Test-Gap-Analyse

Erfahrungen aus drei Jahren Praxiseinsatz

Dr. Andreas Göb

German Testing Day 2016

CQSE
Continuous Quality in Software Engineering

Über mich

Forschung

- Modellierung von Softwarequalität
- Wartbarkeit, Software-Architektur

Beratung

- Qualitäts-Bewertung & Qualitäts-Controlling
- Wirksamer Werkzeugeinsatz in Entwicklung und Test

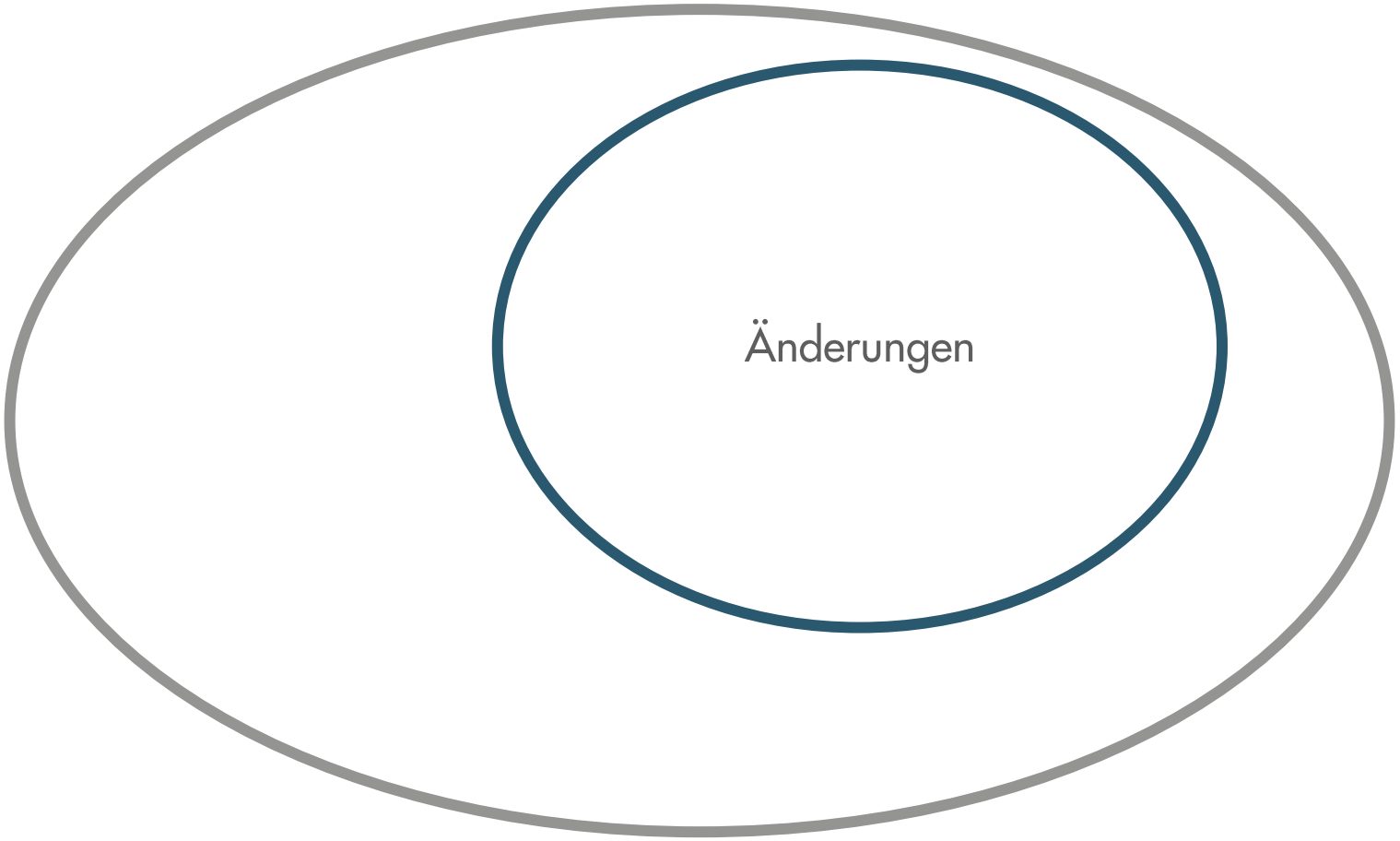
Entwicklung

- Continuous Quality Assessment Toolkit ConQAT
- >400 kLOC, Apache Lizenz, >25.000 Downloads
- Kommerzielle Erweiterung: Teamscale

Agenda

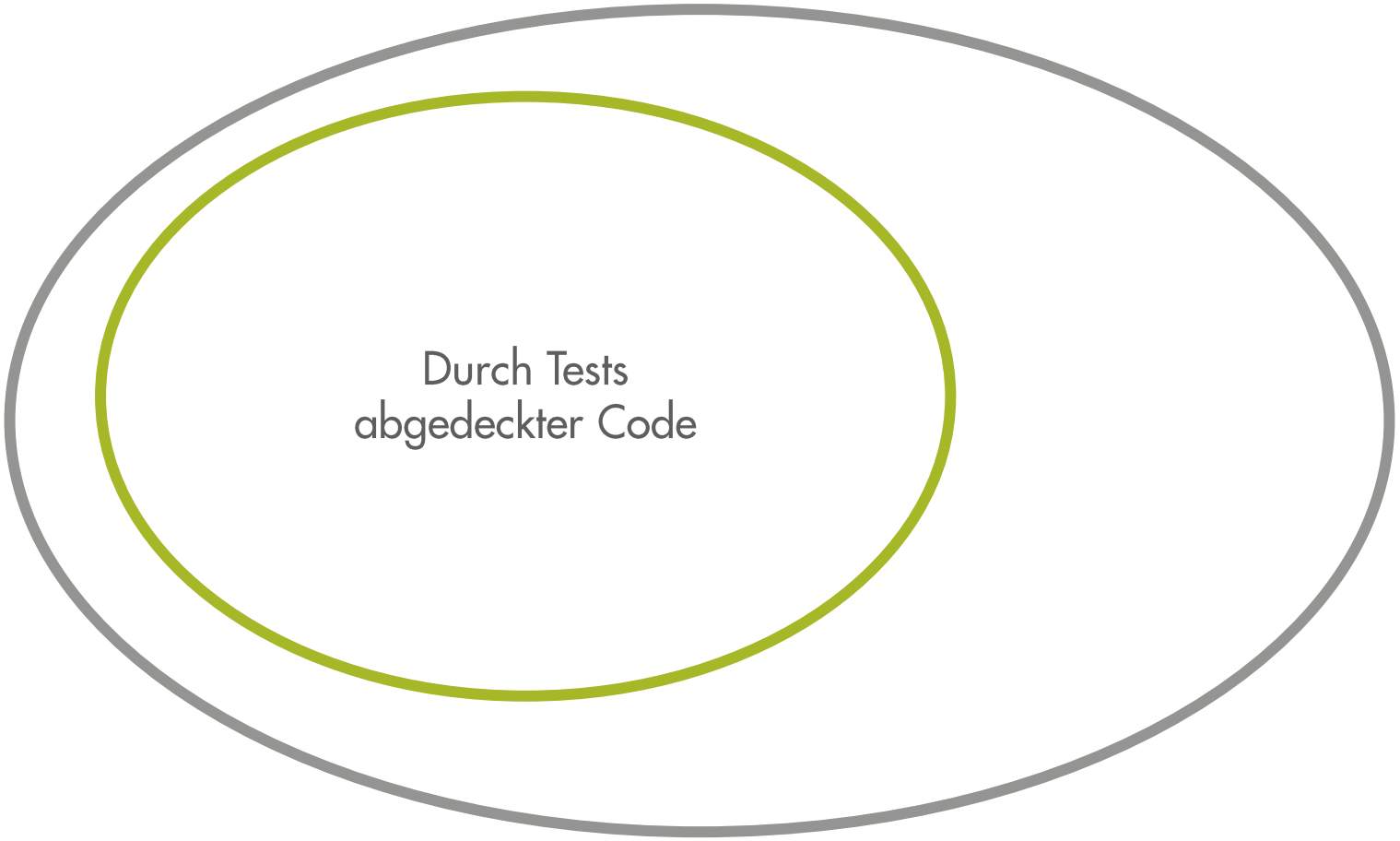
- **Grundlagen**
- Erfahrungen aus der Praxis
- Ausblick und Diskussion

Hintergrund



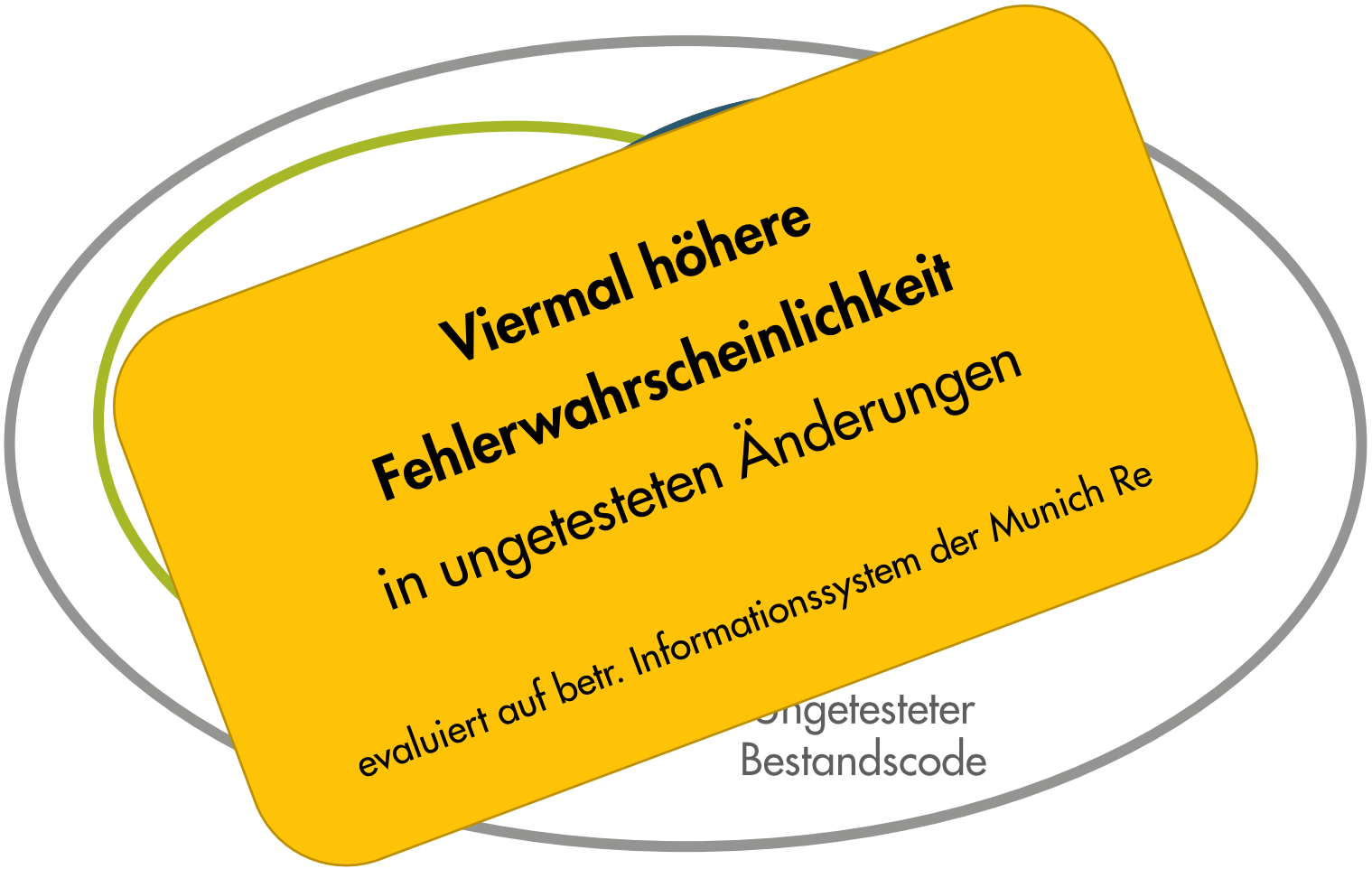
Eder, Jürgens, … *Did We Test Our Changes? Assessment btw. Tests & Development in Practice*, AST@ICSE 2013

Hintergrund



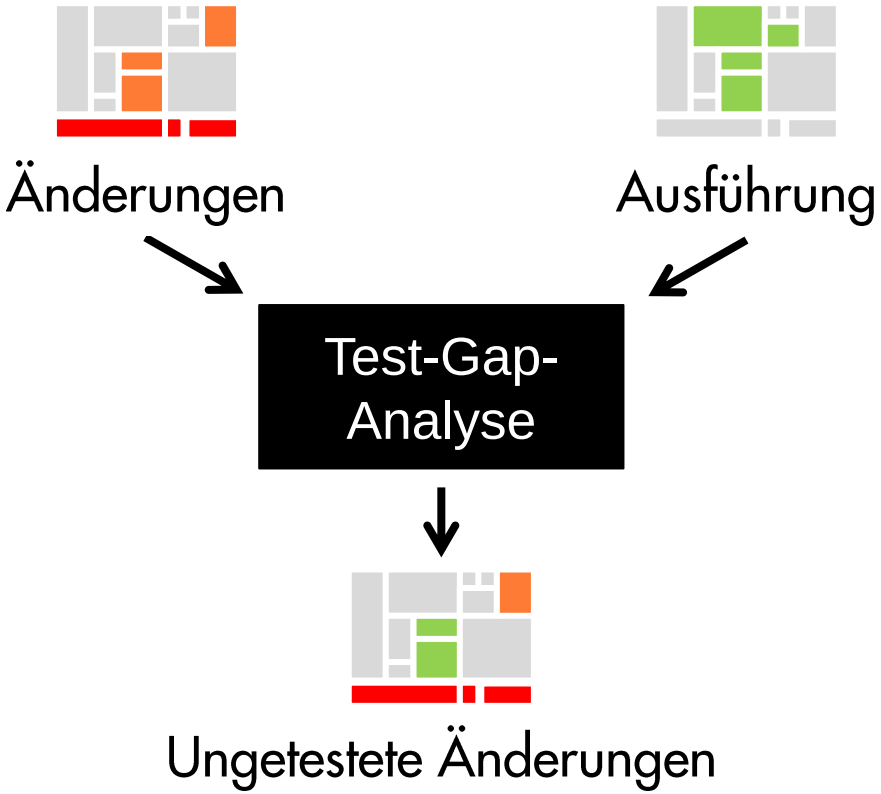
Eder, Jürgens, … *Did We Test Our Changes? Assessment btw. Tests & Development in Practice*, AST@ICSE 2013

Hintergrund

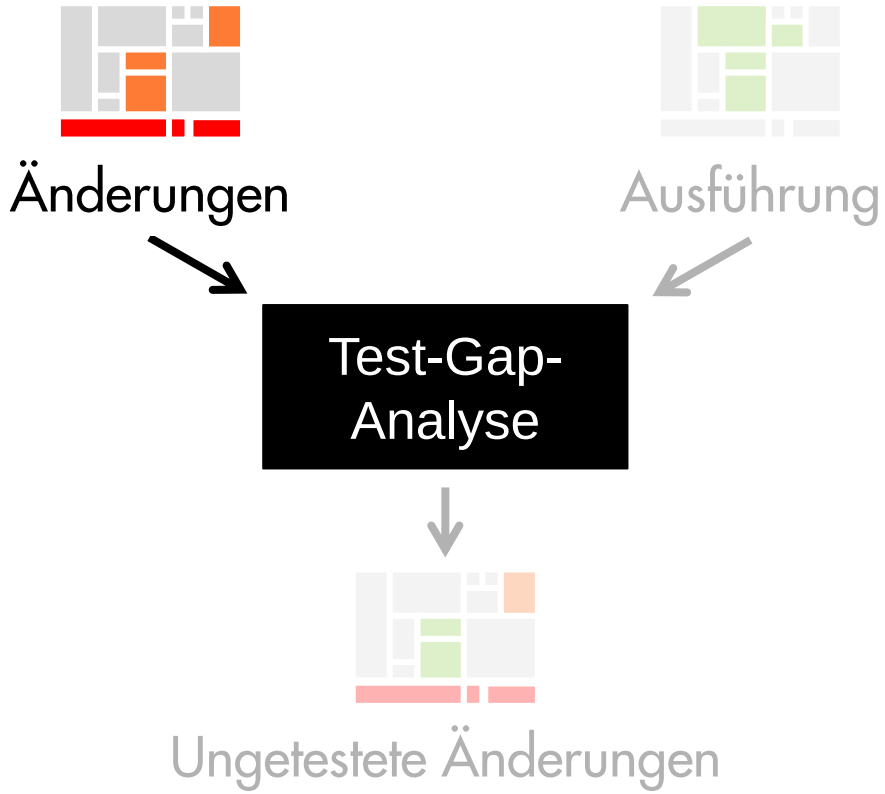


Eder, Jürgens, … *Did We Test Our Changes? Assessment btw. Tests & Development in Practice*, AST@ICSE 2013

Die Test-Gap-Analyse



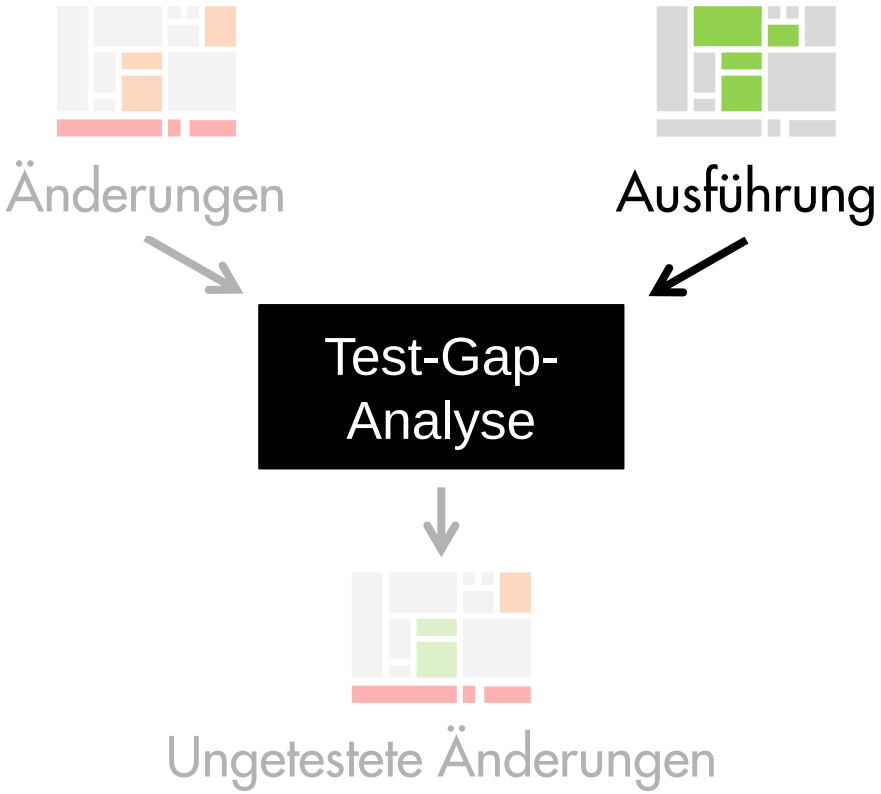
Änderungserkennung



- Gegenüber Referenzstand
- Akkumulierte Analyse

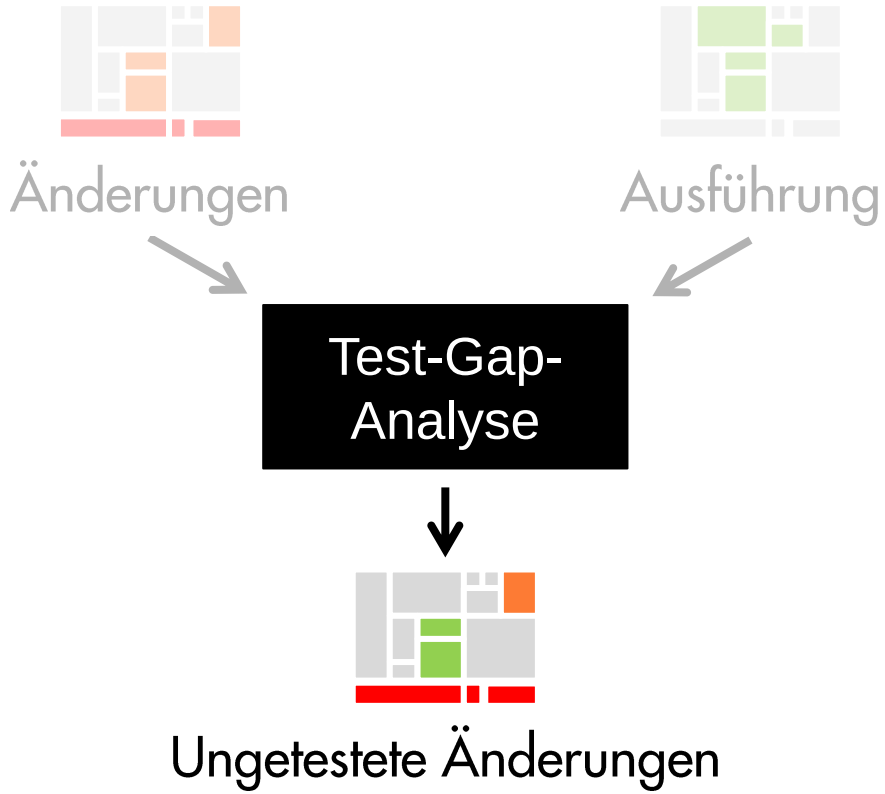
- Nur funktionale Änderungen
  - Keine Kommentare
  - Keine Umbenennungen
  - Keine Verschiebungen
  - Keine Refactorings

