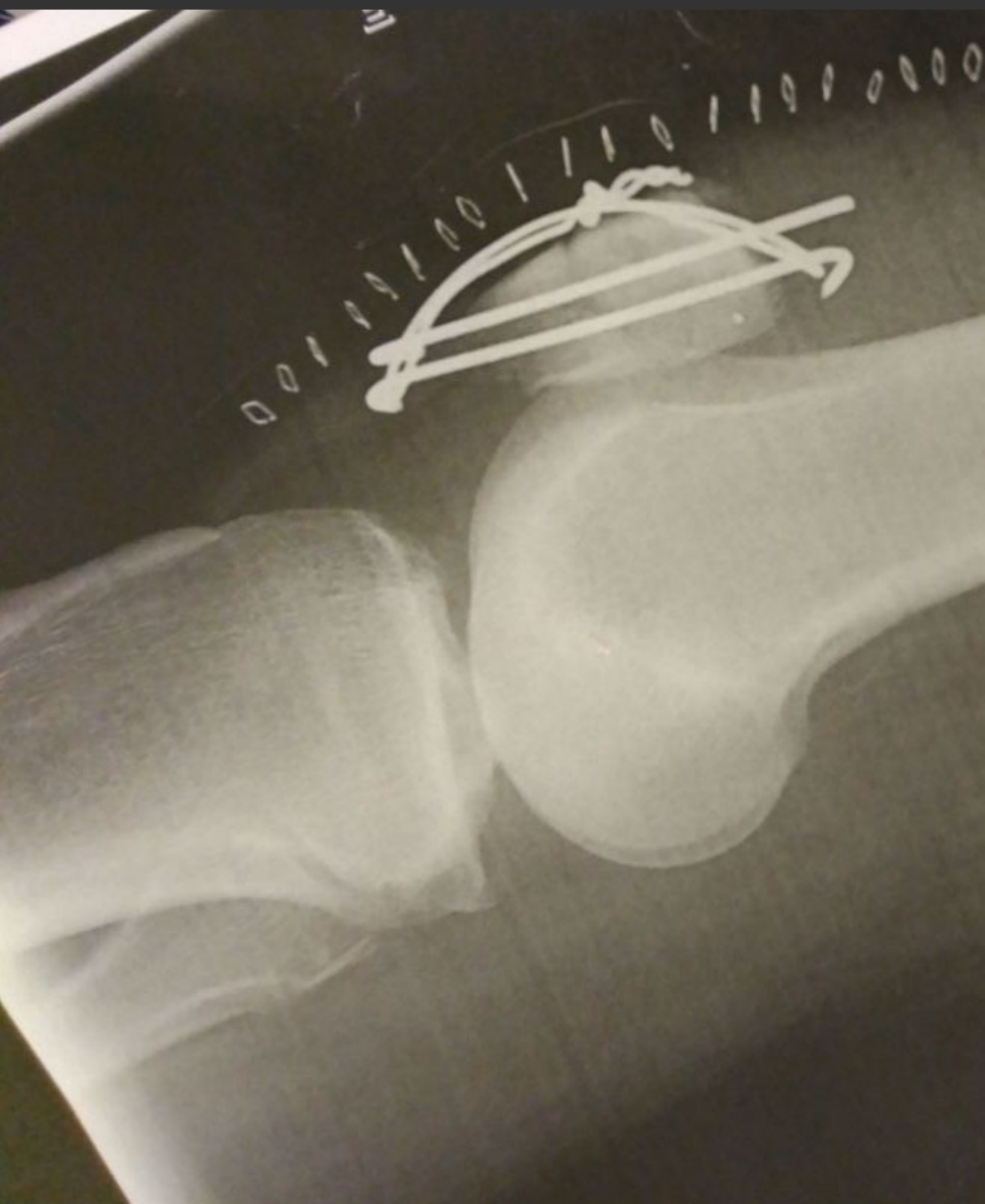
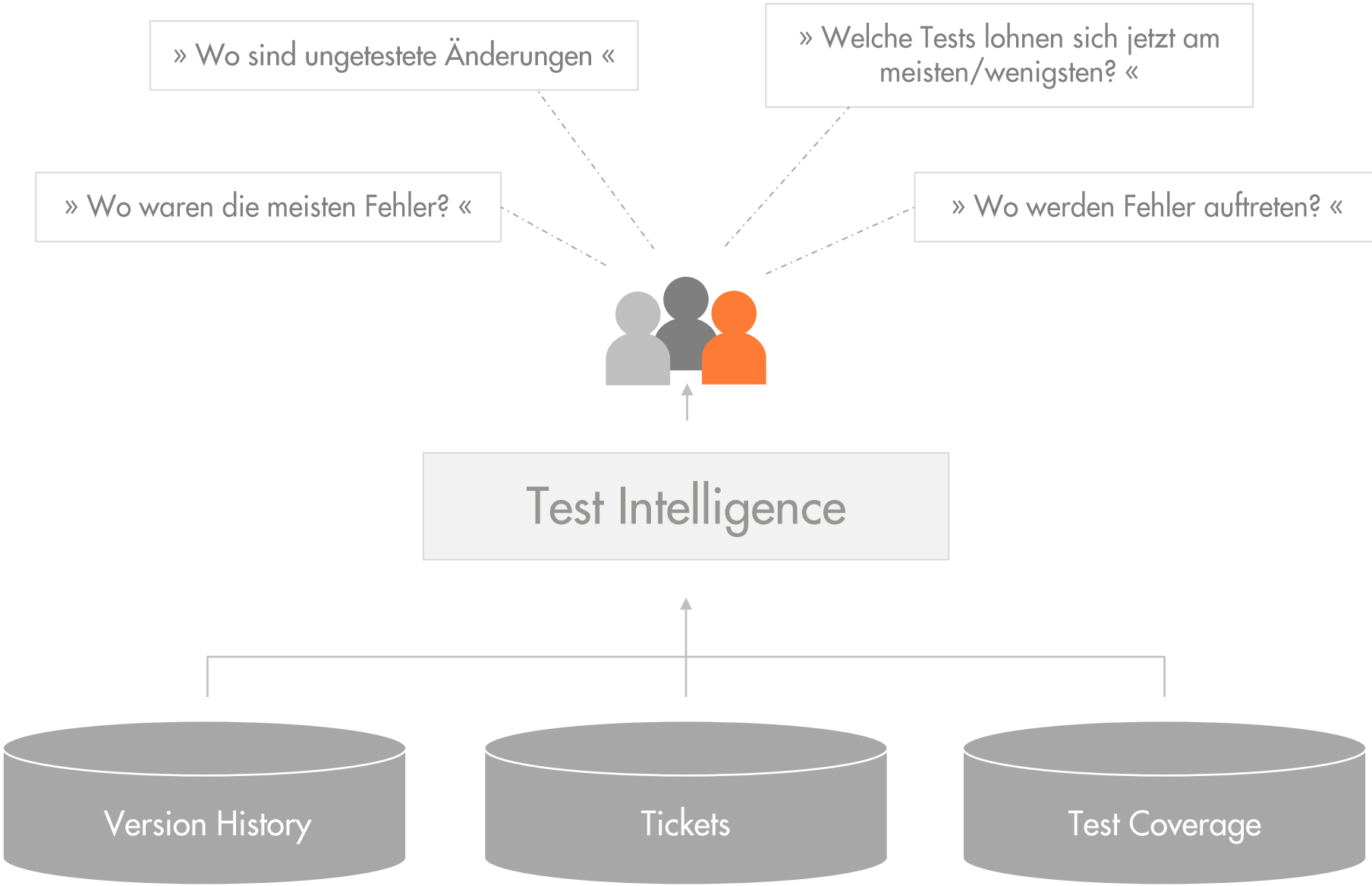
Test Intelligence

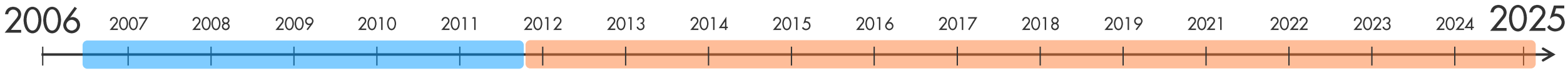
Wie finden wir schneller mehr Fehler in unserem Embedded System?

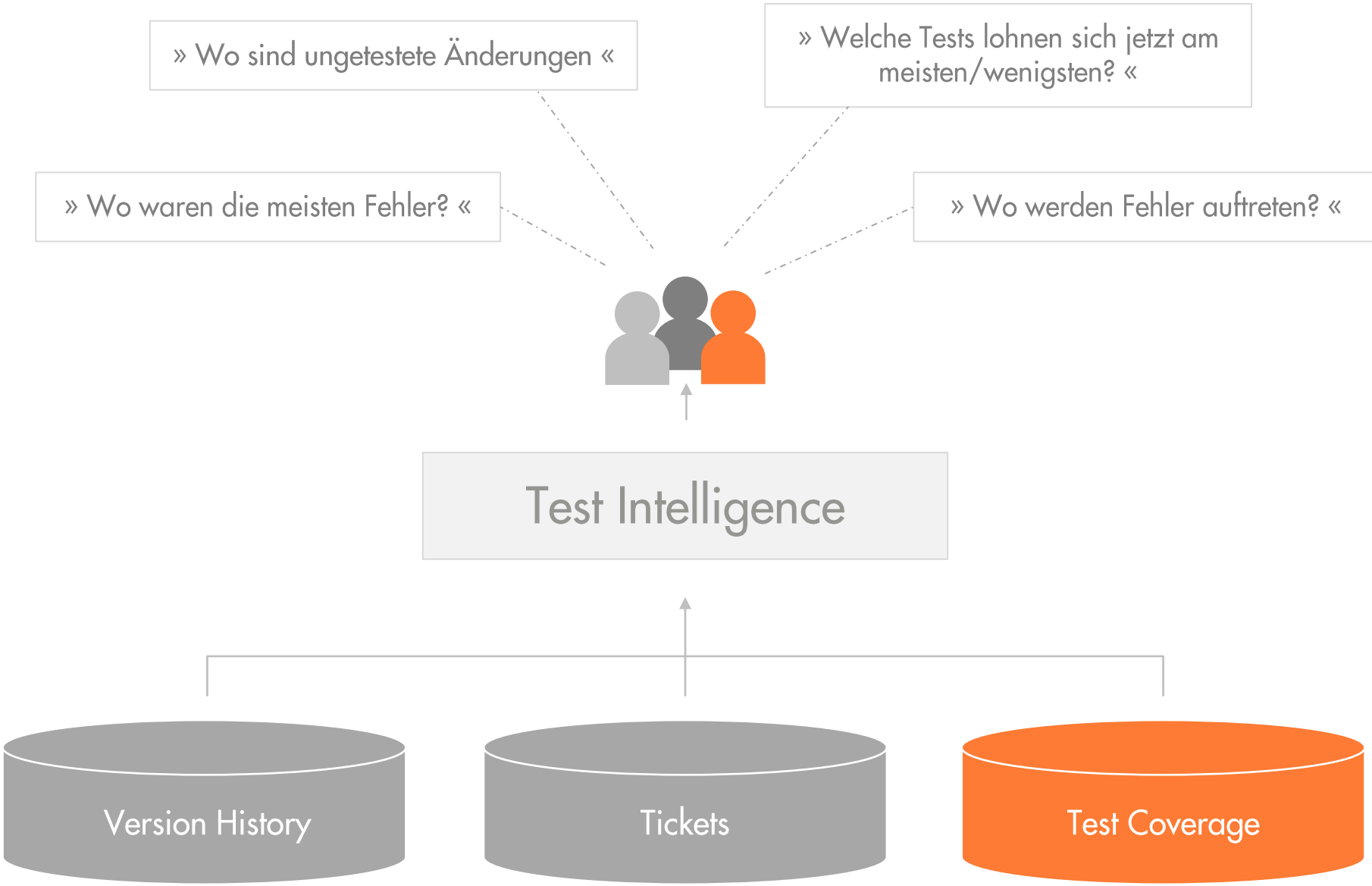
Dr. Elmar Jürgens

CQSE




» Wo sind ungetestete Änderungen «
» Welche Tests lohnen sich jetzt am meisten/wenigsten? «
» Wo waren die meisten Fehler? «
» Wo werden Fehler auftreten? «
Test Intelligence
Version History
Tickets
Test Coverage


2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2021
2022
2023
2024
2025


» Wo sind ungetestete Änderungen «
» Welche Tests lohnen sich jetzt am meisten/wenigsten? «
» Wo waren die meisten Fehler? «
» Wo werden Fehler auftreten? «
Test Intelligence
Version History
Tickets
Test Coverage

Picture1.png - Pixelitor 4.2.3

File Edit Layer Select Image Color Filter View Help

Sample Only the Active Layer/Mask

Untitled1 × Picture1.png ×

Layers

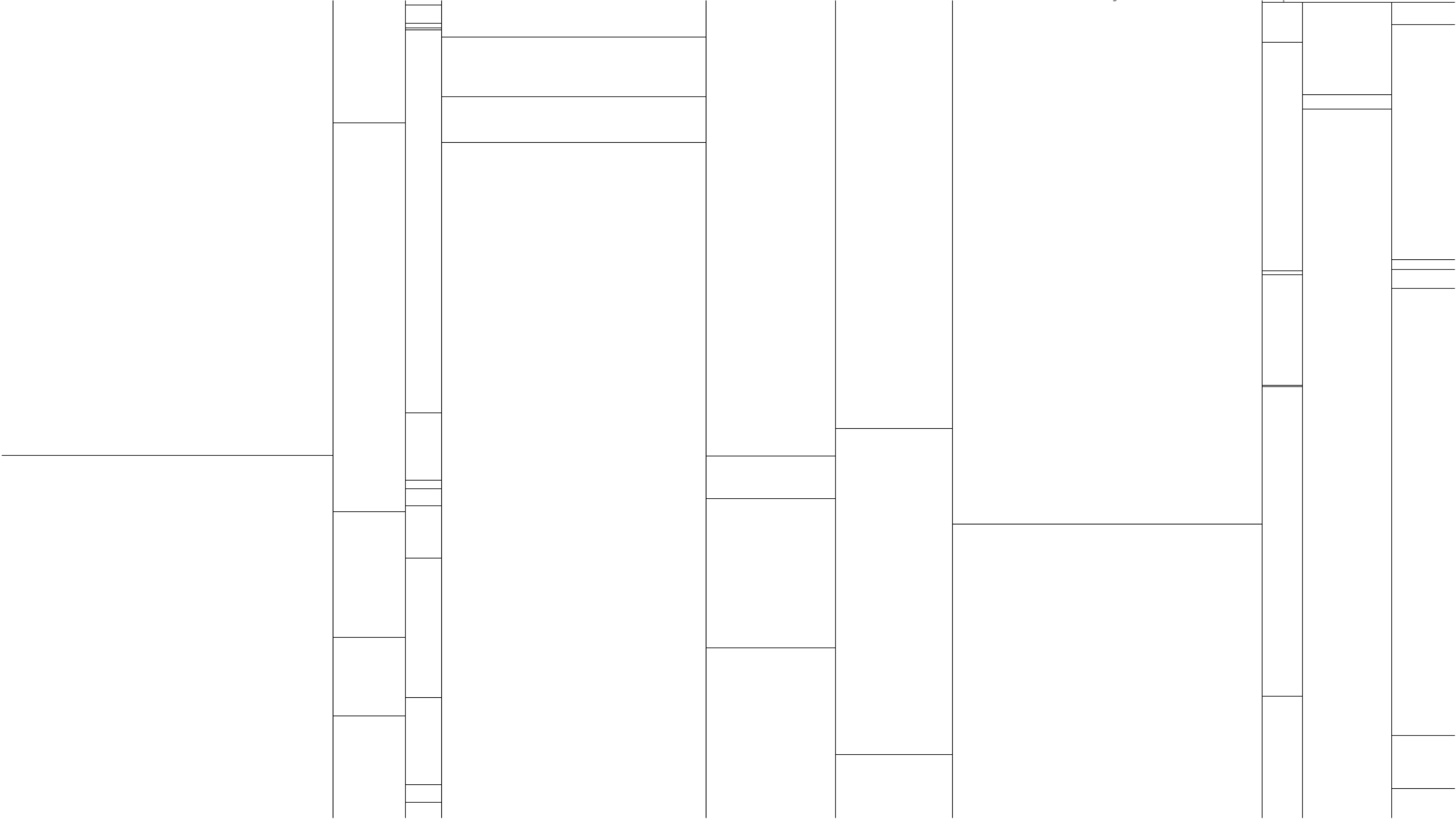
Opacity: 100 % Normal

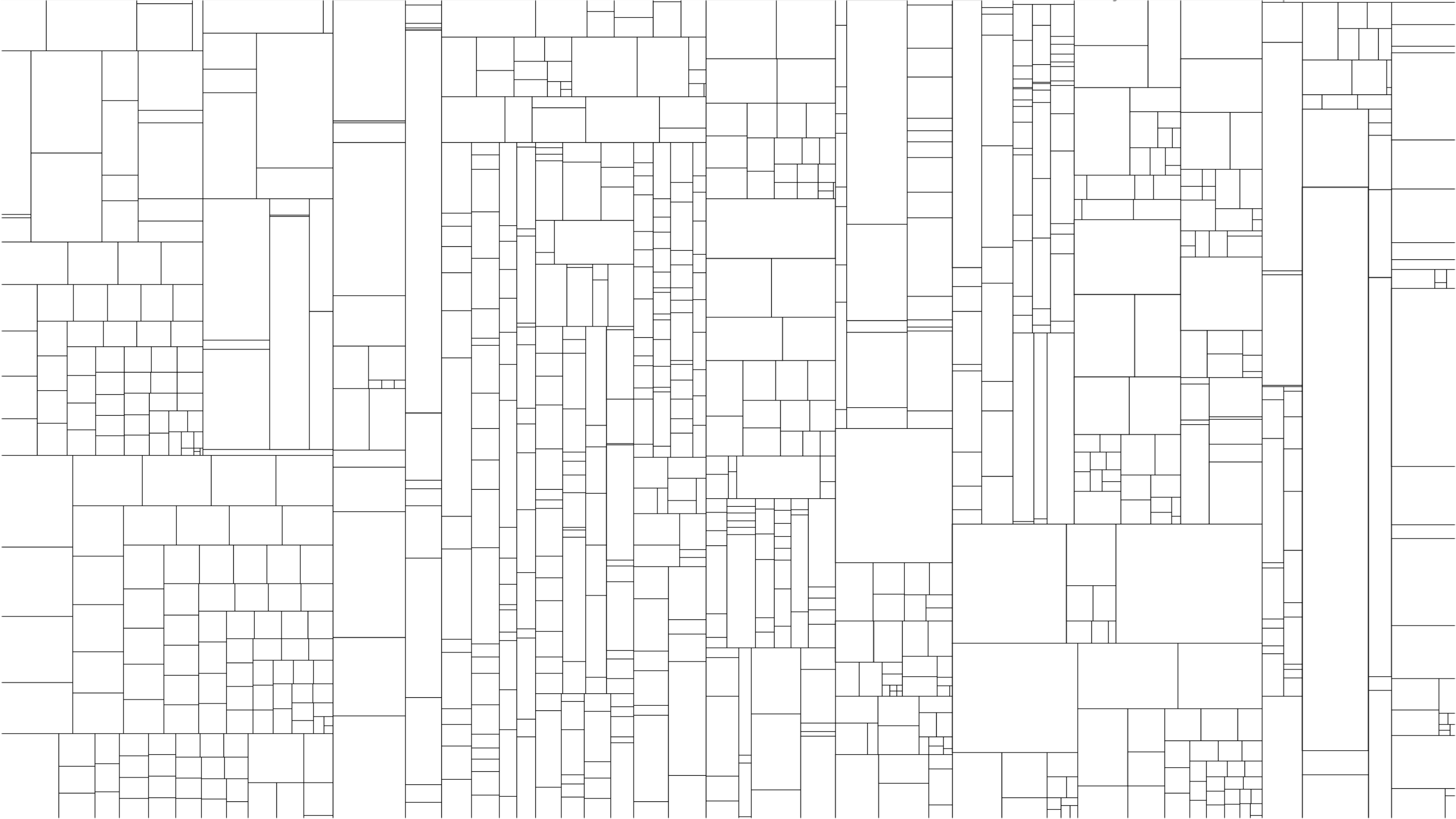
layer 1

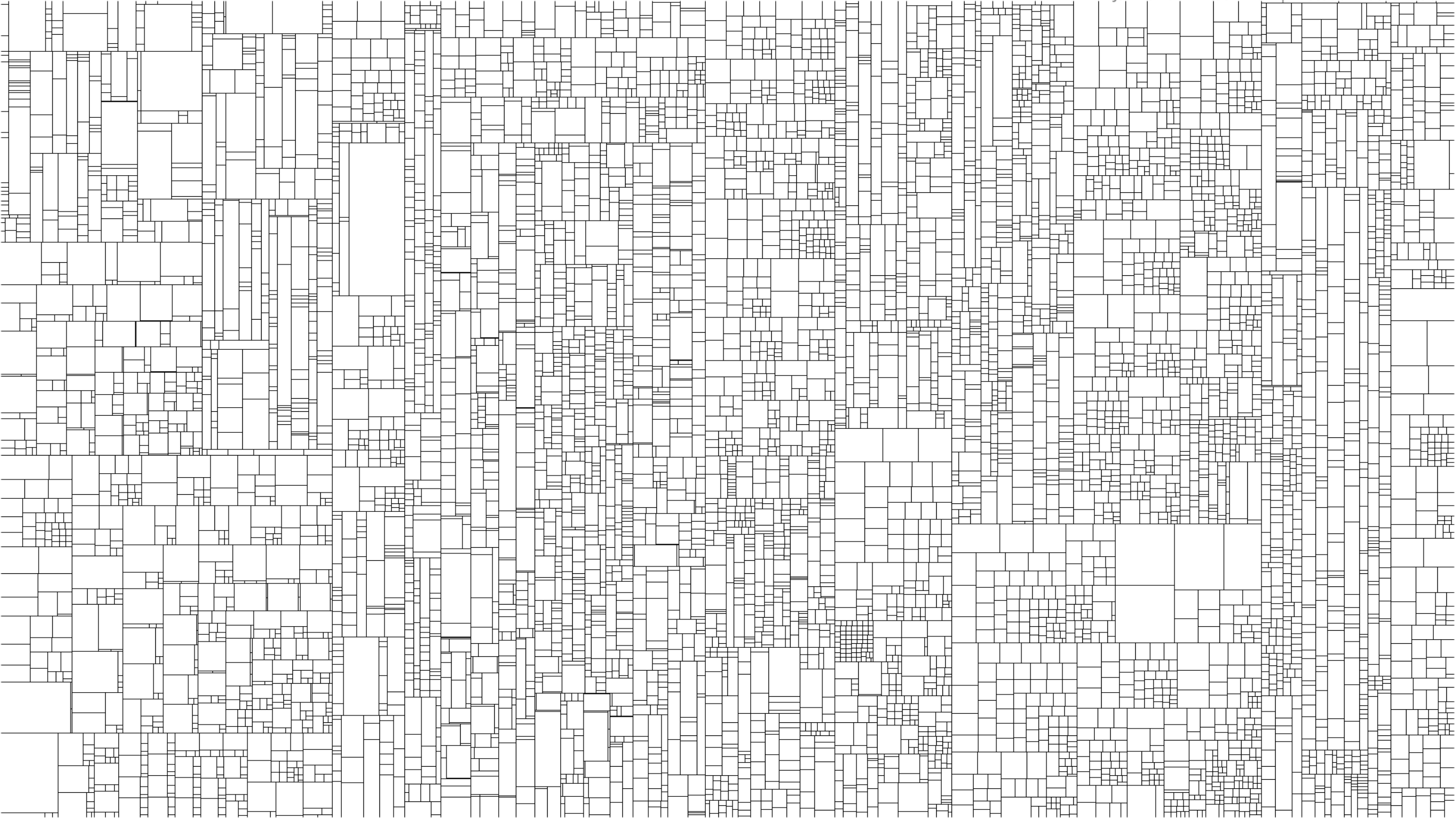
Picture1.png was opened.

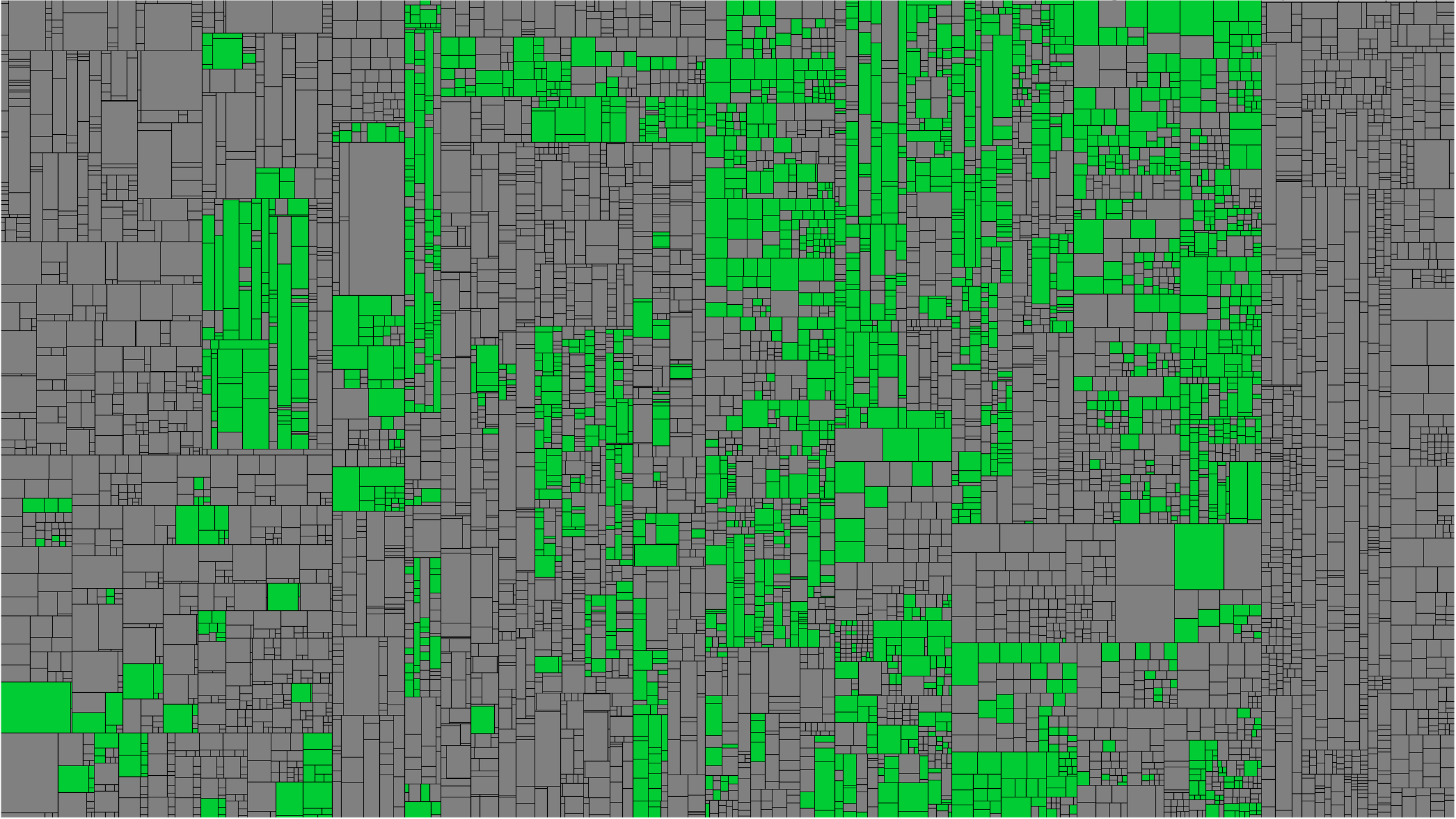
Zoom: 100% Fit 100%

```
114
115     private static void createAndShowGUI(String[] args) {
116         assert calledOnEDT() : threadInfo();
117
118         Messages.setMsgHandler(new GUIMessageHandler());
119
120 //        GlobalKeyboardWatch.showEventsSlowerThan(100, TimeUnit.MILLISECONDS);
121
122         Theme theme = Themes.DEFAULT;
123         // if a LaF was set from the command line, then don't override it
124         if (System.getProperty("swing.defaultlaf") == null) {
125             theme = AppPreferences.loadTheme();
126             Themes.install(theme, false, true);
127         }
128
129         int uiFontSize = AppPreferences.loadUIFontSize();
130         String uiFontType = AppPreferences.loadUIFontType();
131
132         Font defaultFont = UIManager.getFont("defaultFont");
133         if (defaultFont != null) { // if null, we don't know how to set the font
134             if (uiFontSize != 0 || !uiFontType.isEmpty()) {
135                 Font newFont;
136                 if (!uiFontType.isEmpty()) {
137                     newFont = new Font(uiFontType, Font.PLAIN, uiFontSize);
138                 } else {
139                     newFont = defaultFont.deriveFont((float) uiFontSize);
140                 }
141
142                 FontUIResource fontUIResource = new FontUIResource(newFont);
143                 UIManager.put("defaultFont", fontUIResource);
144
145                 if (theme.isNimbus()) {
146                     UIManager.getLookAndFeel().getDefaults().put("defaultFont", fontUIResource);
147                 }
148             }
149         }
150
151         var pw = PixelitorWindow.get();
152         Dialogs.setMainWindowInitialized(true);
153
154         // Just to make 100% sure that at the end of GUI
155         // initialization the focus is not grabbed by
156         // a textfield and the keyboard shortcuts work properly
157         FgBgColors.getGUI().requestFocus();
158
159         TipsOfTheDay.showTips(pw, false);
160
161         MouseZoomMethod.load();
162         PanMethod.load();
163
164         // The IO-intensive preloading of fonts is scheduled
165         // to run after all the files have been opened,
166         // and on the same IO thread
167         openCLFilesAsync(args)
168             .exceptionally(throwable -> null) // recover
169             .thenAcceptAsync(v -> afterStartTestActions(), onEDT)
170             .thenRunAsync(Utils::preloadFontNames, onIOThread)
171             .exceptionally(Messages::showExceptionOnEDT);
172     }
173
```









