

# Software Teams Debuggen

Dr. Elmar Jürgens

# Über Mich

## Forschung

- Clone Detection, Architekturanalyse, ...
- PC Mitglied von MSR, ICPC, ICSE, ...



## Beratung

- Gründer
- Qualitäts-Bewertung & Qualitäts-Controlling



## Gesellschaft für Informatik

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



[www.teamscale.io](http://www.teamscale.io)

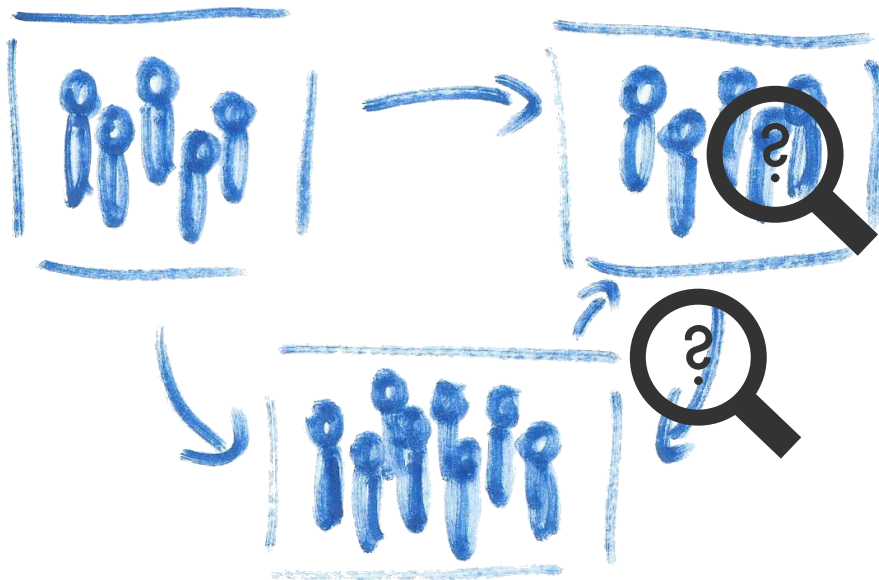


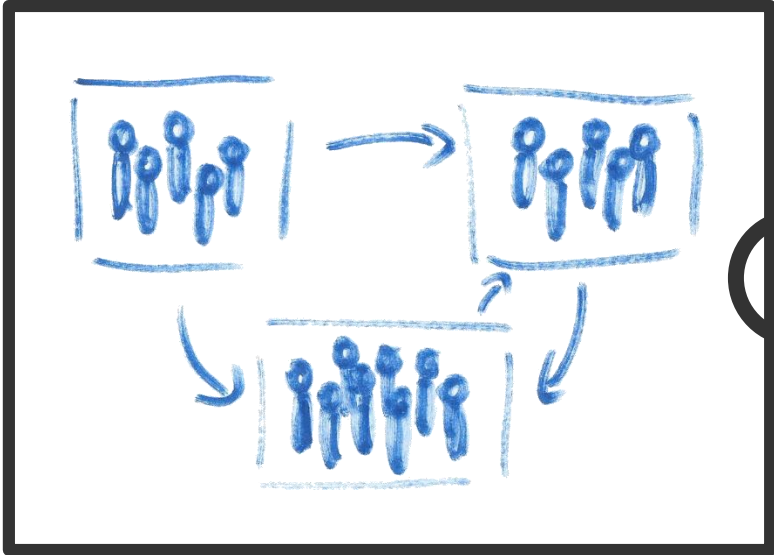
# Conway's „Law“

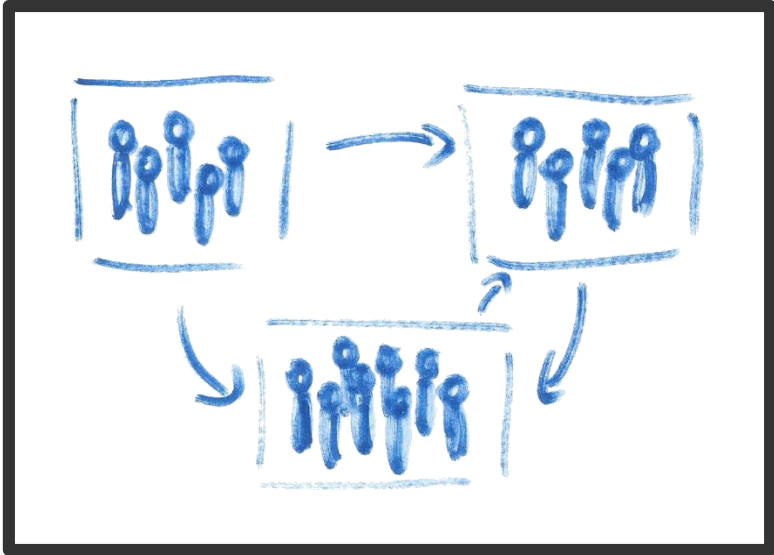
Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations.

Melvin Conway, 1968

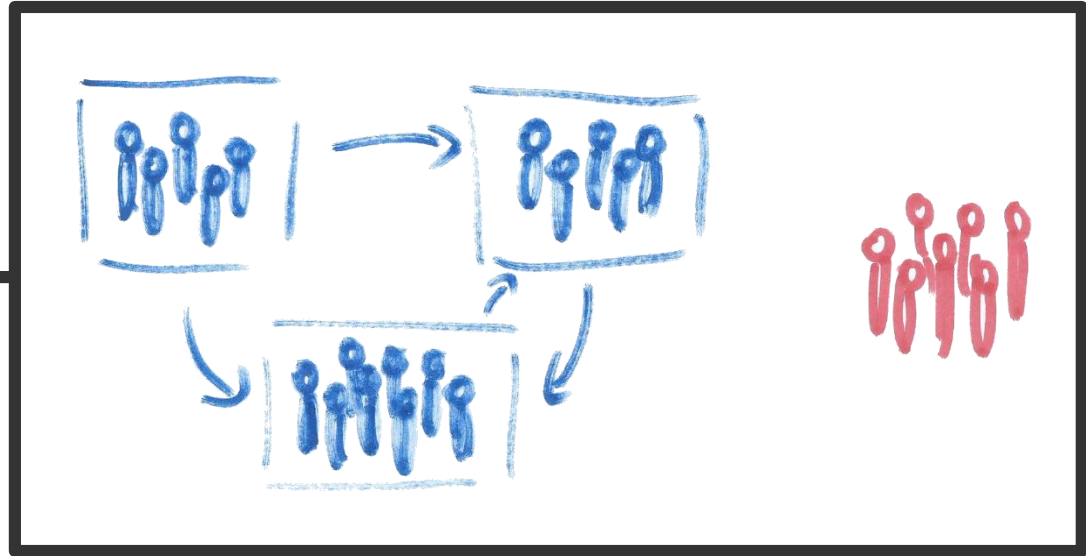
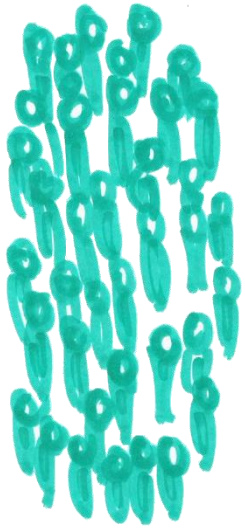


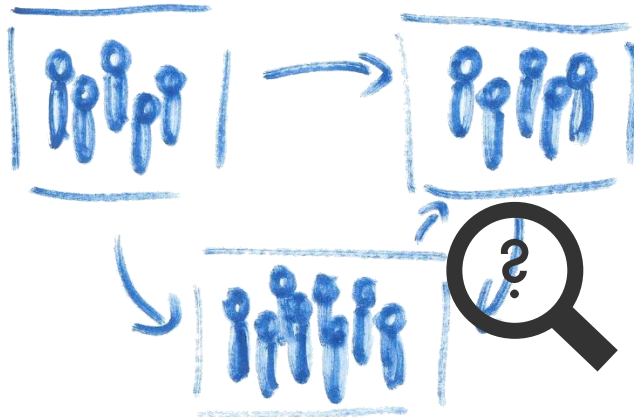
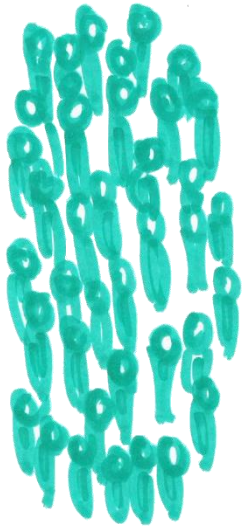




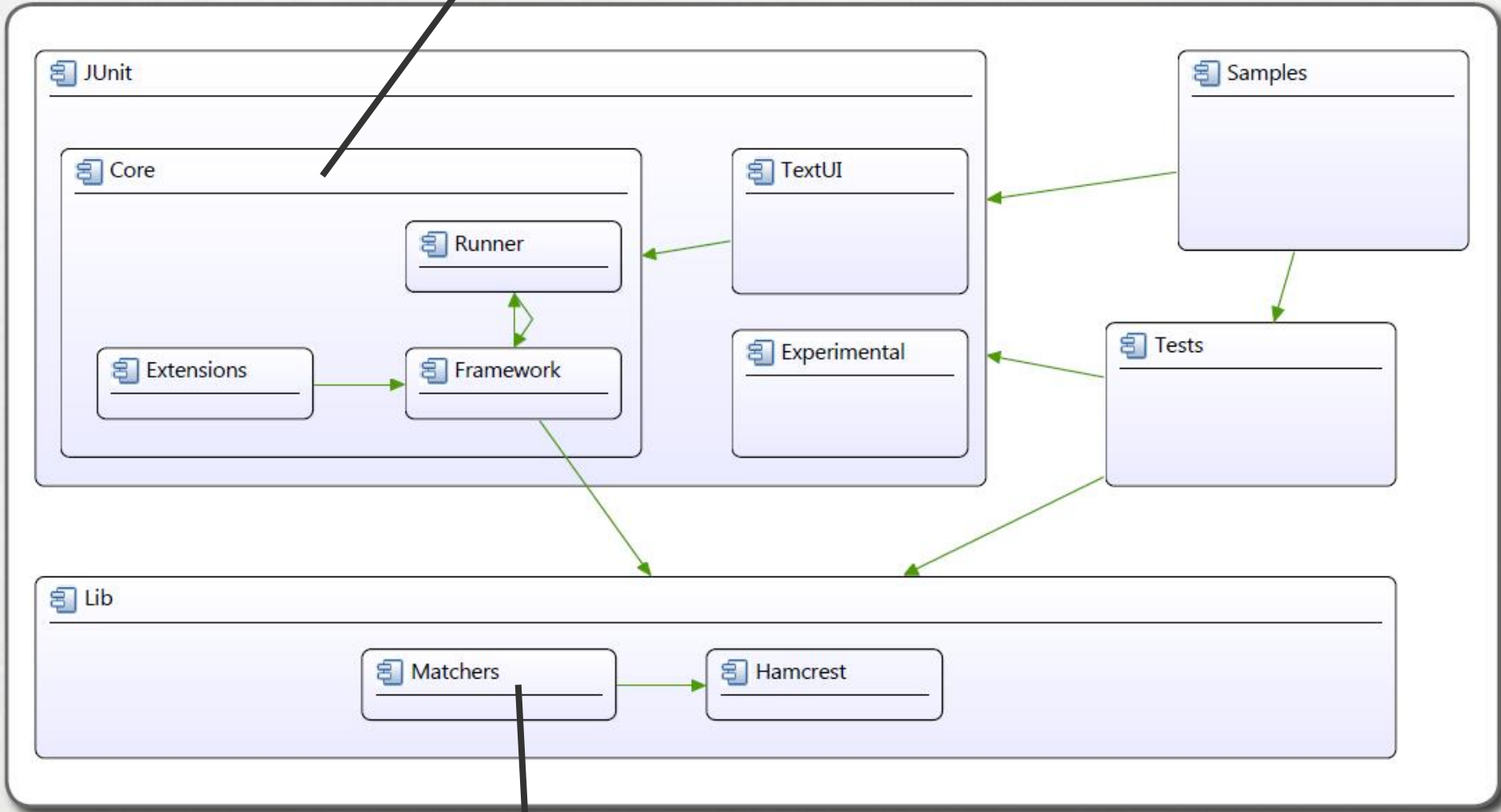








org.junit.core.\*



org.junit.matchers.\*

JUnit

Samples

Core

TextUI

Runner

Extensions

Framework

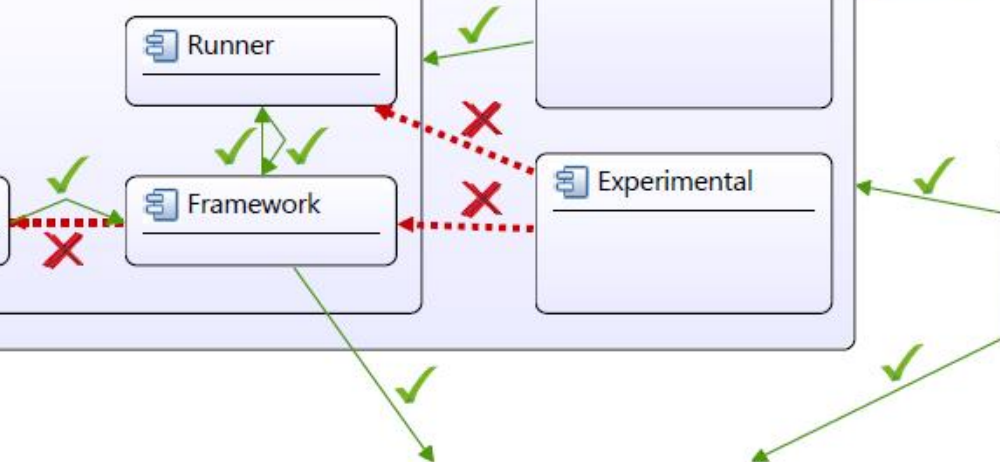
Experimental

Tests

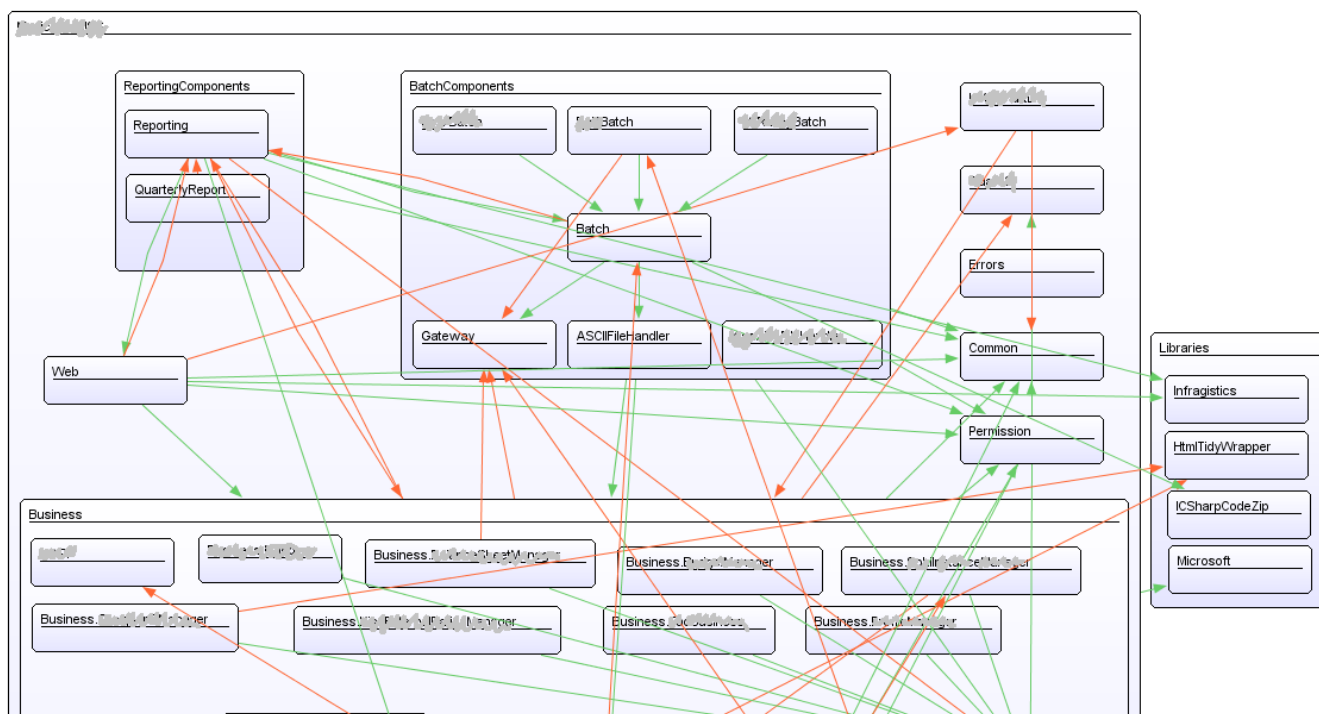
Lib

Matchers

Hamcrest





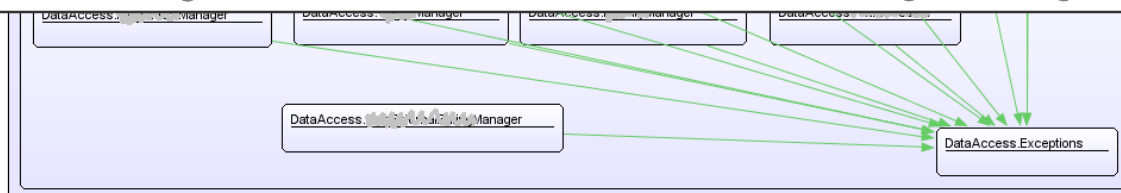


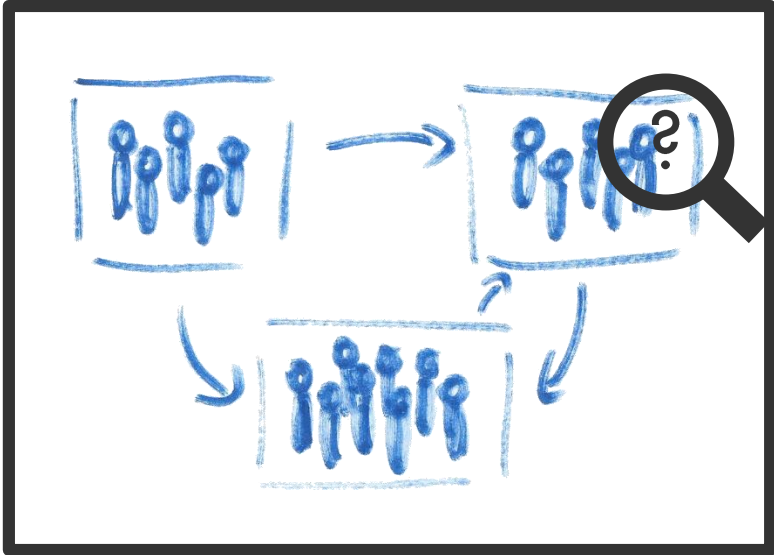
## Studie

- Auslassungen in Dokumentation
- Aufdeckung von Fehlern
- Katalysator für Architekturdiskussionen