Auswertung der Ausführung



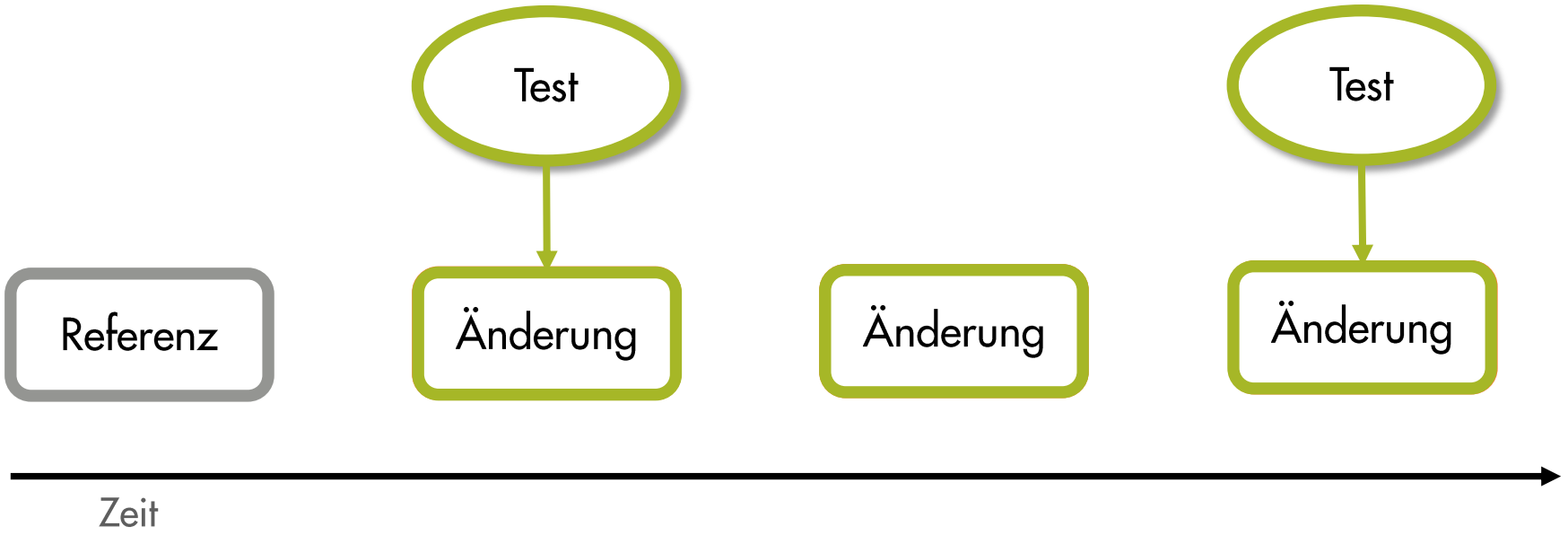
- Granularität: Methoden
- Auf Server oder Clients
- Manuelle oder Unit-Tests
- Minimaler Overhead
- Plattformspezifische Profiler

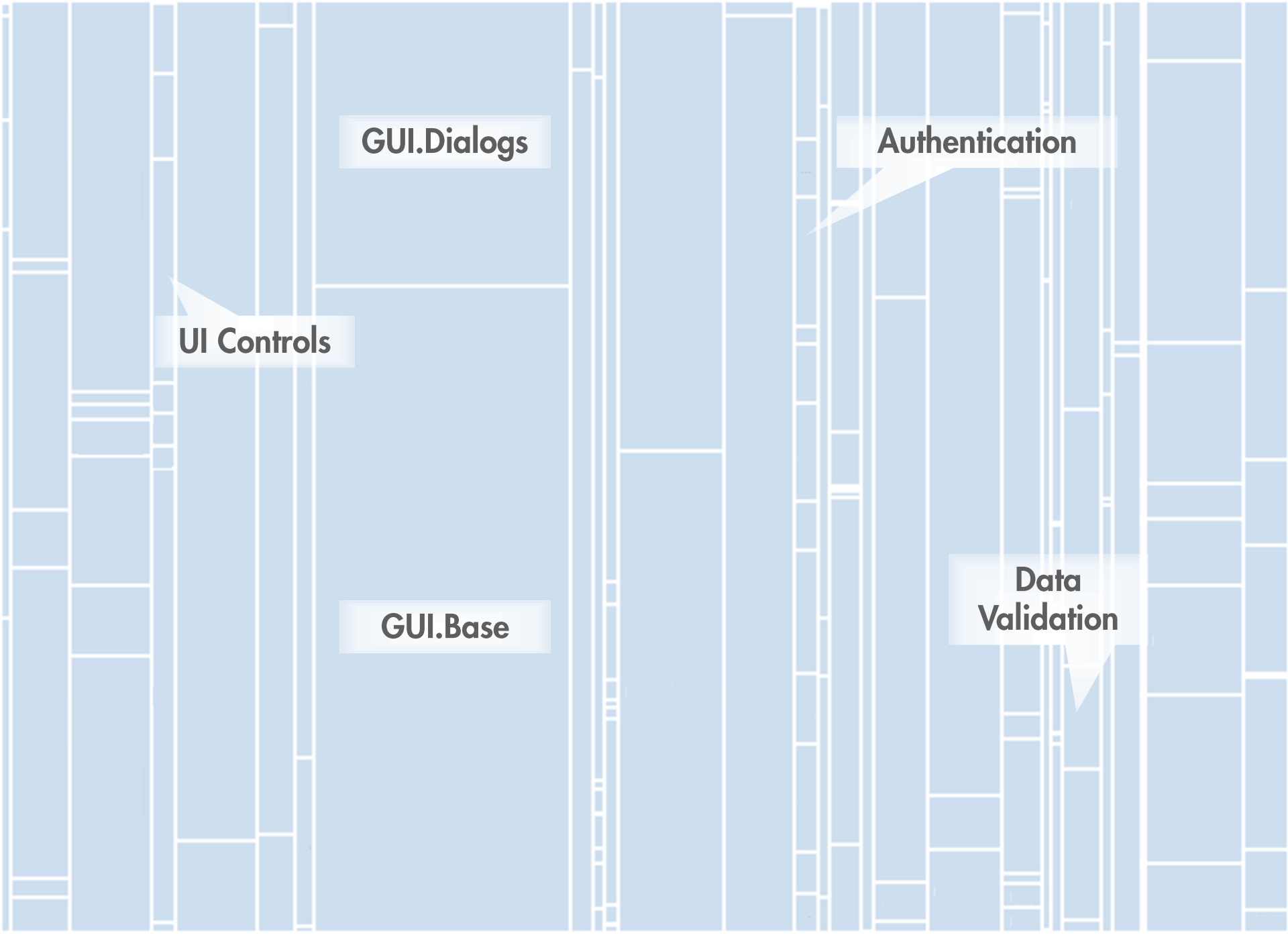
Kombination von Änderung und Ausführung

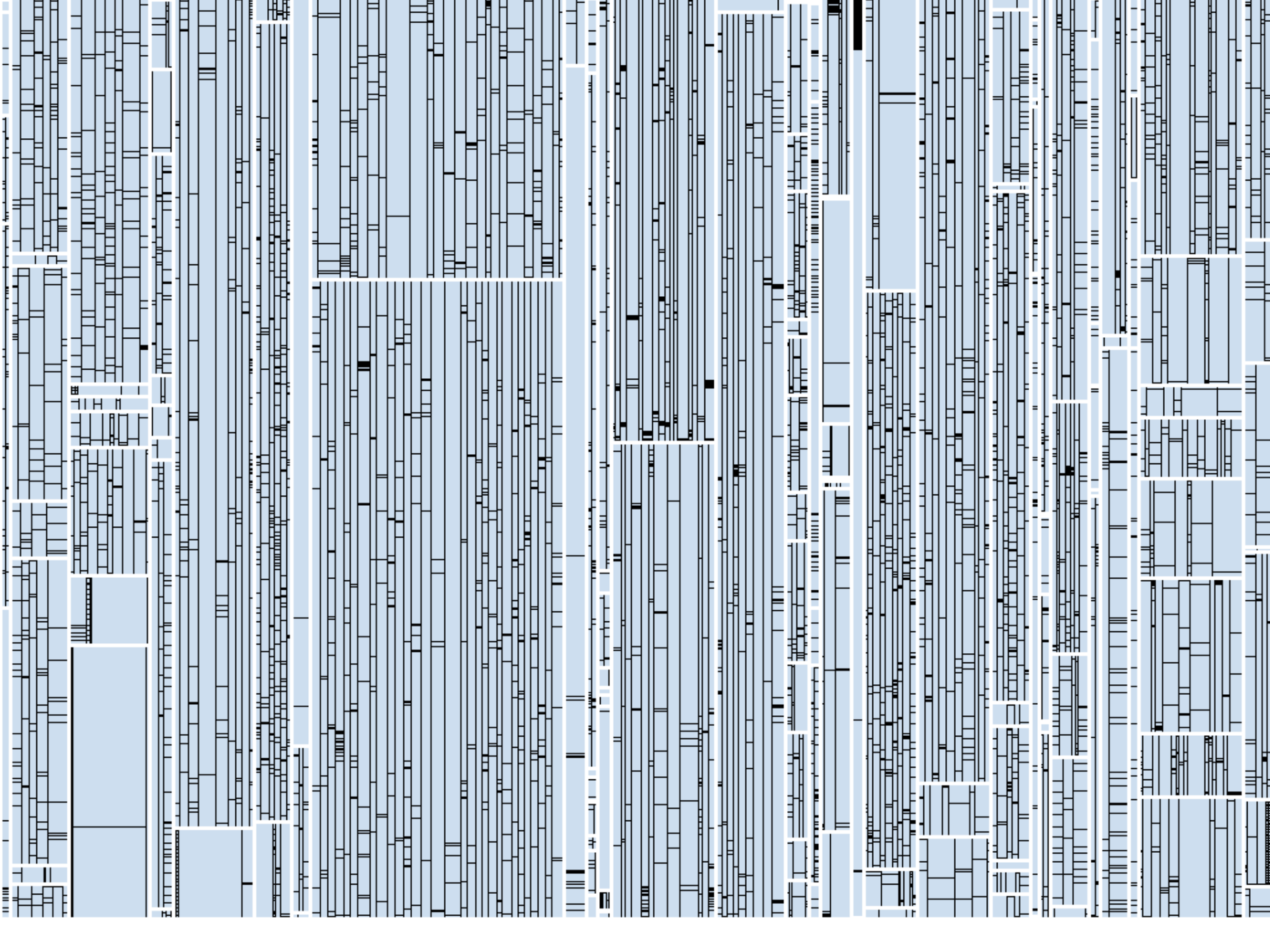


- Täglich aktualisiert
- Inkrementell verzahnt
- Pro Testumgebung
- Aggregierte Sicht

Verzahnung von Änderungen und Ausführung




GUI.Dialogs
Authentication
UI Controls
Data
Validation
GUI.Base



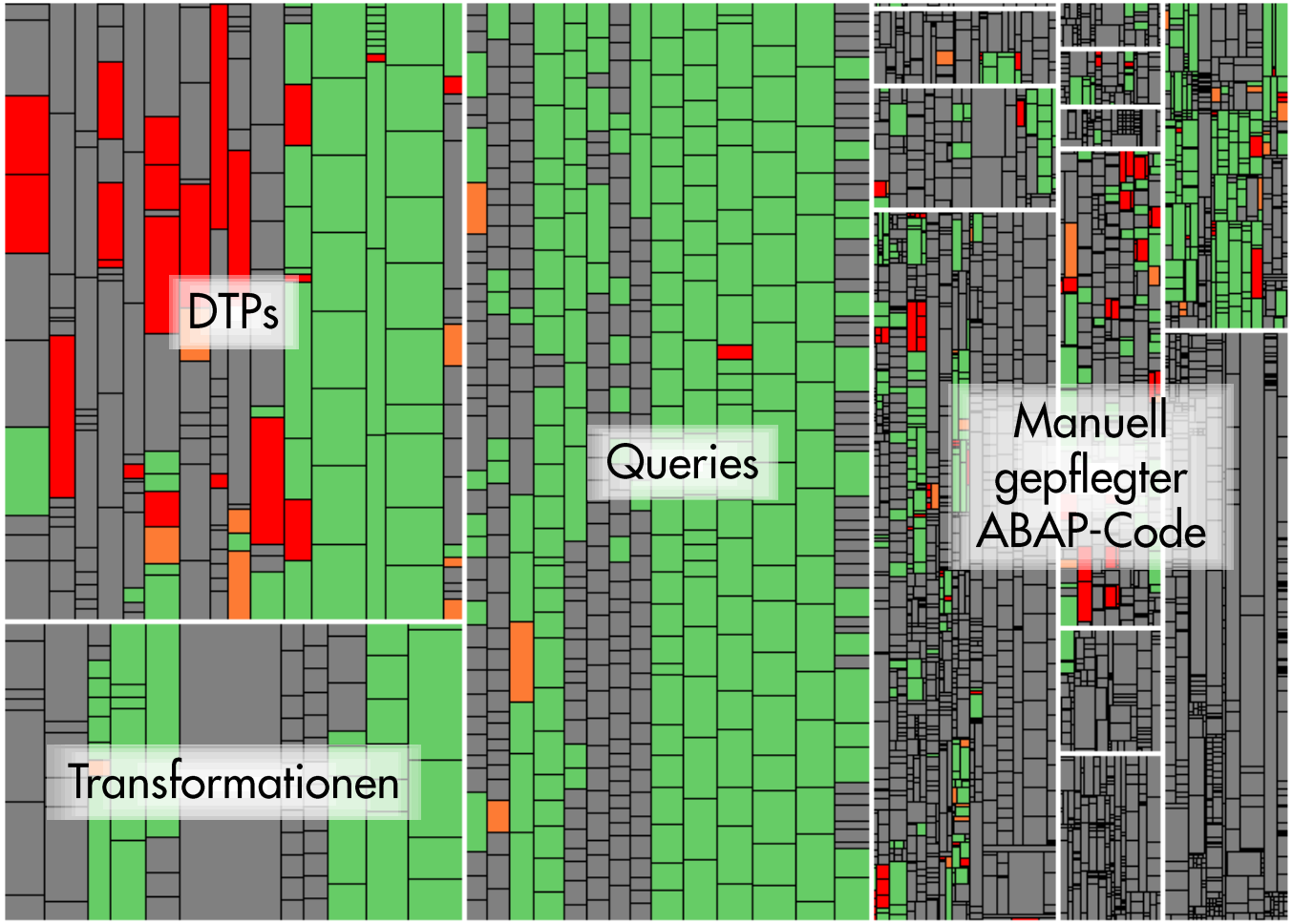
Wie einfach ist das übertragbar?

- Unabhängig von der Technologie:
  - Änderung und Ausführung von Methoden / Prozeduren
  - Aggregation, Darstellung, zeitlicher Verlauf

- Nötige Anpassungen pro Sprache / Technologie:
  - Scanner, Parser (aus ConQAT-Framework) — 17 Sprachen
  - Refactoring-Erkennung — ABAP, C#, Java
  - Profiler (z.B. aus Testwerkzeugen) — ABAP, .NET, Java, Python

- Ggf. Sonderbehandlung für generierten Code (z.B. aus SAP BW)