BULLSEYE TESTING TECHNOLOGY

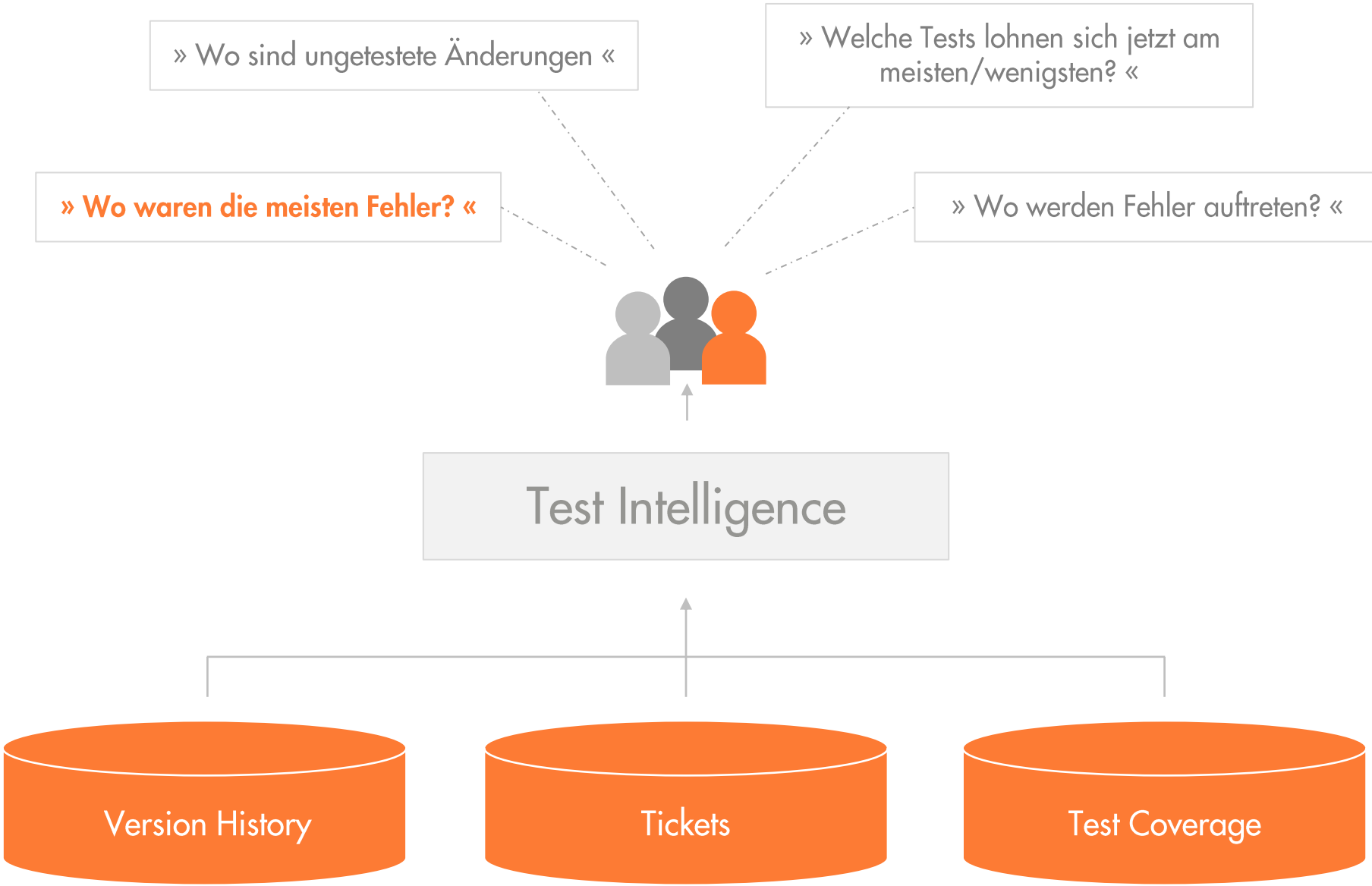
Testwell CTC++

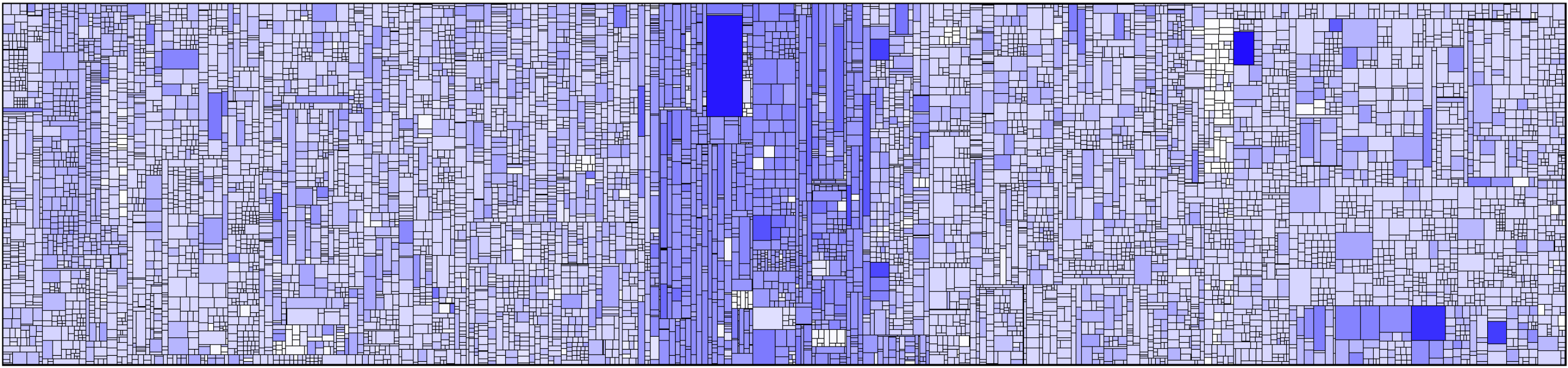
EmbeddedProfiler

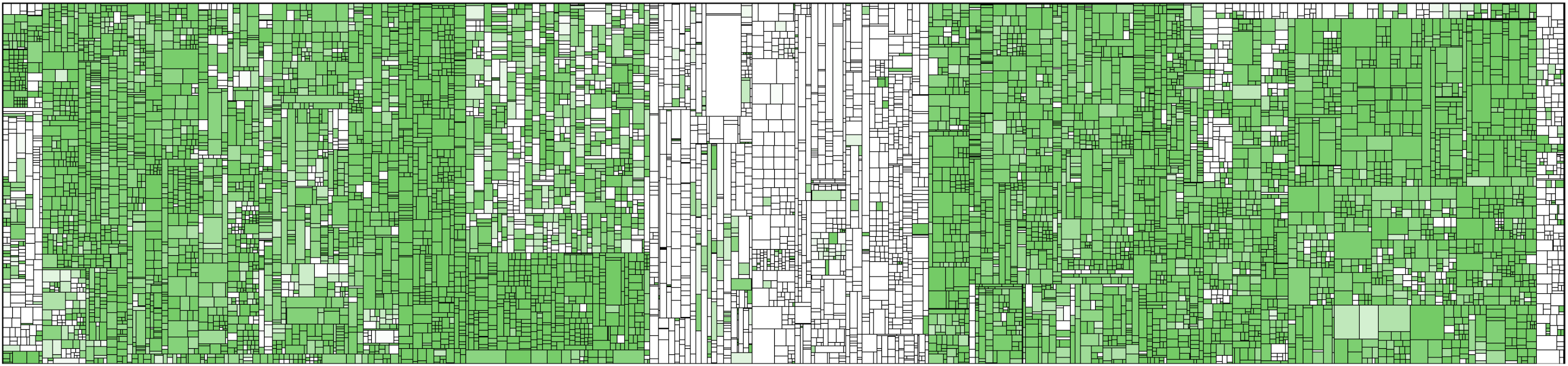
GCC

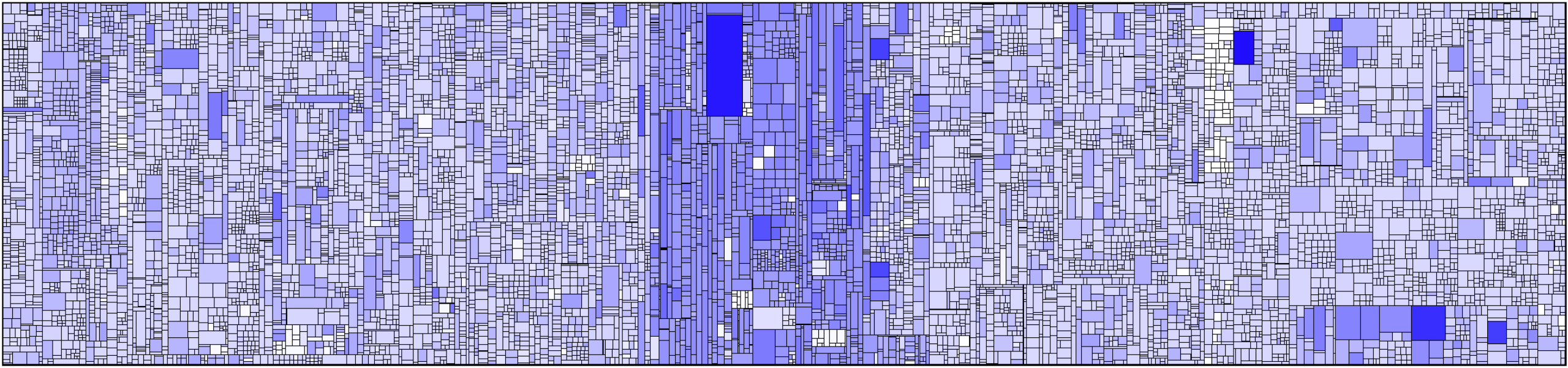
Software-basierte Instrumentierung

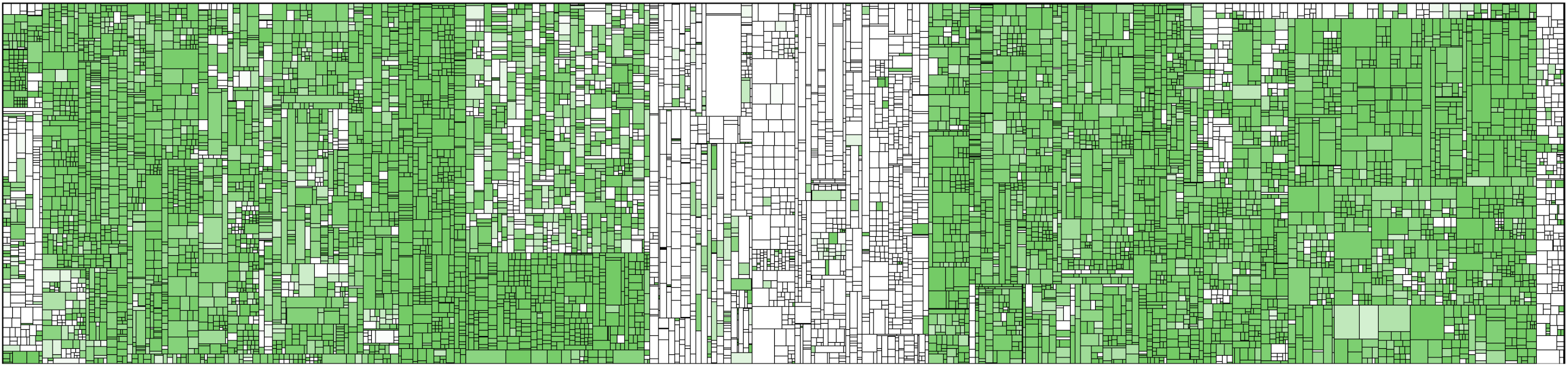
Hardware Debugger

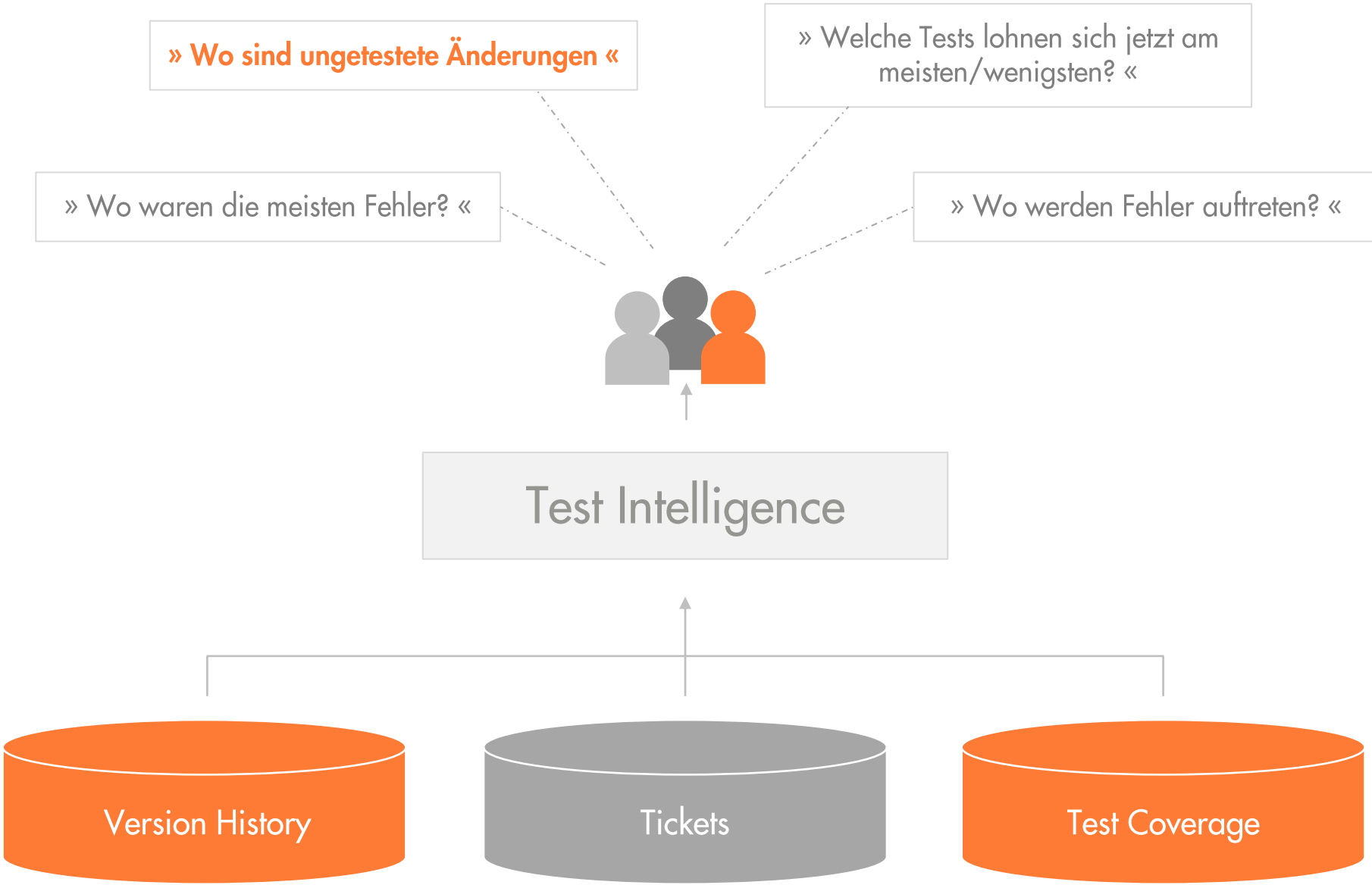

» Wo sind ungetestete Änderungen «
» Welche Tests lohnen sich jetzt am meisten/wenigsten? «
» Wo waren die meisten Fehler? «
» Wo werden Fehler auftreten? «
Test Intelligence
Version History
Tickets
Test Coverage

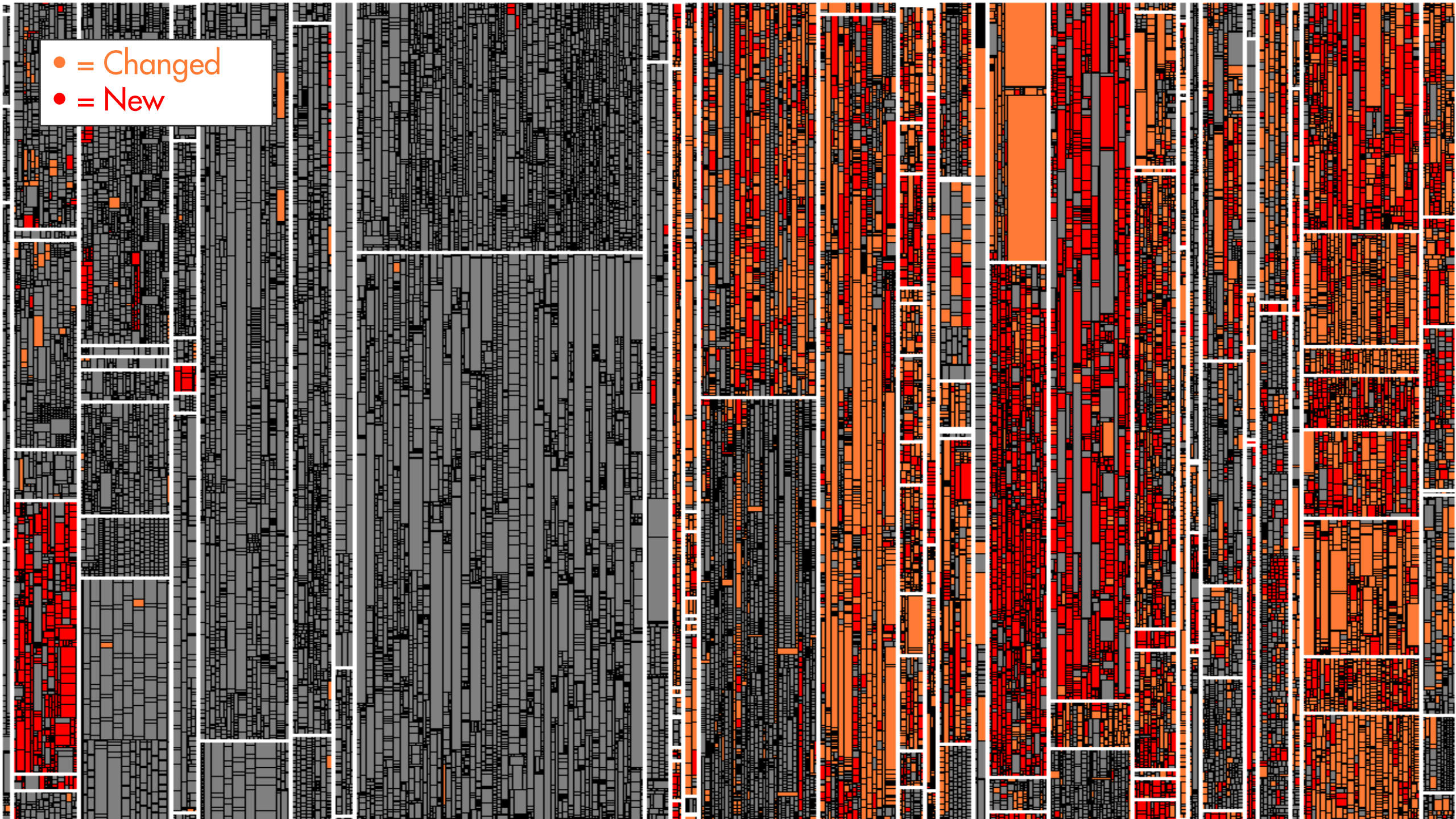


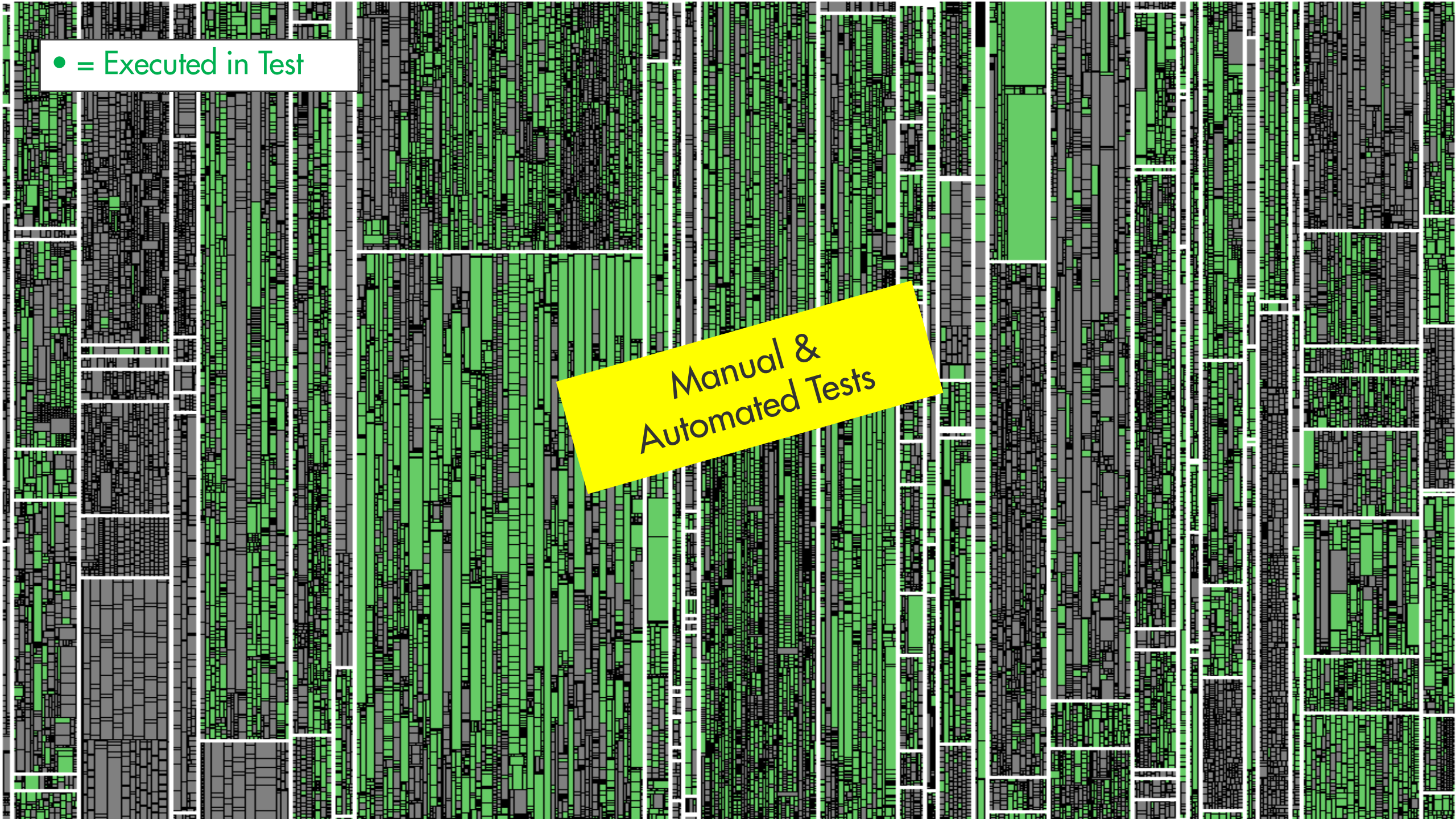


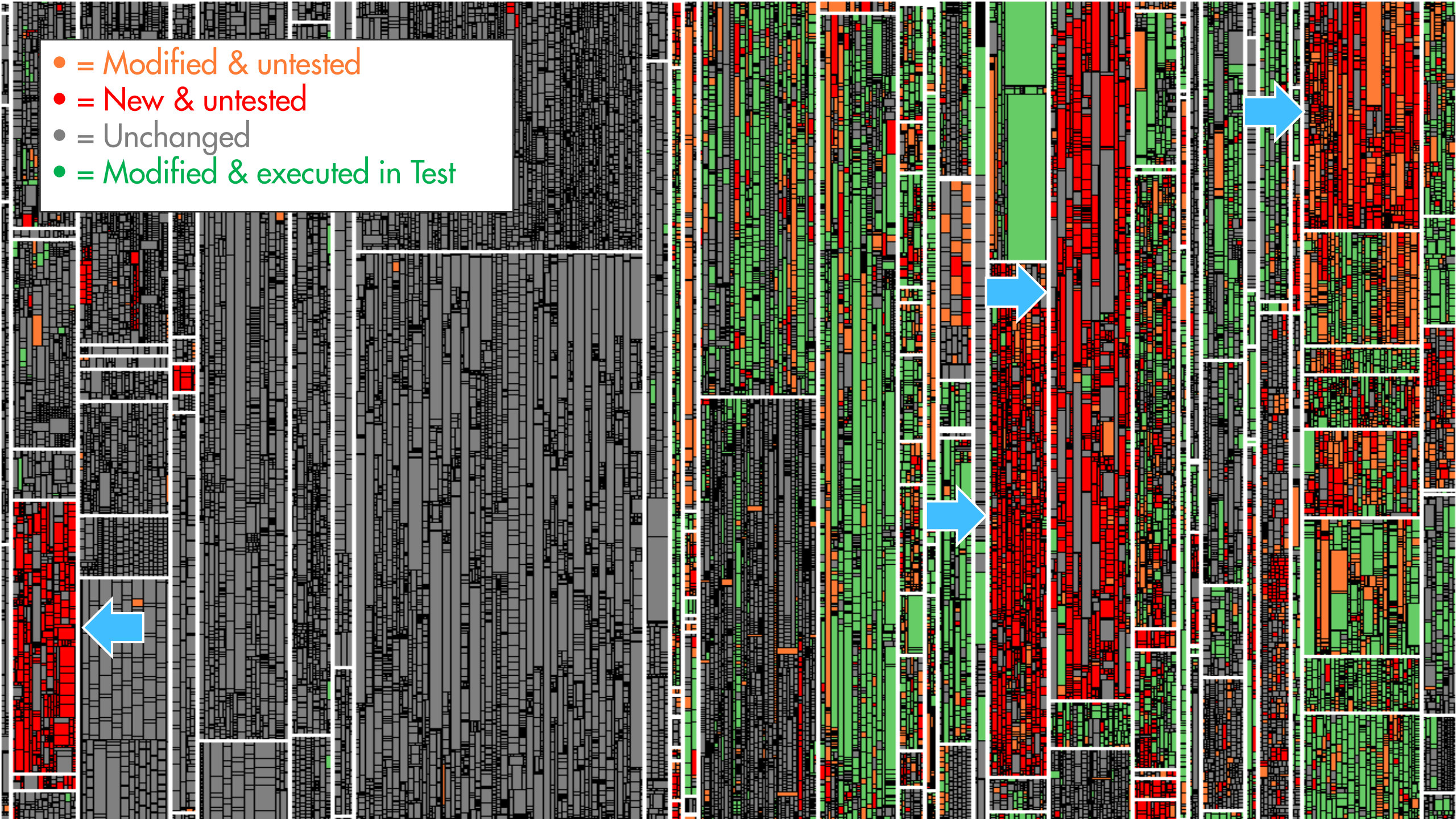





» Wo sind ungetestete Änderungen «
» Welche Tests lohnen sich jetzt am meisten/wenigsten? «
» Wo waren die meisten Fehler? «
» Wo werden Fehler auftreten? «
Test Intelligence
Version History
Tickets
Test Coverage


• = Changed
• = New


• = Executed in Test
Manual &
Automated Tests


• = Modified & untested
• = New & untested
• = Unchanged
• = Modified & executed in Test

Issue #	Subject			Test Gap
TS-10549	Undo/Redo for web-based architecture editor	Done		0%
TS-10784	Fix long method finding in TaintAnalysisRunner	Done		0%
TS-10923	Implement metric 'Nesting Depth' for Simulink	Done		29%
TS-11364	External findings are not registered during first upload	Done		14%
TS-11942	Manual test coverage upload during development	Done		43%
TS-12050	Tool for transferring findings blacklists and tasks	Done		50%
TS-12262	Cannot set or alter alias without reanalysis	Done		0%
TS-13151	Fetch parent relationship of TFS work items	Done		0%

Issue #	Subject			Test Gap
TS-14421	Get rid of TestGapSynchronizer block	Done		0%
TS-14733	Remove Dataflow blocks	Done		22%

Done **Issue TS-14733 - Remove Dataflow blocks**

Creator:  (on Apr 06 2018 19:44)

Last update: Aug 24 2018 09:32

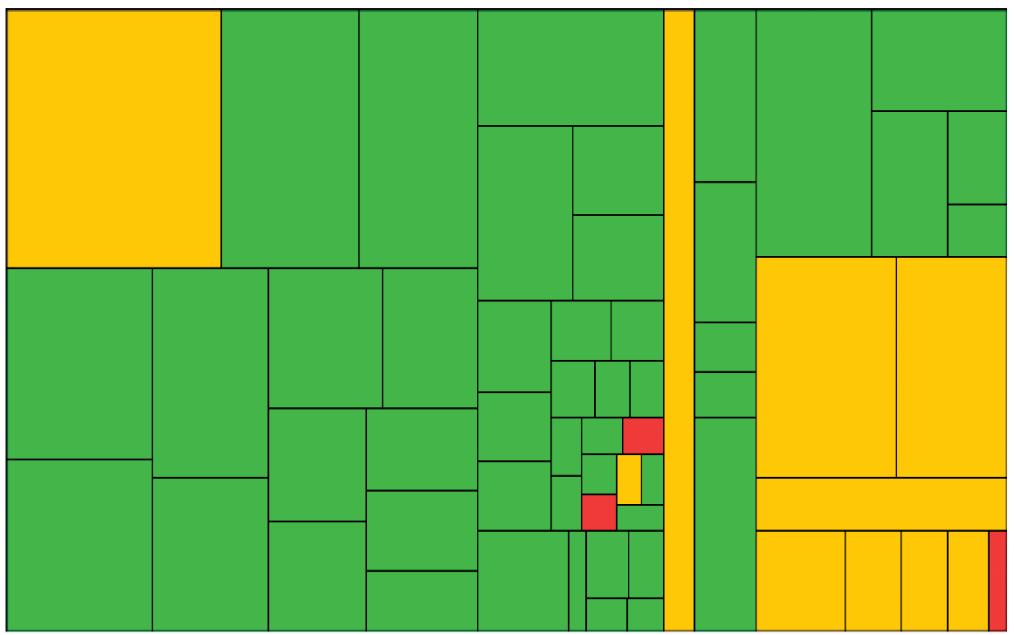
Assignee: 

Project	Type	Priority	Resolution	Fix Version
TS	Maintenance	Normal	Green	Teamscale 4.5

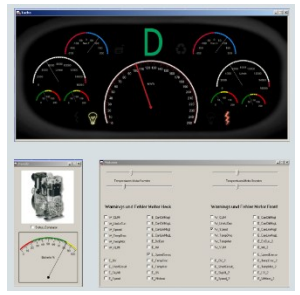
Component	Labels	Affected Version	Customer	Customer Issue
Backend	Performance			

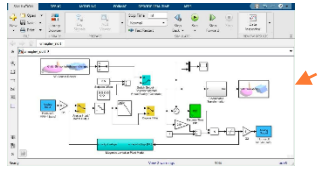
Epic Name	Freshdesk URL	Merge Request
		https://git.cqse.eu/cqse/teamscale/3621

Aug 15 2018 12:37–Now | Test Gap: 22%



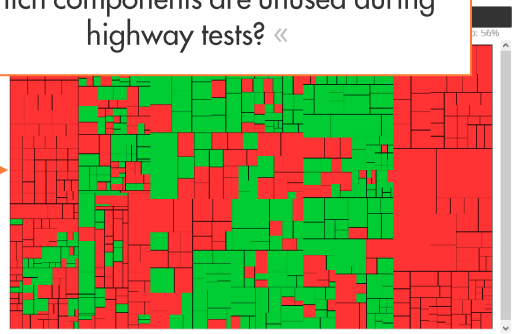
Test Gaps in different Test Environments

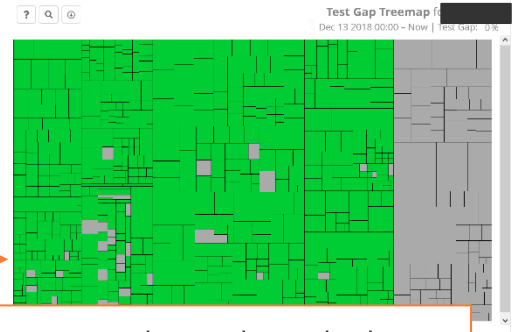




Test Drive
Prototype
HIL Tests
MIL/SIL Tests
Integration Tests
Unit Tests

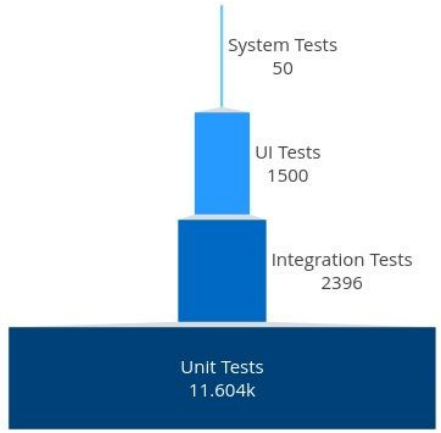
» Which components are unused during highway tests? «





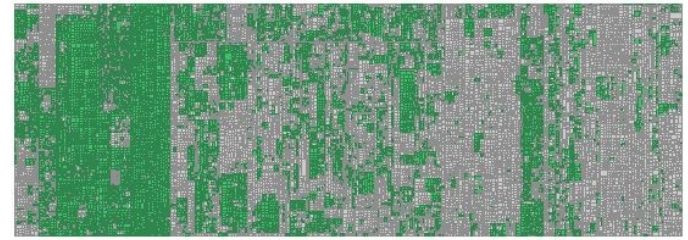
» Are we compliant with standards? «

Test Pyramid



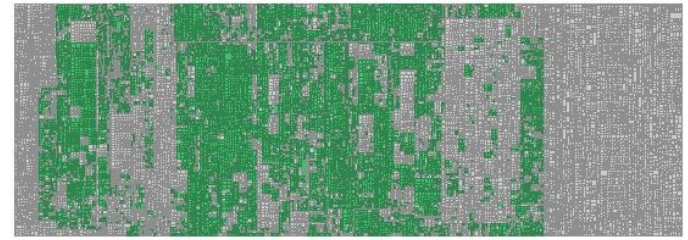
Unit Test Coverage for teamscale

Oct 06 2024 01:00 - Nov 05 2024 10:08 | Execution Ratio: 45.3%



Integration Test Coverage for teamscale

Oct 06 2024 01:00 - Nov 05 2024 10:08 | Execution Ratio: 41.2%



Test Count for teamsca...

16.8k

+152

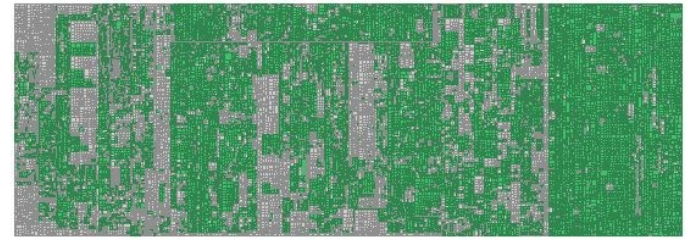
Test Duration for team...

