

Immer kürzere Testphasen?

Mit Ticket Coverage verhindern, dass wichtige Änderungen ungetestet bleiben

Dr. Elmar Jürgens
CQSE GmbH



Über Mich

Forschung

- Clone Detection, Test-Gap-Analyse, ...
- PC Mitglied von MSR, ICPC, ICSE, ...

Beratung

- Gründer
- Qualitäts-Bewertung & Qualitäts-Controlling

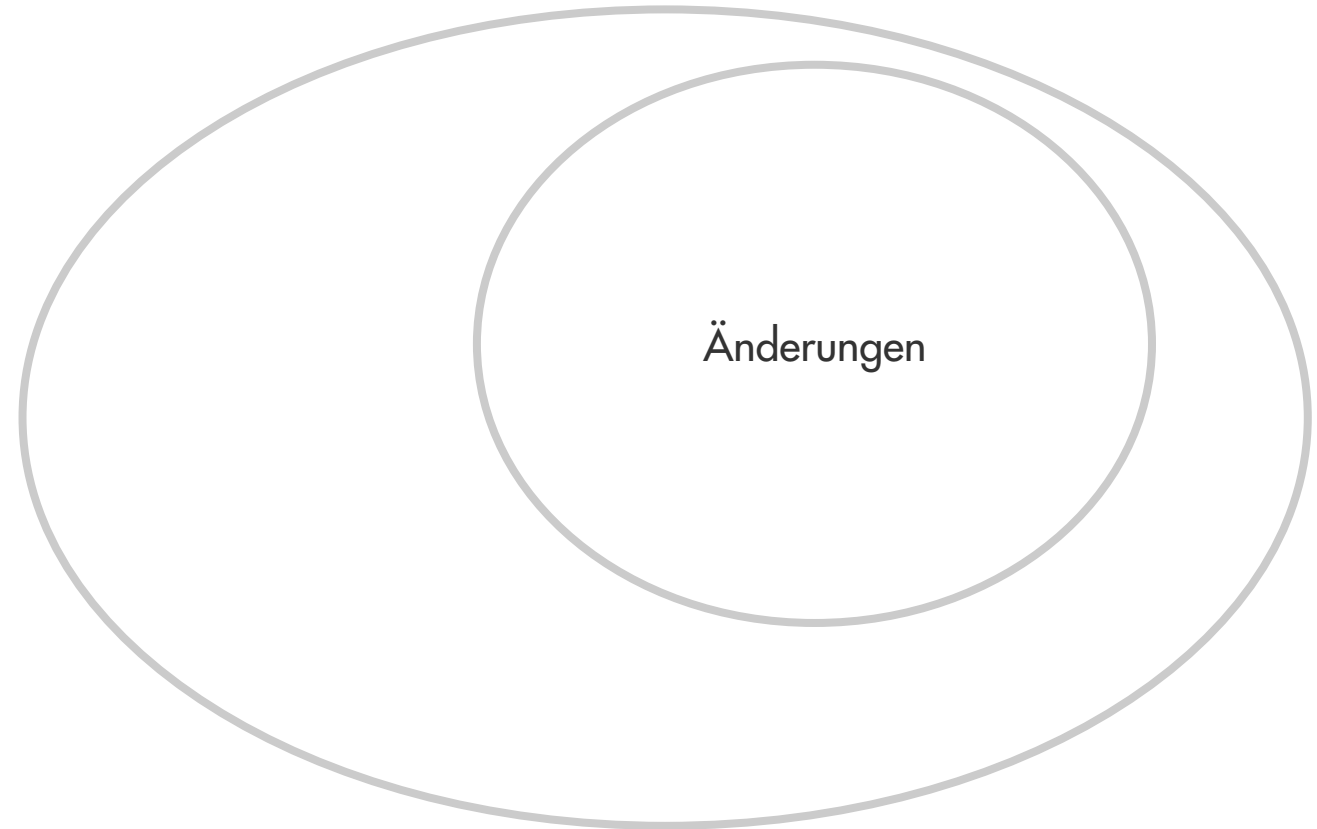
Gesellschaft für Informatik

- Zum Junior-Fellow ernannt
- Erfahrungsaustausch Forschung <-> Praxis



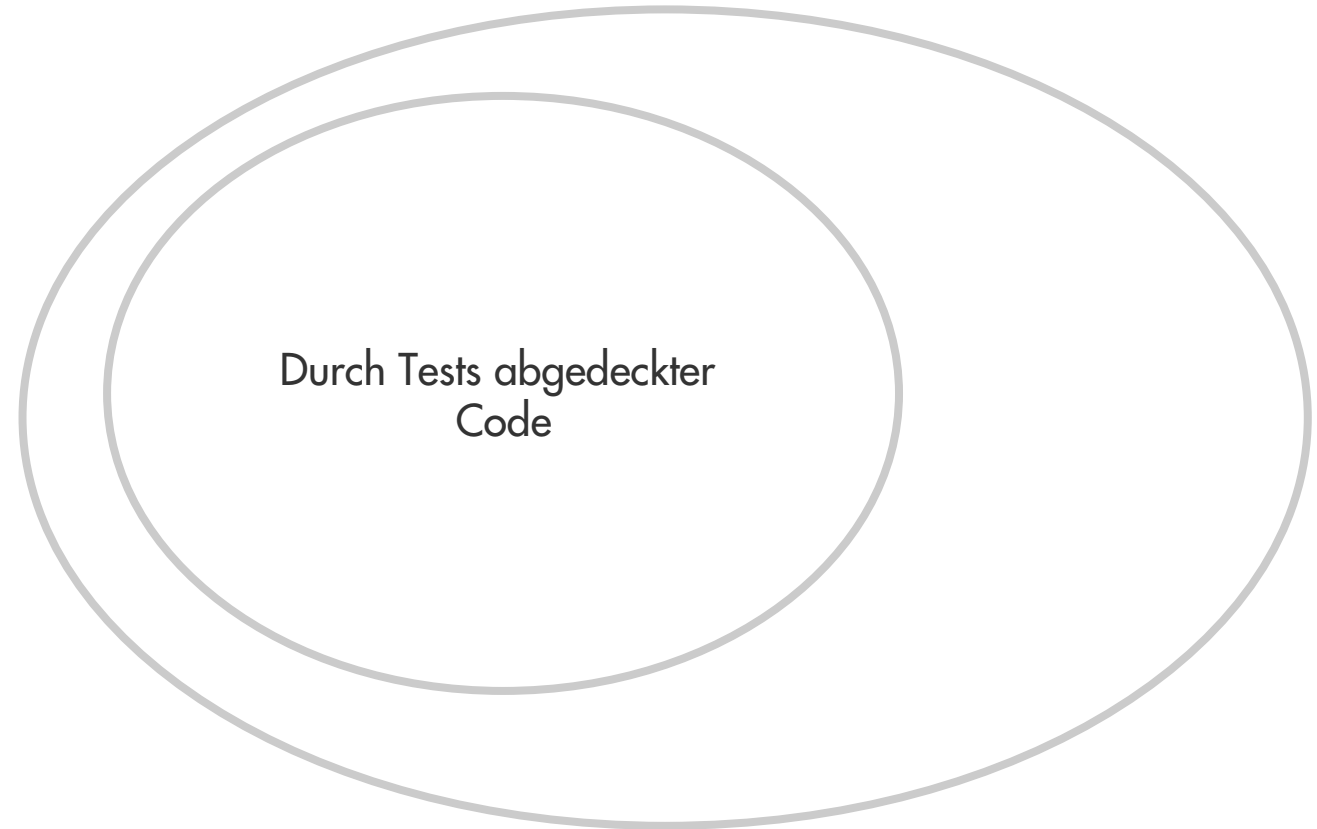
Wieviele Änderungen sind ungetestet?

Studie: C# System @ Munich Re



Wieviele Änderungen sind ungetestet?

Studie: C# System @ Munich Re



Wieviele Änderungen sind ungetestet?

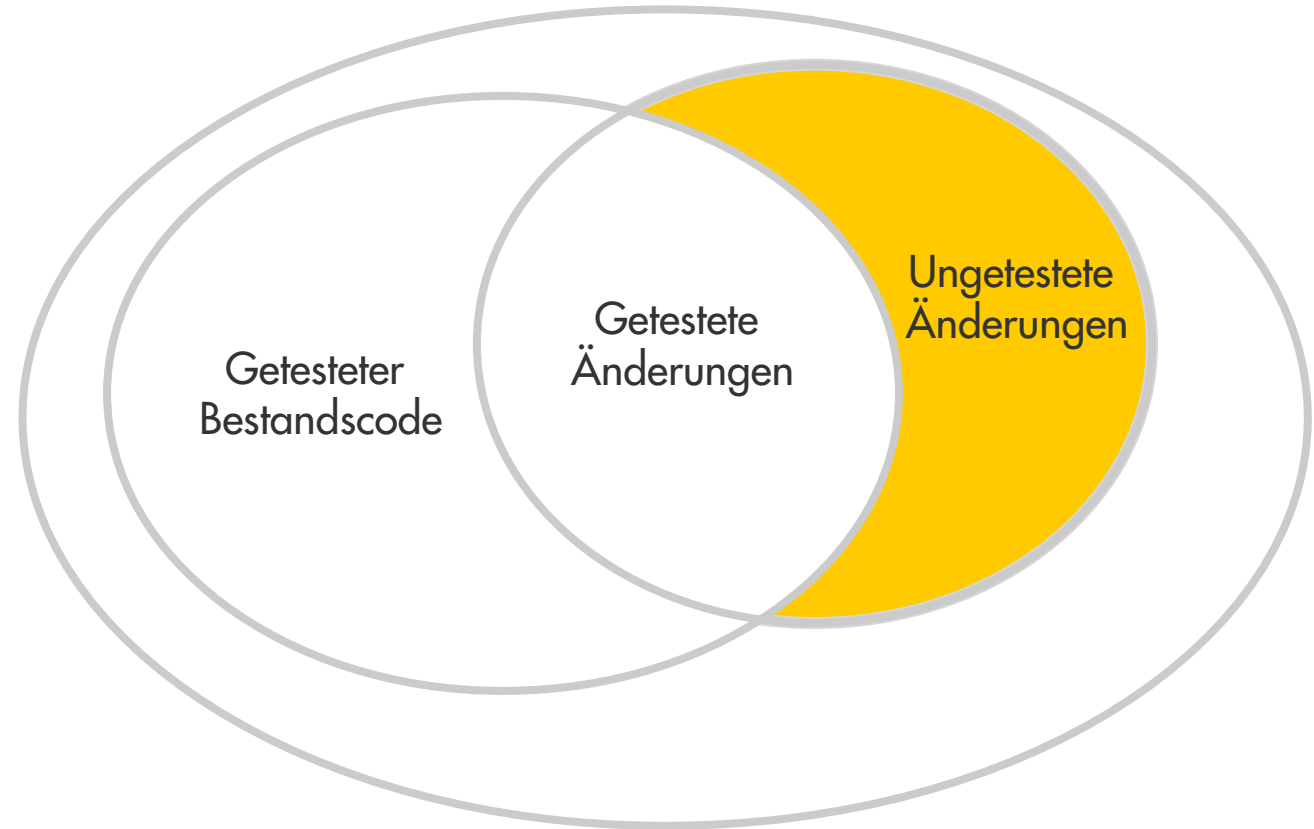
Studie: C# System @ Munich Re

Release A:

15% Code neu/geändert,
>50% ungetestet

Release B:

15% Code neu/geändert,
>60% ungetestet



Feldfehlerwahrscheinlichkeit 5x höher für ungetestete Änderungen!