Munich Re

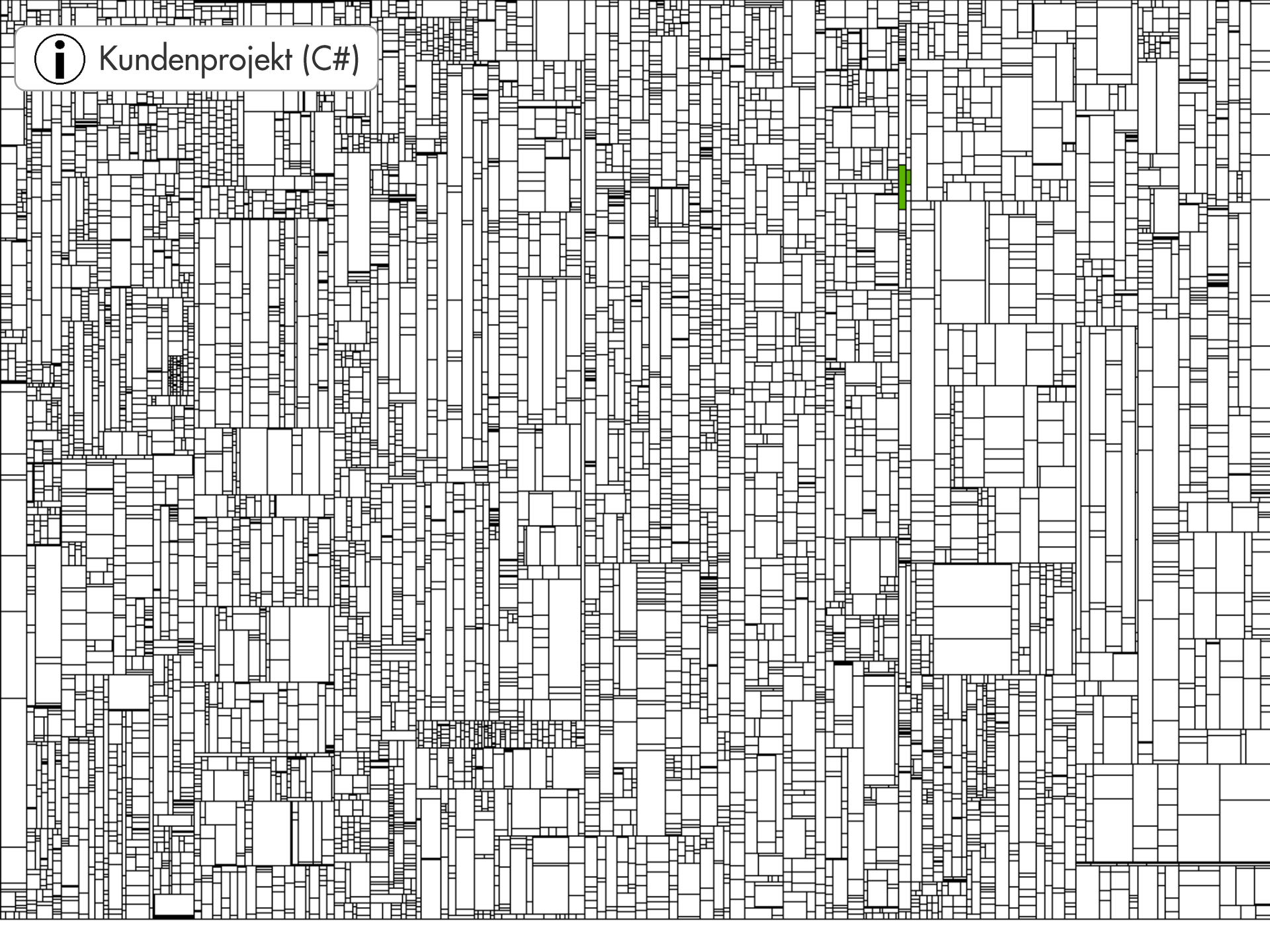
Feilkas, Juergens et al: *Loss of Architectural Knowledge During Evolution* ICPC 2009







Kundenprojekt (C#)







Entwickler

- feilkas
- stemplinger
- plachot
- sahinagic
- bader
- malinskyi
- ribeiro
- svejda
- hodale
- amann
- kanis
- junker
- steidl
- streitel
- beller
- hauptmab
- deissenb
- hummelb
- heineman
- poehlman
- juergens

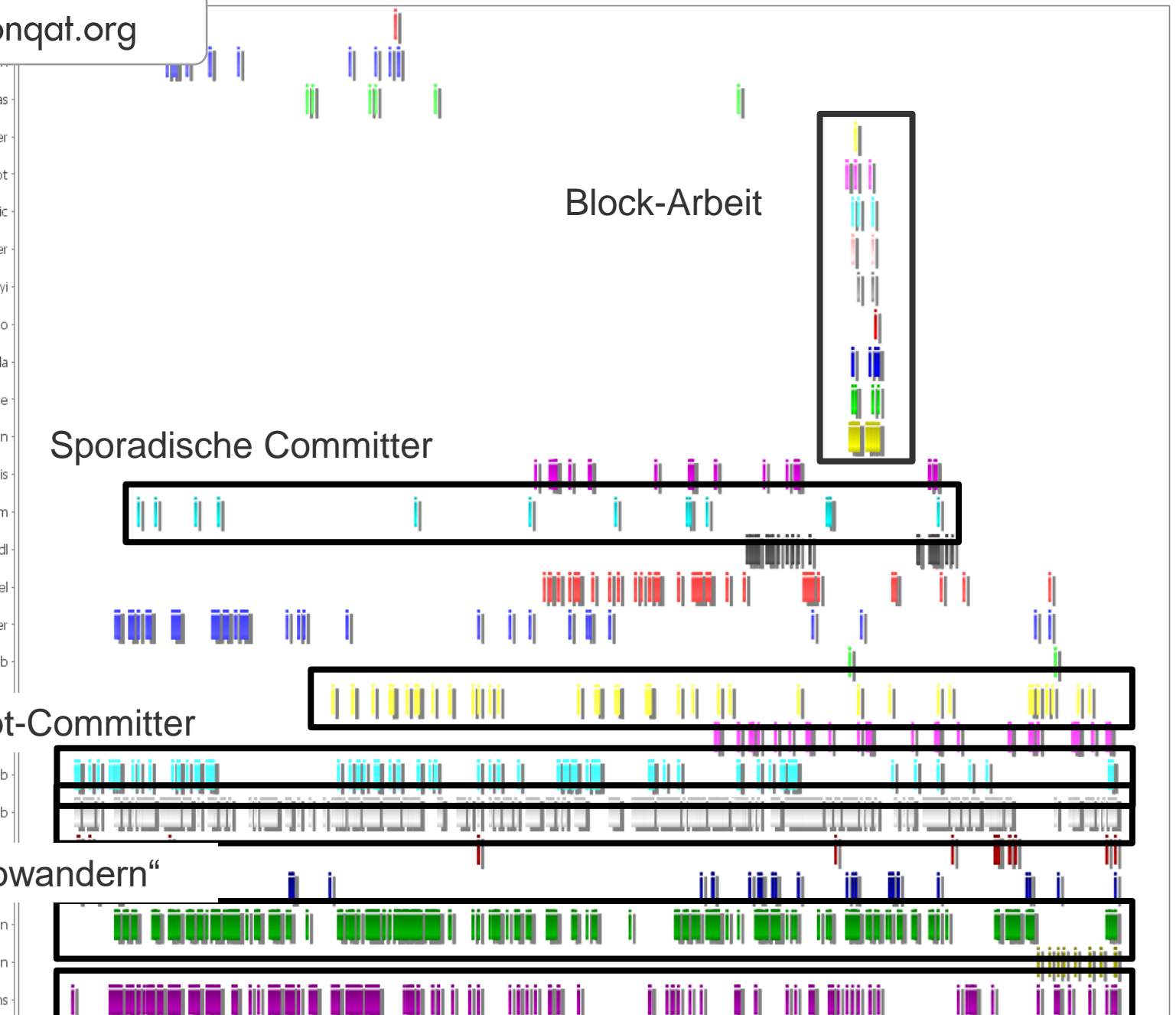
Sporadische Committer

Block-Arbeit

Haupt-Committer

„Abwandern“

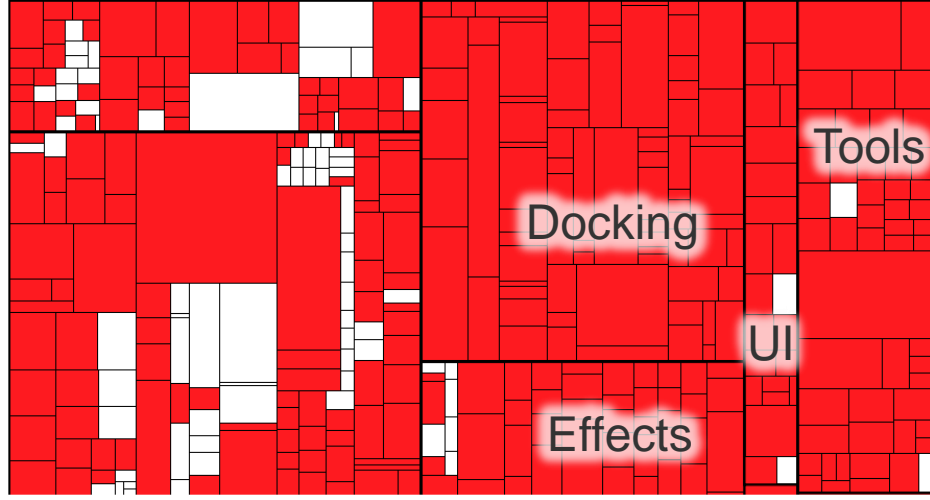
Jun-2011 Jul-2011 Aug-2011 Sep-2011 Okt-2011 Nov-2011 Dez-2011 Jan-2012 Feb-2012 Mrz-2012 Apr-2012 Mai-2012 Jun-2012 Jul-2012



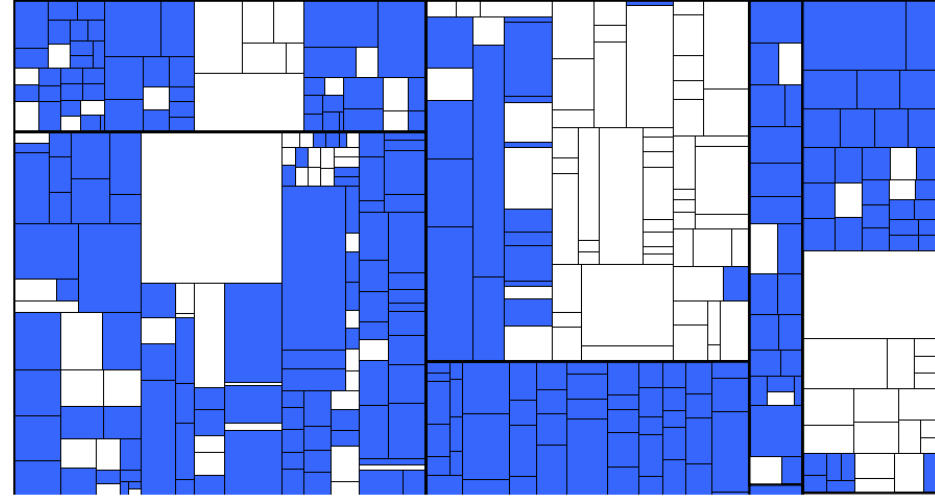




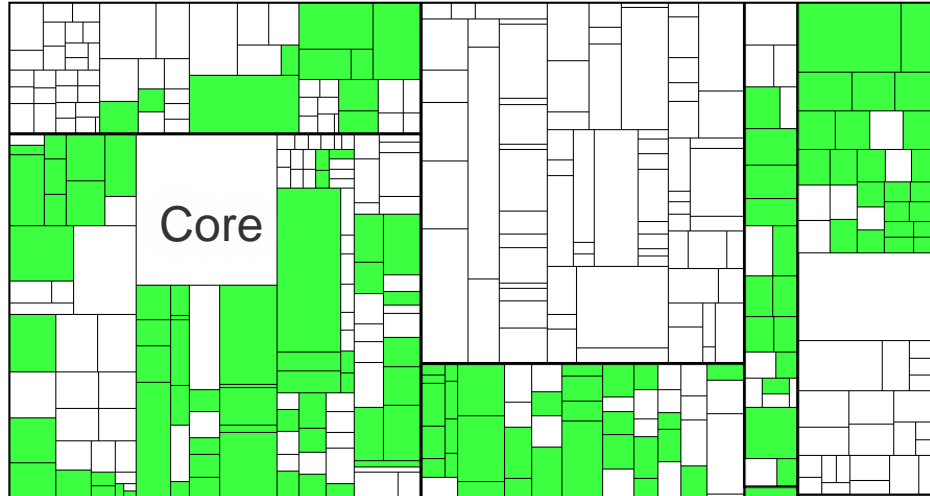
Ownership Cameron White (#1)



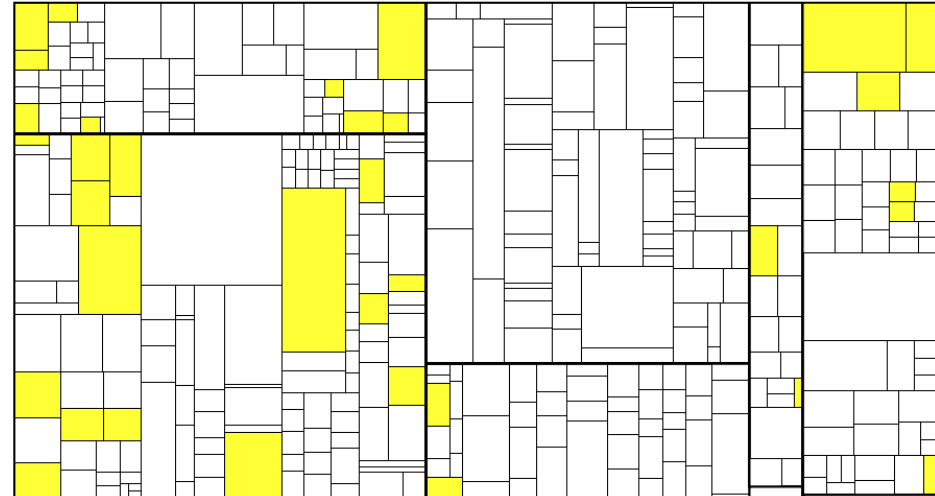
Ownership Jonathan Pobst (#2)