3h 48m 37s 741ms

+47.6

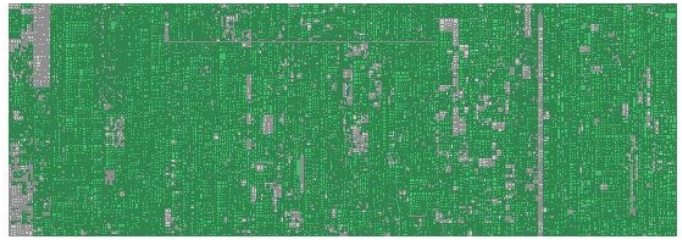
UI Test Coverage for teamscale

Oct 06 2024 01:00 - Nov 05 2024 10:08 | Execution Ratio: 60.8%



Combined Coverage for teamscale

Oct 06 2024 01:00 - Nov 05 2024 10:08 | Execution Ratio: 85.3%

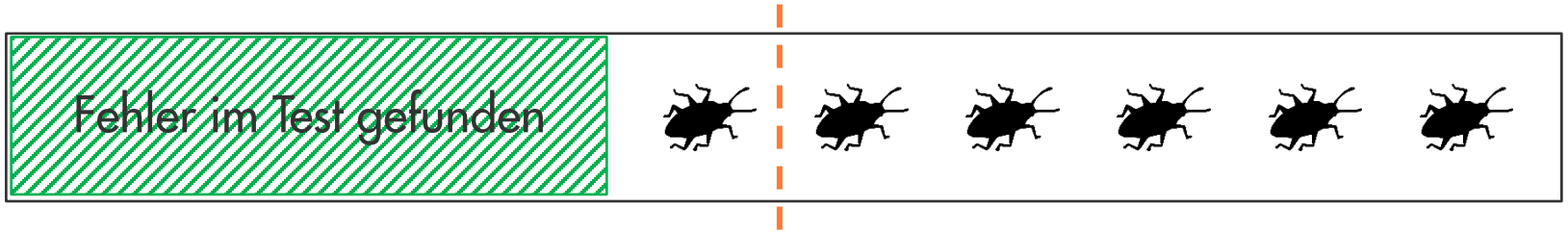


System Test Coverage for teamscale

Oct 06 2024 01:00 - Nov 05 2024 10:08 | Execution Ratio: 41.1%

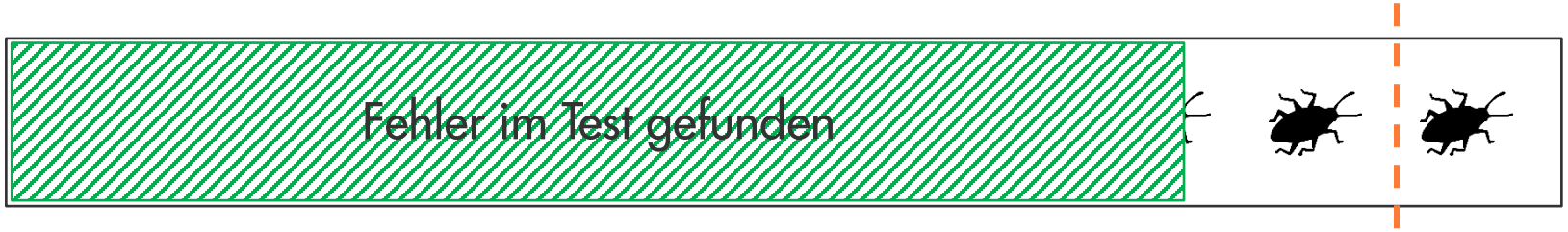
Ohne Test-Gap-Analyse

%Restfehler = 60%



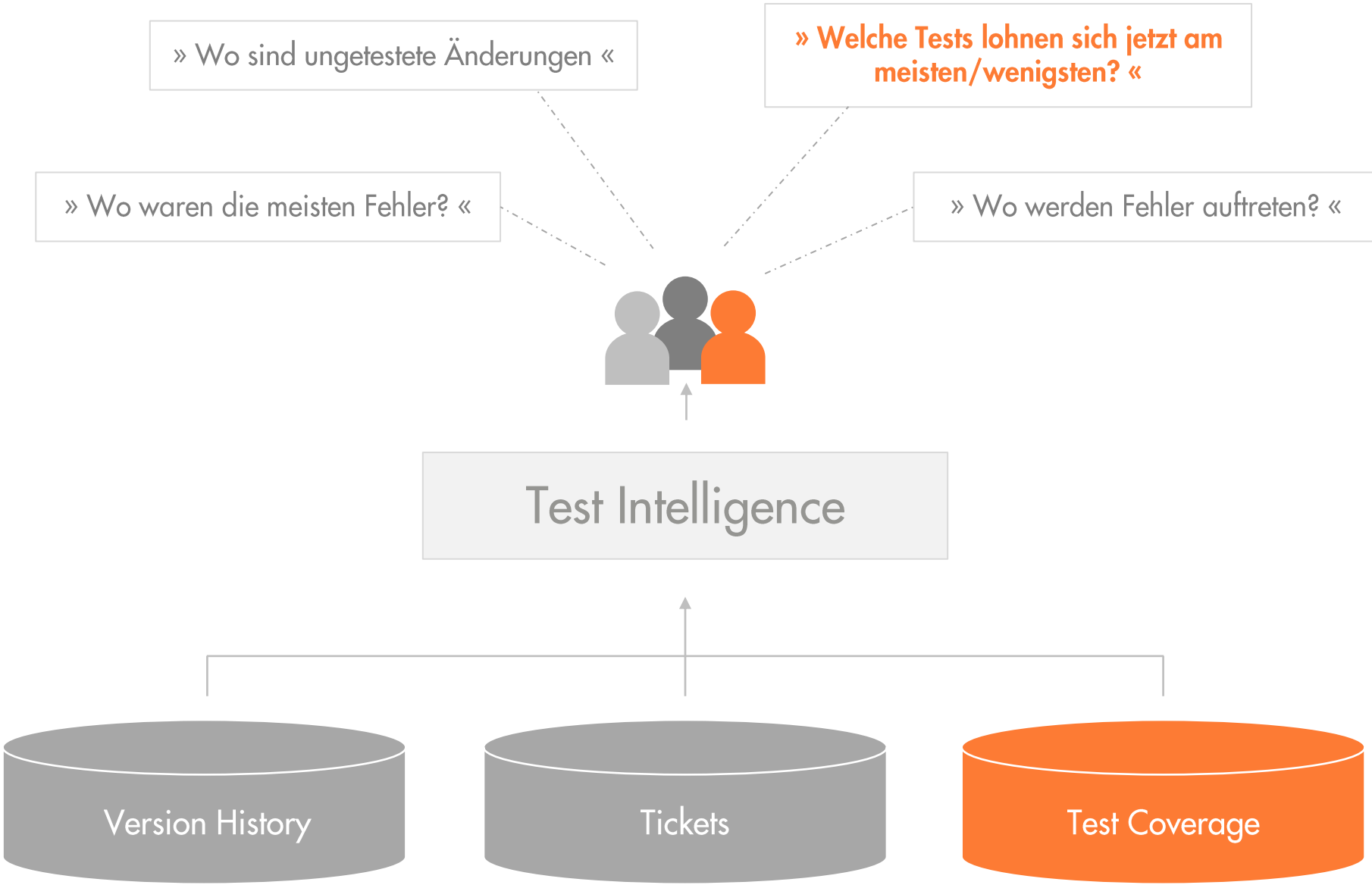
Mit Test-Gap-Analyse

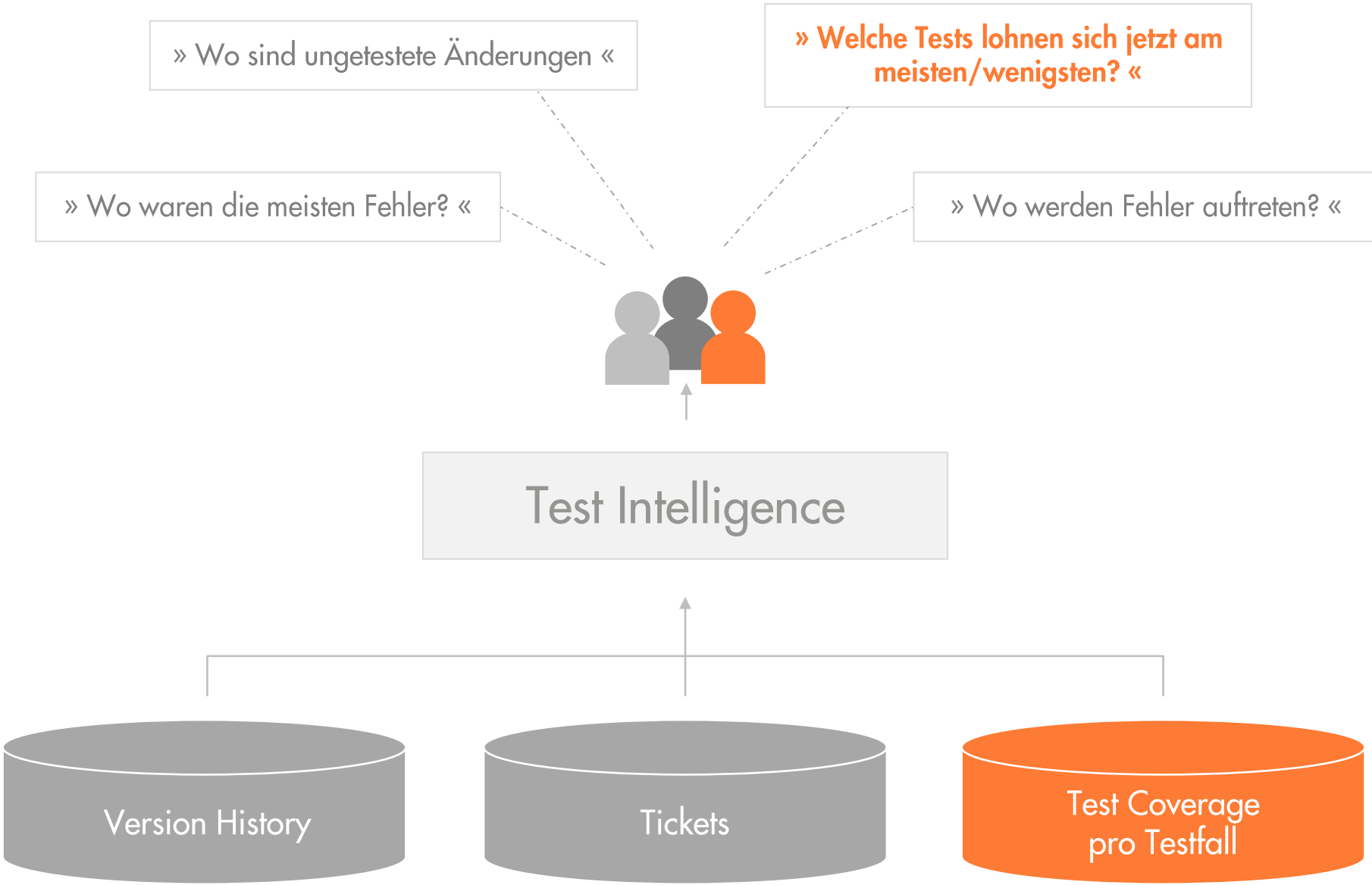
%Restfehler = 28%

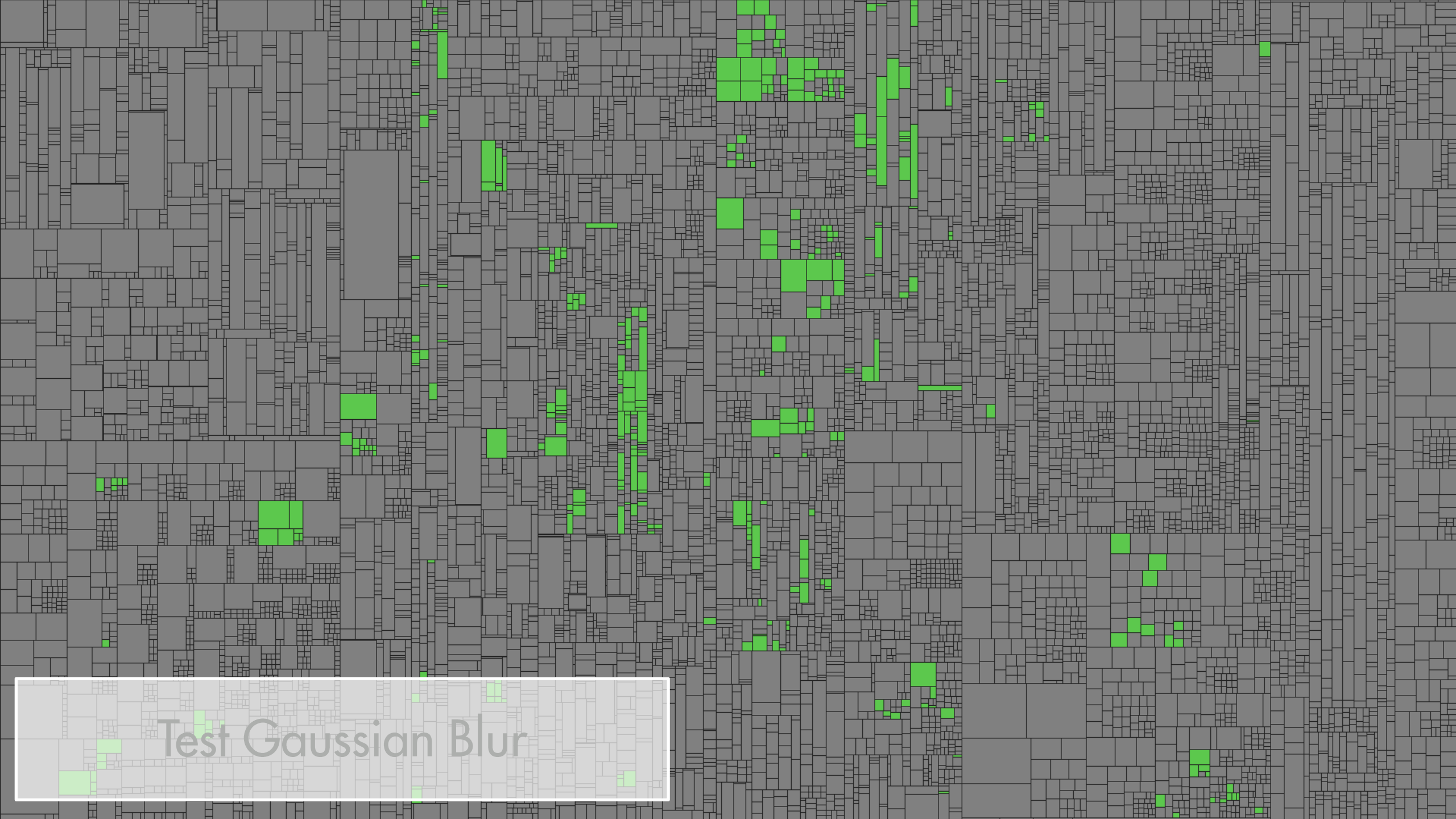


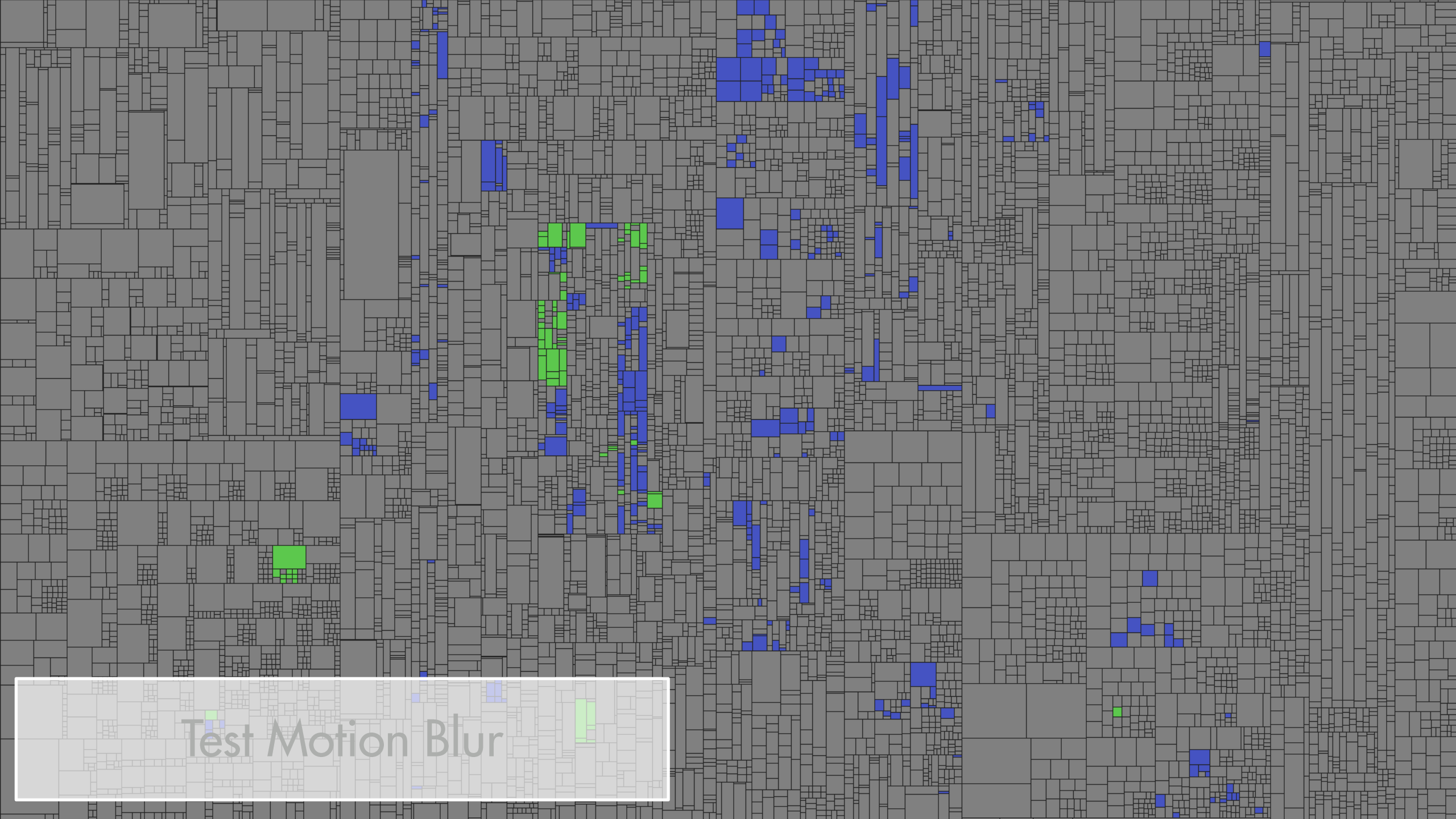
Reduzierte Feldfehler = **50%**

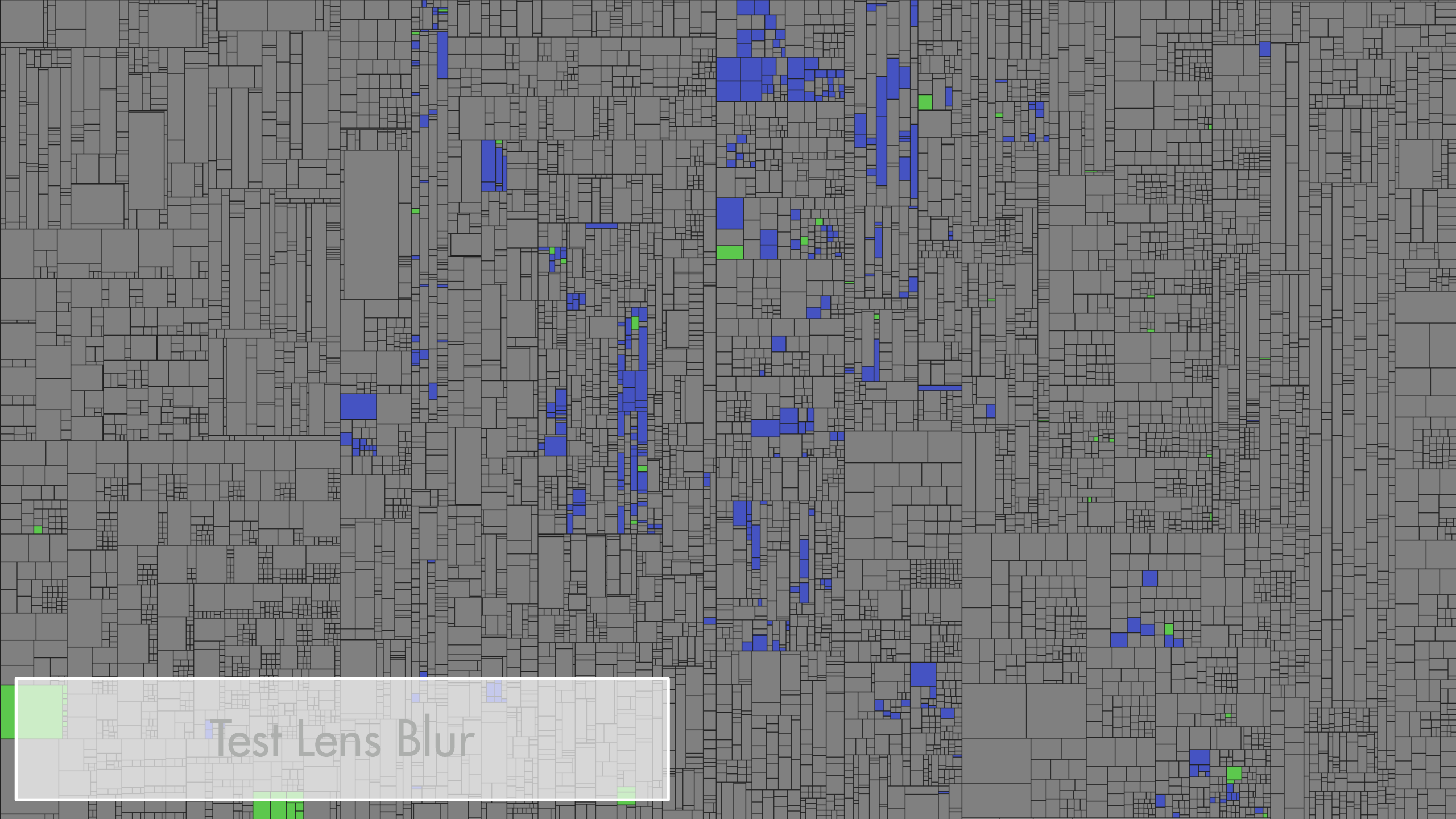
Test-Gap-Analyse reduziert Feldfehler in den Applikationen der Munich Re um ½

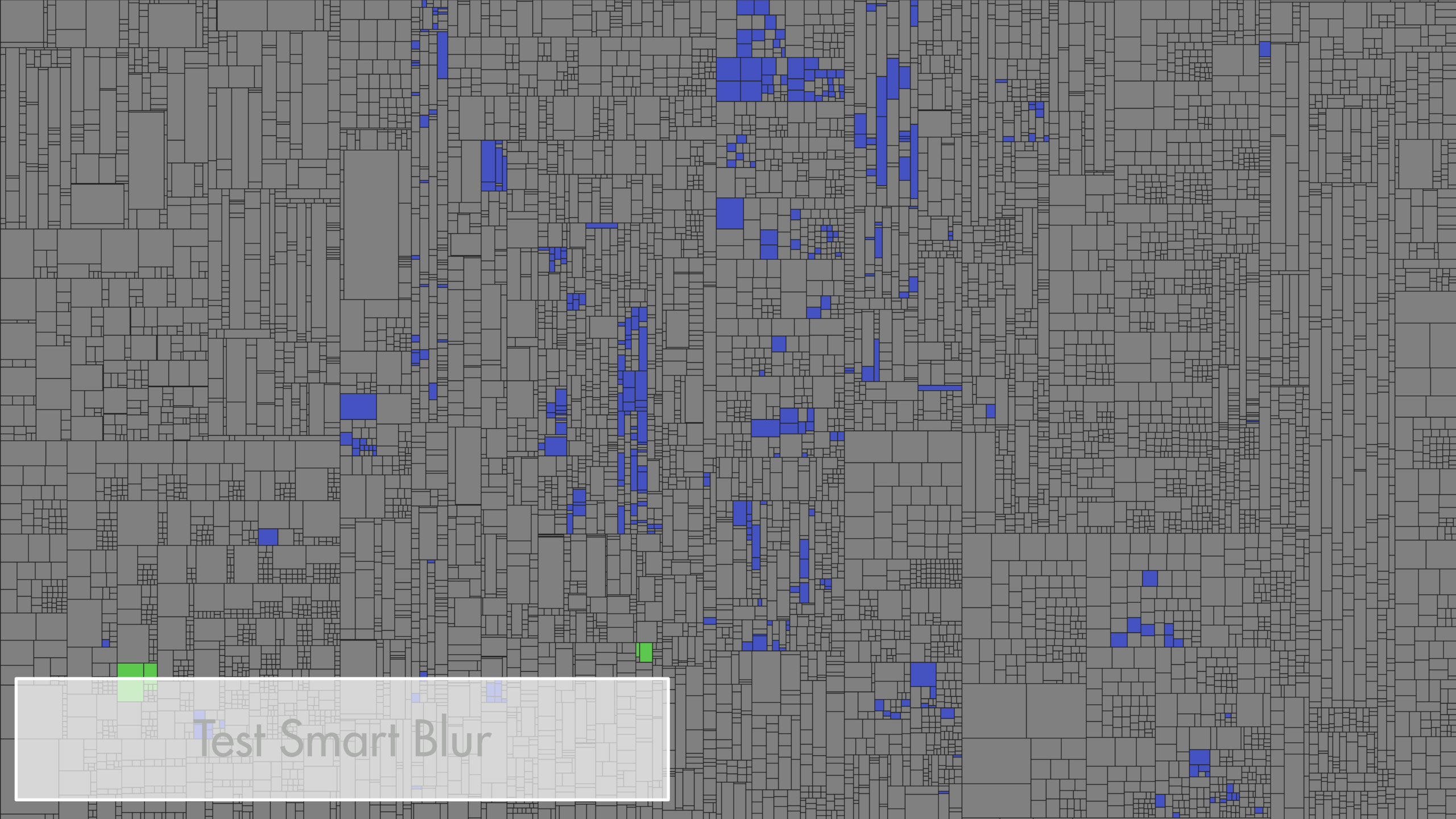

» Wo sind ungetestete Änderungen «
» Welche Tests lohnen sich jetzt am meisten/wenigsten? «
» Wo waren die meisten Fehler? «
» Wo werden Fehler auftreten? «
Test Intelligence
Version History
Tickets
Test Coverage

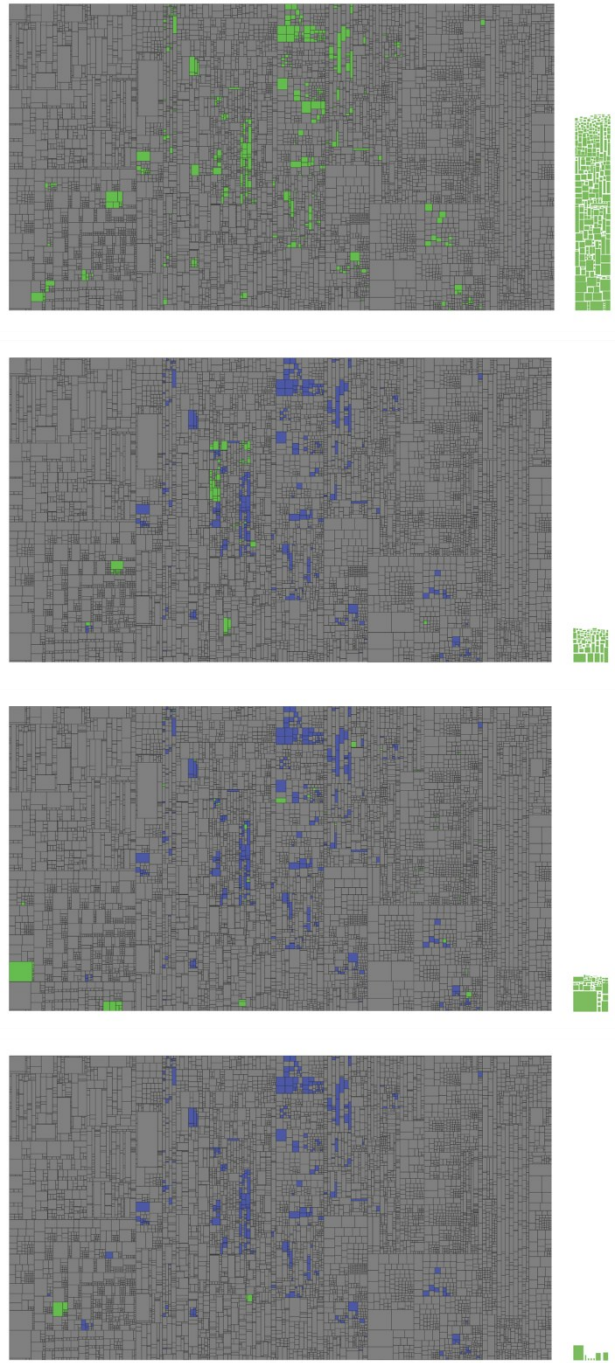

» Wo sind ungetestete Änderungen «
» Welche Tests lohnen sich jetzt am meisten/wenigsten? «
» Wo waren die meisten Fehler? «
» Wo werden Fehler auftreten? «
Test Intelligence
Version History
Tickets
Test Coverage pro Testfall