Änderungen



Test-Gap-Analyse

Ausführung



Ungetestete
Änderungen

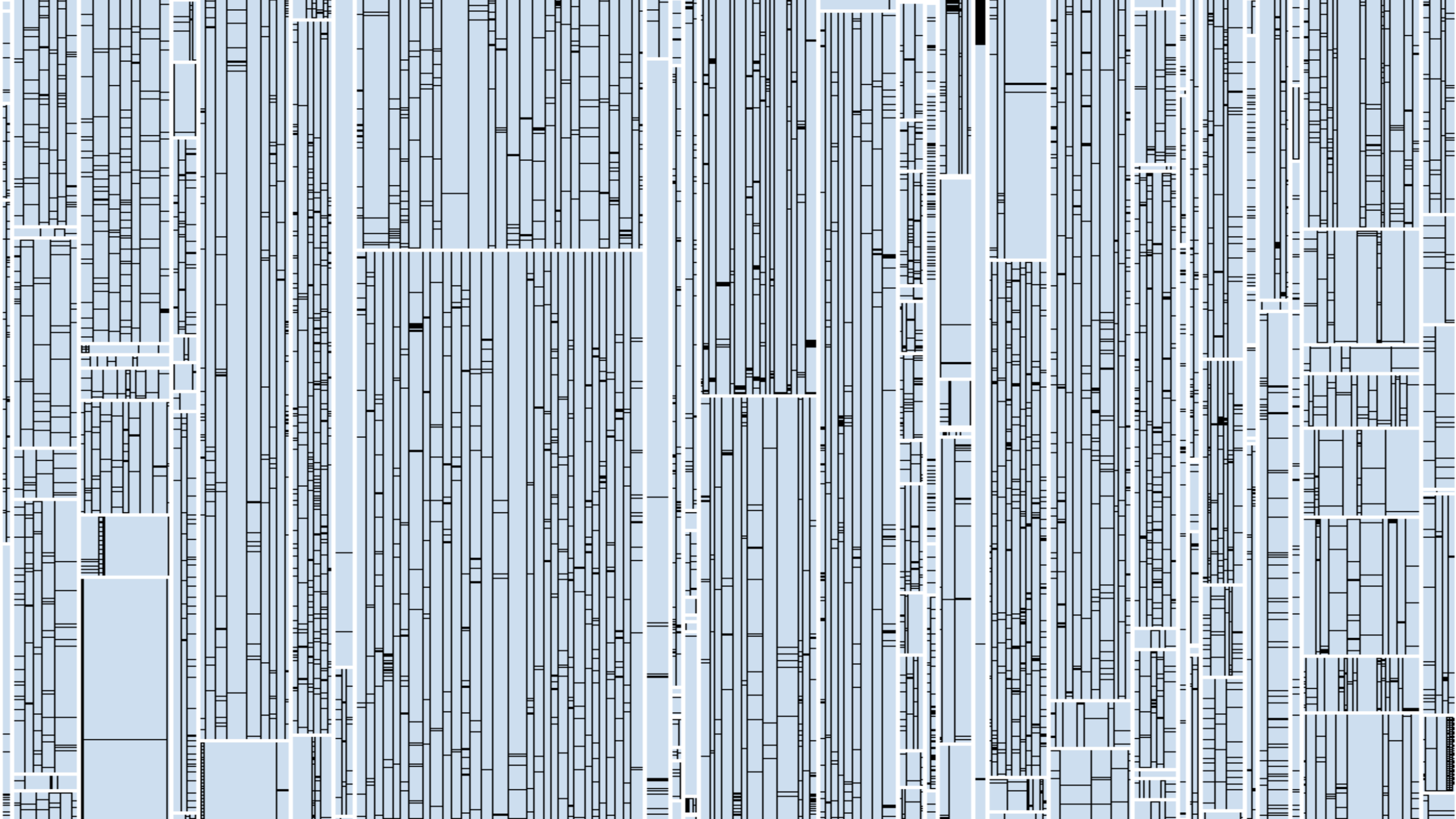
GUI.Dialogs

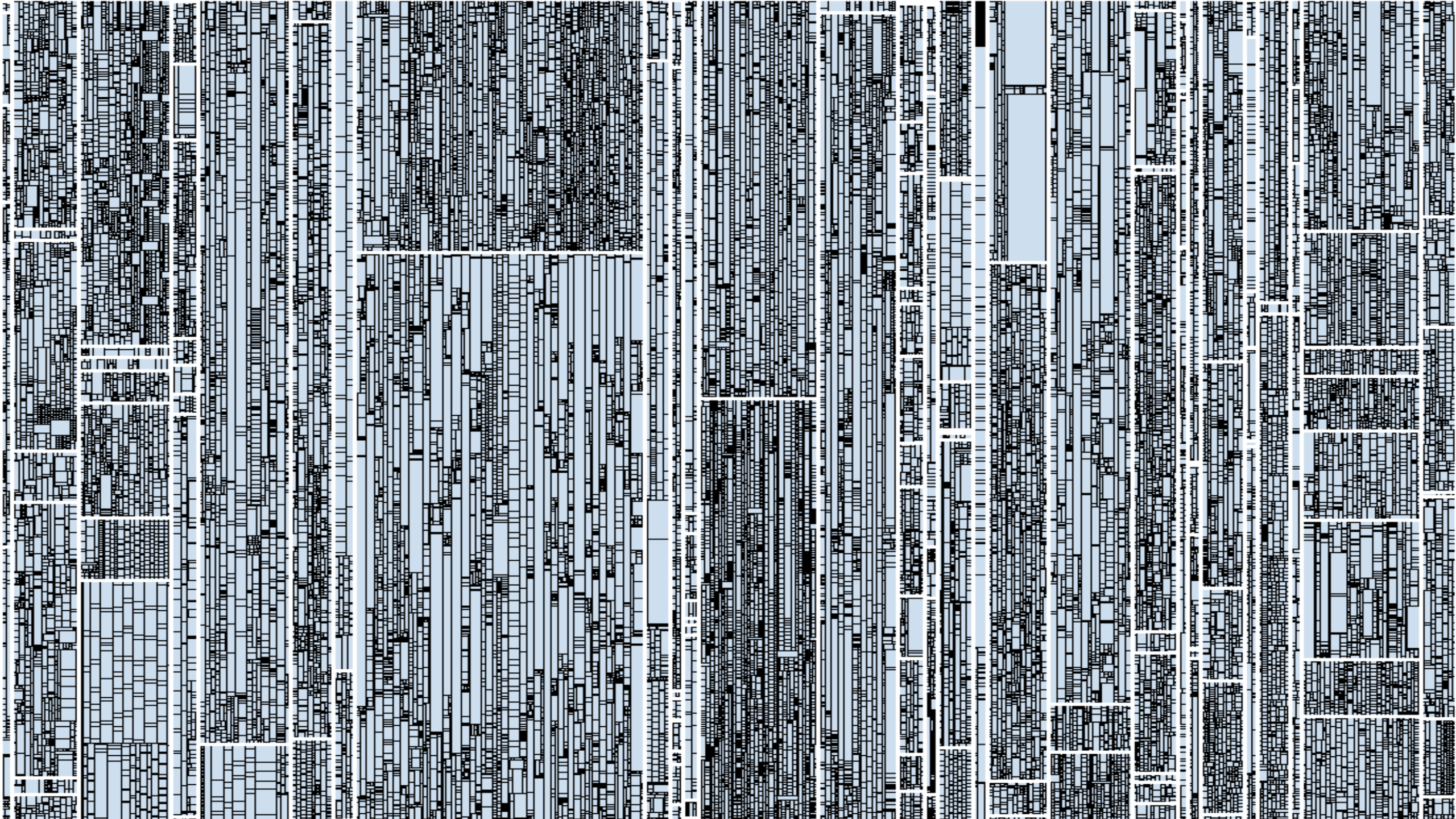
UI Controls

GUI.Base

Authentication

Data
Validation



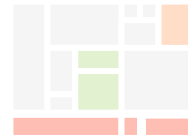


Änderungen



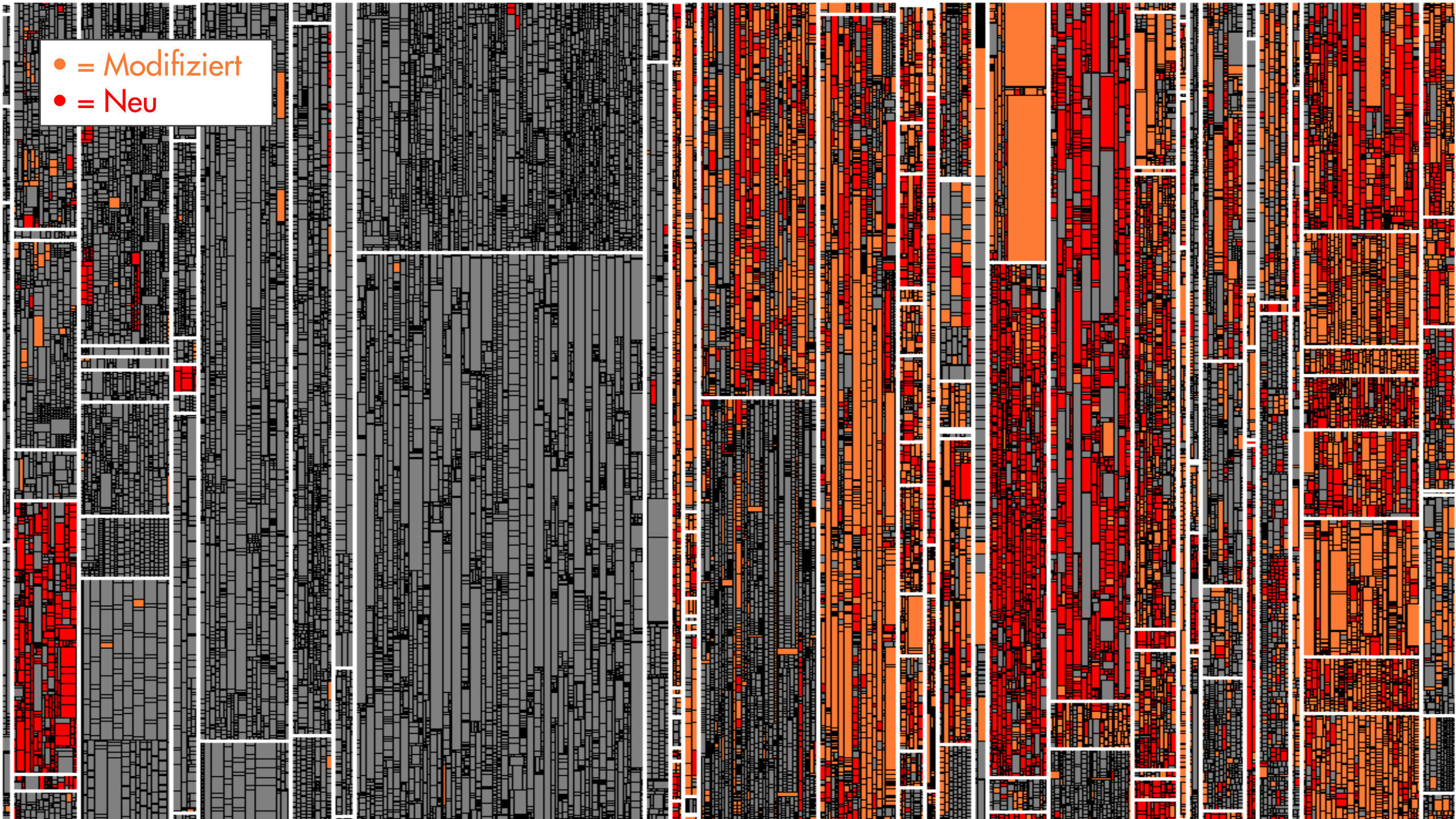
Test-Gap-Analyse

Ausführung

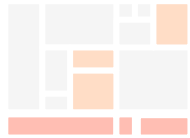


Ungetestete
Änderungen

- = Modifiziert
- = Neu

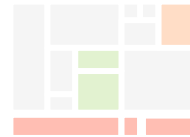


Änderungen



