings provide evidence that the relative position in which files are presented has an impact on code reviews' outcome; we discuss these results and implications for tool design and code review.

Preprint: https://doi.org/10.48550/arXiv.2208.04259
Data and Materials: https://doi.org/10.5281/zenodo.6901285

CCS CONCEPTS

• **Software and its engineering** → *Empirical software validation.*

KEYWORDS

Code Review, Controlled Experiment, Cognitive Bias

ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore

ACM ISBN 978-1-4503-9413-0/22/11.
https://doi.org/10.1145/3540250.3549177

ACM Reference Format:
Enrico Fregnan, Larissa Braz, Marco D'Ambros, Gül Çalıklı, and Alberto Bacchelli. 2022. First Come First Served: The Impact of File Position on Code Review. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3540250.3549177

1 INTRODUCTION

Code review is a popular software engineering practice where developers manually inspect the code written by a peer [7, 40]. Code review aims to find defects [11], improve software quality [3, 12], and transfer knowledge [7, 48]. Over the years, code review has evolved from a formal strictly-regulated process [22] into a less strict practice. Contemporary code reviewing is informal, asynchronous, change-based, and supported by tools [9, 10, 46, 48].

The tools used to conduct code reviews share many similarities [11]. In particular, the vast majority of tools (including the popular Gerrit [28] and GitHub [29]) present the changes to review as a list/sequence of diff hunks [25] grouped by the file they belong to. Tools sort these files alphabetically, therefore the changes to a file named `org/Controller.java` are always presented before those to a file named `org/Model.java`. Could this choice or, more generally, the relative position in which a file is presented influence the outcome of code review?

This hypothesis seems to be supported by at least two factors. First, most developers tend to start their reviews in the order presented by the review tool [12]. Second, code review is a cognitively demanding task [8] whose outcome might be influenced by cognitive factors [42, 51] also related to the position of the file. For example, developers may be influenced by *attention decrement* (a decrease in attention when exposed to a list of elements [6]) or may deplete their *working memory capacity* (the memory for short-term storage during ongoing tasks [61]) near the end of longer reviews.

In this paper, we set to investigate this hypothesis. We do this by triangulating complementary evidence in a two-step study.

In the first step, we focus on the relation between file position and reviewers' activity. We collect and analyze 219,476 Pull Requests (PRs) from 138 GitHub open-source Java projects and investigate whether the position in which a file is presented in a PR is associated with the number of comments the file receives. In fact, the number of comments can be used to approximate reviewers' effectiveness

First Come First Served: The Impact of File Position on Code Review

Enrico Fregnan
fregnan,@ifi.uzh.ch
University of Zurich
Switzerland

Larissa Braz
larissa@ifi.uzh.ch
University of Zurich
Switzerland

Marco D'Ambros
marco.dambros@usi.ch
CodeLounge at Software Institute
Università della Svizzera Italiana,
Switzerland

Gül Çalıklı
handangul.calikli@glasgow.ac.uk
University of Glasgow
Scotland

Alberto Bacchelli
bacchelli@ifi.uzh.ch
University of Zurich
Switzerland

ABSTRACT

The most popular code review tools (*e.g.*, Gerrit and GitHub) present the files to review sorted in alphabetical order. Could this choice or, more generally, the relative position in which a file is presented bias the outcome of code reviews? We investigate this hypothesis by triangulating complementary evidence in a two-step study.

First, we observe developers' code review activity. We analyze the review comments pertaining to 219,476 Pull Requests (PRs) from 138 popular Java projects on GitHub. We found files shown earlier in a PR to receive more comments than files shown later, also when controlling for possible confounding factors: *e.g.*, the presence of discussion threads or the lines added in a file. Second, we measure the impact of file position on defect finding in code review. Recruiting 106 participants, we conduct an online controlled experiment in which we measure participants' performance in detecting two unrelated defects seeded into two different files. Participants are assigned to one of two treatments in which the position of the defective files is switched. For one type of defect, participants are not affected by its file's position; for the other, they have 64% lower odds to identify it when its file is last as opposed to first. Overall, our findings provide evidence that the relative position in which files are presented has an impact on code reviews' outcome; we discuss these results and implications for tool design and code review.

Preprint: https://doi.org/10.48550/arXiv.2208.04259
Data and Materials: https://doi.org/10.5281/zenodo.6901285

CCS CONCEPTS

• **Software and its engineering** → *Empirical software validation.*

KEYWORDS

Code Review, Controlled Experiment, Cognitive Bias

ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore

ACM ISBN 978-1-4503-9413-0/22/11.
https://doi.org/10.1145/3540250.3549177

ACM Reference Format:
Enrico Fregnan, Larissa Braz, Marco D'Ambros, Gül Çalıklı, and Alberto Bacchelli. 2022. First Come First Served: The Impact of File Position on Code Review. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3540250.3549177

1 INTRODUCTION

Code review is a popular software engineering practice where developers manually inspect the code written by a peer [7, 40]. Code review aims to find defects [11], improve software quality [3, 12], and transfer knowledge [7, 48]. Over the years, code review has evolved from a formal strictly-regulated process [22] into a less strict practice. Contemporary code reviewing is informal, asynchronous, change-based, and supported by tools [9, 10, 46, 48].

The tools used to conduct code reviews share many similarities [11]. In particular, the vast majority of tools (including the popular Gerrit [28] and GitHub [29]) present the changes to review as a list/sequence of diff hunks [25] grouped by the file they belong to. Tools sort these files alphabetically, therefore the changes to a file named `org/Controller.java` are always presented before those to a file named `org/Model.java`. Could this choice or, more generally, the relative position in which a file is presented influence the outcome of code review?

This hypothesis seems to be supported by at least two factors. First, most developers tend to start their reviews in the order presented by the review tool [12]. Second, code review is a cognitively demanding task [8] whose outcome might be influenced by cognitive factors [42, 51] also related to the position of the file. For example, developers may be influenced by *attention decrement* (a decrease in attention when exposed to a list of elements [6]) or may deplete their *working memory capacity* (the memory for short-term storage during ongoing tasks [61]) near the end of longer reviews.

In this paper, we set to investigate this hypothesis. We do this by triangulating complementary evidence in a two-step study.

In the first step, we focus on the relation between file position and reviewers' activity. We collect and analyze 219,476 Pull Requests (PRs) from 138 GitHub open-source Java projects and investigate whether the position in which a file is presented in a PR is associated with the number of comments the file receives. In fact, the number of comments can be used to approximate reviewers' effectiveness

"The quality of this paper is such that I would add it to the list of papers that I give to students I work with to show them how research should be carried out and written up."

-- Reviewer 3

ACM SIGSOFT
Distinguished Paper Award
ESEC/FSE 2022

E. Fregnan
zest

L. Braz
zest

M. D'Ambros
CodeLounge@SI

G. Çalikli
U. of Glasgow

Pros and Cons of Track Changes.docx - Microsoft Word

File | Home | Insert | Page Layout | References | Mailings | Review | View

Spelling & Grammar | Research | Thesaurus | Word Count — Proofing
Translate | Language — Language
New Comment | Delete | Previous | Next — Comments
Track Changes | Final: Show Markup | Show Markup | Reviewing Pane — Tracking
Accept | Reject | Previous | Next | Compare | Block Authors | Restrict Editing — Protect

Accept and Move to Next

Accept and Move to Next

Accept the current change and move to the next proposed change.

Click the arrow to accept many changes at once.

Some pros and cons of 'track-changes' feedback on work returned to students electronically

Despite the fact that 'track-changes' is normally used in one-to-one editing and feedback (for example on draft theses, dissertations, reports and so on) it seems likely that 'track-changes' feedback is already well on the way towards replacing 'handwritten comments on students' work in assessment in general. This short discussion is about using the 'track-changes' function in word-processing software to give students feedback when marking their work. ~~This is normally when tutors use the 'track changes' facilities to return to students their original word-processed assignments, duly edited with feedback comments which appear on-screen in another colour.~~ The level of feedback can range from comments providing simple qualitative overall feedback on the whole document or on selected paragraphs or sentences, to very detailed feedback on *individual words or phrases*. This kind of feedback remains very valuable for large-scale work (essays, dissertations, long reports, drafts of articles for publication and so on).

The other side of 'track-changes' is where ~~deletions~~, additions, replaced words or phrases can be suggested, and the original author can accept or reject each change in turn, working

Formatted: Heading 1, Space Before: 0 pt, After: 0 pt, Line spacing: single

Formatted: Font: (Default) Arial, 11

Comment [DA1]:

Formatted: Font: (Default) Arial, 11

Formatted: Font: (Default) Arial, 11 pt, Highlight

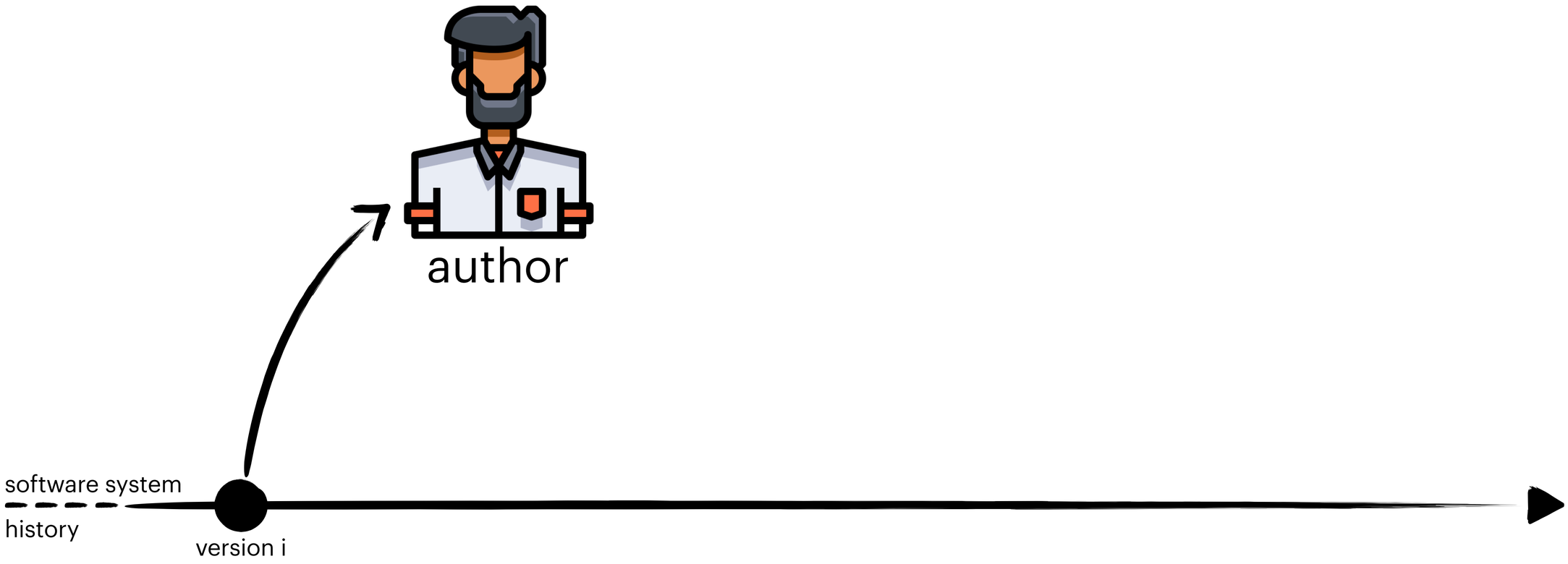
Formatted: Font: (Default) Arial, 11

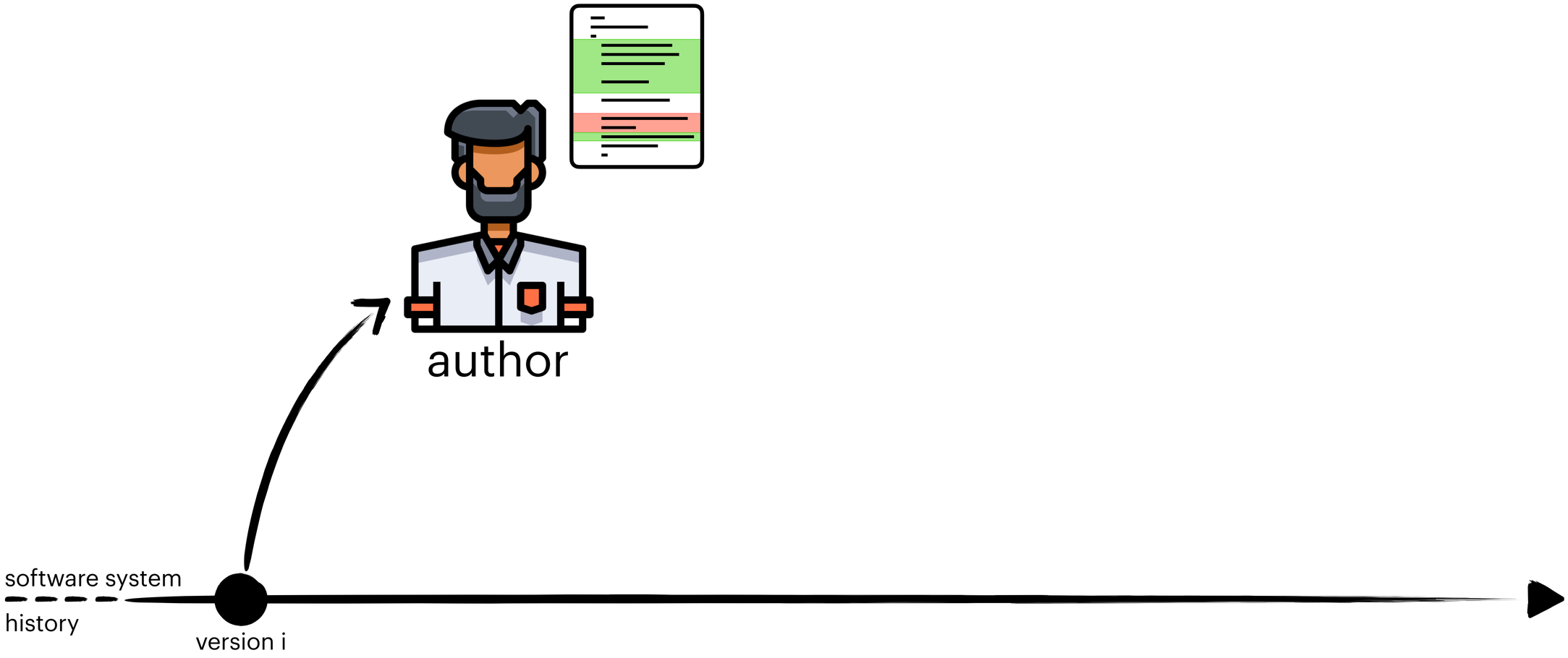
Formatted: Font: (Default) Arial, 11 pt, Italic

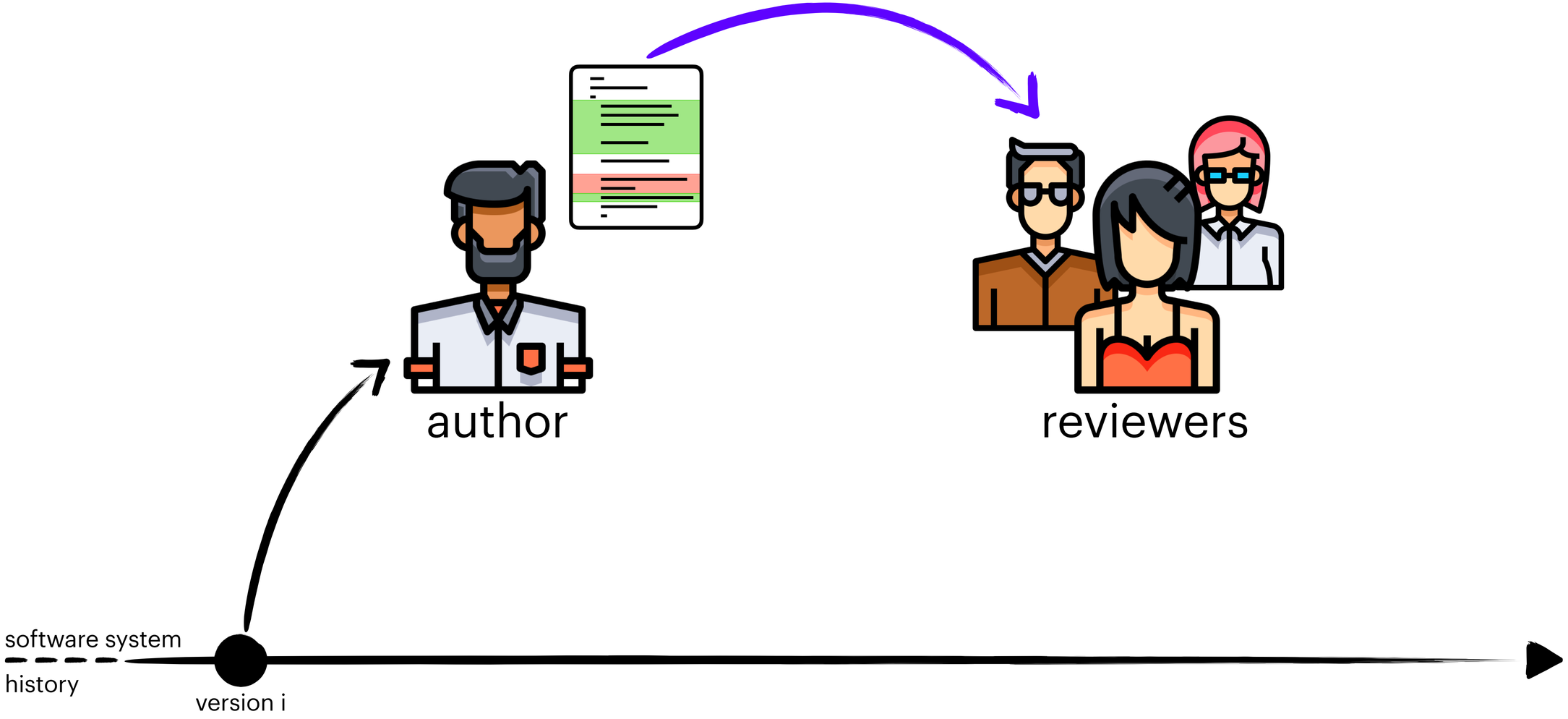
Formatted: Font: (Default) Arial, 11

Comment [DA2]:

Formatted: Font: (Default) Arial, 11

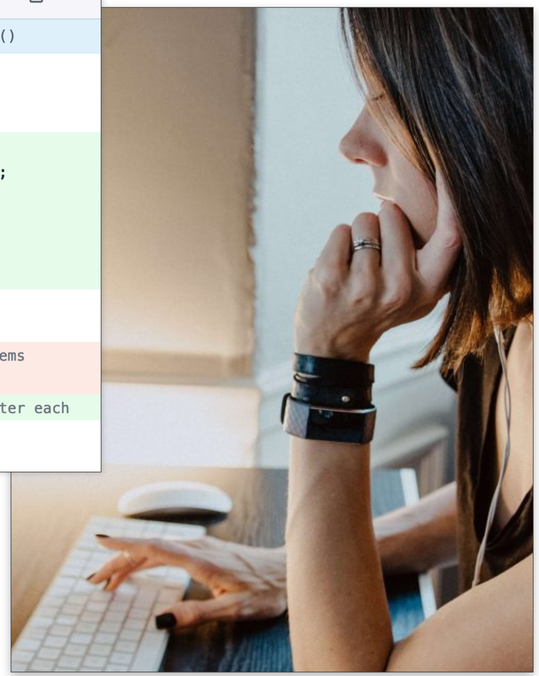

author
software system
history
version i

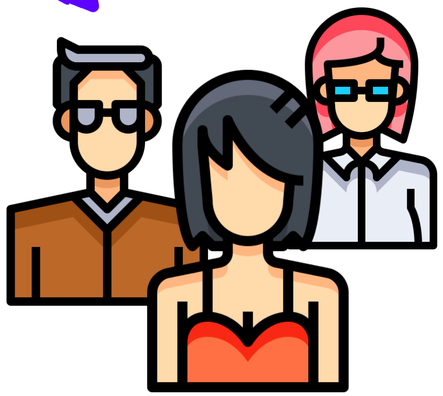

author
software system
history
version i


author
reviewers
software system
history
version i

33 src/System.Collections.Immutable/tests/ImmutableListTest.cs

```diff
@@ -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             {
```





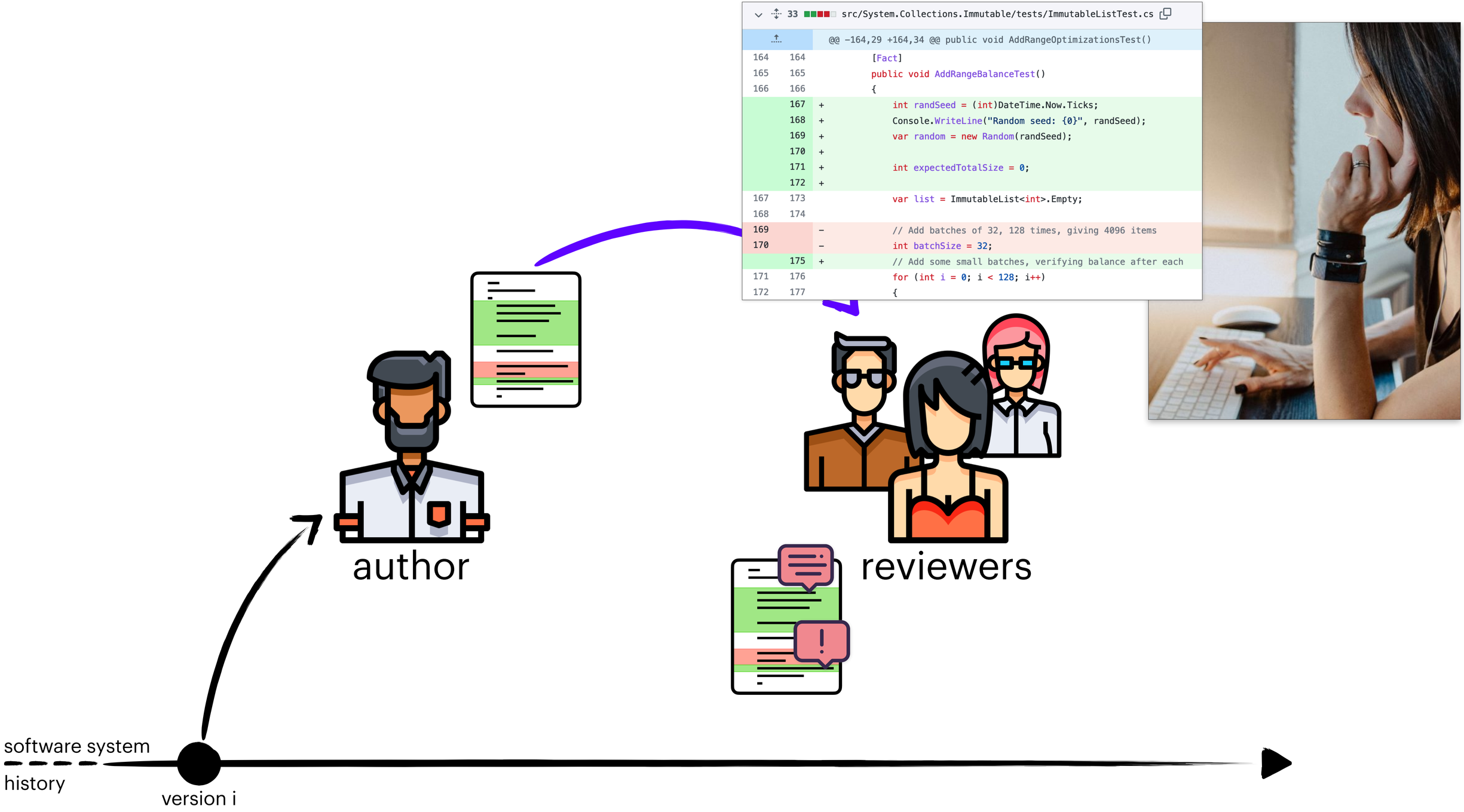
author

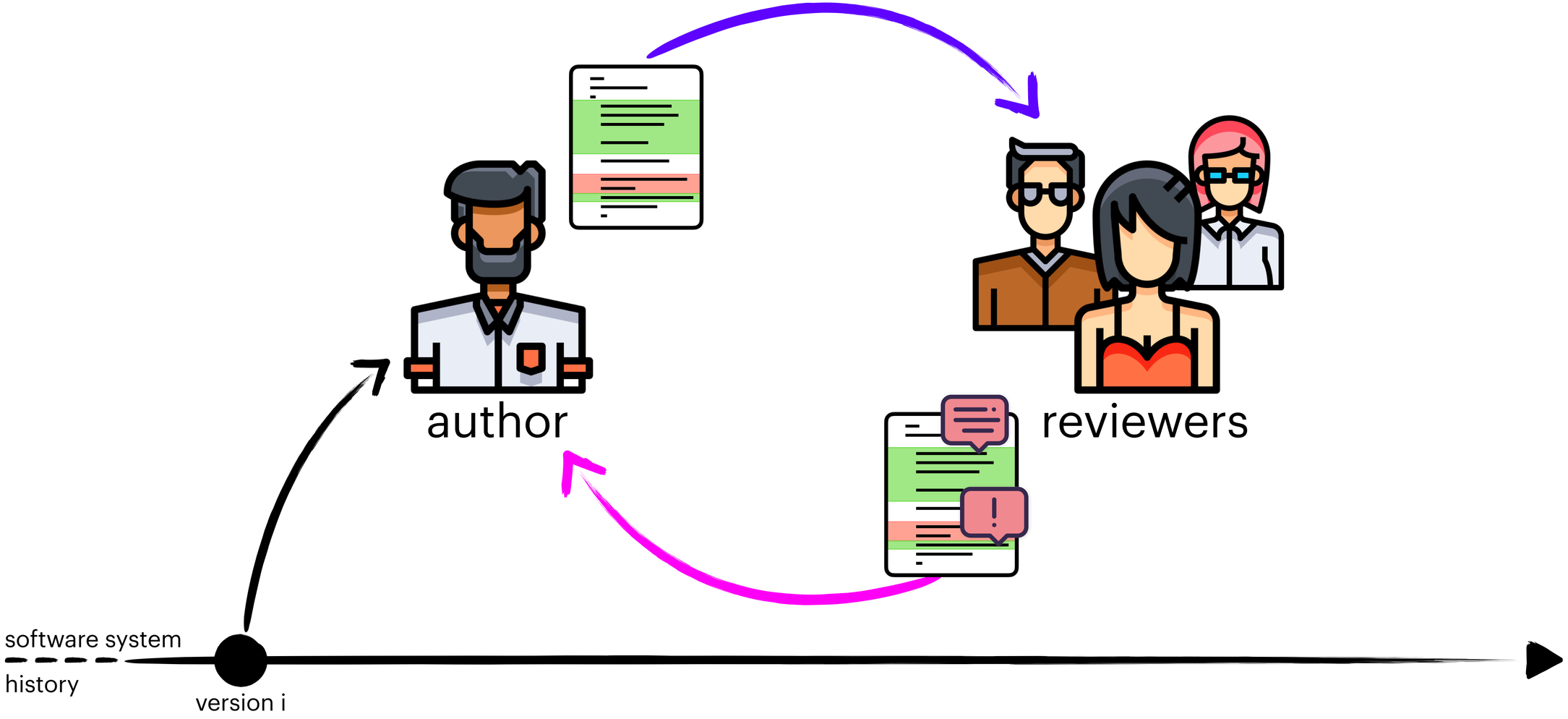
reviewers

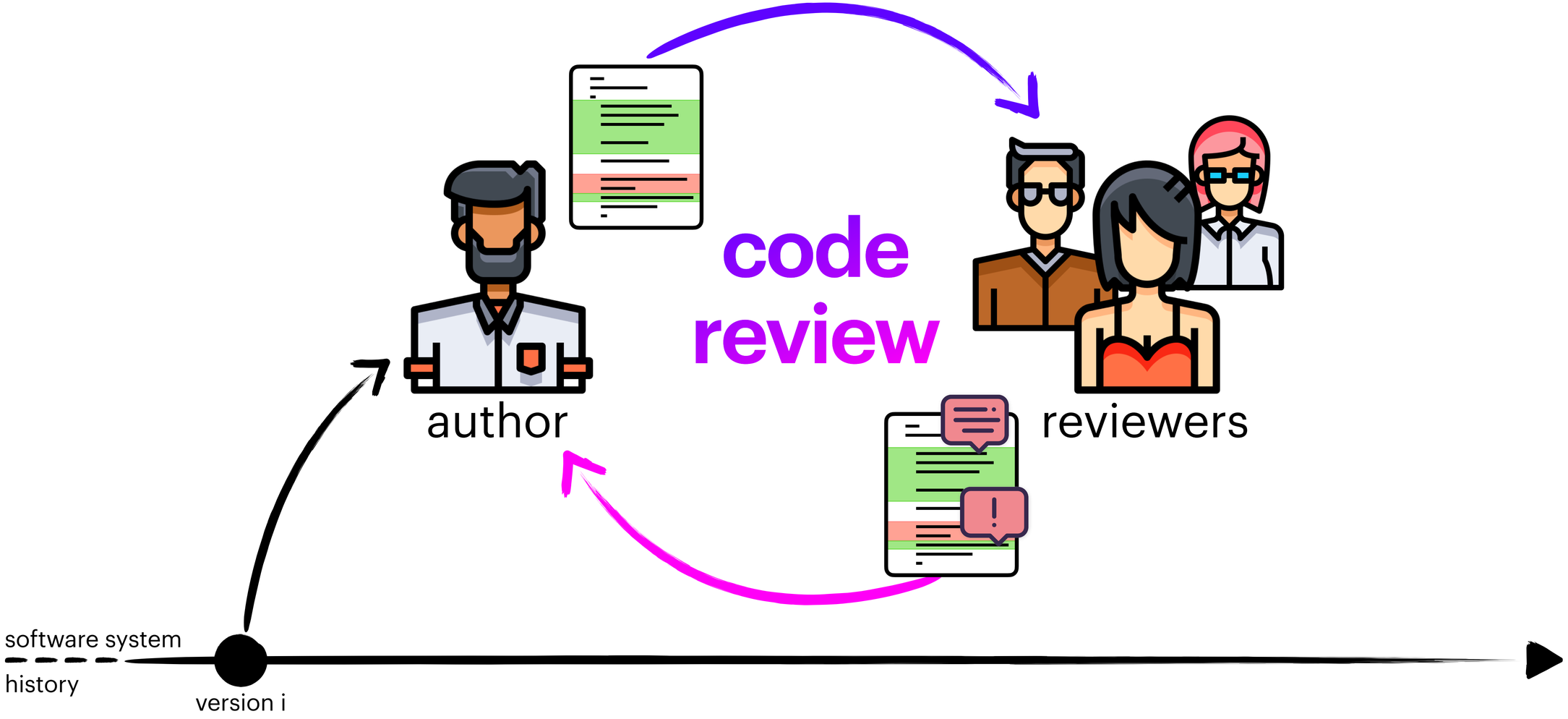
software system

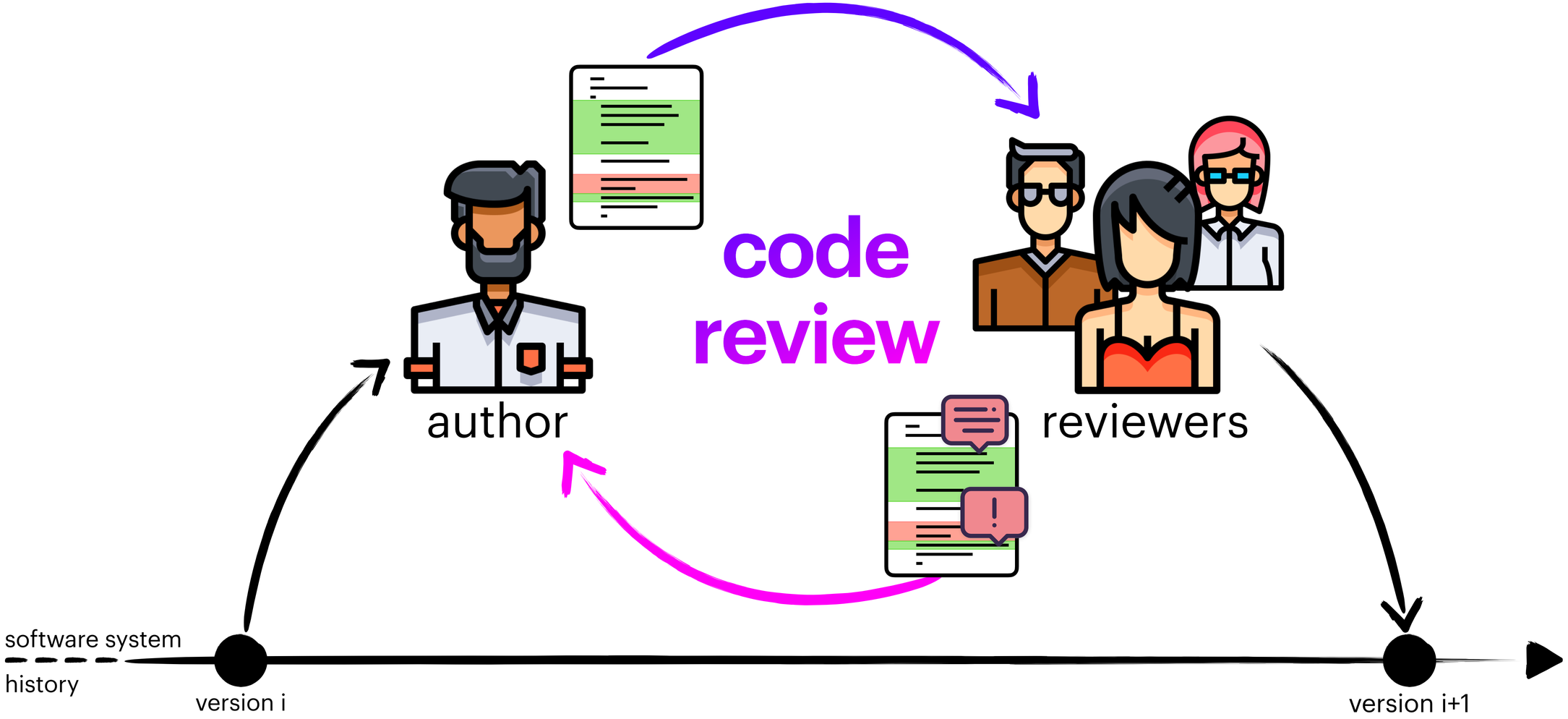
history

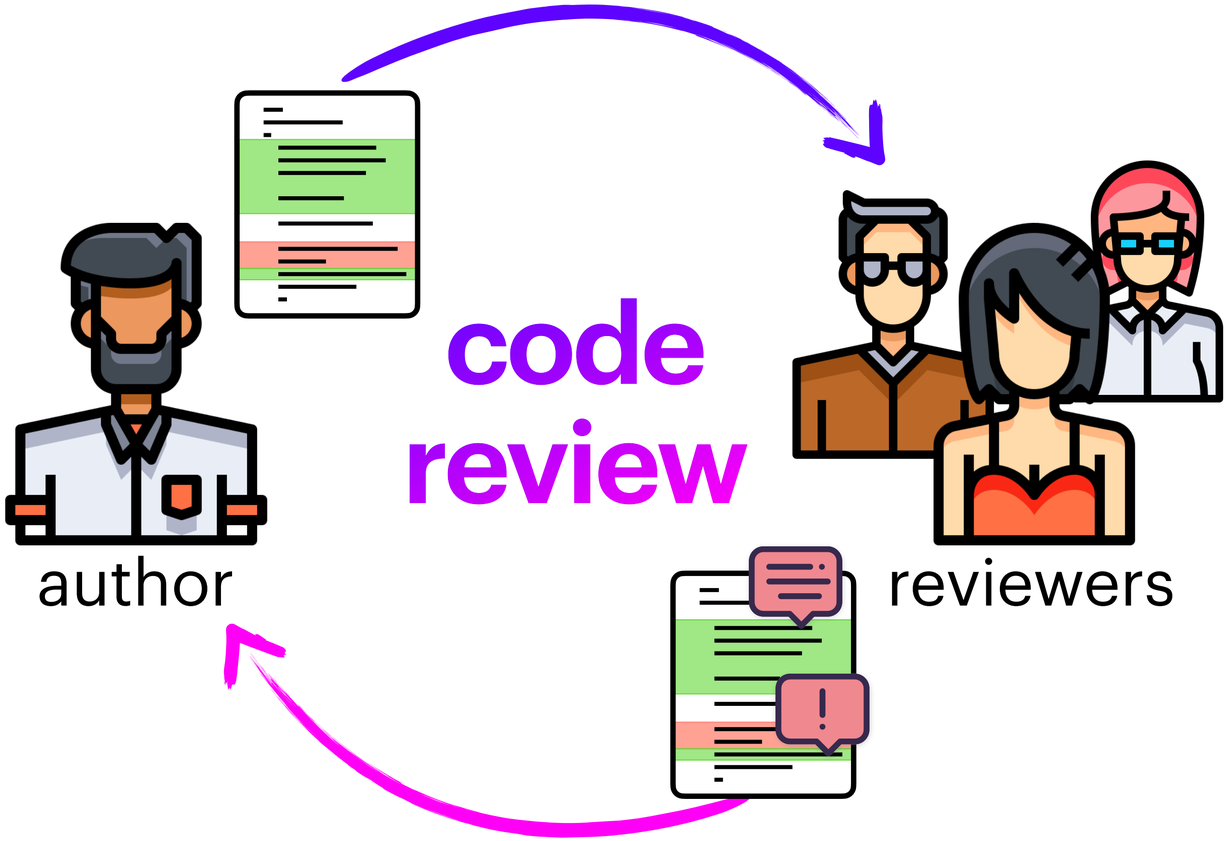
version i


33 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 {
author
reviewers
software system
history
version i