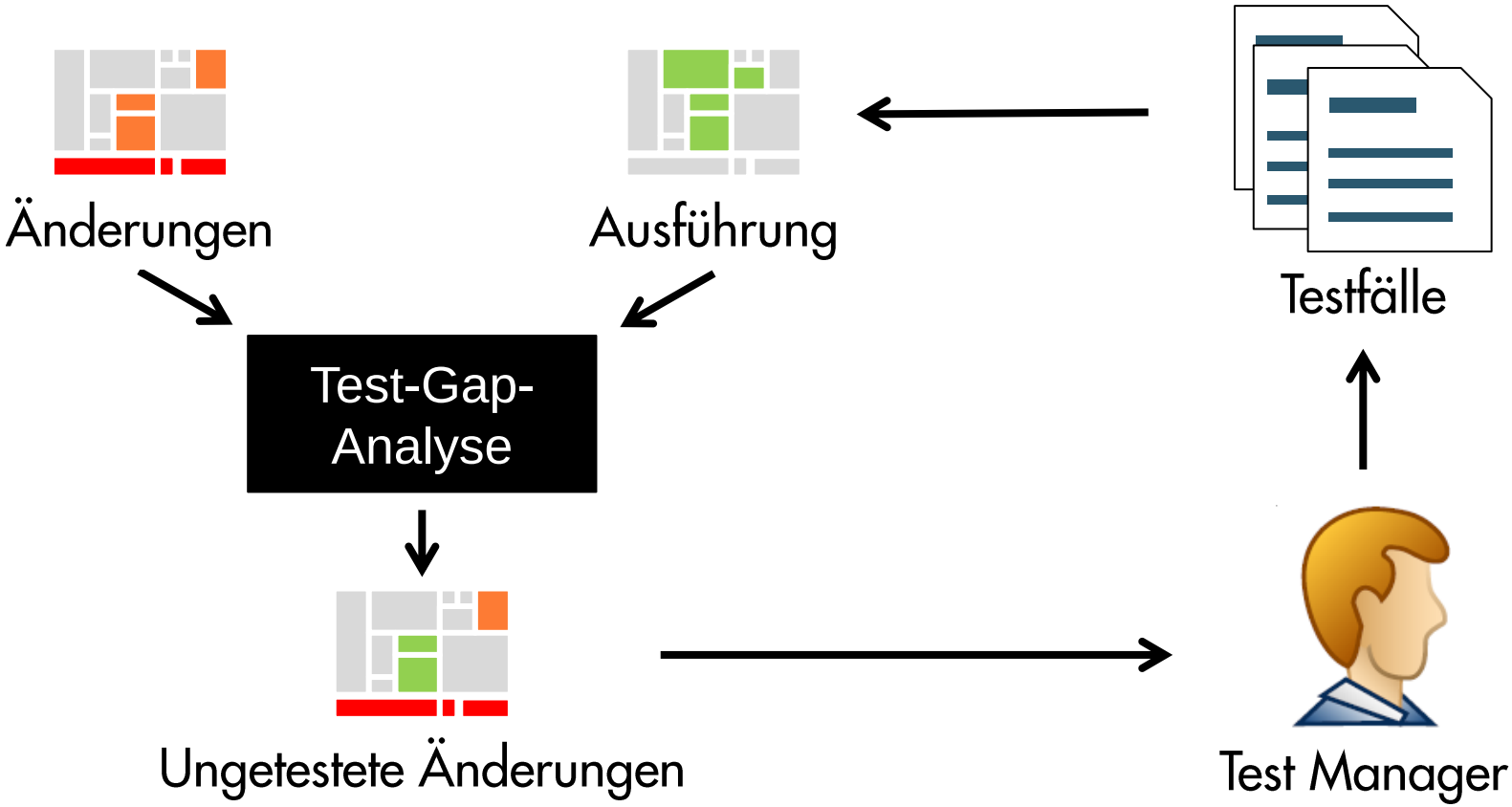
Übertragbarkeit: Beispiel SAP BW

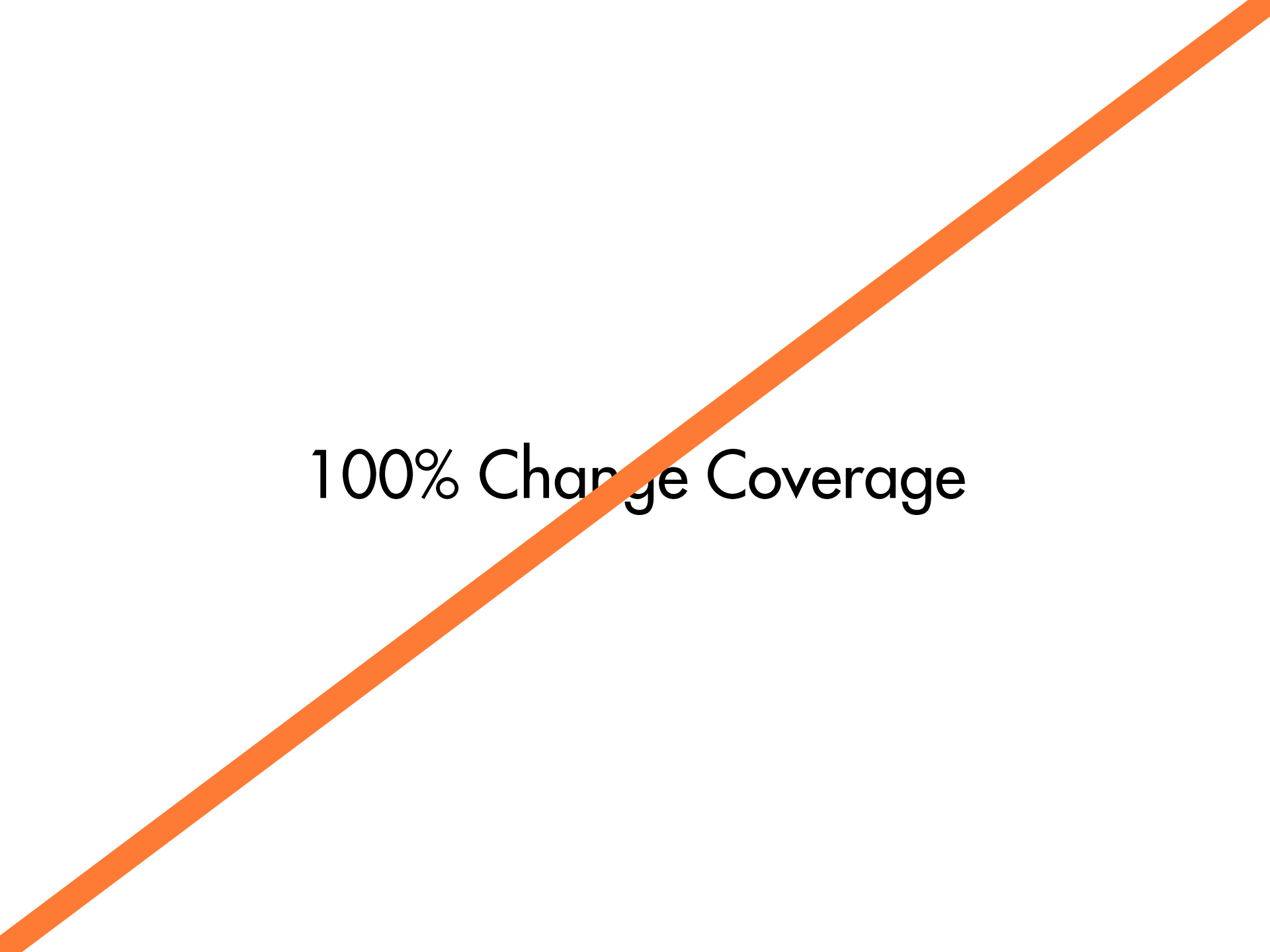


Agenda

- Grundlagen
- **Erfahrungen aus der Praxis**
- Ausblick und Diskussion

Einsatz im Testprozess



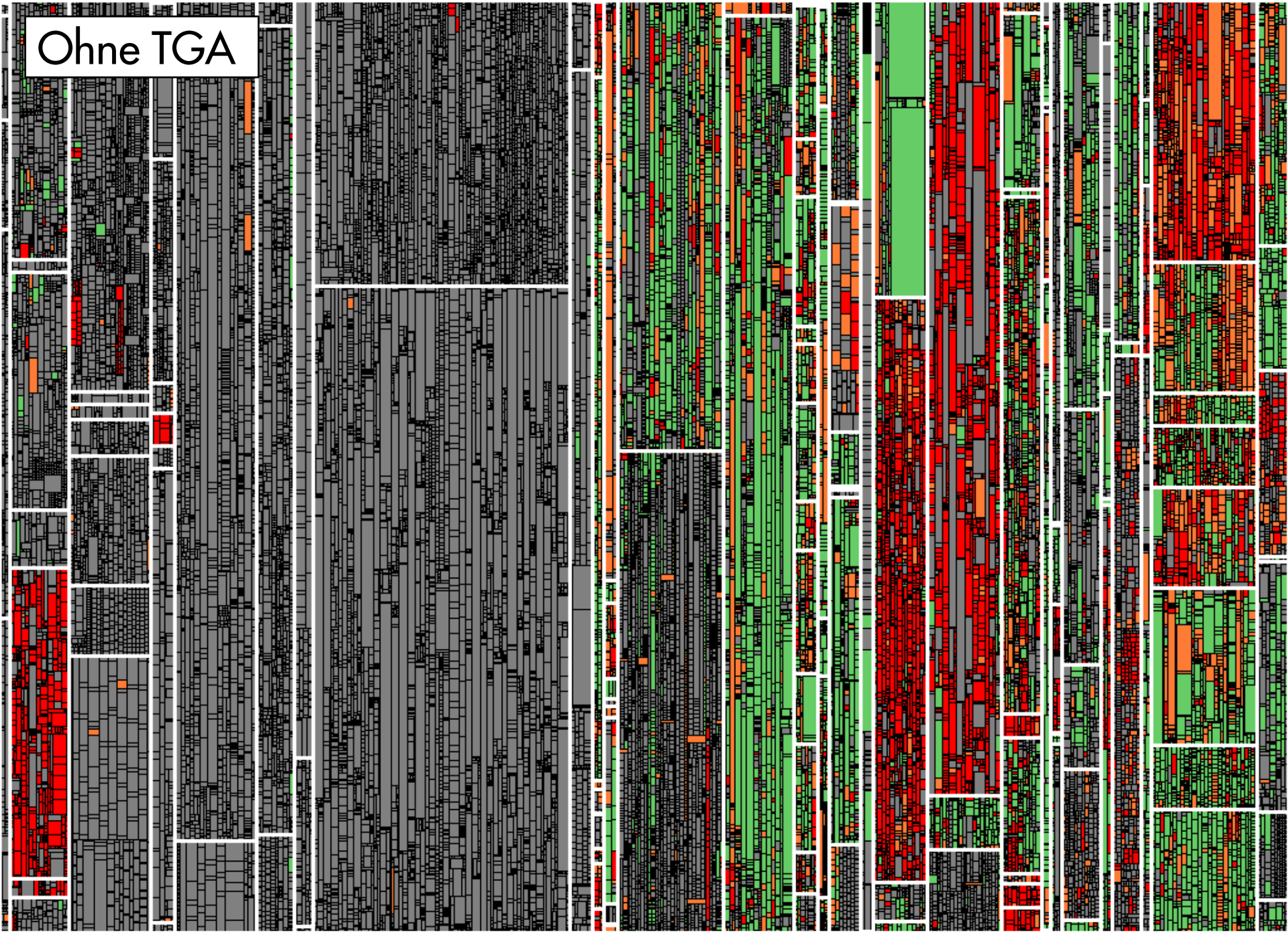

100% Change Coverage

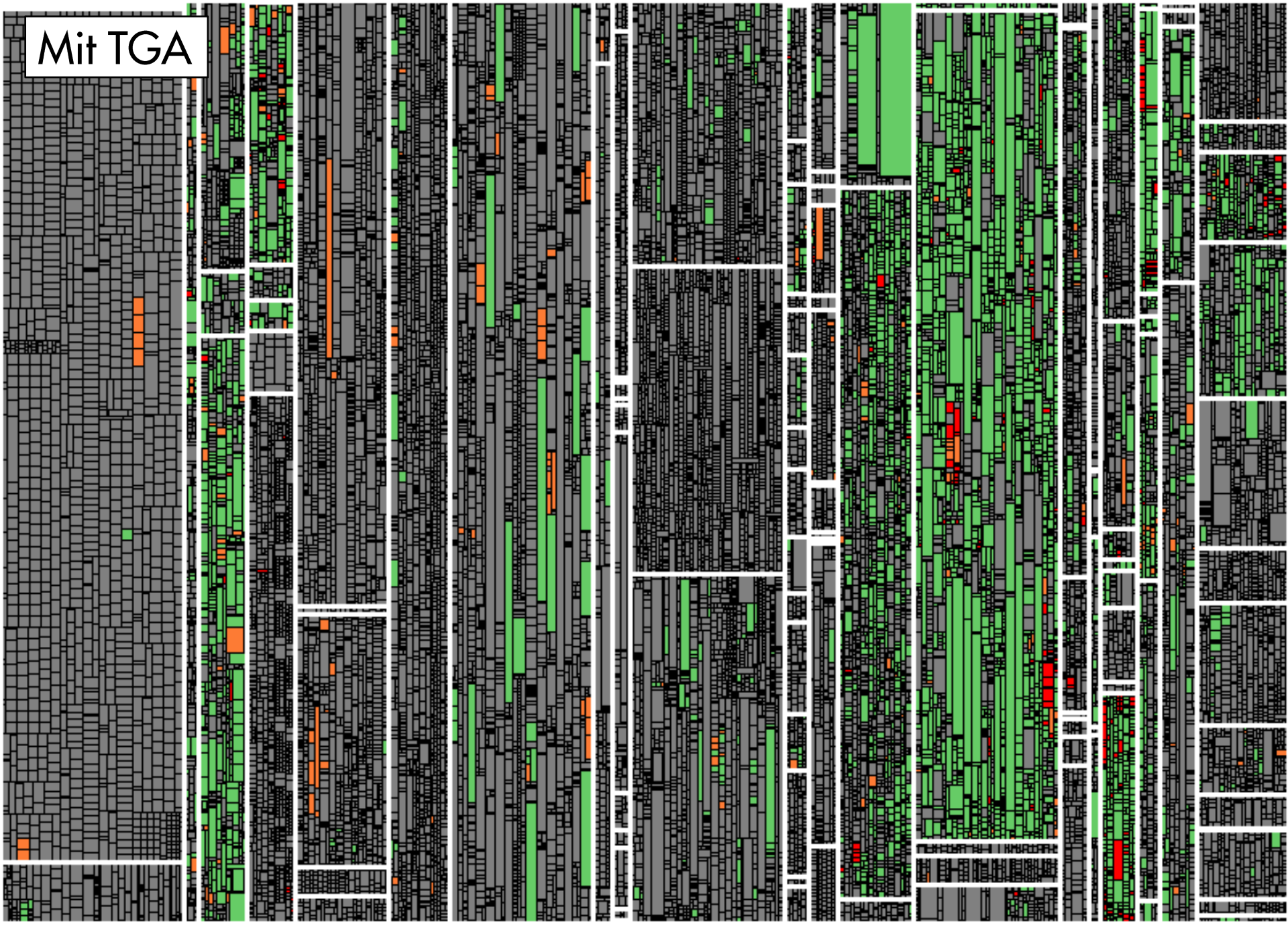
Portfolio-Dashboard

- Zentrale TGA-Informationen für alle Anwendungen auf einen Blick
- Direkte Links auf Projekt-Dashboards mit detaillierter Information
- Macht TGA-Dashboards für alle einfach zu finden
- Erhöht Transparenz über Anwendungen hinweg

Portfolio Overview

System	TGA	Technology	Test Gap
[illegible]	Link	.NET	71.4%
[illegible]	Link	ABAP/BW	28.7%
[illegible]	Link	.NET	38.7%
[illegible]	Link	.NET	5.9%
[illegible]	Link	.NET	7.6%
[illegible]	Link	.NET	11.7%
[illegible]		.NET	
[illegible]	Link	.NET	not yet run
[illegible]	Link	.NET	15.5%
[illegible]		OScript	
[illegible]	Link	ABAP	32.4%
[illegible]	Link	ABAP/BW	28.7%
[illegible]	Link	.NET	10.7%
[illegible]	Link	.NET	42.8%
[illegible]	Link	ABAP	51.1%
[illegible]		JavaScript(DB)	
[illegible]	Link	.NET	not run
[illegible]			
[illegible]			
[illegible]	Link	.NET	50.2%
[illegible]	Link	.NET	41.1%
[illegible]	Link	.NET	93.4%
[illegible]	Link	ABAP/BW	2.8%
[illegible]			
[illegible]	Link	.NET	15.6%
[illegible]	Link	.NET	30.2%
[illegible]	Link	.NET	15.6%
[illegible]	Link	ABAP/BW	28.7%
[illegible]	Link	.NET	5.6%
[illegible]		SAP BPM	


Ohne TGA


Mit TGA

Lessons Learned

Test-Gap-Analyse schafft Transparenz.

Teams vermeiden damit ungewollt ungetesteten Code.

Test-Arten

Hotfix-Test

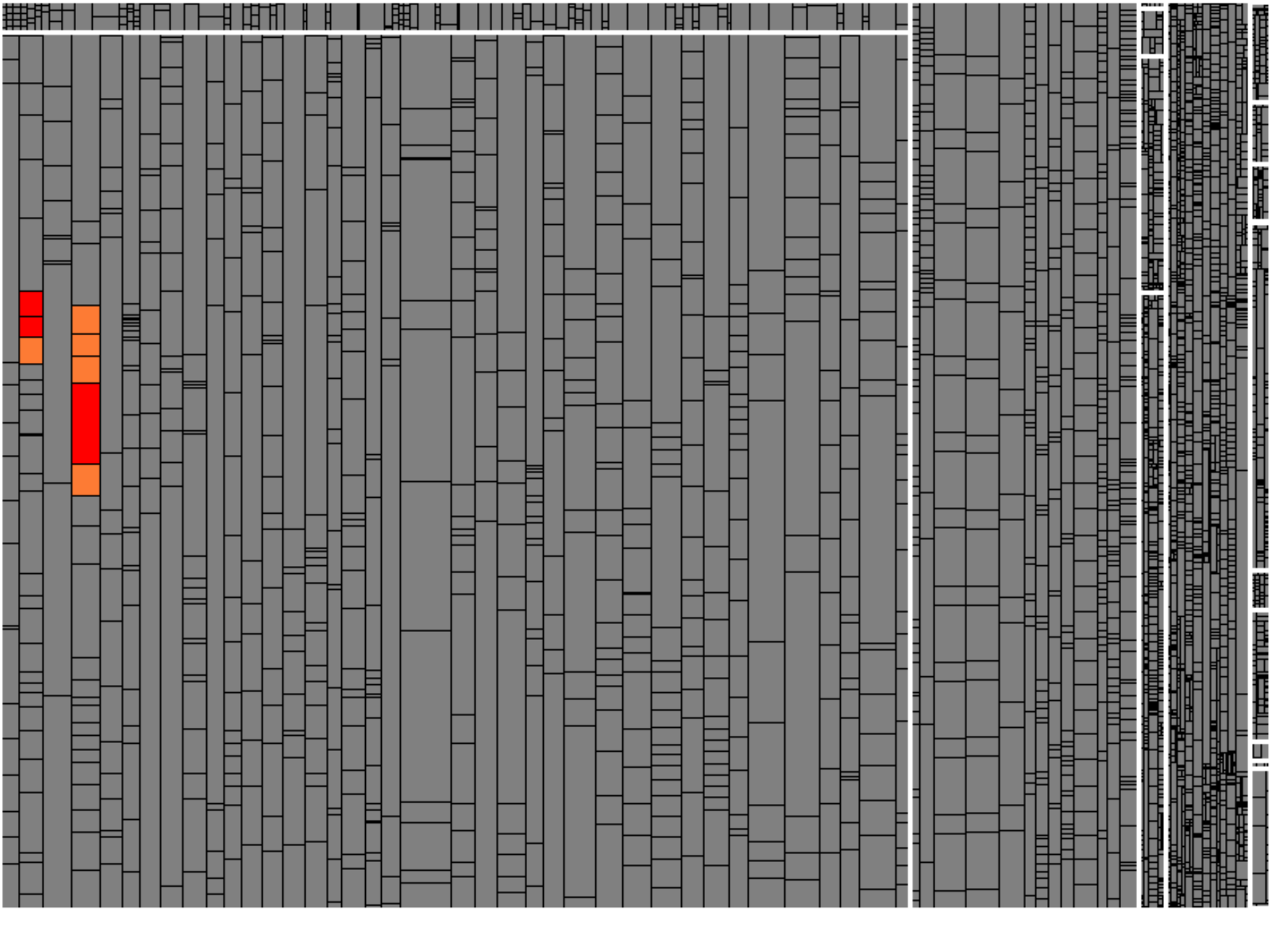
- On-Demand, wenig Zeit
- Getestet werden nur Änderungen des Hotfixes

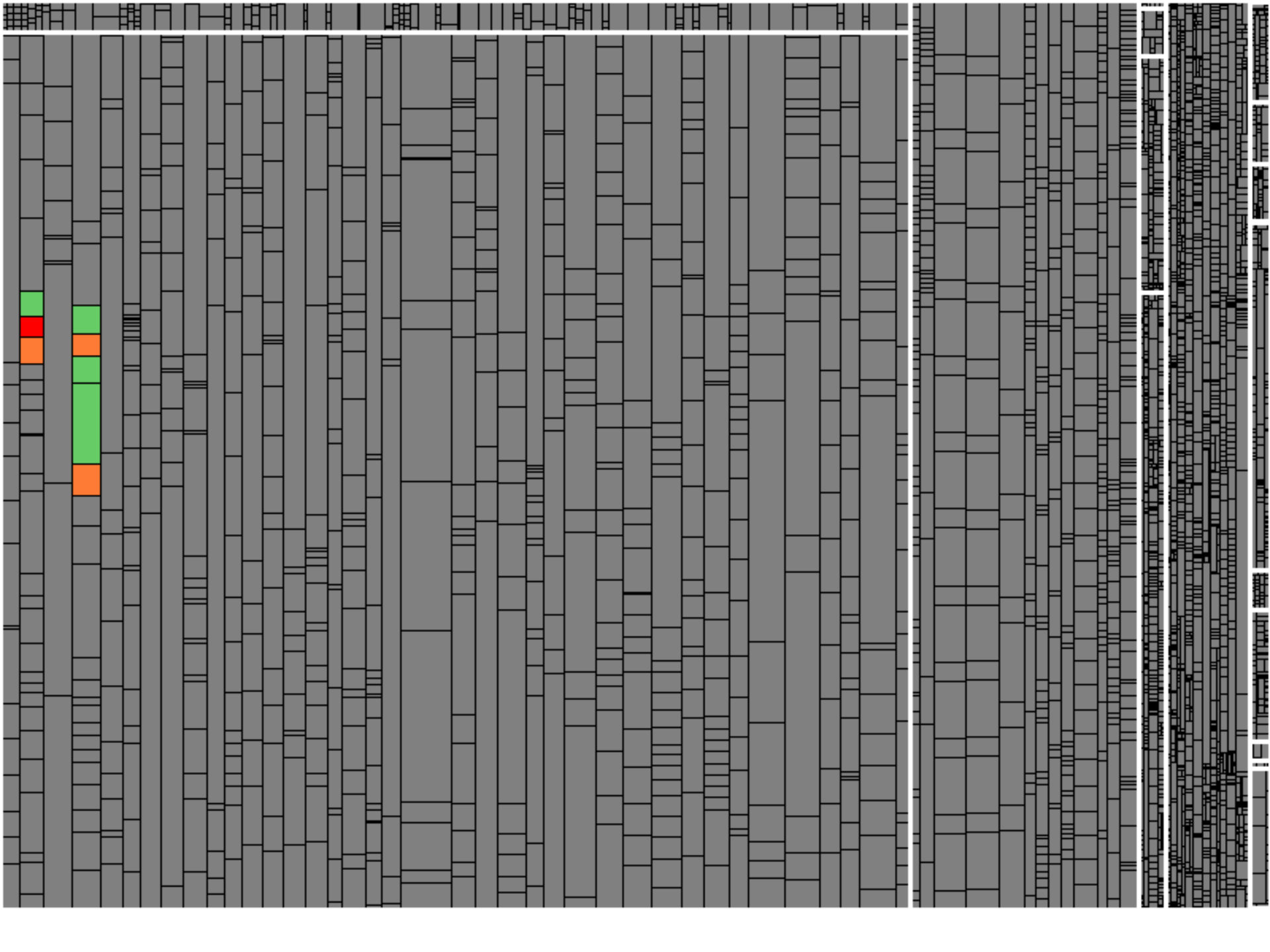
Release-Test

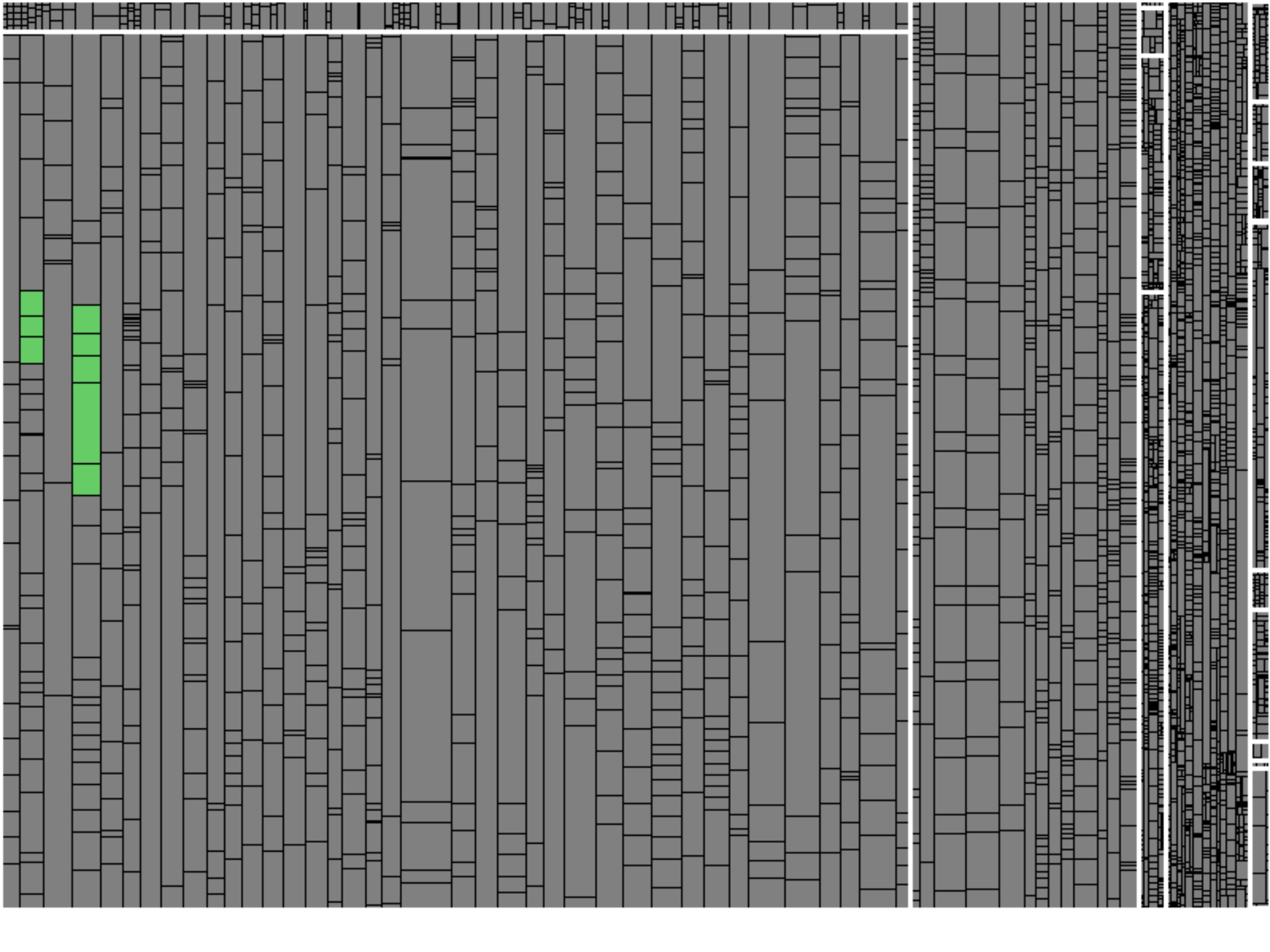
- Nur vor einem Release, Dauer: mehrere Wochen
- Getestet werden alle Änderungen seit letztem Release
- 2-4 Mal pro Jahr (abh. Von Release-Zyklus der Anwendung)