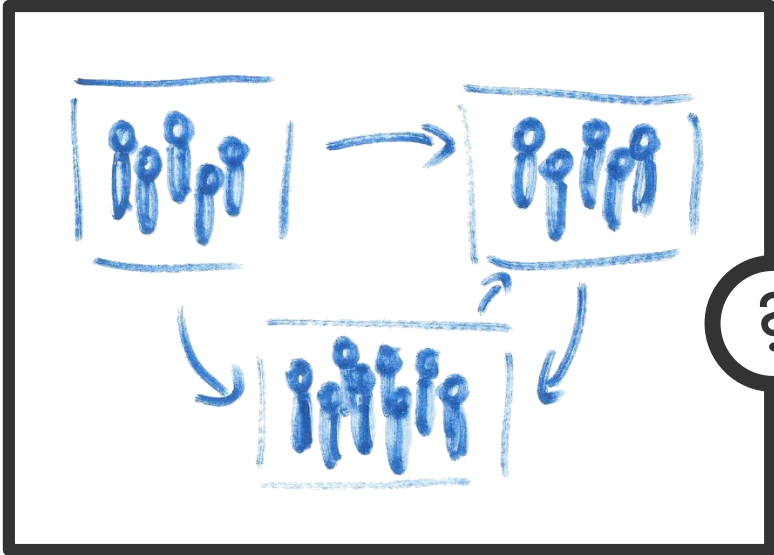


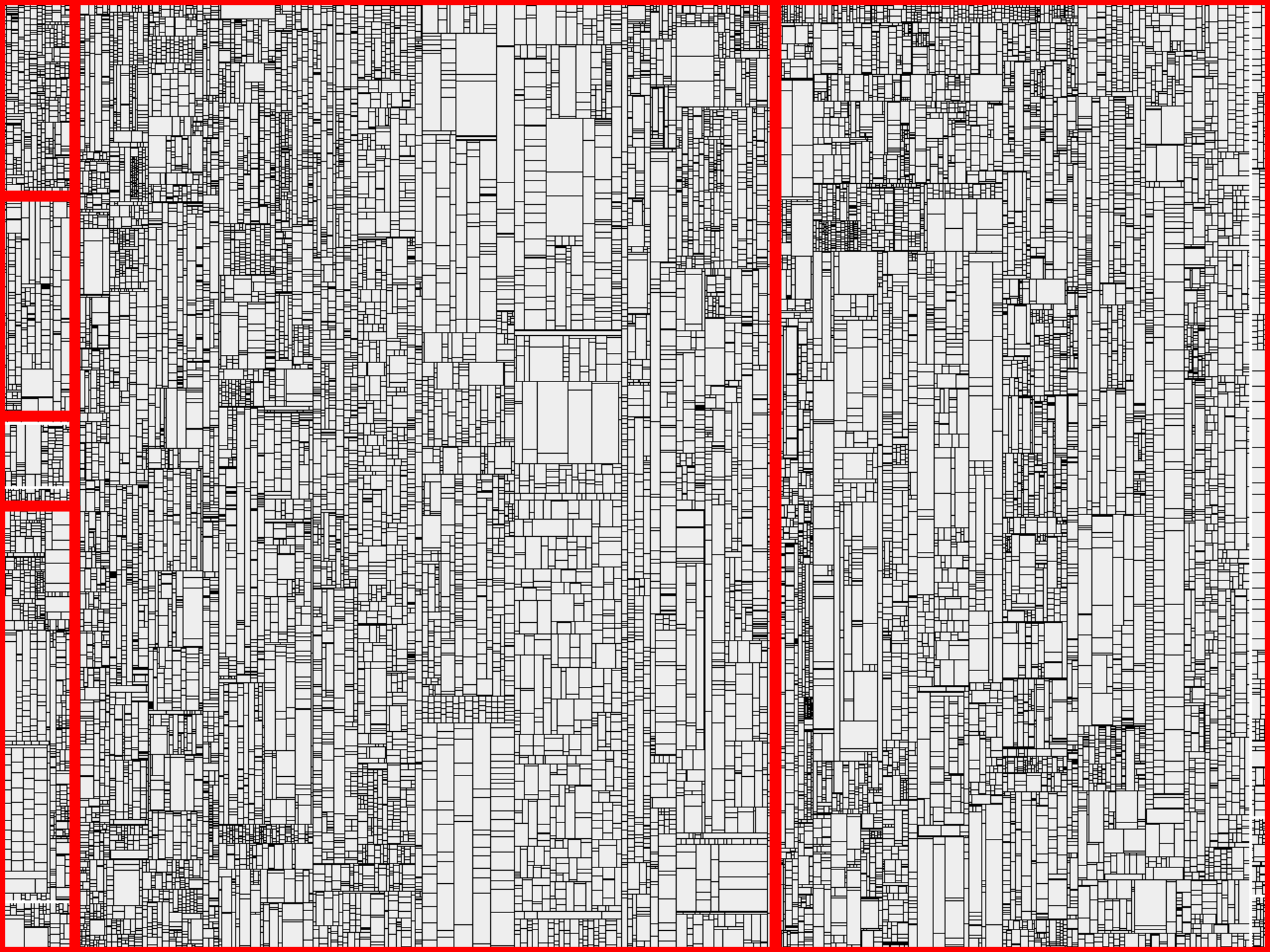
Ownership Olivier Dufour (#3)

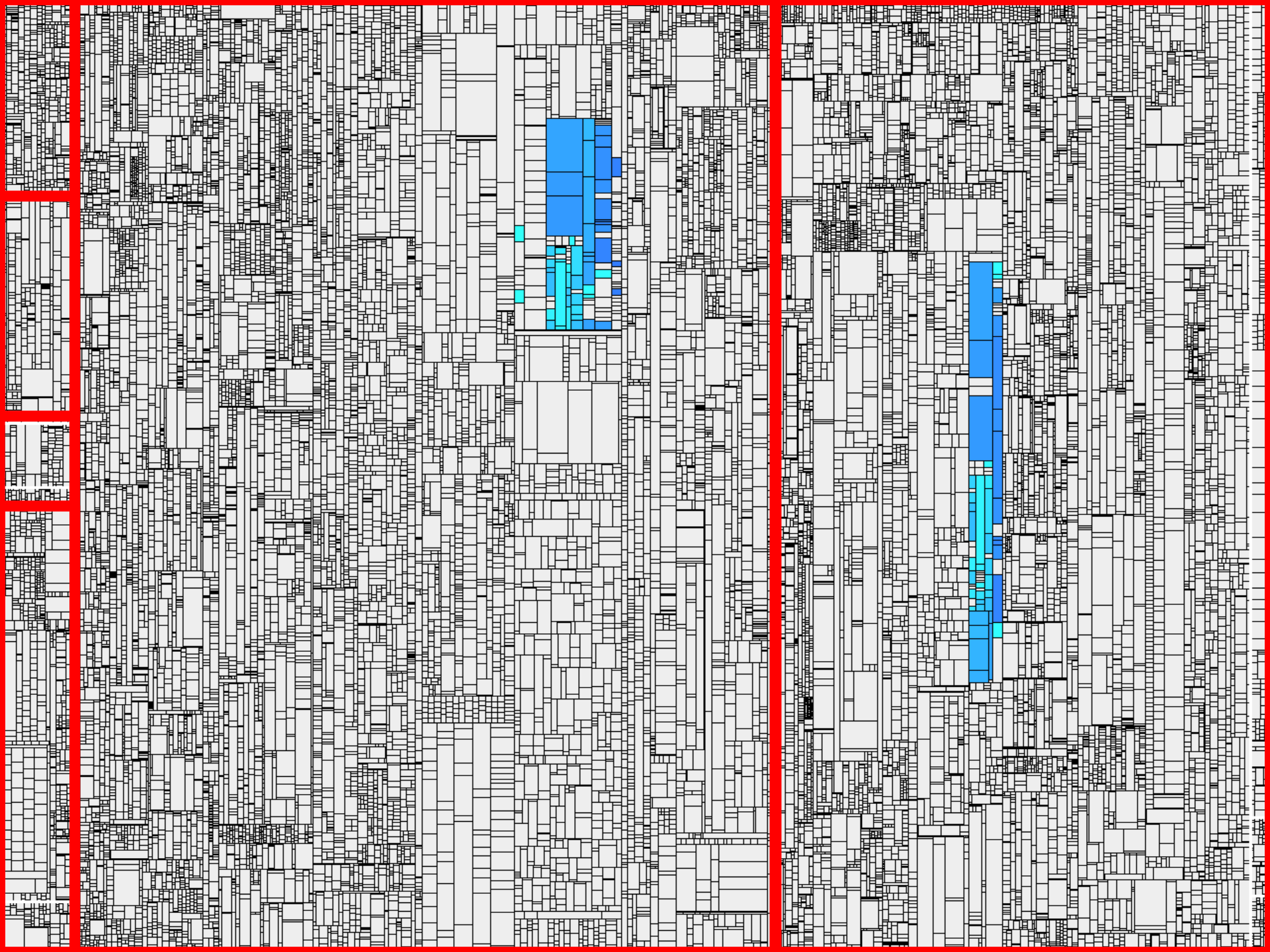


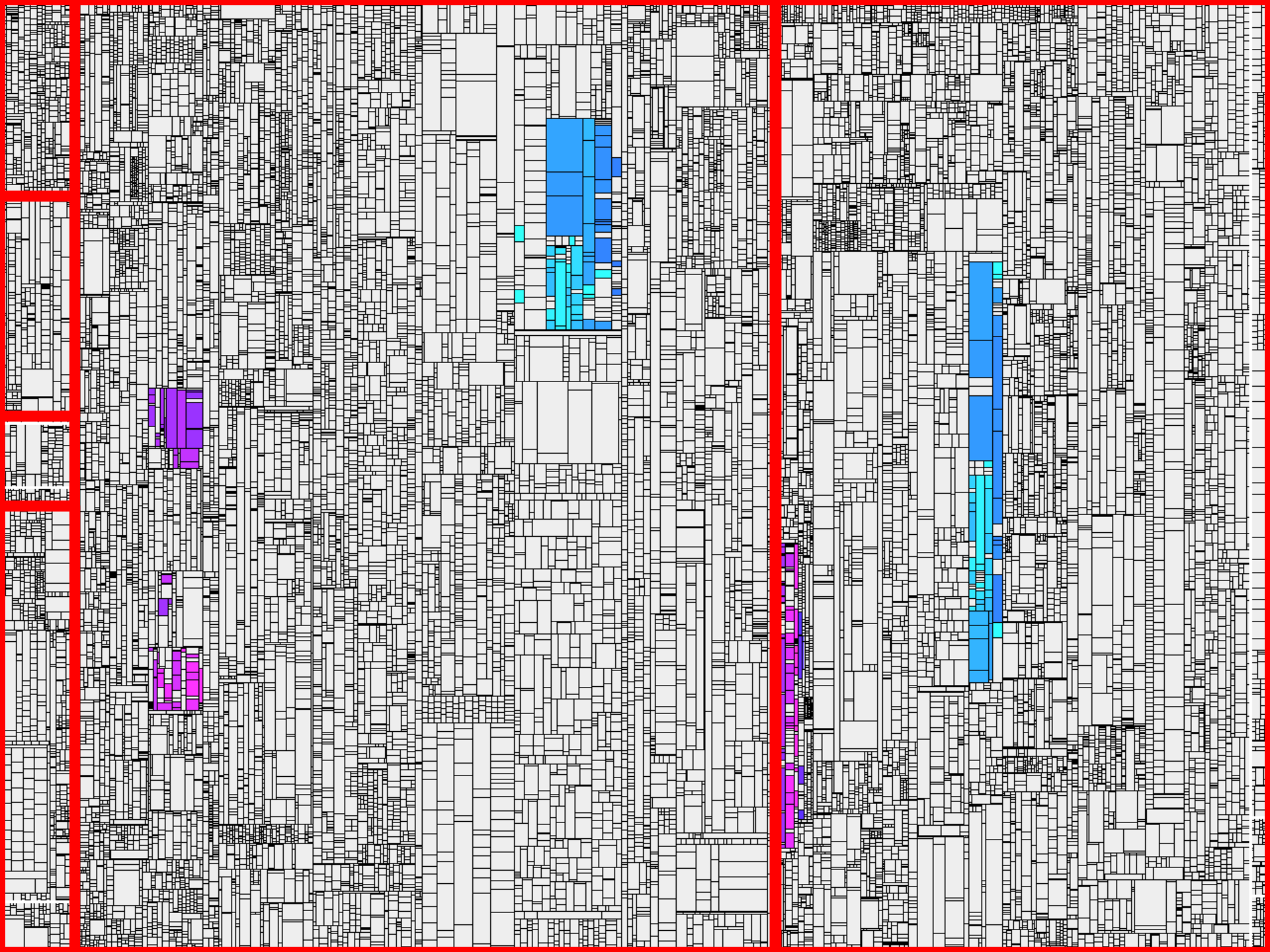
Ownership Robert Nordan (#4)





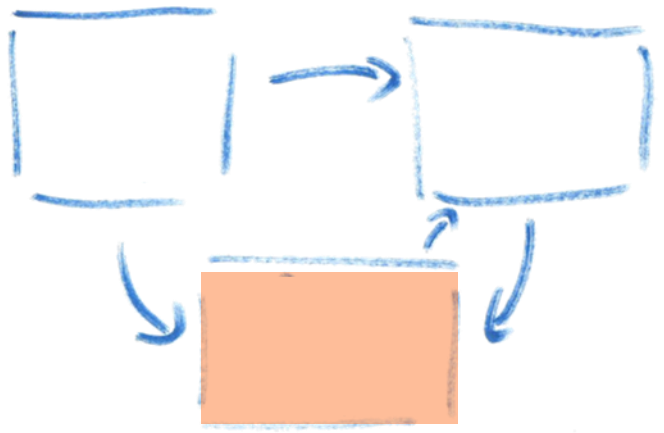
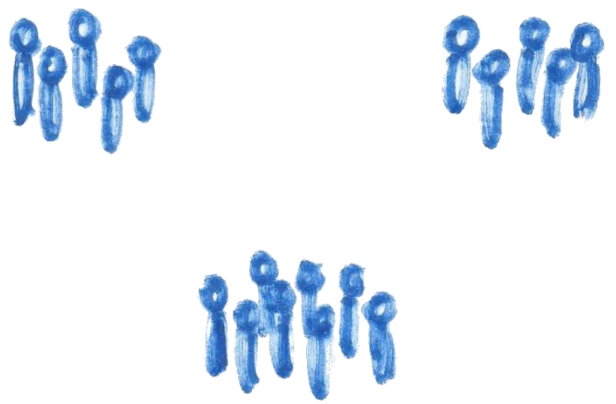