Test-Gap-Analyse

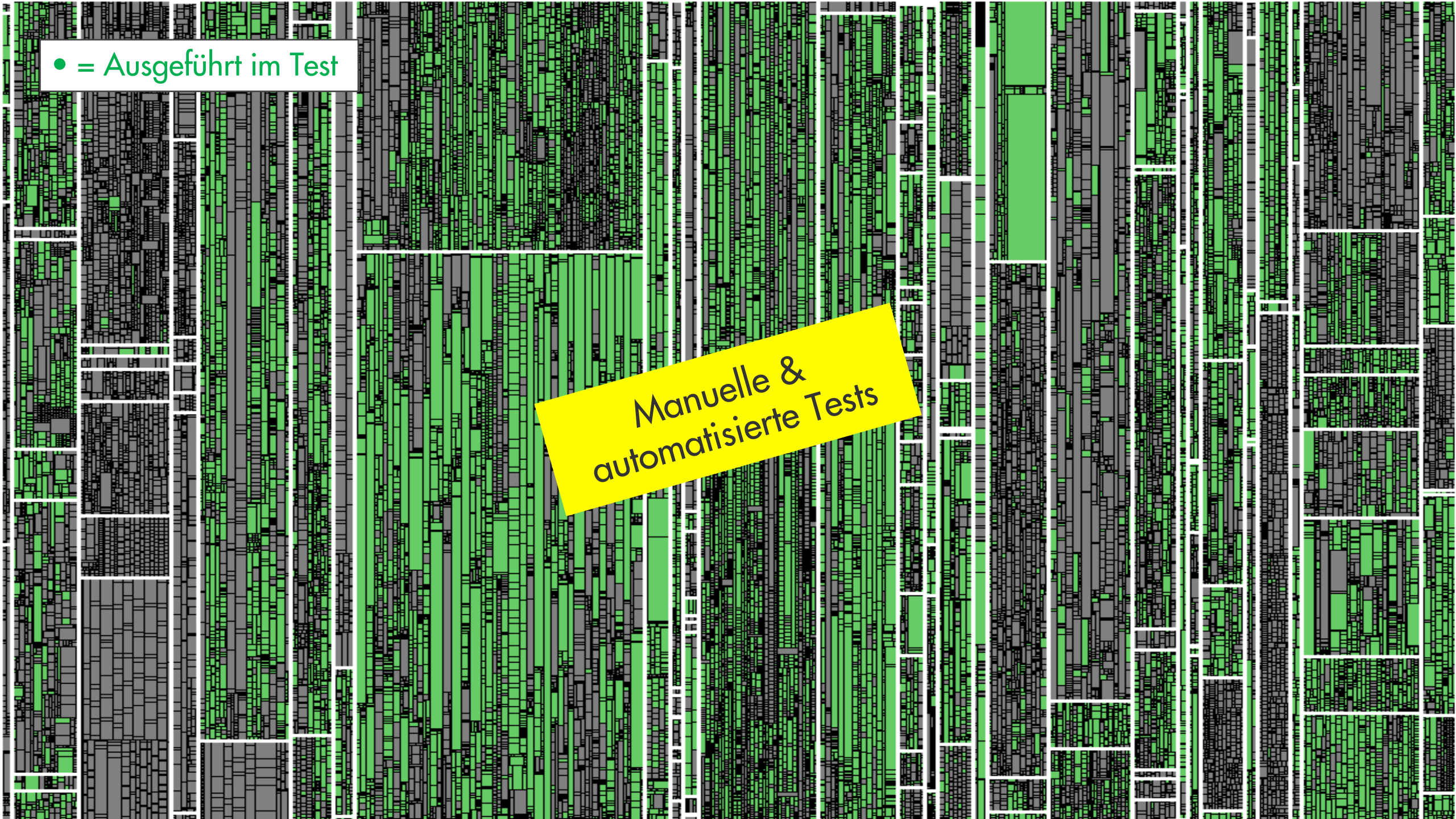
Ausführung



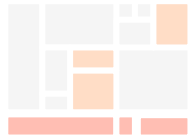
Ungetestete
Änderungen

• = Ausgeführt im Test

Manuelle &
automatisierte Tests

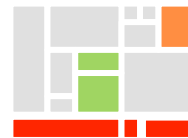
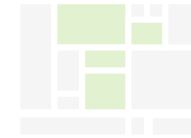


Änderungen



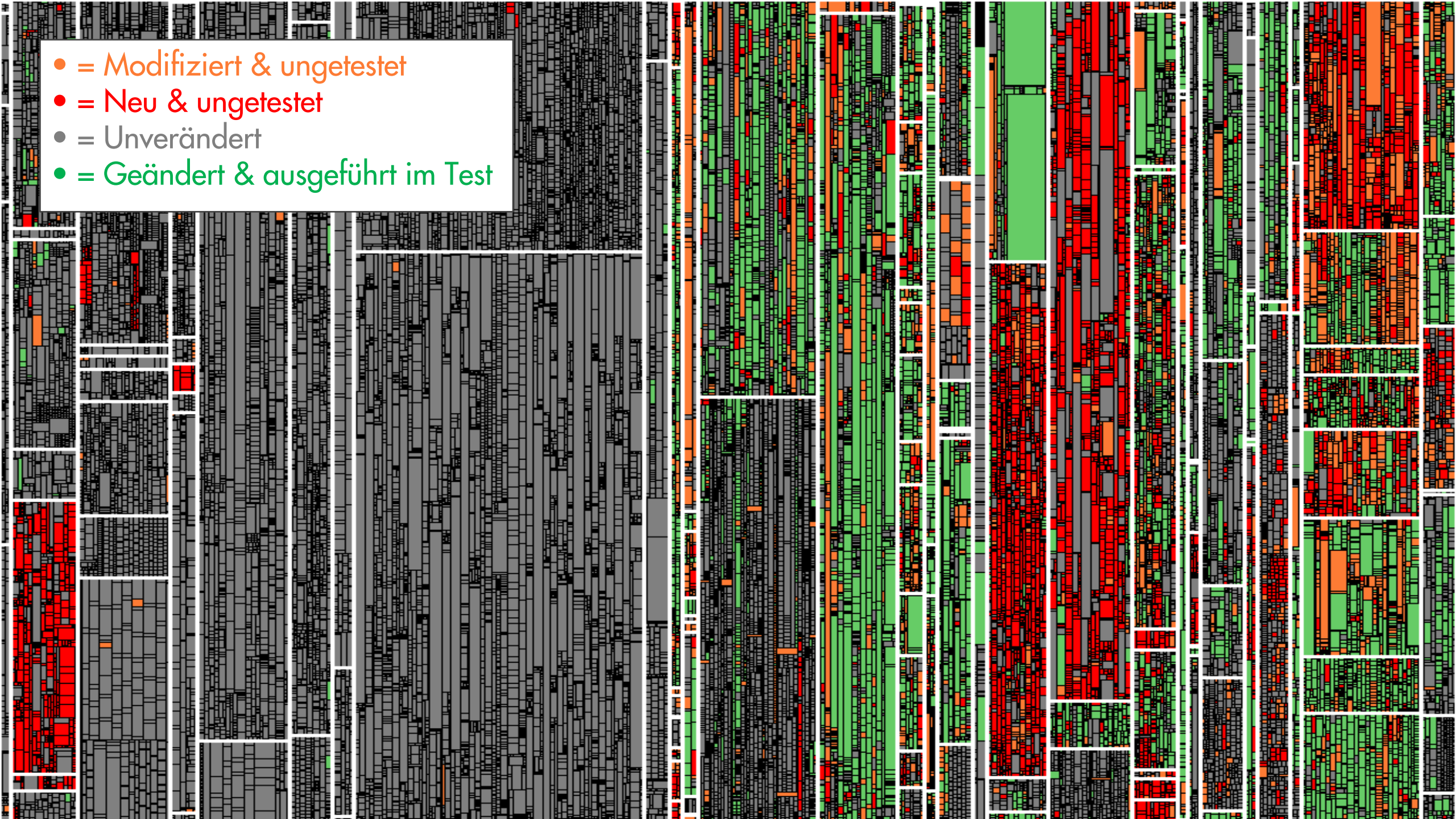
Test-Gap-Analyse

Ausführung



Ungetestete
Änderungen

- = Modifiziert & ungetestet
- = Neu & ungetestet
- = Unverändert
- = Geändert & ausgeführt im Test