Test Gaussian Blur

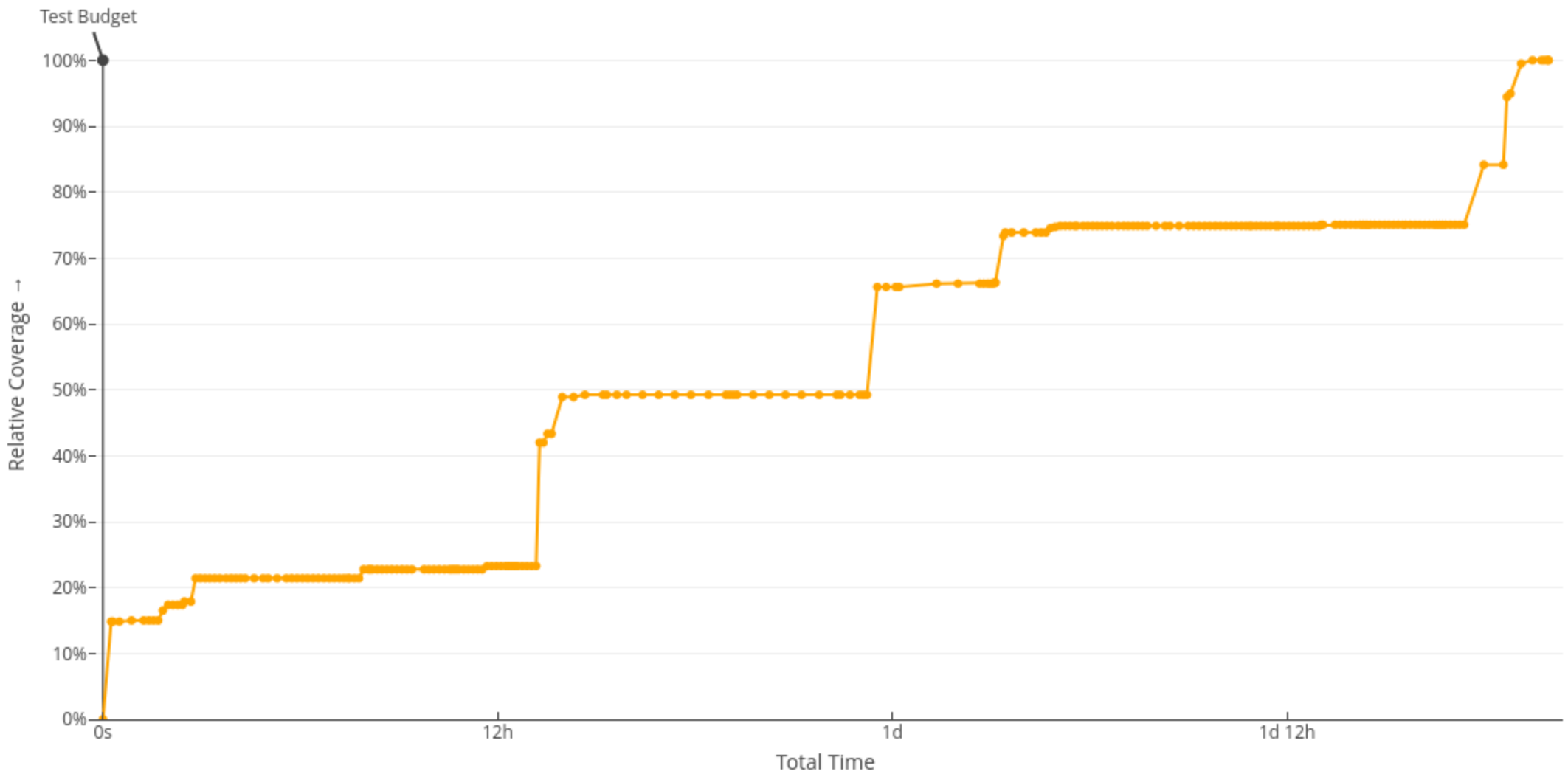

Test Motion Blur


Test Lens Blur


Test Smart Blur



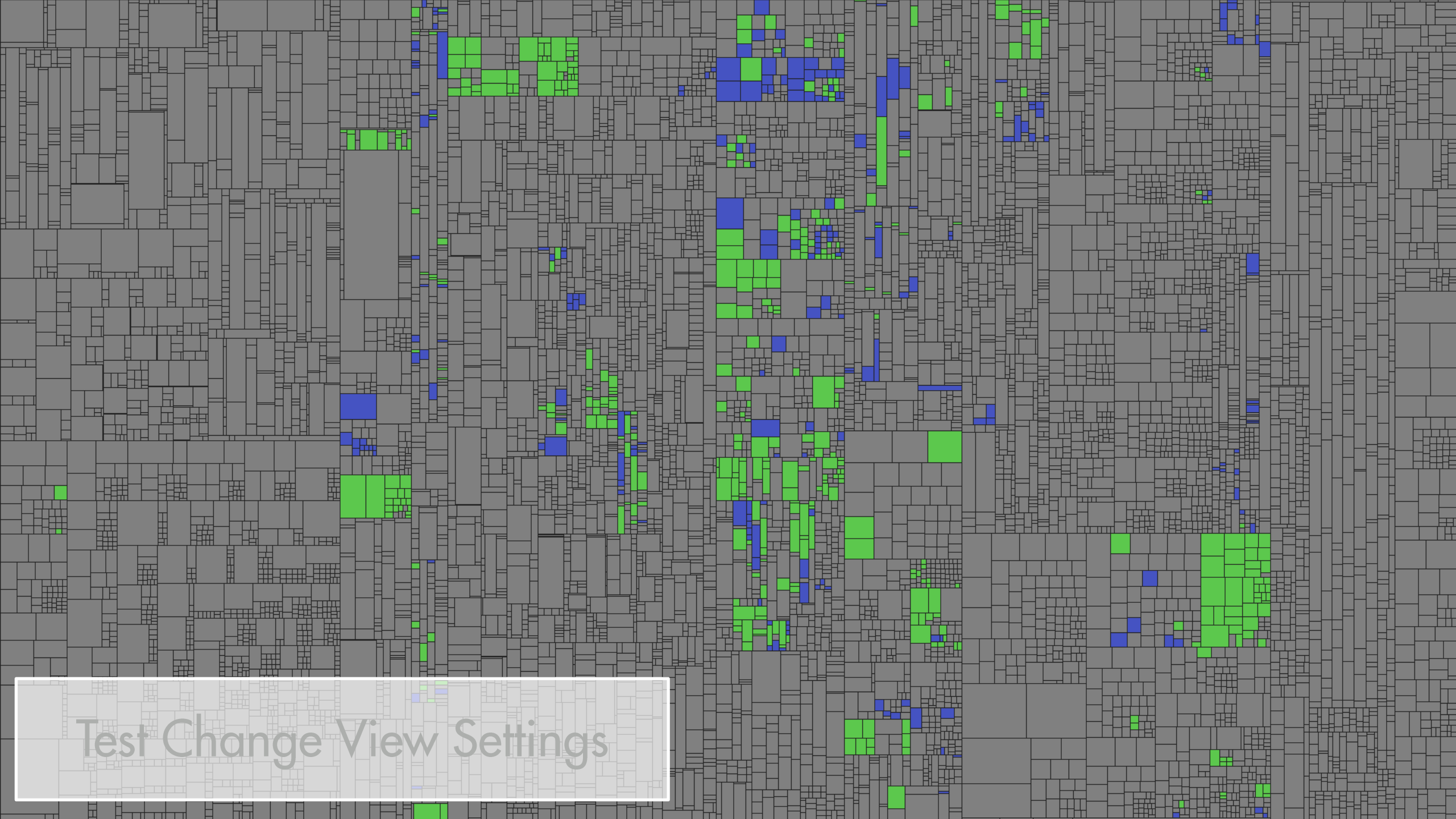
Coverage over Time

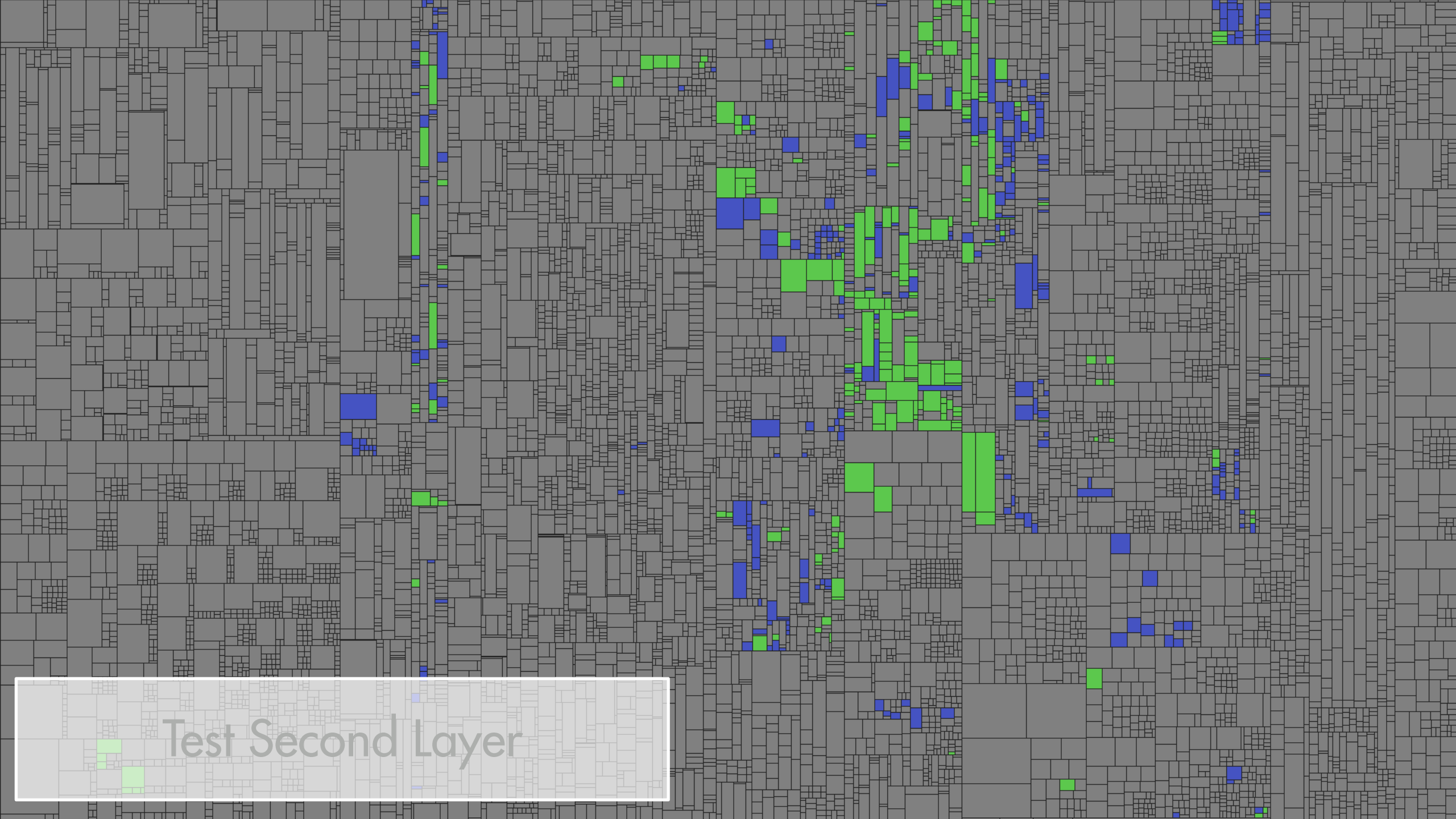


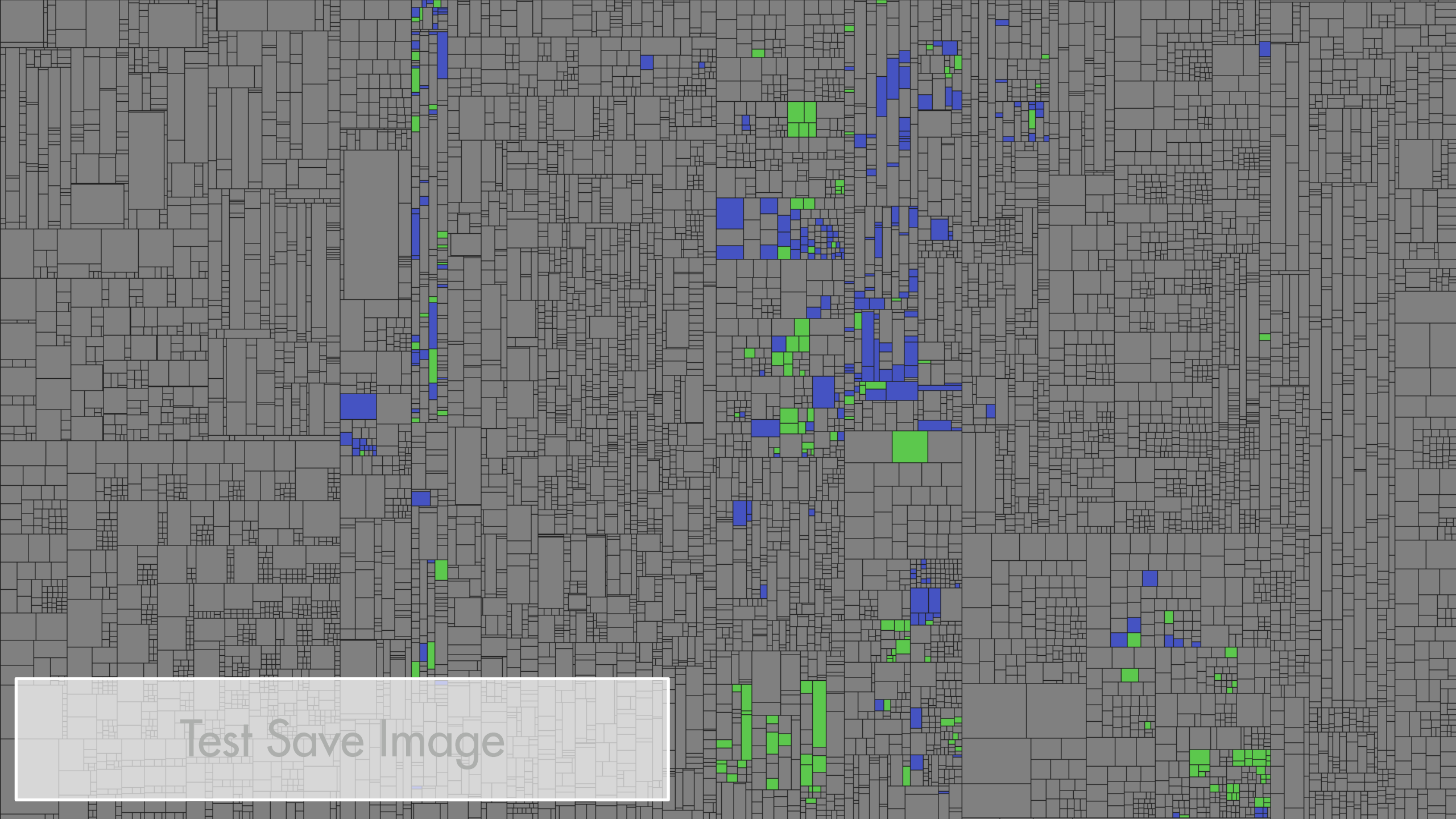
Results for Test Query & Budget Restriction

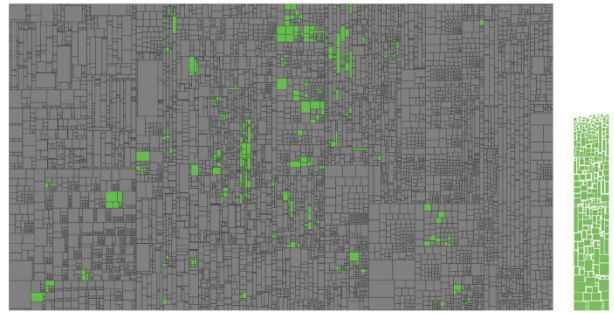
Relative Coverage: 0%, Selected Tests: 0 out of 236 (0%)

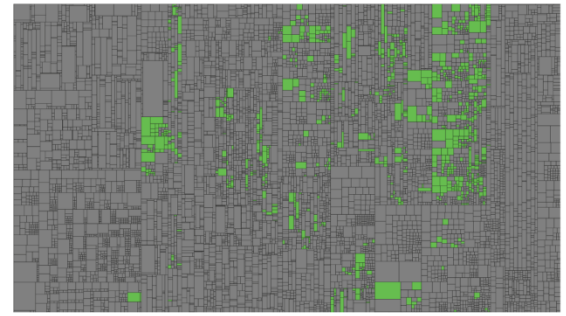

Test Create and Modify
Selection

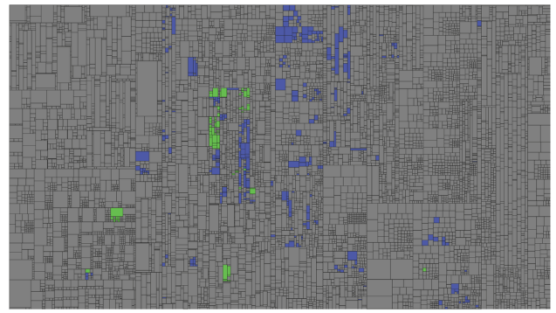

Test Change View Settings

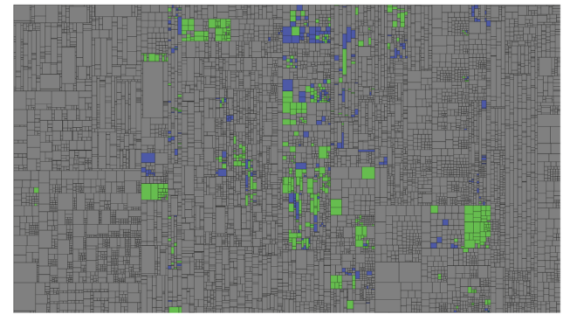

Test Second Layer

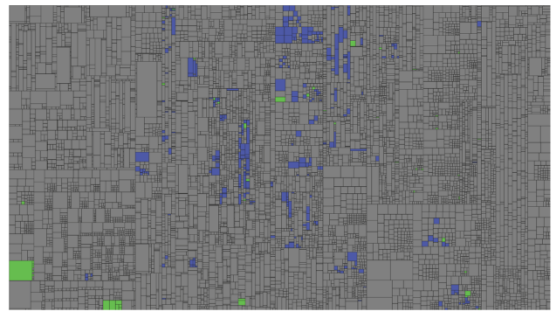

Test Save Image

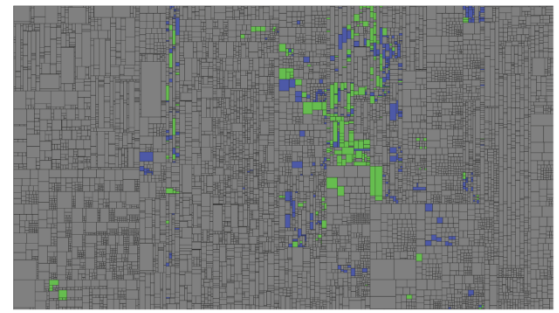


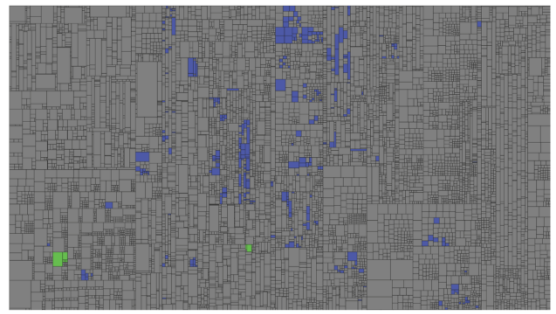


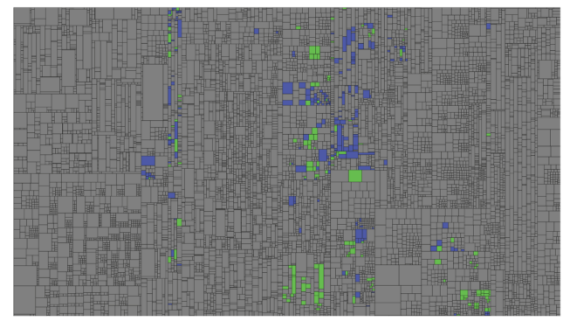





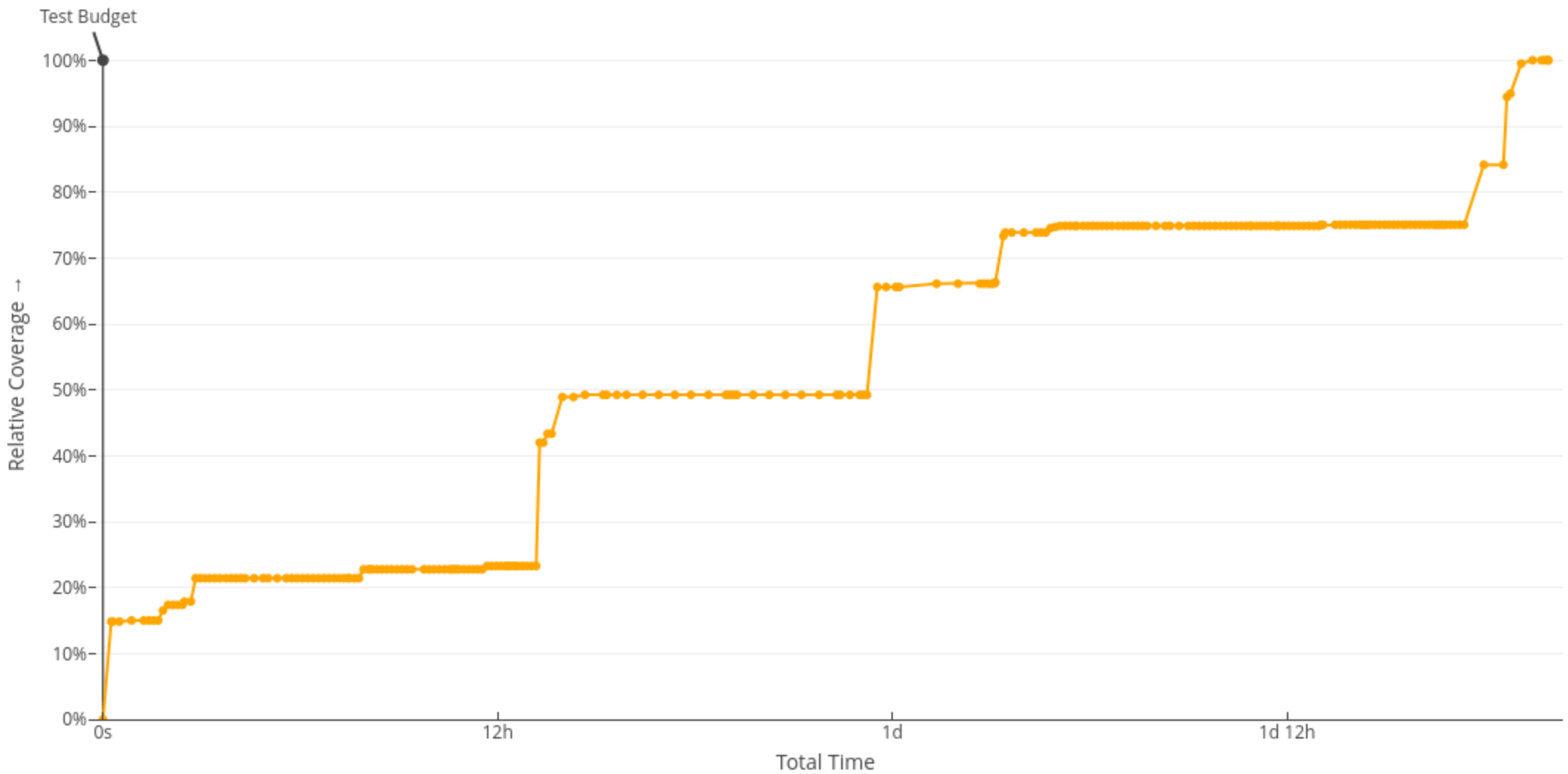








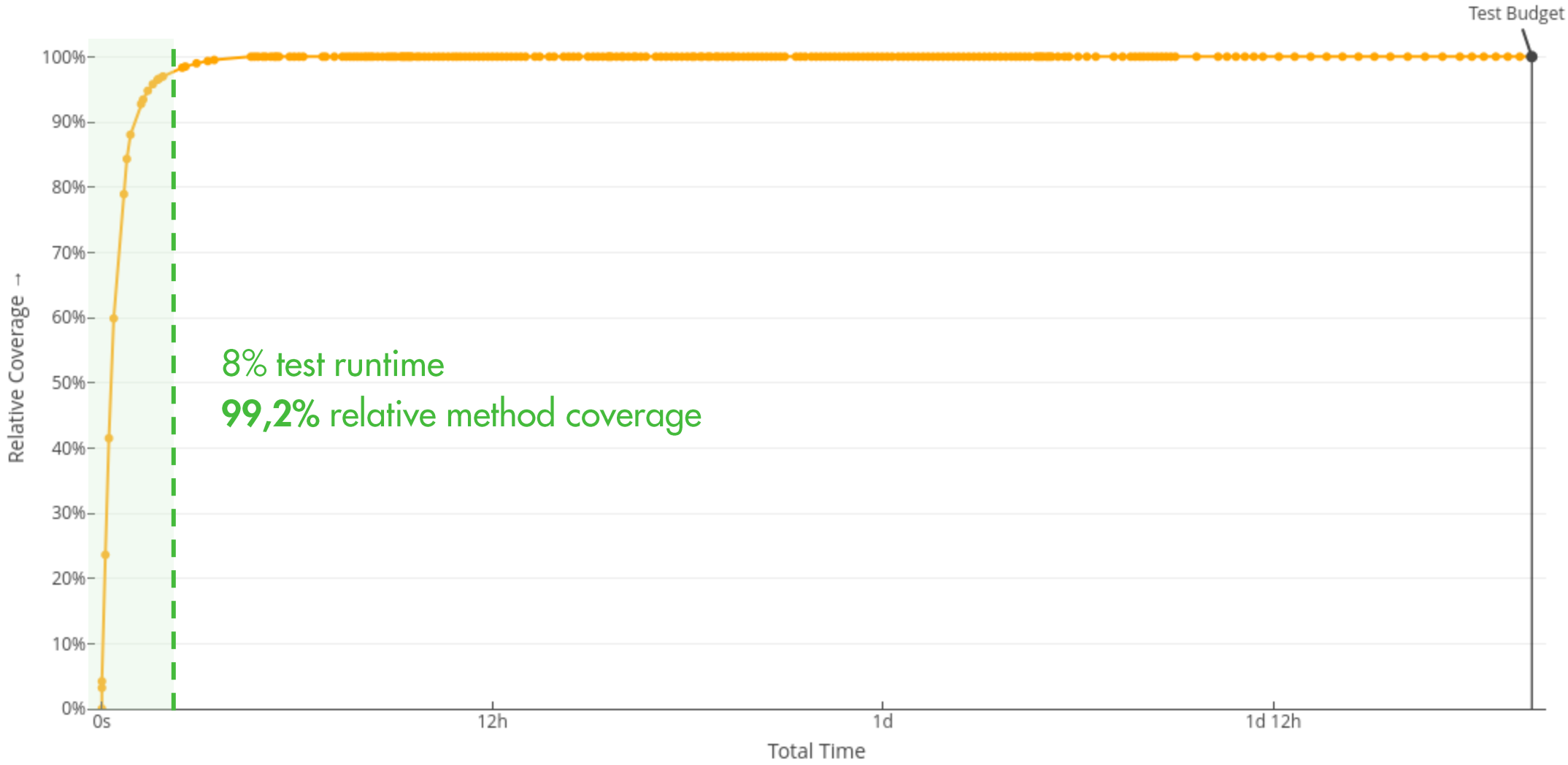

Coverage over Time



Results for Test Query & Budget Restriction

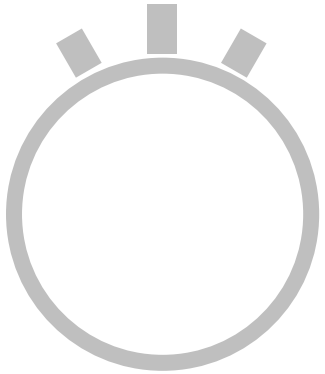
Relative Coverage: 0%, Selected Tests: 0 out of 236 (0%)

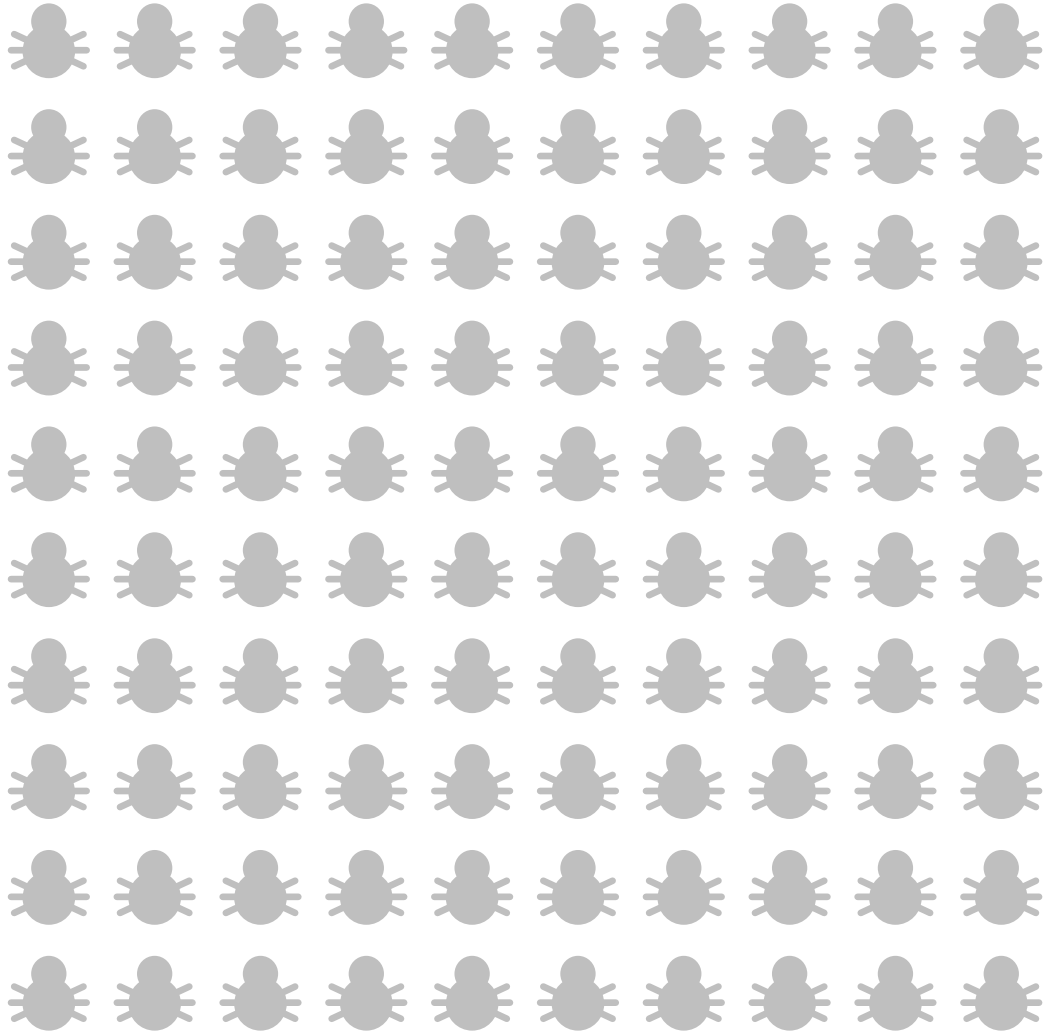
Coverage over Time



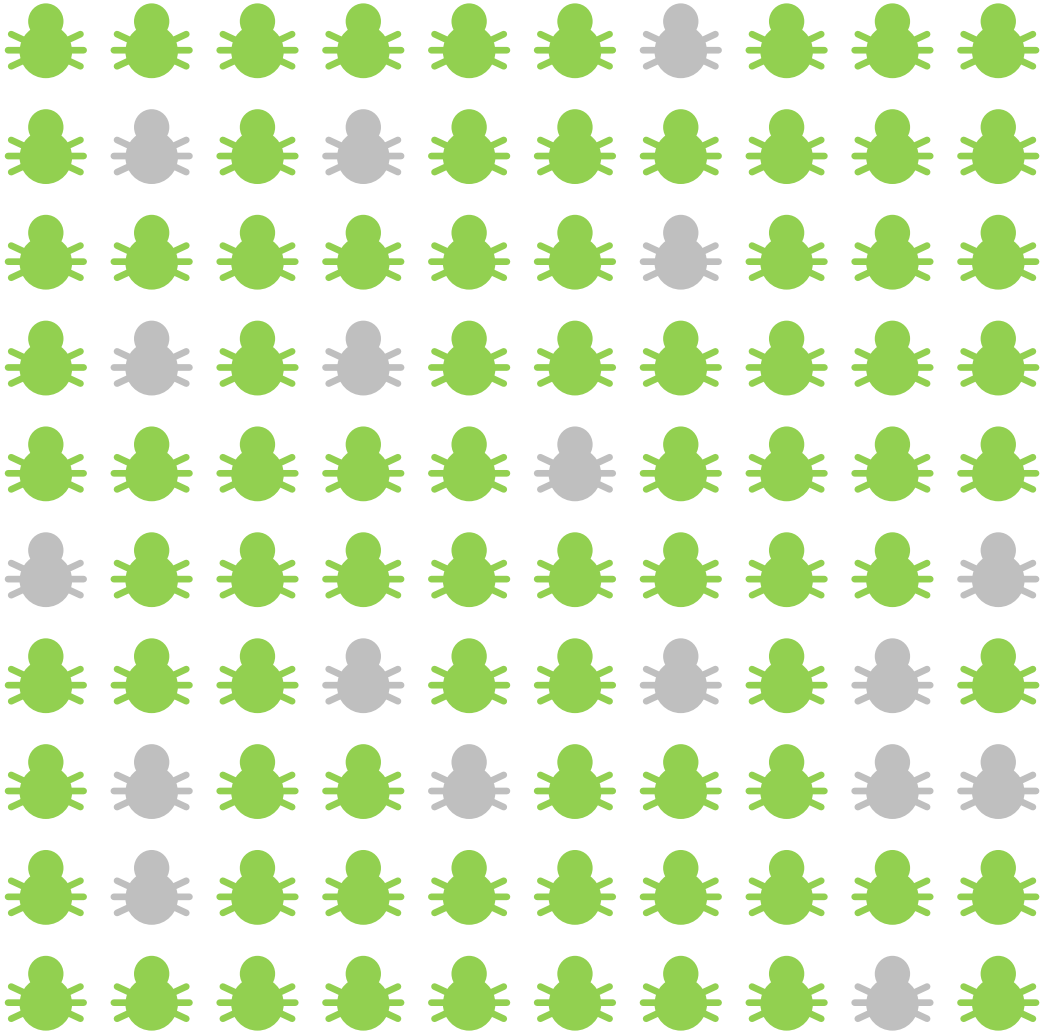
Results for Test Query & Budget Restriction

Relative Coverage: 100%, Selected Tests: 236 out of 236 (100%)

