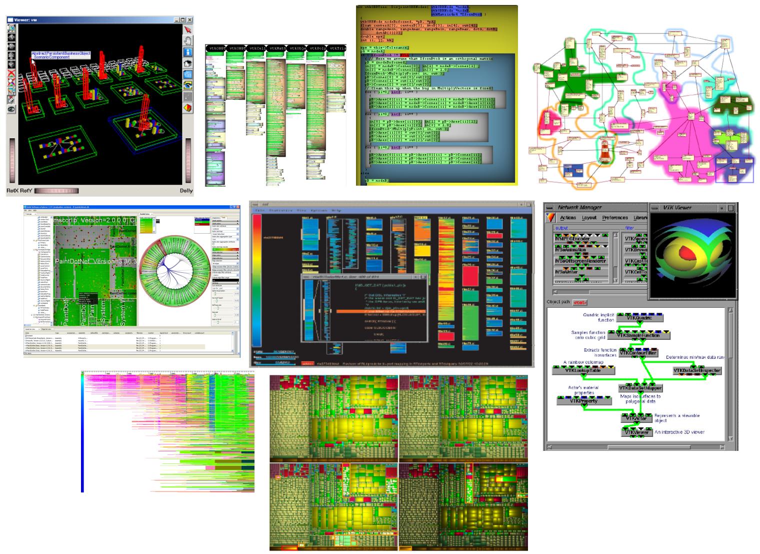
Software Visual Analytics for Testing

Opportunities and Challenges



prof. dr. Alexandru (Alex) Telea

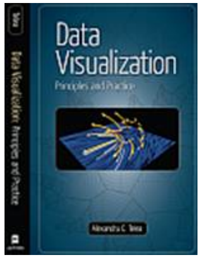
Department of Mathematics and Computer Science
University of Groningen, the Netherlands

Introduction

www.cs.rug.nl/~alext

SolidSource[IT]

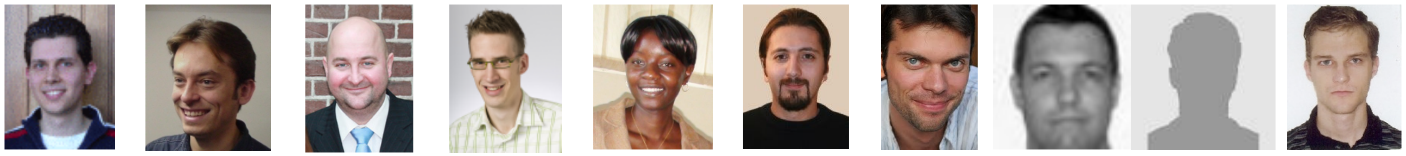
www.solidsourceit.com



Data Visualization: Principles and Practice A. K. Peters, 2008

Professor of Computer Science (Multiscale Visual Analytics),
University of Groningen, the Netherlands

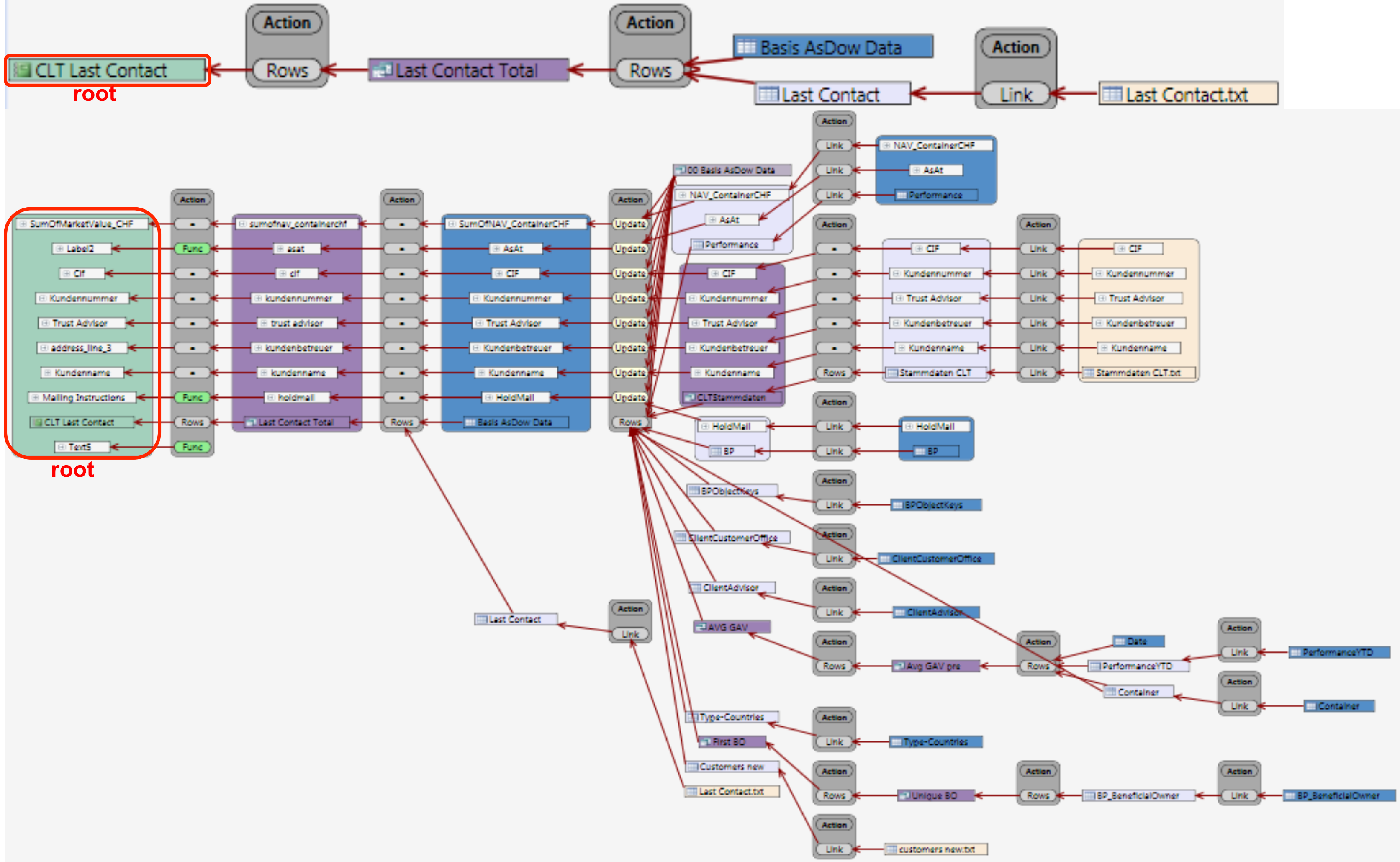
My PhD students…



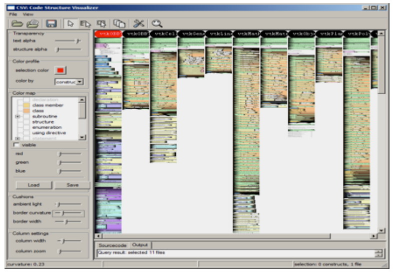
My MSc students…

Avdo Hanjalic, Tijmen Klein, Johan v/d Geest, Mark Ettema, Daniel Kok, Karsten Westra, Yuri Meiburg, Hessel Hoogendorp, Liewe Kwakman, Madalina Florean, Bertjan Broeksema, Mark Stoetzer, Sergio Moreta, Kees van Koten, Frans Boerboom, Arjan Janssen, Freek Nossin, Matthijs van Eede, Martijn van Dortmont, Maurice Termeer, Iwan Vosloo, Gerard Lommerse, Dennie Reniers, Milan Pastrnak, …

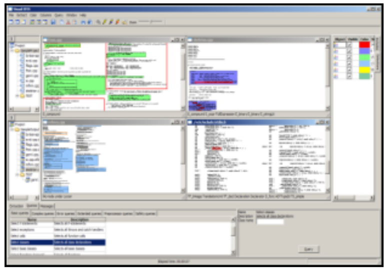
Software Visualization?



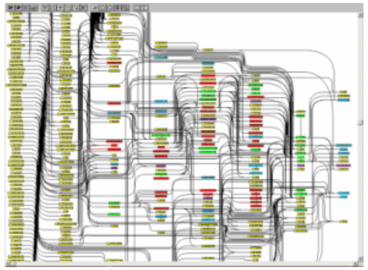
Software Visualization!



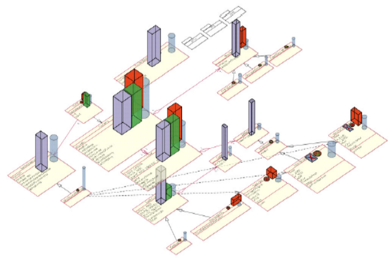
source code



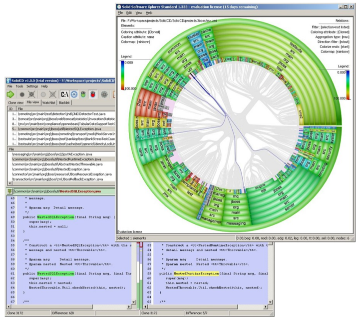
code quality



code dependencies



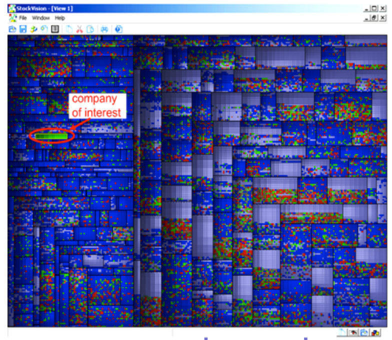
design and metrics



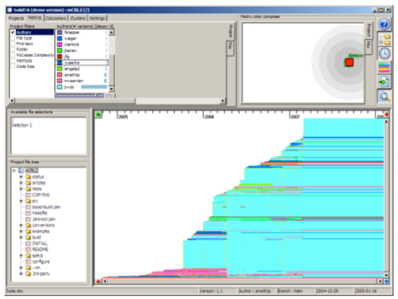
text duplication

How should we deal with **scale**?