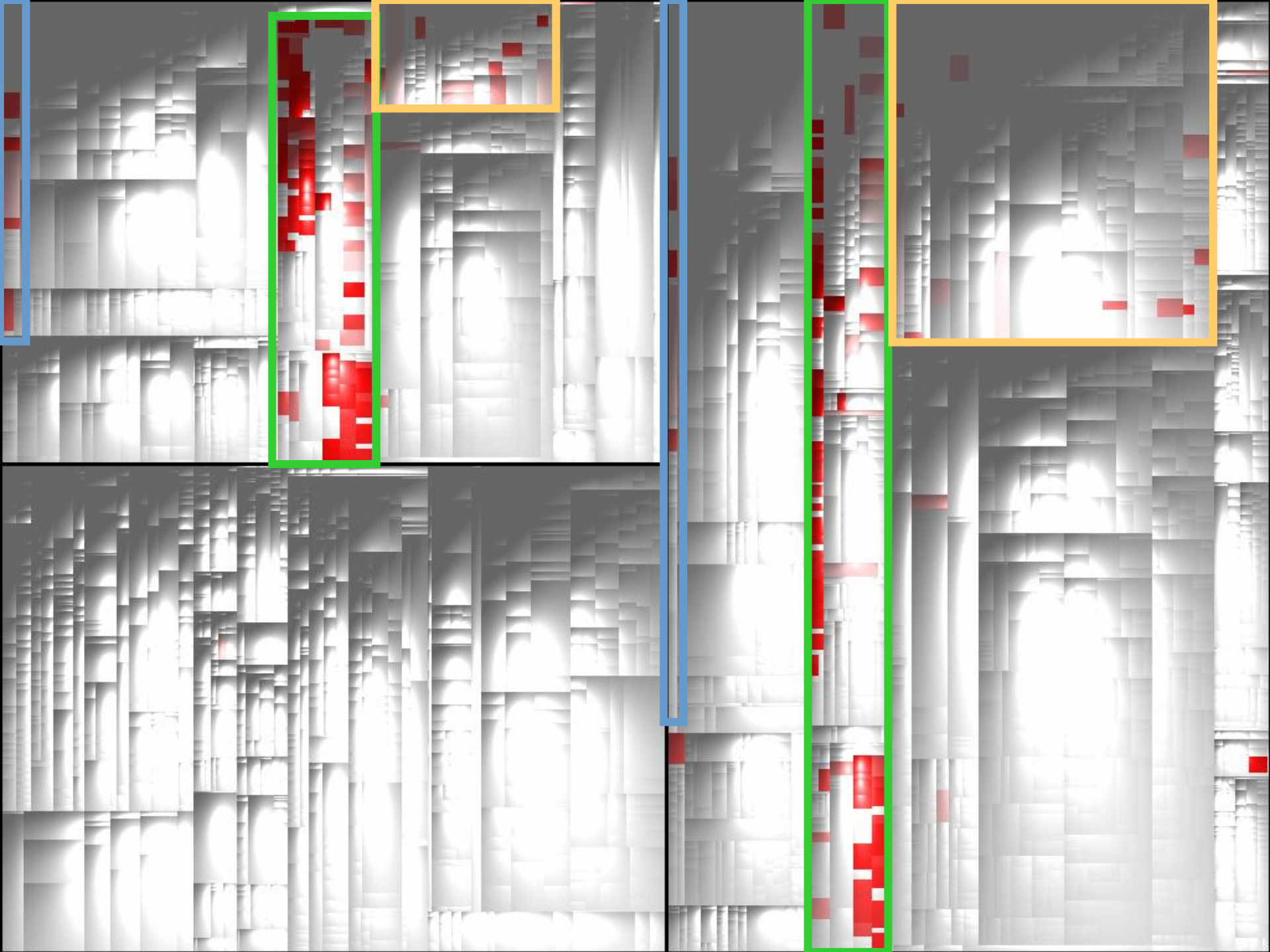


Mehrere Anwendungen  
in einer Abteilung

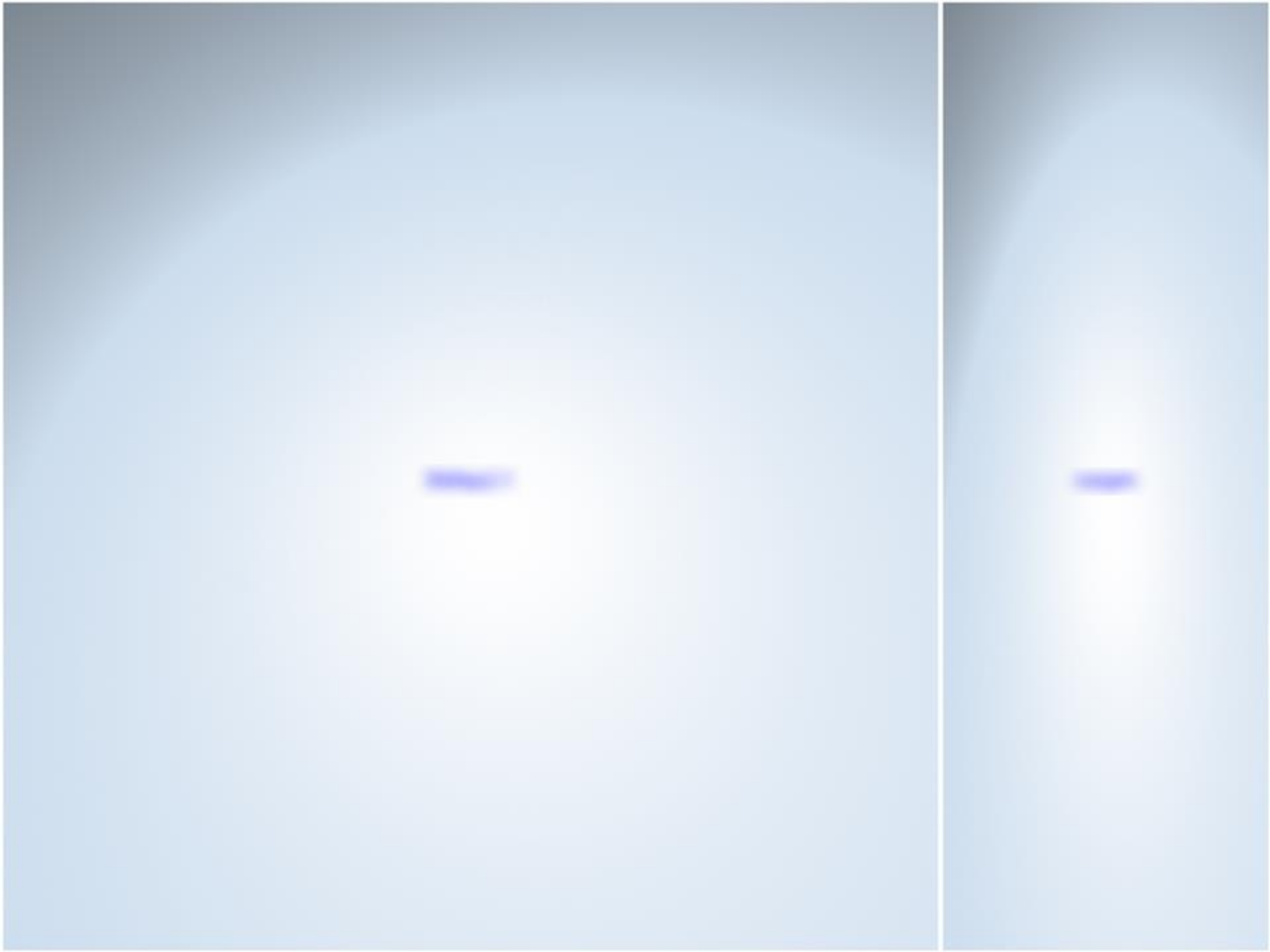
375 kloc

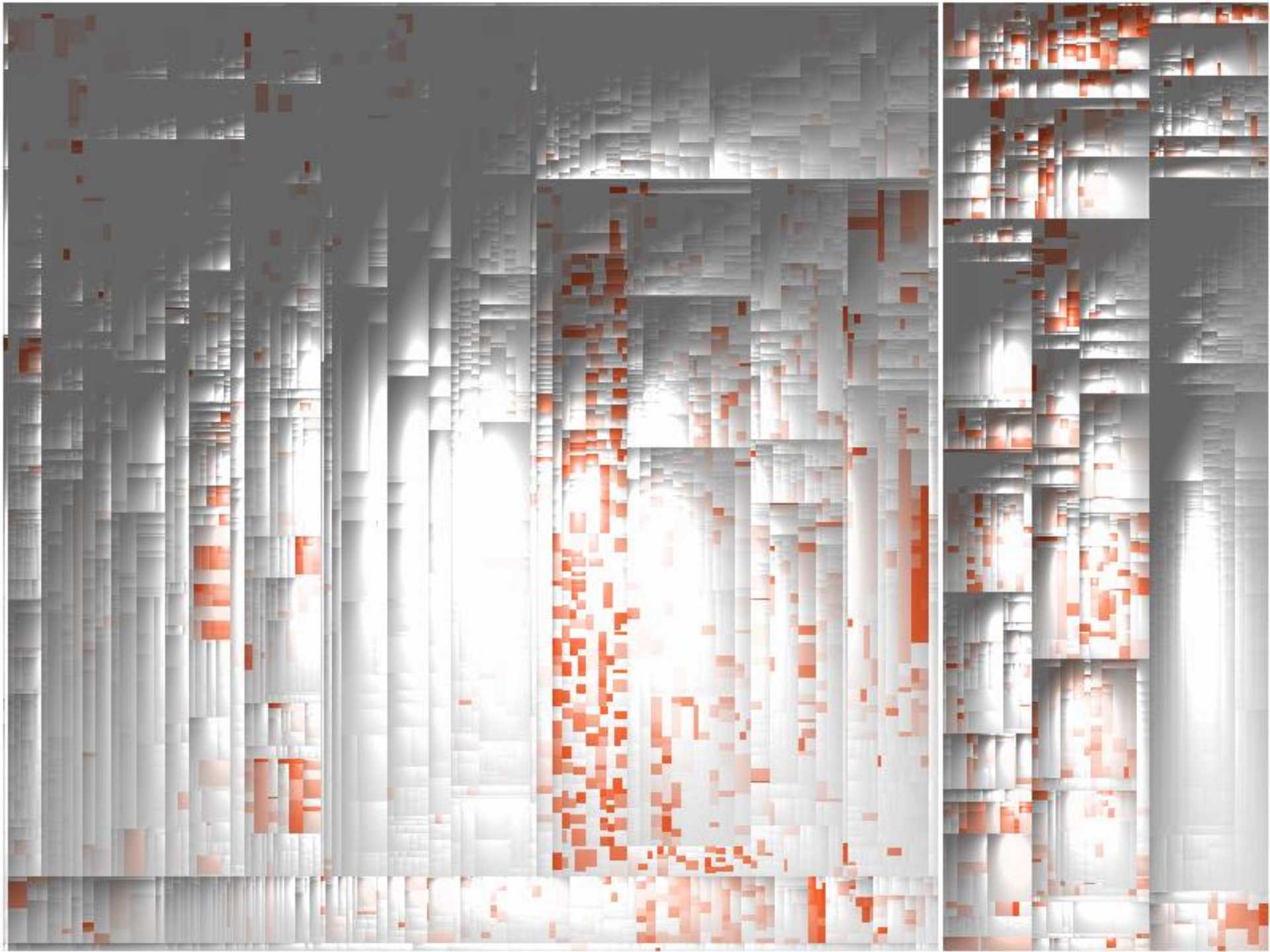
795 kloc

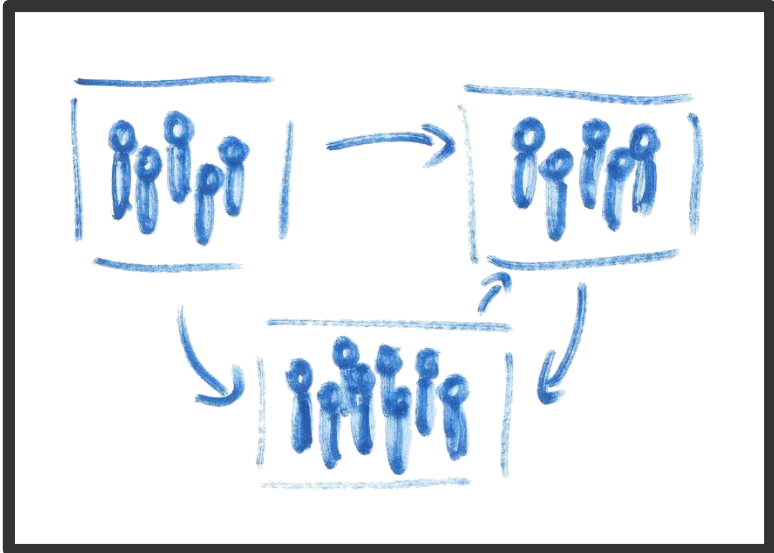
430 kloc



# Architektur einer Produktlinie

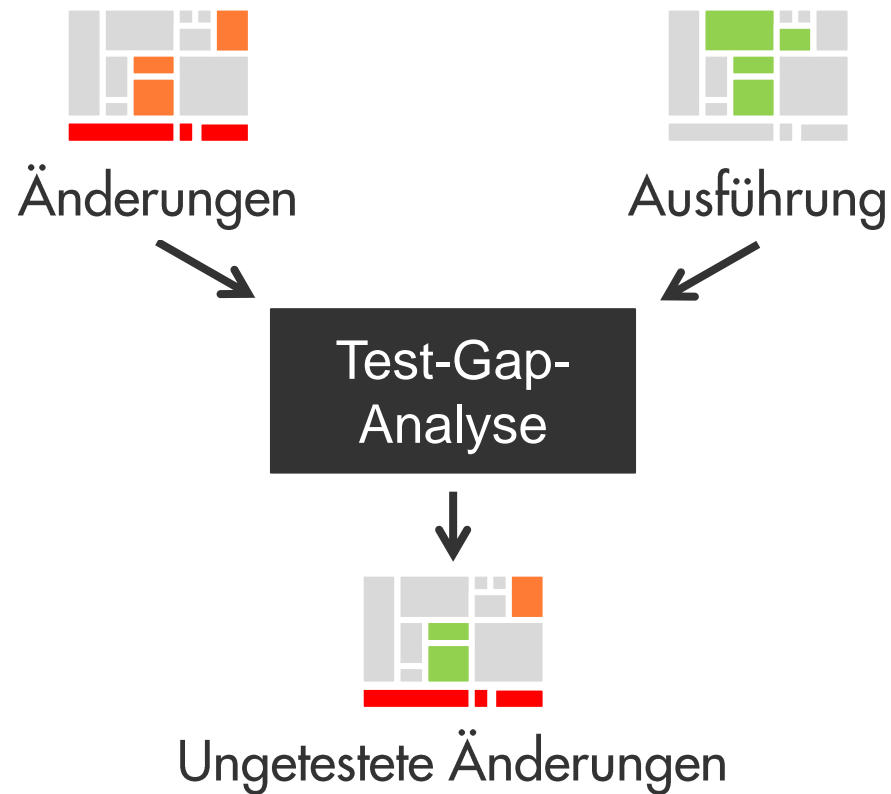