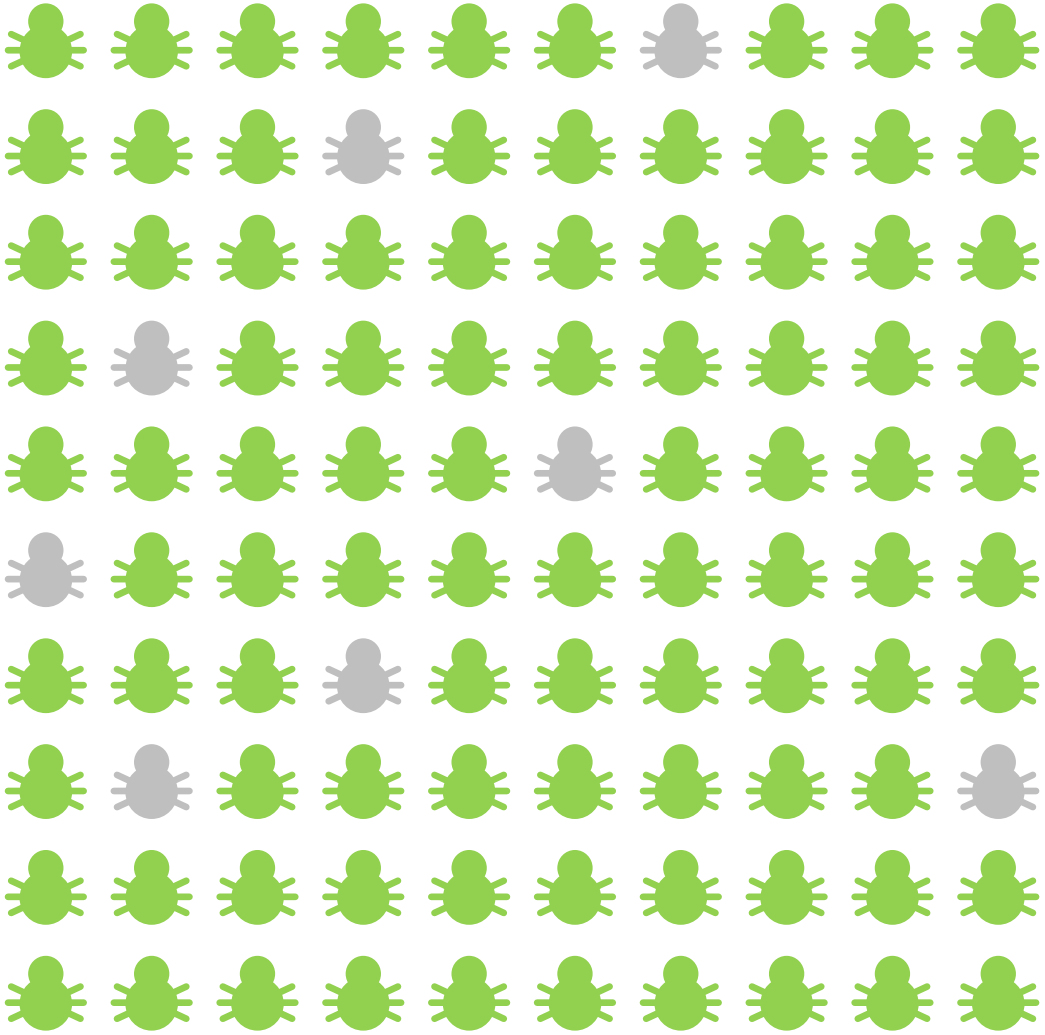


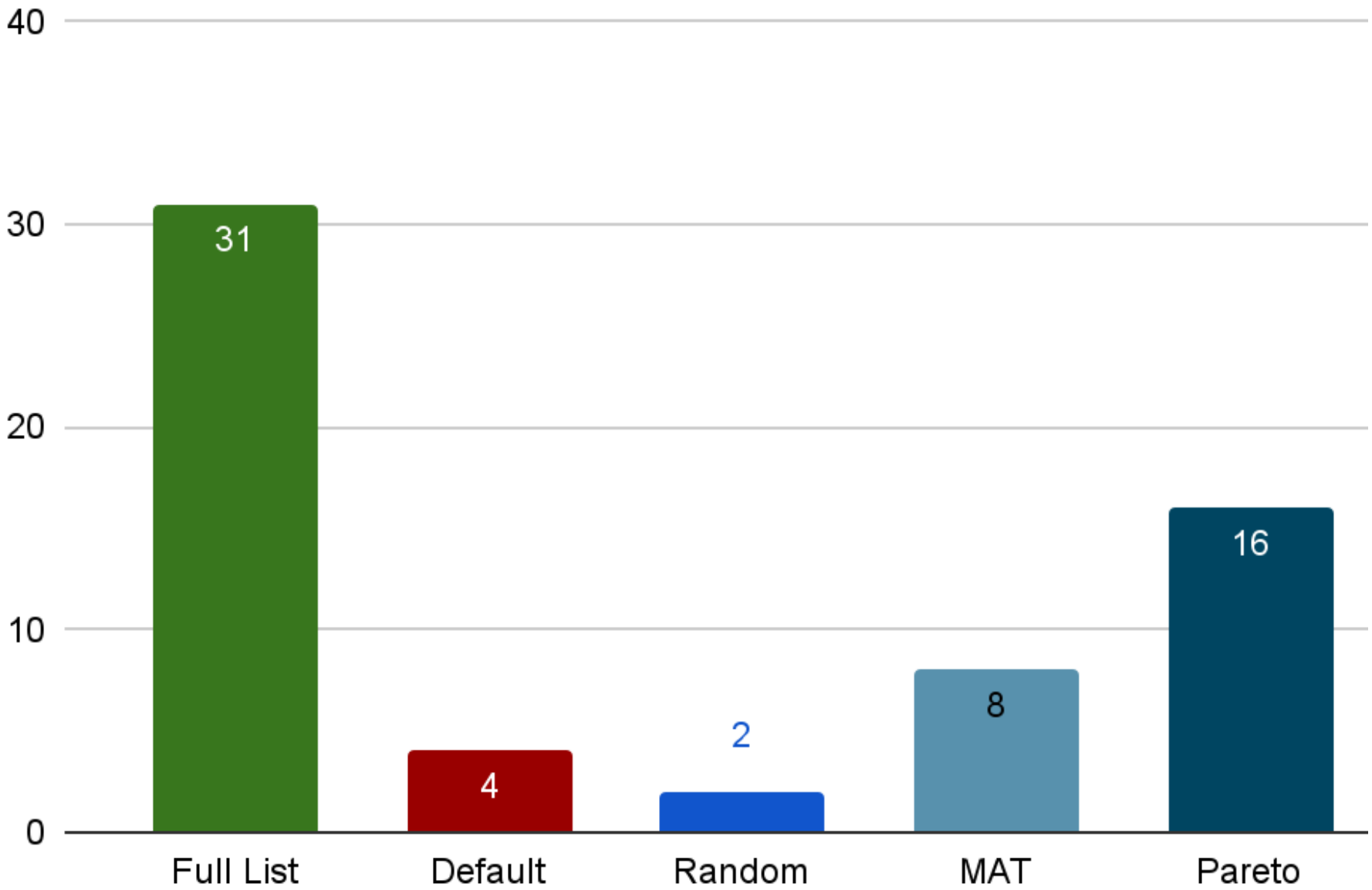


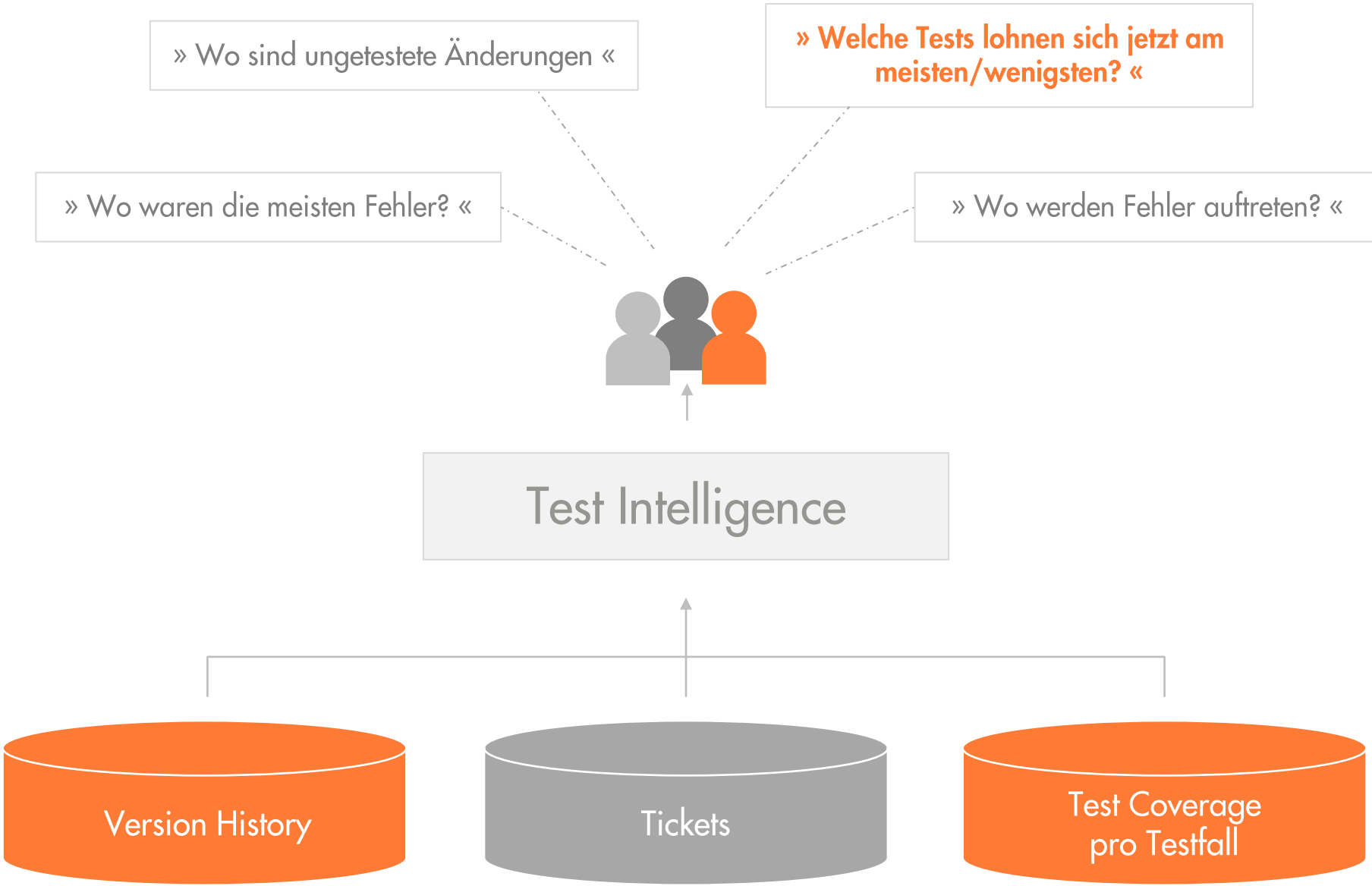
80 %

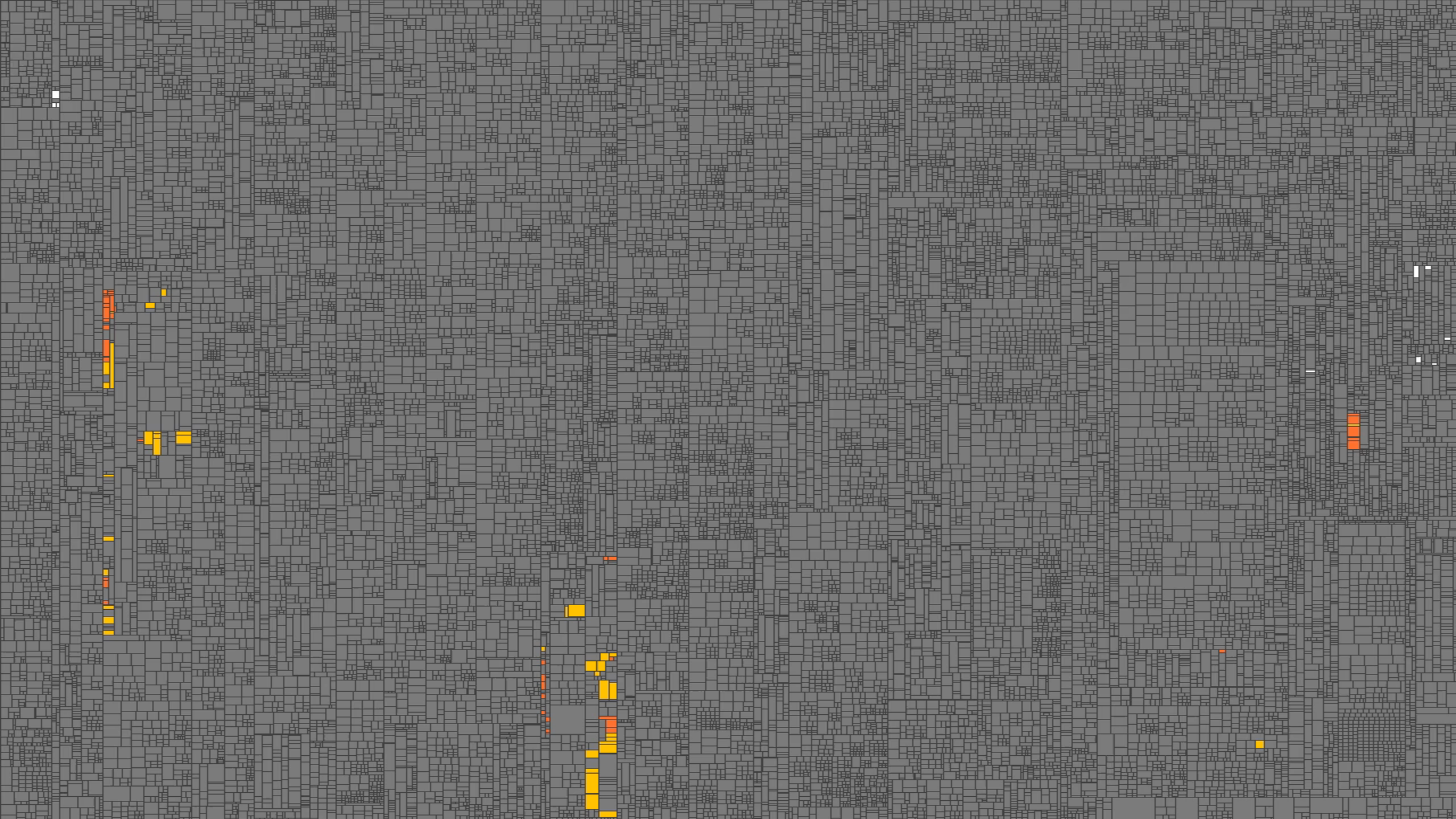


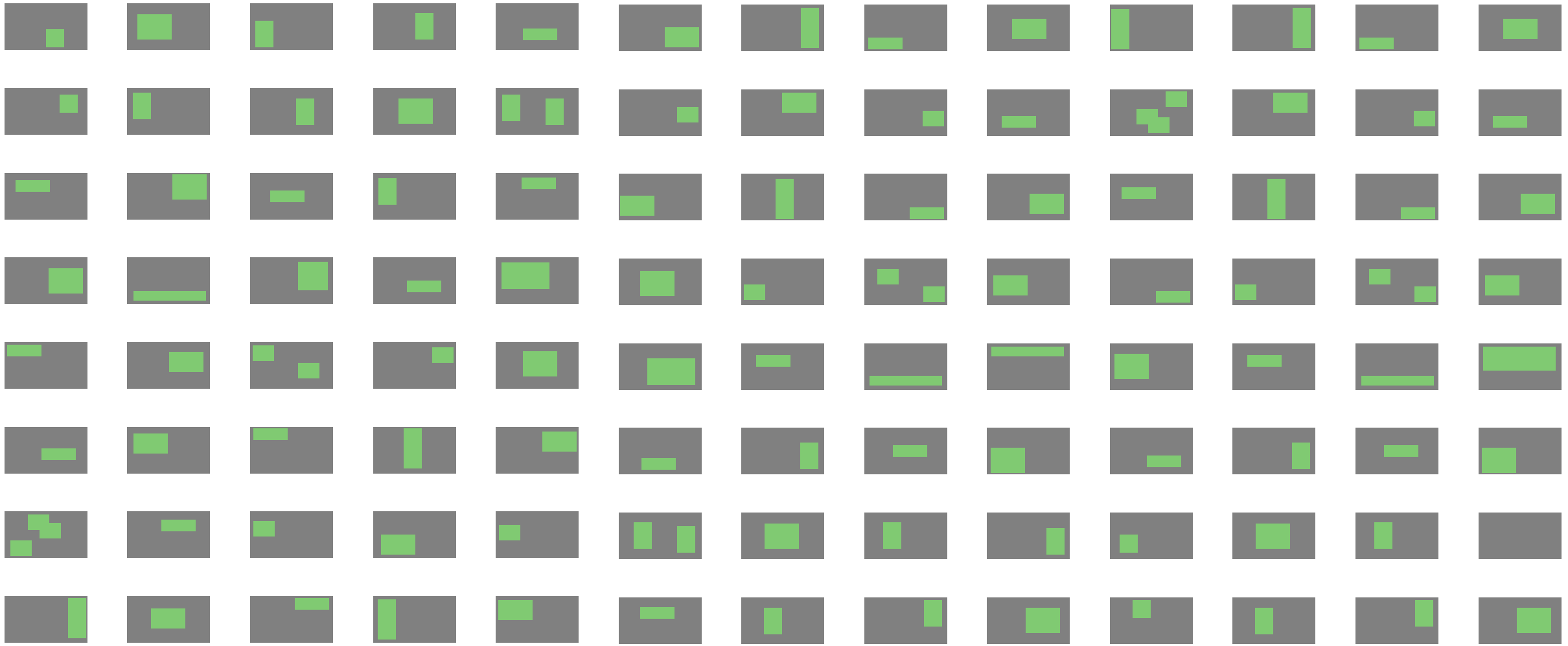
90 %



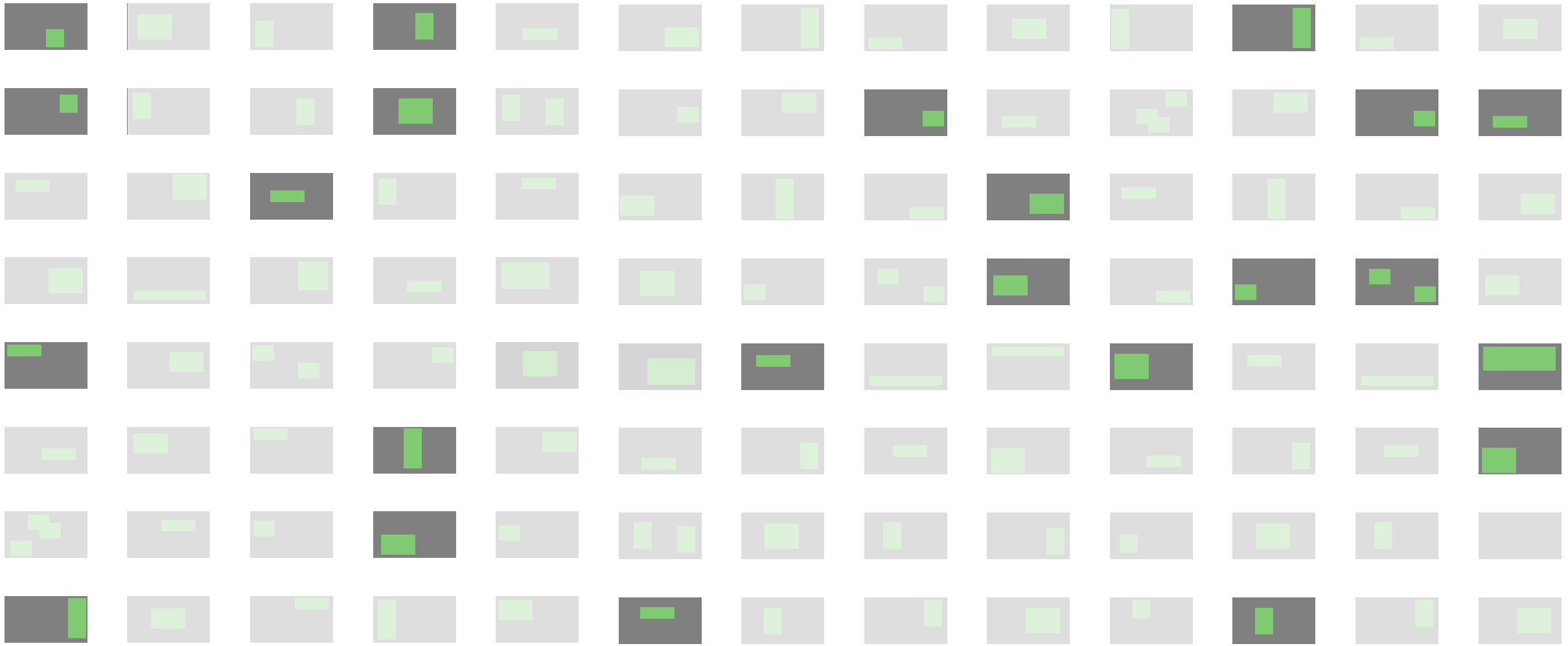

40
30
20
10
0
31
4
2
8
16
Full List
Default
Random
MAT
Pareto


» Wo sind ungetestete Änderungen «
» Welche Tests lohnen sich jetzt am meisten/wenigsten? «
» Wo waren die meisten Fehler? «
» Wo werden Fehler auftreten? «
Test Intelligence
Version History
Tickets
Test Coverage pro Testfall

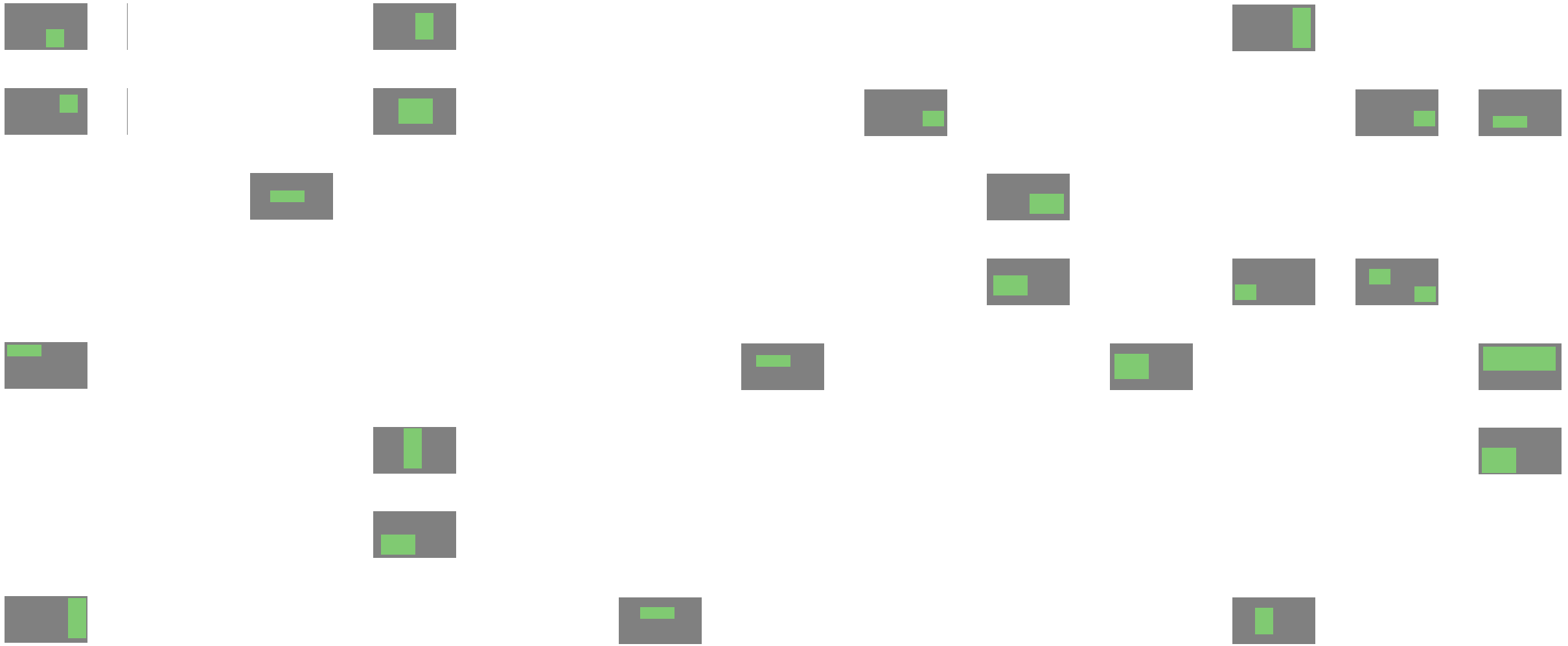




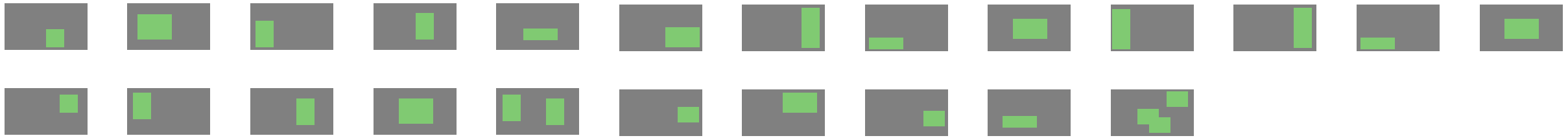
Schritt 1: Selektion betroffener Testfälle



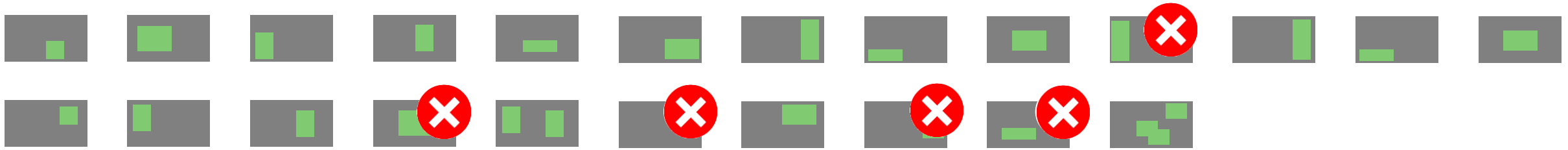
Schritt 1: Selektion betroffener Testfälle



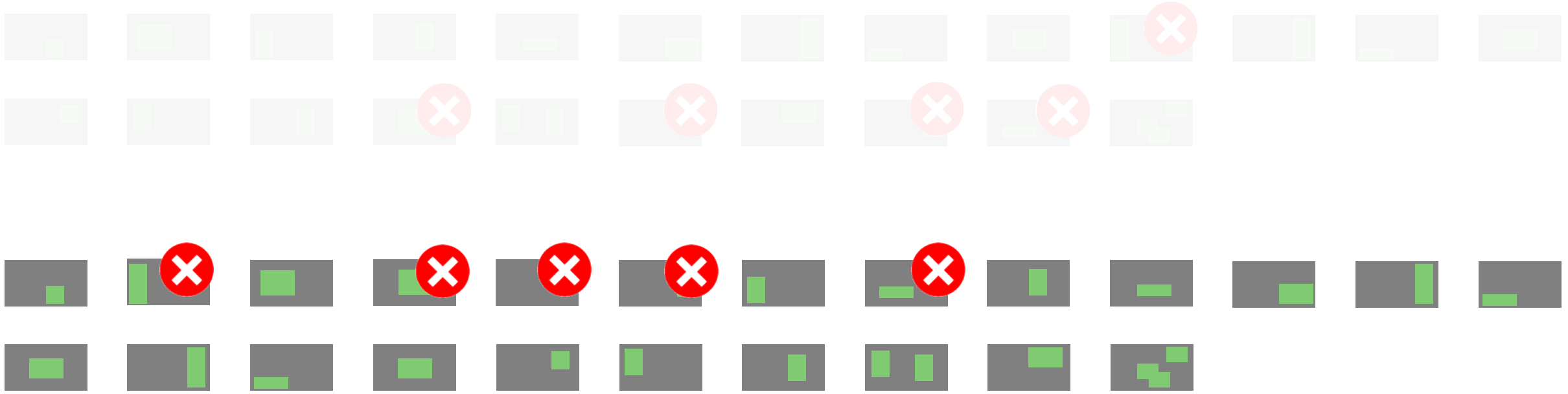
Schritt 1: Selektion betroffener Testfälle

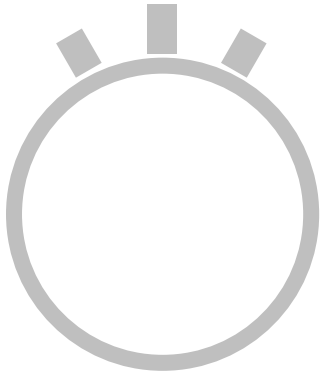


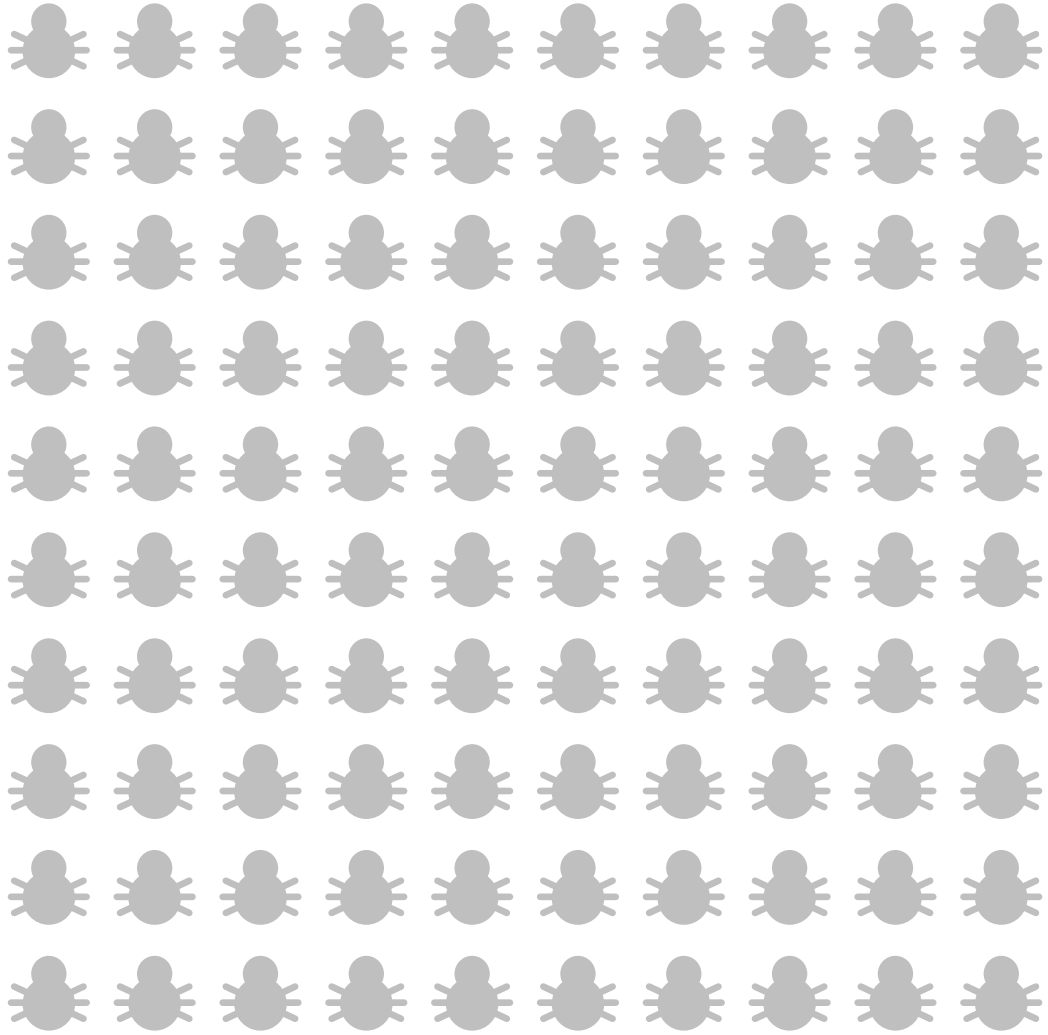
Schritt 2: Priorisierung selektierter Testfälle



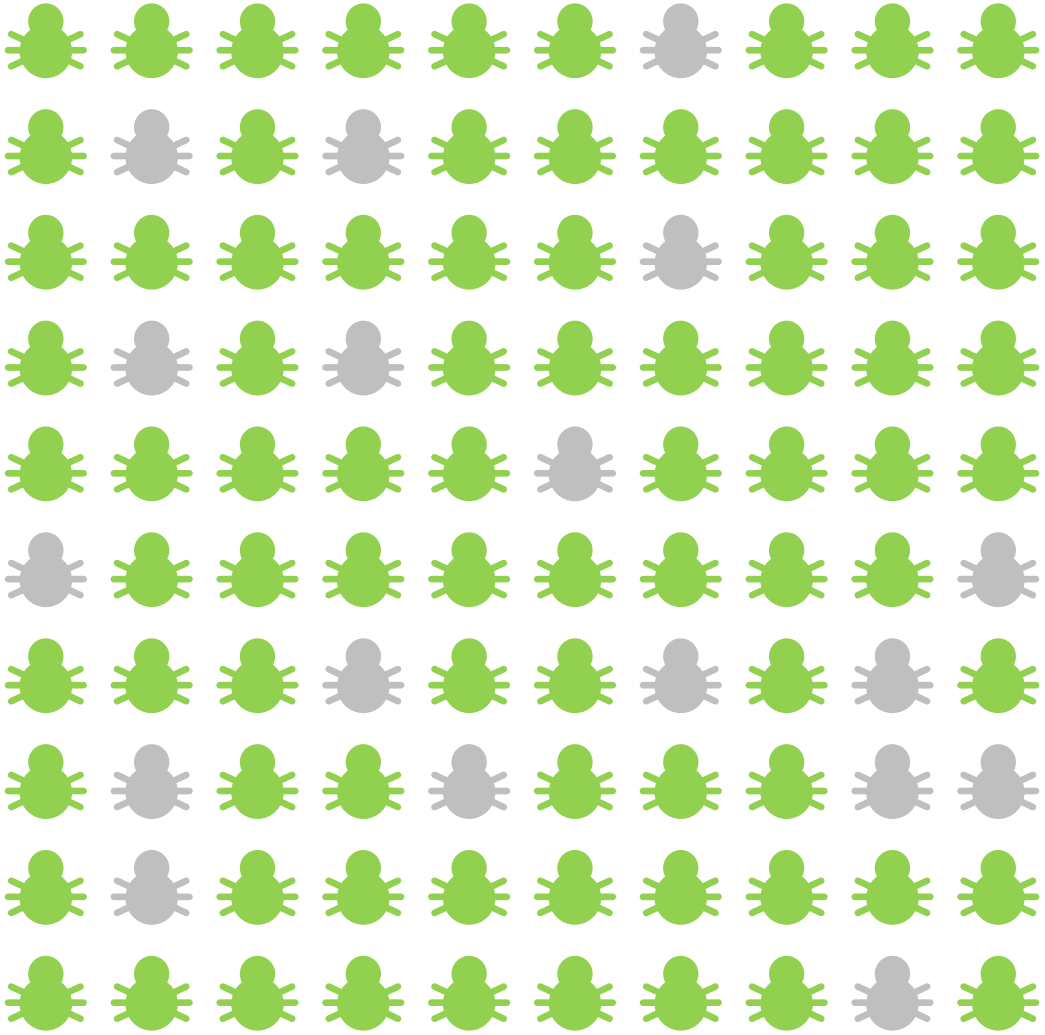
Schritt 2: Priorisierung selektierter Testfälle