author
reviewers
software system
history
version i


code
review
author
reviewers
software system
history
version i

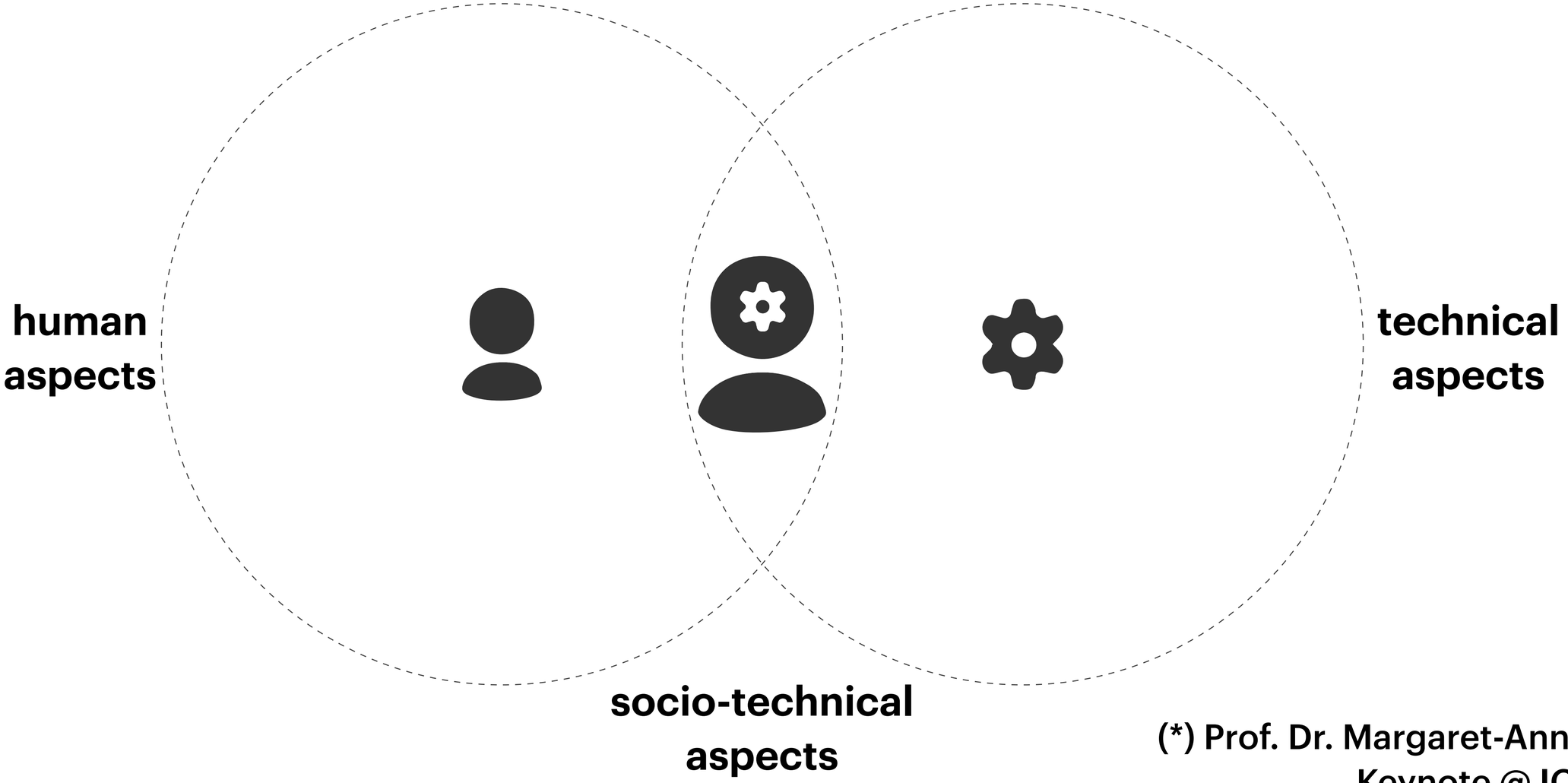

code
review
author
reviewers
software system
history
version i
version i+1



extremely widespread

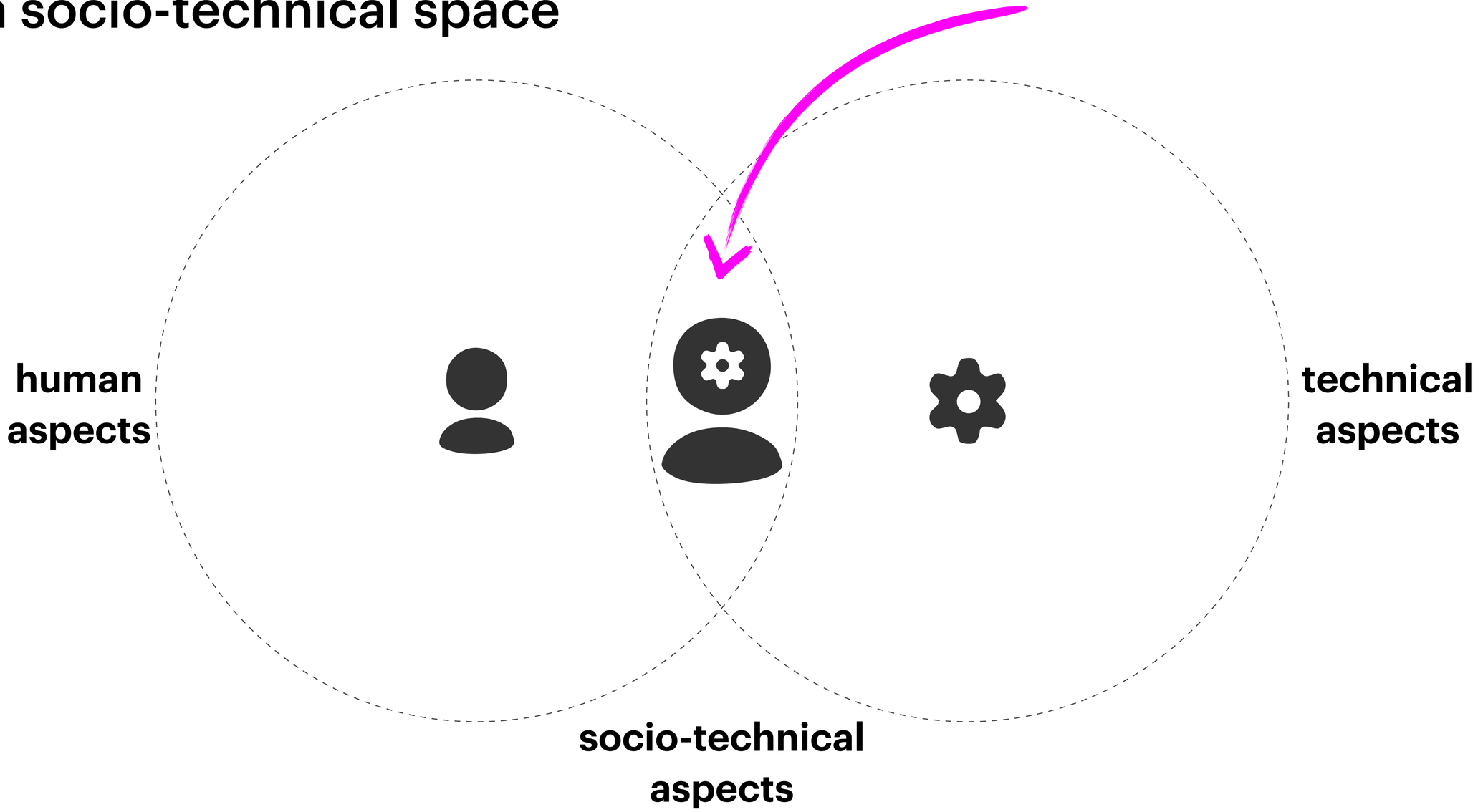
- About 70% of developers spend 2 to 8 hours a week reviewing code.
  [Stack Overflow Dev Survey 2019]
- In 2021, 170M pull requests have been merged in GitHub.
  [The 2021 State of the Octoverse]
- Most (possibly all) code changes at Google, Meta, and Microsoft are reviewed.

software engineering
is a socio-technical space(*)

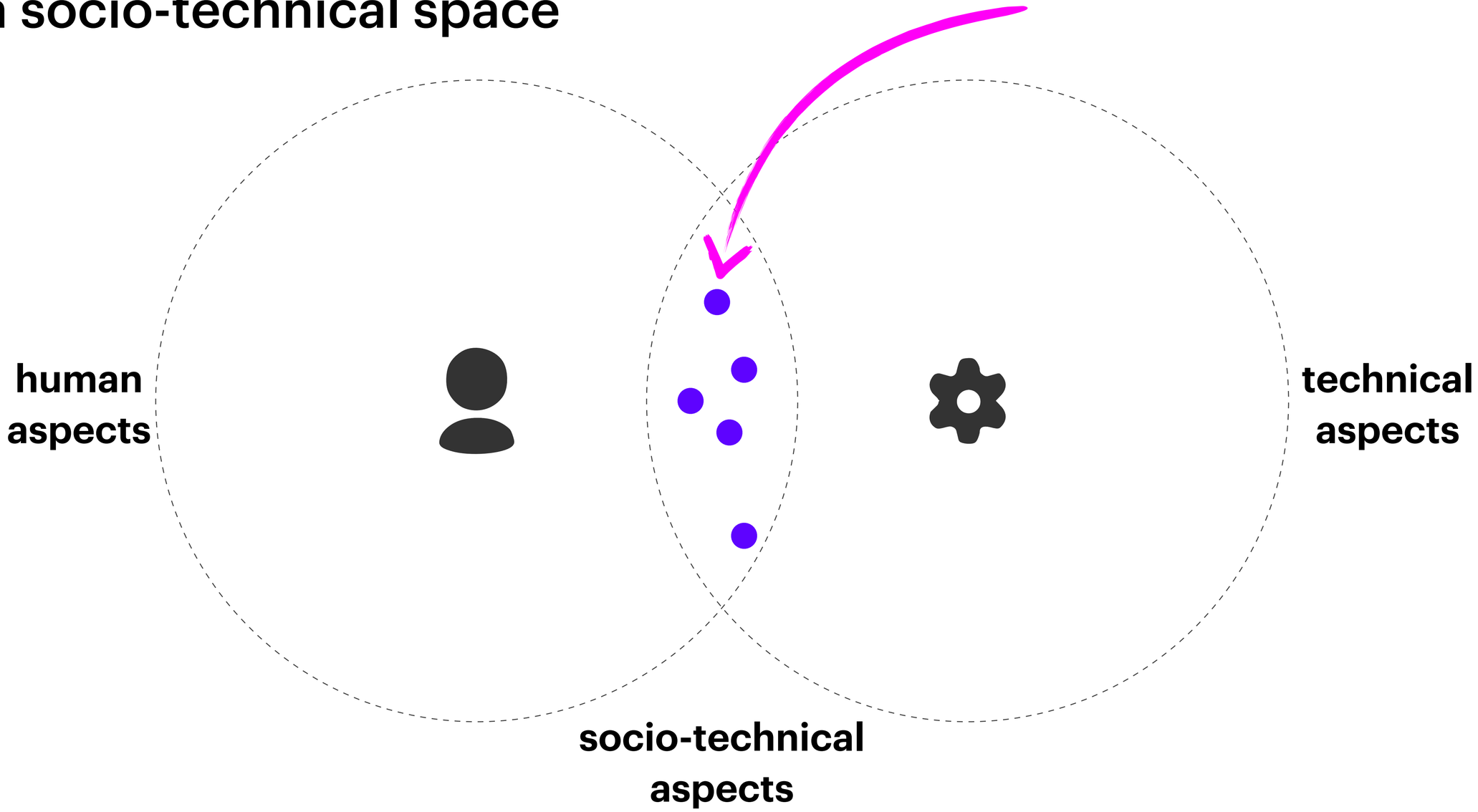


(*) Prof. Dr. Margaret-Anne Storey
Keynote @ ICSE 2018

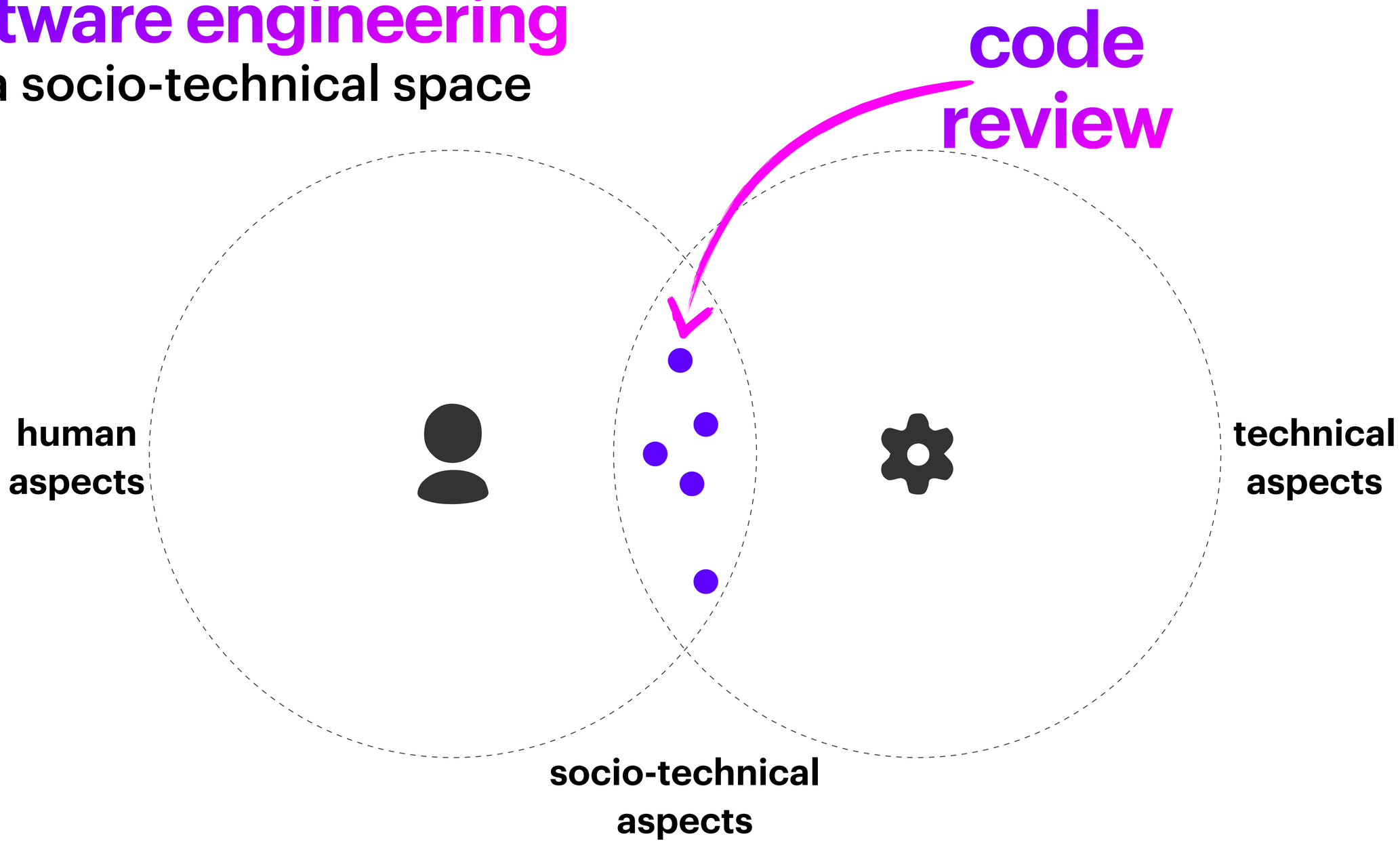
software engineering
is a socio-technical space