Änderungen

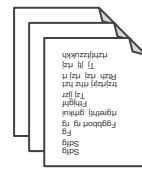


Test-Gap-Analyse

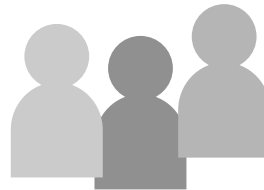


Ungetestete
Änderungen

Ausführung



Testfälle



Entwickler, Tester, Testmanager



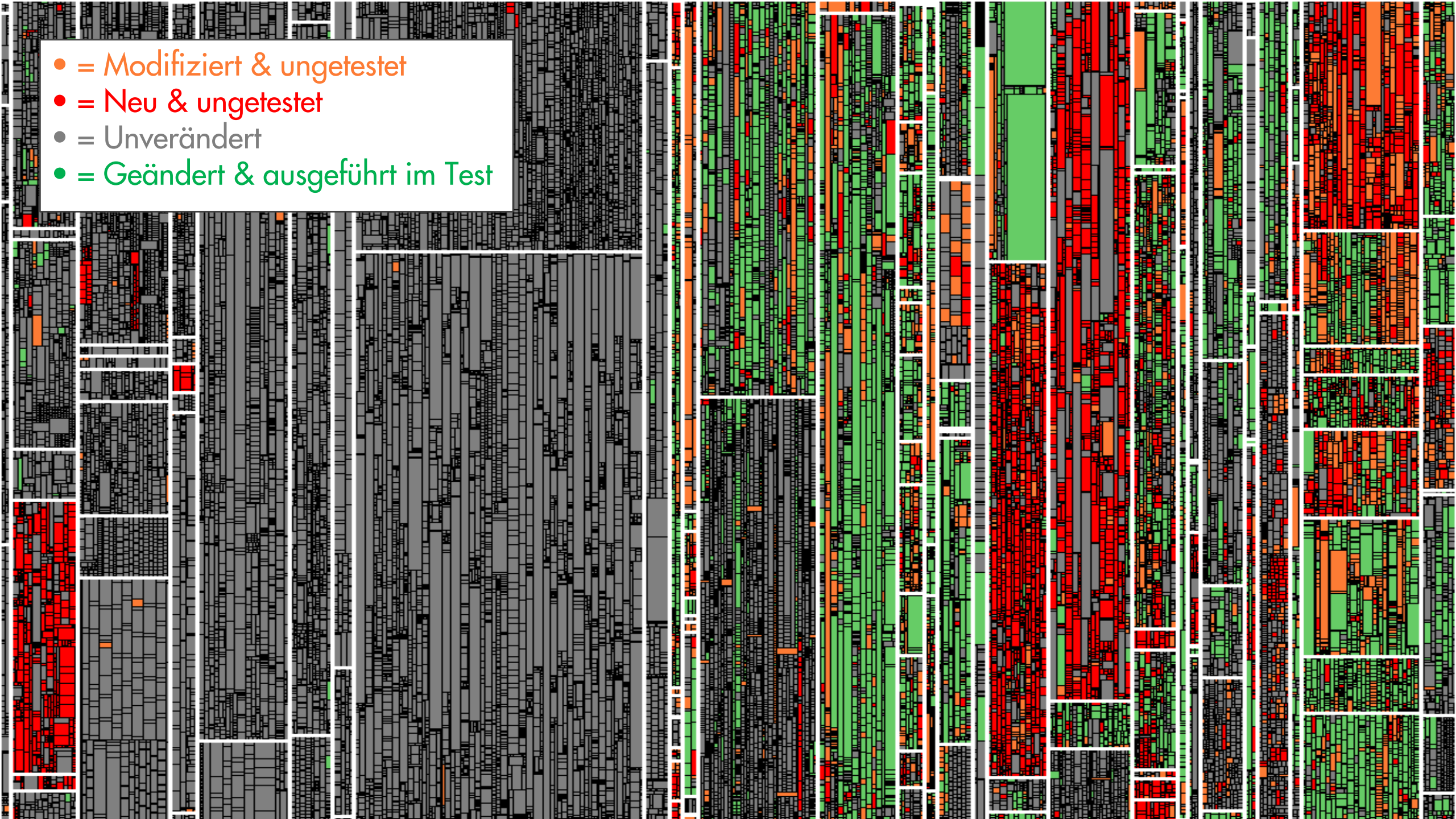


100% Change Coverage

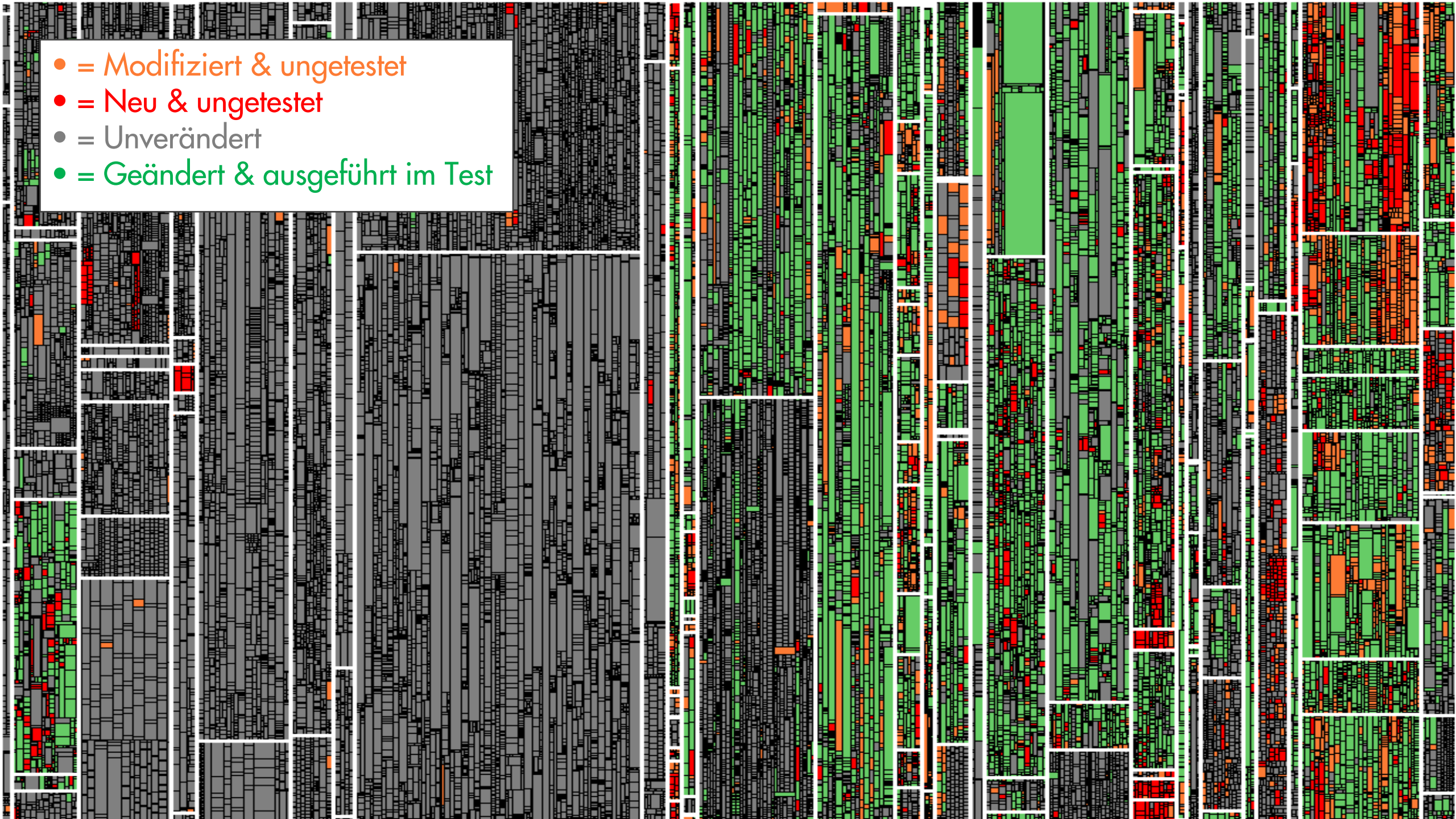


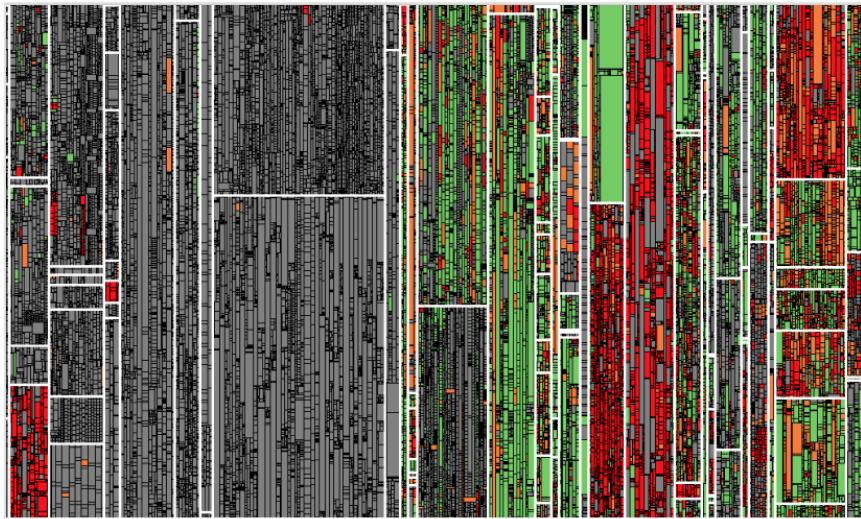
100% Change Coverage → 0 Fehler

- = Modifiziert & ungetestet
- = Neu & ungetestet
- = Unverändert
- = Geändert & ausgeführt im Test



- = Modifiziert & ungetestet
- = Neu & ungetestet
- = Unverändert
- = Geändert & ausgeführt im Test





Haben wir das Richtige getestet? Erfahrungen mit Test-Gap-Analyse in der Praxis



Haben wir das Richtige getestet? Erfahrungen mit Test-Gap-Analyse in der Praxis

Elmar Jürgens
CQSE GmbH
juerg@cqse.de

Dennis Pagano
CQSE GmbH
pagano@cqse.de

Zusammenfassung: Bei limitierter Software-Testzeit sind die Fehler oft, wie viel schneller wird, Testmanpower vermehrt. In der Praxis werden jedoch, wenn Code-Änderungen nicht in gut strukturierten Testprozessen oder ungenügend implementiert sind, häufig Fehler übersehen, die sich erst nach dem Release zeigen. Dies ist ein Problem, das sich durch eine Test-Gap-Analyse lösen lässt.

Nach einer Einführung in Test-Gap-Analyse stellen wir die Erfahrungen vor, die wir in den letzten Jahren im Einsatz bei Kunden und in der eigenen Entwicklung gesammelt haben und geben dabei ein Bild davon ab, wie sich Test-Gap-Analyse in verschiedenen Test-Phasen einsetzen lässt.

WIE GUT WERDEN CODE-ÄNDERUNGEN IN DER PRAXIS DURCH TESTS WIRKLICH ABGEDECKT?

In vielen Systemen machen manuelle Testfälle nach wie vor einen Großteil der Tests aus. In einem großen System ist es oft nicht an der Hand, die manuelle Testfälle zu analysieren, dass sie auch die Änderungen durchdecken, die bei der letzten Testphase durchgeführt wurden und vermutlich die meisten Fehler enthalten.

Um besser zu verstehen, ob die Tests die Änderungen tatsächlich erreichen haben wir eine wissenschaftliche Studie [1] auf einem betrieblichen Informationssystem durchgeführt. Das untersuchte System umfasst ein 50000 Zeilen C#-Code. Wir haben die Studie über 14 Monate Entwicklung durchgeführt und dabei zwei automatisierte Releases unterstützt.

Durch statische Analyse haben wir ermittelt, welche Code-Bereiche für die beiden Releases neu erstellt oder verändert wurden sind. Für beide Releases wurde jeweils etwa 15% des Quelltextes modifiziert. Außerdem haben wir alle Testaktivitäten erhoben. Dafür haben wir die Testbedeckung aller automatisierten und manuellen Tests über mehrere Monate aufzeichnet.

Eine Auswertung der Kombination aus Änderungen und Testdaten zeigt an, dass etwa die Hälfte der Änderungen ungenutzt in Produktion gelangen – obwohl der Quelltext sehr systematisch geprüft und durchgeführt worden war.

WELCHE FOLGEN HABEN UNGENÜTZTE ÄNDERUNGEN?

Um die Konsequenzen der ungenutzten Änderungen für die Anwender des Programms zu quantifizieren, haben wir retrospektiv alle Fehler analysiert, die in den Monaten nach

den Releases aufgetreten sind. Dabei zeigte sich, dass die Fehlerwahrscheinlichkeit in geländerten, ungenutzten Code-Bereichen höher war, als in ungenutzten Code (und auch höher als in genutztem und genutztem Code).

Diese Studie führt uns vor Augen, dass Änderungen in der Praxis sehr häufig ungenutzt in Produktion gelangen und dort die Gefahr der Inzidentalverursachung steigt. Es ist nicht nur, aber auch ein konkreter Ansatzpunkt, um die Testqualität systematisch zu verbessern: wenn es um geringe, Änderungen zu realisieren.

WARUM REISCHT CODE DURCH DEN TEST?

Die Menge an ungenutztem Code in Produktion hat uns offen gezeigt übertrieben ist wie diese Studie zum ersten Mal gemacht haben. Inzwischen haben wir verschiedene Analysen in vielen Systemen, Programmierumgebungen und Firmen durchgeführt und erhalten oft ein ähnliches Bild. Die Ursache für ungenutzten Code liegt jedoch – anders als man vielleicht vermuten könnte – nicht an mangelnder Expertise oder am Einsatz der Tester, sondern vielmehr darin, dass es eine gezielte Analyse sehr schwierig ist, insbesondere in großen Systemen im Test zu realisieren zu entwickeln.

Testmanager orientieren sich bei der Testen oft häufig an den Änderungen, die im Issue-Tracker (Jira, TFS, Redmine, Bugzilla, etc.) dokumentiert sind. Für fast alle dokumentierten Änderungen funktionieren die erfahrungsgemäß auch gut. Testfälle für manuelle Tests beschreiben typischerweise Interaktionen über die Benutzeroberfläche, um gewisse funktionelle Aspekte zu testen. Im Fall der Issue-Tracker Änderungen ist es üblich, wenn die entsprechenden funktionellen Testfälle zur Durchführung ausgewählt.

Unsere Erfahrungen zeigen jedoch, dass Issue-Tracker nur zwei Gründe keine geeignete Informationsquelle sind, um Änderungen vollständig zu finden. Erstens gibt es häufig noch nicht mehr Änderungen, wie beispielsweise Aufnahmefähigkeit oder Anpassungen an neue Versionen von Bibliotheken oder Schnittstellen zu Fremdsystemen. Bei Änderungen, die es für Tester nicht nachvollziehbar, welche funktionellen Testfälle durchgeführt werden müssen, um diese Änderungen zu realisieren.

Zweites ist, dass Änderungen im Issue-Tracker vorliegen, die in den Zeilen oder auf anderen Kanälen (z.B. Slack) sind und die dies im Issue-Tracker nicht sichtbar. Um Änderungen

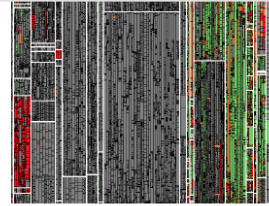


Abbildung 4. Test-Gaps am Ende der Testphase. Unentdeckte Methoden sind grau dargestellt. Gleichzeitige ungenutzte Methoden sind rot, getriggerte ungenutzte Methoden orange dargestellt.

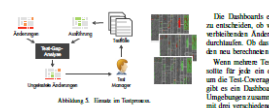


Abbildung 5. Einmal im Testprozess.

WIE KANN TEST-GAP-ANALYSE EINSATZBAR WERDEN?

Nützlich wird die Test-Gap-Analyse dann, wenn sie kontinuierlich angelegt wird, beispielsweise jede Nacht, um einen guten Überblick über die ausgeführten Tests und Änderungen bis zum letzten Abend zu geben. Hierbei werden hauptsächlich die Informationen in den Test-Gaps erstellt, wie in Abbildung 5 gezeigt.

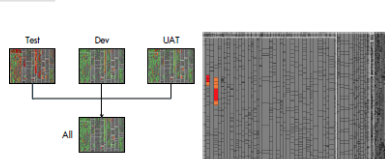


Abbildung 6. Einmal von mehreren Durchläufen für statische Analyse.

FÜR WELCHES PRODUKT IST TEST-GAP-ANALYSE EINSATZBAR?

Wir haben Test-Gap-Analyse bereits in den unterschiedlichsten Projekten eingesetzt: von betrieblichen Informationssystemen bis hin zu eingebetteter Software, von C++ über Java, C# und Python bis hin zu ABAP. Faktoren, die die Komplexität der Implementierung beeinflussen, umfasst unter anderem:

- Ausführungsplattform, Virtuelle Maschinen (z.B. Java, C#, ABAP) erschweren die Erhebung von Test-Generatoren
- Architektur: Bei Server-basierten Anwendungen müssen die Test-Coverage-Daten auf weniger Maschinen erhoben werden, als bei Full-Client-Anwendungen
- Betriebssysteme: Testpläne erschweren die Planung und Realisierung

WAS BRINGT TEST-GAP-ANALYSE IM HOMER TEST?