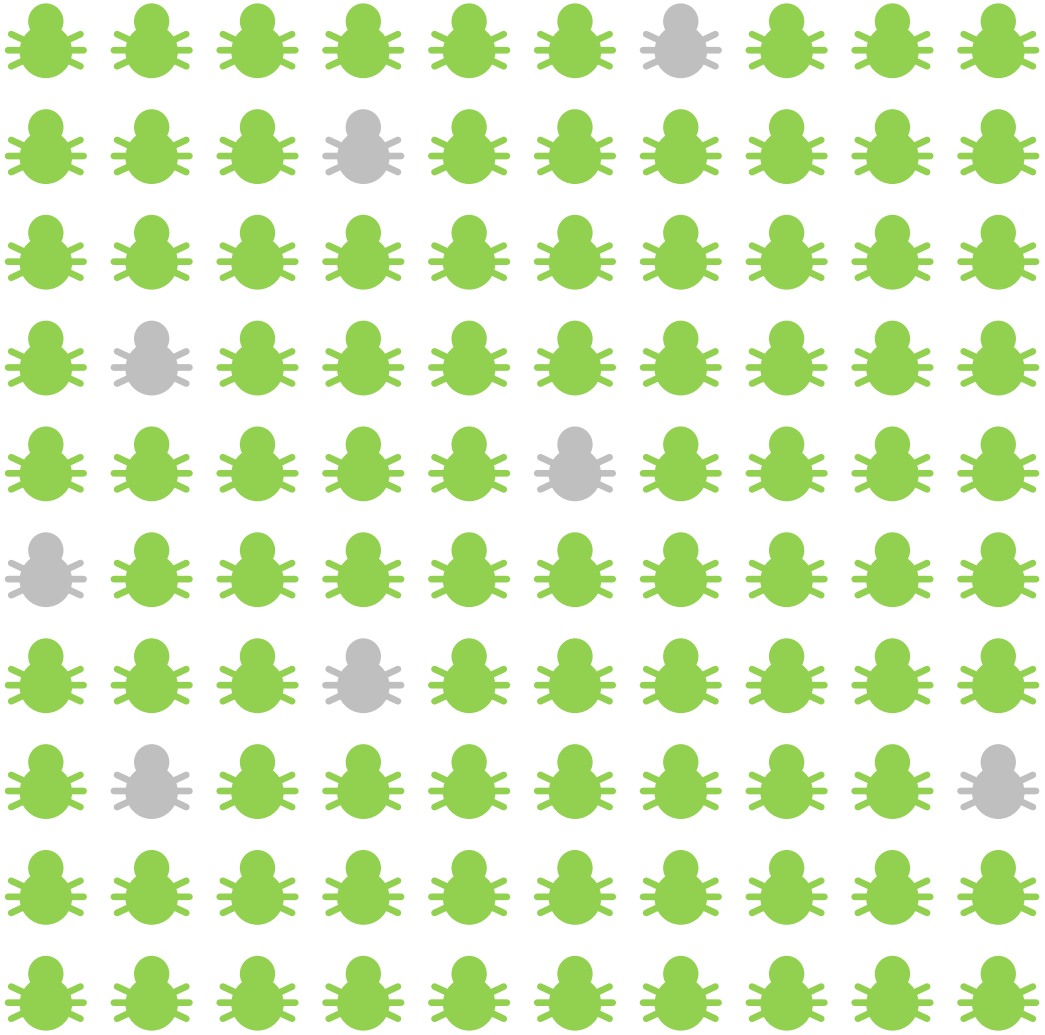




80 %



90 %



Test-Impact-Analyse bei Dolby Germany

Dolby

Vor TIA

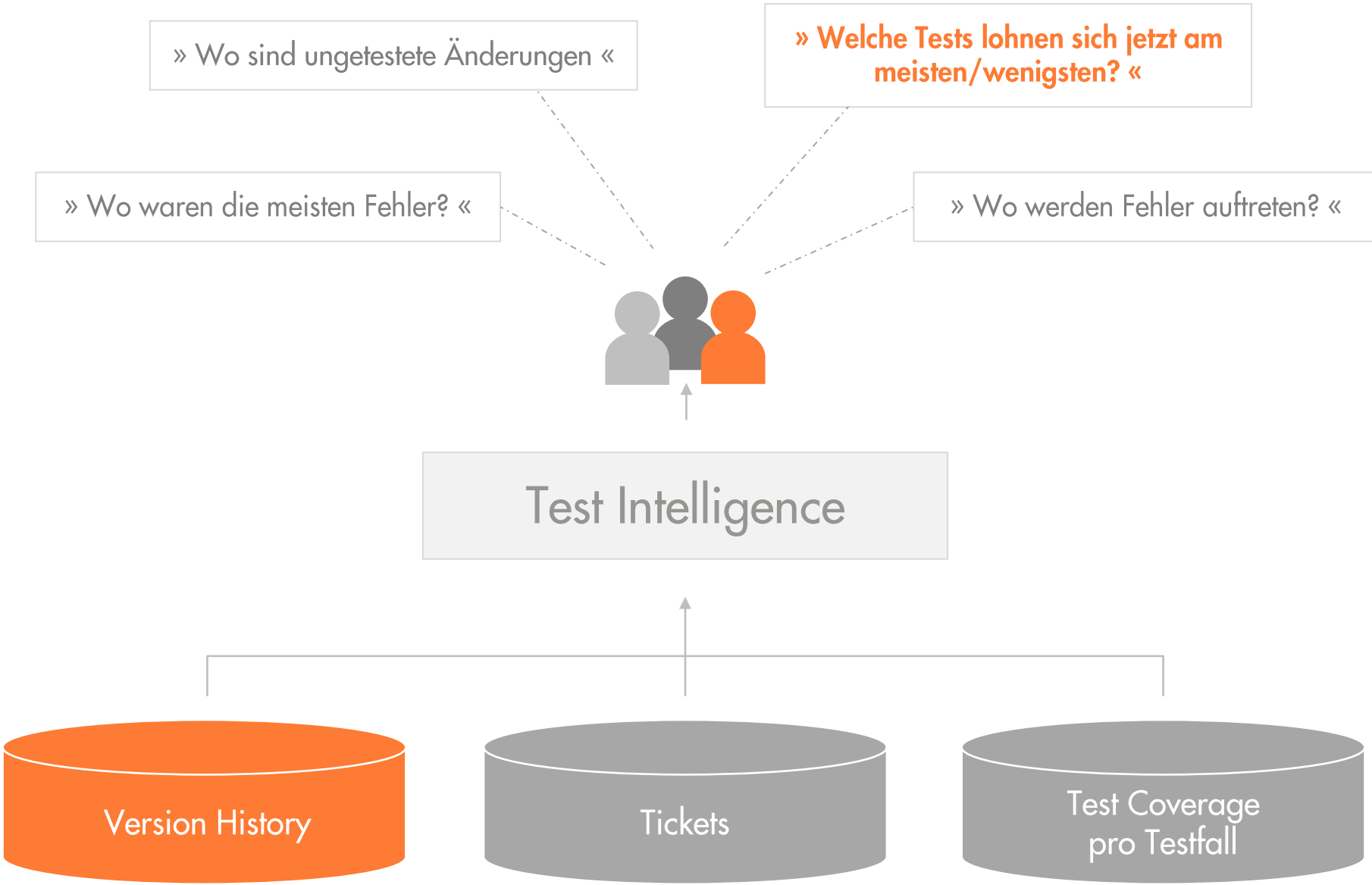
Testlaufzeiten von 3-12 Stunden durch eine Vielzahl von Parameterkombinationen.

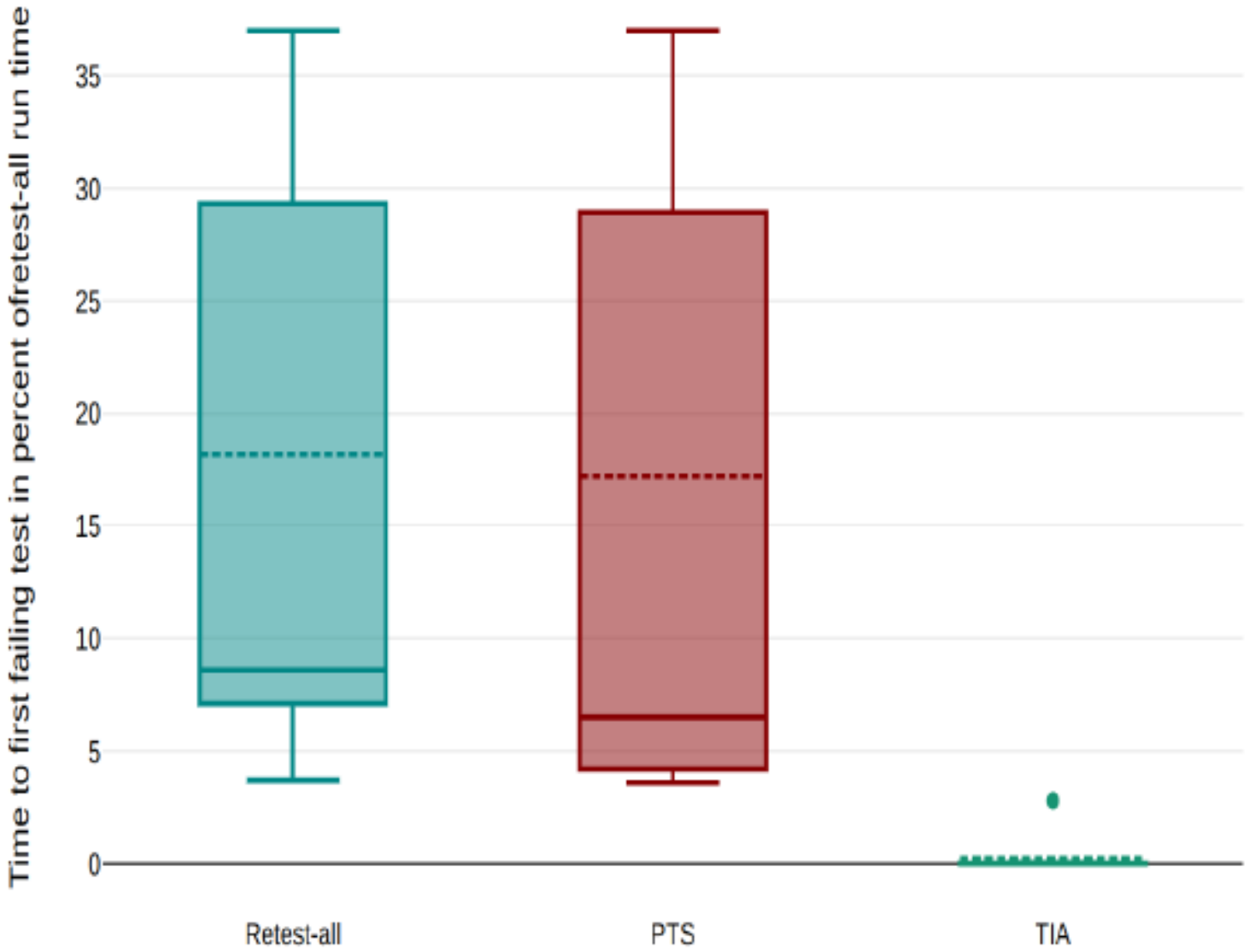
Ziel

Test-Feedback aus der CI innerhalb von 10 Minuten.

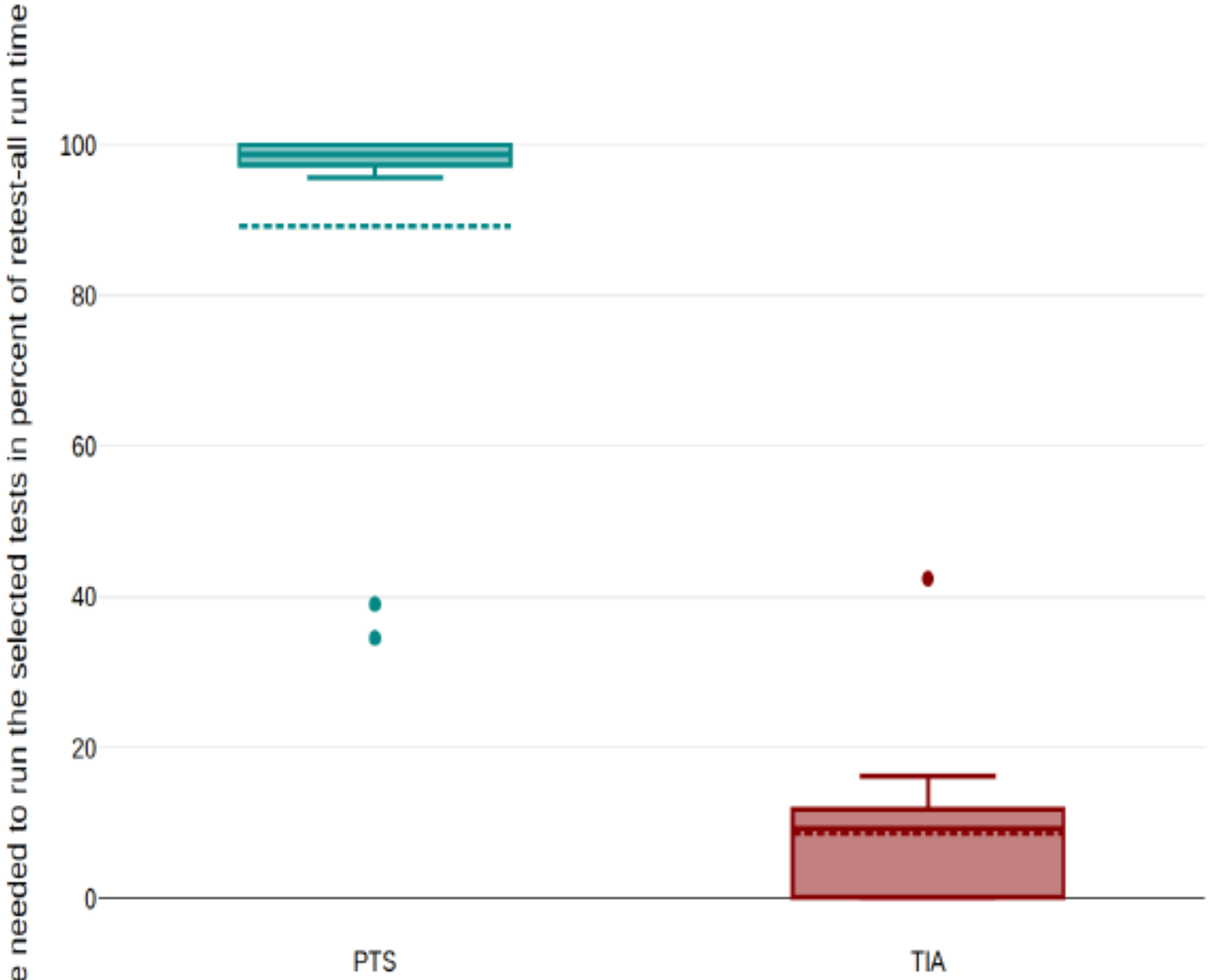
Mit TIA

Seit einem Jahre kommt das Test-Feedback immer innerhalb von 10 Minuten, obwohl die Gesamtzahl der Tests im selben Zeitraum um 57% zunahm.

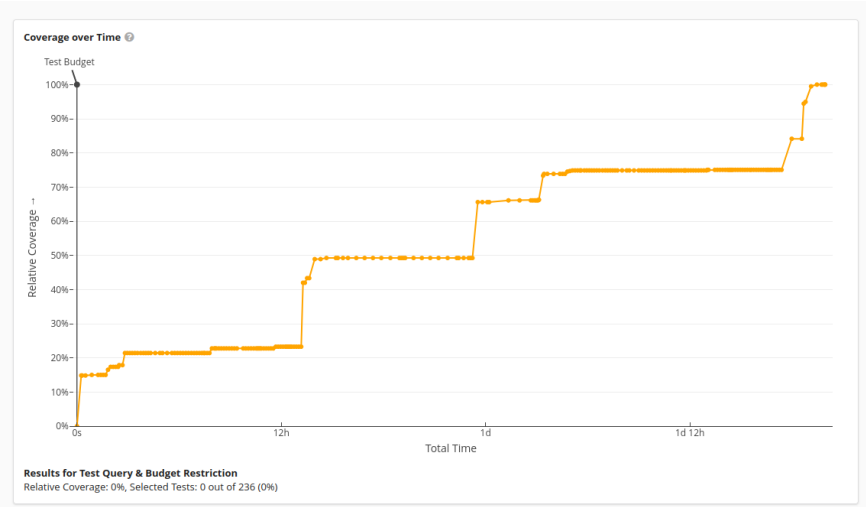

» Wo sind ungetestete Änderungen «
» Welche Tests lohnen sich jetzt am meisten/wenigsten? «
» Wo waren die meisten Fehler? «
» Wo werden Fehler auftreten? «
Test Intelligence
Version History
Tickets
Test Coverage pro Testfall

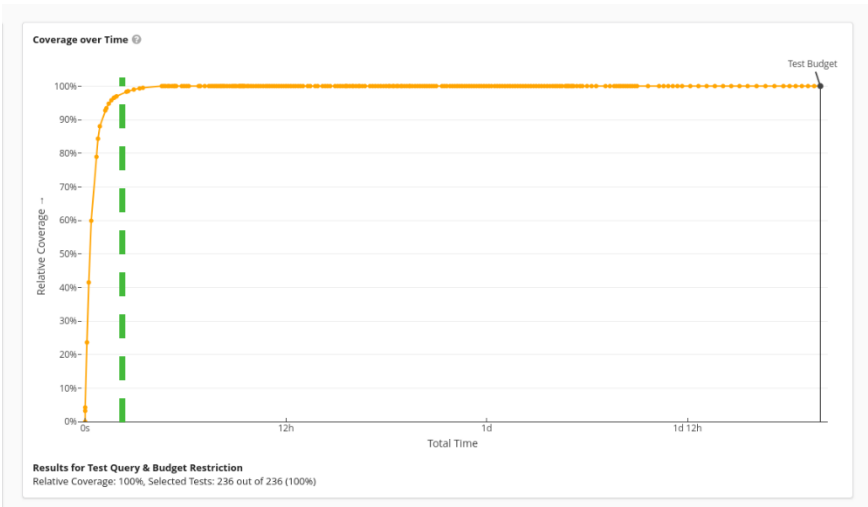


Ben Slimane, 2023, TU München: *Comparative Analysis of Different Approaches for Test Impact Analyses for Real World Test Suites*

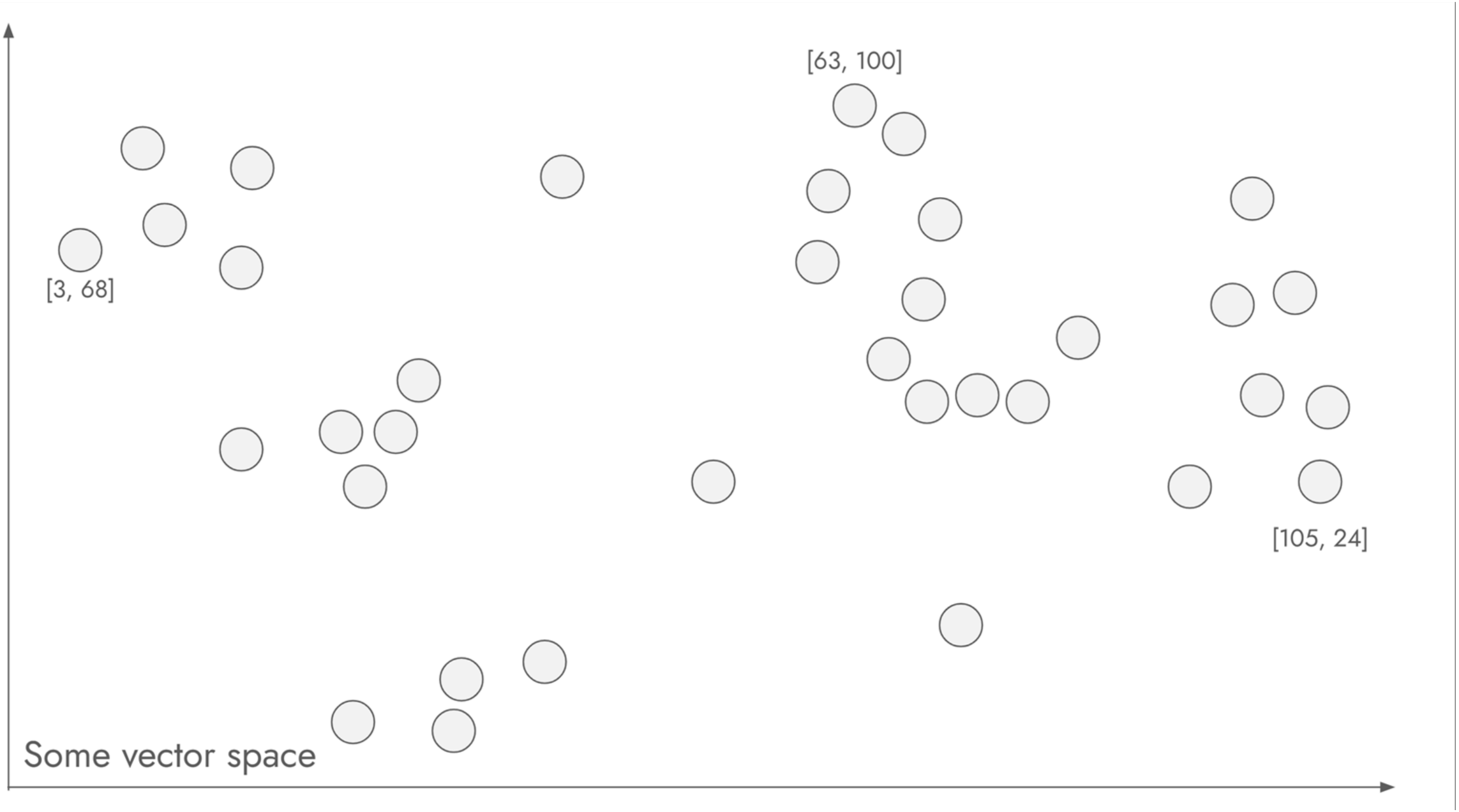


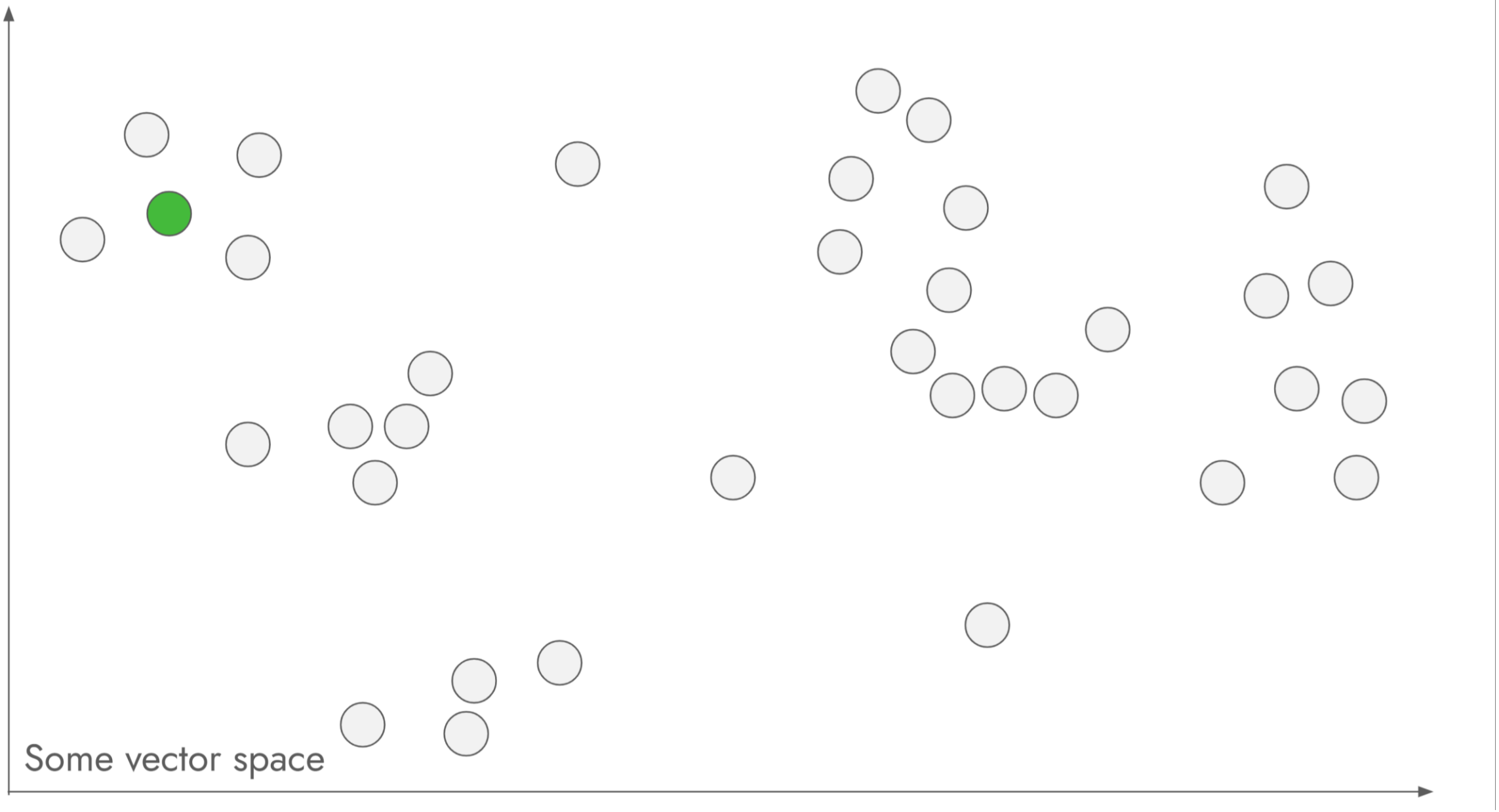
Ben Slimane, 2023, TU München: *Comparative Analysis of Different Approaches for Test Impact Analyses for Real World Test Suites*