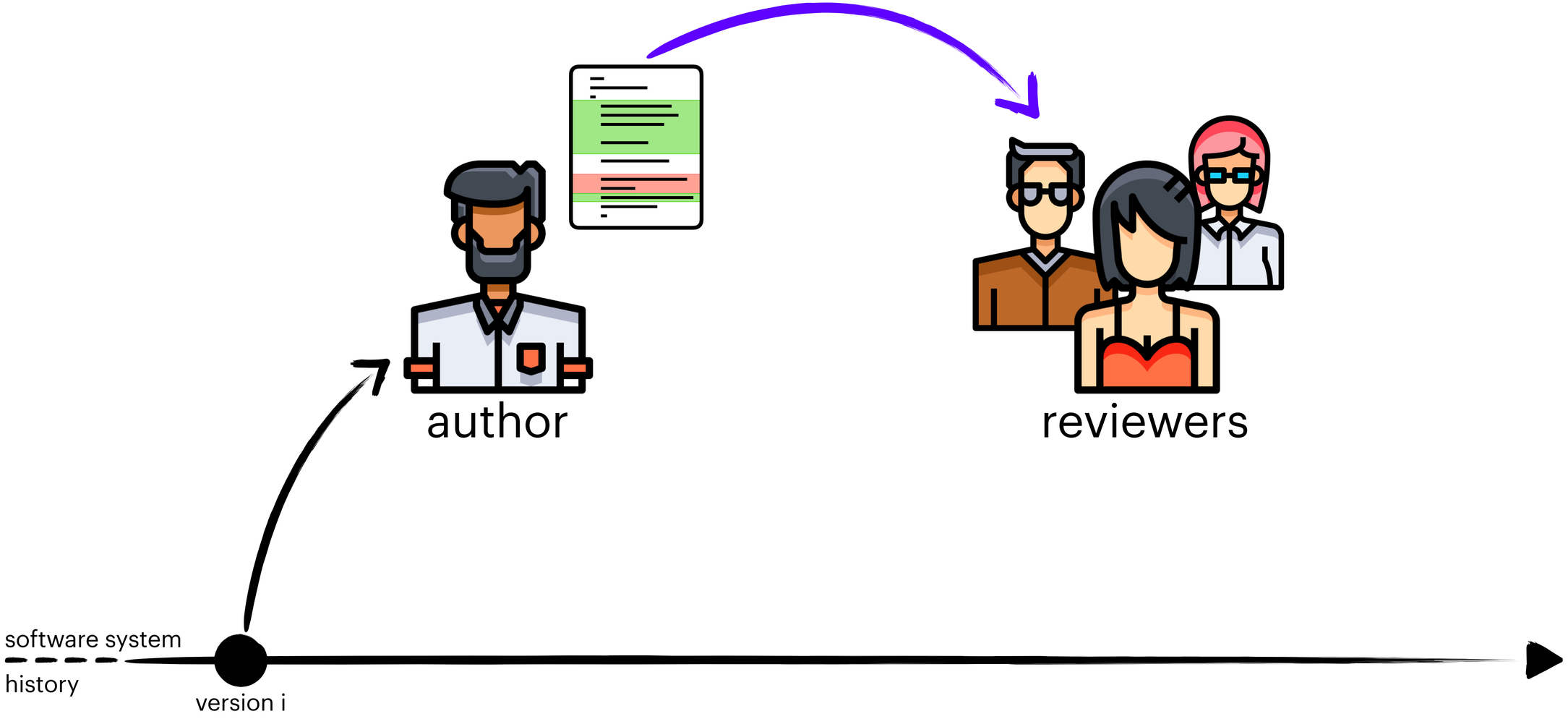


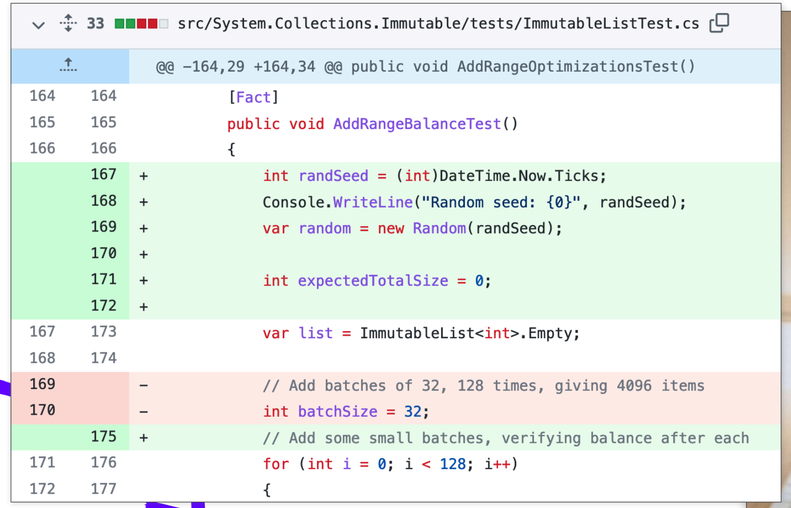
software engineering
is a socio-technical space



software engineering
is a socio-technical space

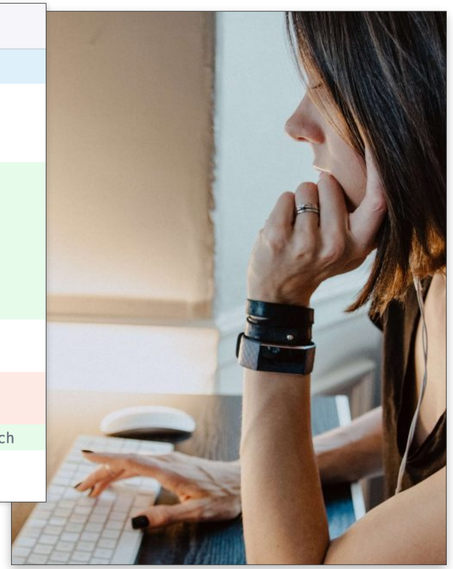



author
reviewers
software system
history
version i



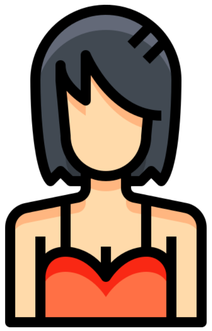
33 src/System.Collections.Immutable/tests/ImmutableListTest.cs

```diff
@@ -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              {
```

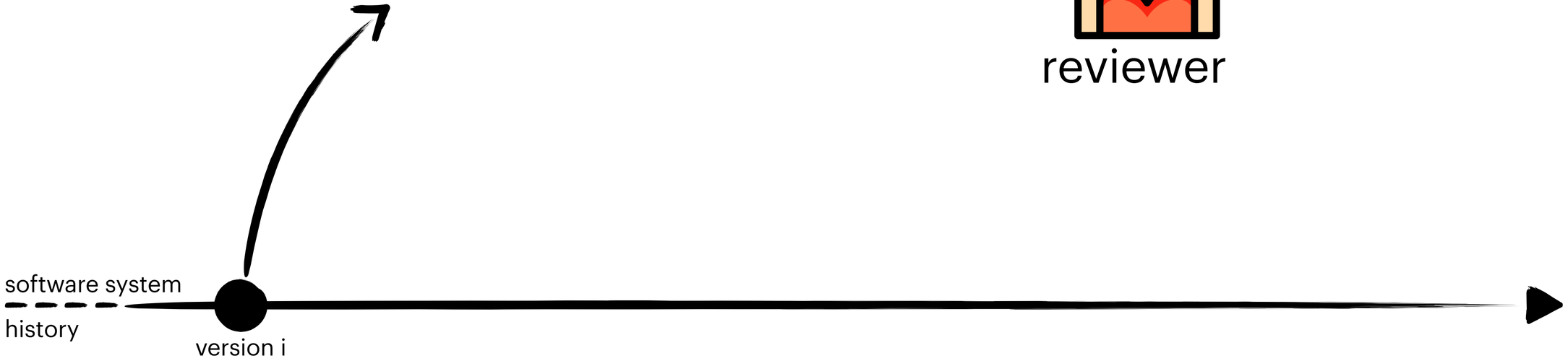


individual phases

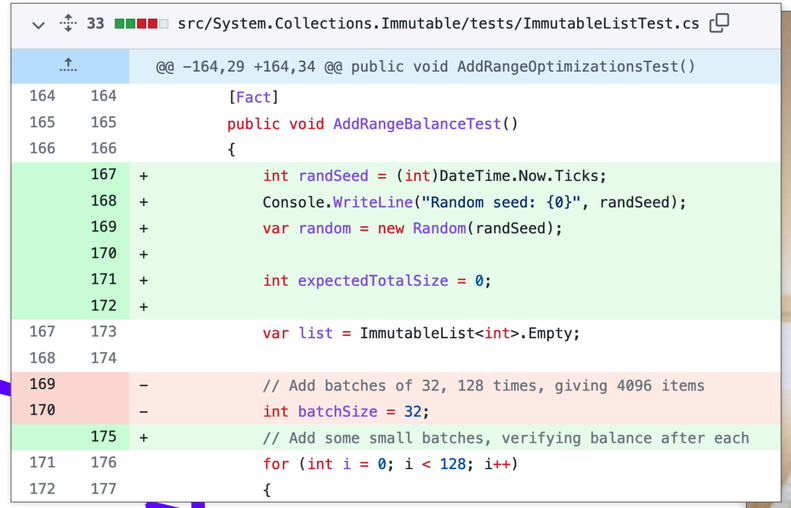
cognitive-aspects heavy



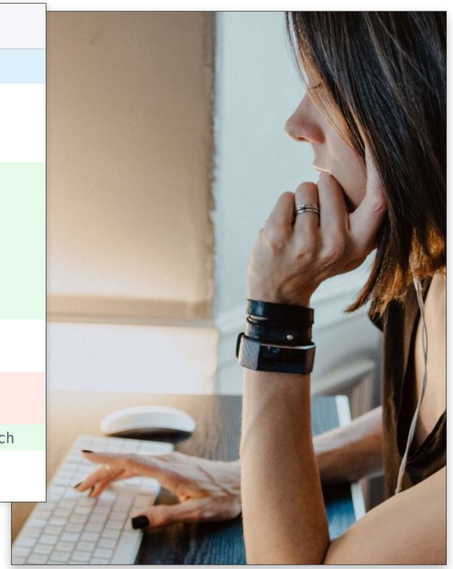
reviewer

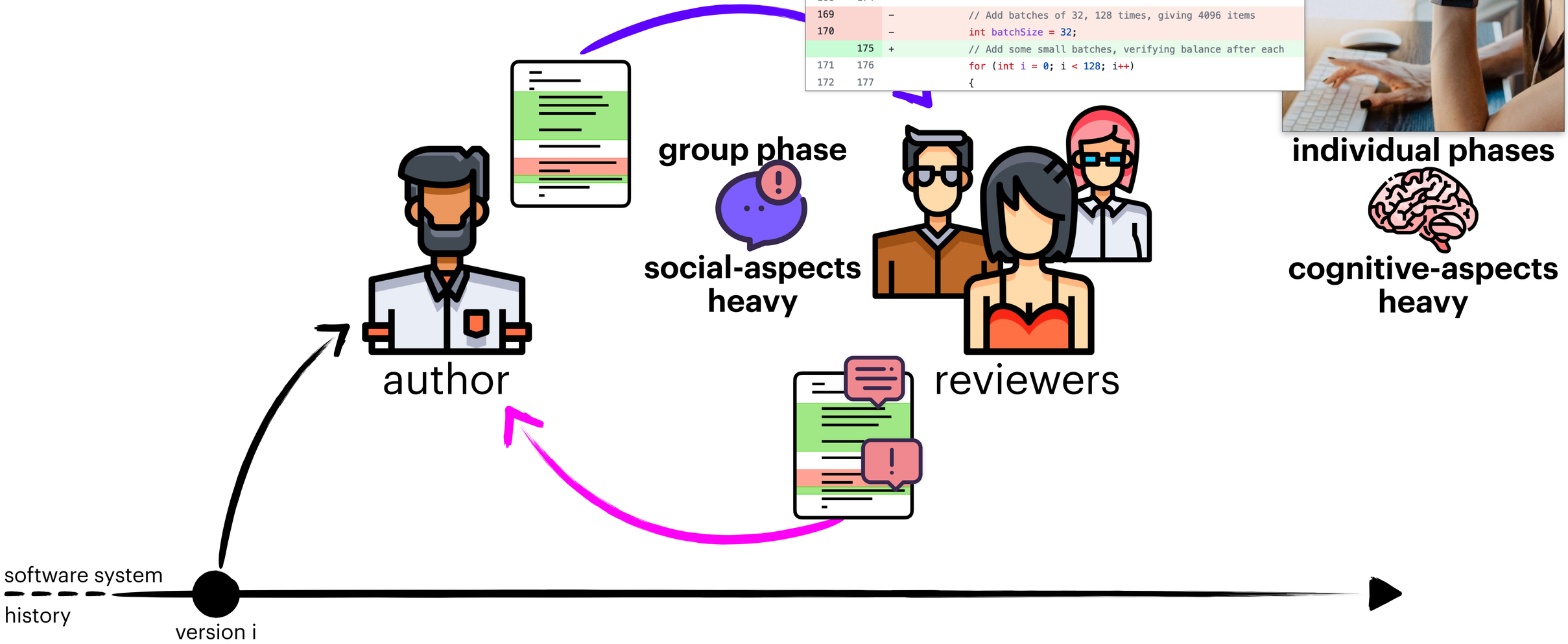


The Dual Nature
Of Code Review

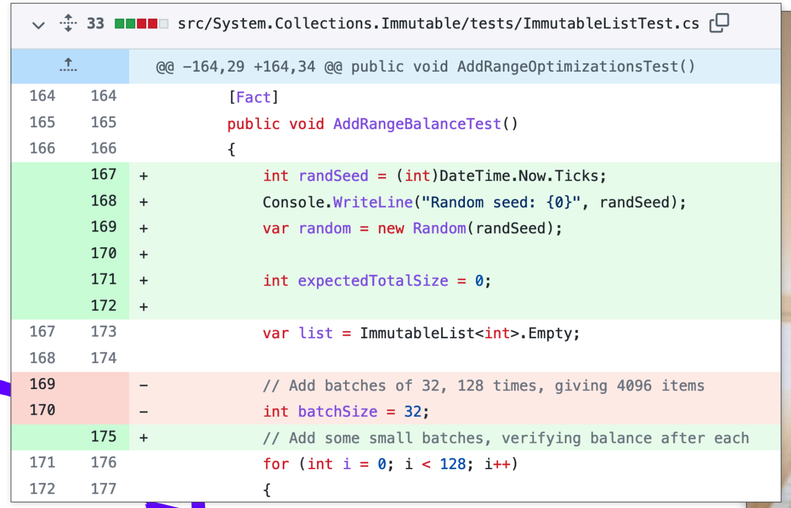


```
33 src/System.Collections.Immutable/tests/ImmutableListTest.cs
@@ -164,29 +164,34 @@ public void AddRangeOptimizationsTest()
164  164          [Fact]
165  165          public void AddRangeBalanceTest()
166  166          {
     167 +            int randSeed = (int)DateTime.Now.Ticks;
     168 +            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              {
```



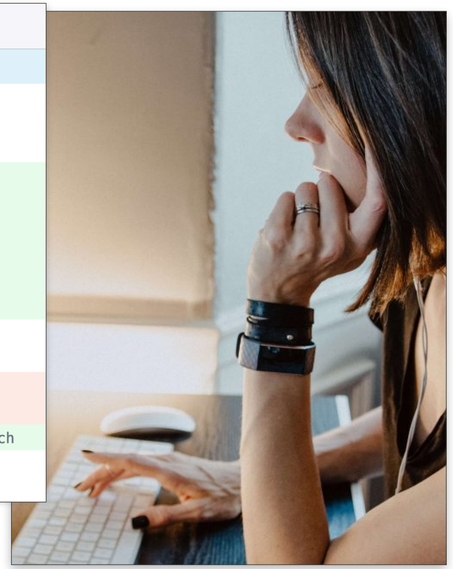


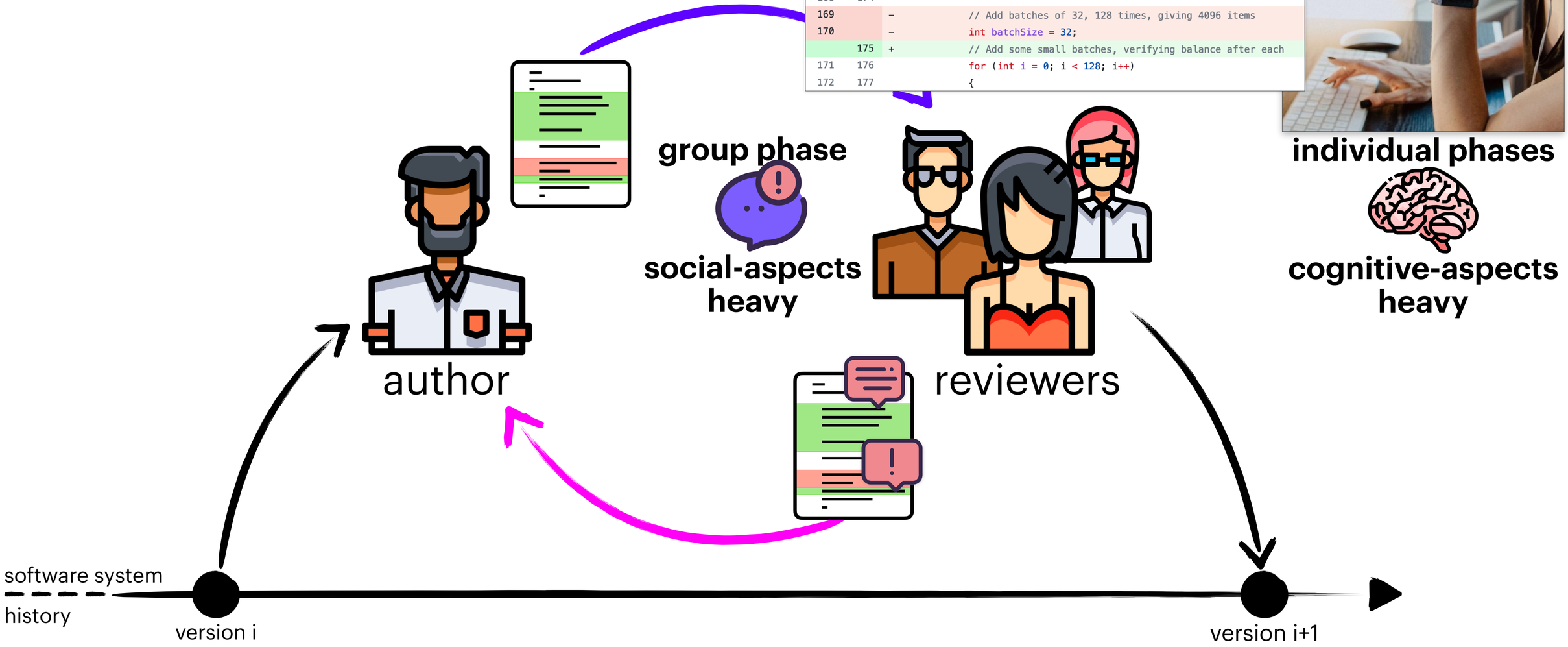
The Dual Nature
Of Code Review

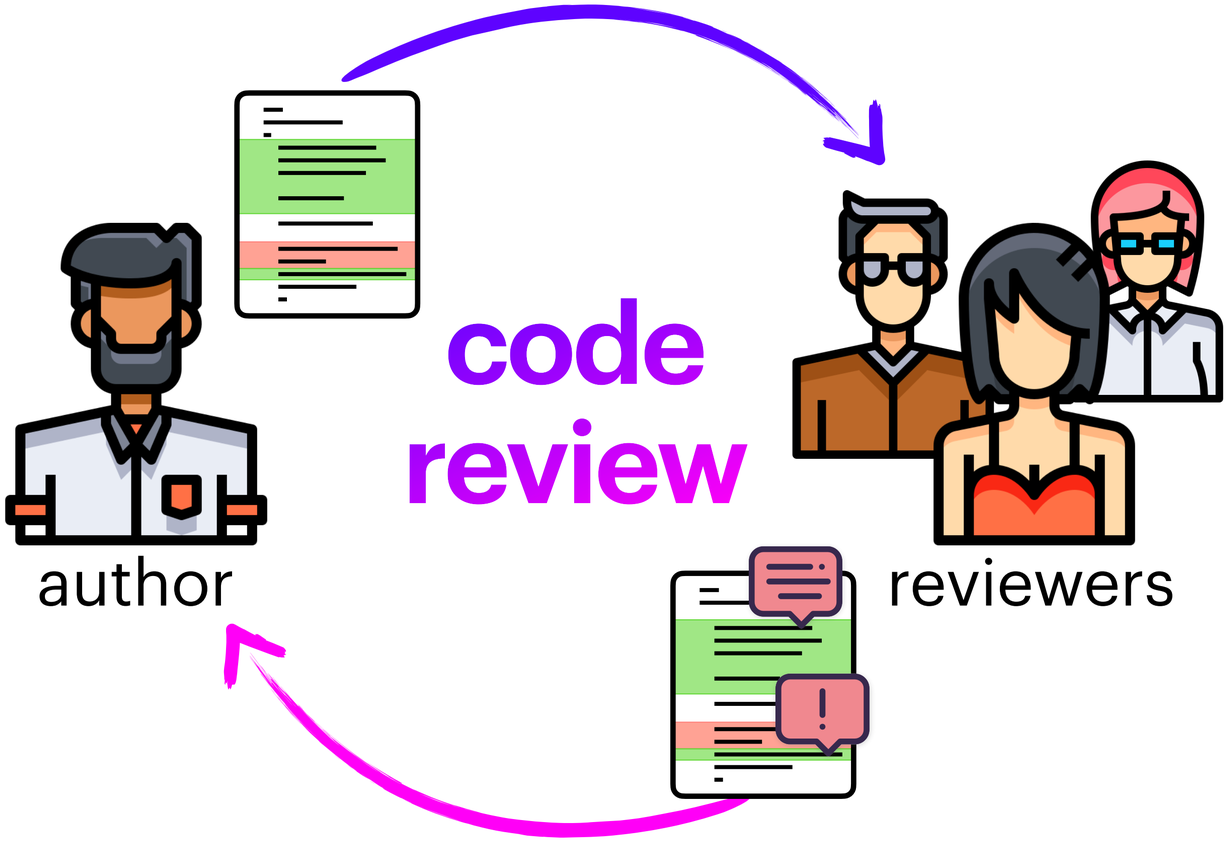


33 src/System.Collections.Immutable/tests/ImmutableListTest.cs

```
@@ -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              {
```