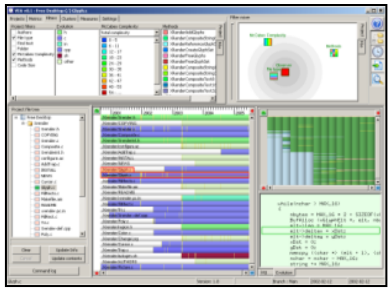
- simplified visualizations?
- continuous simplification?
- what to simplify exactly?
- reinvent wheel for each app?



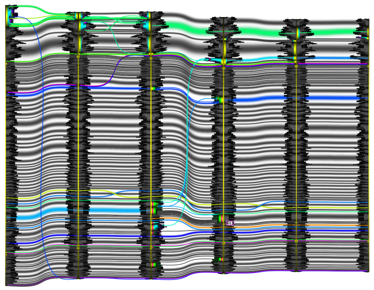
program dynamics



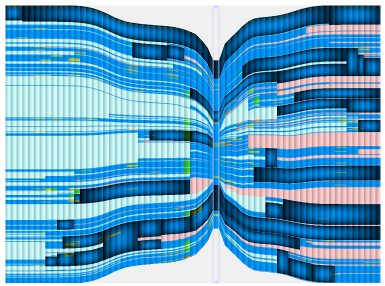
code repositories



evolution metrics



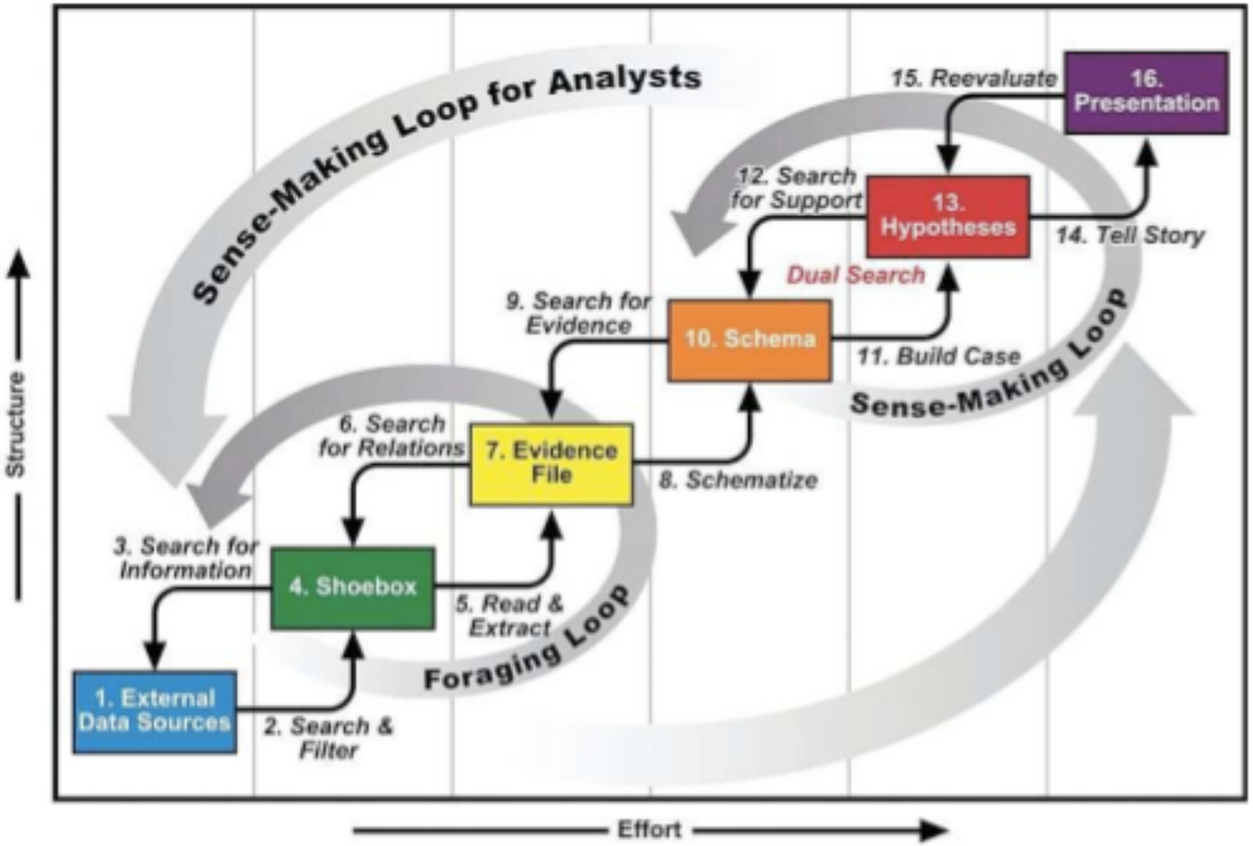
structure evolution



team analysis

Software Visual Analytics – Process View

"The science of analytical reasoning facilitated by interactive visual interfaces"

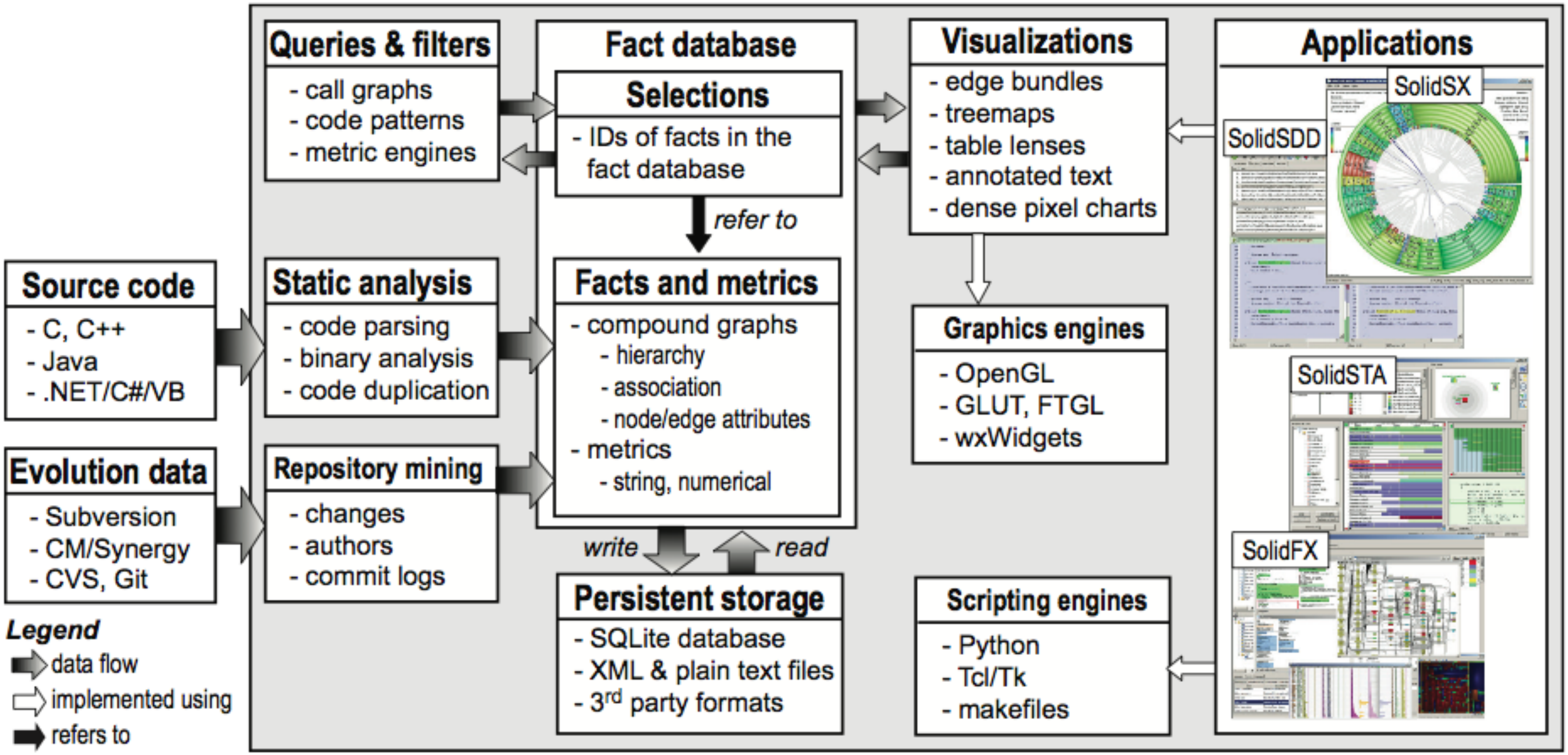


The Sensemaking Loop

- going from r**aw data** to **meaning** (semantics) to **insight** to **decisions**
- data → hypothesis → (in)validation → conclusions → presentation
- put simply: **combine analysis and visualization**

P. Wong, J. Thomas, Visual analytics, IEEE Comp. Graphics & Applications, 24(5), 2004
J. Thomas, K. Cook*, Illuminating the Path: The R&D Agenda for Visual Analytics*, NVAC, 2005