Bei Test von Full-Client-Anwendungen ist es sehr wenig Zeit zur Verfügung, falls im Home-Test sind ein wenig sicherzustellen, dass der behobene Fehler nicht mehr auftritt und zu vermeiden, dass dabei keine neuen Fehler eingebaut wurden. Ihr letztes sollte empfindlich schmerzhaft werden, dass alle im Home-Test durchgeführten Änderungen durchlaufen werden. Hierbei sind in der Test-Gap-Analyse der Release-Stand als Referenzversion definiert und alle Änderungen ermittelt, die für das Home-Test, auf einem eigenen Branch durchgeführt wurden, wie in Abbildung 7 dargestellt.

Mit Hilfe der Test-Gap-Analyse wird dann ermittelt, ob im Home-Test tatsächlich alle Änderungen durchlaufen wurden sind. Im Beispiel in Abbildung 8 zeigt sich, dass ein Teil der Methoden nicht ausgeführt wird. Unsere Erfahrungen haben gezeigt, dass sich gerade in Home-Test durch Test-Gap-Analyse nachvollziehbar und einfach die Sicherheit erhöhen lässt, durch die Änderungen keine neuen Fehler einschleusen.

WAS BRINGT TEST-GAP-ANALYSE IM RELEASE-TEST?

An Release-Test beschäftigen wir in diesem Artikel die Test-Phase vor einem großen Release, in der typischerweise

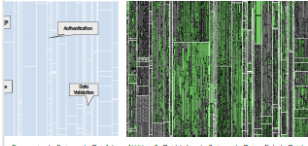


Abbildung 7. Test-Gaps am Ende der Testphase. Unentdeckte Methoden sind grau dargestellt. Gleichzeitige ungenutzte Methoden sind rot, getriggerte ungenutzte Methoden orange dargestellt.

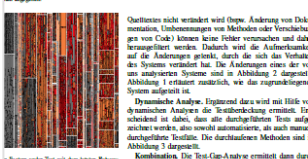


Abbildung 8. Test-Gaps am Ende der Testphase. Unentdeckte Methoden sind grau dargestellt. Gleichzeitige ungenutzte Methoden sind rot, getriggerte ungenutzte Methoden orange dargestellt.

Die Menge an ungenutztem Code in Produktion hat uns offen gezeigt übertrieben ist wie diese Studie zum ersten Mal gemacht haben. Inzwischen haben wir verschiedene Analysen in vielen Systemen, Programmierumgebungen und Firmen durchgeführt und erhalten oft ein ähnliches Bild. Die Ursache für ungenutzten Code liegt jedoch – anders als man vielleicht vermuten könnte – nicht an mangelnder Expertise oder am Einsatz der Tester, sondern vielmehr darin, dass es eine gezielte Analyse sehr schwierig ist, insbesondere in großen Systemen im Test zu realisieren zu entwickeln.

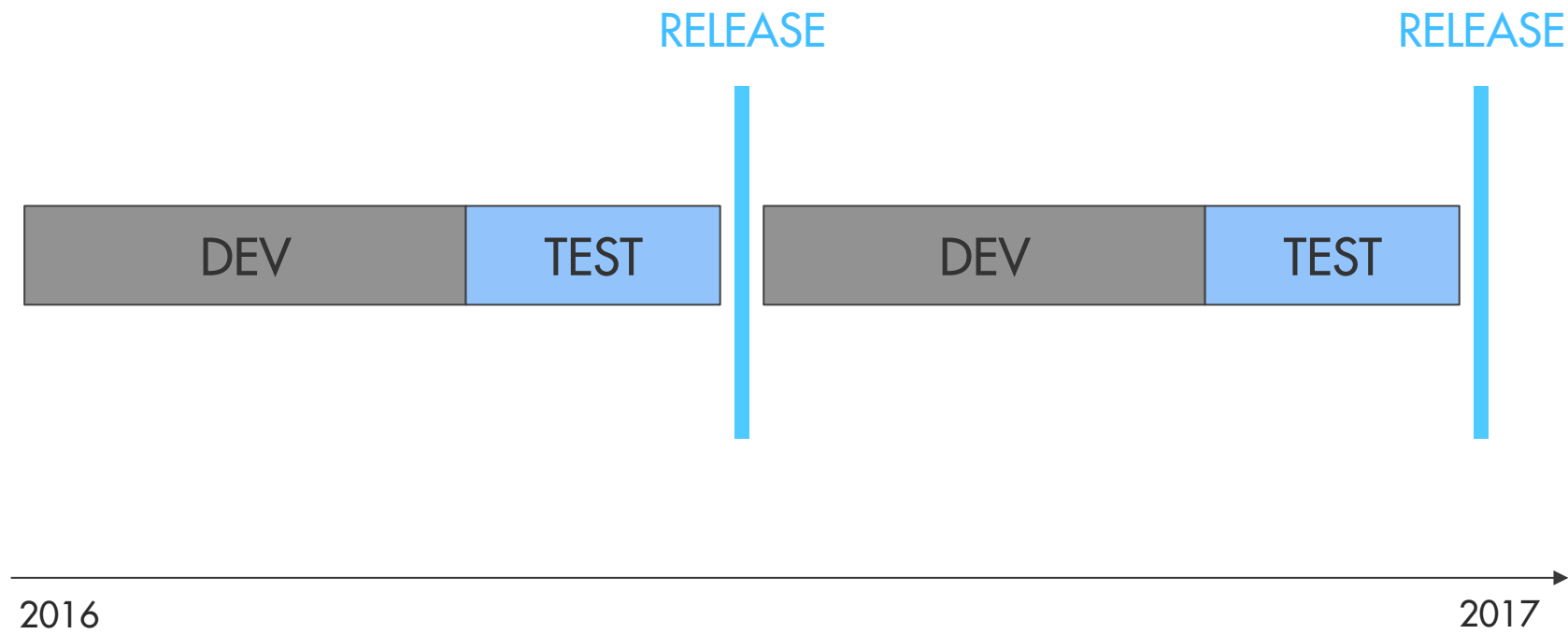
Testmanager orientieren sich bei der Testen oft häufig an den Änderungen, die im Issue-Tracker (Jira, TFS, Redmine, Bugzilla, etc.) dokumentiert sind. Für fast alle dokumentierten Änderungen funktionieren die erfahrungsgemäß auch gut. Testfälle für manuelle Tests beschreiben typischerweise Interaktionen über die Benutzeroberfläche, um gewisse funktionelle Aspekte zu testen. Im Fall der Issue-Tracker Änderungen ist es üblich, wenn die entsprechenden funktionellen Testfälle zur Durchführung ausgewählt.

Unsere Erfahrungen zeigen jedoch, dass Issue-Tracker nur zwei Gründe keine geeignete Informationsquelle sind, um Änderungen vollständig zu finden. Erstens gibt es häufig noch nicht mehr Änderungen, wie beispielsweise Aufnahmefähigkeit oder Anpassungen an neue Versionen von Bibliotheken oder Schnittstellen zu Fremdsystemen. Bei Änderungen, die es für Tester nicht nachvollziehbar, welche funktionellen Testfälle durchgeführt werden müssen, um diese Änderungen zu realisieren.

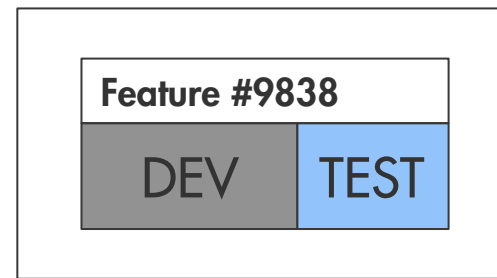
Zweites ist, dass Änderungen im Issue-Tracker vorliegen, die in den Zeilen oder auf anderen Kanälen (z.B. Slack) sind und die dies im Issue-Tracker nicht sichtbar. Um Änderungen