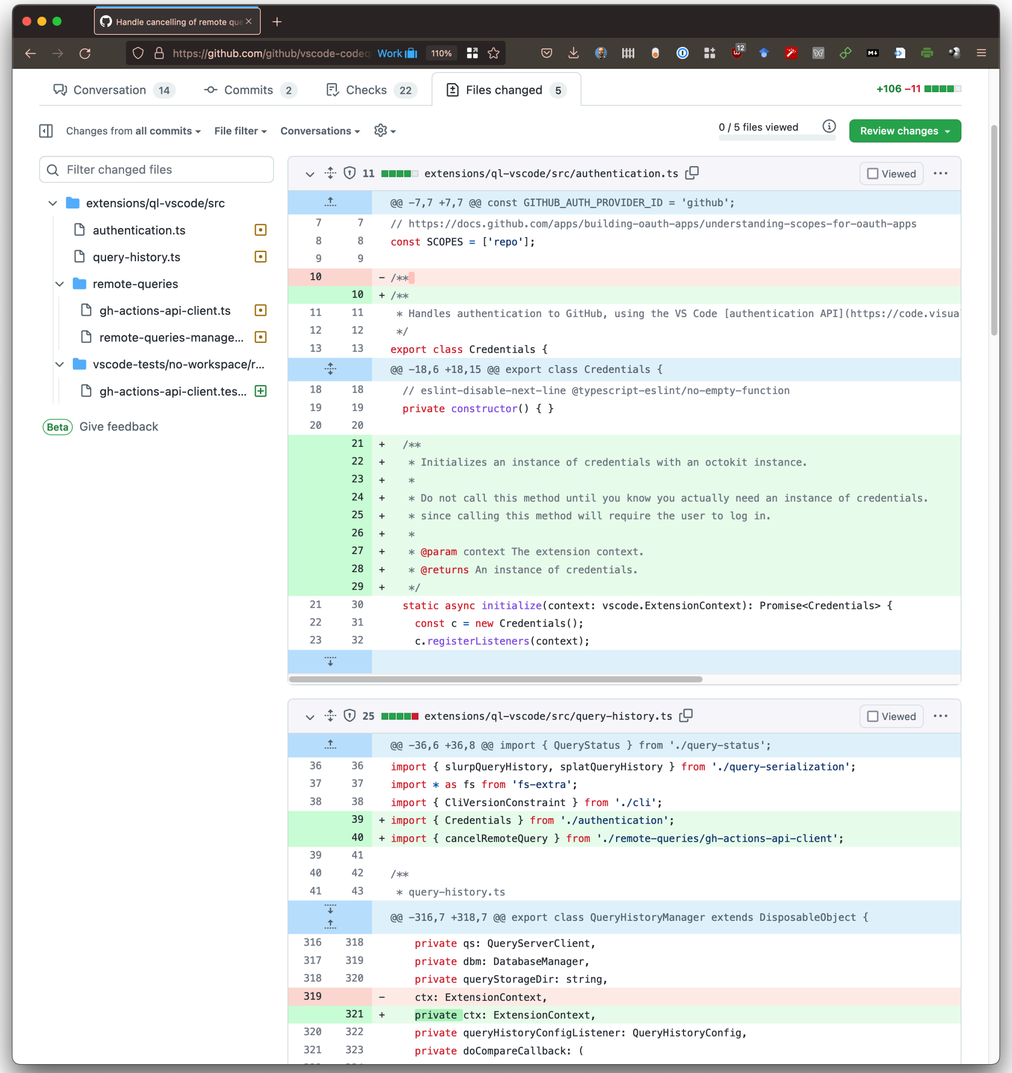
zest's take on code review

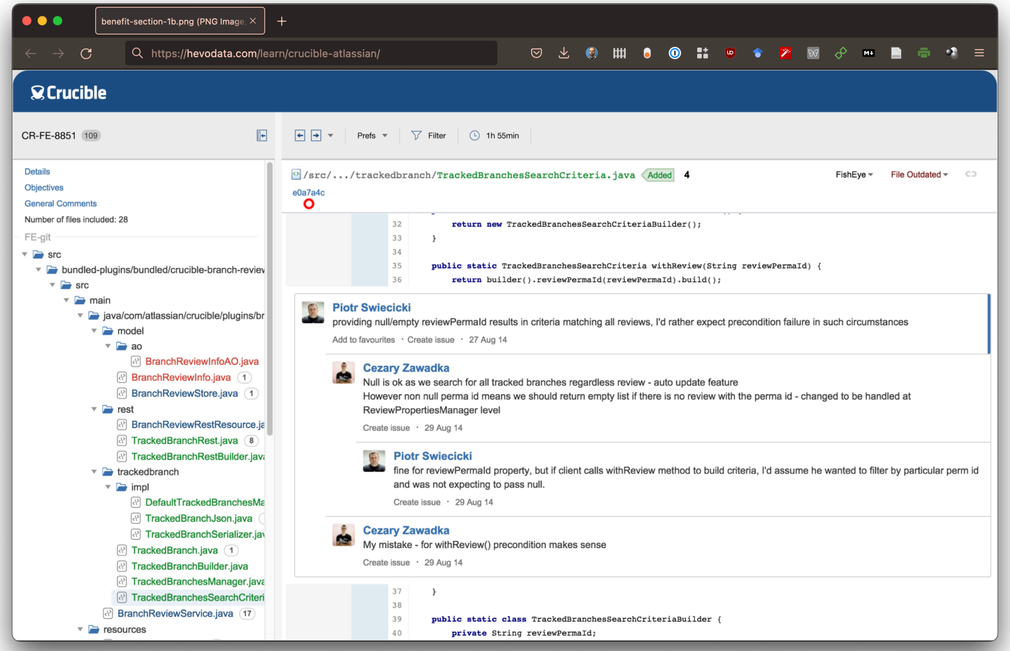
- relevance, simplicity, innovation, & interdisciplinarity
- focus on:
  - tooling for people
  - developers' behavior
    - cognitive aspects
    - collaboration
  - education & training

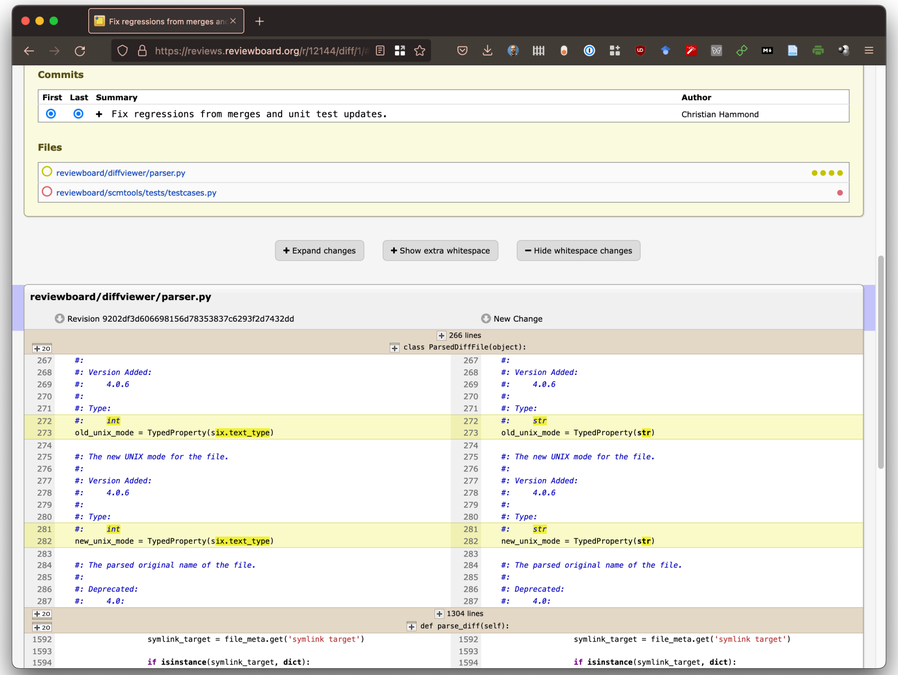
z urich
e mpirical
s oftware engineering
t eam

code review

tools







code review

Files are ordered linearly and alphabetically.

Could this choice have an effect on code review's results?

How would <u>you</u> test this hypothesis?

Handle cancelling of remote que

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-actions
```

an observational study

Add bot filter, co-author logic by

https://github.com/JetBrains-Research/bus-factor-explorer/pu

Merged **Add bot filter, co-author logic** #1

Changes from all commits · File filter · Conversations · 0 / 6 files viewed · Review in codespace · Review changes

Filter changed files

- src/main/kotlin/org/jetbrains/rese...
  - calculation
    - BusFactor.kt
    - BusFactorComputationCon...
  - mappers
    - UserMapper.kt
  - processors
    - CommitProcessor.kt
  - service
    - github
      - GitHubClient.kt
    - task
      - ComputeBusFactorJob.kt

3

...main/kotlin/org/jetbrains/research/ictl/riskypatterns/servic...

Viewed

```
63  63          executionEnvironment.logFile.log(repositoryCloned)
64  64
    65  +       val bots = gitHubClient.loadBots(payload.owner, payload.repo)
```

egorklimov marked this conversation as resolved. Hide resolved

egorklimov on May 25 · Collaborator

Remove it from the time measurement scope, please.

Reply...