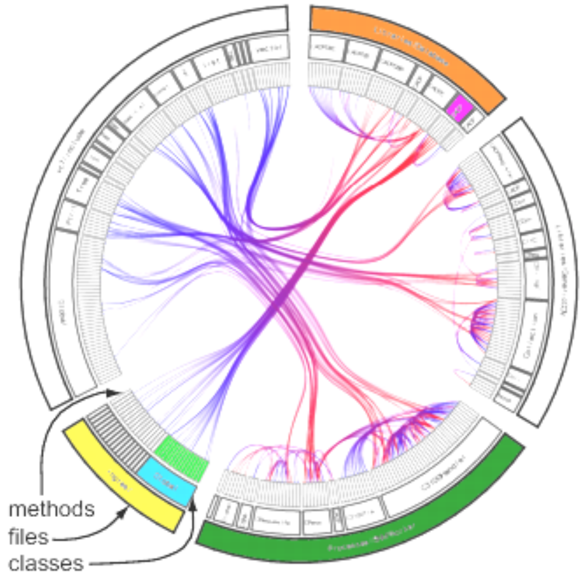
Software Visual Analytics – Technical View



Many types of **data** and **questions** → many types of **visualizations**

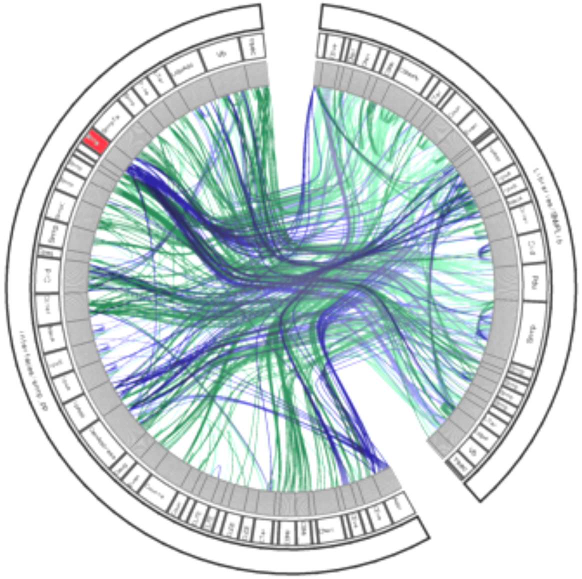
1. Assessing system modularity

Modular system



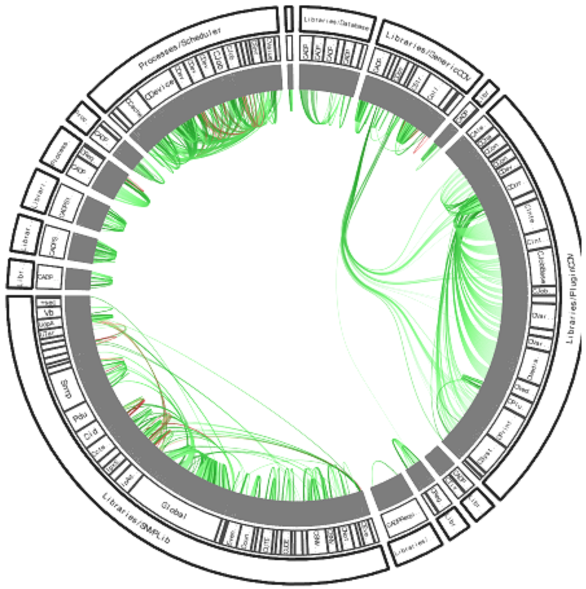
- blue = caller, red = called
- all functions in the yellow file call the purple class
- green file has many self-calls

Monolithic system



- blue = virtual, green = static functions
- red class has many virtual calls (possible interface class)

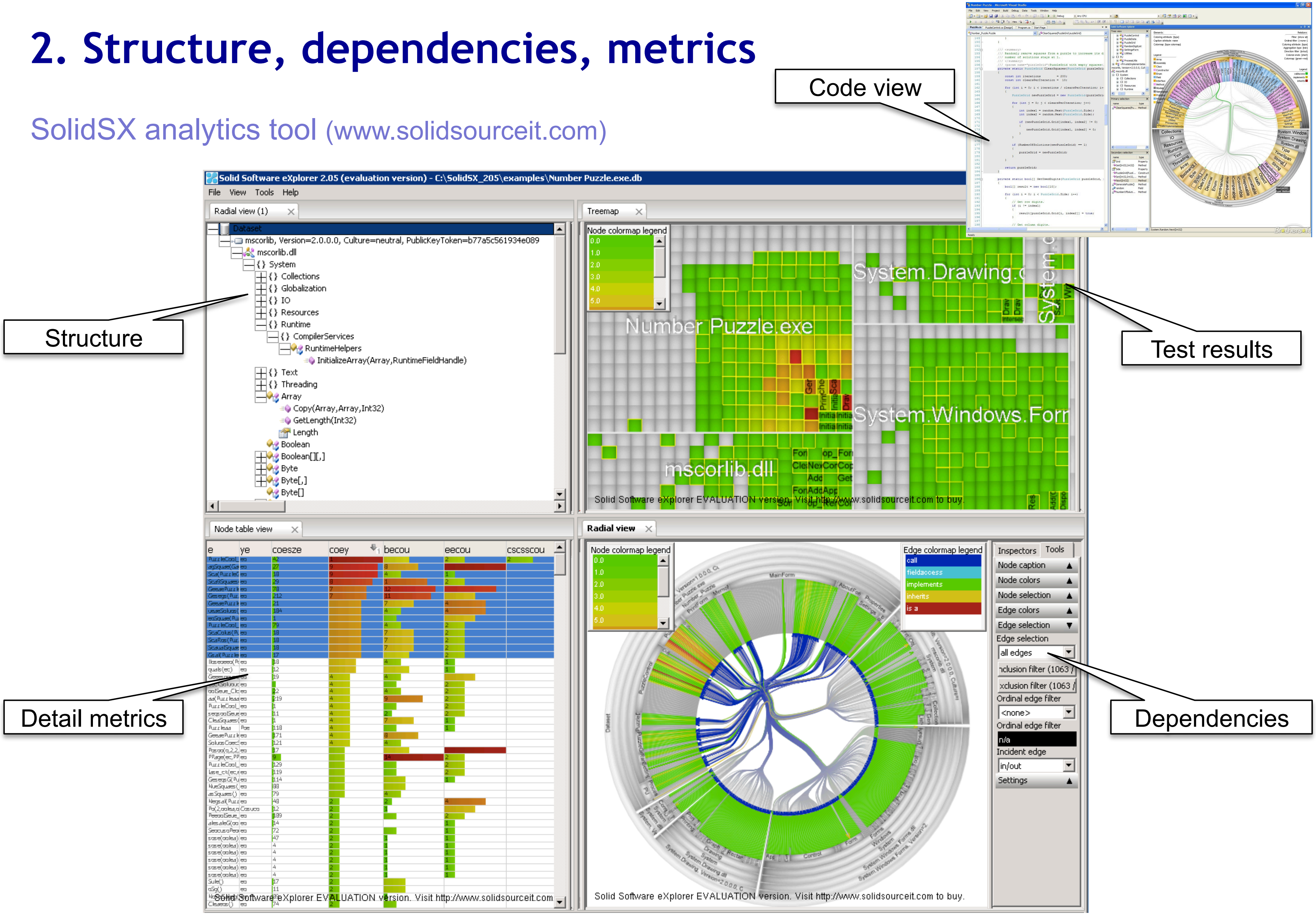
Decoupled system



- many intra-module calls
- few inter-module calls
- typical for library software