# Anatomie Test-Gap-Analyse





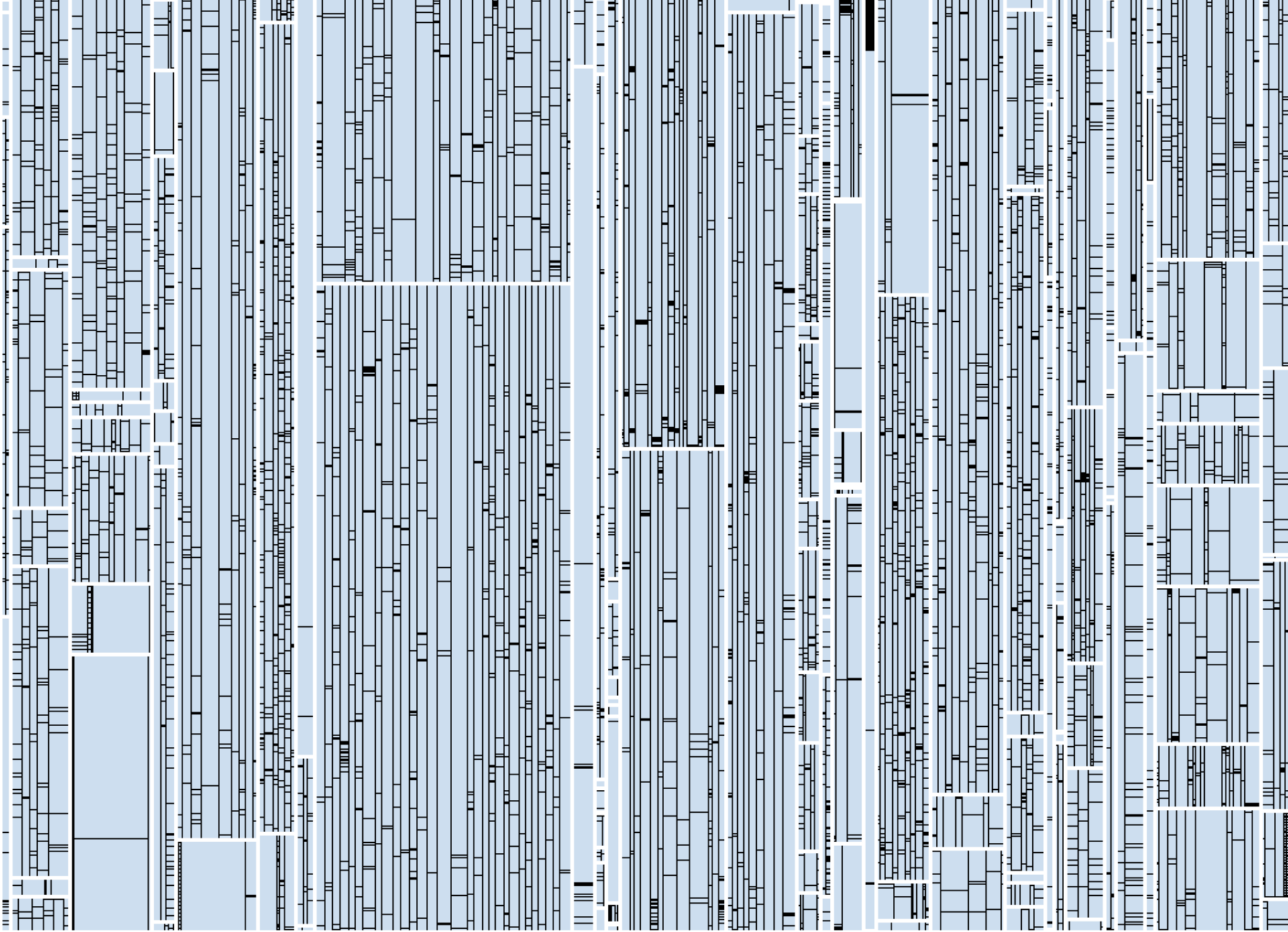
**GUI.Dialogs**

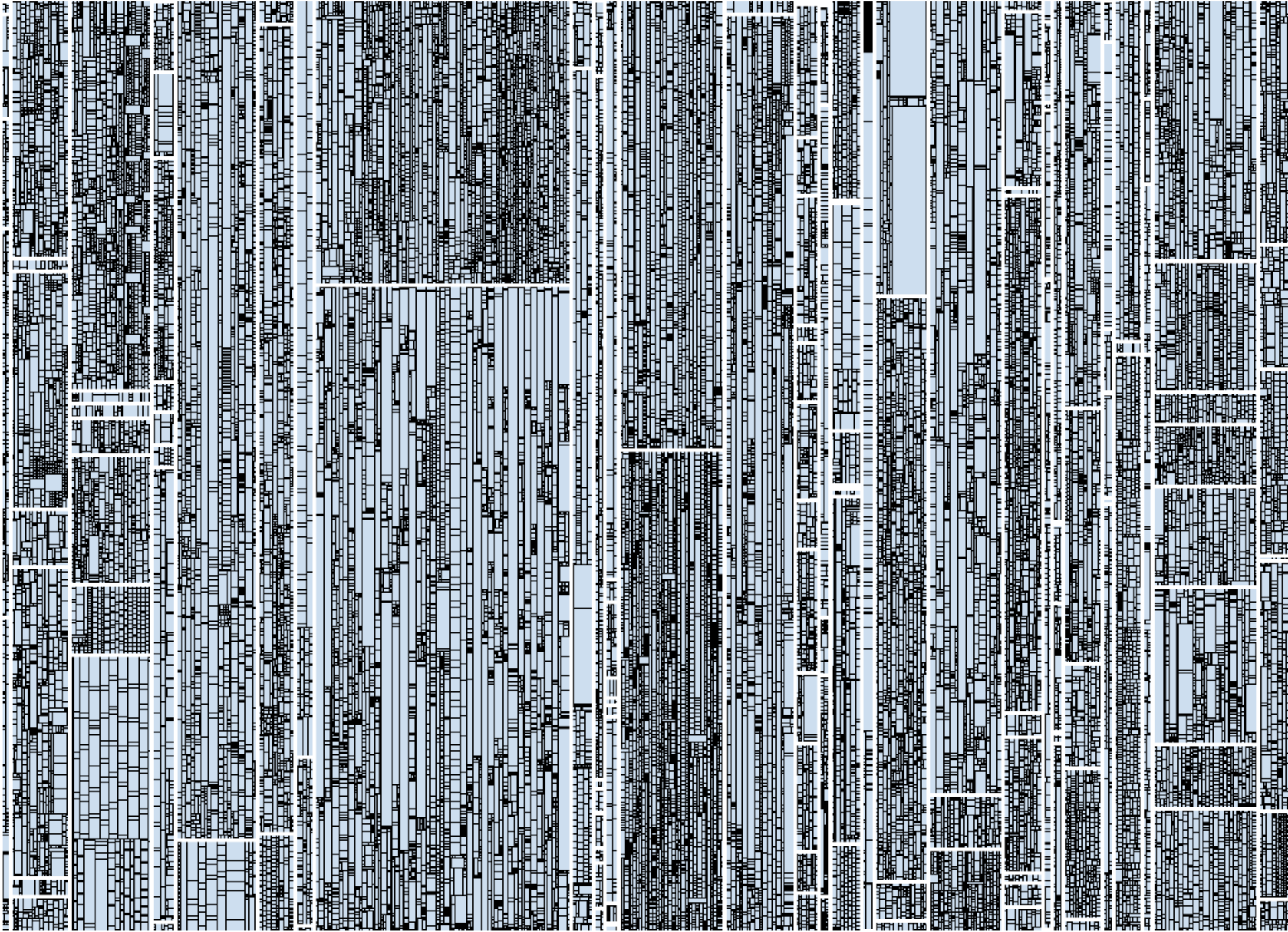
**Authentication**

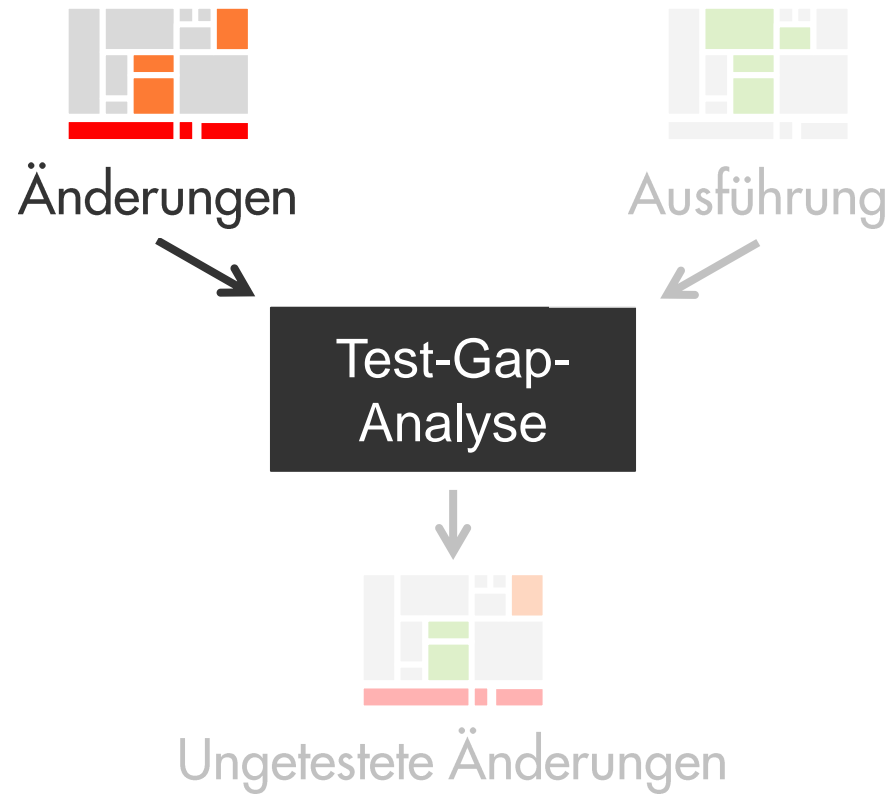
**UI Controls**

**GUI.Base**

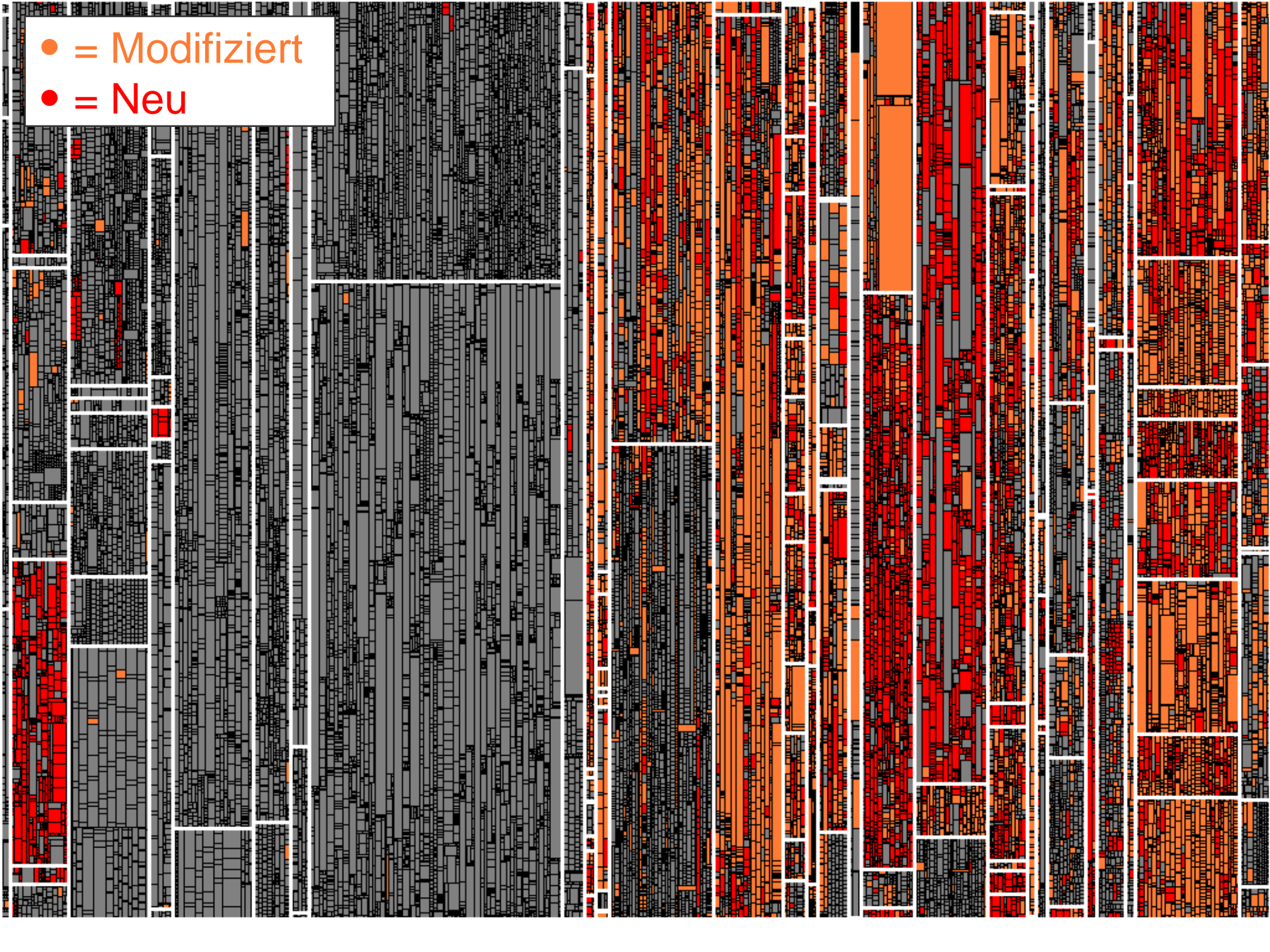
**Data  
Validation**

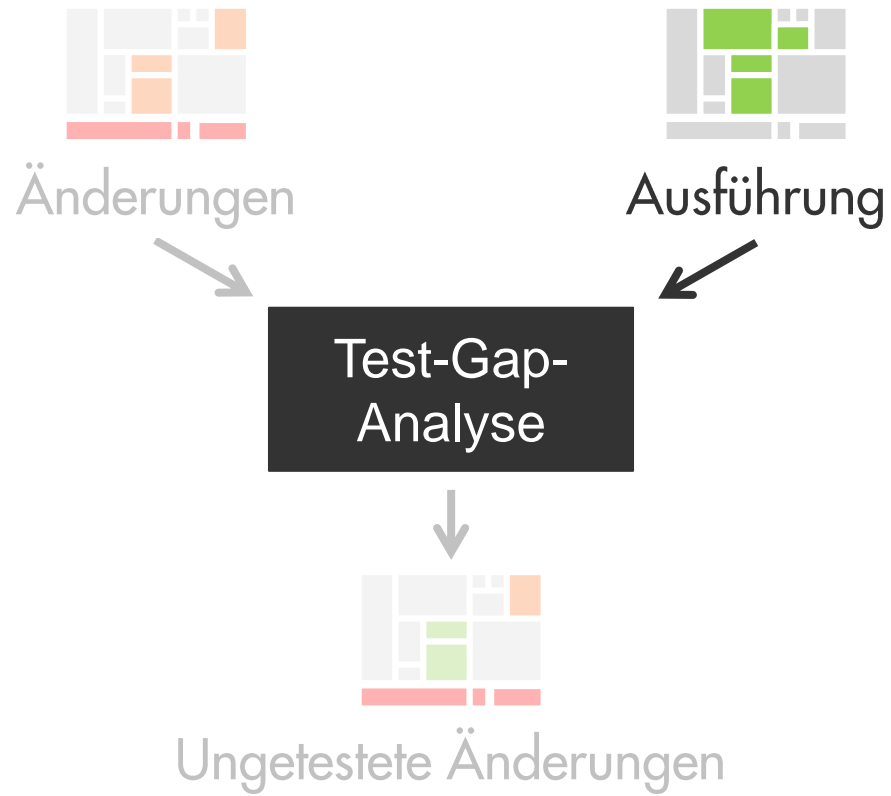