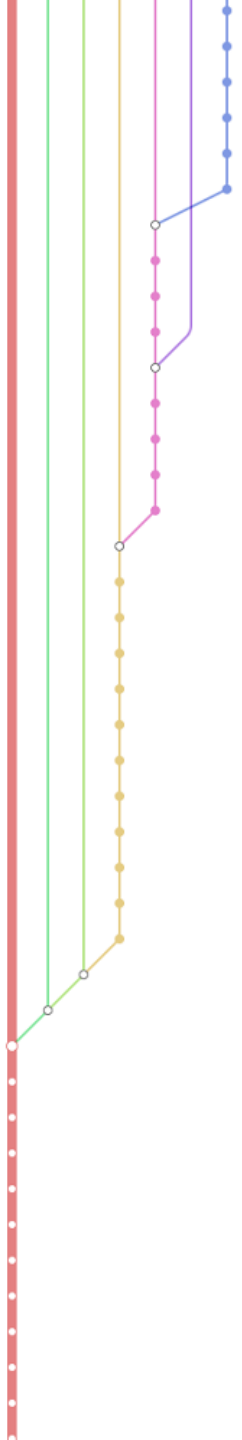


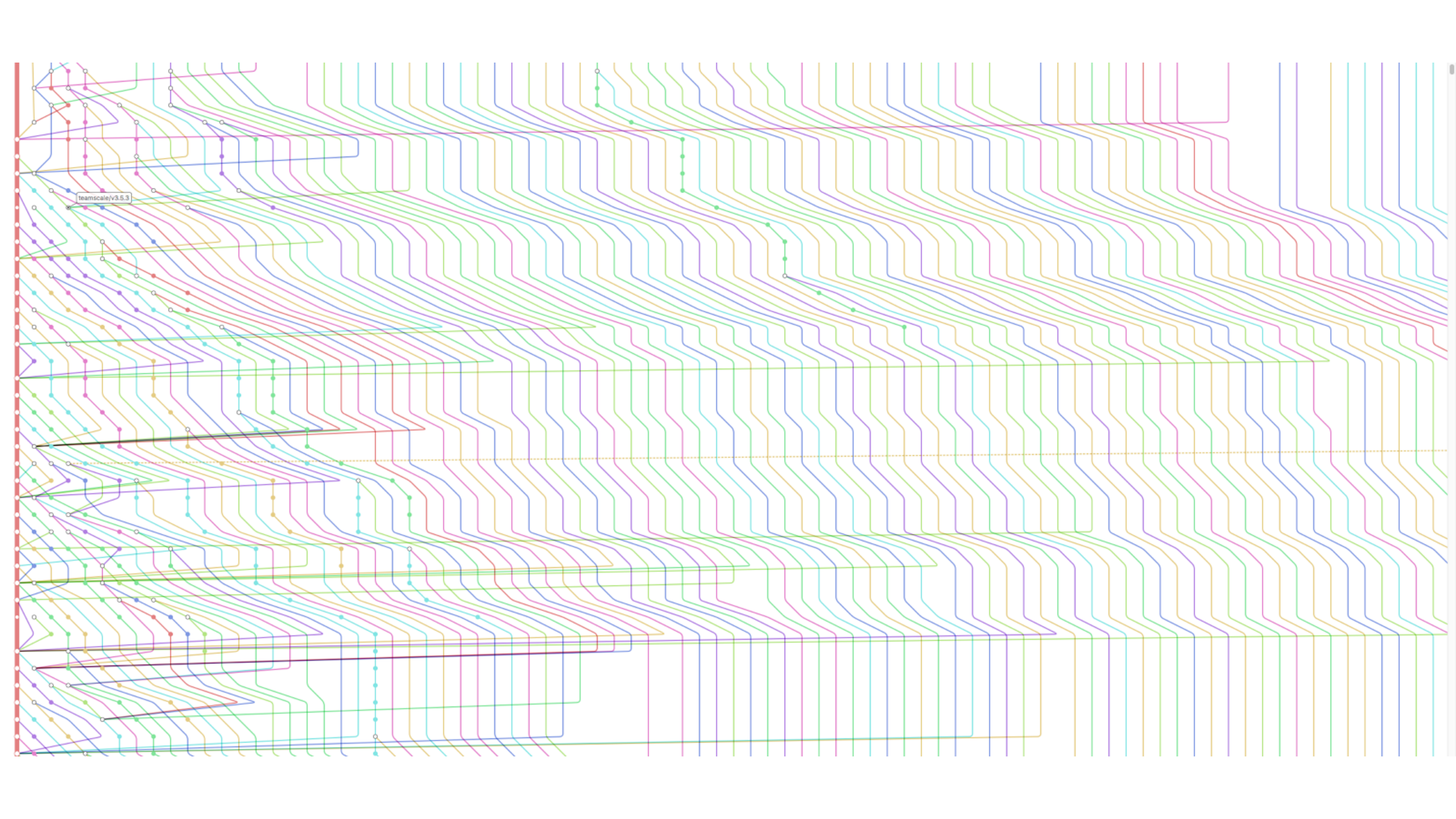
Test Focus: Release

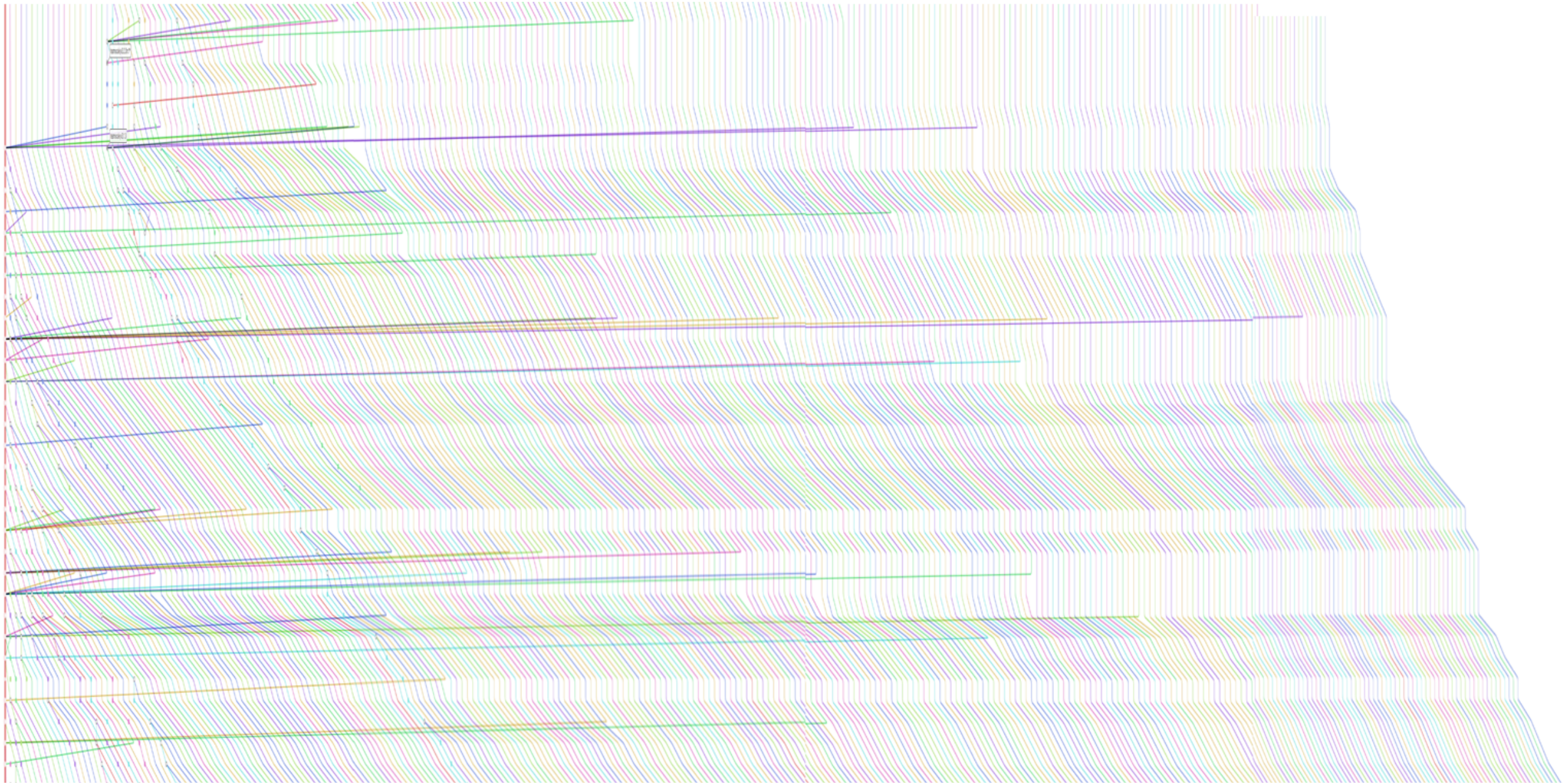


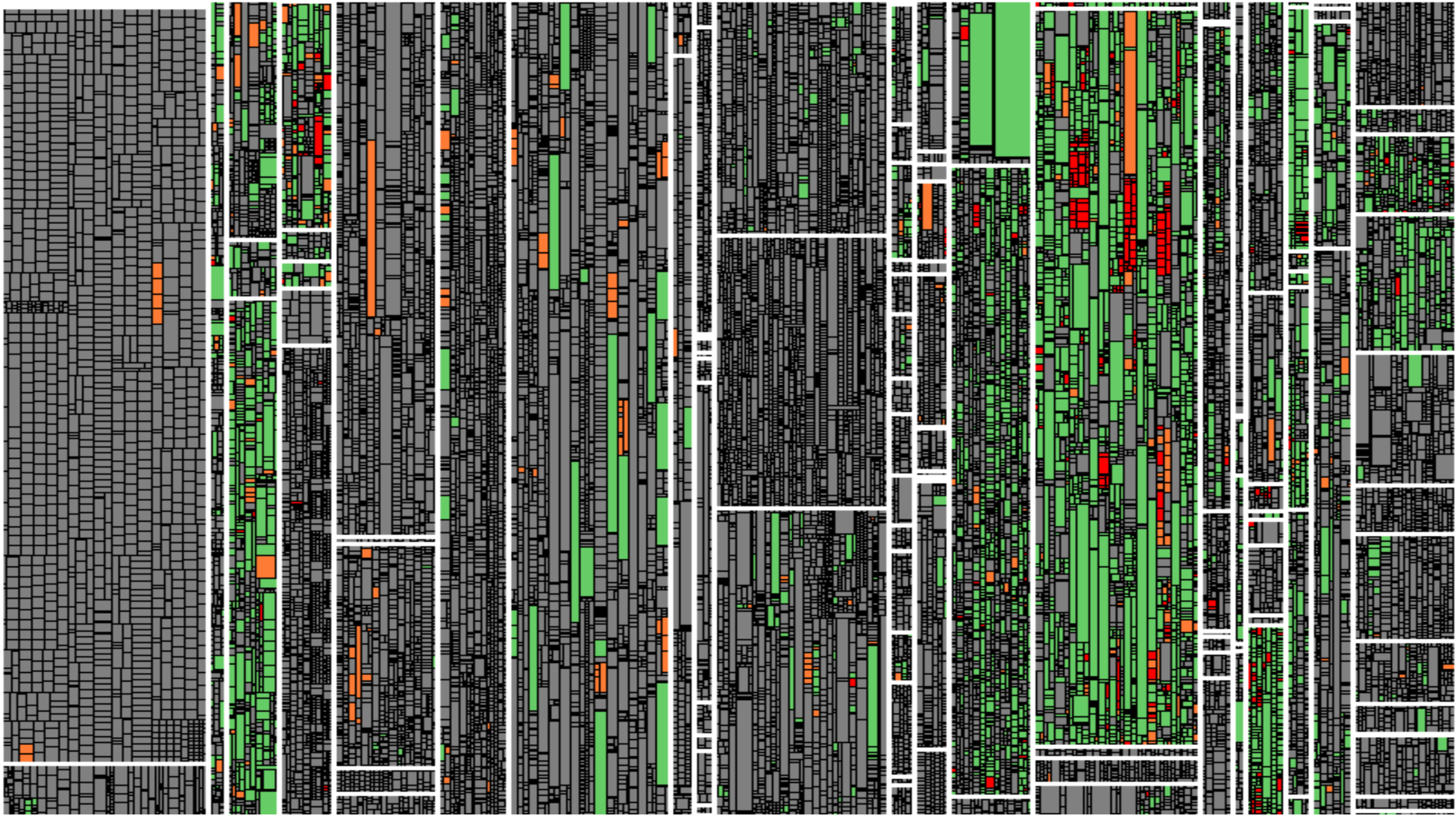
Test Focus: Ticket

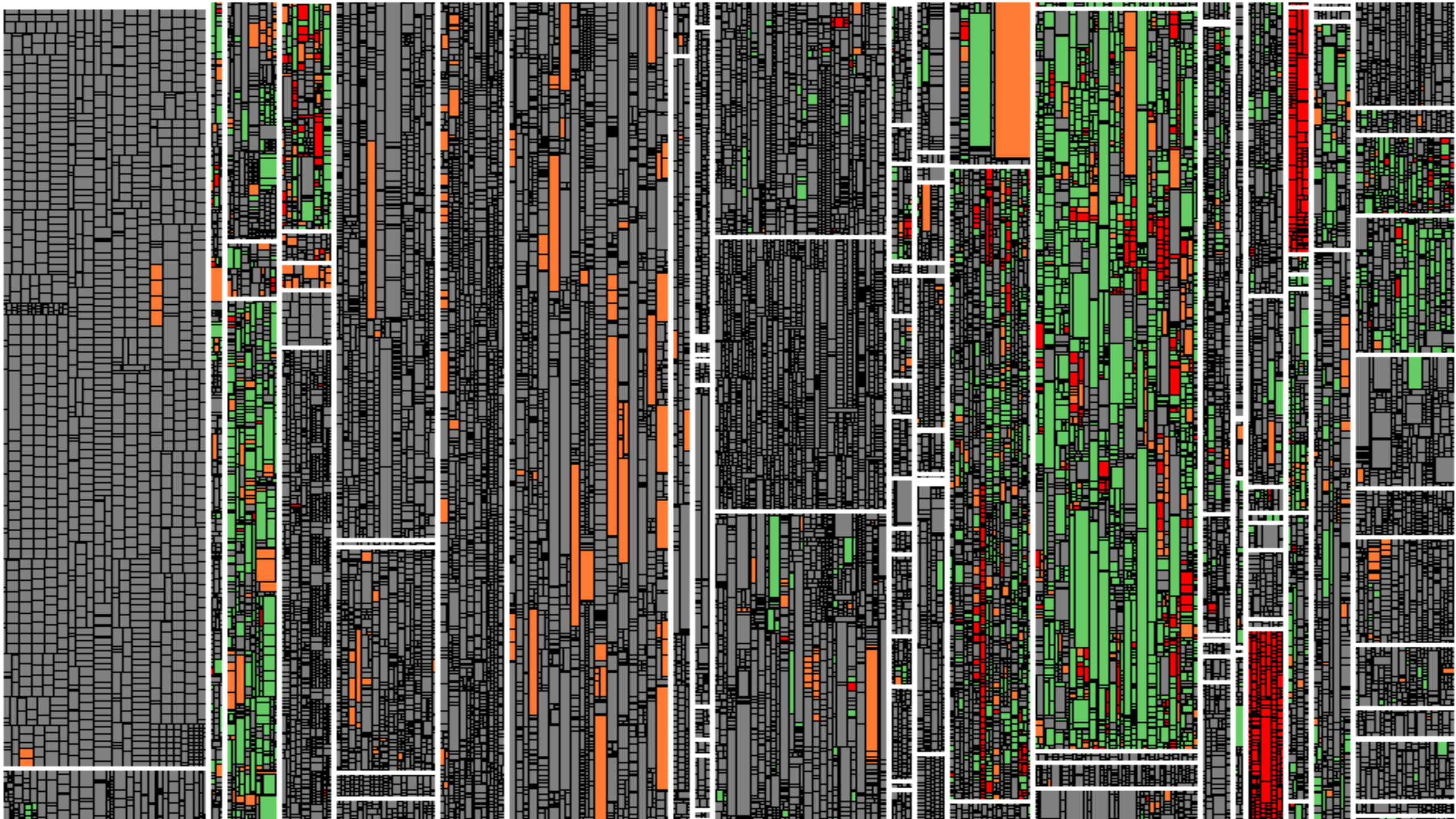



























Ticket Coverage

Ticket Im Issue Tracker verwaltete Unit of Work. Bug, Issue, Change Request, User Story, ...