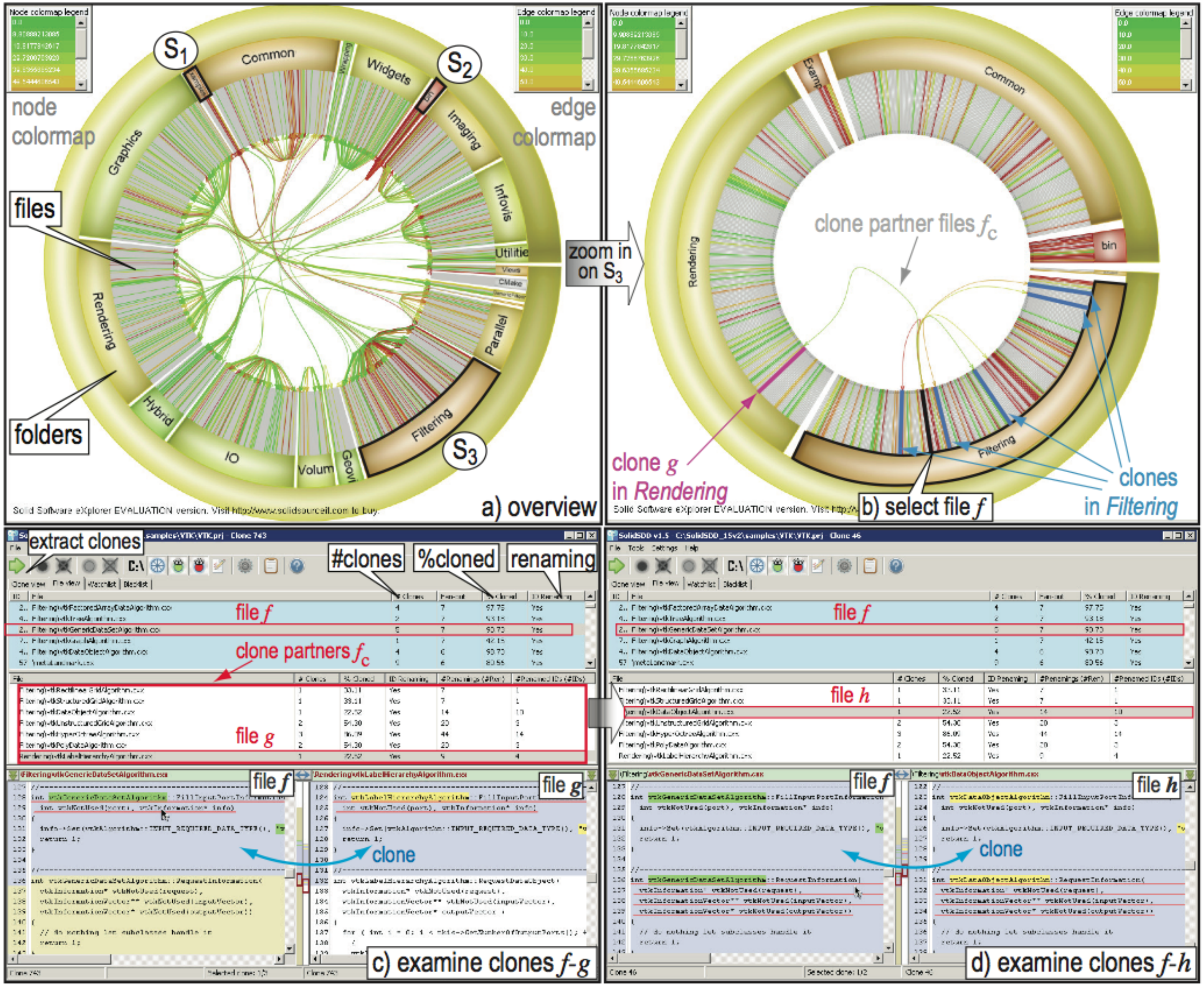
2. Structure, dependencies, metrics

SolidSX analytics tool (www.solidsourceit.com)



3. Code duplication

SolidSDD tool (www.solidsourceit.com)

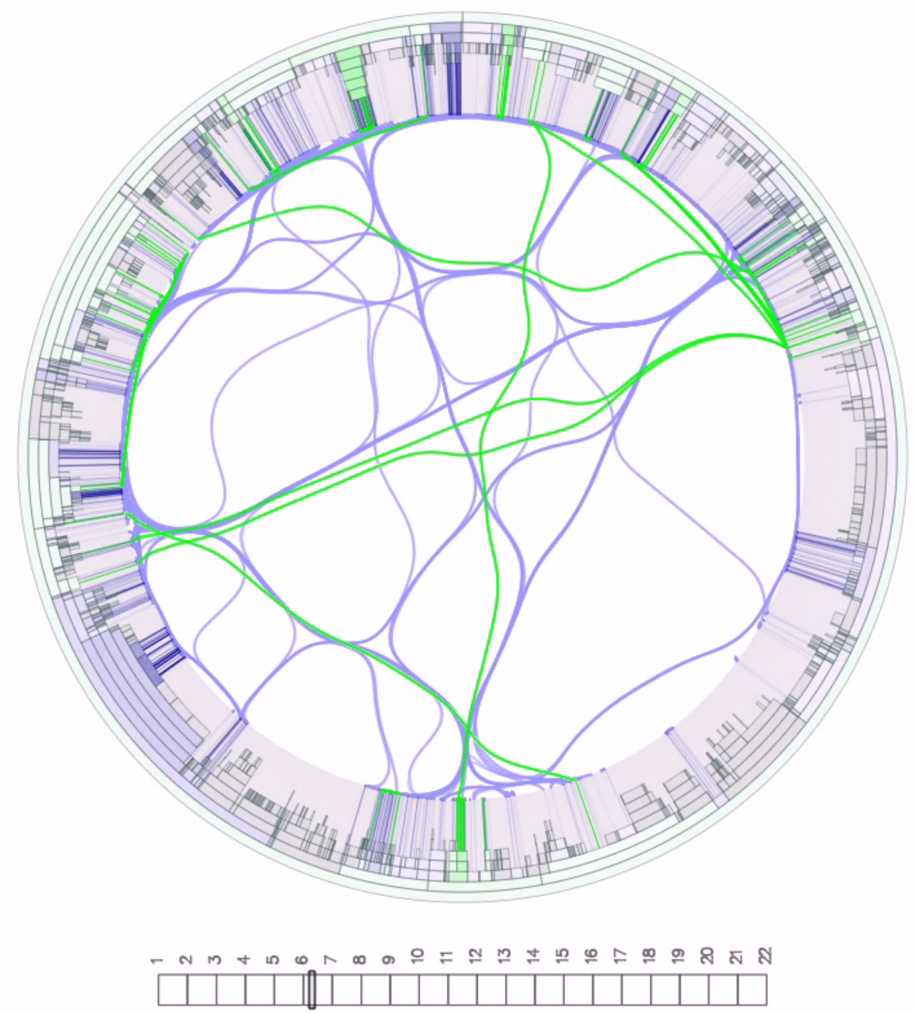


Color legend: non-cloned code; clone partner not shown; clone shown in both windows; identifier renamed (shown as in right window)

4. Clone evolution

Questions

- how does code duplication change in time?
- which clones are added, removed, merged, or split? And why?

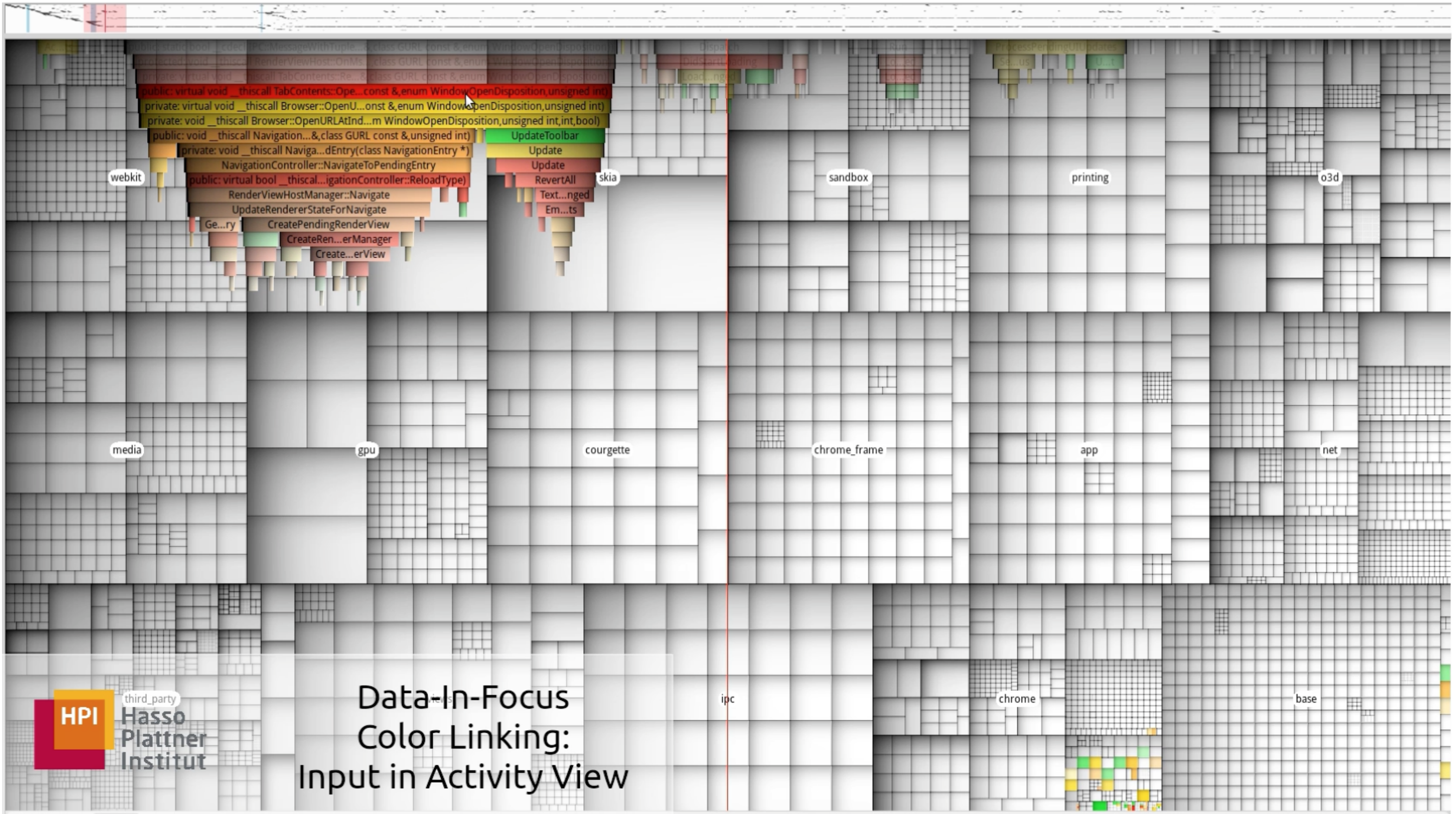


Evolution of clones in Mozilla Firefox (~55K clone relations)

5. Program trace and structure

Questions

- where (in the program structure) are the calls executed now?
- when (during execution) are calls to this subsystem done?