Iterations-Test

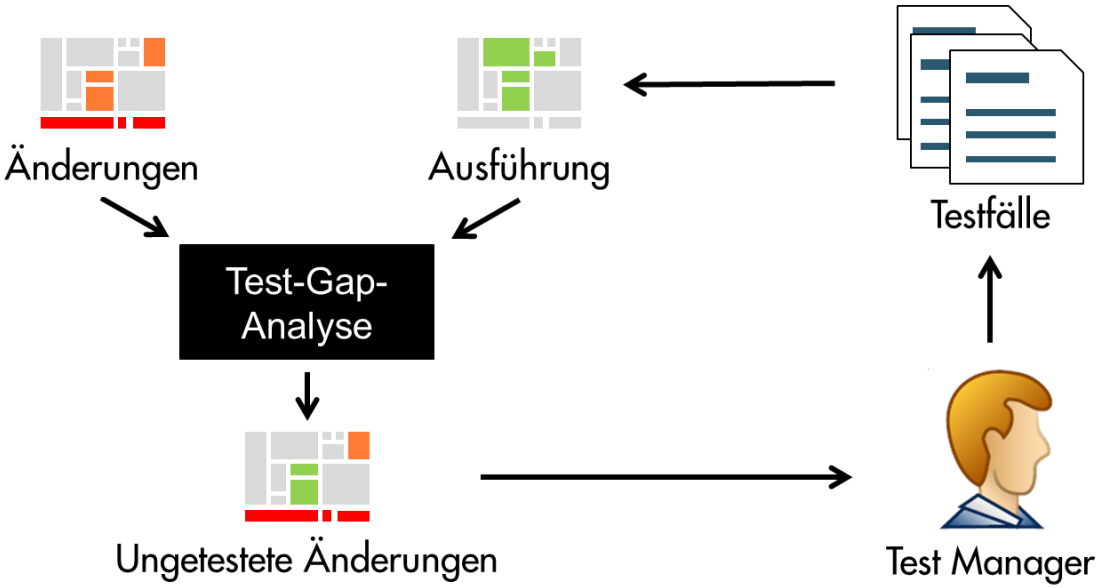
- Entwicklungsbegleitend, kontinuierlich, Iterationsdauer (2-4 Wochen)
- Getestet werden Änderungen aus der Iteration

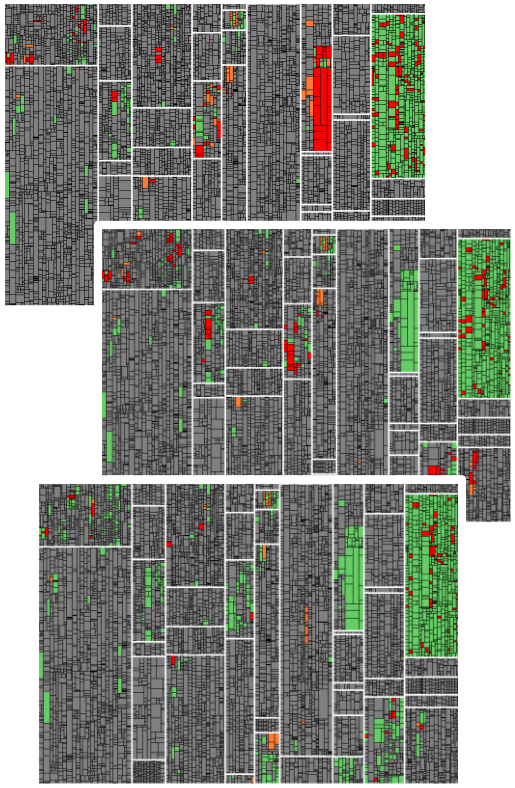


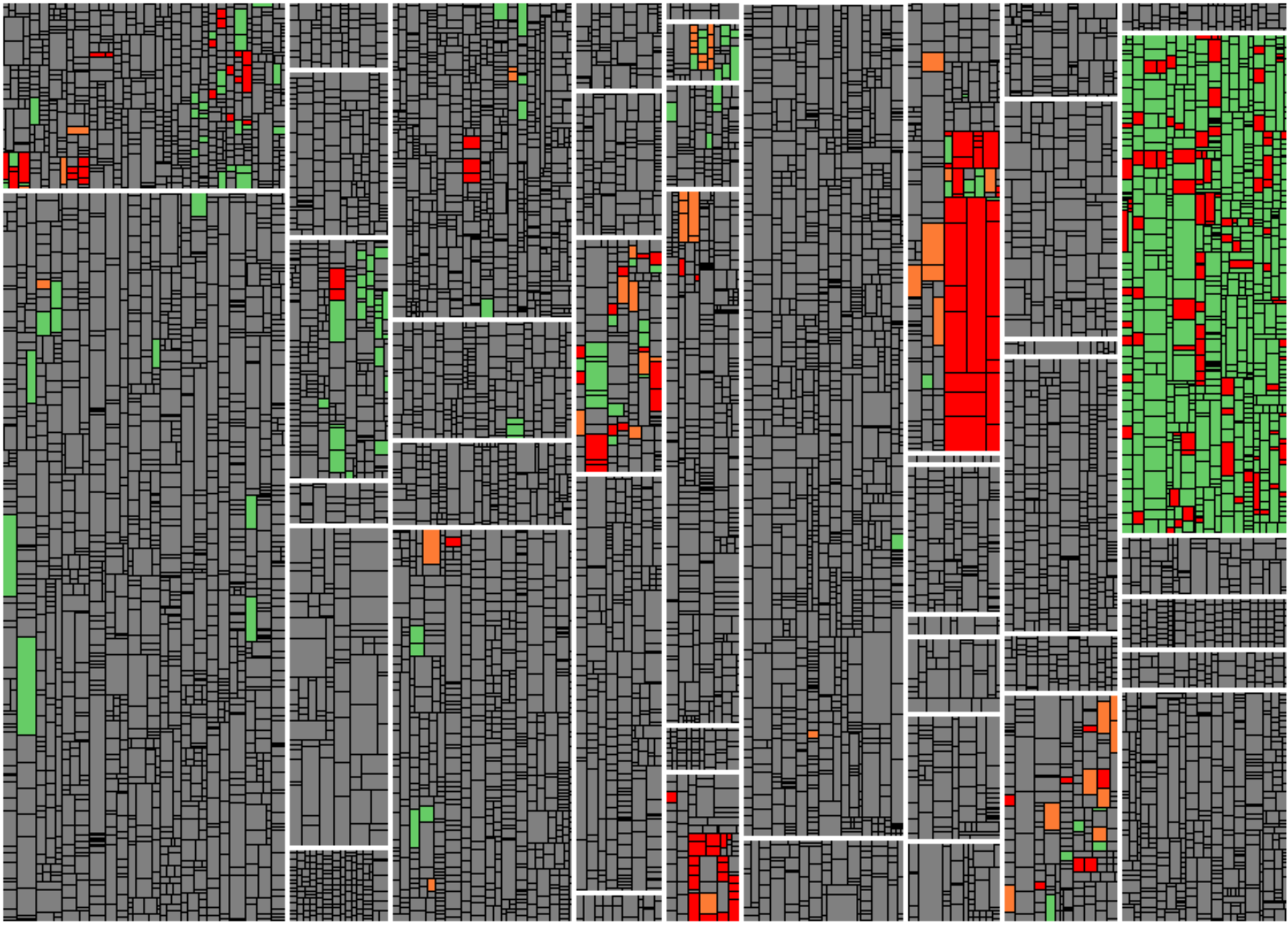


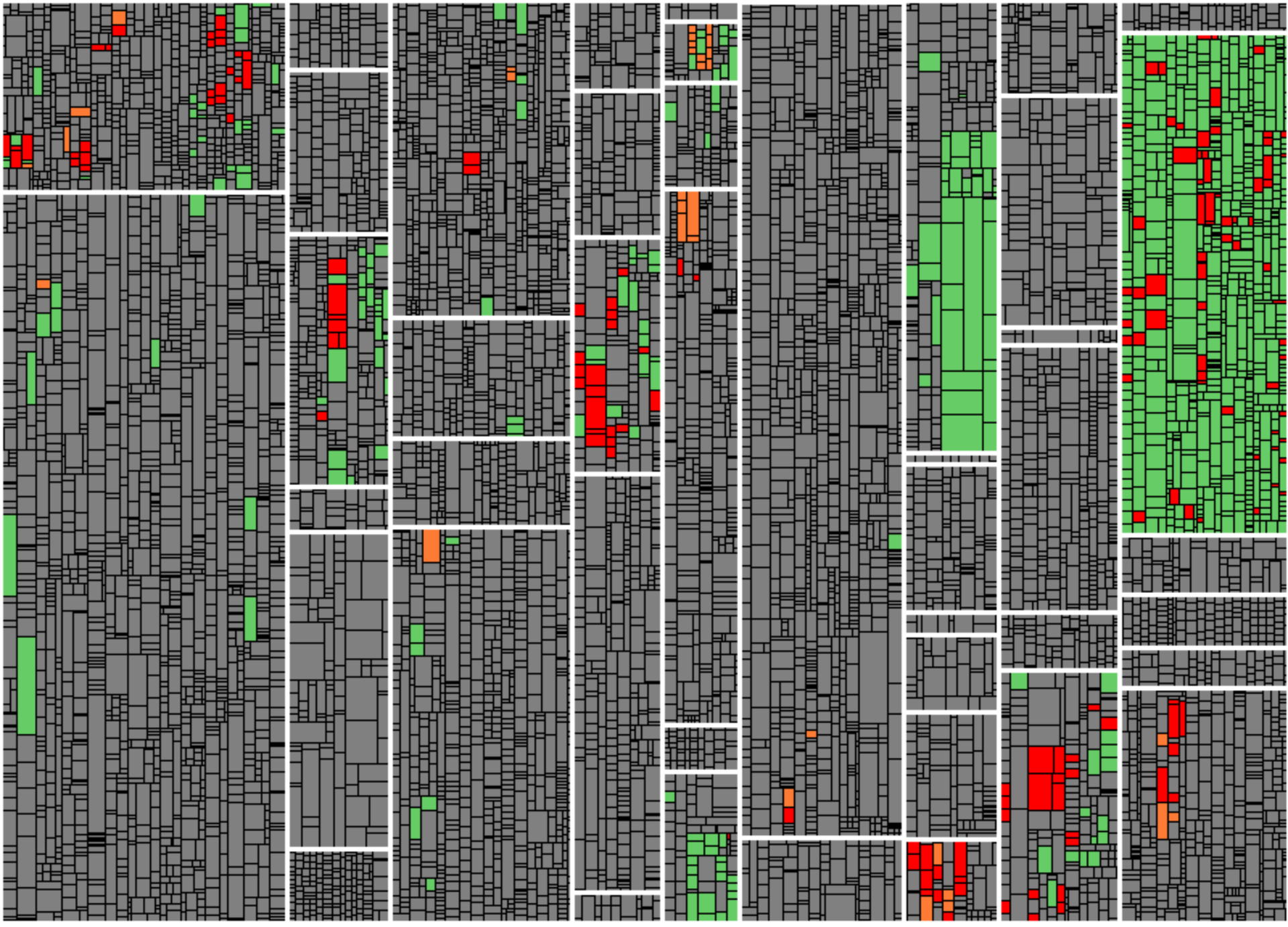


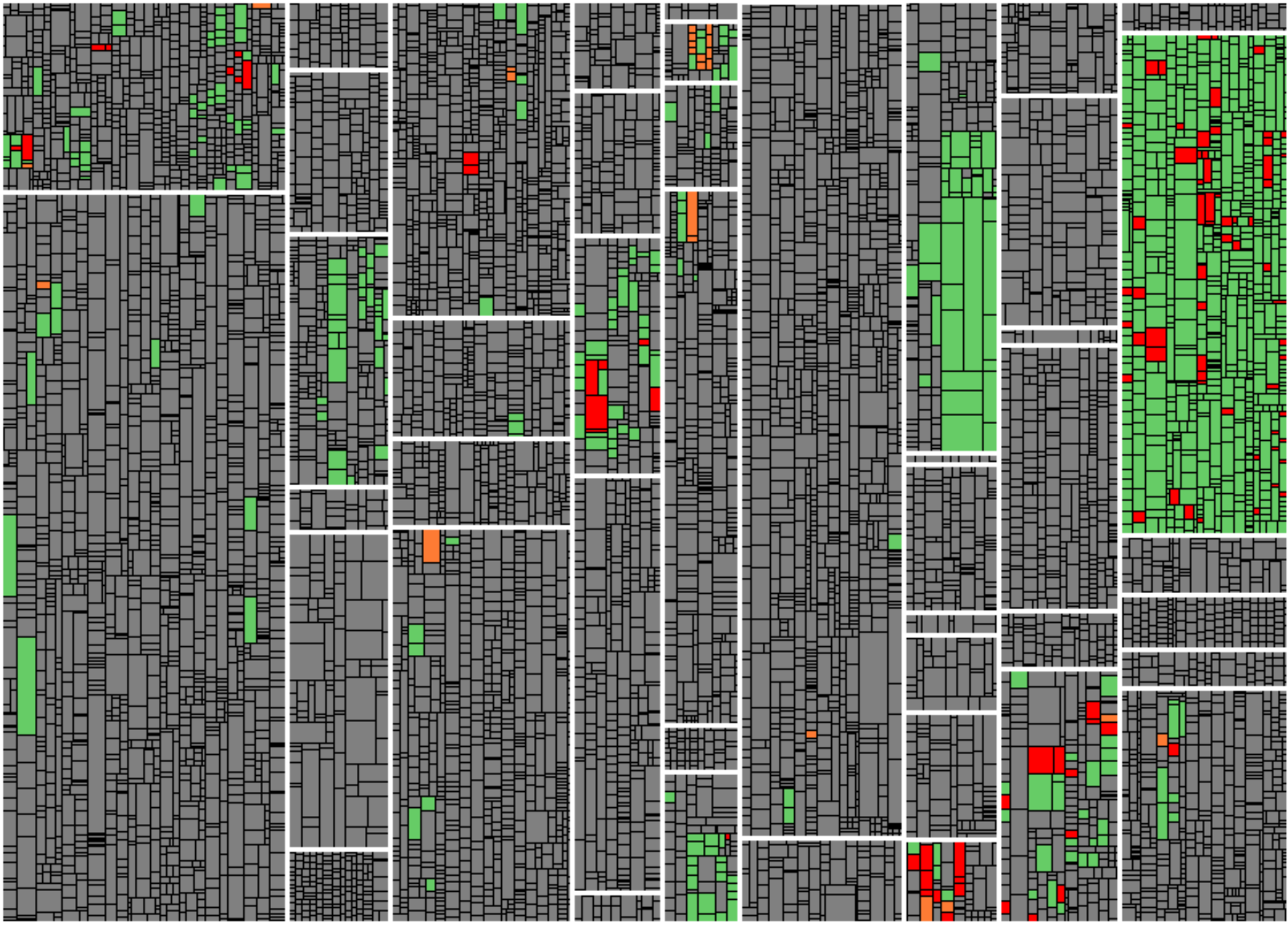
Release-Test: Beispiel (1)