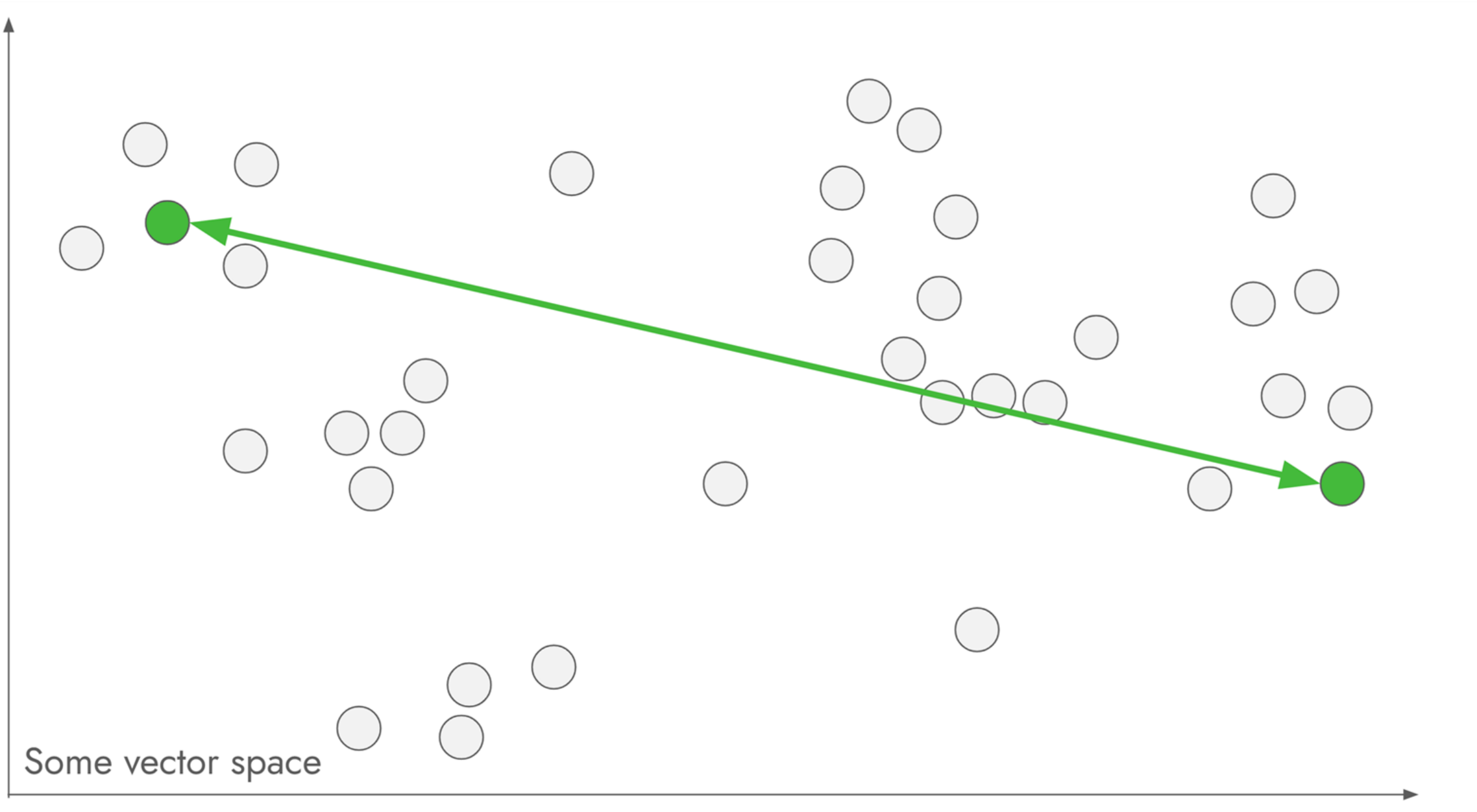


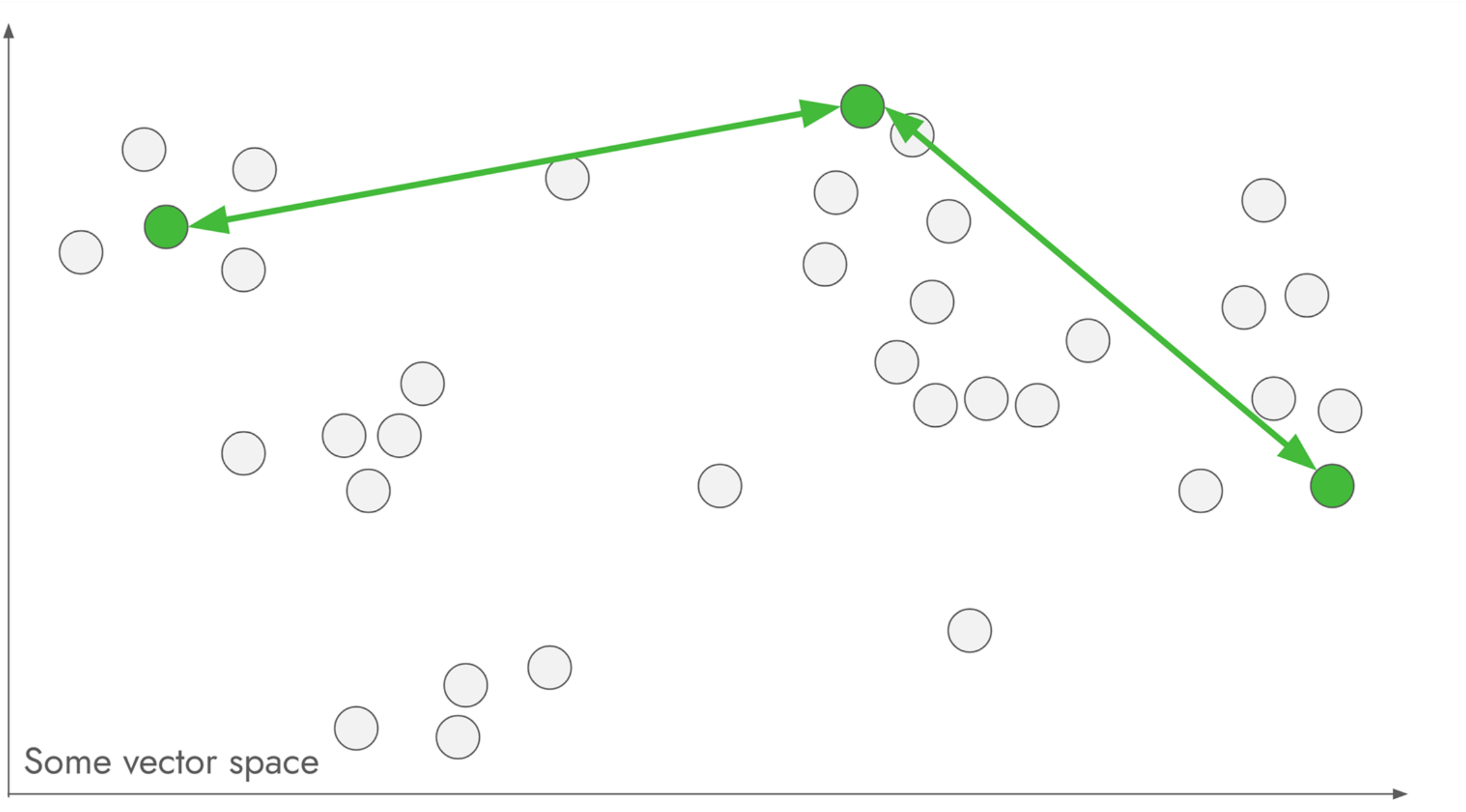


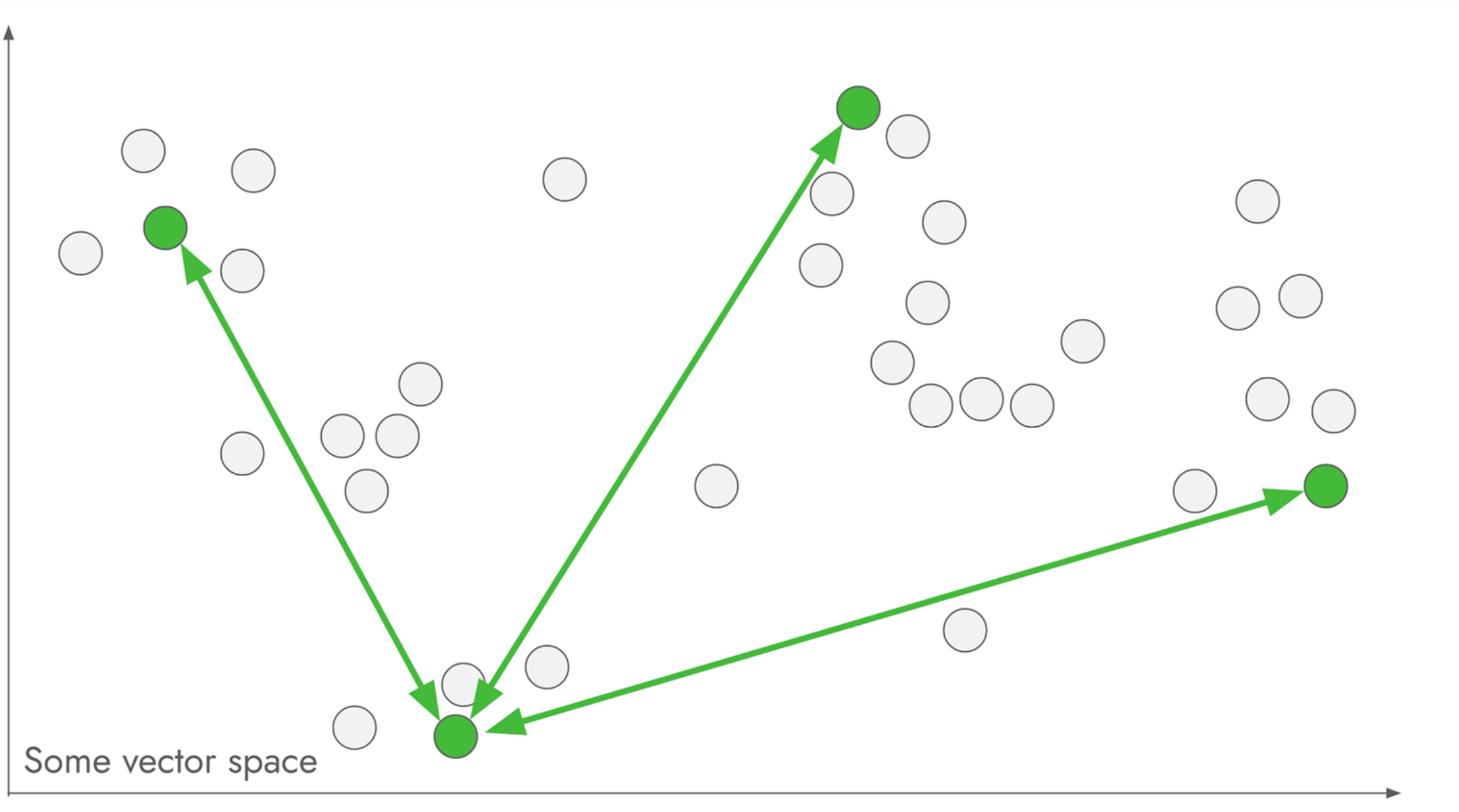
Sortieren von Tests nach “Unähnlichkeit”


[63, 100]
[3, 68]
[105, 24]
Some vector space


Some vector space


Some vector space


Some vector space


Some vector space

An Evaluation of Distance Based Test Suite Reduction Techniques

Alessandro Escher
Technical University of Munich
Munich, Germany
alessandro.escher@tum.de

Raphael Nömmer
Technical University of Munich
Munich, Germany
noemmer@cqse.eu

***Abstract*—Efficient test suite selection is crucial in software testing due to the high cost of running extensive tests, particularly on large industry projects. Coverage-based techniques aim to maximize system execution within time constraints but often suffer from costly and complex coverage recording processes. This study explores alternative selection methods using test metadata and source code. Hierarchical Agglomerative Clustering (HAC) and a greedy approach were evaluated alongside distance measures based on package path distance and vector representations of test code.**

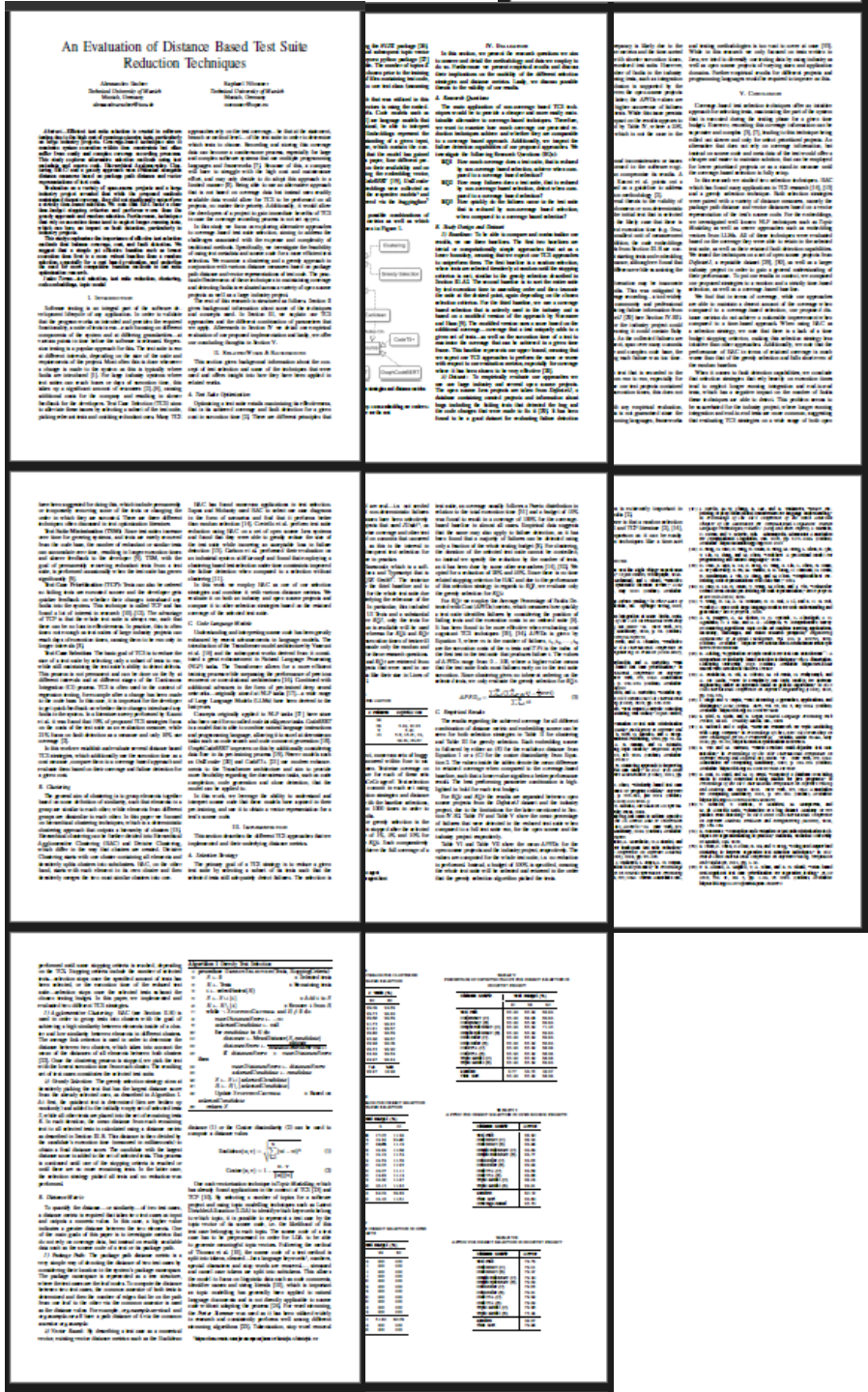
Evaluation on a variety of open-source projects and a large industry project revealed that while the proposed methods maintained decent coverage, they did not significantly outperform a strictly time-based selection. We note that HAC lacks a clear time-budget stopping criterion and performs worse than the greedy approach and random selection. Furthermore, techniques that rely on execution times tend to neglect longer-running tests, which can have an impact on fault detection, particularly in industry projects.

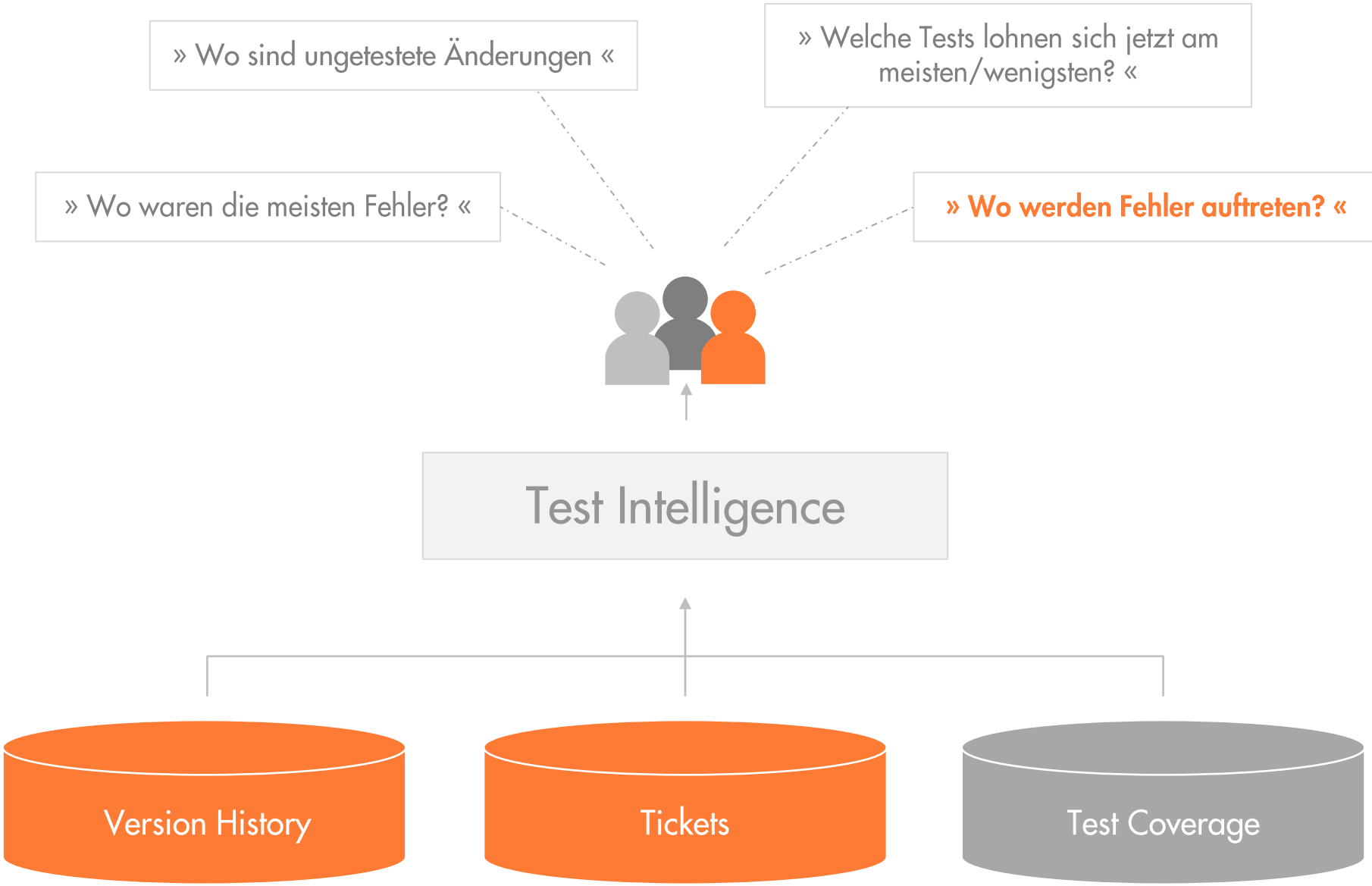
This study emphasizes the importance of effective test selection methods that balance coverage, cost, and fault detection. We suggest that a simple yet effective baseline such as lowest execution time first is a more robust baseline than a random selection, especially for a cost based evaluation, and underline the need for more competitive baseline methods in test suite optimization research.

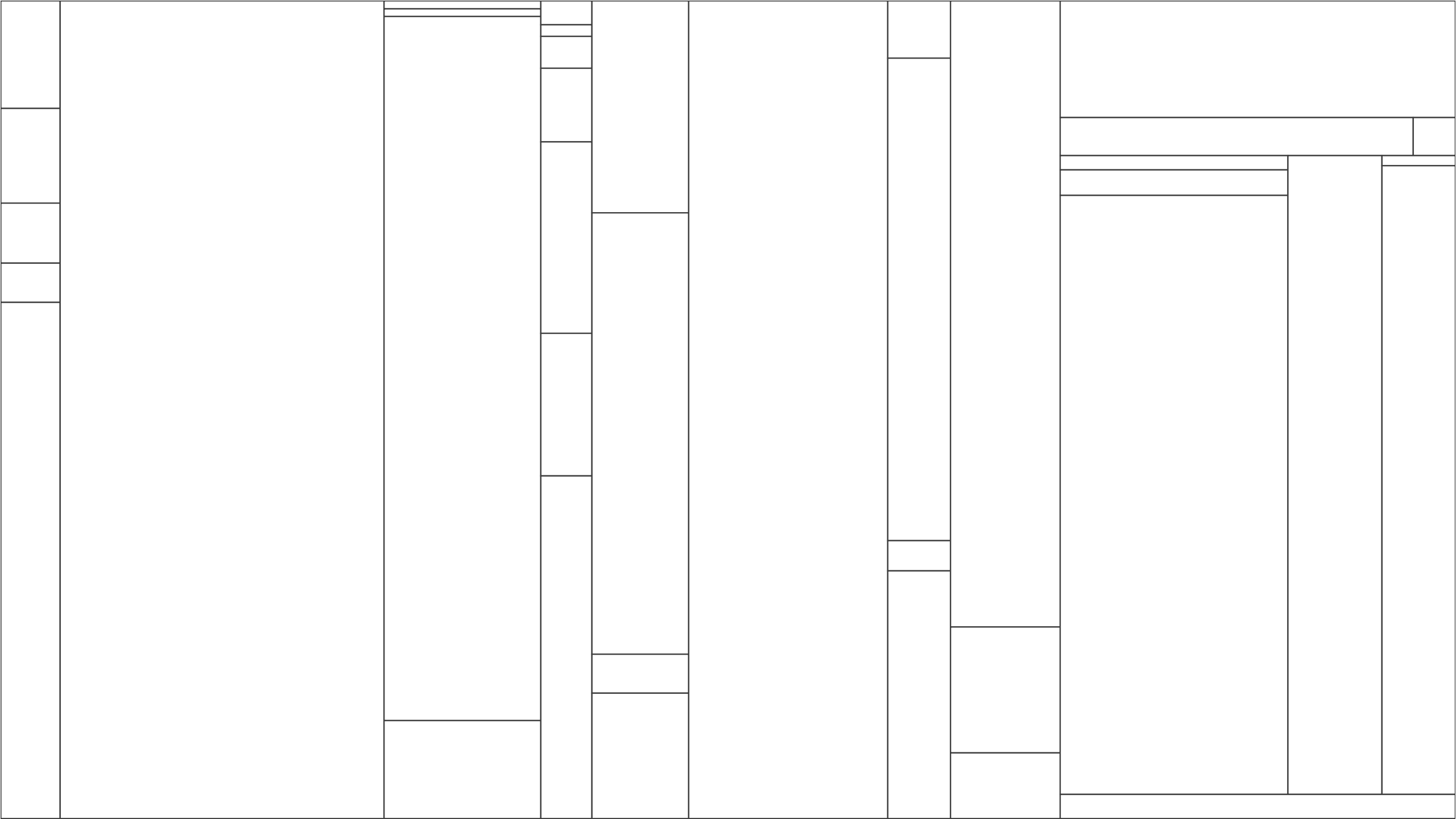
***Index Terms*—test selection, test suite reduction, clustering, code embeddings, topic model**

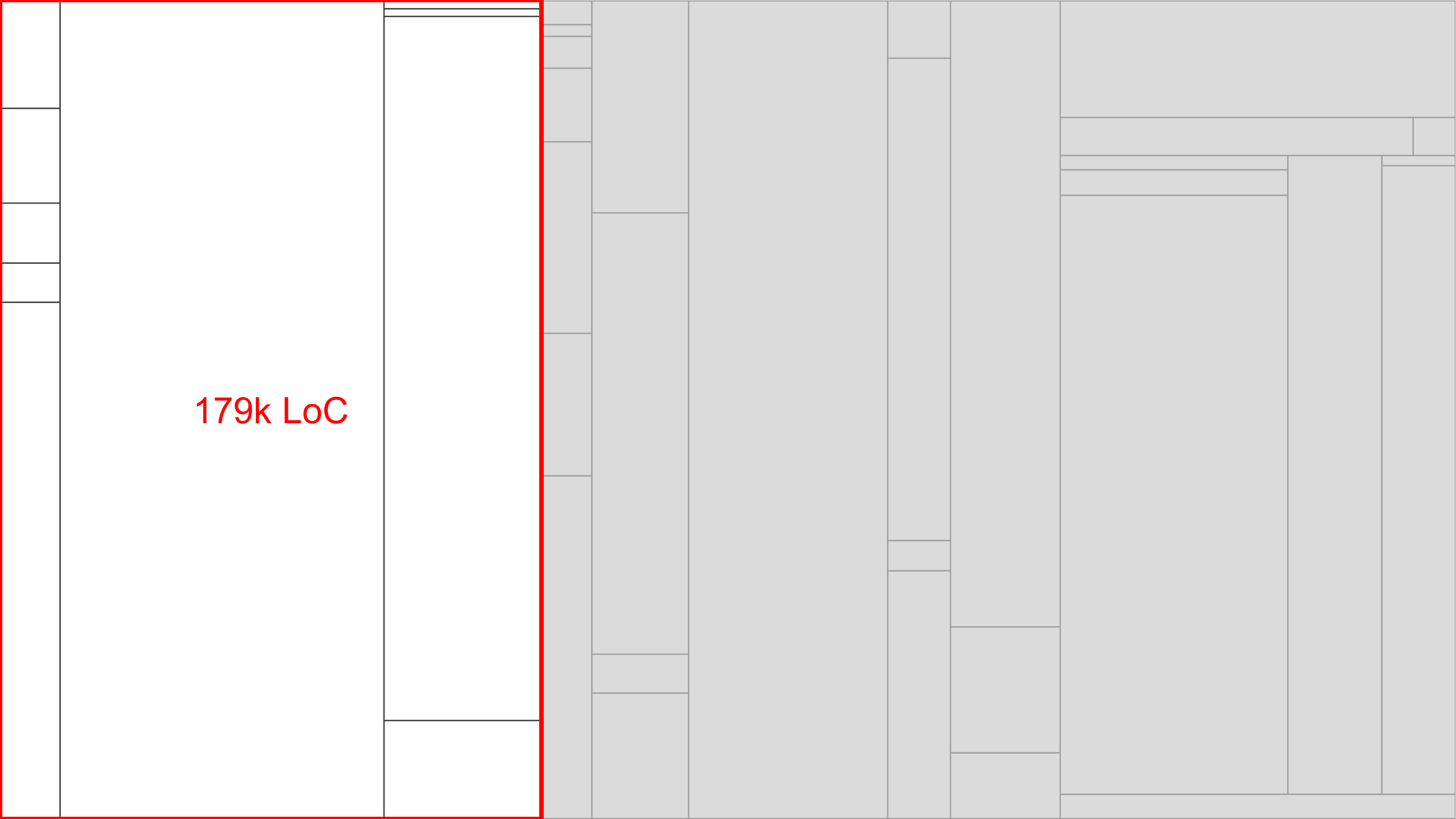
approaches rely on the test coverage—be that at the statement, branch or method level—of the test suite in order to determine which tests to choose. Recording and storing this coverage data can become a cumbersome process, especially for large and complex software systems that use multiple programming languages and frameworks [7]. Because of this, a company will have to struggle with the high cost and maintenance effort, and may only decide to do adopt this approach in a limited manner [8]. Being able to use an alternative approach that is not based on coverage data but instead uses readily available data would allow for TCS to be performed on all projects, no matter their priority. Additionally, it would allow the developers of a project to gain immediate benefits of TCS in case the coverage recording process is not set up yet.

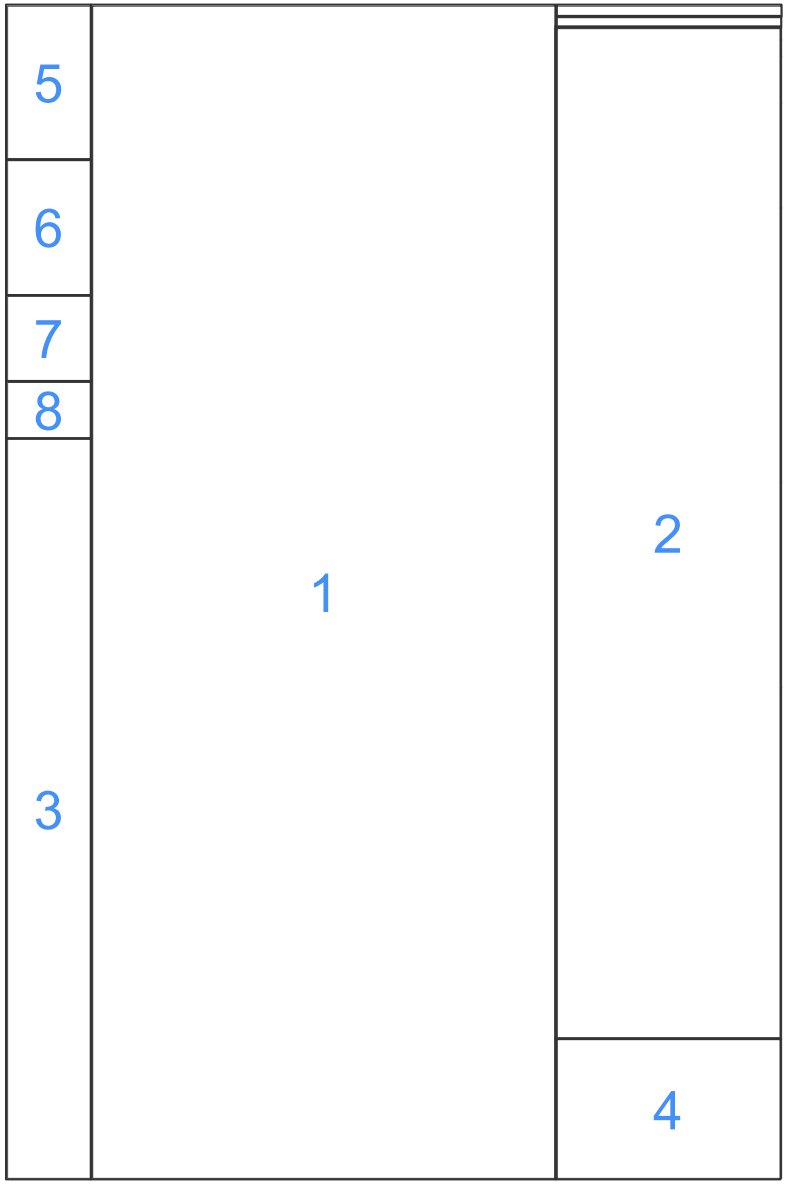
In this study we focus on exploring alternative approaches to coverage-based test suite selection, aiming to address the challenges associated with the expense and complexity of traditional methods. Specifically, we investigate the feasibility of using test metadata and source code for a more efficient test selection. We examine a clustering and a greedy approach in conjunction with various distance measures based on package path distance and vector representations of test code. The practical effectiveness of these techniques in maintaining coverage and detecting faults is evaluated across a variety of open source

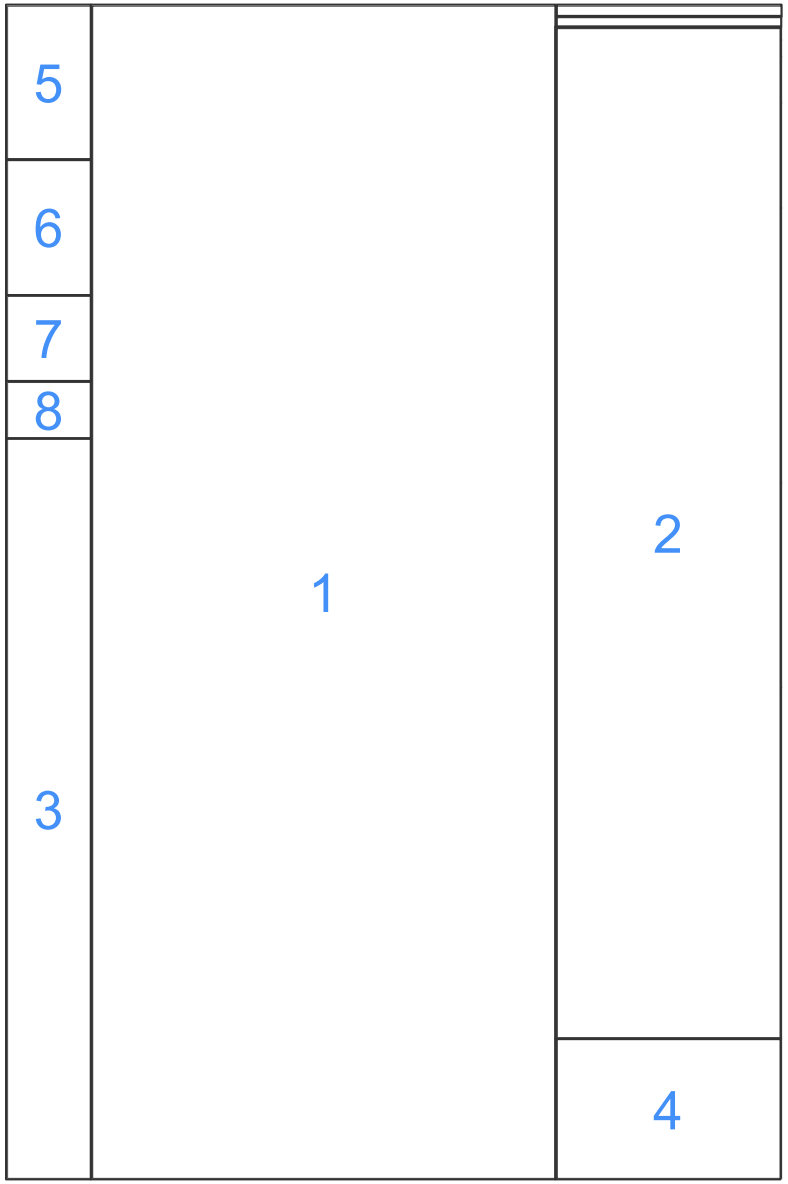


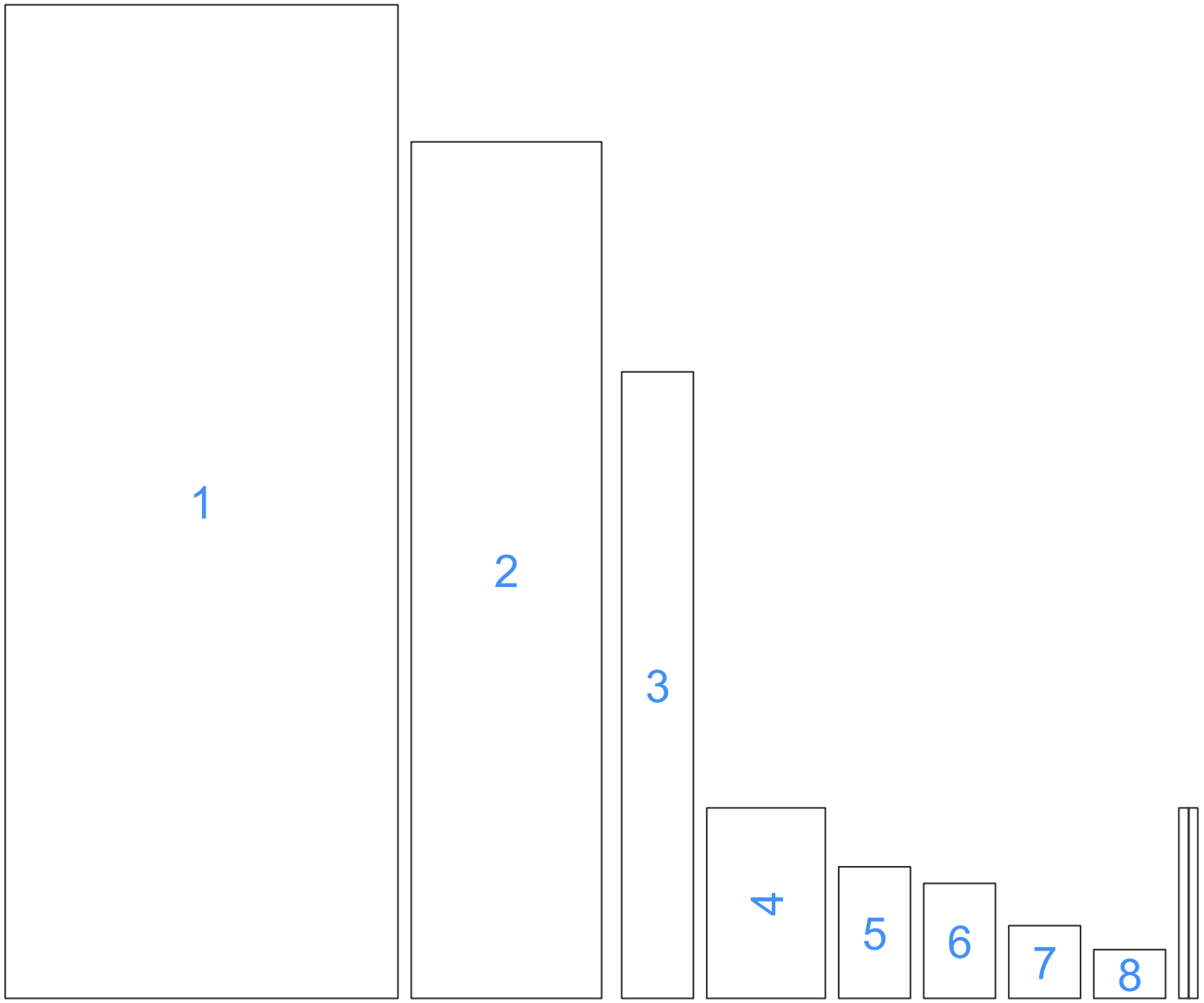

» Wo sind ungetestete Änderungen «
» Welche Tests lohnen sich jetzt am meisten/wenigsten? «
» Wo waren die meisten Fehler? «
» Wo werden Fehler auftreten? «
Test Intelligence
Version History
Tickets
Test Coverage