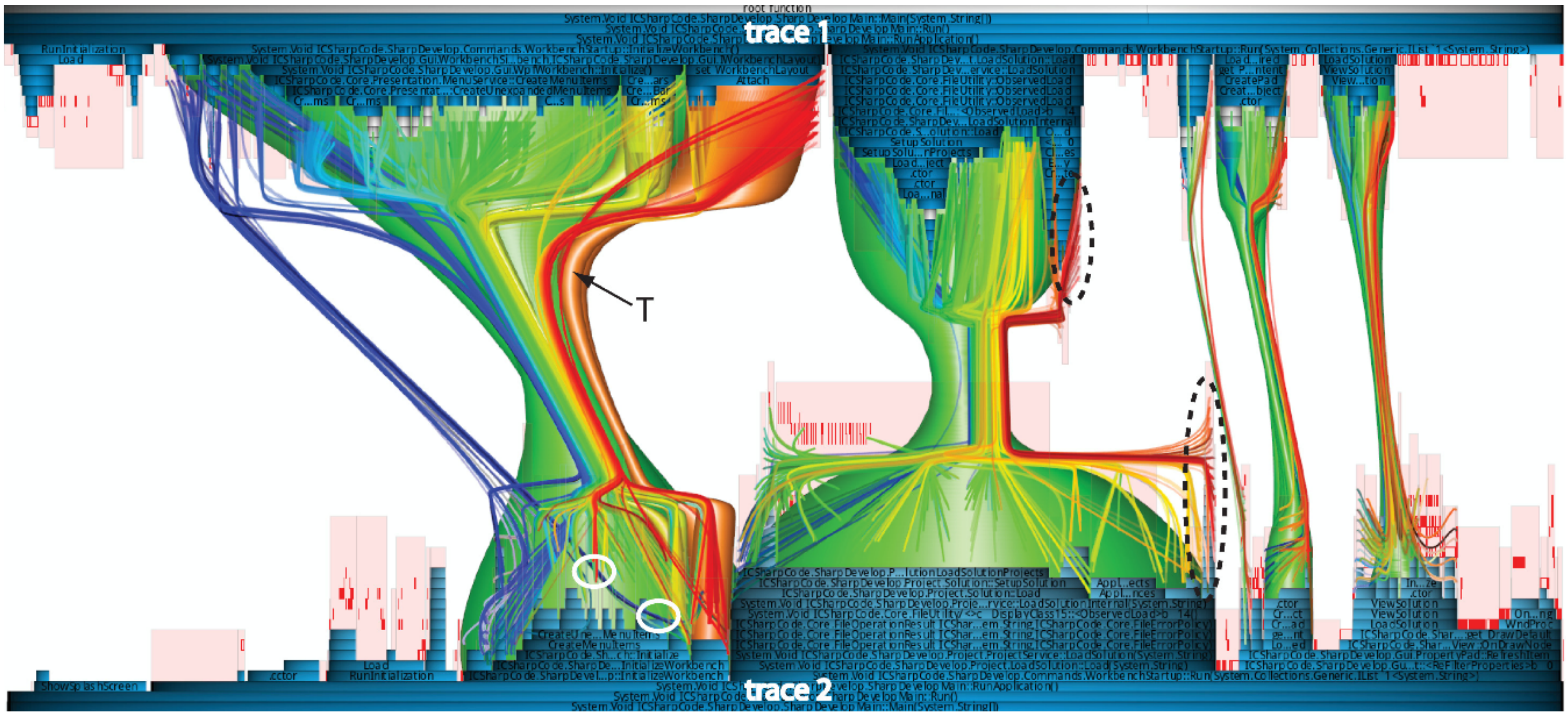


Code: Chrome browser (2.7 MLOC C/C++, 8900 files+folders)
Trace: 9000 calls to 914 functions

6. Comparing program traces

Questions

- given 2 traces, where are similar and where are different call-blocks?
- how to spot differences in call moment, duration, and called functions?

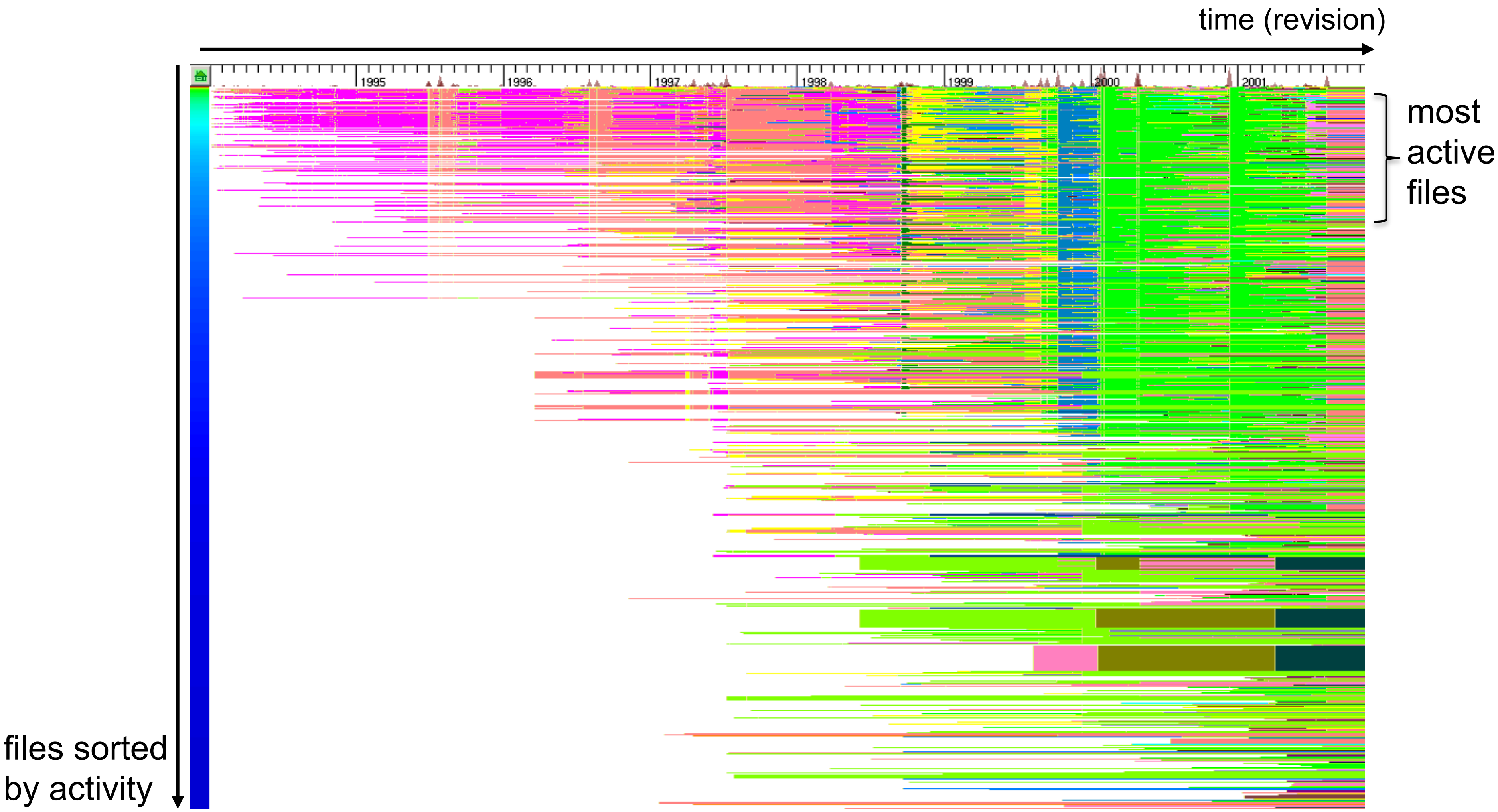


Code: 1MLOC C#, 45 developers, 8 years
Traces: 2x150K calls to 1500 functions

7. Software Evolution

Questions

- how to correlate **metrics** over large software repositories (>10K files, >100K commits?)
- how to detect **trends** to predict the future (cost, effort, risk)?

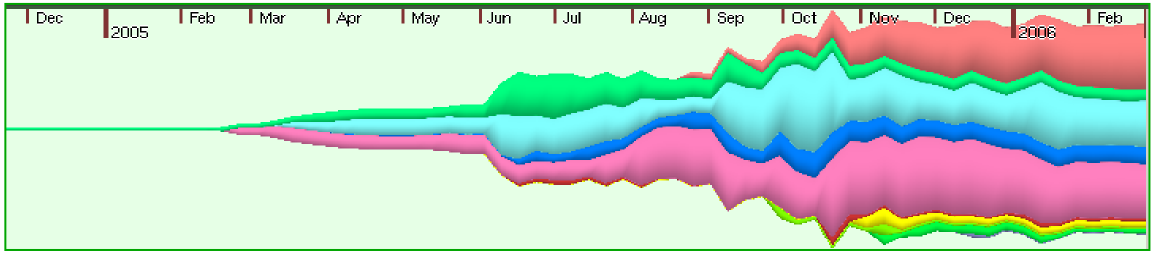


Analyzing developer effort

Show aggregated **developer impact** (#files modified by each developer) over time

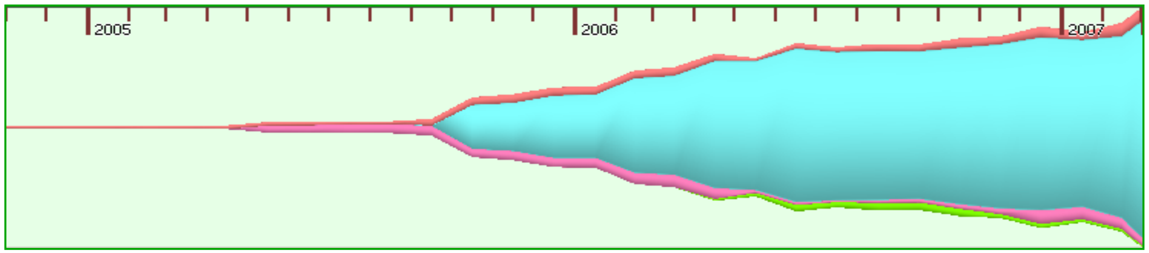
Project A (open-source)