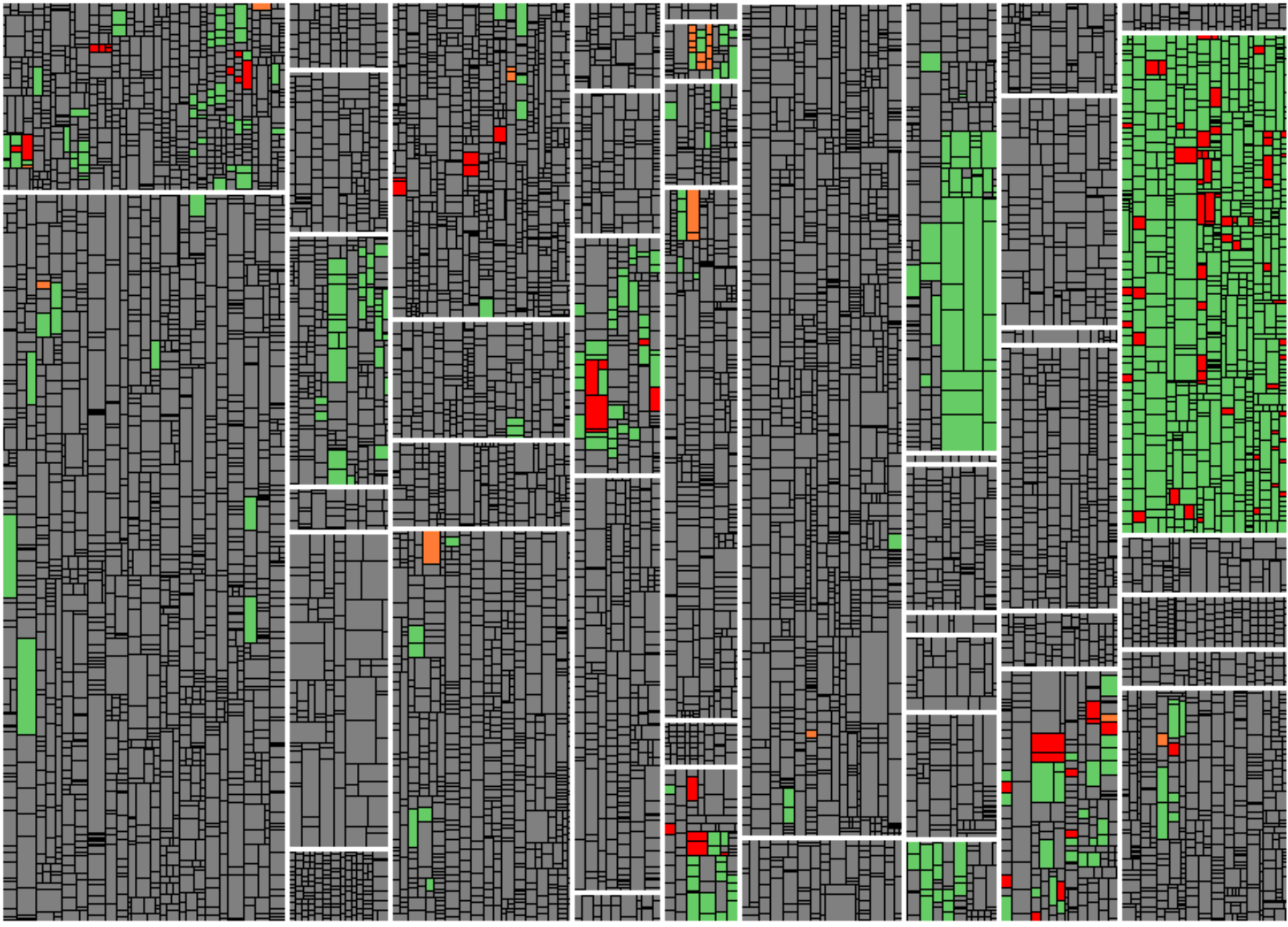


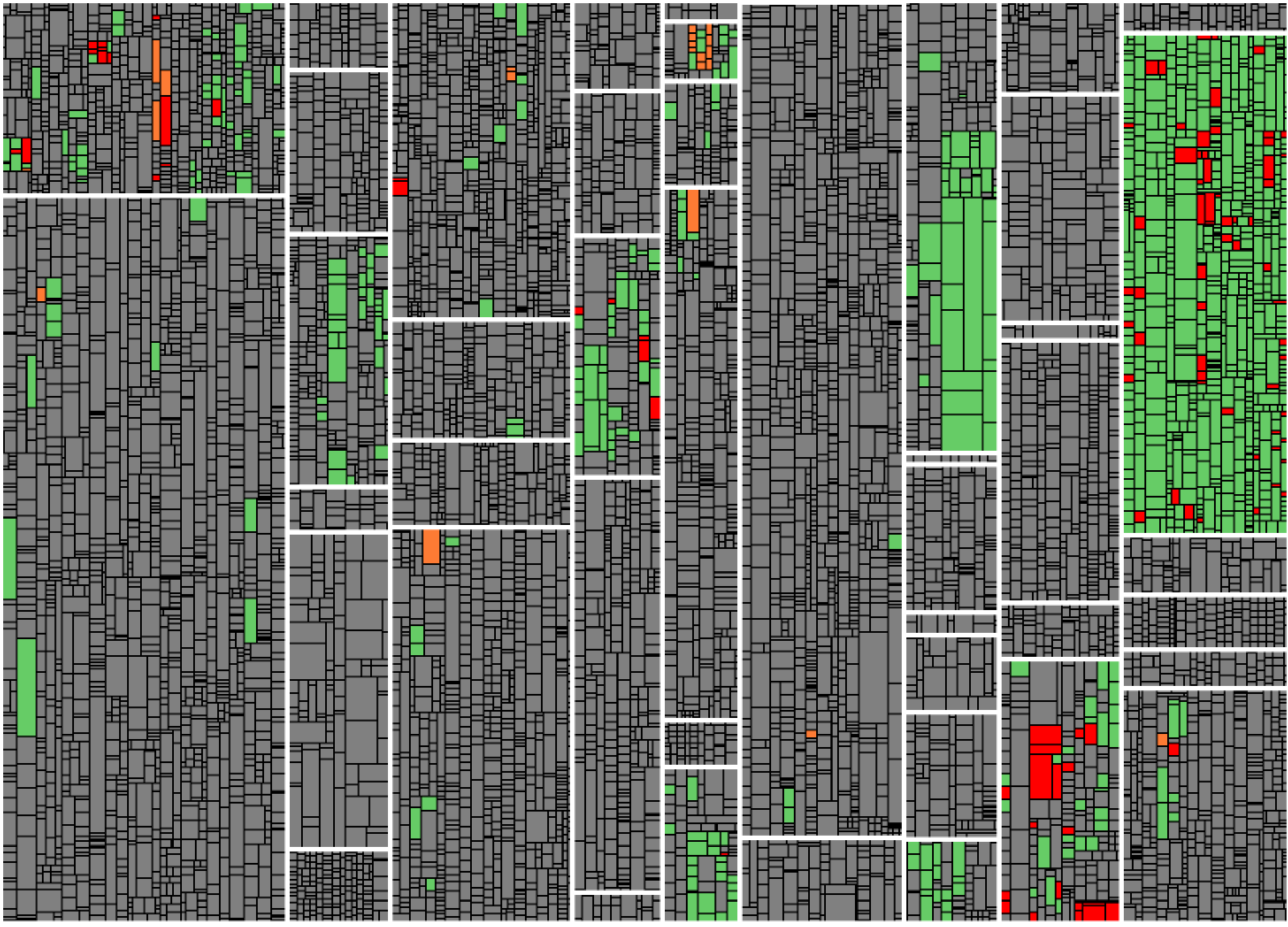


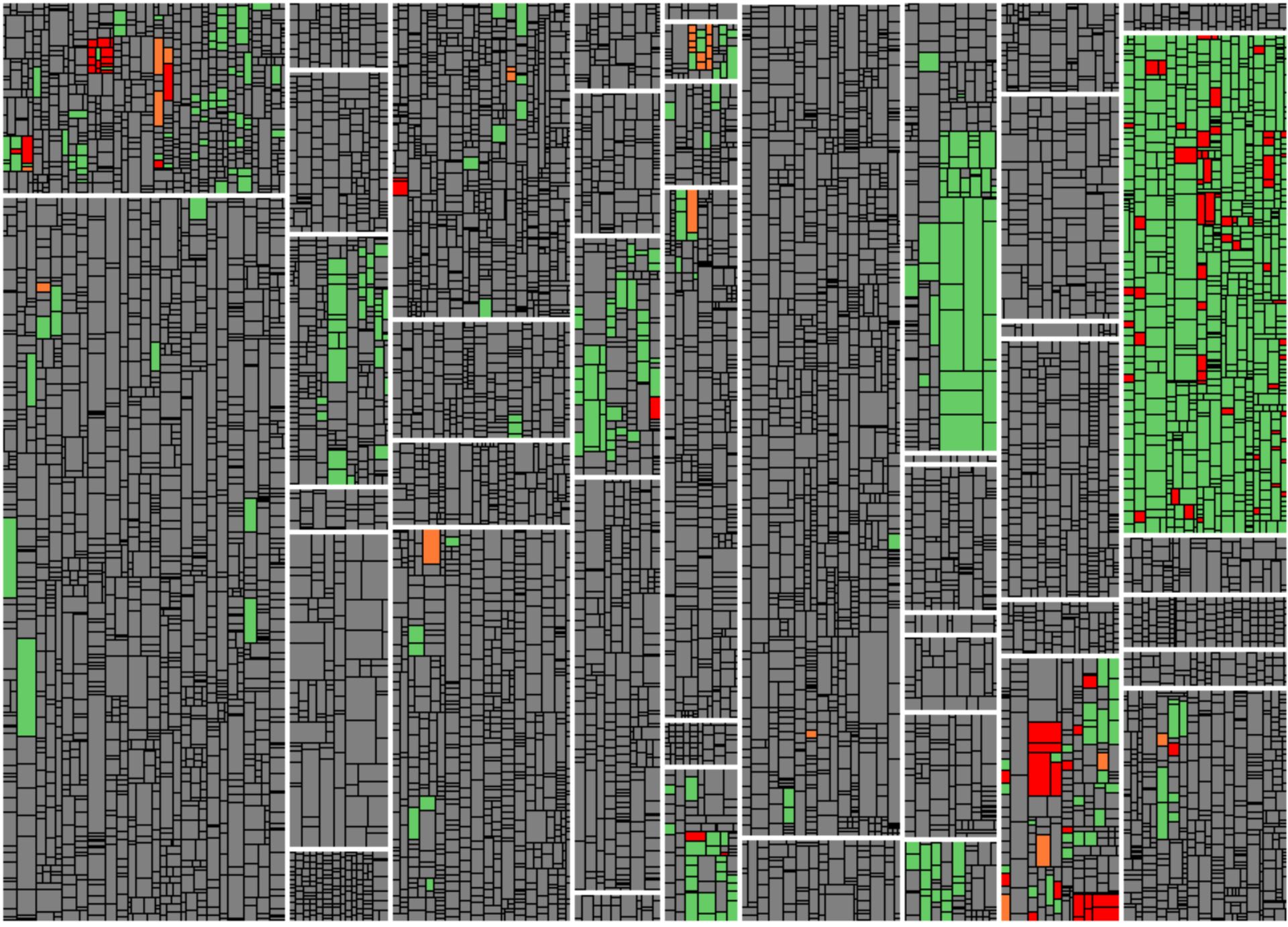


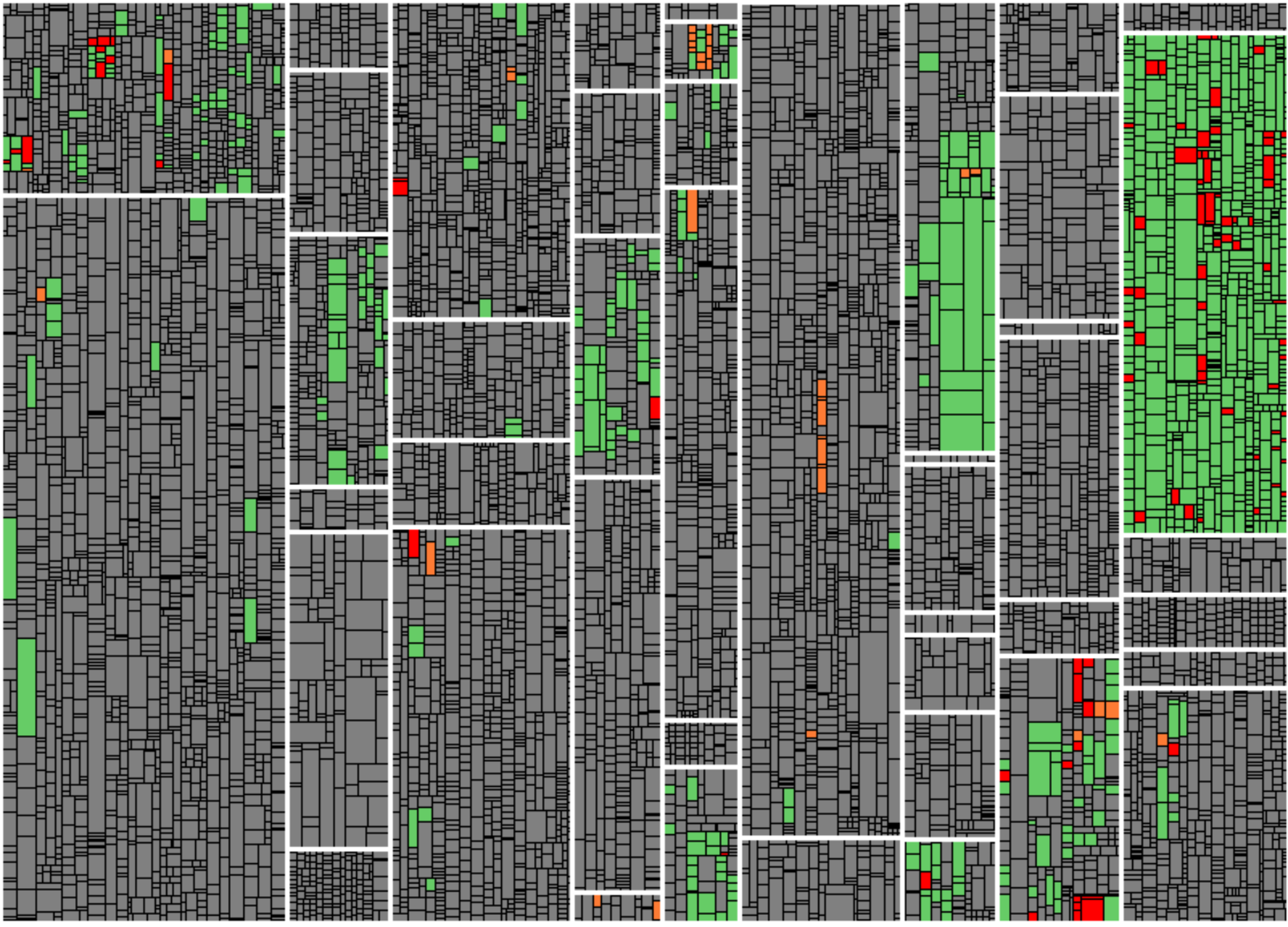


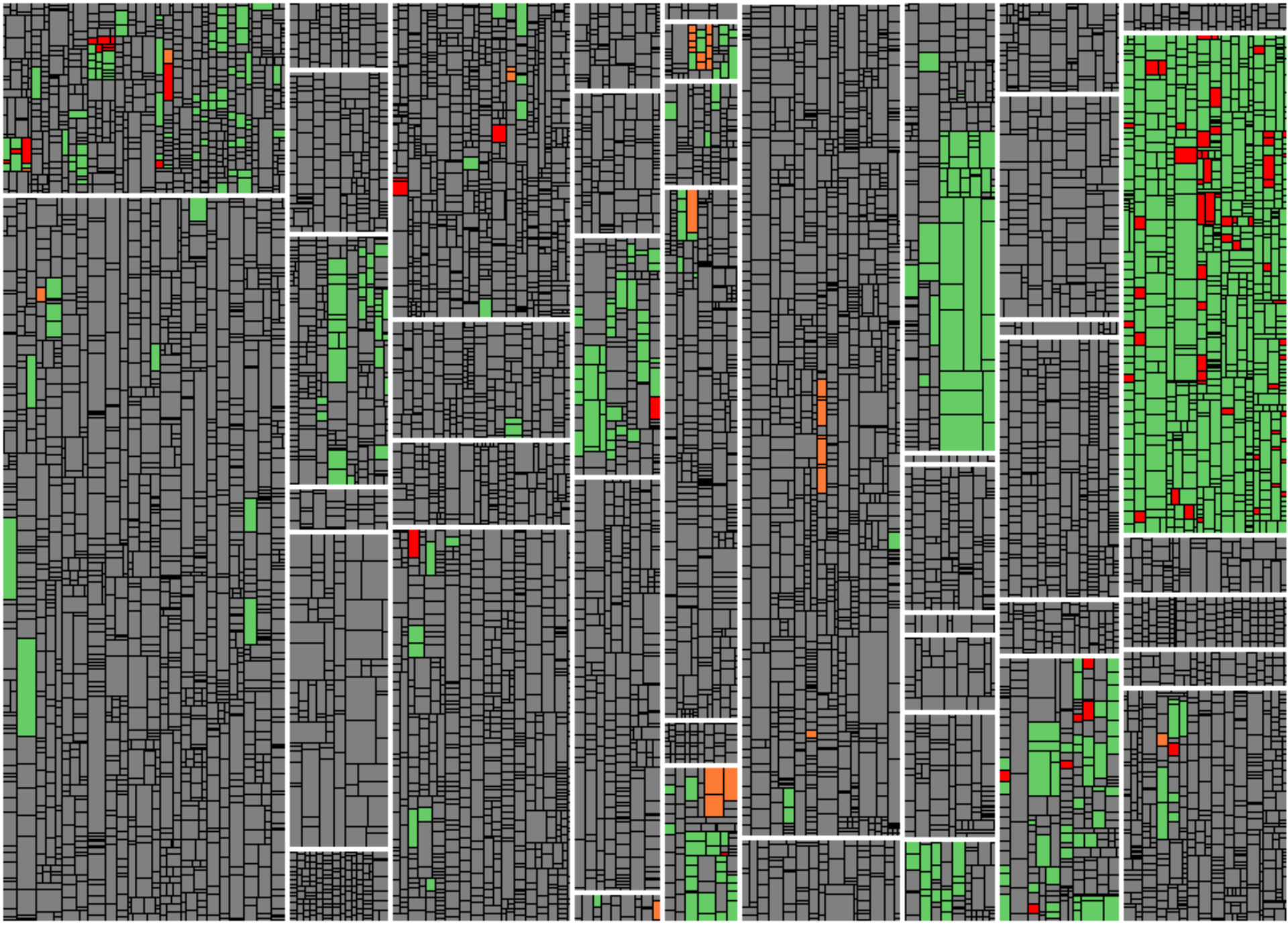