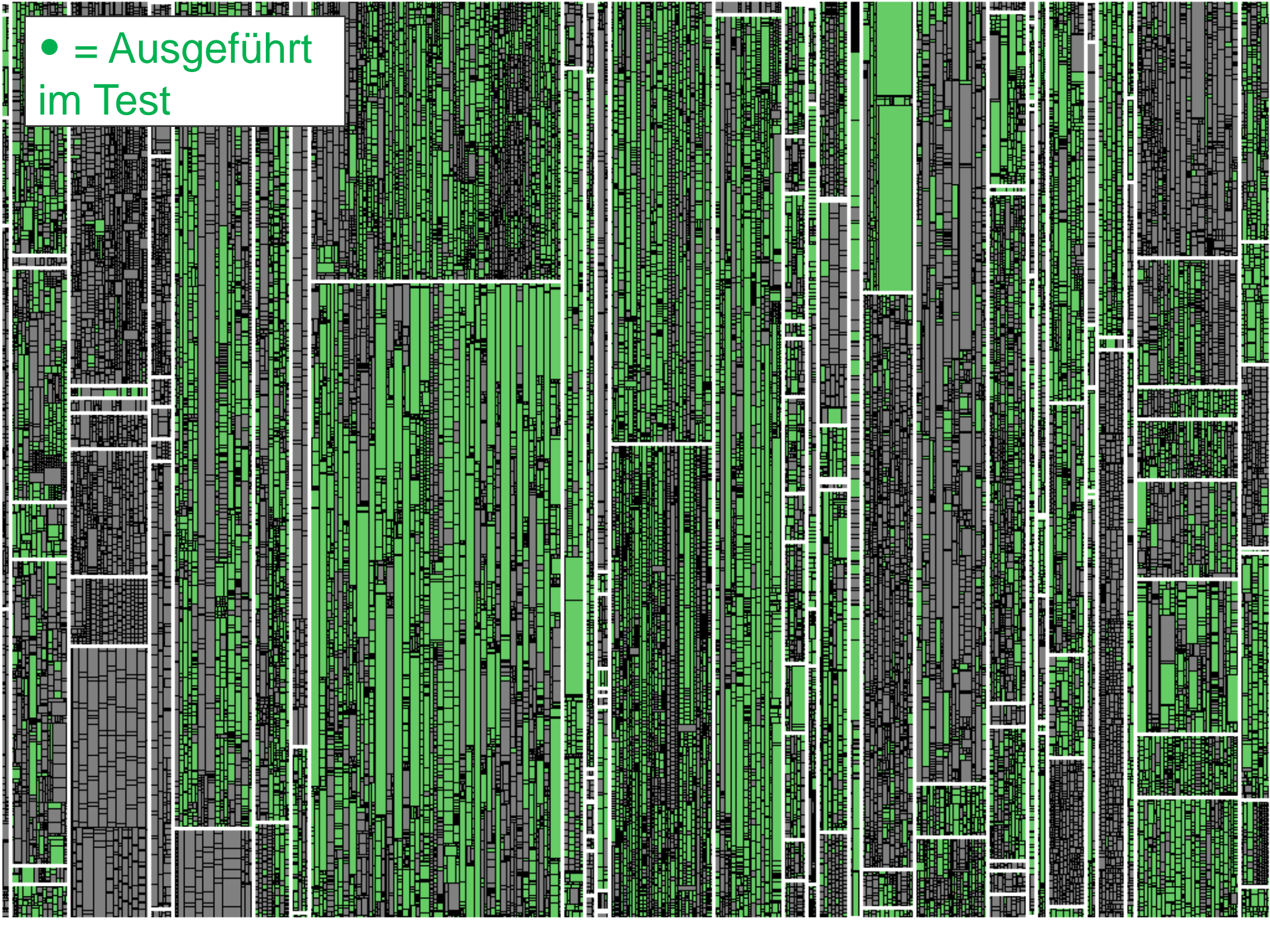
- = Modifiziert
- = Neu

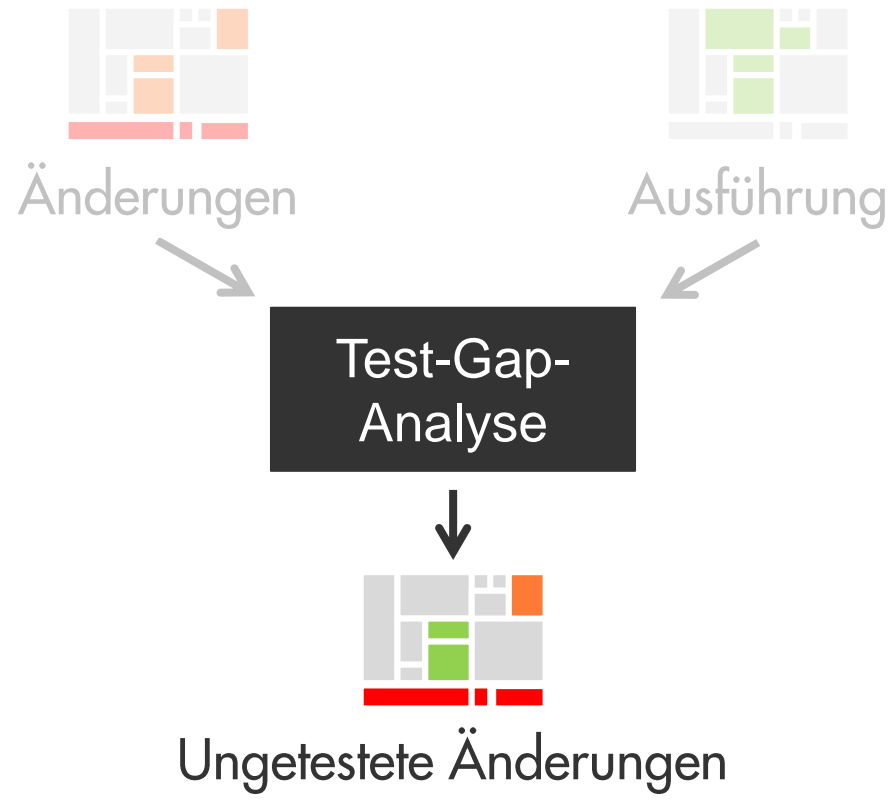




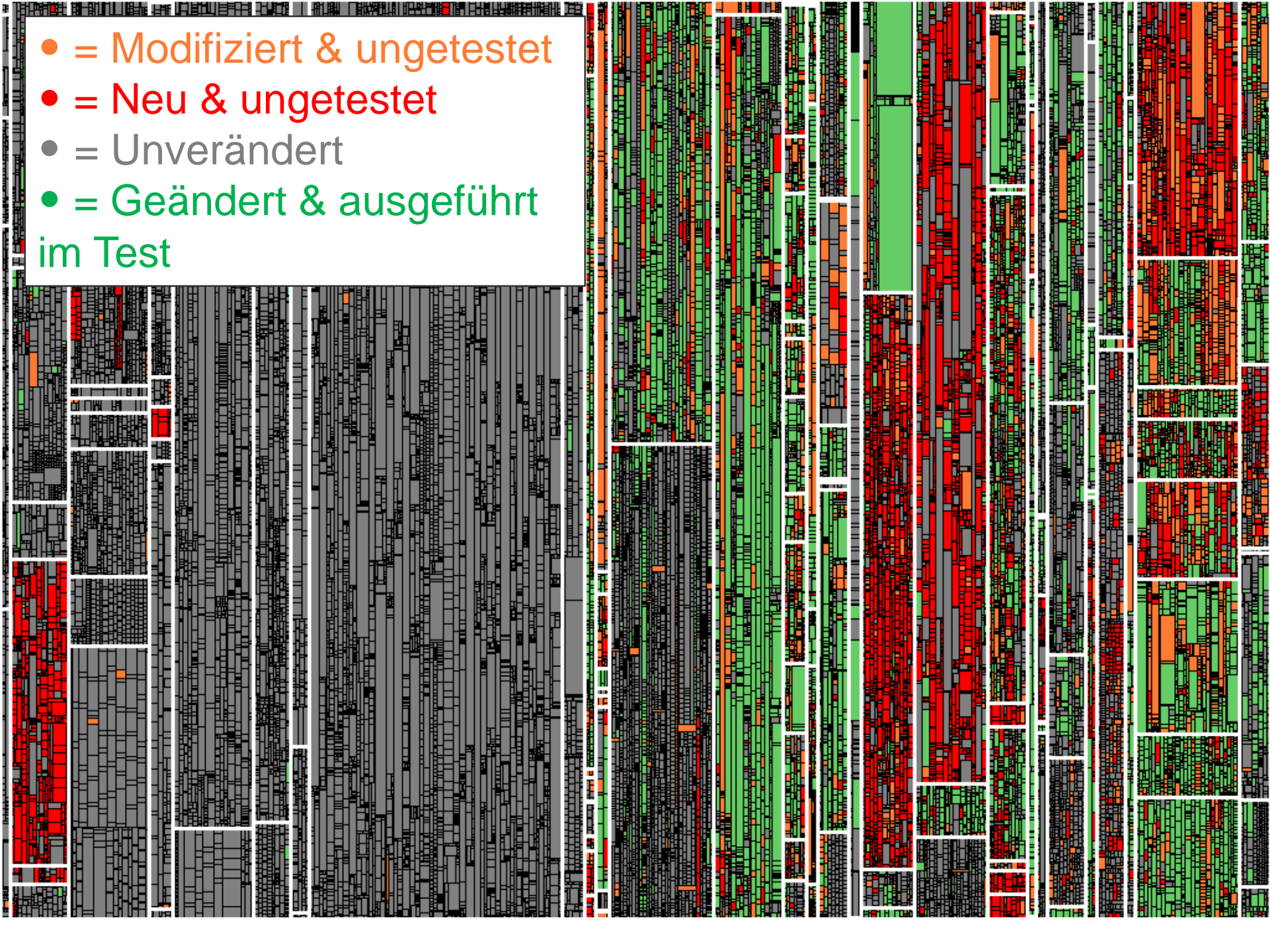


● = Ausgeführt  
im Test

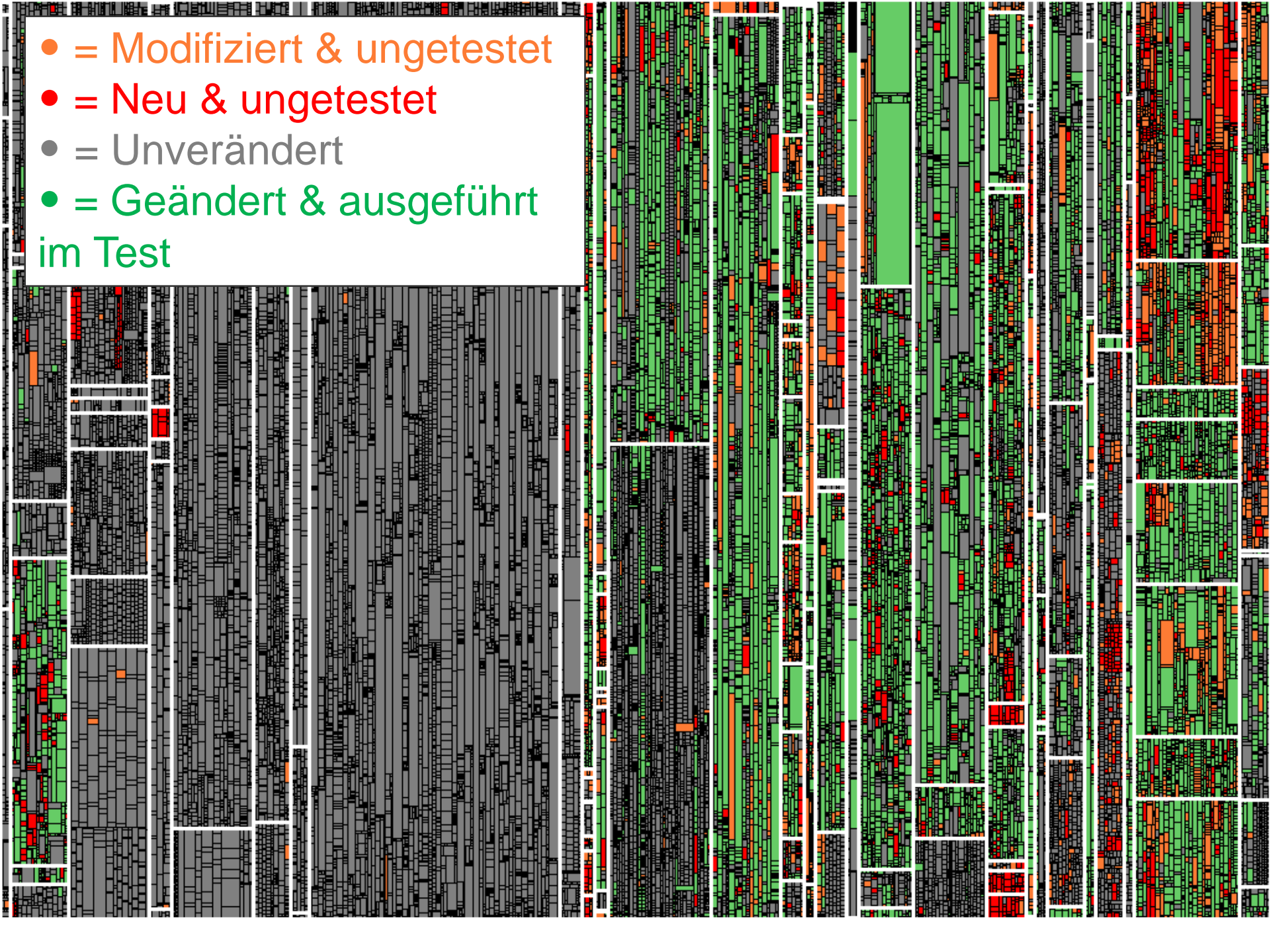




- = Modifiziert & ungetestet
- = Neu & ungetestet
- = Unverändert
- = Geändert & ausgeführt im Test