Ticket Code Code, der bei der Implementierung eines Tickets angefasst (geändert oder neu geschrieben) wurde.

Ticket Coverage Anteil des Ticket Codes, der im Test zur Ausführung kam.

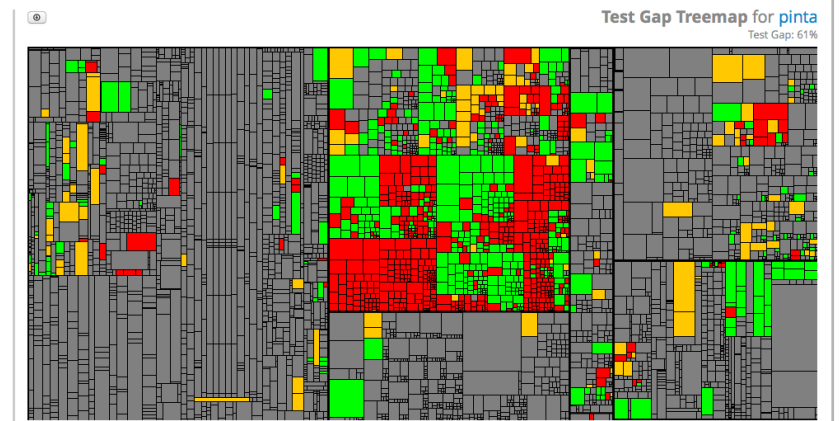
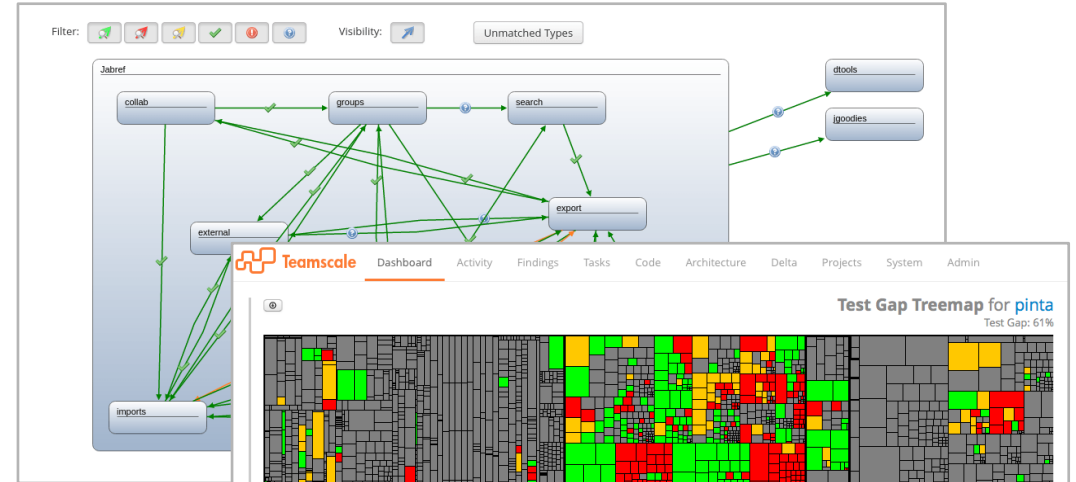
Number	Subject	Ticket Coverage
 TS-4928	Icons for buttons in Dashboard Edit UI	0.0%
 TS-5227	See full revision history of (deleted) files	20.0%
 TS-5241	Support inconsistent clones	74.0%
 TS-5253	Storing file unit size in chunks	100.0%
 TS-5291	Provide means to filter/not report LSL findings for JS constructors (in our code)	0.0%
 TS-5433	Case-insensitive file name support for IDE Plugins	7.0%
 TS-5469	Adding/removing metrics to chart widget changes colors of other metrics	33.3%
 TS-5557	Ratio metric is shown in percentage format but boundary values are interpreted as float	14.2%
 TS-5666	IDE VS > VS Client should show visual feedback for finding location	0.8%
 TS-5667	IDE VS > VS Client should jump to the tracked finding position on selecting a finding from the finding window.	40.0%
 TS-5730	Provide visual feedback / hide menus for files not mapped to teamscale in VS-IDE	100.0%
 TS-5834	Beautify start page of architecture perspective	100.0%
 TS-6088	Allow to inspect clones by clone class	90.0%
 TS-6099	Show 'potentially outdated' nightly finding more defensively	50.0%
 TS-6183	Refactor project mappings?	75.0%
 TS-6211	Several improvements to Teamscale IDE	66.7%
 TS-6213	Allow to filter displayed findings to files with local modifications	55.0%
 TS-6292	Connector validation should also check other fields, like include patterns	0.0%
 TS-6299	Support deep comment analysis (category metrics, findings) also for JavaScript	11.0%



100% Ticket Coverage → 0 Fehler

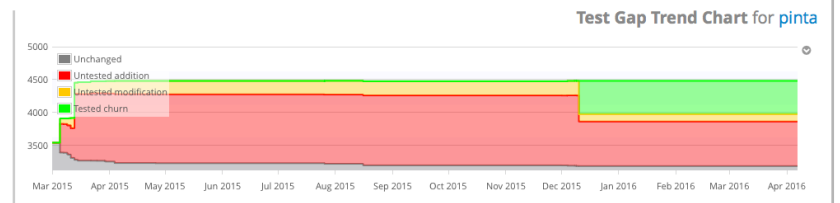
Studienobjekt: Teamscale

- 750 kLOC Code (Java, JavaScript)
- 11 Jahre Historie im VCS
- >70k Commits
- 30 aktive Entwickler
- Code Peer Reviews



```
/* Process the input string received prior to the
newline. */
do
{
    /* Pass the string to FreeRTOSCLI. */
    xMoreDataToFollow = FreeRTOS_CLIProcessCommand( cInput
    /* Send the output generated by the command's
implementation. */
    sendto( xSocket, cOutputString, strlen( cOutputString
) while( xMoreDataToFollow != pdFALSE ); /* Until the com
/* All the strings generated by the command processing
have been sent. Clear the input string ready to receive
the next command. */
cInputIndex = 0;
memset( cInputString, 0x00, cmdMAX_INPUT_SIZE );
/* Transmit a spacer, just to make the command console
easier to read. */
sendto( xSocket, "\r\n", strlen( "\r\n" ), 0, ( SOCKADDR
}
else
{
    if( cInChar == '\n' )
    {
        /* Ignore the character. Newlines are used to
detect the end of the input string. */
    }
    else if( cInChar == '\b' )
    {
        /* Backspace was pressed. Erase the last character
in the string - if any. */
        if( cInputIndex > 0 )
        {
            cInputIndex--;
            cInputString[ cInputIndex ] = '\0';
        }
    }
    else
    {

```



```
else
{
    if( cInChar == '\n' )
    {
        /* Ignore the character. Newlines are used to
detect the end of the input string. */
    }
    else if( cInChar == '\b' ) || ( cInChar == cmdASCII_DEL )
    {
        /* Backspace was pressed. Erase the last character
in the string - if any. */
        if( cInputIndex > 0 )
        {
            cInputIndex--;
            cInputString[ cInputIndex ] = '\0';
        }
    }
    else
    {

```

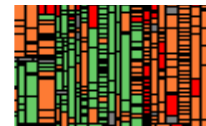
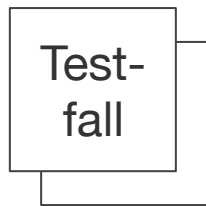


Forschungsfragen der empirischen Studie

- 1) Finden **Tester** mittels Ticket Coverage relevante Test-Gaps bei **strukturierten** Tests?
- 2) Finden **Entwickler** mittels Ticket Coverage relevante Test-Gaps bei **explorativen** Tests?

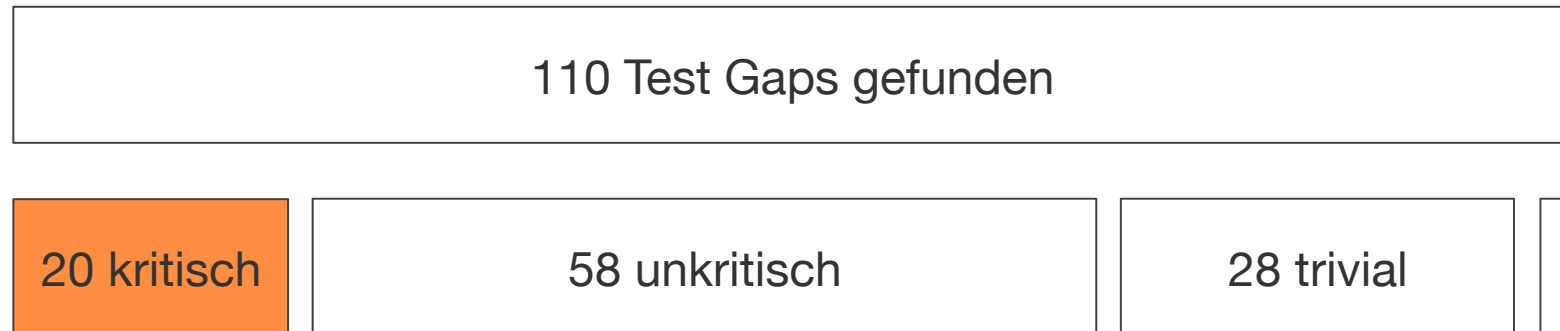
Study Design

Finden **Tester** mittels Ticket Coverage relevante Test-Gaps bei **strukturierten** Tests?



- 54 Tickets analysiert
- < 20 Monate alt
- ≤ 3 pro Entwickler

Ergebnis



Information in Tickets (und daraus abgeleiteten Testfällen) oft unvollständig.
Irrelevante Test-Gaps haben zu Verbesserung der Analyse beigetragen.

Empirische Studie (2)

Finden **Entwickler** mittels Ticket Coverage relevante Test-Gaps bei **explorativen** Tests?



⌵ Affected methods (22) 

Ticket Coverage: 86%

Class	Method	Line Coverage	Uncovered Lines ^	Change type
ProjectCreationService	retrieveProjectConfig	100%	0	changed
ProjectCreationService	fieldChangeRequiresReAnalysis	100%	0	added
ProjectService	elementUpdateQuery	100%	0	added
ProjectReanalysisService	process	100%	0	changed
ProjectCreator	refreshProject	100%	0	added
ProjectCreationService	nullOrToString	66%	1	added
ProjectCreationService	findConnectorByName	75%	1	added
ProjectCreationService	connectorRequiresReAnalysis	82%	3	added
ProjectCreationService	processPutRequest	84%	3	changed
ProjectCreationService	validateProjectConfiguration	80%	3	changed
ProjectCreationService	projectReAnalysisRequired	61%	5	added
ProjectCreationService	connectorsRequireReAnalysis	50%	7	added
ConfigOptionDescriptorBase	ConfigOptionDescriptorBase	100%	0	changed
NamingConventionConfiguration	NamingRegexOption	100%	0	changed
ProjectService	ProjectUpdateResult	100%	0	added

Empirische Studie (2)

Finden **Entwickler** mittels Ticket Coverage relevante Test-Gaps bei **explorativen** Tests?



95 Methoden analysiert. 30 enthielten Test-Gaps. **23** davon relevant.

Für Entwickler sind Ticket & Code-Änderungen unzureichend für vollständigen Test.

Fazit

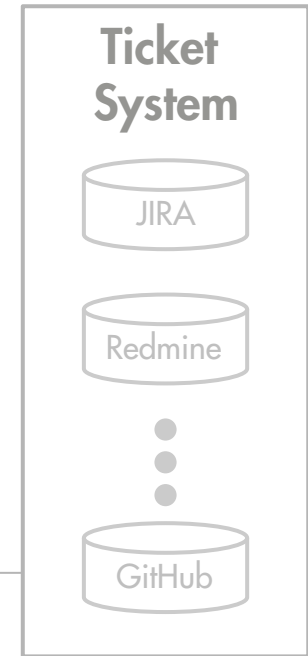
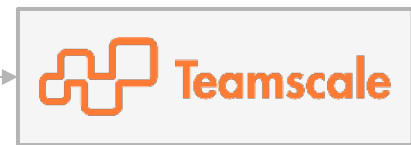
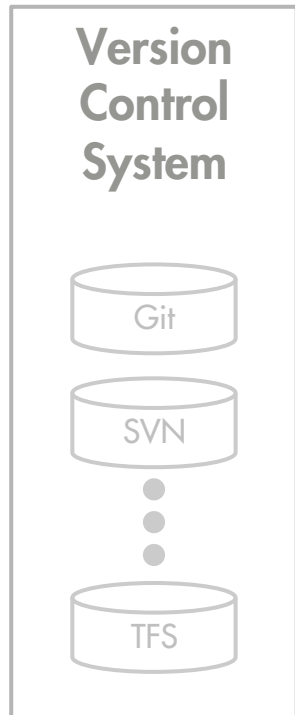
1) Finden **Tester** mittels Ticket Coverage relevante Test-Gaps bei **strukturierten** Tests?