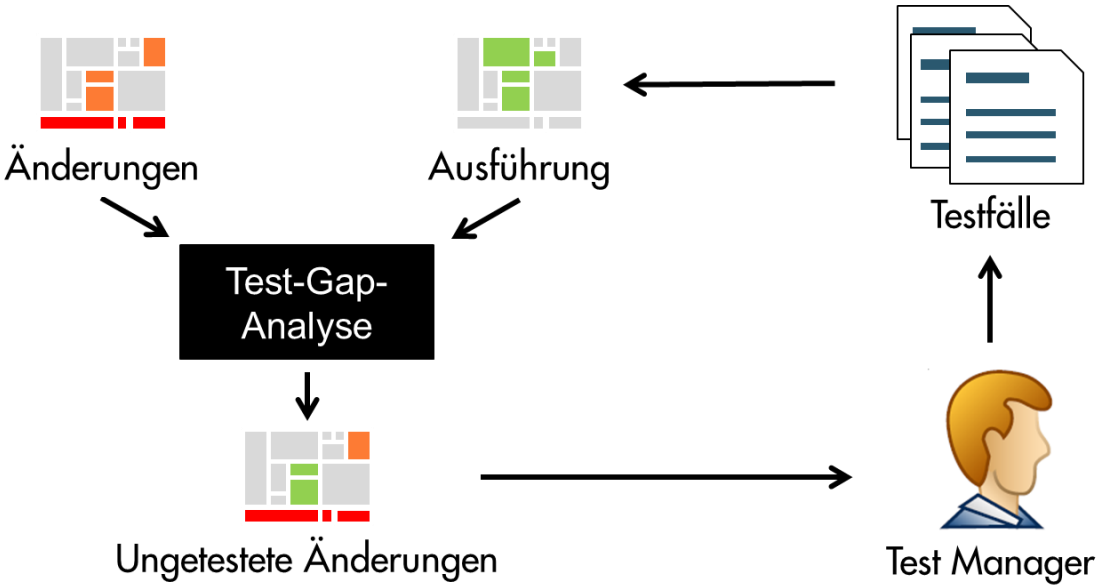


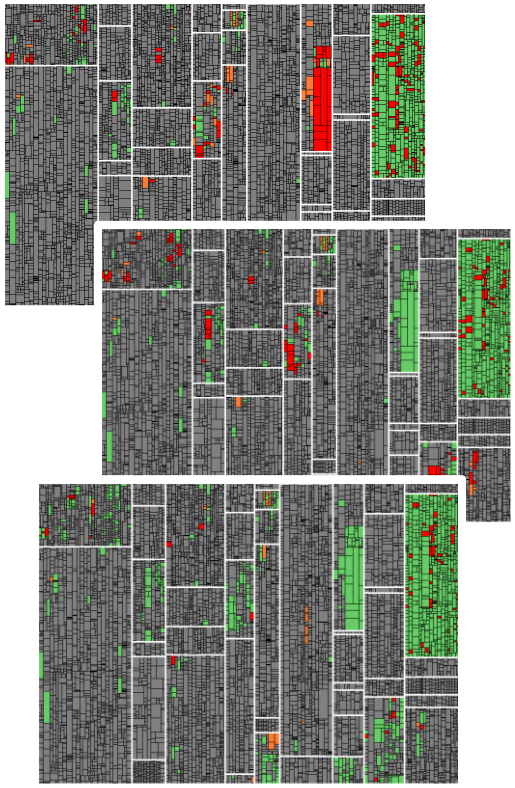


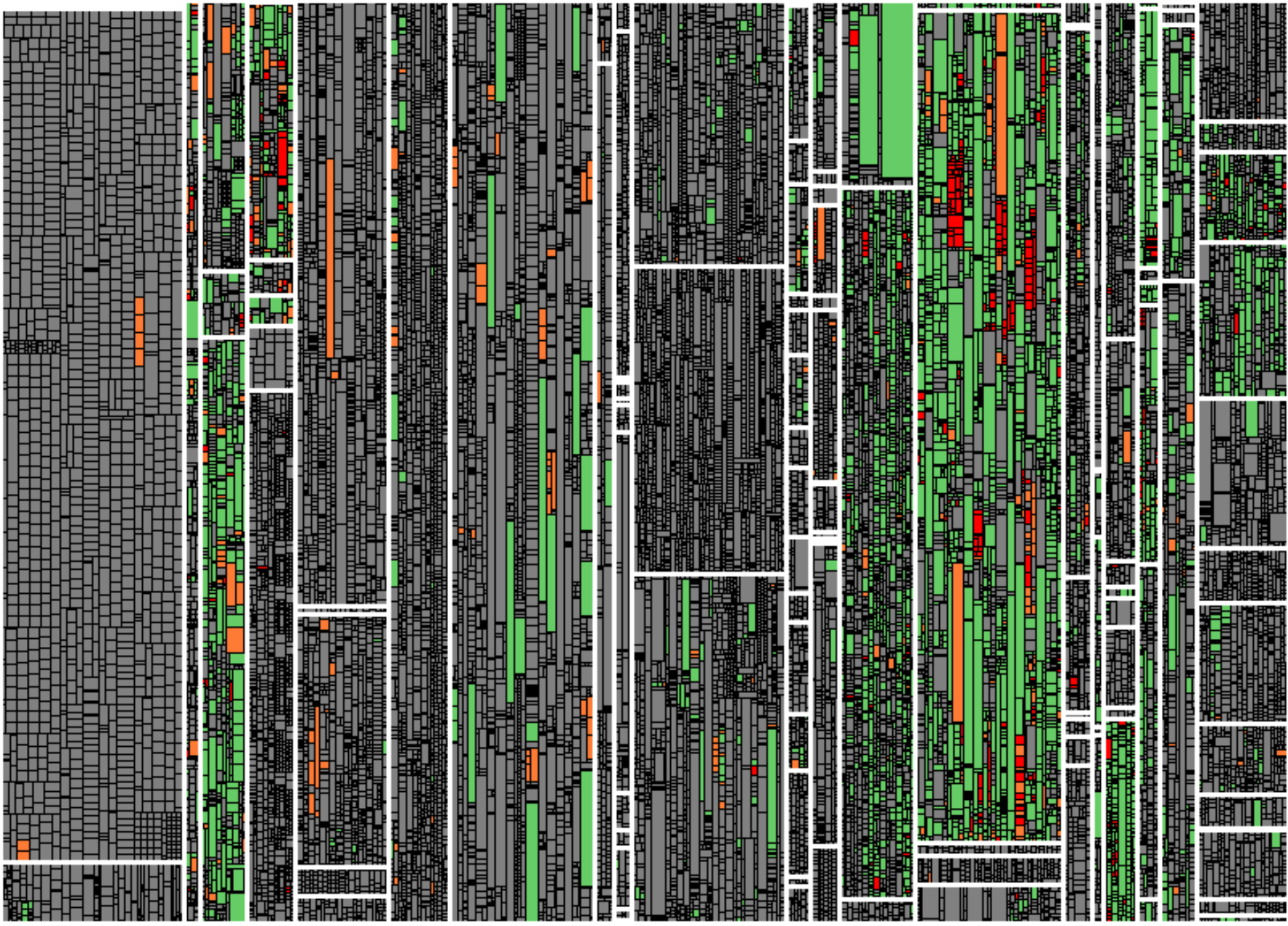


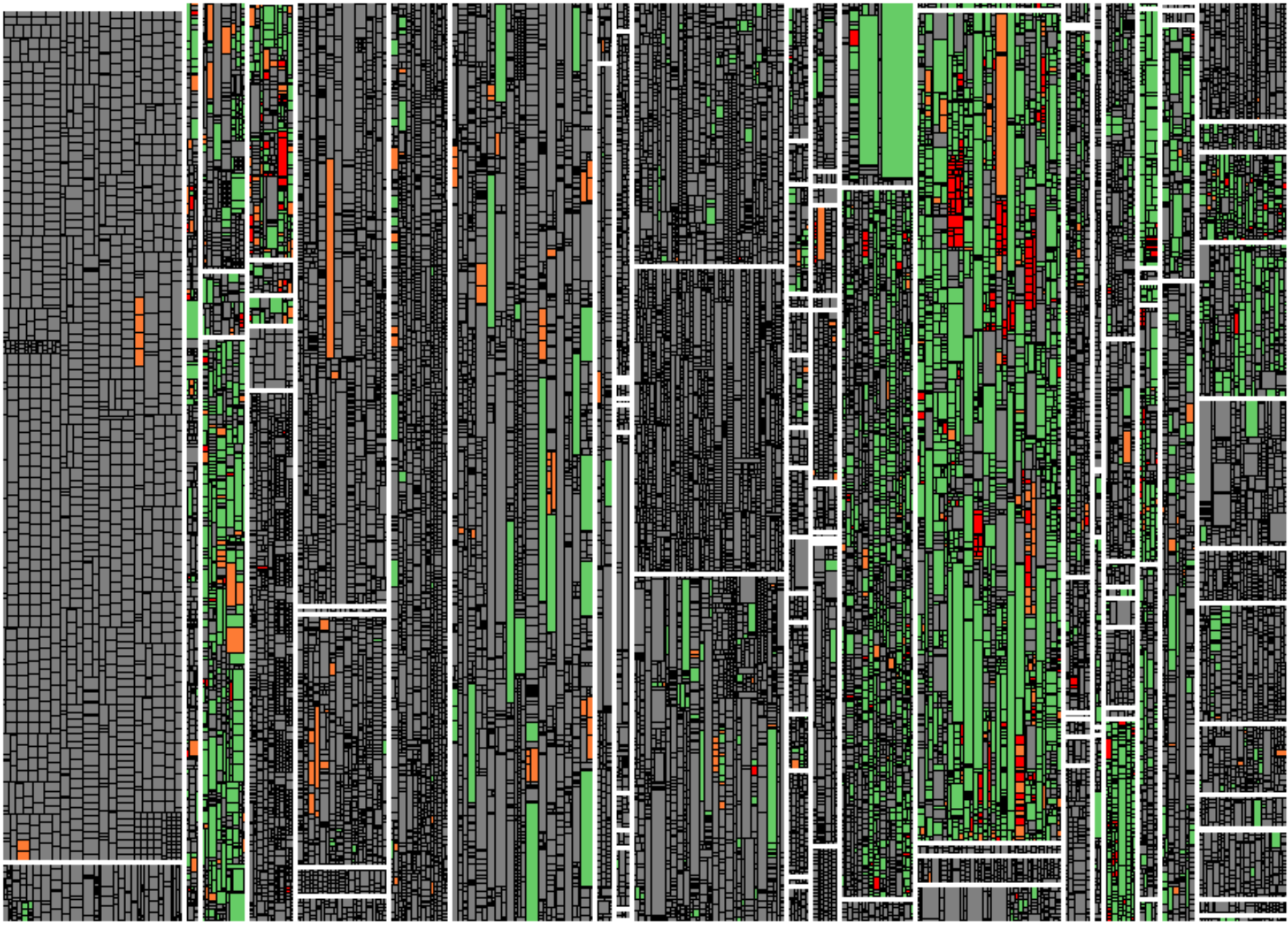


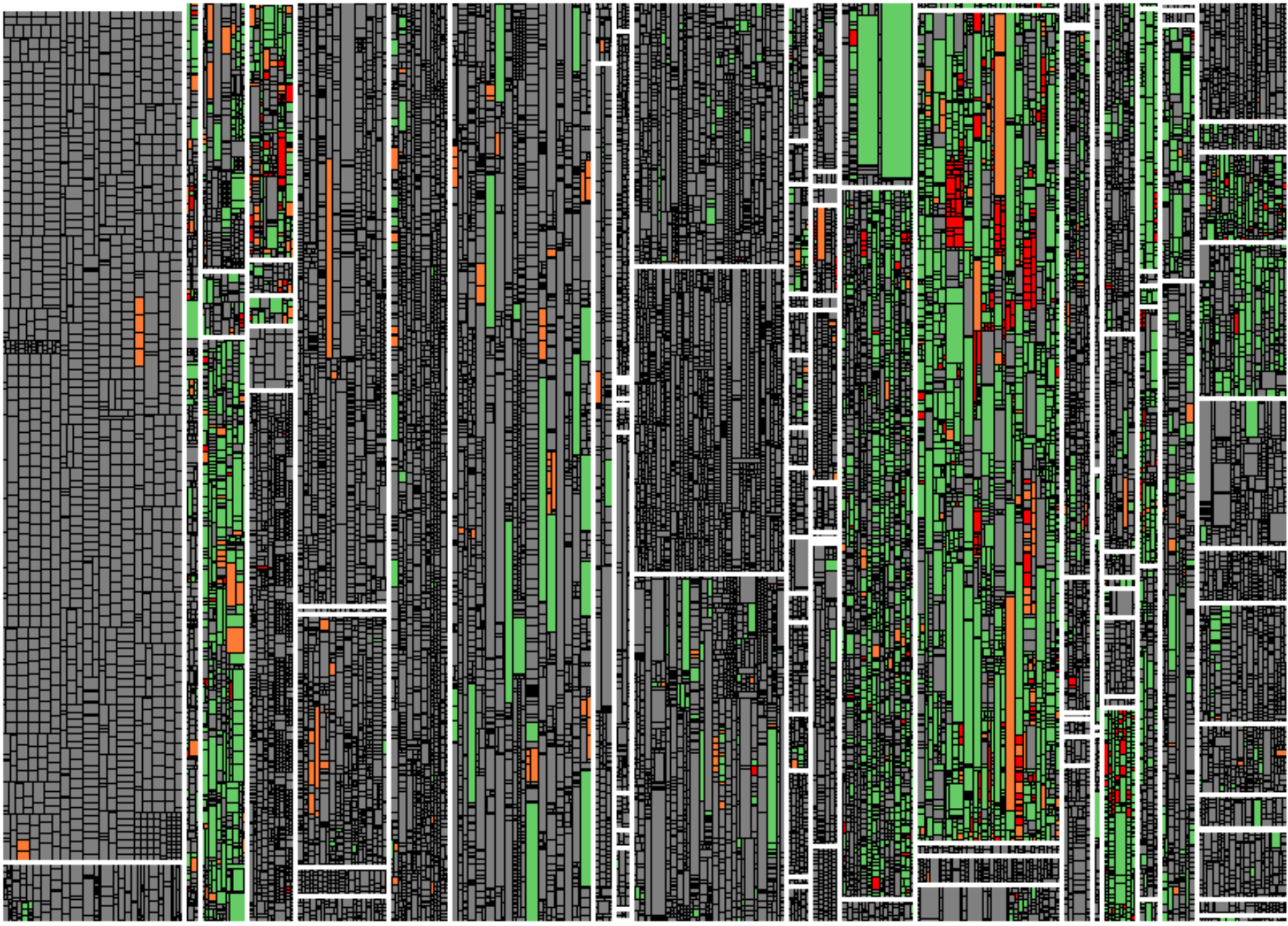
Release-Test: Beispiel (2)