```
65  66          val started = System.currentTimeMillis()
66      -       val busFactor = BusFactor(File(executionEnvironment.gitDir, ".git"))
    67  +       val busFactor = BusFactor(File(executionEnvironment.gitDir, ".git"), bots)
67  68          val tree = busFactor.calculate(payload.fullName)
68  69          val ended = System.currentTimeMillis()
69  70
```

Pull requests · microsoft/vscode

m/microsoft/vscode/pulls?page=3&q=is%3Ap

ed September 2023

ations of the same set of scopes ✓ — Approved September 2023

ractive window is open ✓ — 1 — 3 — ed September 2023

settings UI ✓ — 2 — ved September 2023

escriptions ✓ — 1 — ved September 2023

scription ✓ — ved September 2023

s cache for improved performance ✗ — 1 — 3 — aft

tension ✓ — 3 — Approved September 2023

ttings UI ✓ — ved September 2023

og message ✓ git — ved September 2023

king ✓ — proved September 2023

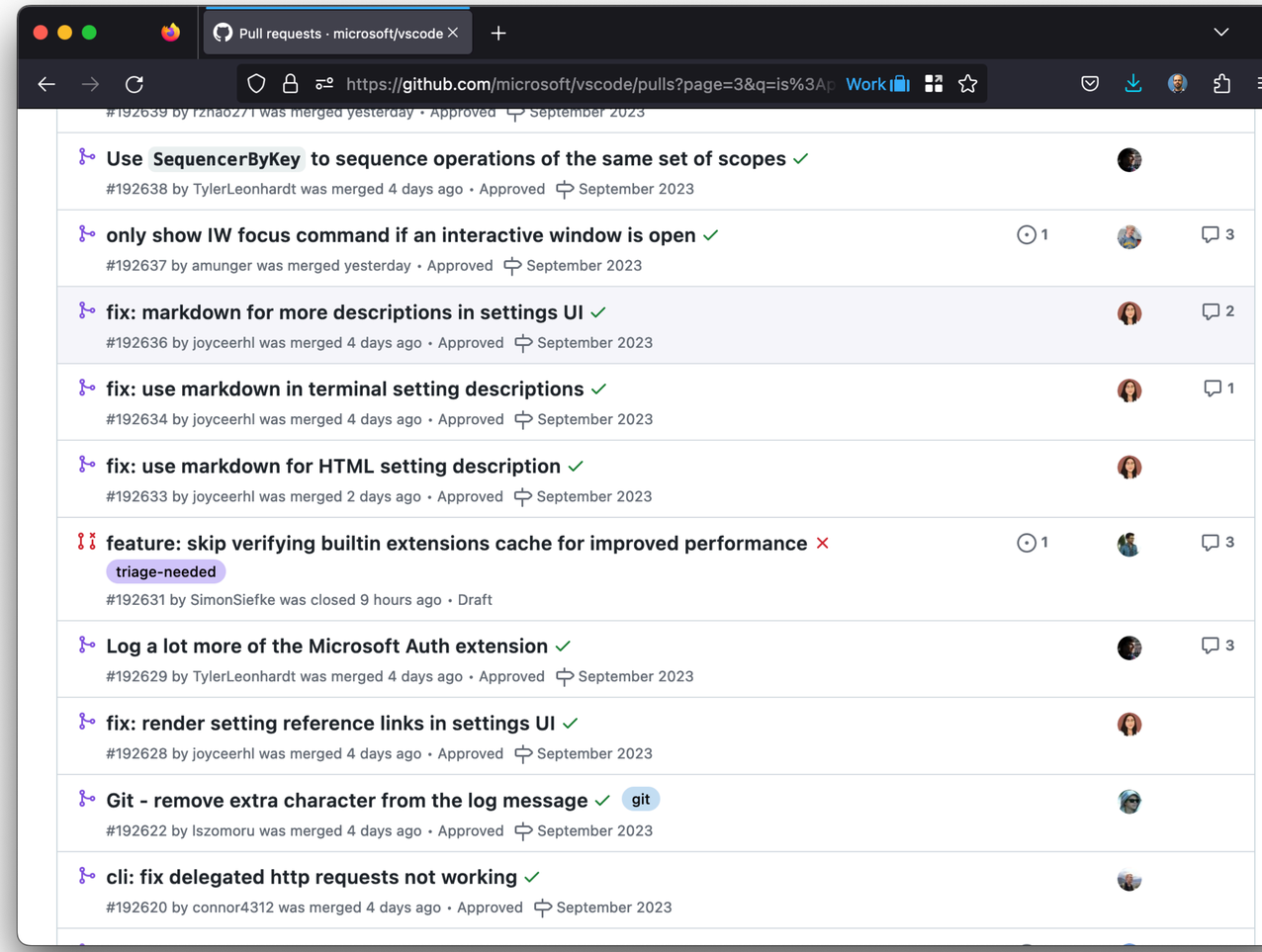
an observational study

Comments as **proxy** for reviewers' activity

What (else) could affect the number of comments on files (i.e., **confounding factors**)?

- change size
- test files
- number of participants
- bots
- threads
- … ?
  - big data matters

M. D'Ambros
CodeLounge@SI



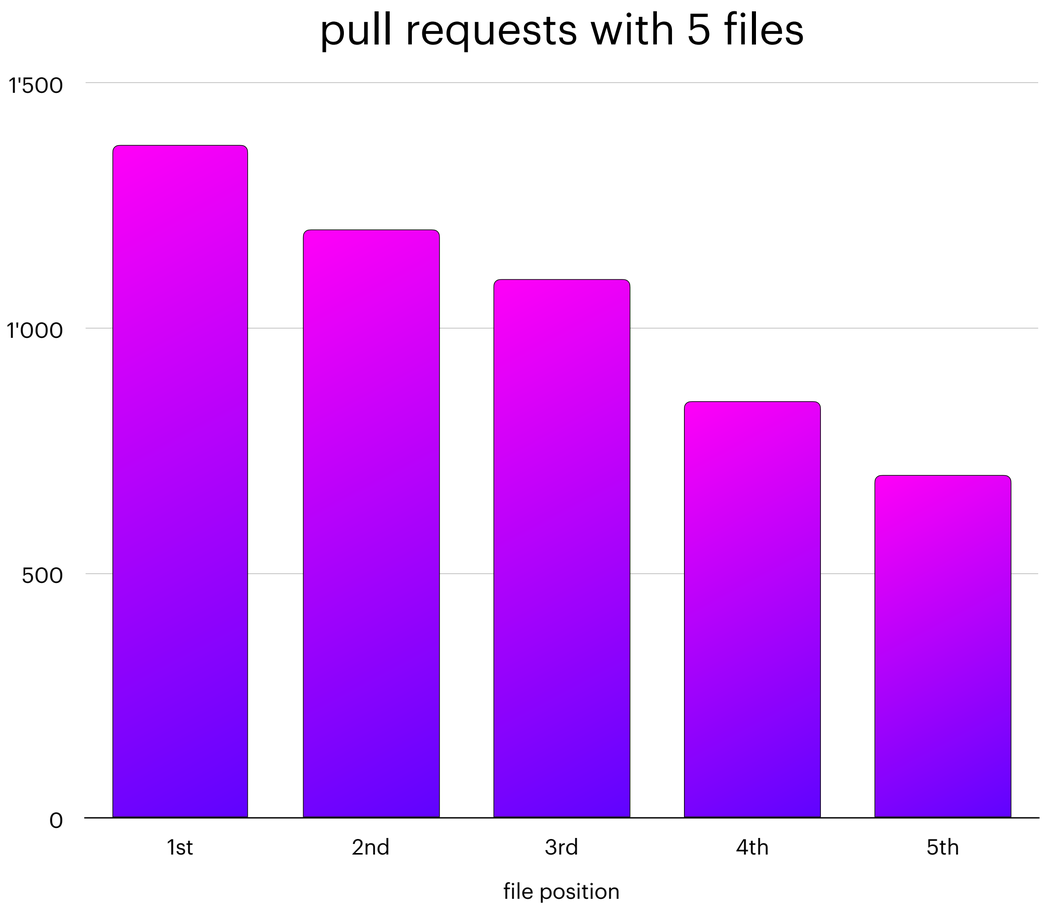
an observational study

Comments as **proxy** for reviewers' activity

We analyzed **~200K** pull requests from **138** popular GitHub projects (Java-based with > 1k stars)...

... and saw this.

cumulative number of review comments by file position

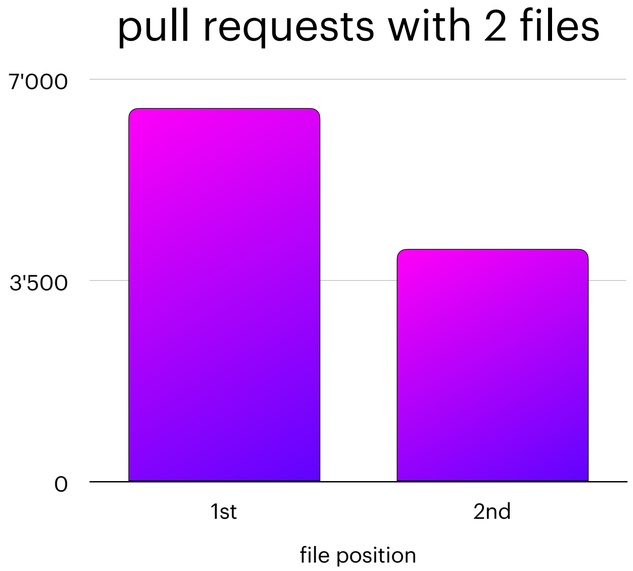


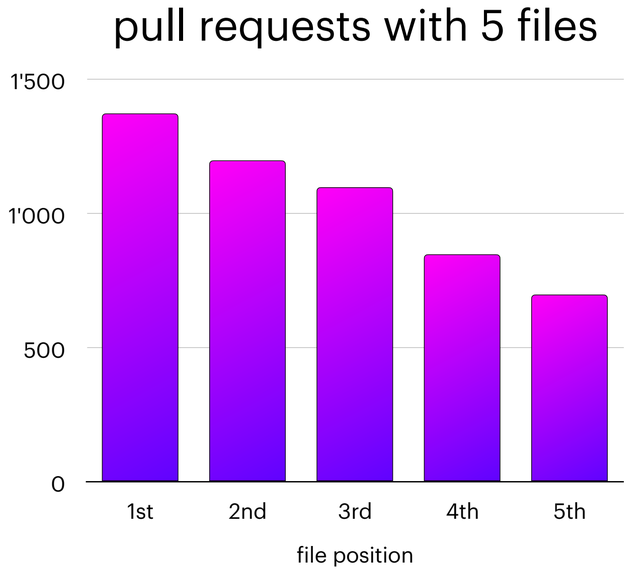
an observational study

Comments as **proxy** for reviewers' activity

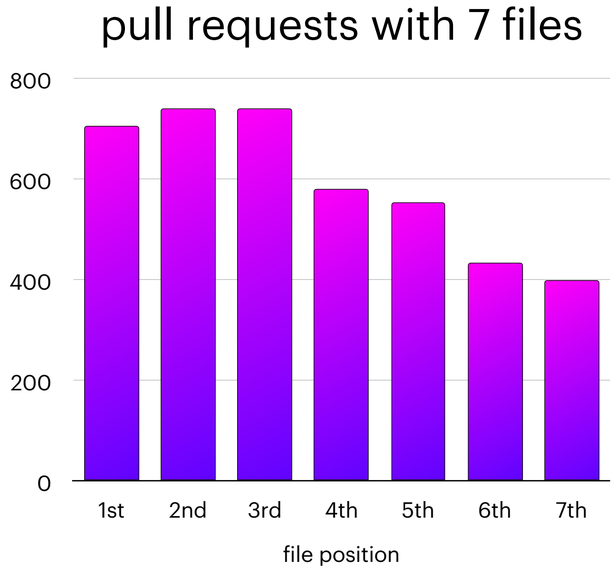
We analyzed ~200K PRs
from 138 popular GitHub projects
(Java-based with > 1k stars)...

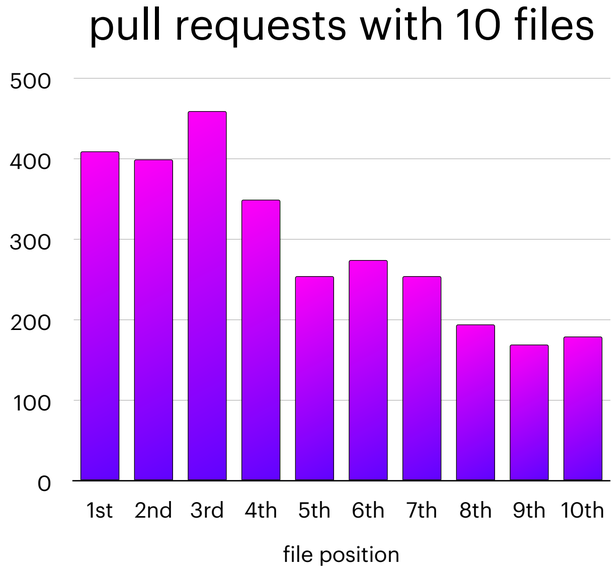
... and saw this.





cumulative number of review comments by file position



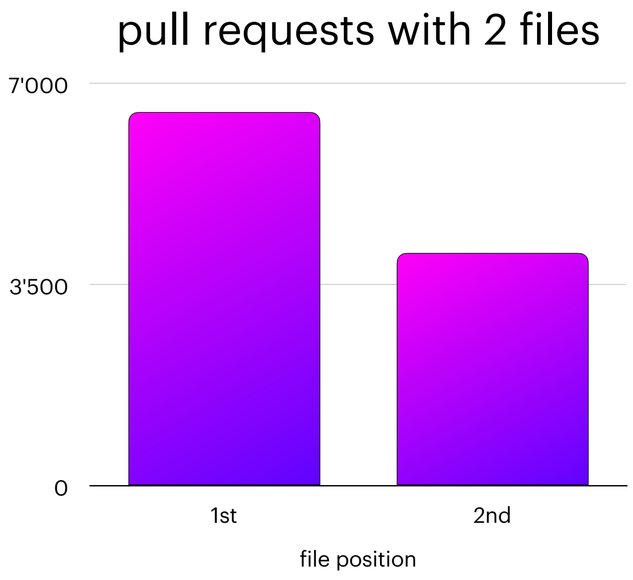


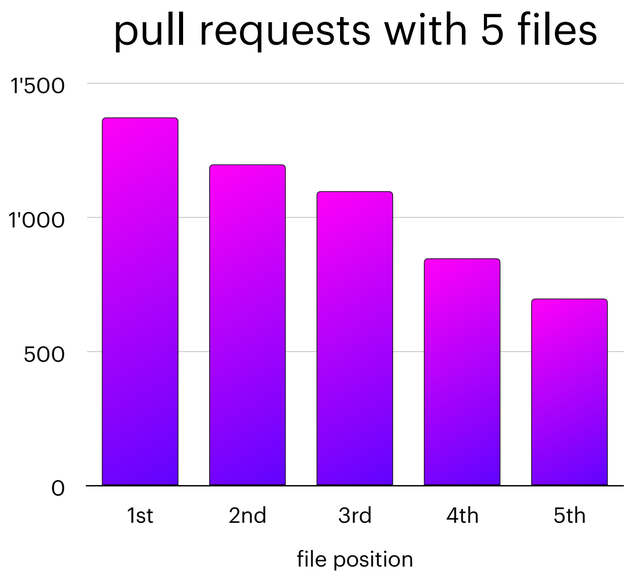
an observational study

Comments as **proxy** for reviewers' activity

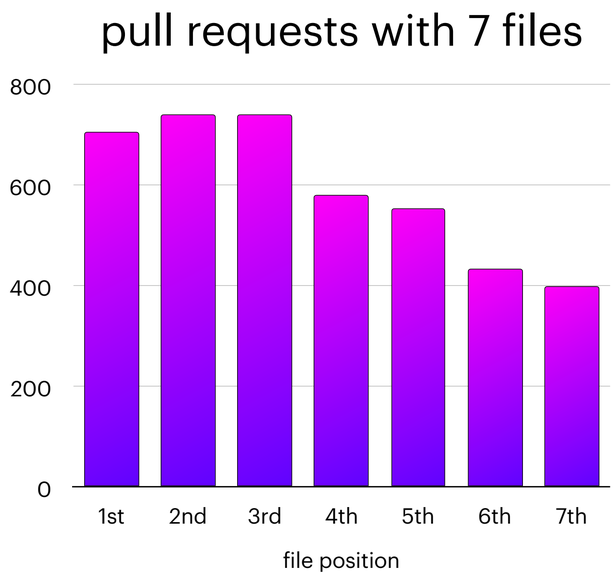
We analyzed ~200K PRs
from 138 popular GitHub projects
(Java-based with > 1k stars)...

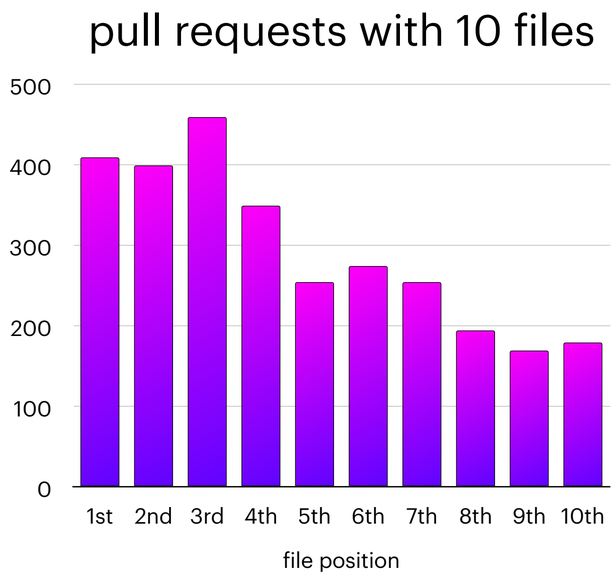
... and **saw** this.





cumulative number of review comments by file position





an observational study

Comments as **proxy** for reviewers' activity

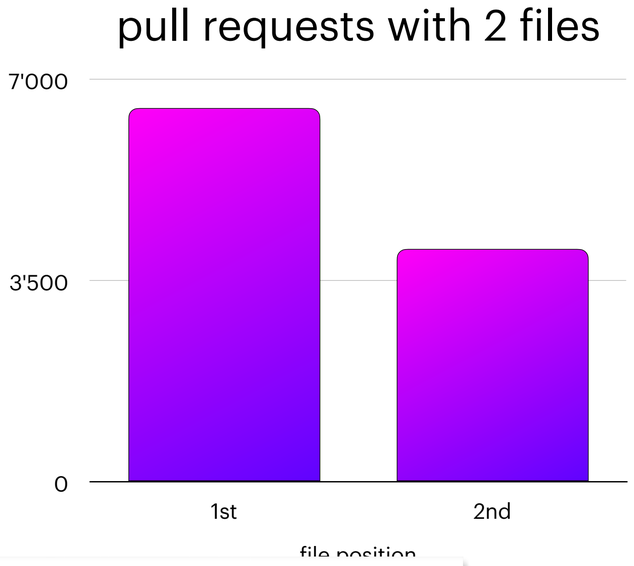
We analyzed ~200K PRs
from 138 popular GitHub projects
(Java-based with > 1k stars)...

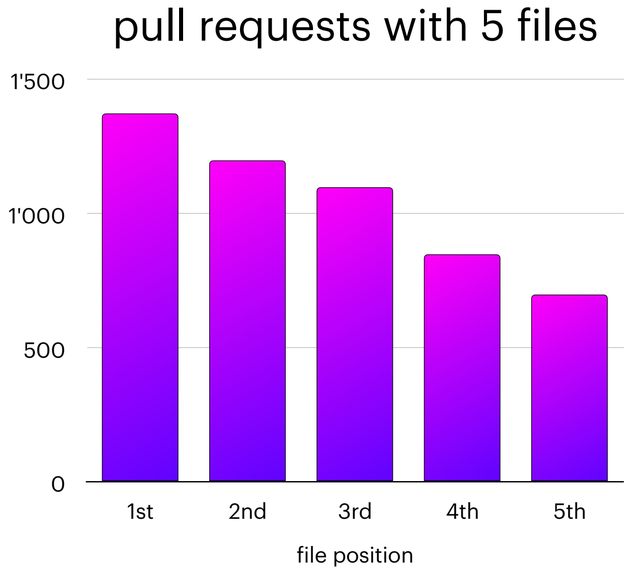
... and saw this.

1) where is the error?

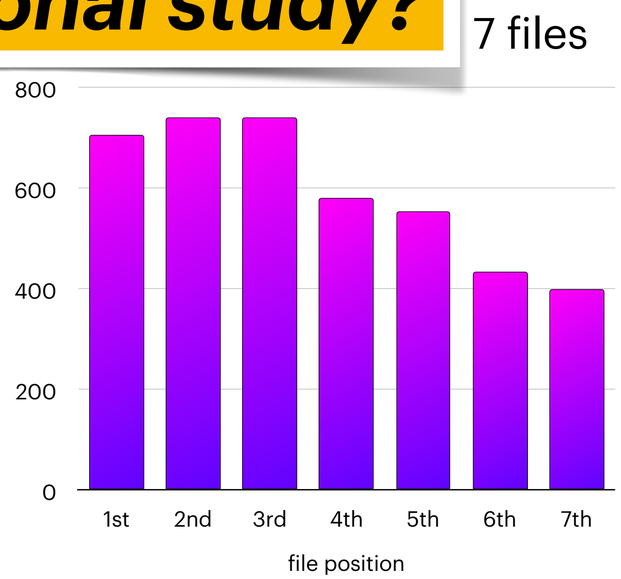
2) does it hold with statistics?

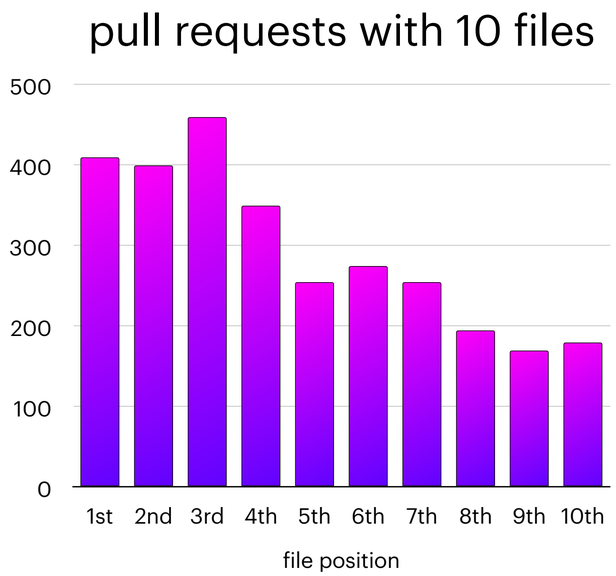
what are the limitations of an observational study?





nulative number of
eview comments
by file position





a controlled experiment

a.k.a. the gold standard for causal inference

main ingredients

- randomization
- control
- manipulation

What would be the <u>perfect</u> experiment?

a controlled experiment

a.k.a. the gold standard for causal inference

main ingredients

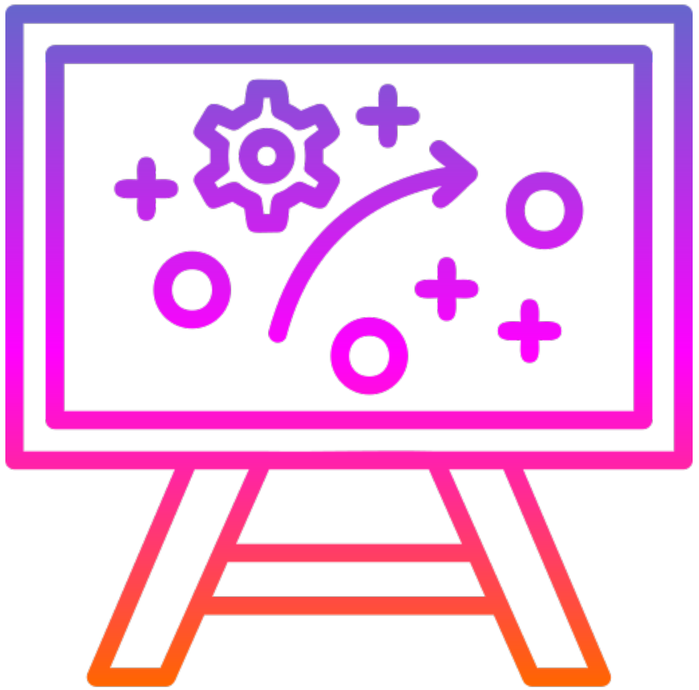
- randomization
- control
- manipulation

lessons learned

- the **perfect experiment** is often infeasible, but it's a good reference point
- it's ok to **trade-off** some realism to increase the visibility of the effect
- more **participants** is better than better participants

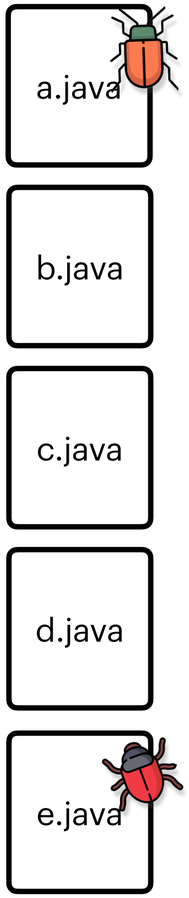
a controlled experiment

design



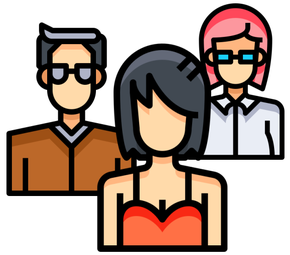
a controlled experiment

design

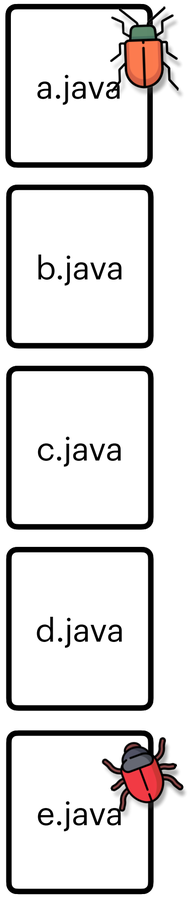


a controlled experiment

design



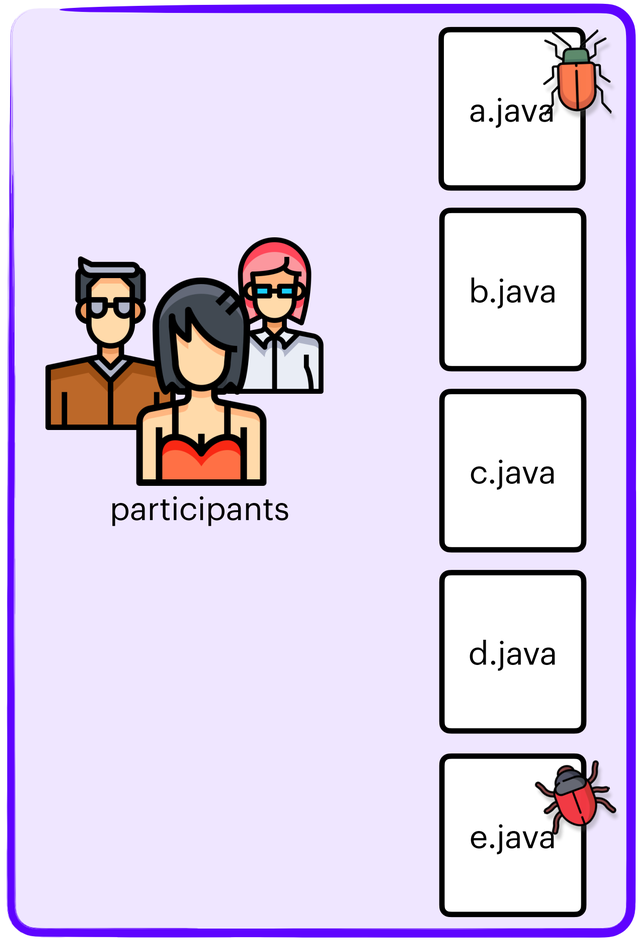
participants

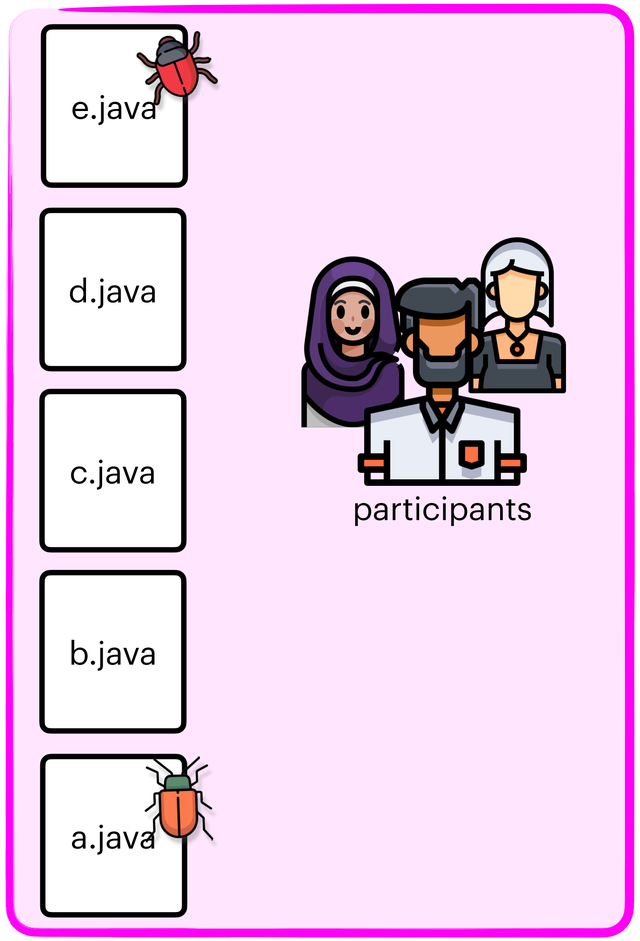


participants

a controlled experiment

design

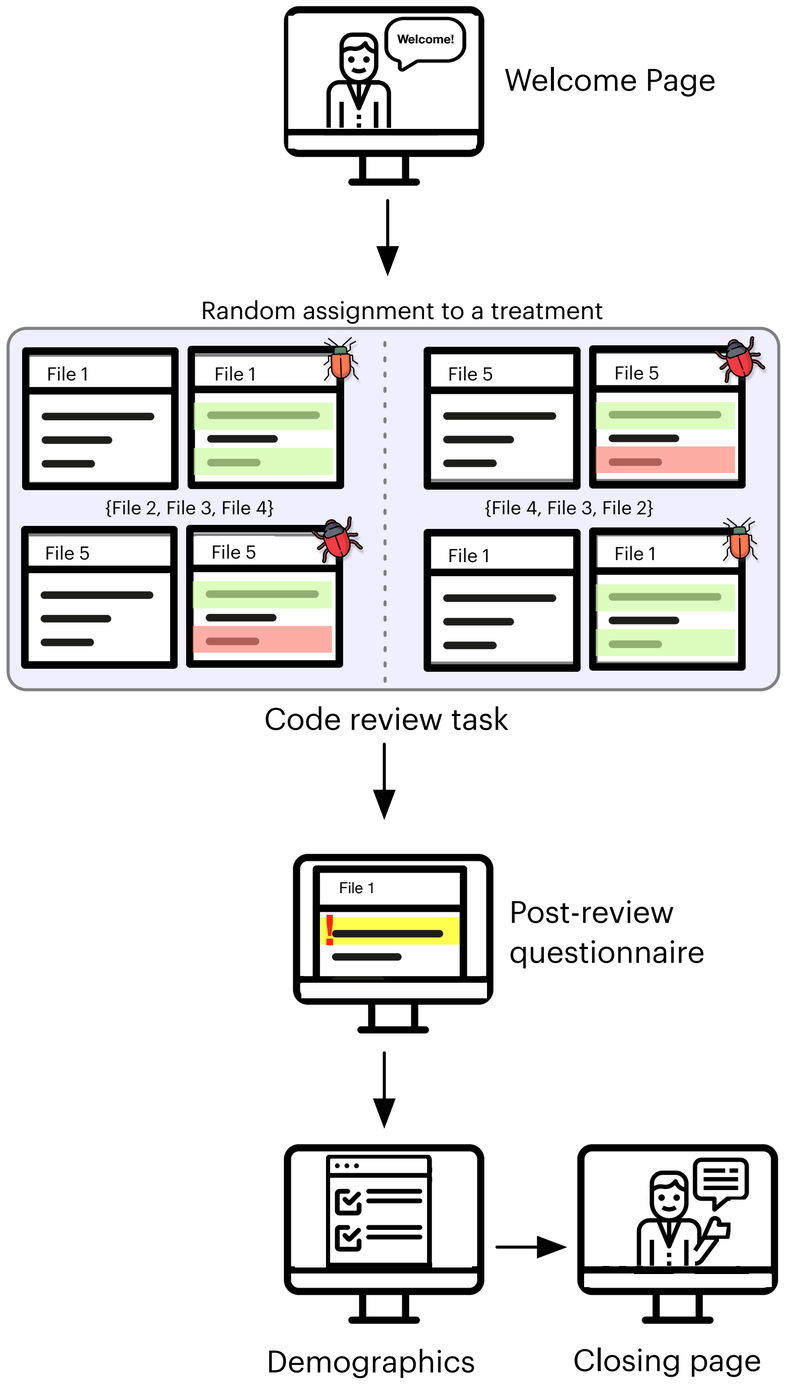




a controlled experiment
design

aspects to consider

- experiment platform
- objects
  - changes
  - bugs
- confounding factors
- participants
  - consent
  - recruiting



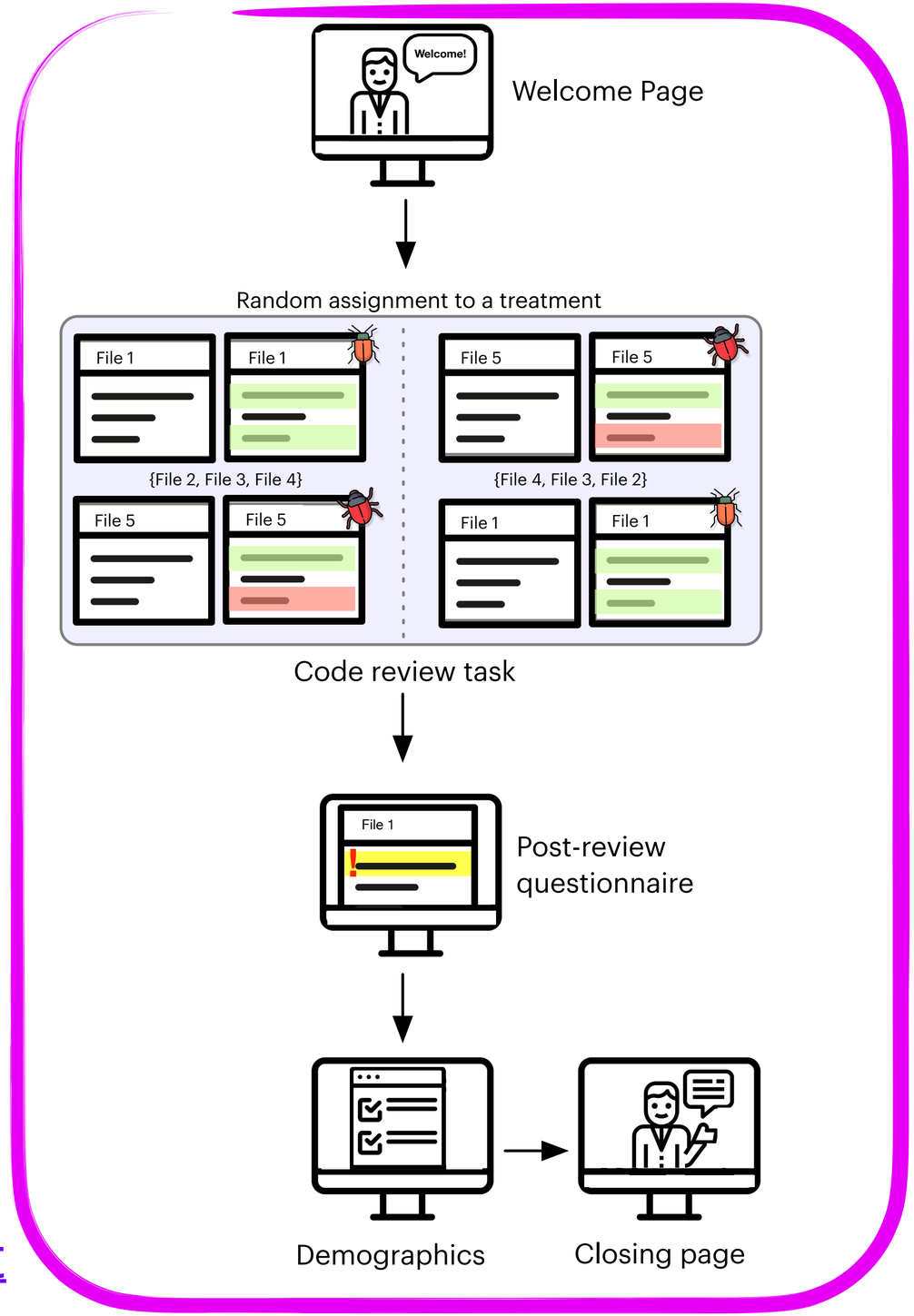
a controlled experiment
design

aspects to consider

- **experiment platform**
- objects
  - changes
  - bugs
- confounding factors
- participants
  - consent
  - recruiting

D. Spadini

https://github.com/ishepard/CRExperiment



a controlled experiment

design

aspects to consider

- experiment platform
- **objects**
  - changes
  - bugs
- confounding factors
- participants
  - consent
  - recruiting

a controlled experiment

design

aspects to consider

- experiment platform
- objects
  - **changes**
  - bugs
- confounding factors
- participants
  - consent
  - recruiting

a.java

b.java

c.java

d.java

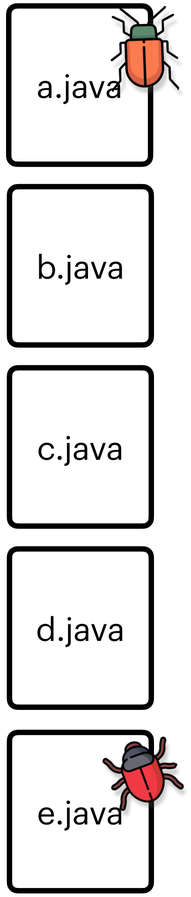
e.java

a controlled experiment

design

aspects to consider

- experiment platform
- objects
  - changes
  - **bugs**
- confounding factors
- participants
  - consent
  - recruiting

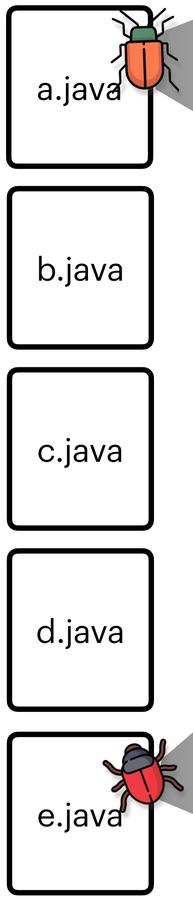


a controlled experiment

design

aspects to consider

- experiment platform
- objects
  - changes
  - **bugs**
- confounding factors
- participants
  - consent
  - recruiting




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