JA: Information in Tickets (und daraus abgeleiteten Testfällen) oft unvollständig.

2) Finden **Entwickler** mittels Ticket Coverage relevante Test-Gaps bei **explorativen** Tests?

JA: Selbst für Entwickler sind Ticket & Code-Änderungen unzureichend für vollständigen Test.

CR#9838: Added TODO	26.07.16 16:38
CR#9838: Adjust naming	26.07.16 15:33
CR#9533: RED	26.07.16 15:13
CR#9533: GREEN	26.07.16 15:12
CR#10181: Added new finding for deprecated classes, methods and fields	26.07.16 14:43
CR#10037: Moved ReviewMetricsSynchronizer to Crucible package and made some improvements to its internal structure	26.07.16 14:31
CR#10037: Updated aggregation strategy of open reviews so each review is only counted once, even over multiple files	26.07.16 13:04
CR#10203: Fixed "field could be made final" for Java interfaces	26.07.16 12:16
CR#10200: Rename pathRestriction -> subPath (1)	26.07.16 11:35
CR#10200: Rename pathRestriction -> subPath (1)	26.07.16 11:35
CR#10172: Removed unwanted colons from headers in the commit view of the activity perspective	26.07.16 11:20
CR#9838: Fix: only one color of a threshold is specified in a corridor	26.07.16 11:14
CR#0: Fix findings	26.07.16 11:01
CR#9838: minor improvement	26.07.16 10:56
CR#10199: Mail notifications do now support starTLS	26.07.16 10:52
CR#9533: working on developer feedback	26.07.16 09:50
CR#9838: Amend last commit	26.07.16 09:38
CR#9838: minor refactoring	26.07.16 09:05
CR#9838: Fix NPE	26.07.16 09:01



```

public static synchronized void log(String methodIdentifier) {
    if (s_mode == LoggingMode.TESTING) {
        s_testingLog.add(methodIdentifier);
    } else if (s_mode == LoggingMode.FRAMING) {
        s_framingLog.add(methodIdentifier);
    } else {
        throw new IllegalStateException();
    }
}

public static synchronized Set<String> getFramingLog() {
    return s_framingLog;
}

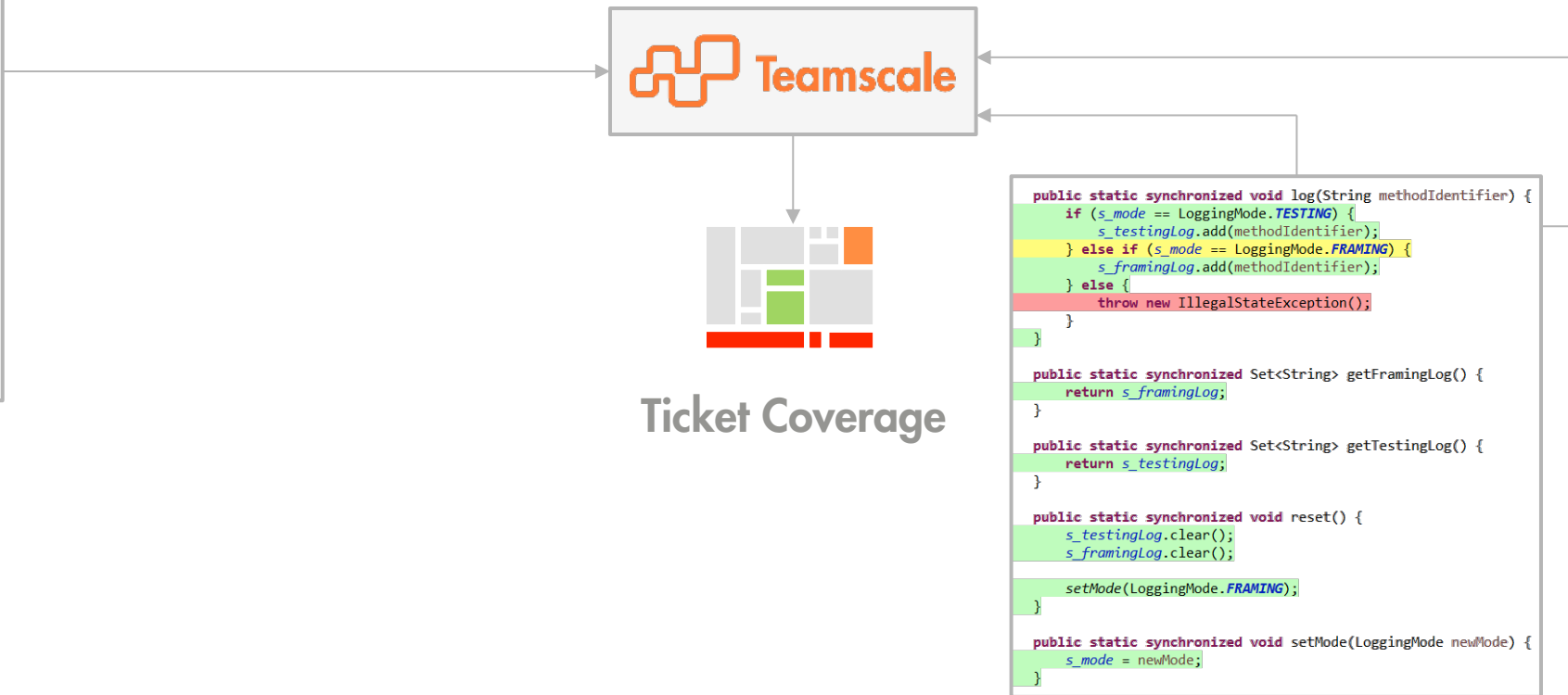
public static synchronized Set<String> getTestingLog() {
    return s_testingLog;
}

public static synchronized void reset() {
    s_testingLog.clear();
    s_framingLog.clear();

    setMode(LoggingMode.FRAMING);
}

public static synchronized void setMode(LoggingMode newMode) {
    s_mode = newMode;
}

```



ABAP

Ada

C#

C/C++

Cobol

Delphi

Fortran

Groovy

Gosu

HANA SQLScript

HANA Views

IEC 61131-3 ST

Java

JavaScript

Magik

Matlab

Open CL

OScript

PHP

PL/SQL

Python

Rust

SQLScript

Simulink/StateFlow

Swift

TypeScript

Visual Basic .NET

Xtend

Branch: master

Timetravel: Current Dashboard



Feb 01 2015 00

Pinta

Search, e.g. files...

Issue details:

1438022: Improve how selection changes are detected. (Closed)



Created by Cameron White on Mar 17 2015 07:35



Assigned to Cameron White

[Open in bugtracker](#)

Last updated on Jul 26 2015 08:35

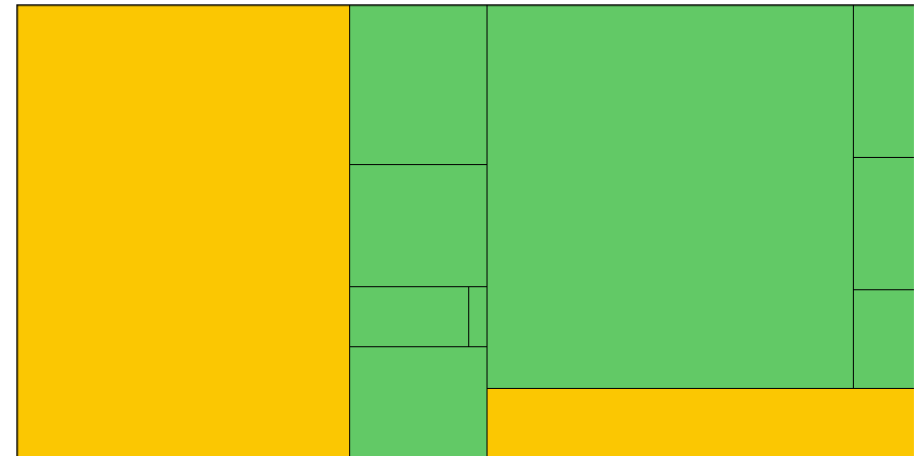
Selection changes are currently not handled properly.

Affected files (6)

Path	Change type (aggregated)
Pinta.Core/Classes/Document.cs	changed
Pinta.Core/Classes/DocumentSelection.cs	changed
Pinta.Core/Classes/SelectionModeHandler.cs	changed
Pinta.Core/HistoryItems/SelectionHistoryItem.cs	changed
Pinta.Core/Managers/ActionManager.cs	changed
Pinta.Core/Managers/WorkspaceManager.cs	changed

Test Gap Treemap

Jul 26 2015 04:23 - Now | Test Gap: 18.18%



Associated repository changes (1):



Fix #1438022 - Improve how selection changes are detected. by Cameron White in revision 3f4b4b105ee6c777d5ce999f664049aeb0f3f002 in branch master (GitHub)

Jul 26 2015 04:23

Files: 6 changed Findings: 6 10 1

Track 1 Requirements	Track 2 Testautomation	Track 3 Testdesign	Track 4 Testmanagement	Track 5 Special Topics
<p>11:40 - 12:25 Uhr</p> <p>Vortrag</p> <p>Security Requirements für Home Automation Systeme</p> <p>Prof. Dr. Hartmut Pohl softScheck GmbH</p>	<p>11:40 - 12:25 Uhr</p> <p>Vortrag</p> <p>Testen und Testautomatisierung von Wearables</p> <p>D.Dahnelt, P. Steensen, Hochschule Flensburg Prof. Dr. Milena Zachow, Fachhochschule Lübeck</p>	<p>11:40 - 12:25 Uhr</p> <p>Vortrag</p> <p>Die vier Ebenen der Datenqualität – Teststrategien für Analytics Projekte</p> <p>Christian Alexander Graf Qualitätssicherung und Statistik</p>	<p>11:40 - 12:25 Uhr</p> <p>Vortrag</p> <p>Immer kürzere Testphasen? Mit Ticket Coverage verhindern, dass wichtige Features ungetestet bleiben</p> <p>Dr. Elmar Jürgens CQSE GmbH</p>	<p>11:40 - 12:25 Uhr</p> <p>Vortrag</p> <p>Sicherheitszertifizierung für die Digitale Transformation: Erfahrungsbericht zum Industrial Data Space</p> <p>Prof. Dr. Jan Jürjens Fraunhofer Inst. f. Software- u. Systemtechn. ISST</p>
Pause/Raumwechsel				
Solution-Workshop	Solution-Workshop	Solution-Workshop	Solution-Workshop	Solution-Workshop
<p>12:35 - 13:20 Uhr</p> <p>IoT Talk</p> <p>Dr. Armin Metzger ASQF</p>	<p>12:35 - 13:20 Uhr</p> <p>Keyword-Driven Testing und Variantenmanagement</p> <p>Fabian Klaffke Klaros Testmanagement</p>	<p>12:35 - 13:20 Uhr</p> <p>Haben wir alle wichtigen Features getestet? Ticket-Coverage live am Beispiel</p> <p>Dr. Dennis Pagano CQSE GmbH</p>	<p>12:35 - 13:20 Uhr</p> <p>Test- und Quality-Reporting leicht gemacht</p> <p>Dierk Engelhardt imbus AG</p>	<p>12:35 - 13:20 Uhr</p> <p>KI Testing</p> <p>Special Interest Group</p>

Fazit

Kürzere Release-Zyklen führen zu Parallelisierung in Entwicklung und Test. Dadurch steigt das Risiko ungetesteter Änderungen.

Ticket Coverage hilft, wichtige Änderungen zuverlässig zu testen.

Wir unterstützen gerne bei der Evaluierung und Einführung.

Kontakt



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

Dr. Dennis Pagano · pagano@cqse.eu · +49 159 04062957

CQSE GmbH
Lichtenbergstraße 8
85748 Garching bei München
www.cqse.eu