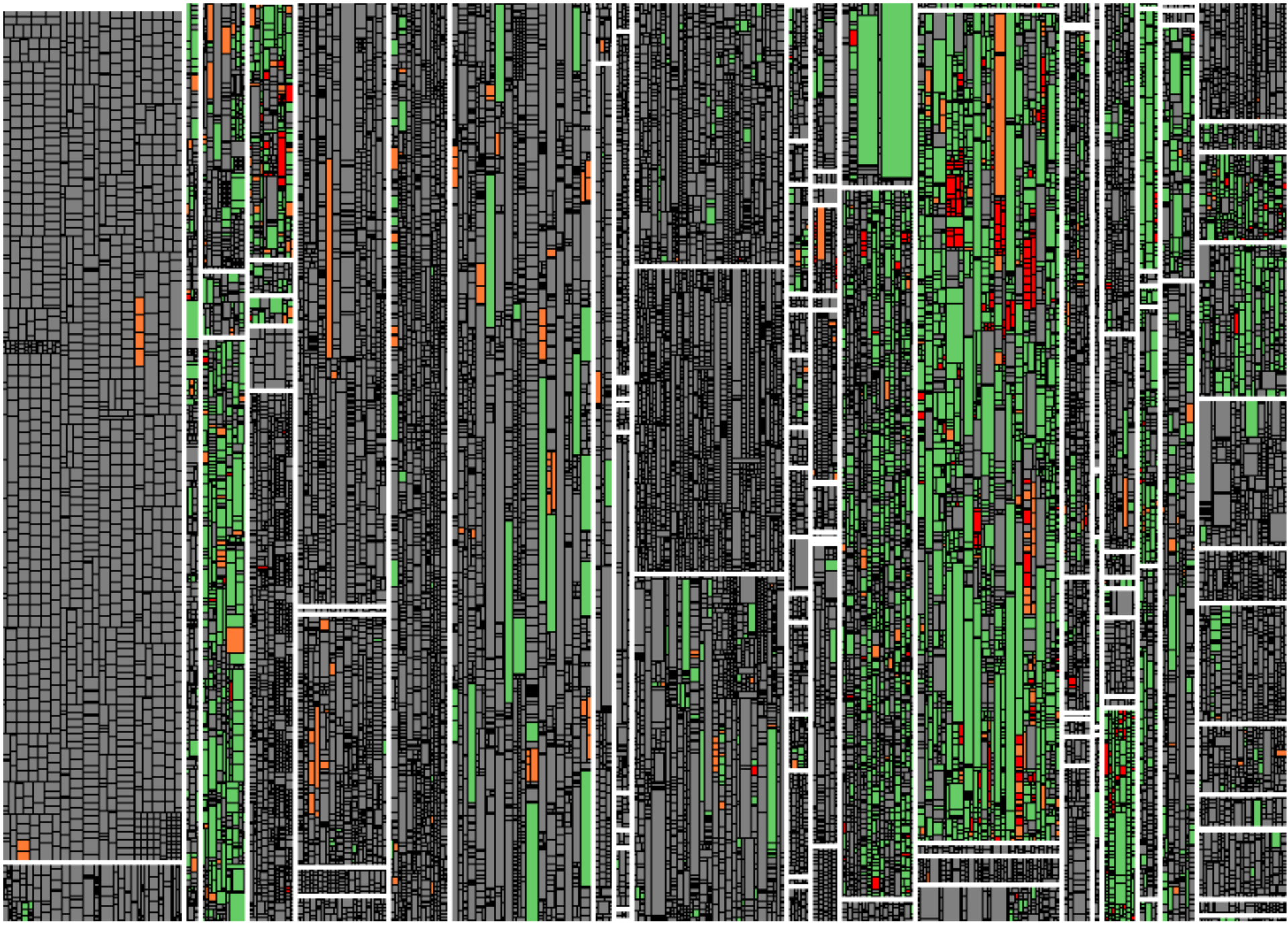


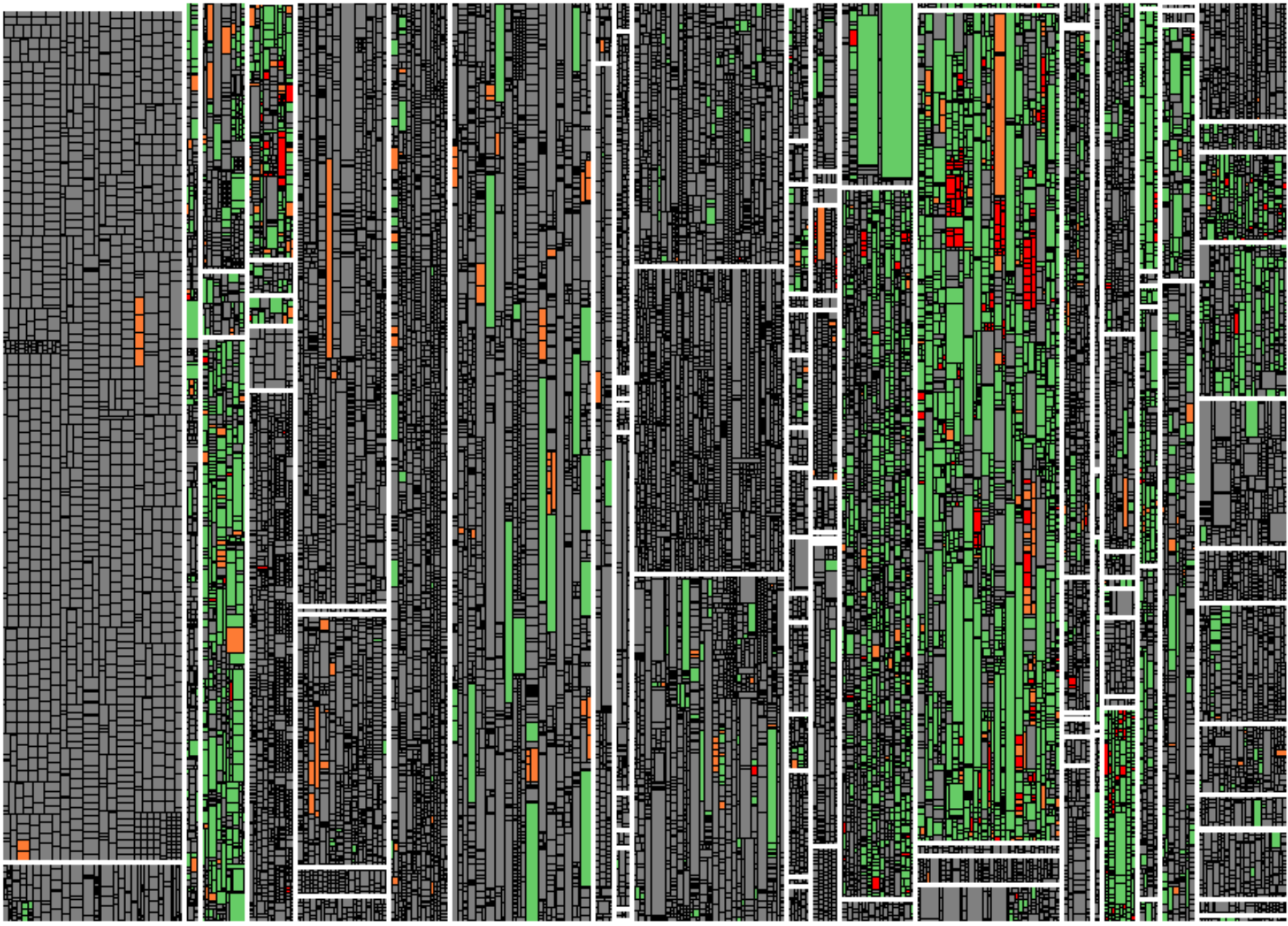


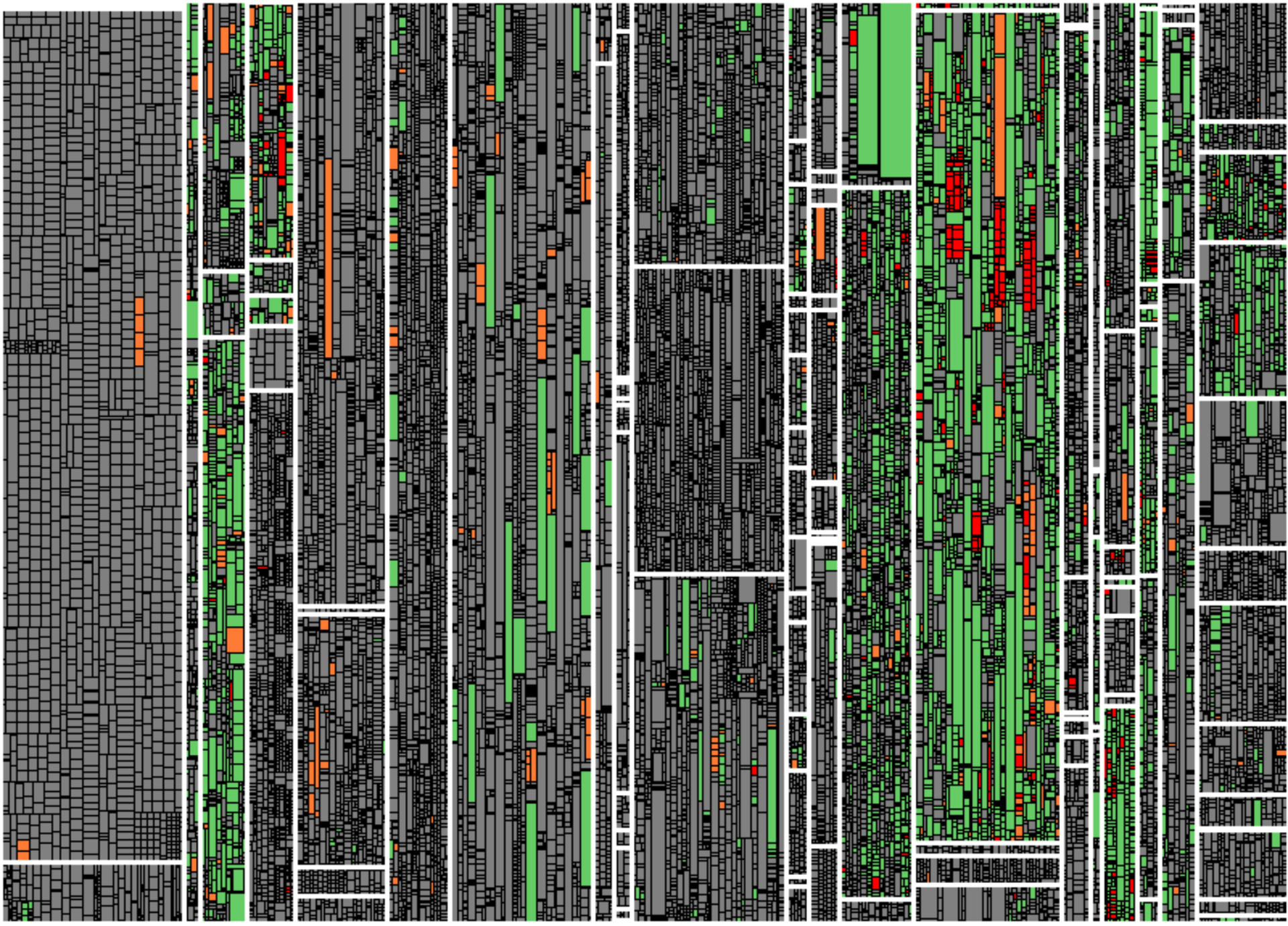


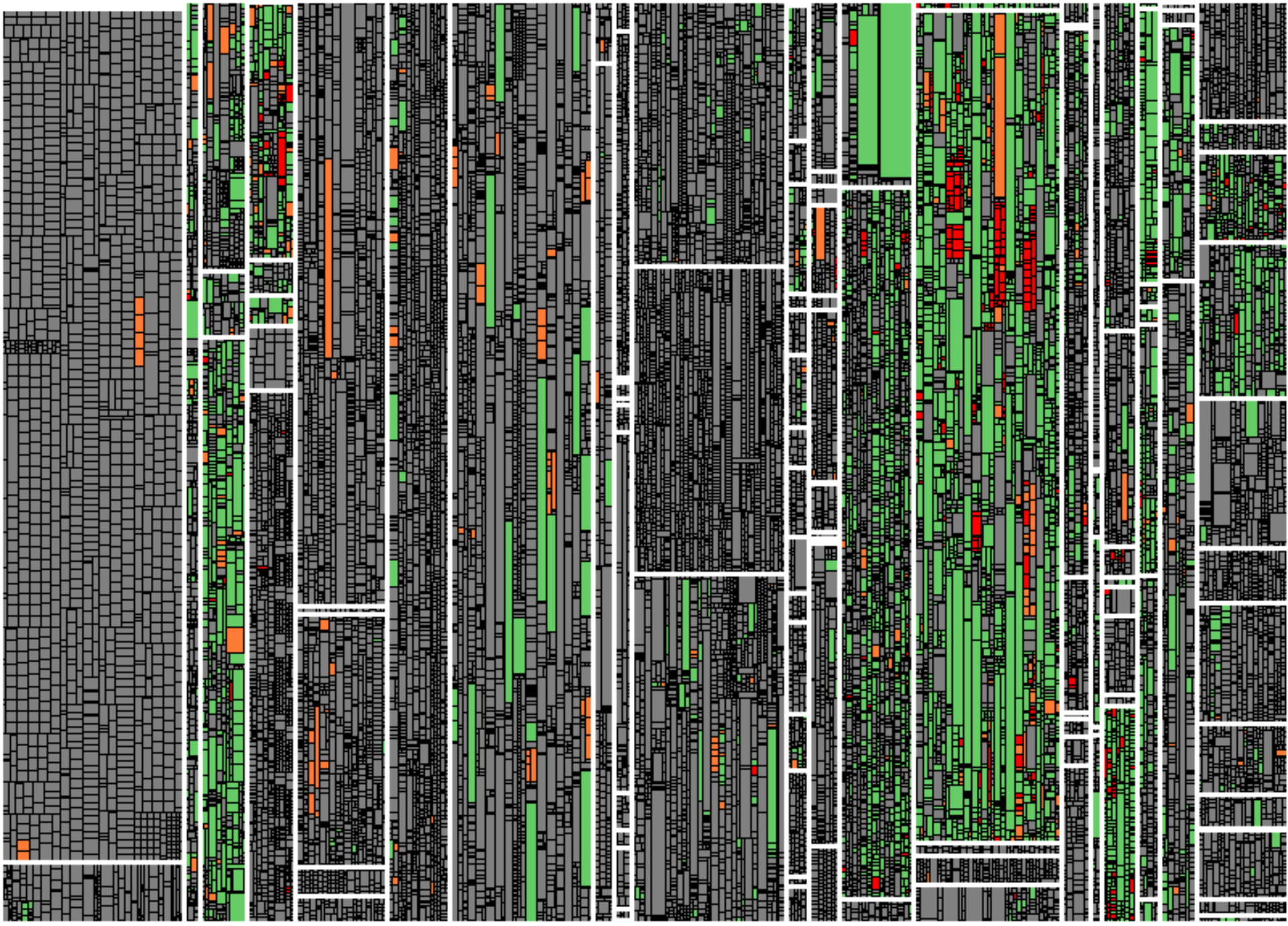


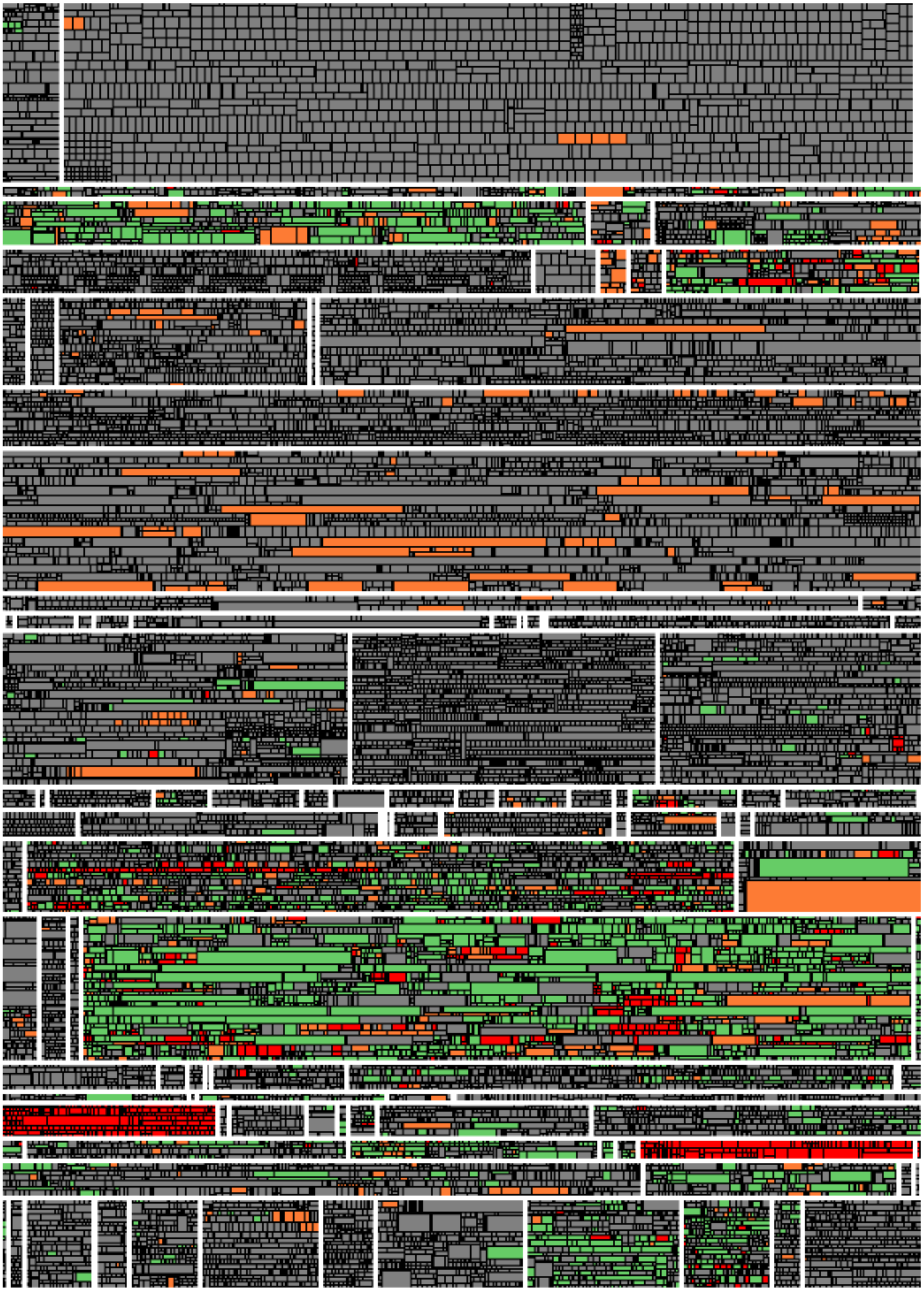


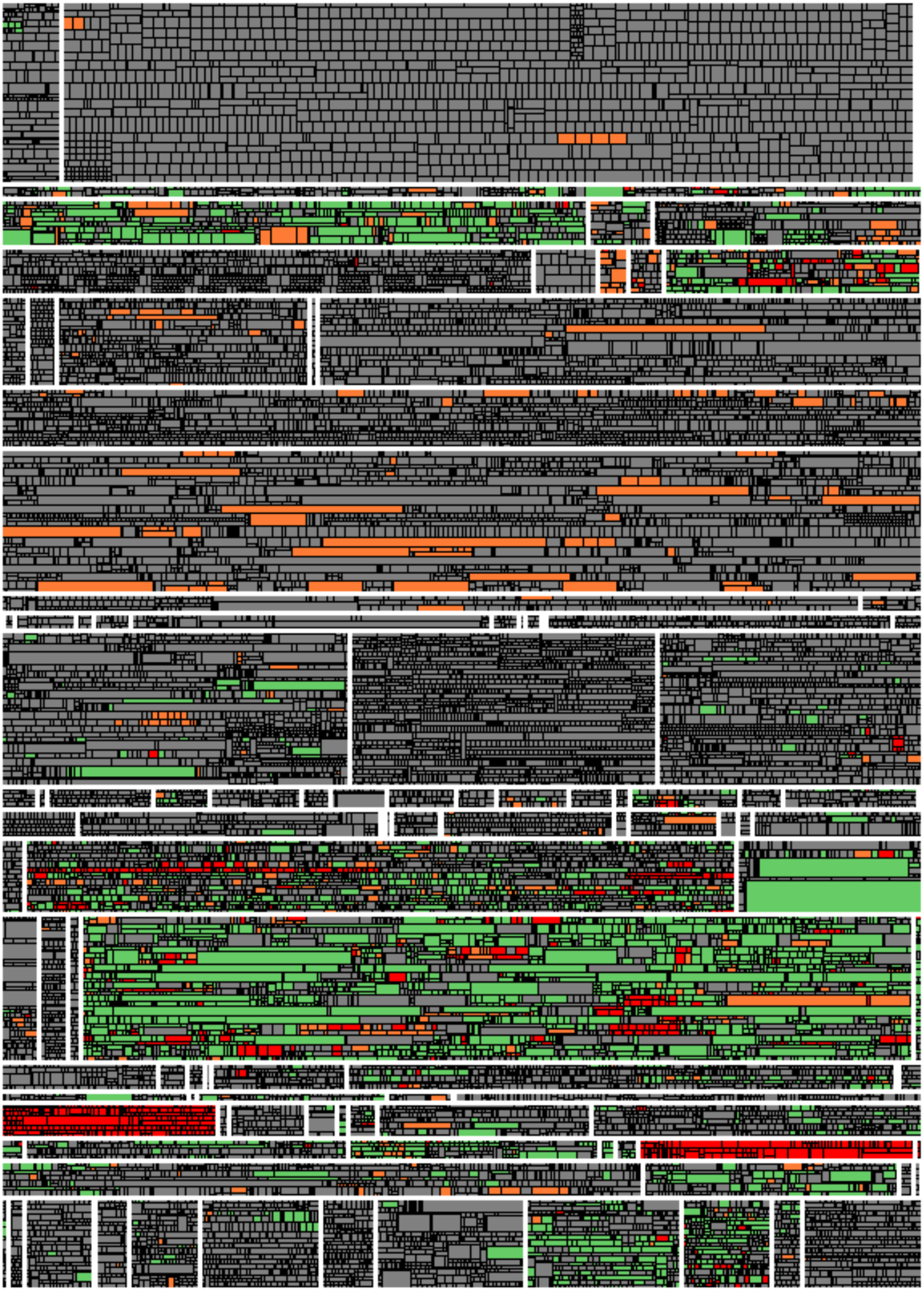


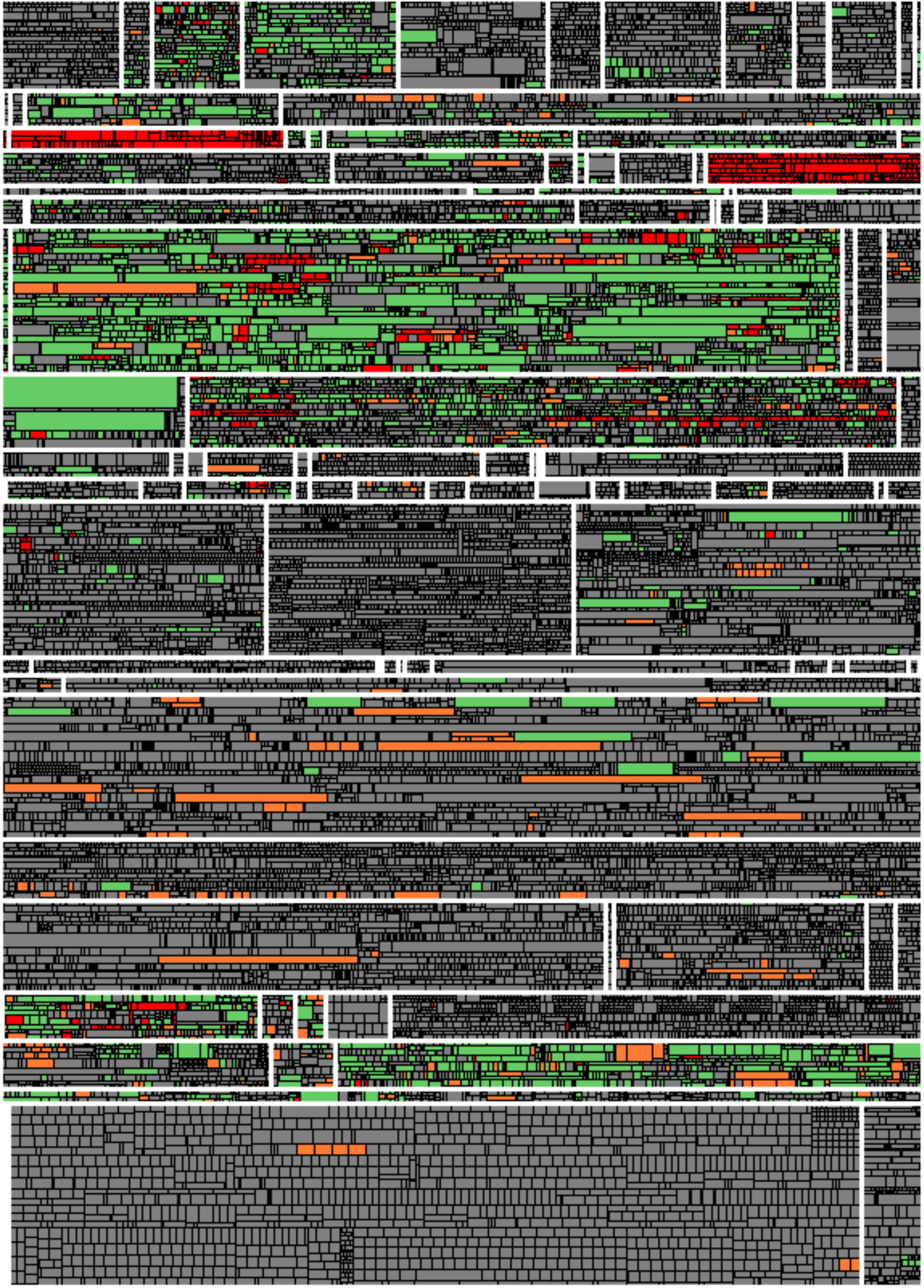


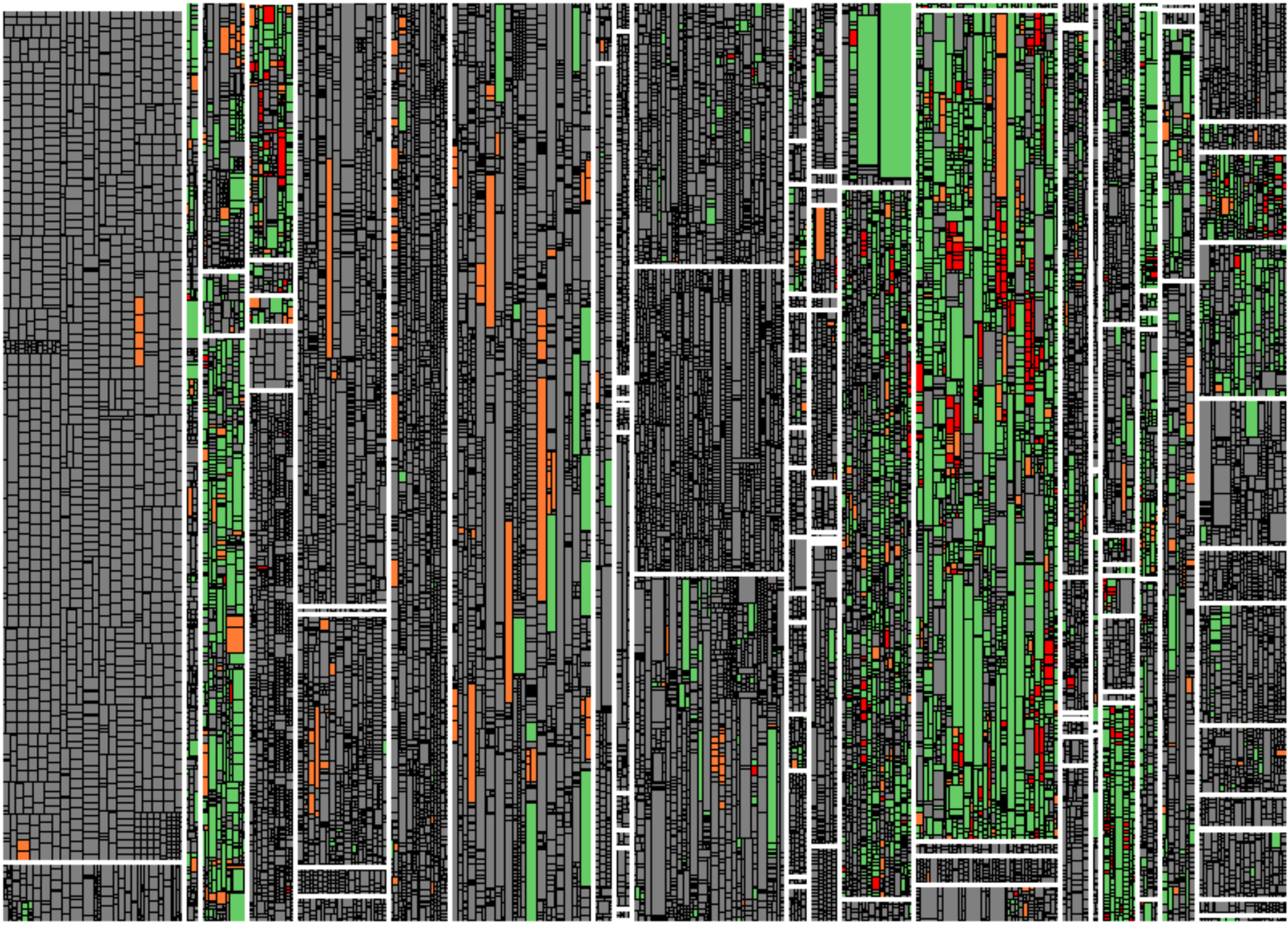


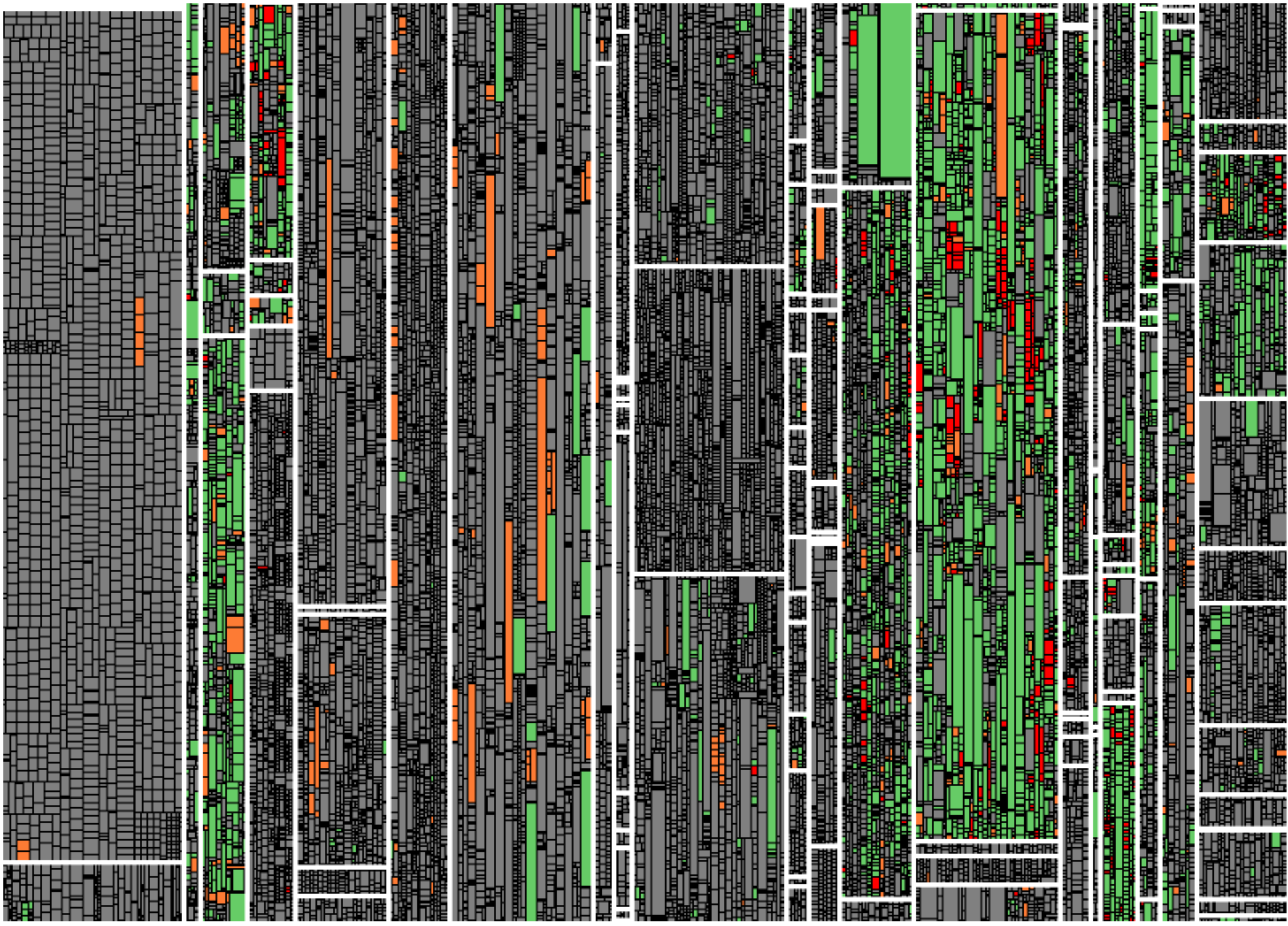


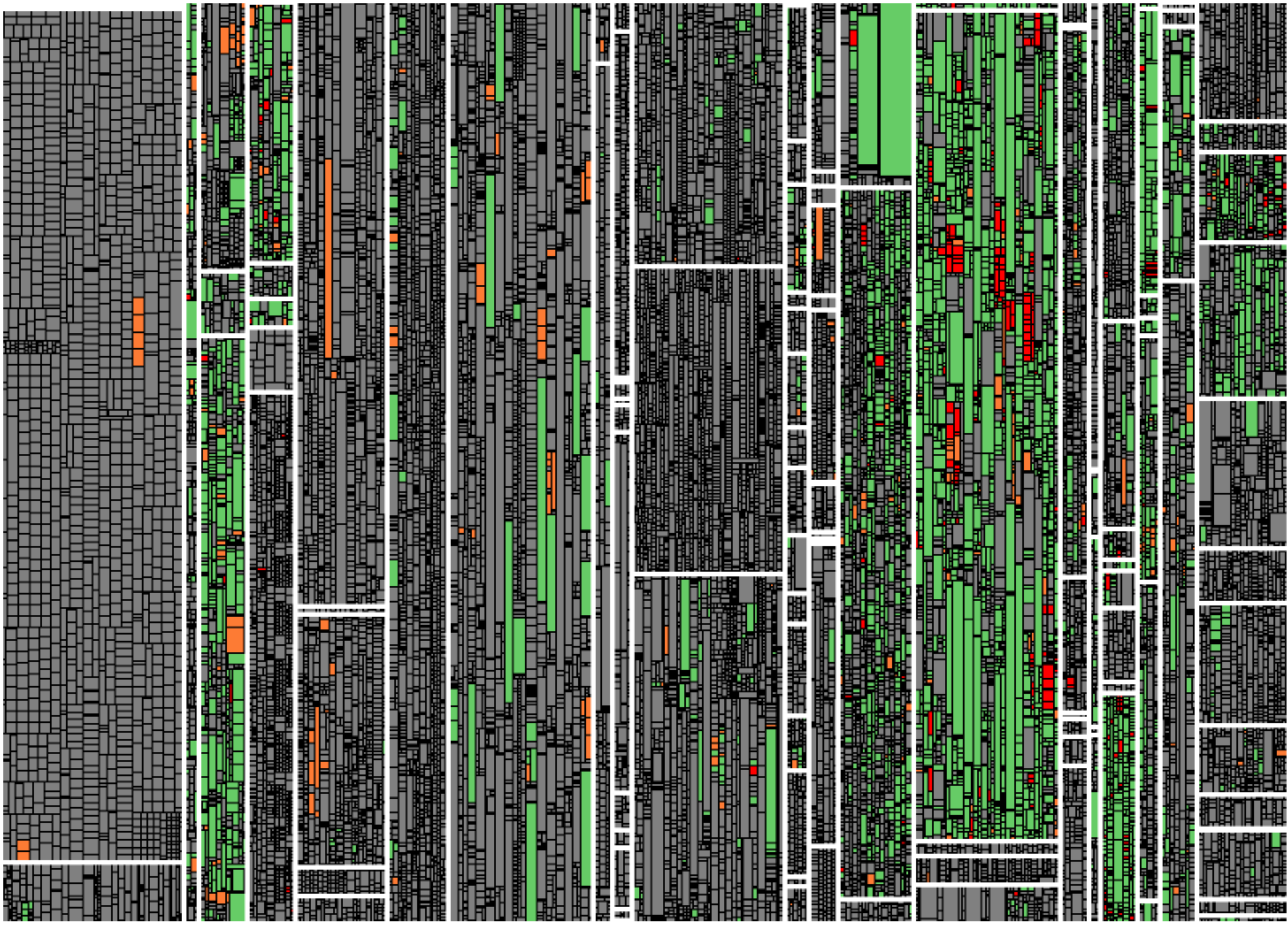


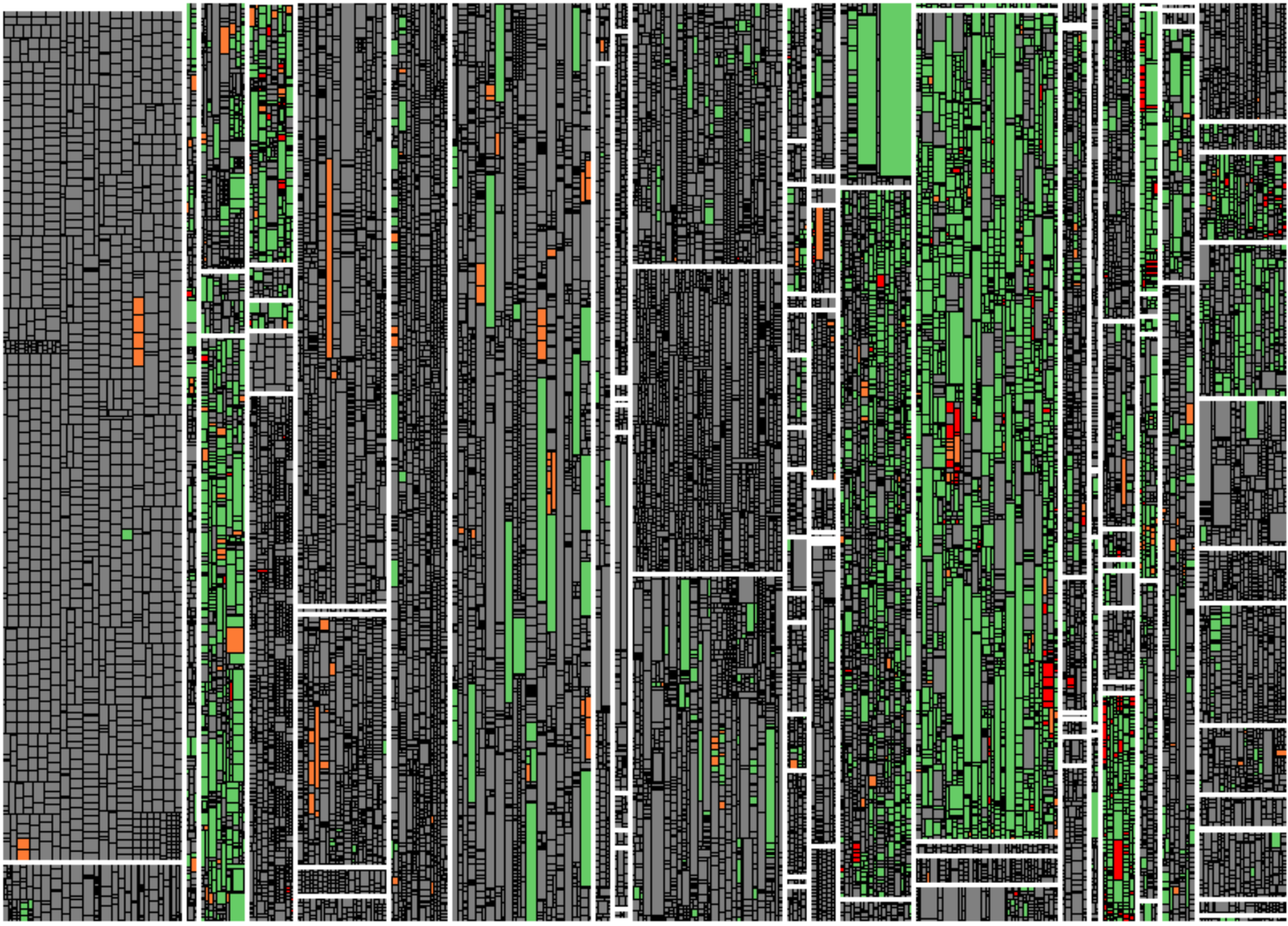










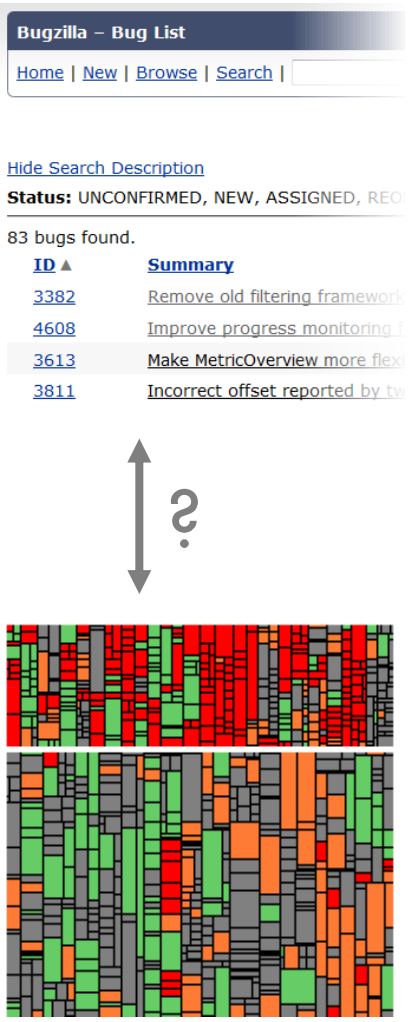


TGA im Iterationstest

- Arbeit in Iterationen in *Work Items* strukturiert. (a.k.a. Change Requests, Issues, Tickets, …).
- Test-Gap-Information liegt aktuell auf Ebene von Code vor, nicht auf Ebene von Work-Items.
- Zusammenhang Test Gaps ↔ Work Items unklar.

- Einige Teams wollen während des Iterationstests nur *einzelne* Work Items testen.
- Unklar, welche Test Gaps das betrifft.

⇒ Roadmap: Filtern von Test Gaps nach Work Items und Iterationen.



Lessons Learned

Für Hotfix-Tests und Release-Tests funktioniert die Betrachtung aller Änderungen seit dem letzten Release.

Für Iterationstests ist es notwendig, auf einzelne Änderungen filtern zu können.

Phasen der Einführung

1. Mit den Projekten beginnen, die viel Interesse haben
2. Show-Cases für andere Projekte ableiten
3. Für alle Projekte einführen
   - Erhebung Messdaten in Testumgebungen integrieren
   - Einrichtung TGA-Dashboards sehr einfach machen

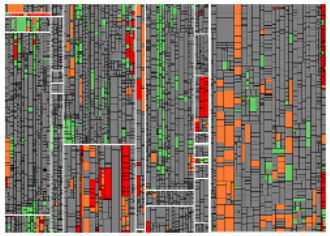
Kontinuierlich Aufwände und Fehlerdaten messen

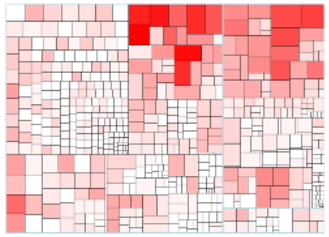
TEST SERVICE PORTAL | MUNICH TEST SERVICE | TEST MANAGEMENT TOOLS | TEST AUTOMATION | TGA / TSA | TEST PRACTICE | TEST COMMUNITY

» Test Service Portal » **Test Gap Analysis & Test Smell Analysis**

Site Content
- Documents
- public
- TGA Backlog
- TGA dashboards
- TSA Backlog
- TSA dashboards
- TGA-TSA-Project

This is the public MOSS for Test Gap Analysis and Test Smell Analysis. It hosts documentation about method and tools and the links to all dashboards.





Test Gap Analysis

Overview Presentation gives a general introduction to the Test Gap method.

Links to all available dashboards are listed on a central page.

The Usage Concept describes how to work with Test Gap Analysis.

For any project / system a ramp-up and installation needs to be done. There are two check lists for necessary information - .NET / SAP.

There is continuous improvement of TGA - see product backlog. If you have ideas for enhancements please propose there.

IT Test Community

INNOVATION BLOG | FORUM | WIKIS | MEMBERS | REPORTING

- TGA / TSA user guide & best practices
  - Test Gap Analysis