```
18              discountRate = 0.1;
19          }
20          if(membershipLevel >= 5) {
21              discountRate = 0.25;
22          }
23          return discountRate;
24      }
```

a controlled experiment

design

aspects to consider

- experiment platform
- **objects**
  - changes
  - bugs
- confounding factors
- participants
  - consent
  - recruiting

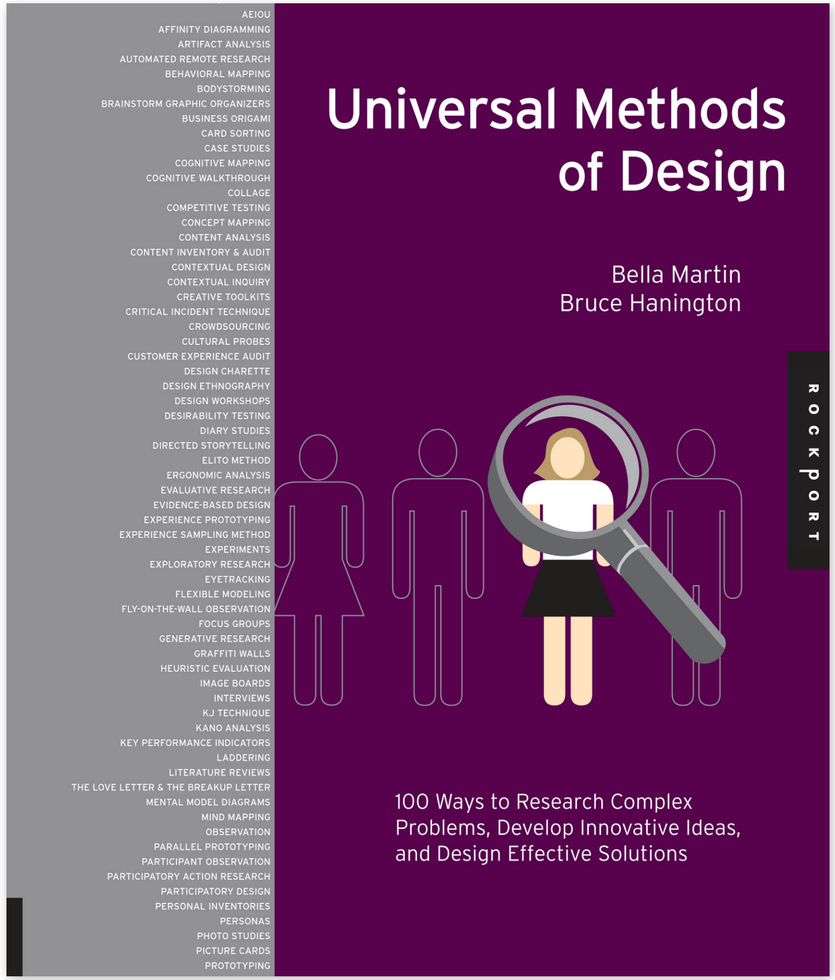
finding the right objects is an art...

you need to pilot your experiment!

<u>15 participants</u> (using RITE)

RITE

Rapid Iterative Testing & Evaluation



AN EXAMPLE TEST CYCLE USING THE RITE METHOD[3]

Totals: 11 participants
4 revised prototypes

a controlled experiment

design

aspects to consider

- experiment platform
- objects
  - changes
  - **bugs**
- confounding factors
- participants
  - consent
  - recruiting

a.java

c.java

d.java

e.java

how do we know if they found the bug?

```
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":
                    shippingCost = shippingCost * 1.27;
```

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

```
                default:
                    shippingCost = shippingCost * 2;
            }
            return shippingCost;
```

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

```
18           discountRate = 0.1;
19       }
20       if(membershipLevel >= 5) {
21           discountRate = 0.25;
22       }
23       return discountRate;
24   }
```

a controlled experiment

design

aspects to consider

- experiment platform
- objects
  - changes
  - bugs
- **confounding factors**
- participants
  - consent
  - recruiting

What could be <u>confounding factors</u>?

a controlled experiment

design

aspects to consider

- experiment platform
- objects
  - changes
  - bugs
- **confounding factors**
- participants
  - consent
  - recruiting

confounding factors we considered

- time
- interruptions
- practice
- experience
- education level

a controlled experiment

design

aspects to consider

- experiment platform
- objects
  - changes
  - bugs
- confounding factors
- **participants**
  - consent
  - recruiting

a controlled experiment

design

aspects to consider

- experiment platform
- objects
  - changes
  - bugs
- confounding factors
- participants
  - **consent**
  - recruiting

a controlled experiment

design

aspects to consider

- experiment platform
- objects
  - changes
  - bugs
- confounding factors
- participants
  - consent
  - **recruiting**

How many participants do we need?

a controlled experiment

design

aspects to consider

- experiment platform
- objects
  - changes
  - bugs
- confounding factors
- participants
  - consent
  - **recruiting**

How many participants do we need?

compute it with **Power Analysis**

- you need some ideas of what effect to expect
- find more info in this amazing book: https://lakens.github.io/statistical_inferences/
- go beyond the value you found

a controlled experiment

design

aspects to consider

- experiment platform
- objects
  - changes
  - bugs
- confounding factors
- participants
  - consent
  - **recruiting**

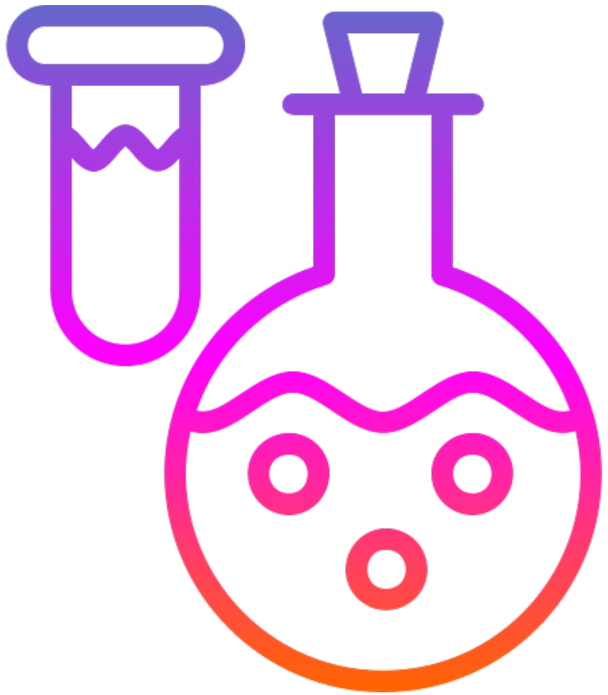
how to recruit participants

- personal network
- professional network
- social networks (X, LinkedIn, ...)
- reddit
- ...

be nice & offer donations if you can

a controlled experiment

data analysis



a controlled experiment

data analysis

what to do

- filter out non-serious participants
- use the right statistics
  - read Dr. Laken's book!
- conduct robustness testing

a controlled experiment

data analysis

what to do

- **filter** out non-serious participants
- use the right statistics
  - read Dr. Laken's book!
- conduct robustness testing

a controlled experiment

data analysis

what to do

- filter out non-serious participants
- use the right **statistics**
  - read Dr. Laken's book!
- conduct robustness testing

a controlled experiment

data analysis

what to do

- filter out non-serious participants
- use the right statistics
  - read Dr. Laken's book!
- conduct **robustness testing**

What could be potential biases?

a controlled experiment

data analysis

what to do

- filter out non-serious participants
- use the right statistics
  - read Dr. Laken's book!
- conduct **robustness testing**

potential problems we ruled out:

- participants' groups are not homogeneous
- one defect might influence participants in finding the other
- the defects are too easy/difficult
- a low number of participants

a controlled experiment

results

a controlled experiment

results