- software grows in time
- impact: balanced over most developers

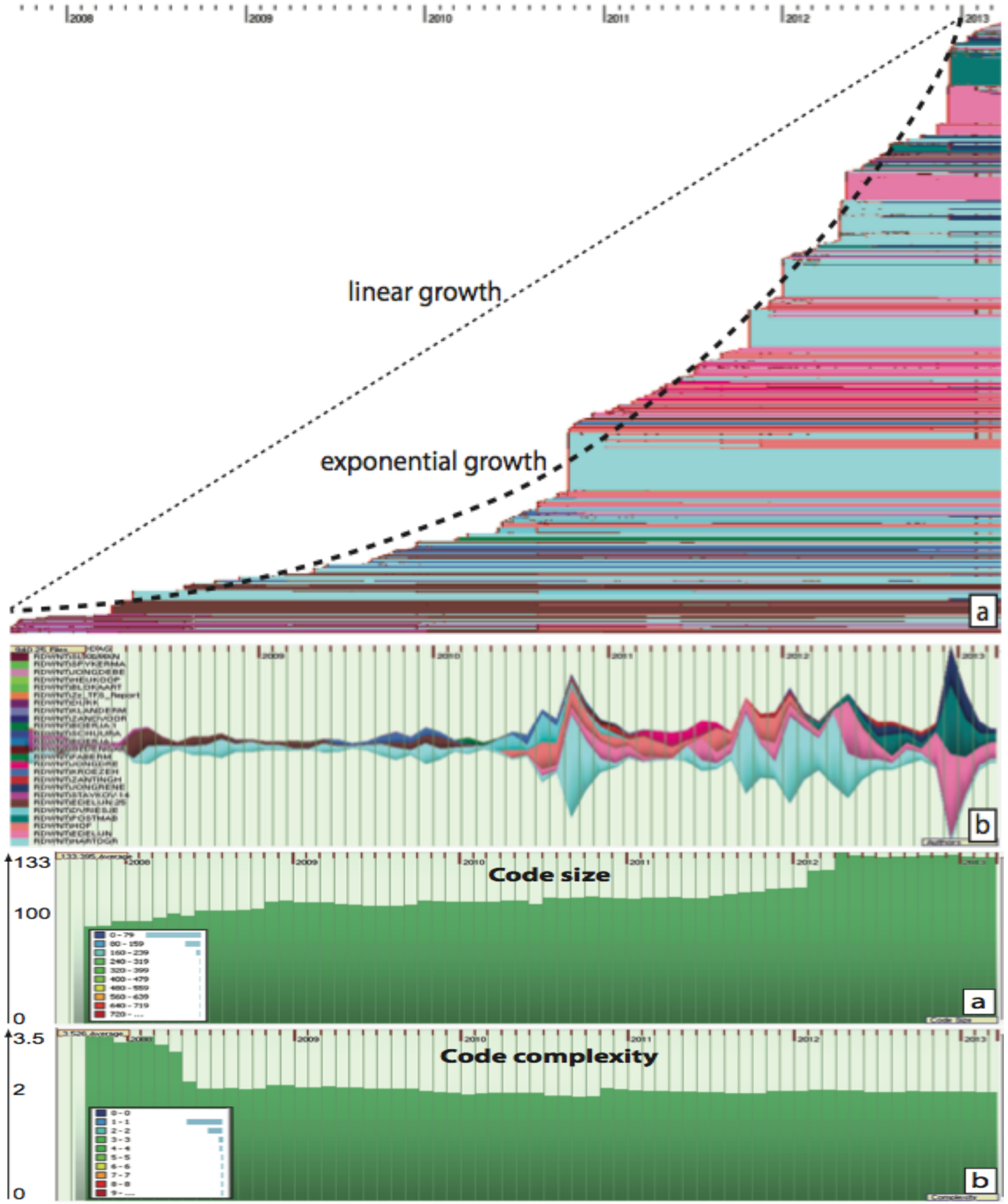


Project B (commercial)

- software grows in time at about the same rate
- but one developer owns most of the code
- what if this person leaves the team?!