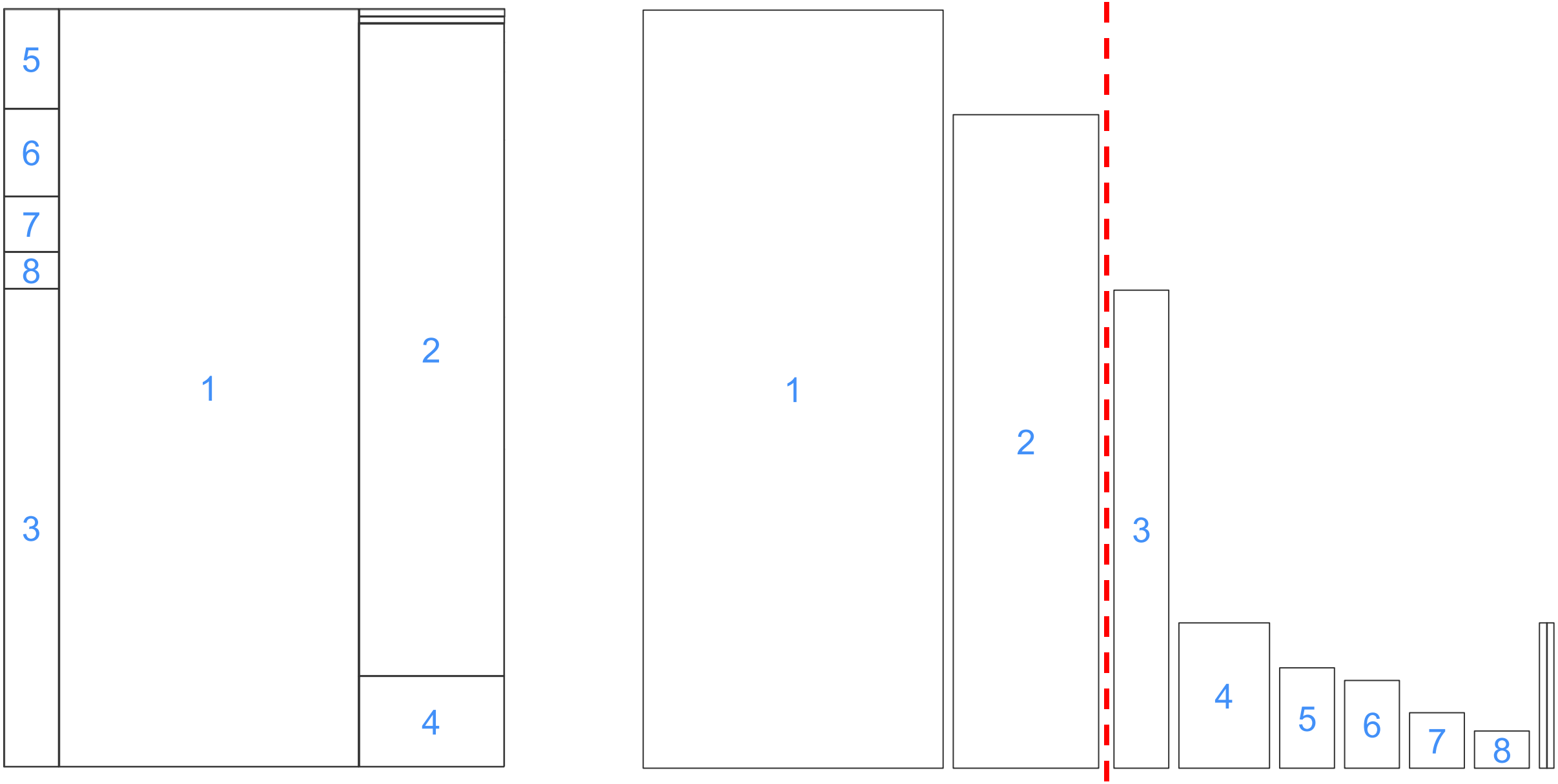


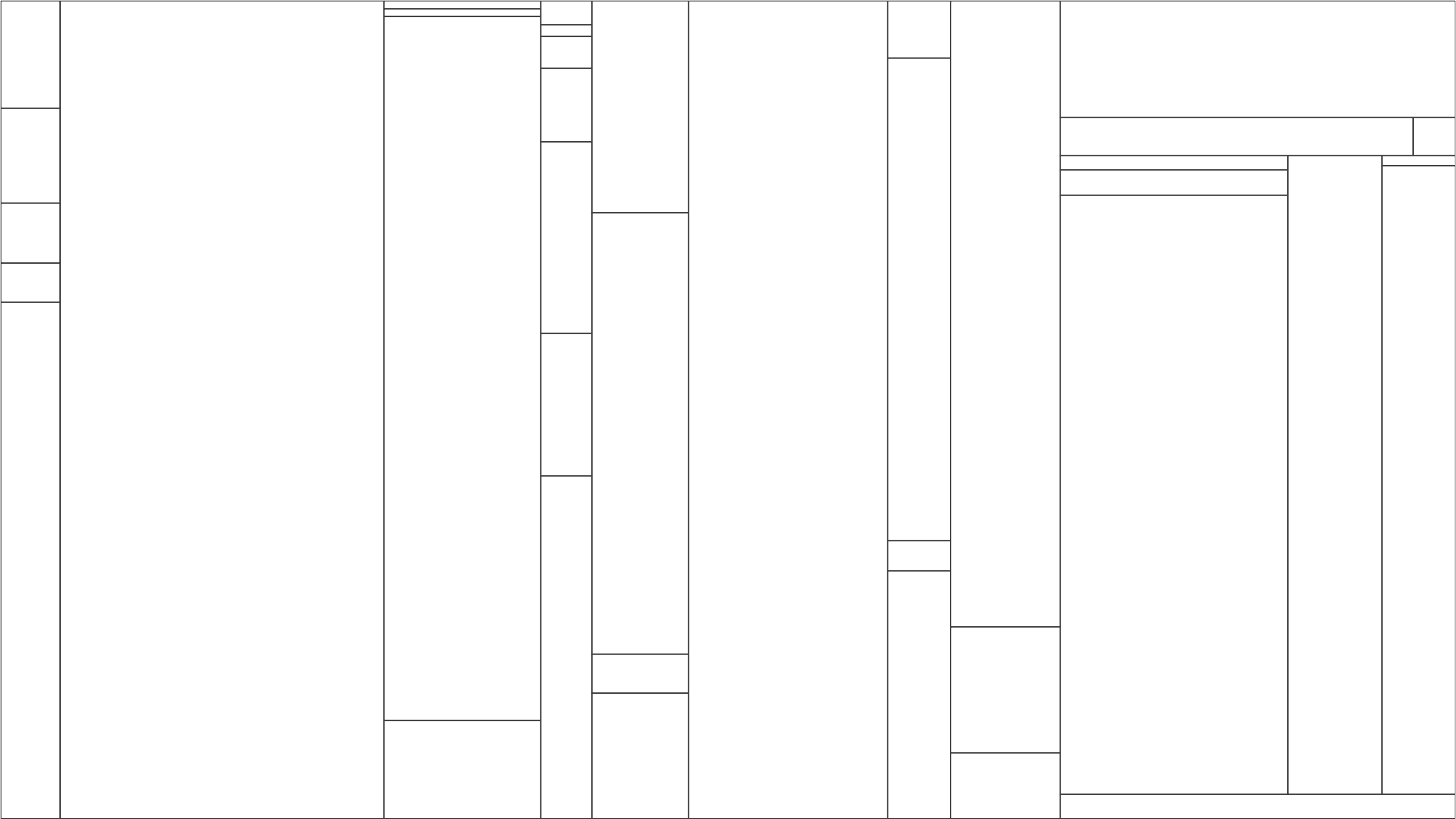

179k LoC

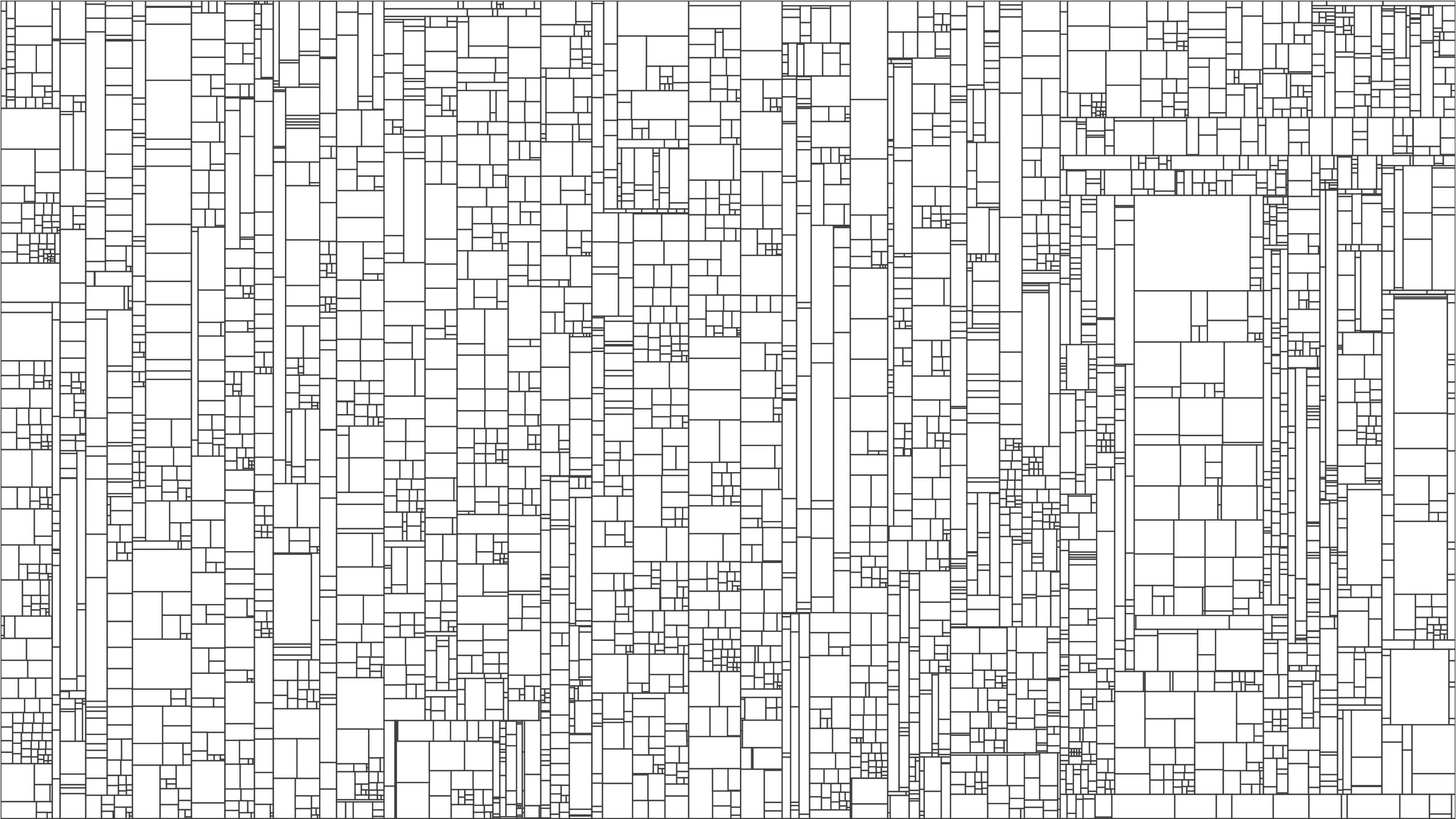

5
6
7
8
1
2
3
4

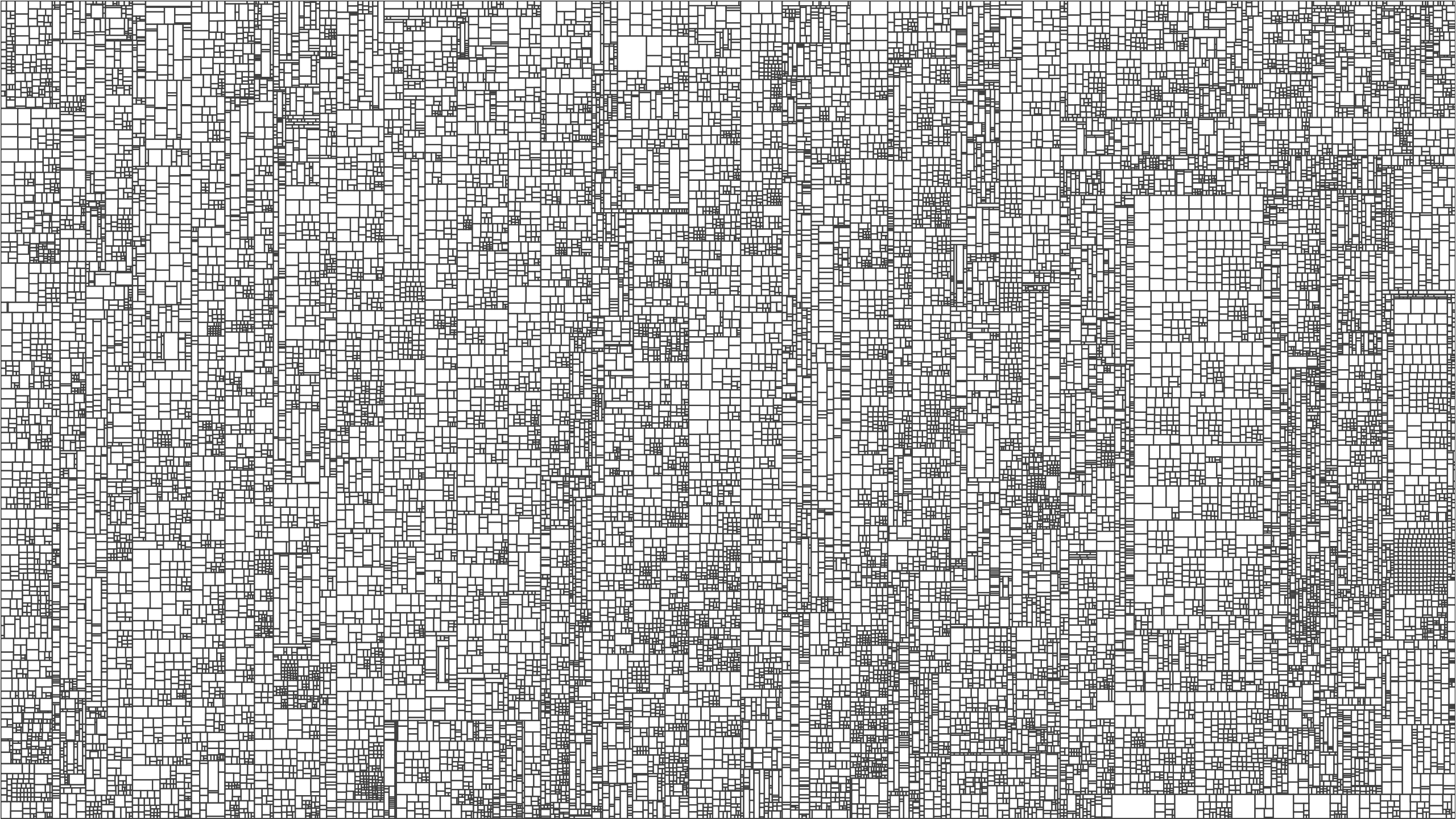

5
6
7
8
3
1
2
4

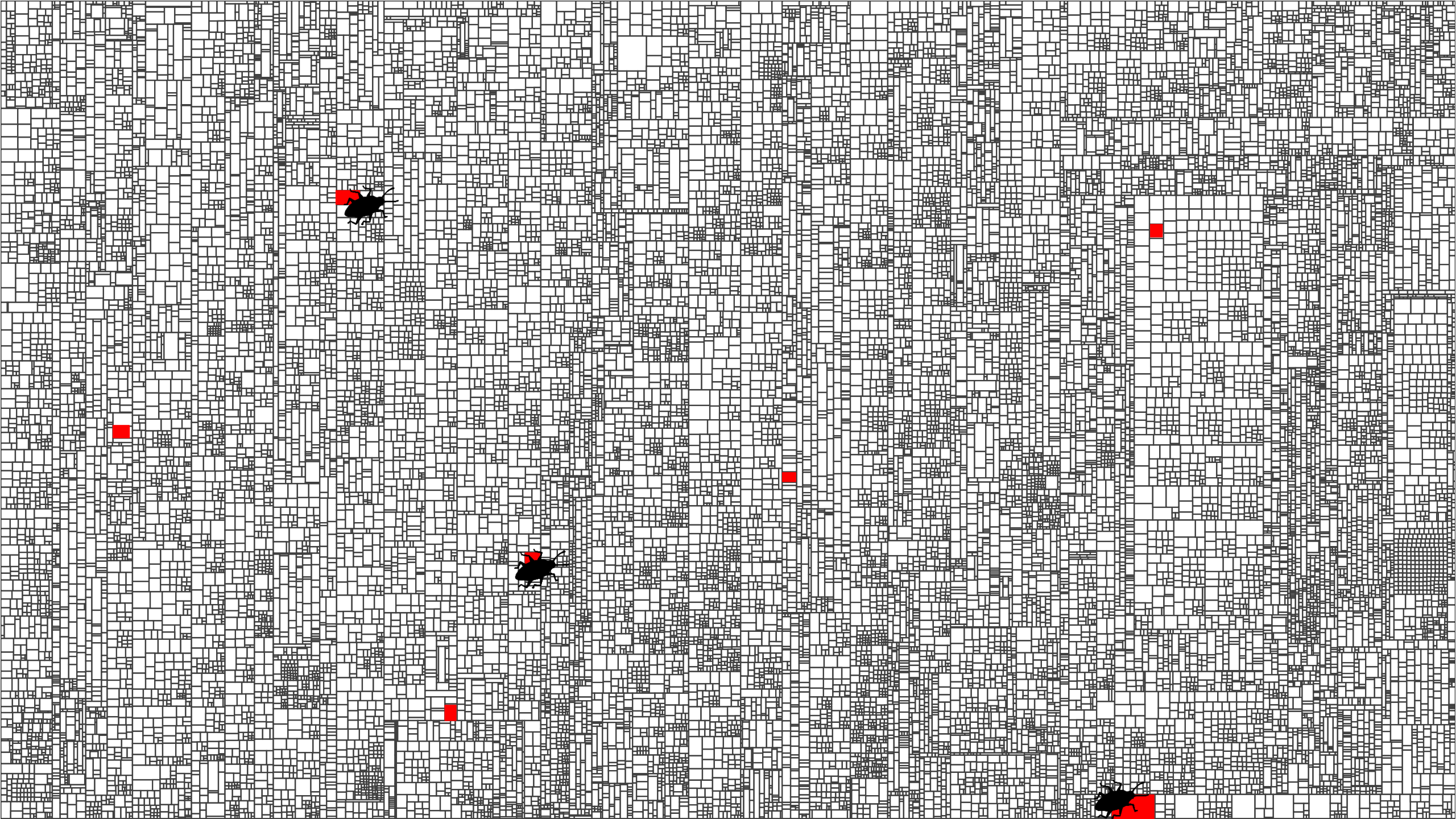

1
2
3
4
5
6
7
8

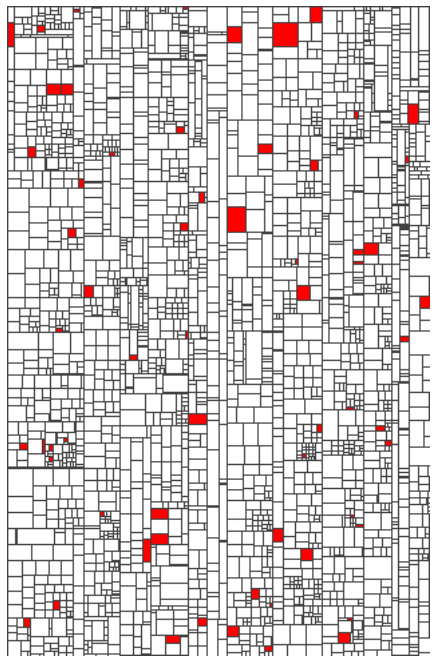

153k LoC
85%
26,5k LoC
15%
5
6
7
8
1
2
3
4
1
2
3
4
5
6
7
8

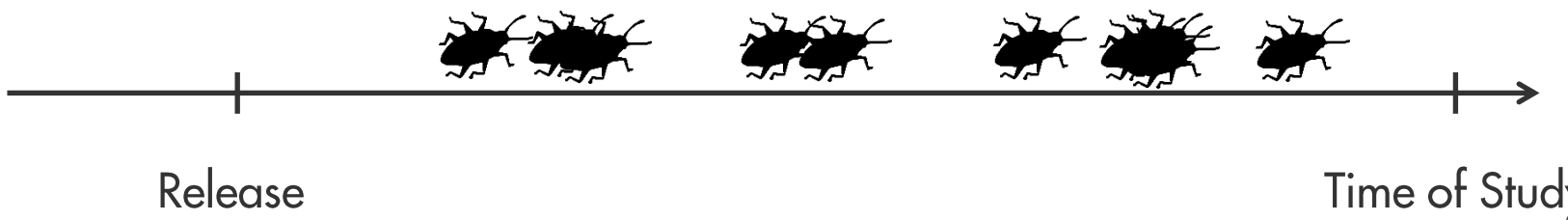


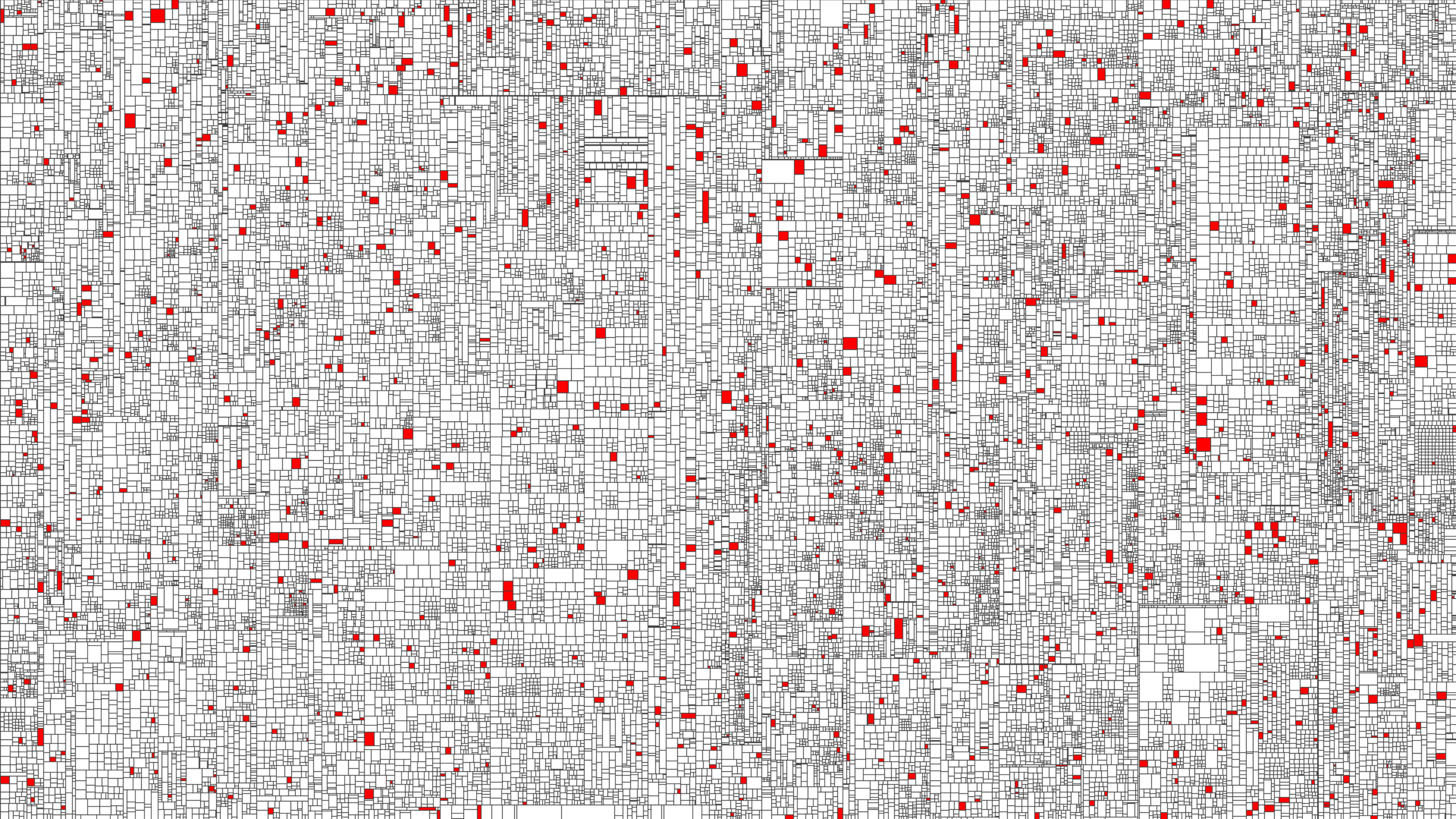


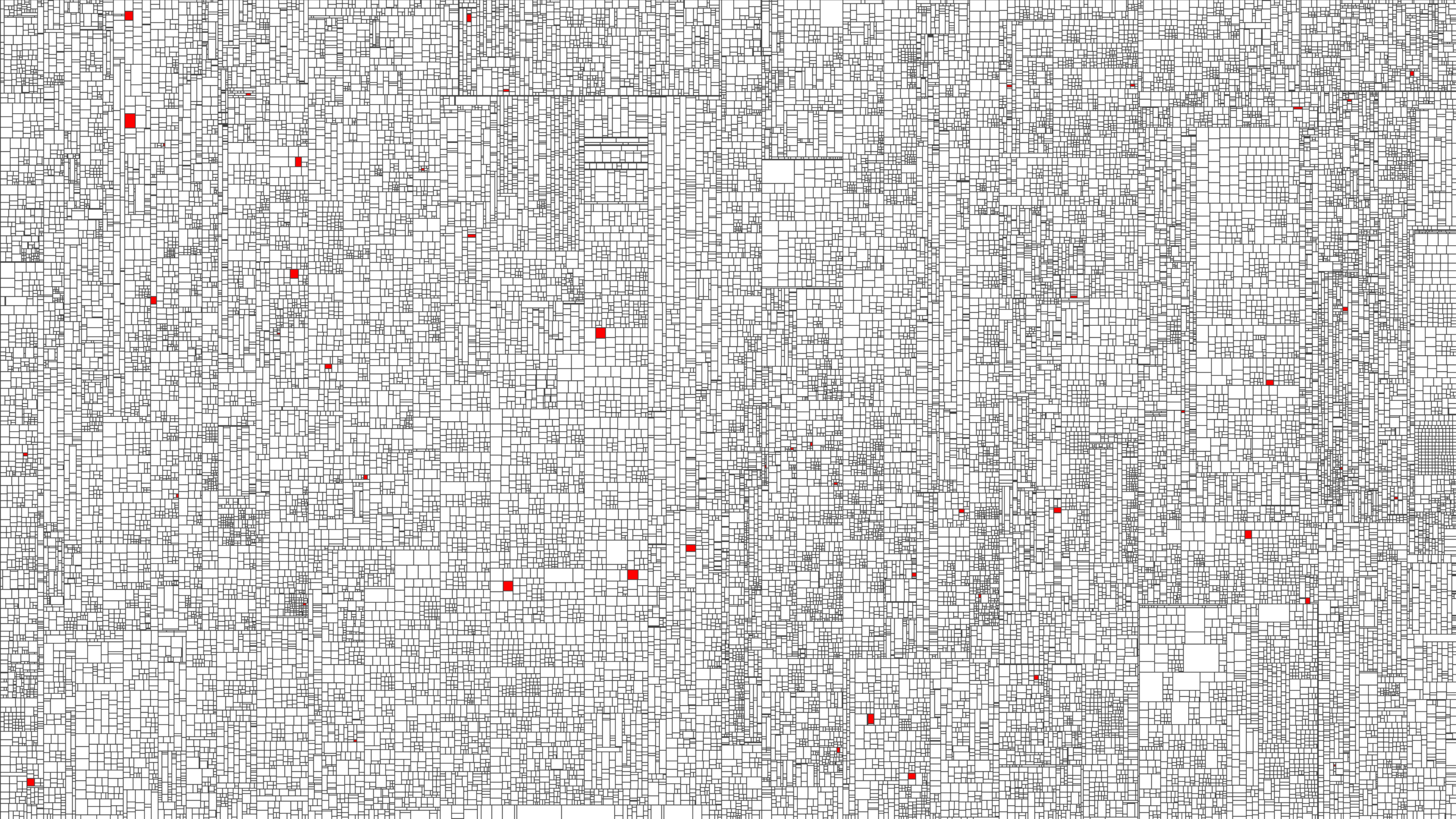







Release
Time of Study





Evaluation

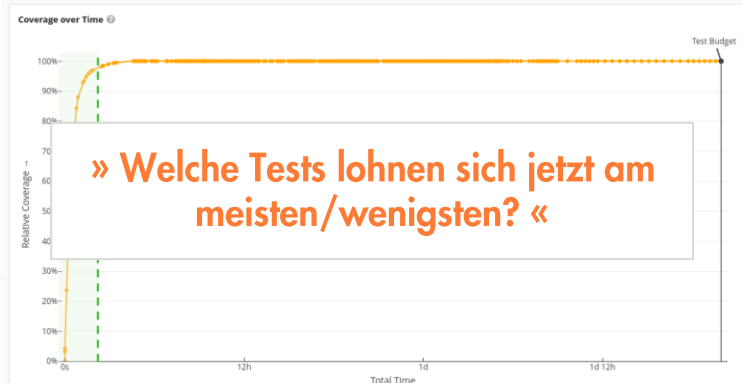
Release	# „defect prone“ Methods	# Bugs (Top 50)
1.4:	1127	0
2.0:	1176	0

Pascarella, Palomba, Bacchelli, *Re-evaluating Method-Level Bug Prediction*, 2018:
Prediction nicht besser als zufällige Klassifikation.

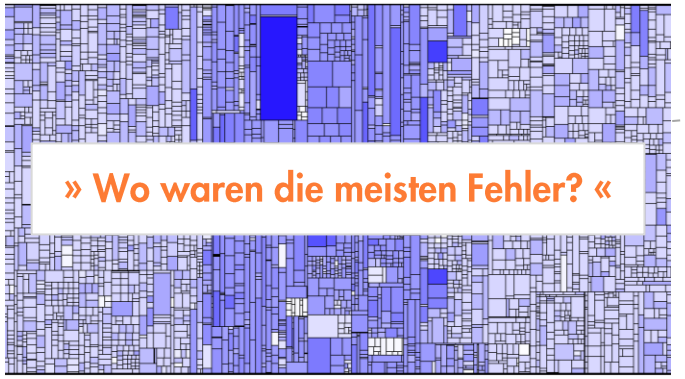
Chowdhury, Uddin, Hemmati, Holmes, *Method-Level- Bug Prediction: Problems and Promise*, 2024:

Method-Level Bug Prediction performance „extremely poor“.

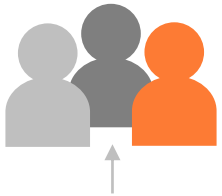
» Wo sind ungetestete Änderungen «



» Welche Tests lohnen sich jetzt am meisten/wenigsten? «



» Wo waren die meisten Fehler? «



» Wo werden Fehler auftreten? «

Test Intelligence

Version History

Tickets

Test Coverage

CQSE Workshop

Test-Gap-Analyse

Ungetestete Änderungen im Quelltext aufdecken

Aufzeichnungen unter
tmscl.me/tga-et25

Schnelles Feedback trotz langsamer Tests

Testselektion für historisch gewachsene Test-Suites

Aufzeichnungen unter
tmscl.me/ts-et25

Fazit

Historisch gewachsene Test-Suites testen oft gleichzeitig zuviel (wegen redundanten Testfällen) und zuwenig (wegen Test-Gaps).

Die Auswertung der Daten aus unserem eigenen Entwicklungsprozess (Versionshistorie, Test-Coverage und Tickets) ermöglicht uns, mehr Fehler in kürzerer Zeit zu finden.

Ich teile unsere Erfahrung hierzu gerne.

Kontakt – Ich freue mich auf Diskussionen ☺

Dr. Elmar Jürgens · juergens@cqse.eu · +49 179 675 3863

CQSE GmbH
Centa-Hafenbrädl-Str 59
81249 München
www.cqse.eu