  - Test Smell Analysis
    - Core Concepts
    - TSA for Managers
    - TSA for Test Analysts

Core Concepts

This section describes core concepts related to Test Smell analysis. In general, Test Smell analysis aims at helping projects to write better test cases by applying several heuristics. Some of these use natural language processing. By raising awareness about possible problems in test cases, teams are enabled to take conscious decisions about where improvements make sense.

What is Measured?

The analysis tries to find possible problems in test case descriptions using smell detection, i.e. searching for certain patterns that might be problematic during the execution or maintenance of test cases. The categories we currently look at are:

- *Vague sentences* that leave room for interpretation and may therefore not lead to repeatable test execution.
- *Conditional sentences* that introduce complexity into a single test step, where an additional test case may be more appropriate.
- *Long steps* that may be split up into separate steps, so that additional check points are available in-between.
- *Long sentences* that may be hard to understand by the tester.
- *Clones* that introduce additional maintenance effort should the cloned part need to be changed in the future.

The parameters for these analyses are shown in every dashboard. See the following tables for the current defaults.

Configurable Parameters

Parameter Name	Description	Value

Lesson Learned

Damit Test-Gap-Analyse langfristig eingesetzt wird, muss ***Change Management*** betrieben werden.

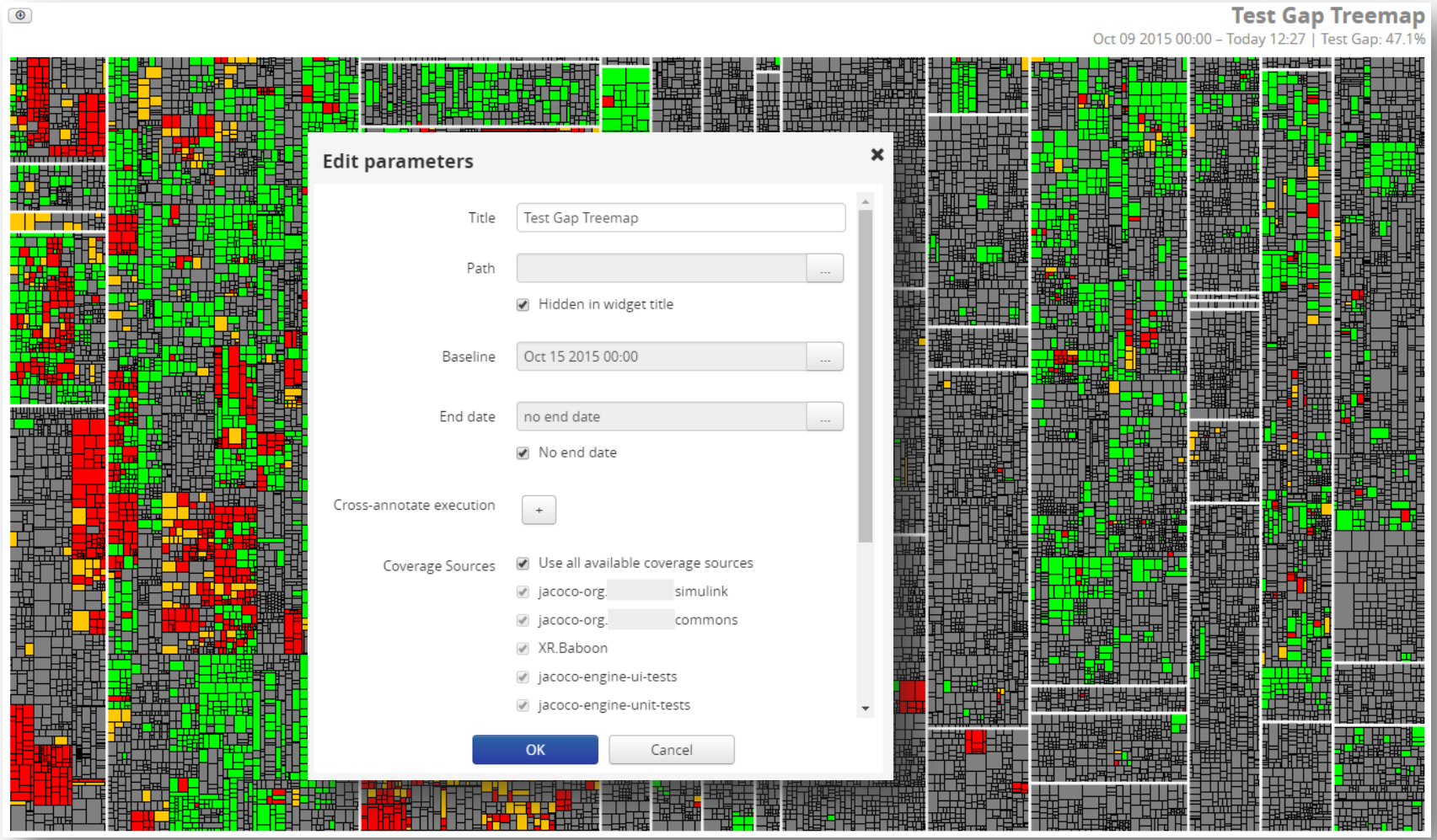
Fazit

Test-Gap-Analyse schafft ***Transparenz***. Sie wird von Teams genutzt, um ungewollt ungetestete Änderungen zu vermeiden.

Jede Testart hat eigene Anforderungen. Test-Gap-Analyse ist am einfachsten bei ***Hotfix- & Release-Tests*** einsetzbar.

Damit Test-Gap-Analyse langfristig eingesetzt wird, muss ***Change Management*** betrieben werden.

Weiterentwicklung: Test-Gap-Analyse interaktiv



Software Quality Blog

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
em. Therefore it makes sense to use
fy those changed but untested areas.

Several ti
In the ma
from Pyth
containin
may prov
Analysis i

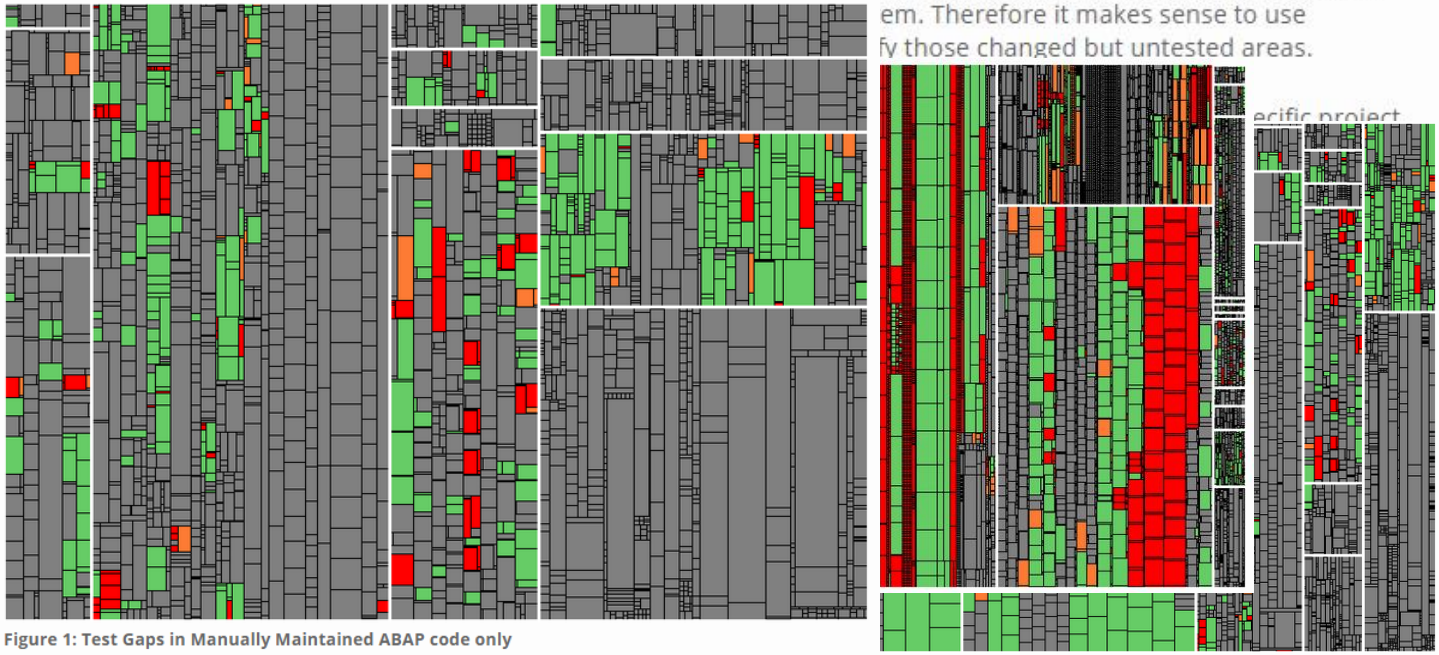


Figure 1: Test Gaps in Manually Maintained ABAP code only

https://www.cqse.eu/en/blog/testing-changes-in-sap-bw/

Kontakt

Ich freue mich auf Diskussionen, auch im Anschluss!

Dr. Andreas Göb · goeb@cqse.eu · +49 176 101 552 25

@a_goeb
www.cqse.eu/en/blog

CQSE GmbH
Lichtenbergstraße 8
85748 Garching bei München

CQSE
Continuous Quality in Software Engineering