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

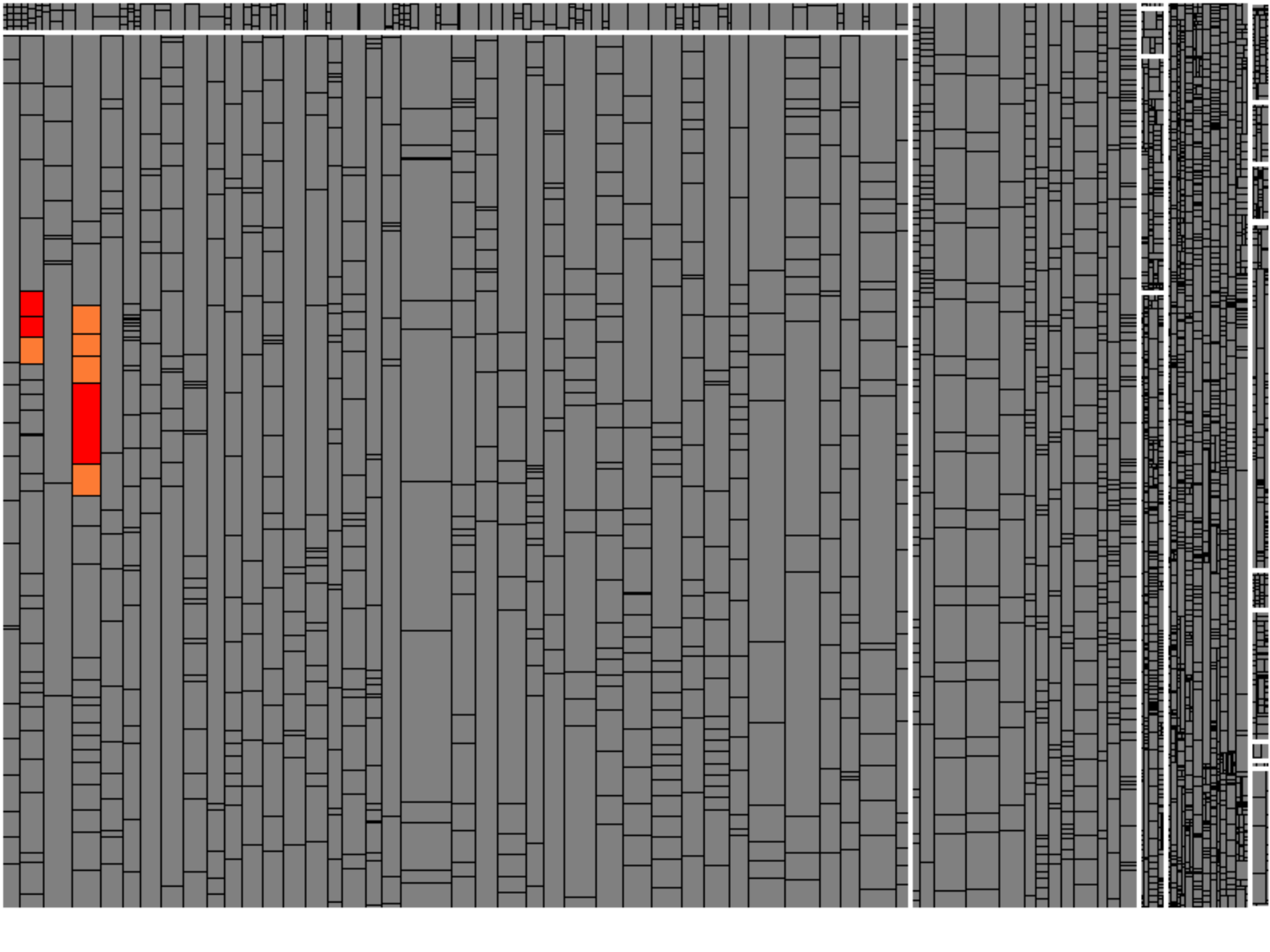


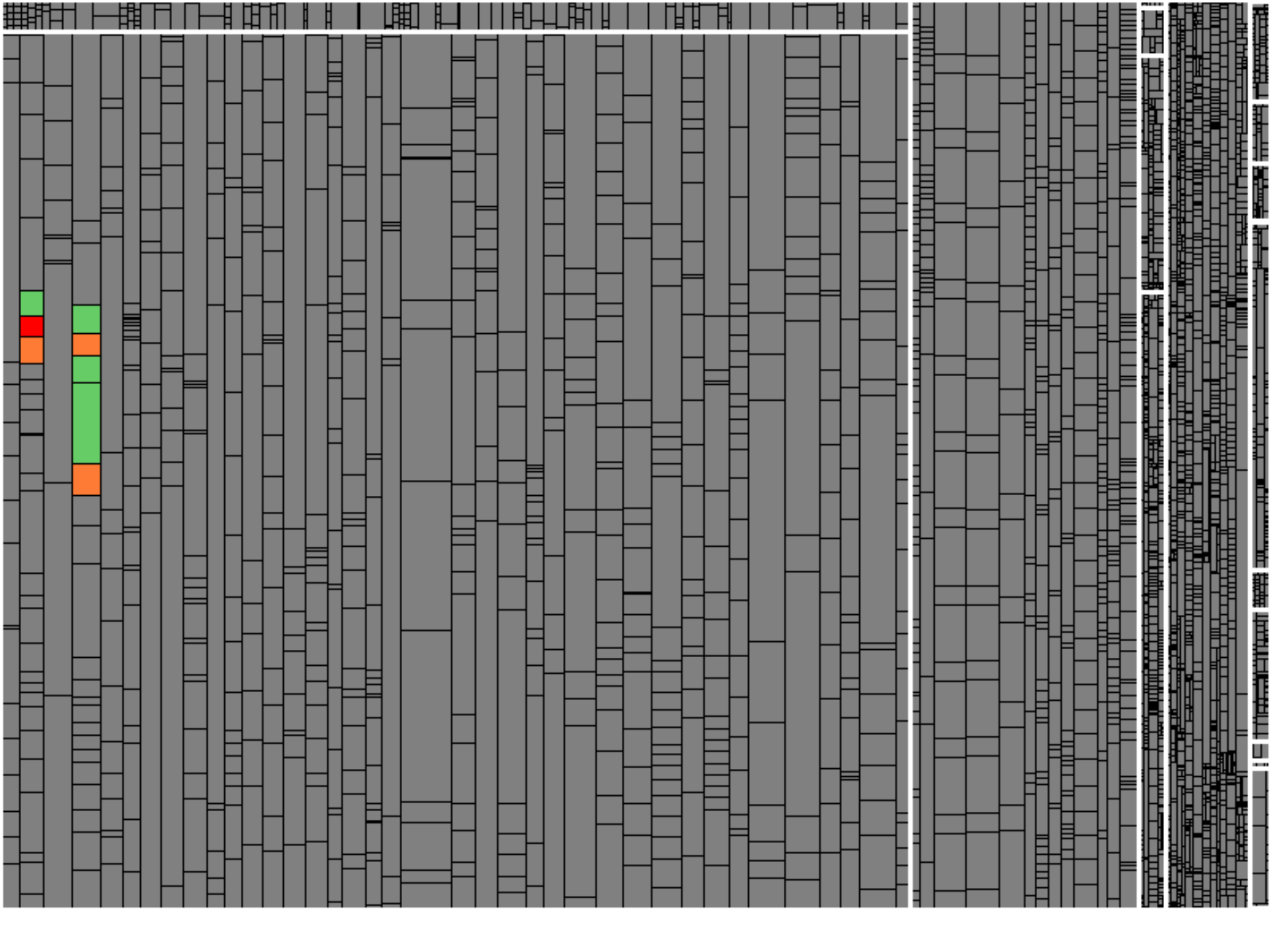




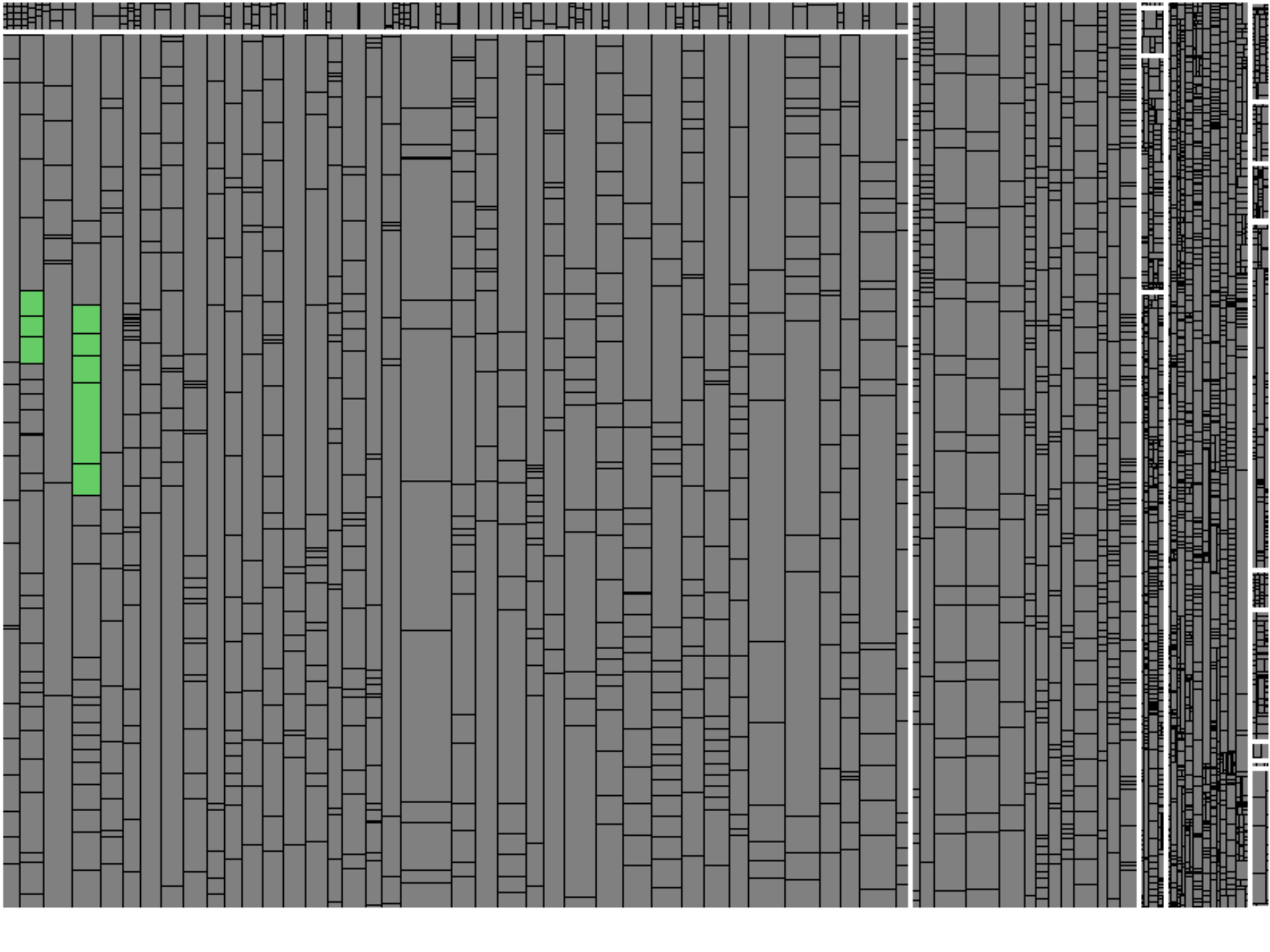
100% Change Coverage → 0 Fehler

# Fehlernachtest Bei Hotfix

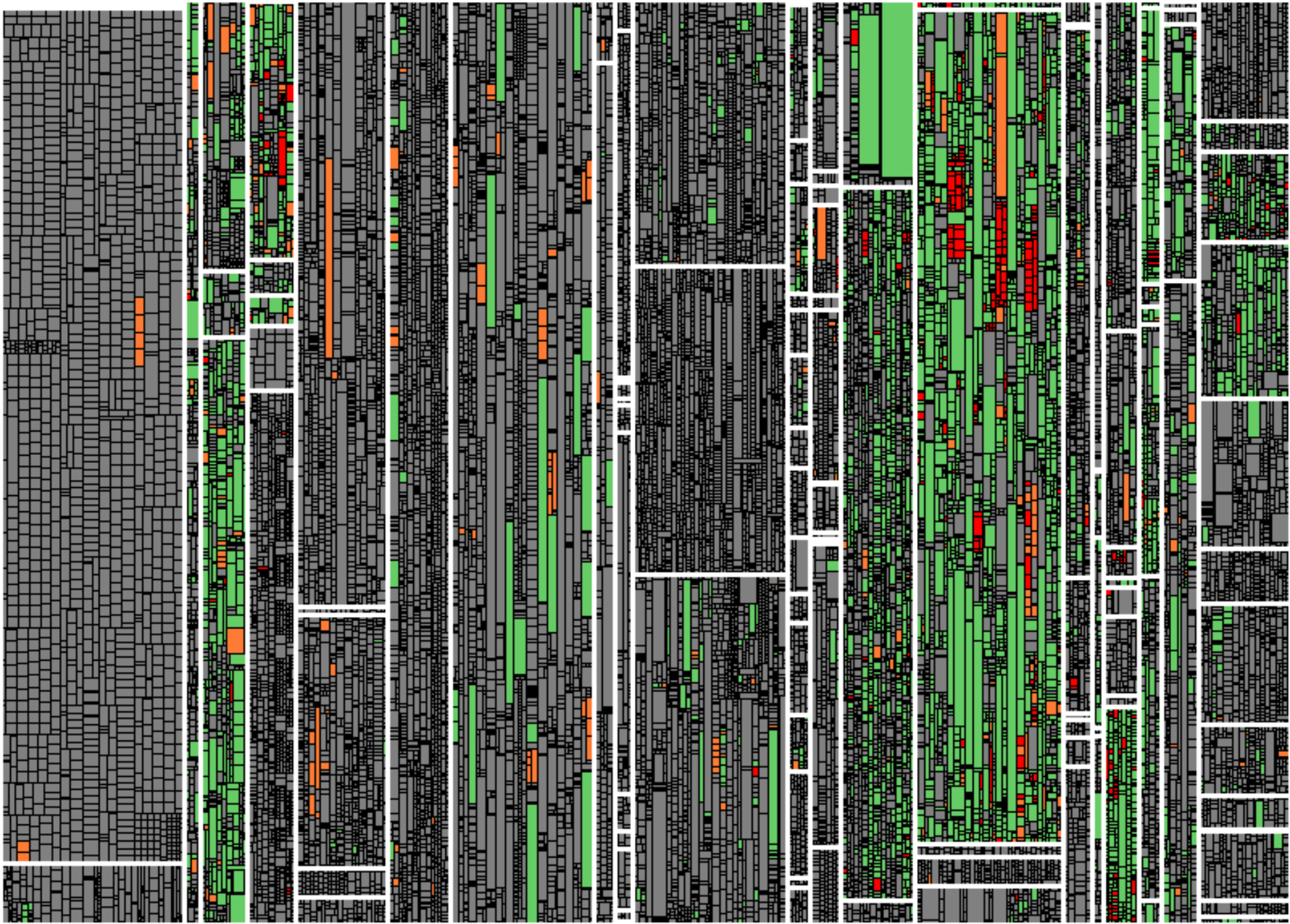


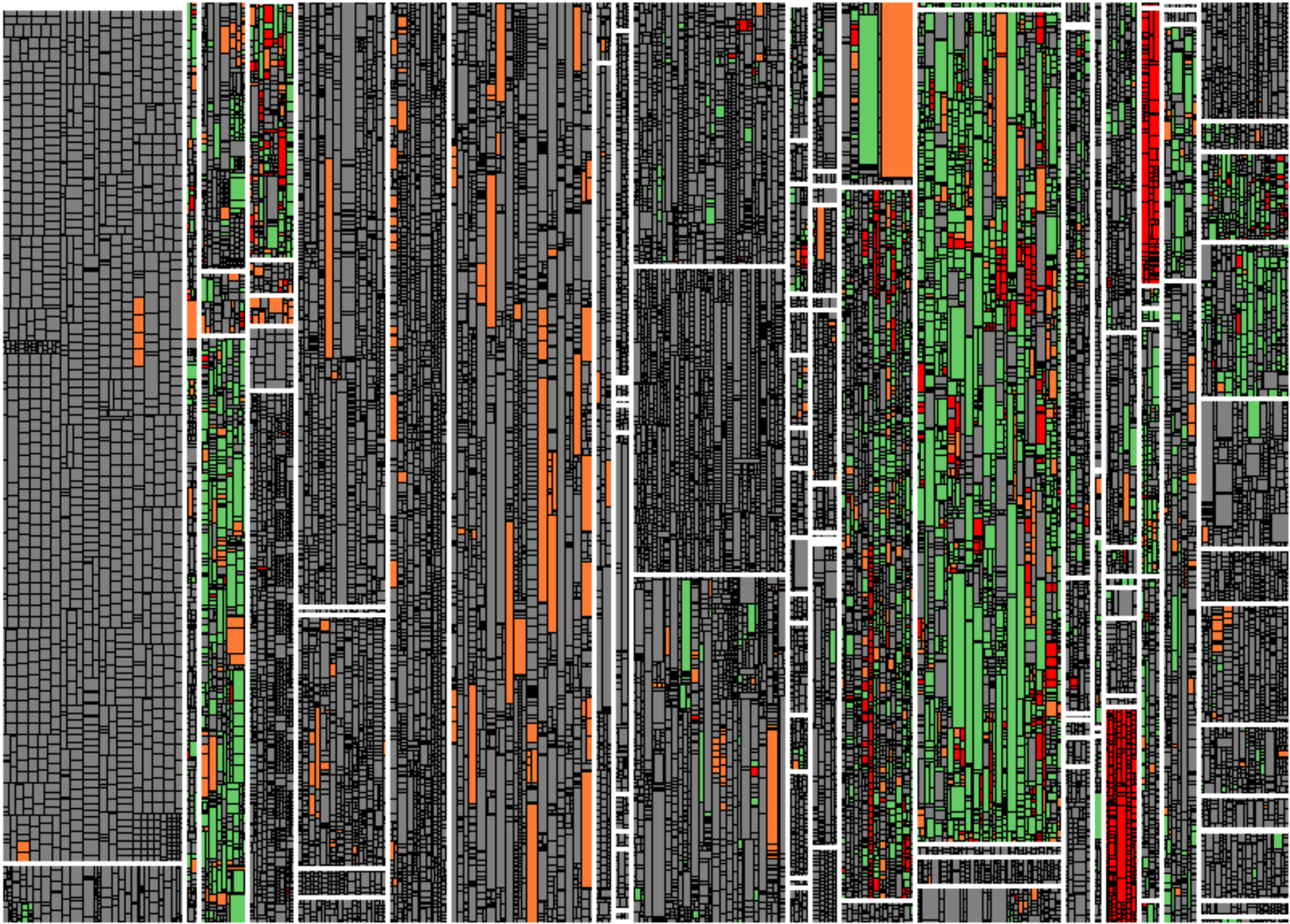


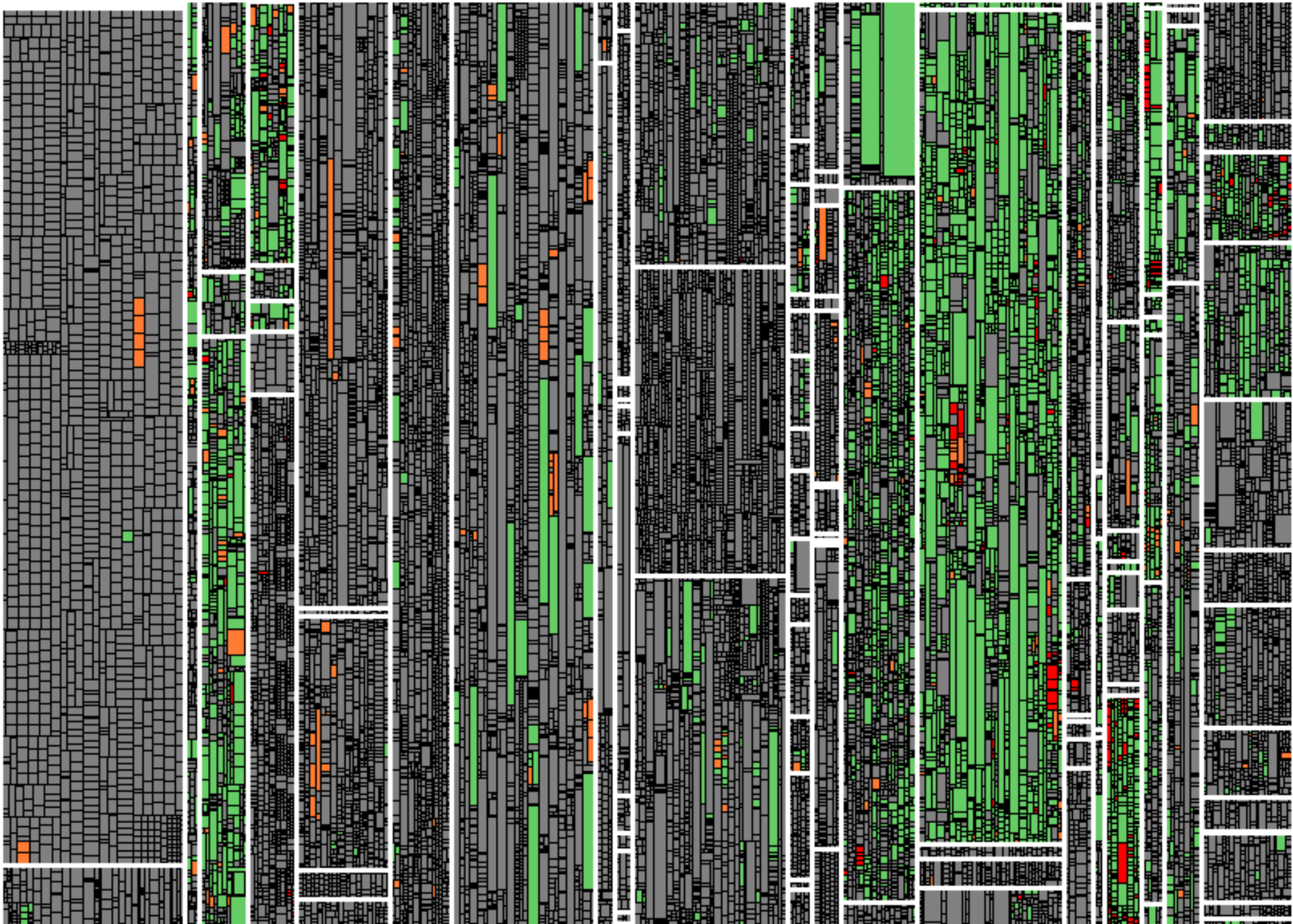


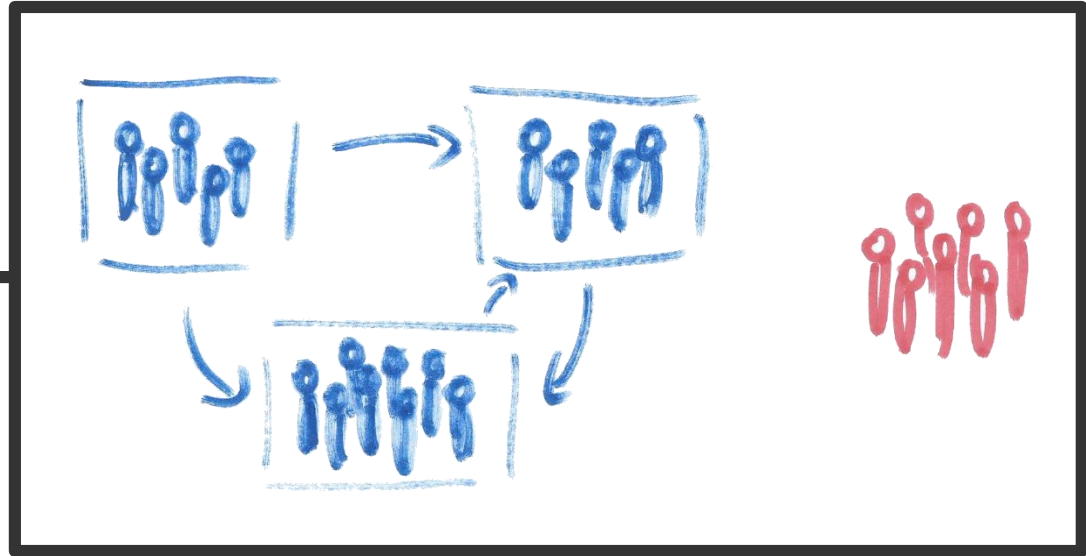
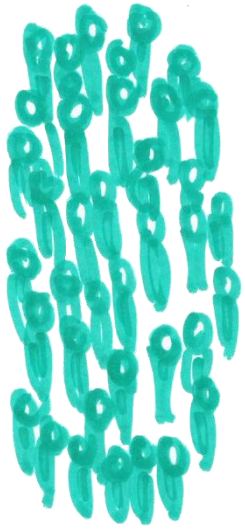