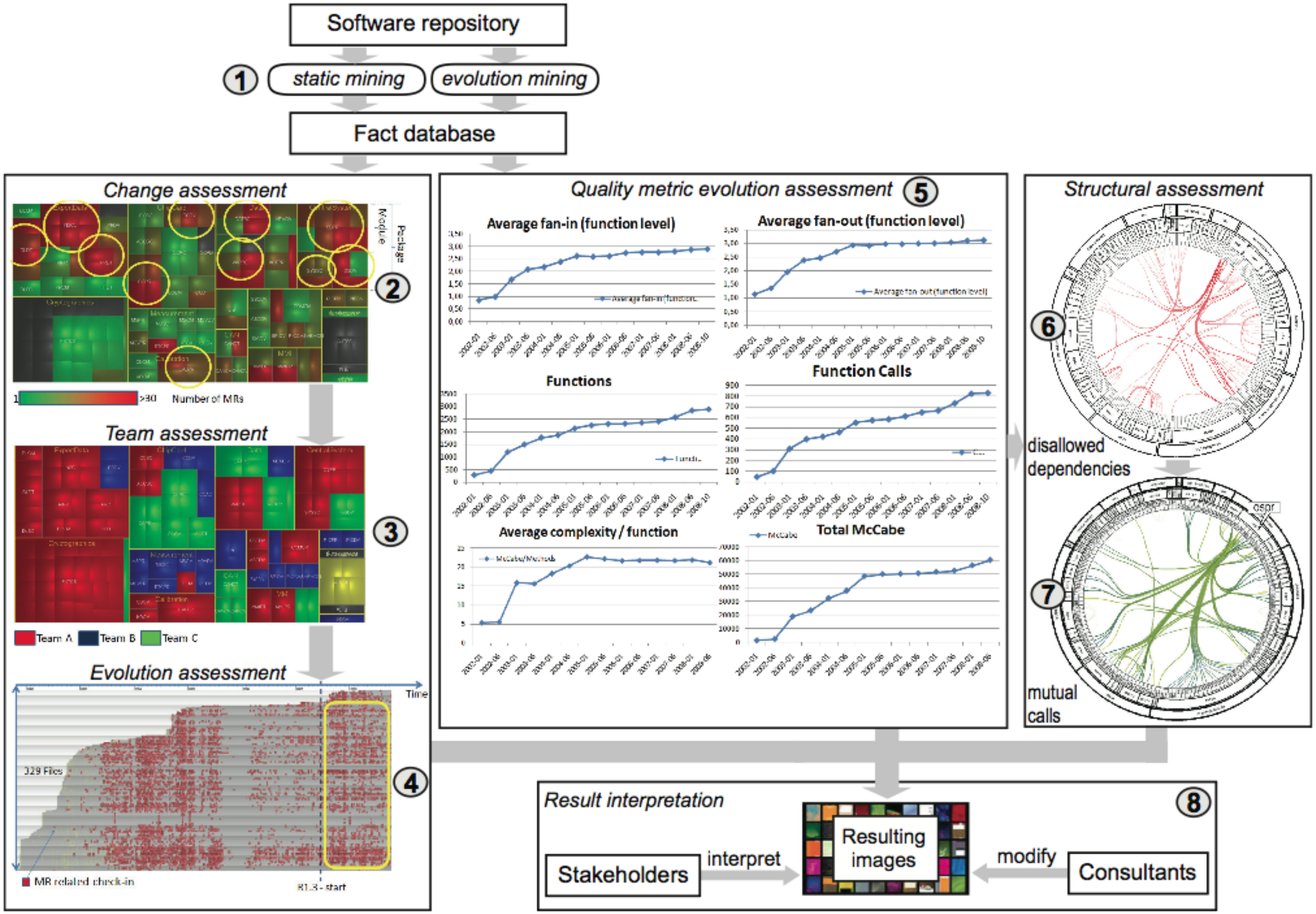
Correlating quality metrics



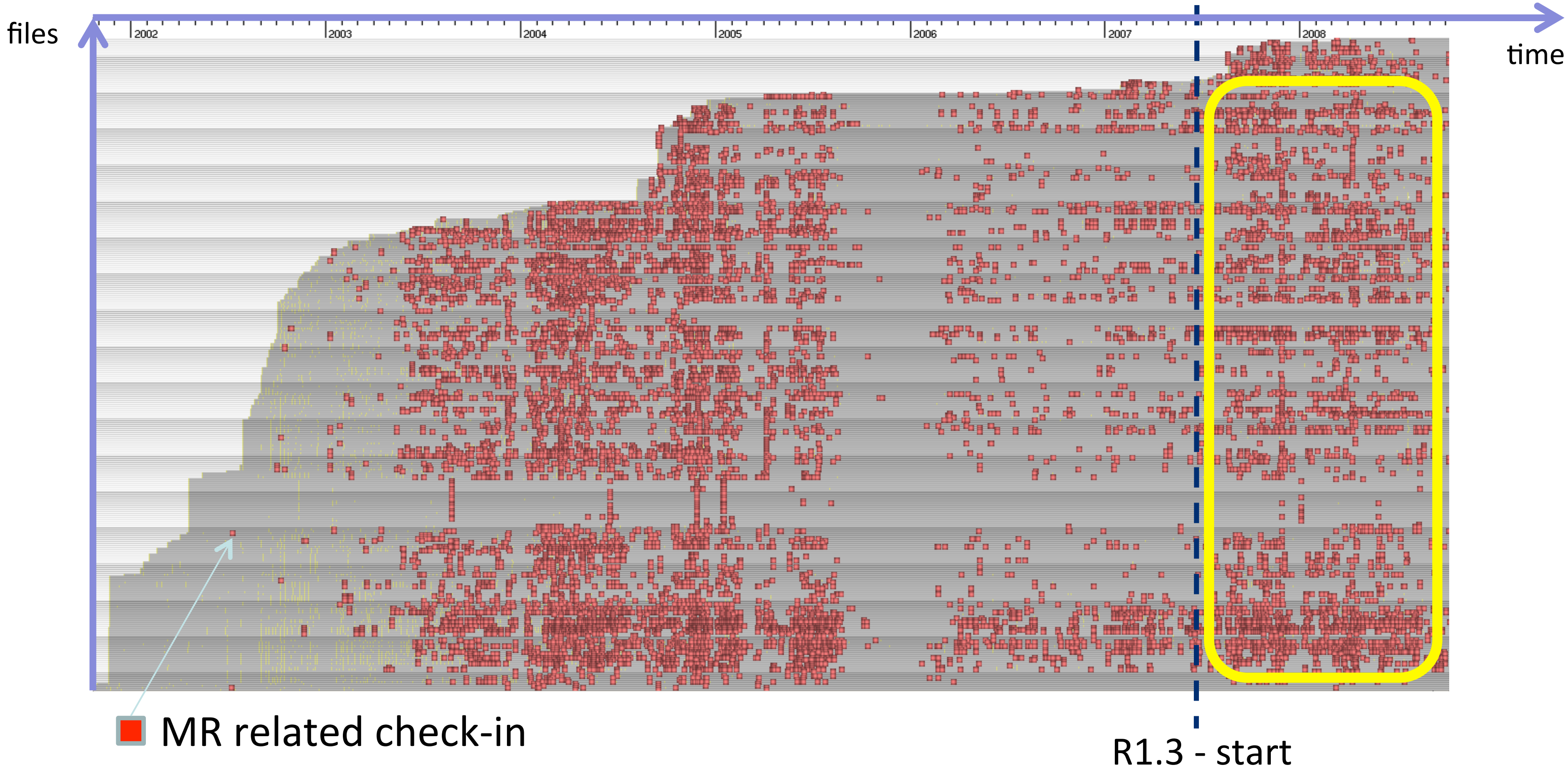
8. Application: Post-Mortem Assessment

Questions

- automotive project: 8 years, 3.5 MLOC embedded C, 15 releases, 60 developers
- project failed to deliver. Why?

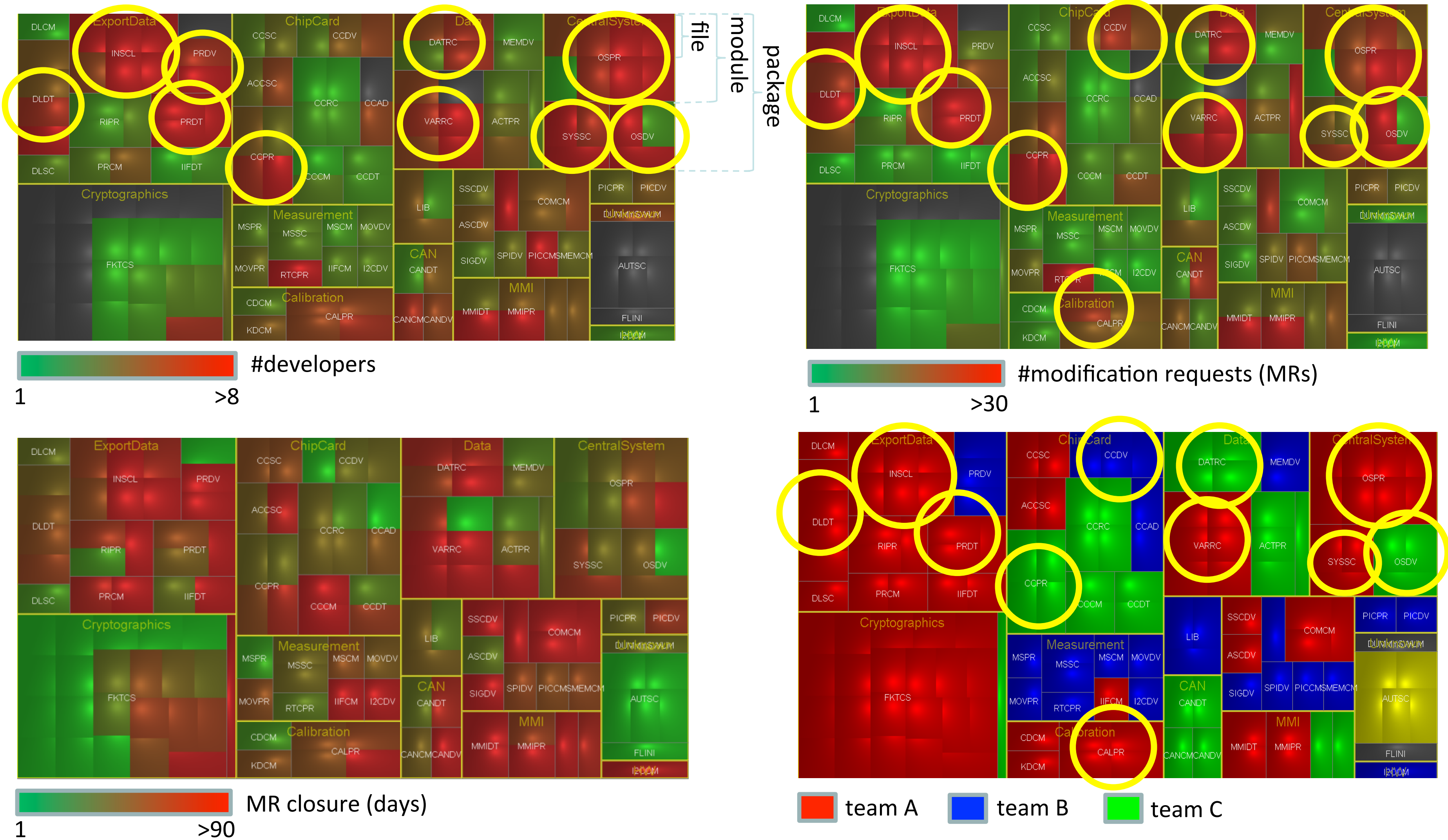


Analysis 1: Modification Request (MR) Lifetime



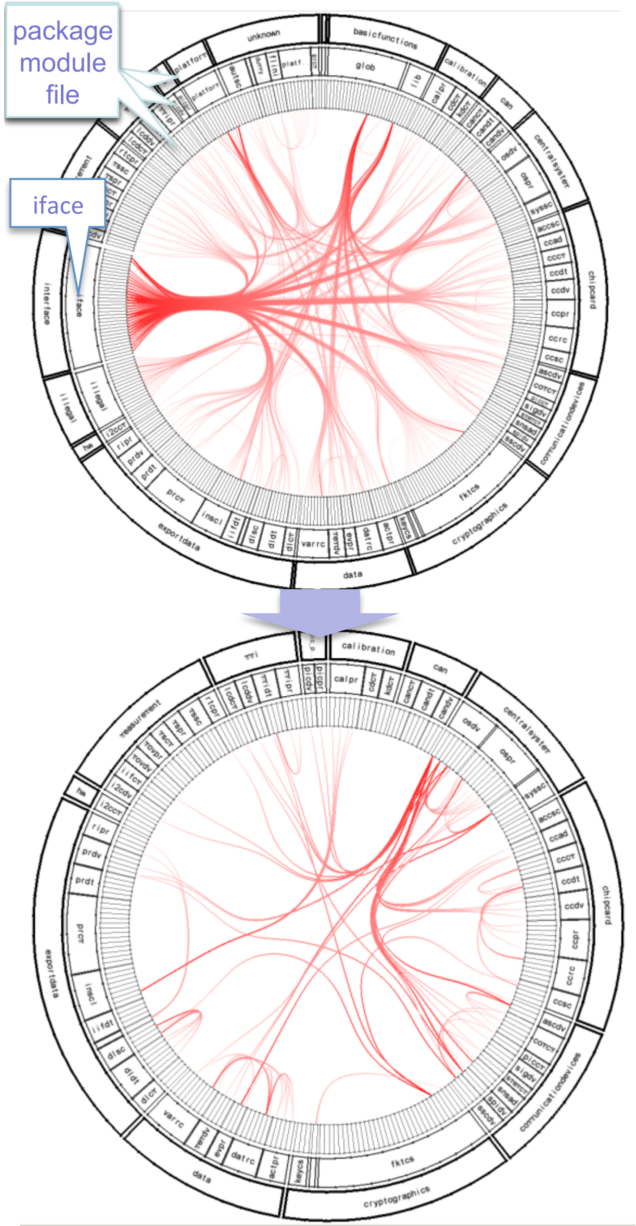
Little increase in the file curve – most activity in *old* files suggests too long maintenance & closure of requirements

Analysis 2: Team Code Ownership



Large part of software affected by long open-standing MRs
Most of these are assigned to team A (largest team)…
…and this team was reported to have communication problems!