```
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;
```

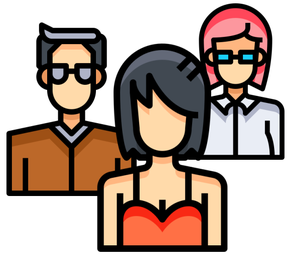
MB: Missing Break defect: Here a break statement is missing. In this way, when the country is UK, the execution will fall through the default case and a wrong tax of 1.27 * 2 will be applied.

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

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

```
18            discountRate = 0.1;
19        }
20        if(membershipLevel >= 5) {
21            discountRate = 0.25;
22        }
23        return discountRate;
24    }
```



participants

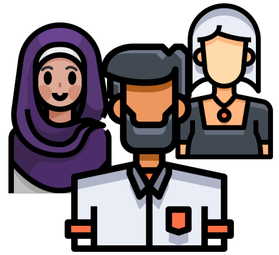
a.java

b.java

c.java

d.java

e.java



participants

a controlled experiment

results

```
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;
```

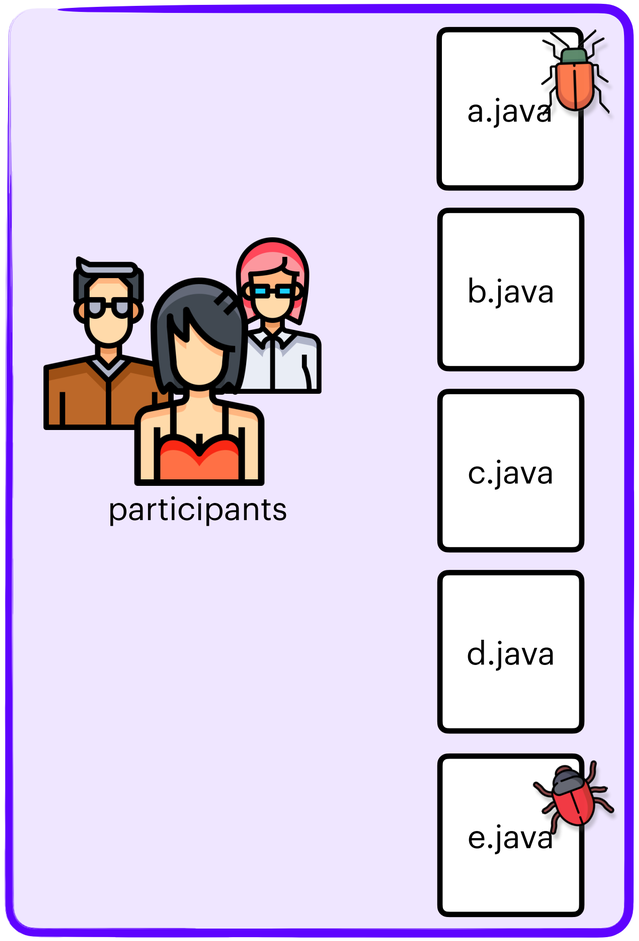
MB: Missing Break defect: Here a break statement is missing. In this way, when the country is UK, the execution will fall through the default case and a wrong tax of 1.27 * 2 will be applied.

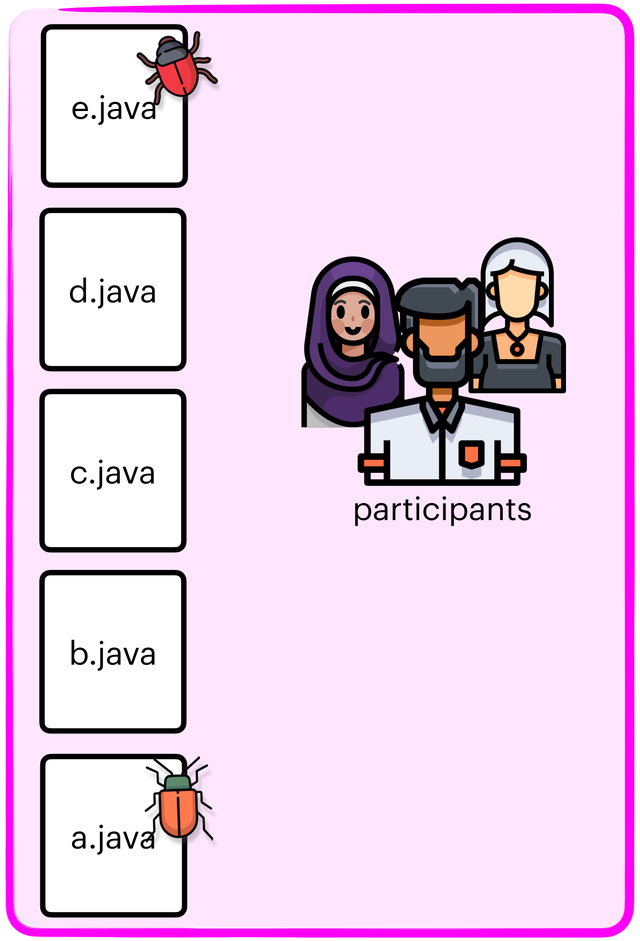
```
38            default:
39                shippingCost = shippingCost * 2;
40        }
41        return shippingCost;
```

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

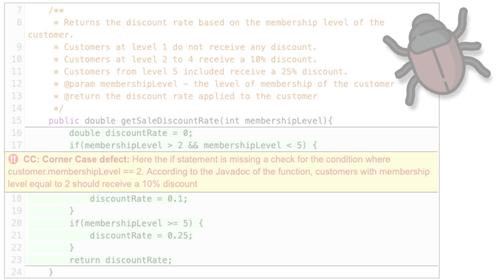
```
18            discountRate = 0.1;
19        }
20        if(membershipLevel >= 5) {
21            discountRate = 0.25;
22        }
23        return discountRate;
24    }
```





a controlled experiment

results

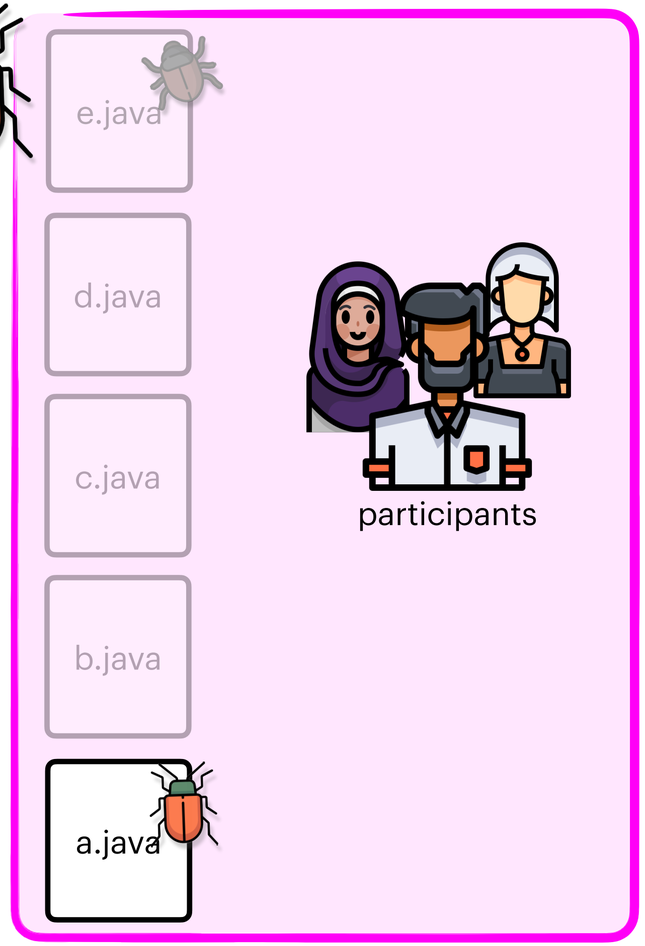



```
26        switch (destinationAddress.getCountry()) {
27            case "USA":
28                shippingCost = shippingCost * 1.2;
29                break;
30            case "Canada":
31                shippingCost = shippingCost * 1.18;
32                break;
33            case "Mexico":
34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

same likelihood of finding the bug (42%)



a controlled experiment

results

```
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;
```

175% more likely
to find the bug

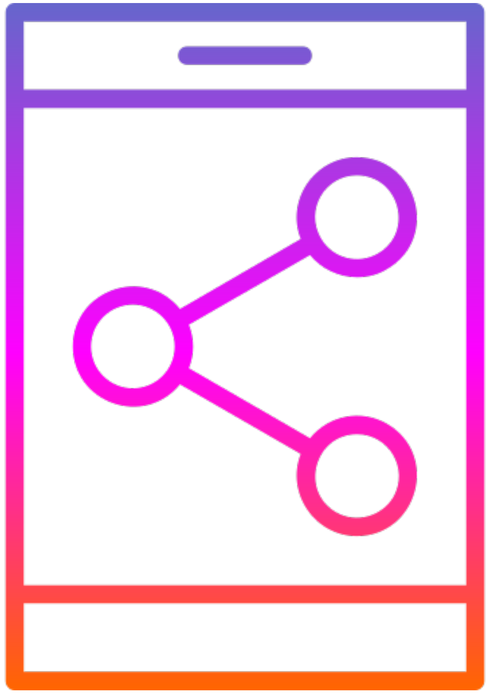

```
7       /**
8        * Returns the discount rate based on the membership level of the customer.
9        * Customers at level 1 do not receive any discount.
10       * Customers at level 2 to 4 receive a 10% discount.
11       * Customers from level 5 included receive a 25% discount.
12       * @param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

```
18              discountRate = 0.1;
19          }
20          if(membershipLevel >= 5) {
21              discountRate = 0.25;
22          }
23          return discountRate;
24      }
```

an empirical study

sharing data & materials



an empirical study

sharing data & materials

what to use

- arxiv (pre-print)
- zenodo (data & materials)
- github
  - yes, to maintain your tools!
  - but do not use for archiving

ABSTRACT

The most popular code review tools (*e.g.*, Gerrit and GitHub) present the files to review sorted in alphabetical order. Could this choice or, more generally, the relative position in which a file is presented bias the outcome of code reviews? We investigate this hypothesis by triangulating complementary evidence in a two-step study.

First, we observe developers' code review activity. We analyze the review comments pertaining to 219,476 Pull Requests (PRs) from 138 popular Java projects on GitHub. We found files shown earlier in a PR to receive more comments than files shown later, also when controlling for possible confounding factors: *e.g.*, the presence of discussion threads or the lines added in a file. Second, we measure the impact of file position on defect finding in code review. Recruiting 106 participants, we conduct an online controlled experiment in which we measure participants' performance in detecting two unrelated defects seeded into two different files. Participants are assigned to one of two treatments in which the position of the defective files is switched. For one type of defect, participants are not affected by its file's position; for the other, they have 64% lower odds to identify it when its file is last as opposed to first. Overall, our findings provide evidence that the relative position in which files are presented has an impact on code reviews' outcome; we discuss these results and implications for tool design and code review.

Preprint: https://doi.org/10.48550/arXiv.2208.04259
Data and Materials: https://doi.org/10.5281/zenodo.6901285

an empirical study

thank you to all co-authors

[ESEC/FSE 2023] First Come First Served: The Impact of File Position on Code Review 🏆
Enrico Fregnan, Larissa Braz, Marco D'Ambros, Gül Çalikli, Alberto Bacchelli

[ICSE 2022] Less is More: Supporting Developers in Vulnerability Detection during Code Review
Larissa Braz, Christian Aeberhard, Gül Çalikli, Alberto Bacchelli

[ICSE 2021] Why Don't Developers Detect Improper InputValidation?`'; DROP TABLE Papers; --` 🏆
Larissa Braz, Enrico Fregnan, Gül Çalikli, Alberto Bacchelli

[CHI2020] UI Dark Patterns and Where to Find Them: A Study on Mobile Applications and User Perception
Linda Di Geronimo, Larissa Braz, Enrico Fregnan, Fabio Palomba, Alberto Bacchelli

[ICSE 2020] Primers or Reminders? The Effects of Existing Review Comments on Code Review 🥇
Davide Spadini, Gül Çalikli, Alberto Bacchelli

[ICSE 2019] Test-Driven Code Review: An Empirical Study
Davide Spadini, Fabio Palomba, Tobias Baum, Stefan Hanenberg, Magiel Bruntink, Alberto Bacchelli


D. Spadini
zest


F. Palomba
zest


T. Baum
U. Hannover


S. Hanenberg
U. Duisburg-Essen


M. Bruntink
SIG


L. Di Geronimo
zest


G. Çalikli
zest

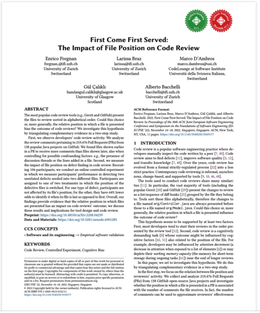

L. Braz
zest


E. Fregnan
zest


C. Aeberhard
zest

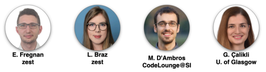

M. D'Ambros
CodeLounge@SI

Dissecting Empirical Research in Software Engineering

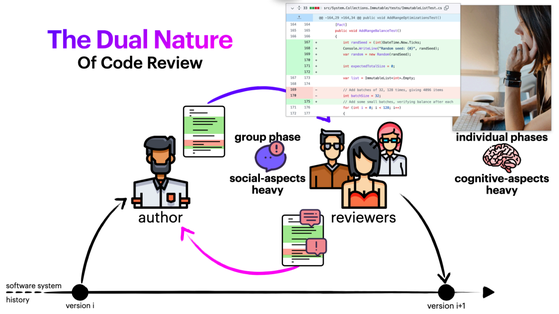

First Come First Served:
The Impact of File Position on Code Review

"The quality of this paper is such that I would add it to the list of papers that I give to students I work with to show them how research should be carried out and written up."
-- Reviewer 3

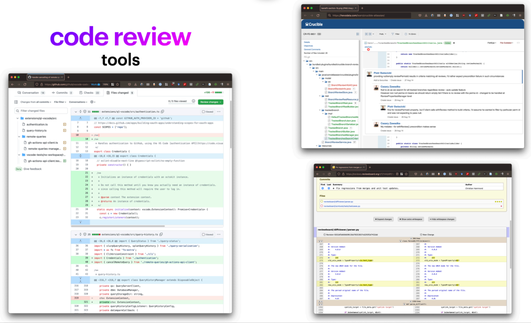
ACM SIGSOFT
Distinguished Paper Award
ESEC/FSE 2022


E. Fregnan zest · L. Braz zest · M. D'Ambros CodeLounge@SI · G. Çalikli U. of Glasgow

The Dual Nature
Of Code Review



code review
tools

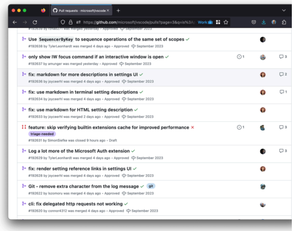


an observational study

Yet, what else could affect the number of comments on files (i.e., **confounding factors**)?

- change size
- test files
- number of participants
- bots
- threads
- ... ?
  - big data matters

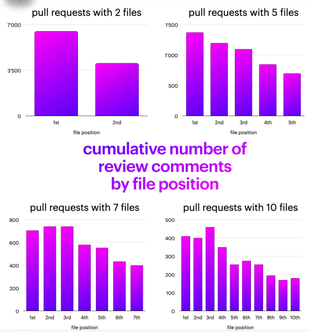

M. D'Ambros
CodeLounge@SI



an observational study

We analyzed ~200K PRs from 138 popular GitHub projects (Java-based with > 1k stars)...

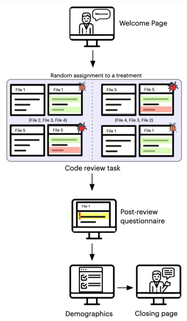
... and **saw** this.



a controlled experiment
design

aspects to consider

- experiment platform
- objects
  - changes
  - bugs
- confounding factors
- participants
  - consent
  - recruiting



a controlled experiment
data analysis

what to do

- filter out non-serious participants
- use the right statistics
  - read Dr. Laken's book!
- conduct **robustness testing**

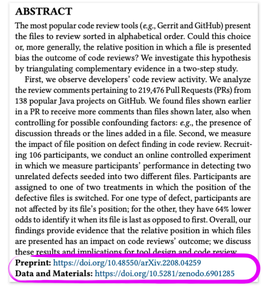
potential problems we ruled out:

- participants' groups are not homogeneous
- one defect might influence participants in finding the other
- the defects are too easy/difficult
- a low number of participants

an empirical study
sharing data & materials

what to use

- arxiv (pre-print)
- zenodo (data & materials)
- github
  - yes, to maintain your tools!
  - but do not use for archiving


ABSTRACT
The most popular code review tools (e.g., Gerrit and GitHub) present the files to review sorted in alphabetical order. Could this choice or, more generally, the relative position in which a file is presented bias the outcome of code reviews? We investigate this hypothesis by triangulating complementary evidence in a two-step study.
First, we observe developers' code review activity. We analyze the review comments pertaining to 219,476 Pull Requests (PRs) from 138 popular Java projects on GitHub. We found files shown earlier in a PR to receive more comments than files shown later, also when controlling for possible confounding factors: e.g., the presence of discussion threads or the lines added in a file. Second, we measure the impact of file position on defect finding in code review. Recruiting 106 participants, we conduct an online controlled experiment in which we measure participants' performance in detecting two unrelated defects seeded into two different files. Participants are assigned to one of two treatments in which the position of the defective files is switched. For one type of defect, participants are not affected by its file's position; for the other, they have 64% lower odds to identify it when its file is last as opposed to first. Overall, our findings provide evidence that the relative position in which files are presented has an impact on code reviews' outcome; we discuss these results and implications for tool design and code review.
Preprint: https://doi.org/10.48550/arXiv.2208.04259
Data and Materials: https://doi.org/10.5281/zenodo.6901285

Alberto Bacchelli
ASSOCIATE PROFESSOR
HEAD OF ZEST

zest
University of Zurich[UZH]

Binzmühlestrasse 14, 8050 Zurich, Switzerland
ADDRESS

zest@ifi.uzh.ch
EMAIL

@ZESTuzh
TWITTER

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