Release-Test



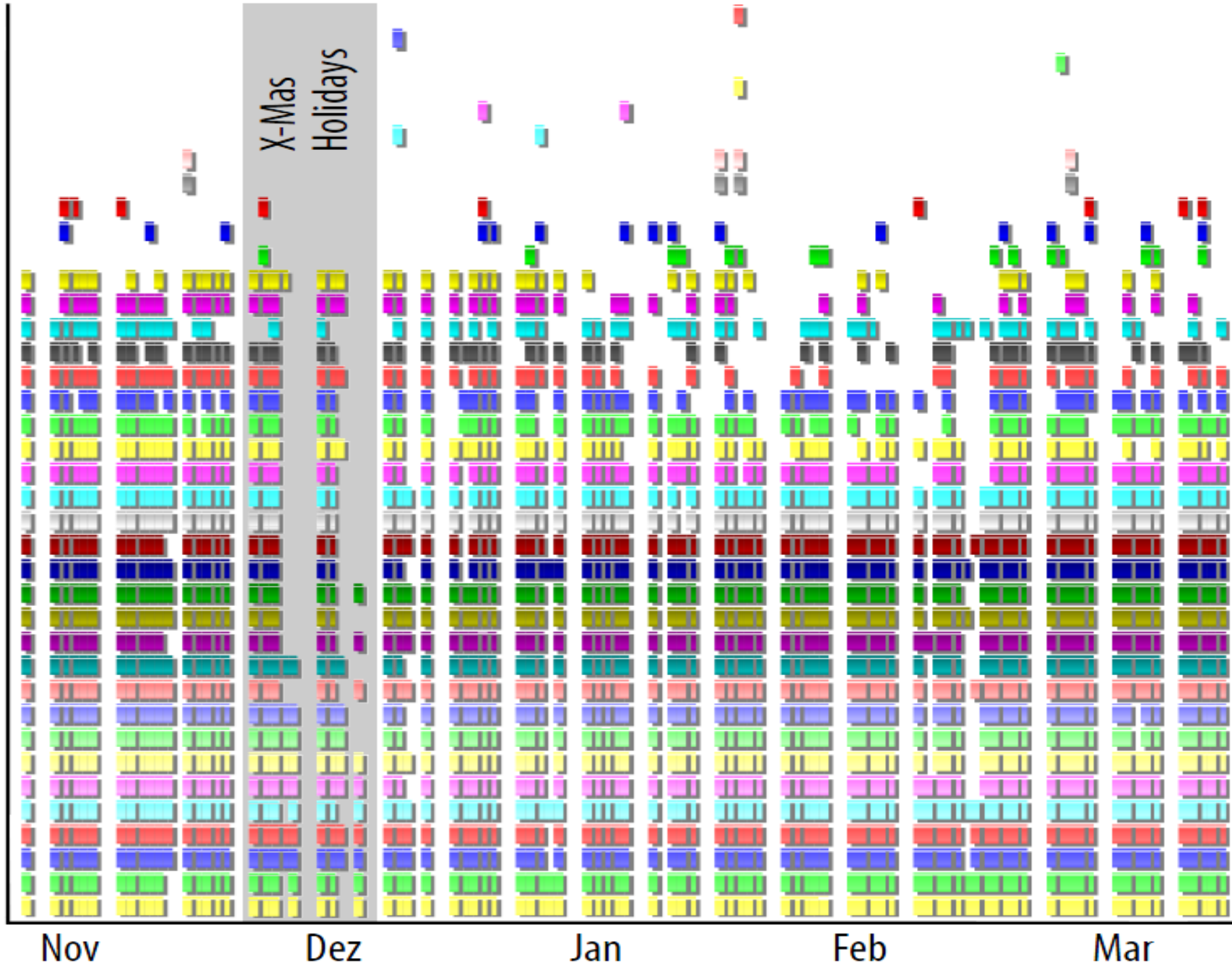






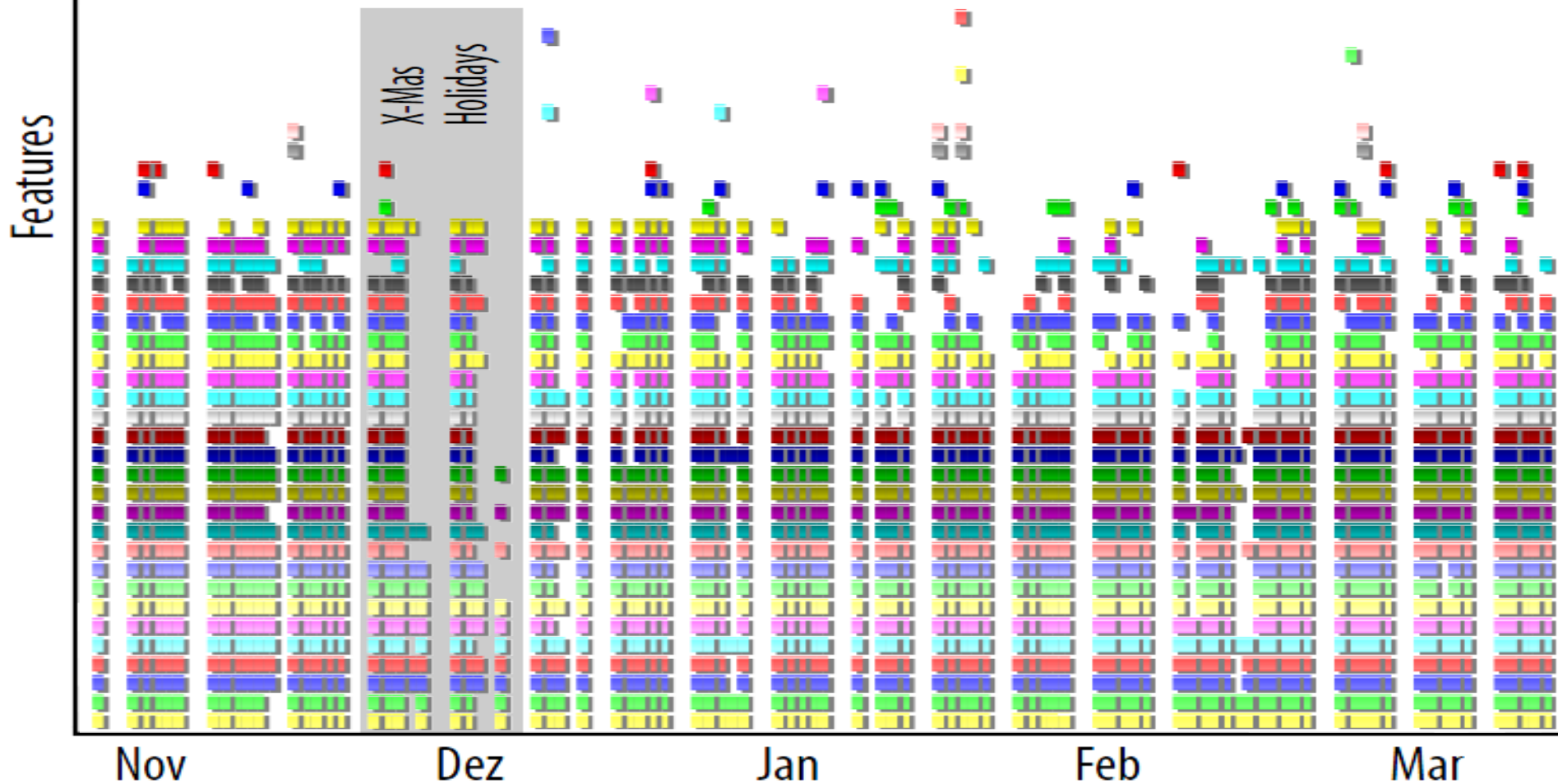


Features

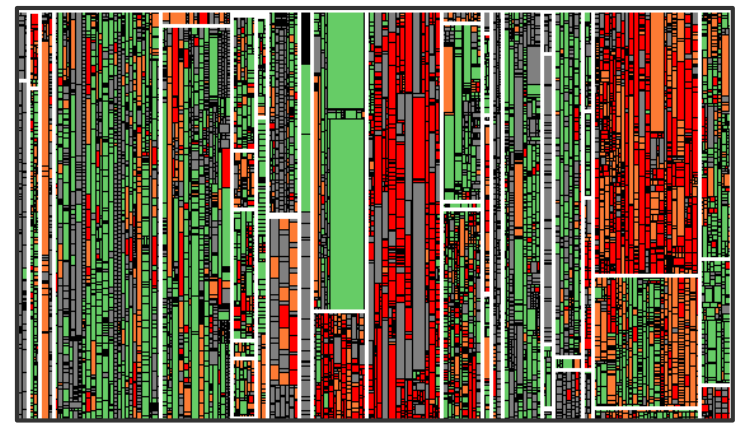
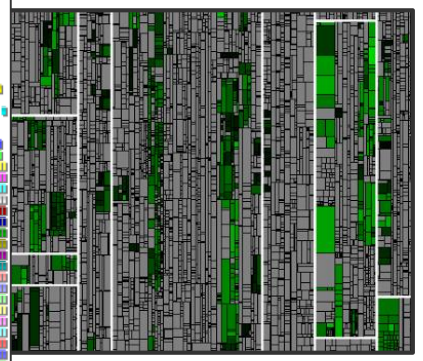
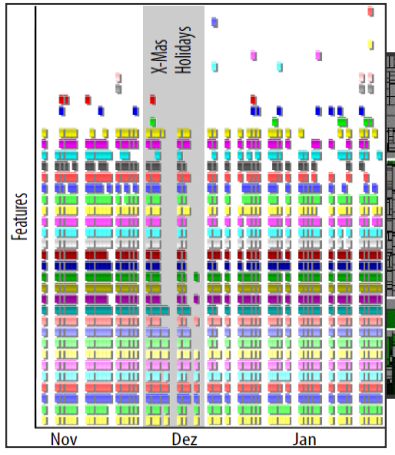
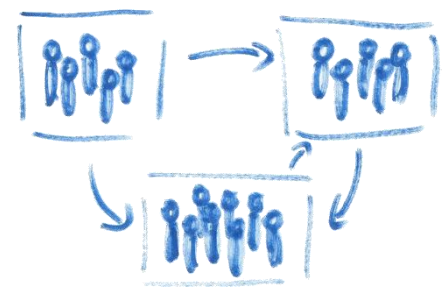
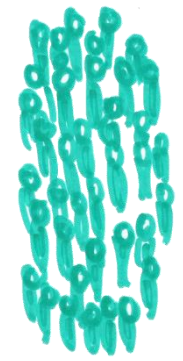
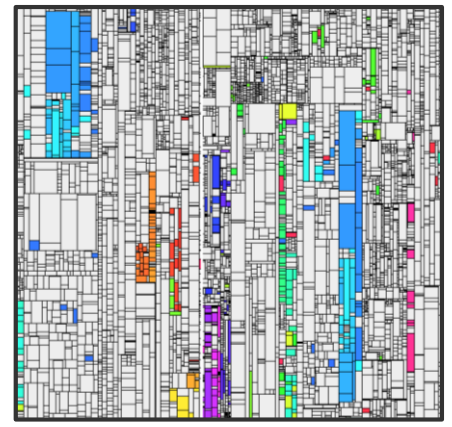
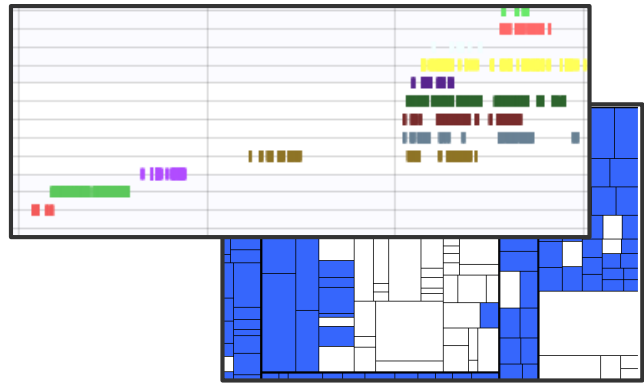
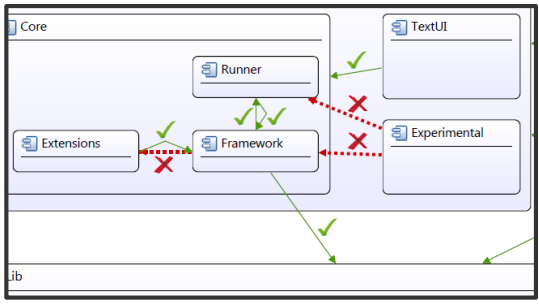


# 28% Features ungenutzt (15/53)

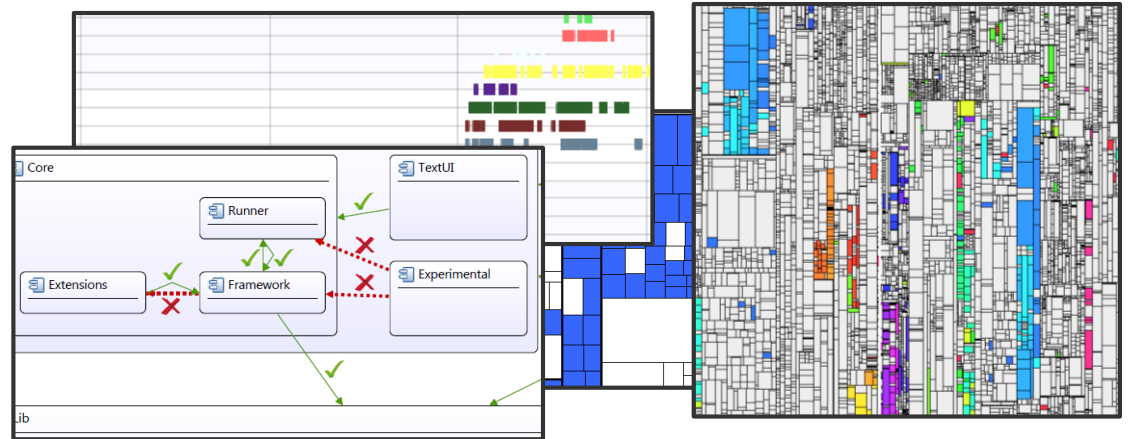
- 11 unerwartet



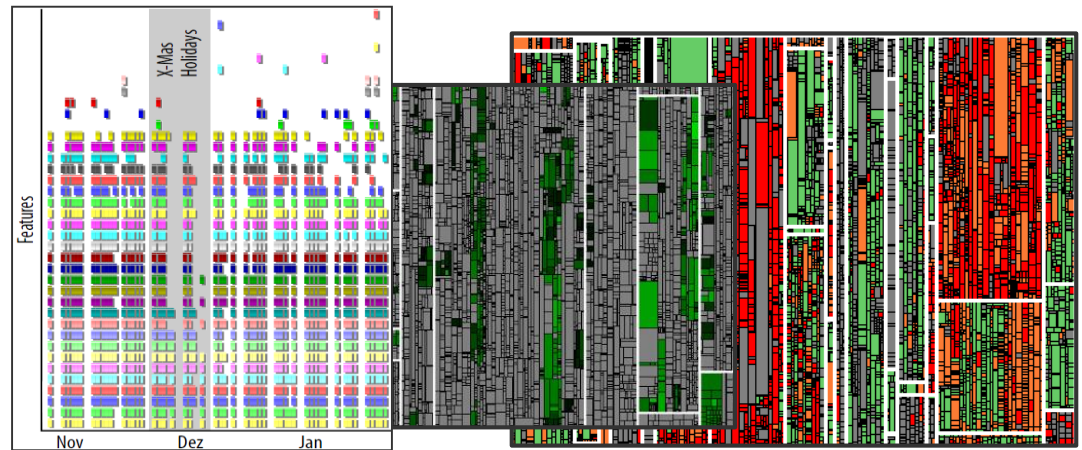




# Statische Analysen



# Dynamische Analysen



[www.teamscale.io](http://www.teamscale.io)







# Fazit

Es treten oft Probleme auf, wenn Architektur und Organisation nicht gut zusammenpassen.

Es gibt nützliche dynamische und statische Analysen, die Symptome ermitteln können, damit Ursachen abgestellt werden können.

Sie sind zu einem großen Grad automatisiert, erfordern aber Erfahrung in Konfiguration und Interpretation.

Wir unterstützen gerne.

# <https://www.cqse.eu/en/blog>

## Bridge your Test Gaps with Teamscale

Posted on 04/27/2016 by Dr. Dennis Pagano

Many companies employ sophisticated testing processes, but still bugs find their way into production. Often they hide among the subset of changes that were not tested. Actually, we found that untested changes are five times more error prone than the rest of the system.



The screenshot displays the Teamscale interface. At the top, the navigation bar includes 'Teamscale' and various menu items: Dashboard, Activity, Findings, Tasks, Code, Architecture, Delta, Projects, System, and Admin. The main content area is titled 'Test Gaps for pinta' with a subtitle 'Test Gap: 77.78%'. It features a treemap visualization where different colored blocks (red, yellow, green) represent various test gaps. To the right, a sidebar titled 'Editing Test Gaps Pinta' provides a list of visualization options: Label, Numeric Metric Value, Assessment Pie Chart, Metrics Table, Metrics Bar Chart, Metric Distribution Table, Metric Distribution Pie Chart, Metrics Hotspot Table, Metrics Change Table, Findings Summary Table, Findings Summary Bar Chart, Findings Churn, Metrics Trend Chart, Assessment Trend Chart, Treemap, Test Gap Treemap, Test Gap Trend Chart, Code City, and Metric File Distribution. At the bottom of the sidebar are 'Save', 'Save as', and 'Cancel' buttons.

To avoid code that is not consciously

A short and quick reference

Present

Obviously

To use

Please contact

Test

Test Gaps tested test gaps

ed

u a st,

ge regularly.

verage can be found here.

ges made on the way have been want to show you how to find



## Testing Changes in SAP BW Applications

Posted on 04/29/2015 by Dr. Andreas Göb

As my colleague Fabian explained a few weeks ago, a combination of change detection and execution logging can substantially increase transparency regarding which recent changes of a software system have actually been covered by the testing process. I will not repeat all the details of the Test Gap Analysis approach here, but instead just summarize the core idea: Untested new or changed (informative) information. Therefore it makes sense to use only those changed but untested areas.

Several times  
In the main  
from Python  
containing  
may provide  
Analysis in

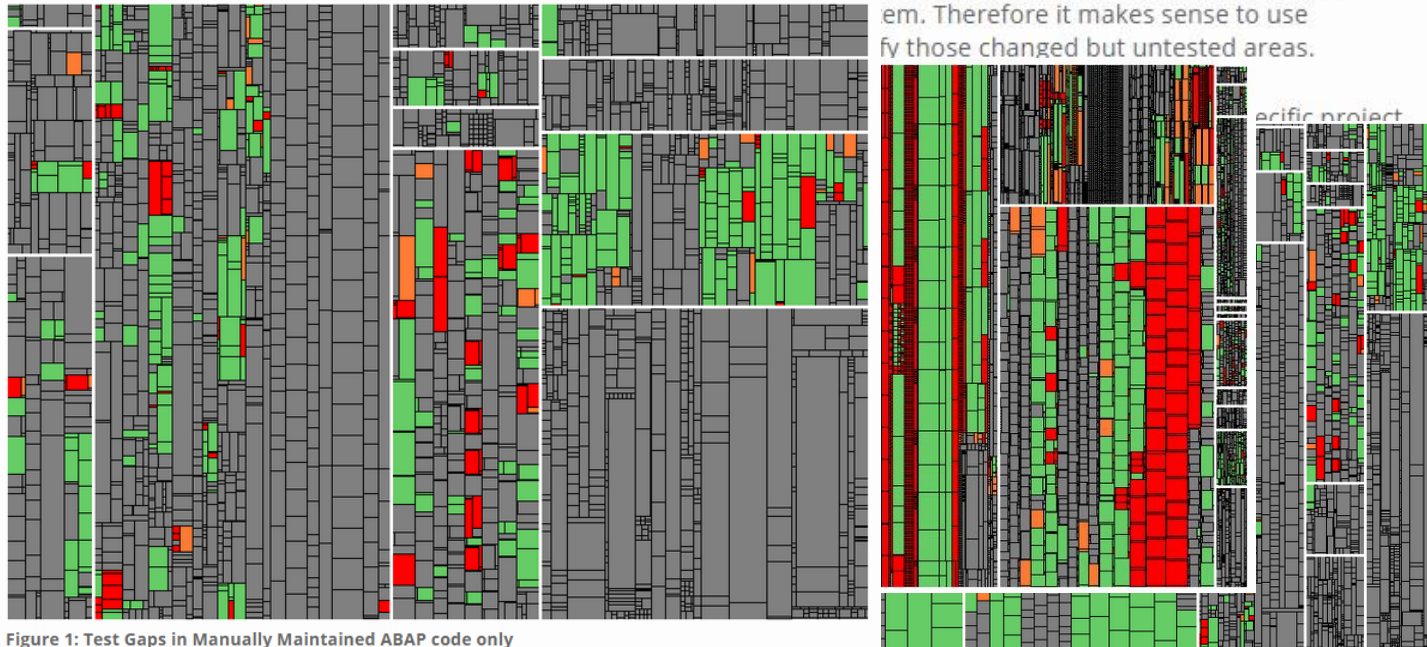


Figure 1: Test Gaps in Manually Maintained ABAP code only



# Software Quality Blog

## Practical Guide to Code Clones (Part 1)

Posted on 07/16/2014 by Dr. Benjamin Hummel

One well known principle in software engineering states *don't repeat yourself*, also known as the DRY principle. A very obvious violation of DRY is the application of copy/paste to create duplicates of large portions of source code within the same code base. These duplicate pieces of code, also known as *code clones*, have been subject to lots of research in the last two decades. In this two-part post I want to summarize those parts of the current knowledge that I find most relevant to the practitioner, especially the impact of clones on software development.



## Practical Guide to Code Clones (Part 2)

Posted on 07/30/2014 by Dr. Benjamin Hummel

In the previous part we introduced the notion of code clones and discussed, whether and under which circumstances cloning in your code base can be a problem for development and maintenance. In this post, I will introduce ways and tools to deal with code clones in your code base. After reading this, you should be able to select and apply a detection tool to inspect the clones in your own code base.



# Kontakt

Dr. Elmar Jürgens · [juergens@cqse.eu](mailto:juergens@cqse.eu) · +49 179 675 3863

@ElmarJuergens

[www.cqse.eu/de/ressourcen/blog](http://www.cqse.eu/de/ressourcen/blog)

CQSE GmbH

Lichtenbergstraße 8

85748 Garching bei München