Analysis 3: Code Dependencies



uses = call, type, variable, macro, …

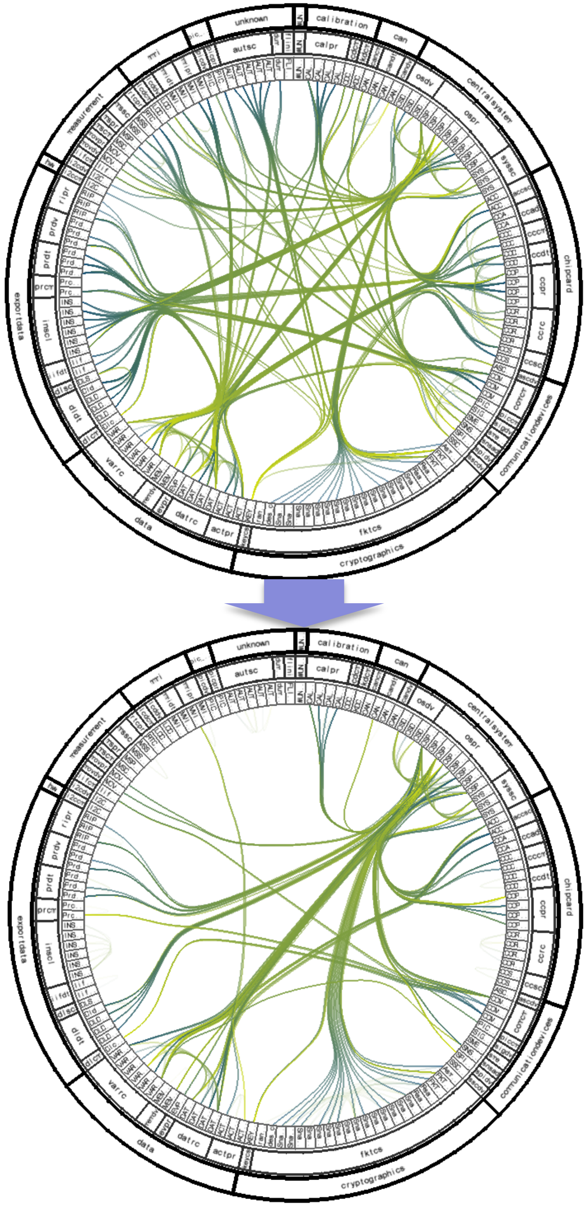
is used

Most dependencies occur via the <u>iface</u>, <u>basicfunctions</u> and <u>platform</u> packages

Filter out these allowed dependencies…
…to discover *unwanted* dependencies

These are accesses that bypass established interfaces
There are several such accesses (bad)

Analysis 4: Code Call graph

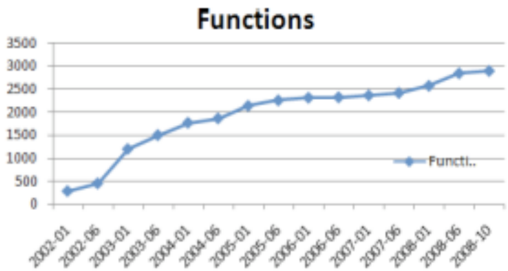


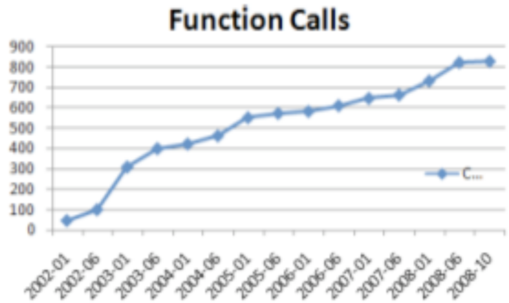
High coupling at package level
This image does not tell us very much

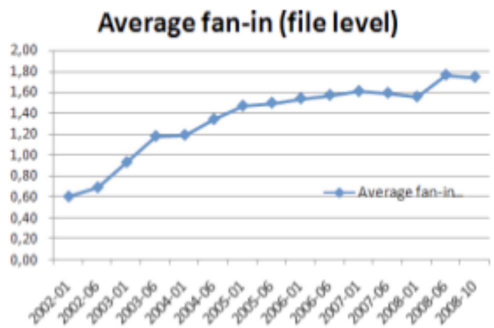
Select only modules which are *mutually call dependent*…
…to discover *layering violations*

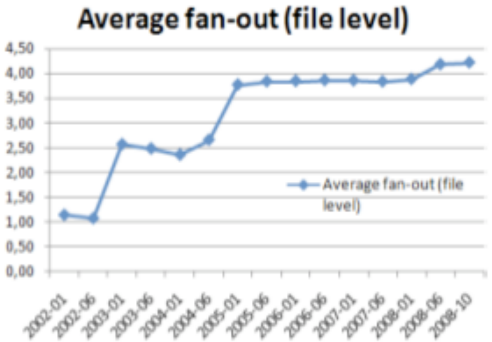
Not a strict layering in the system (as it should be)
Thus, the architecture is violated.

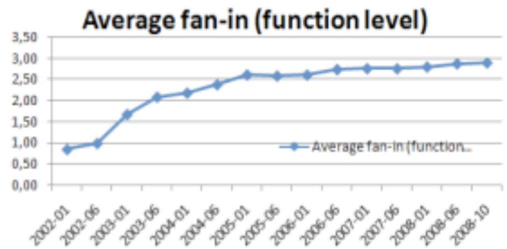
Analysis 5: Code Quality Metrics

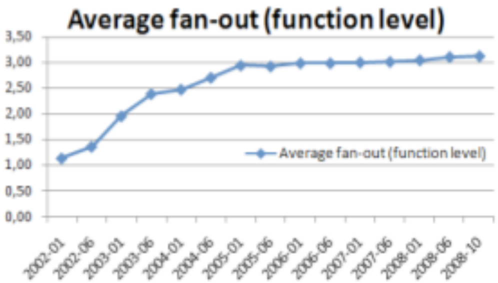


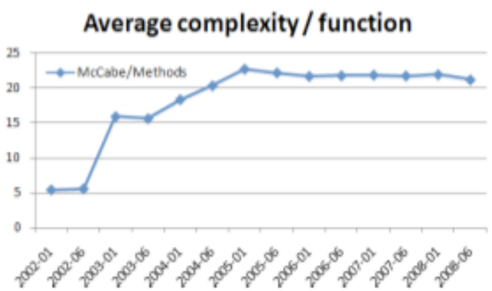


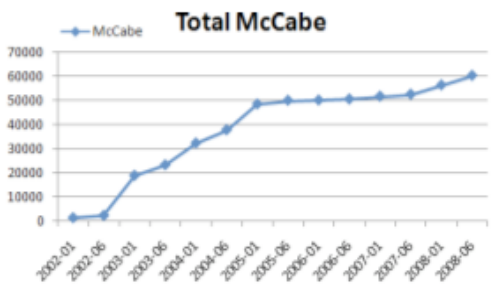












Moderate code + dependency growth

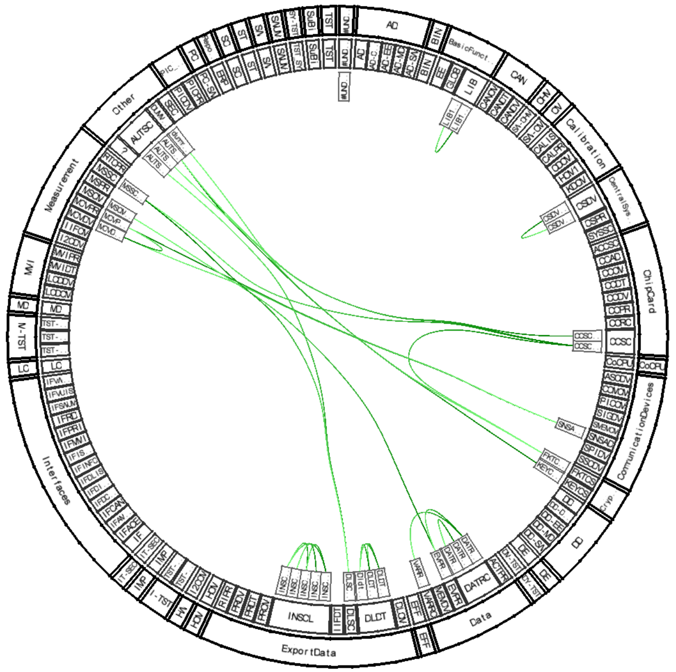
- does not explain products problems

Average complexity/function > 20

Total complexity: up 20% in R1.3

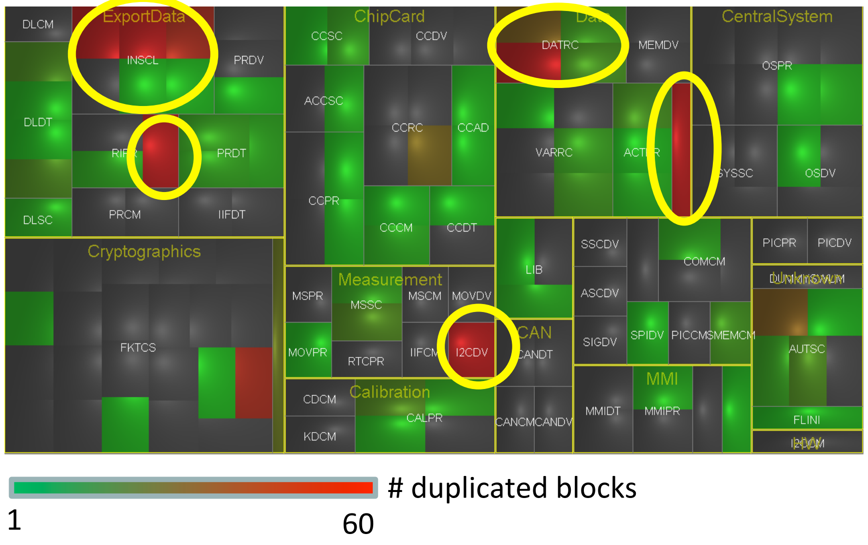
- testing can be hard!
- **possible cause** of product's problems

Analysis 6: Code Duplication



External duplication

- show modules having similar code blocks of >25 LOC

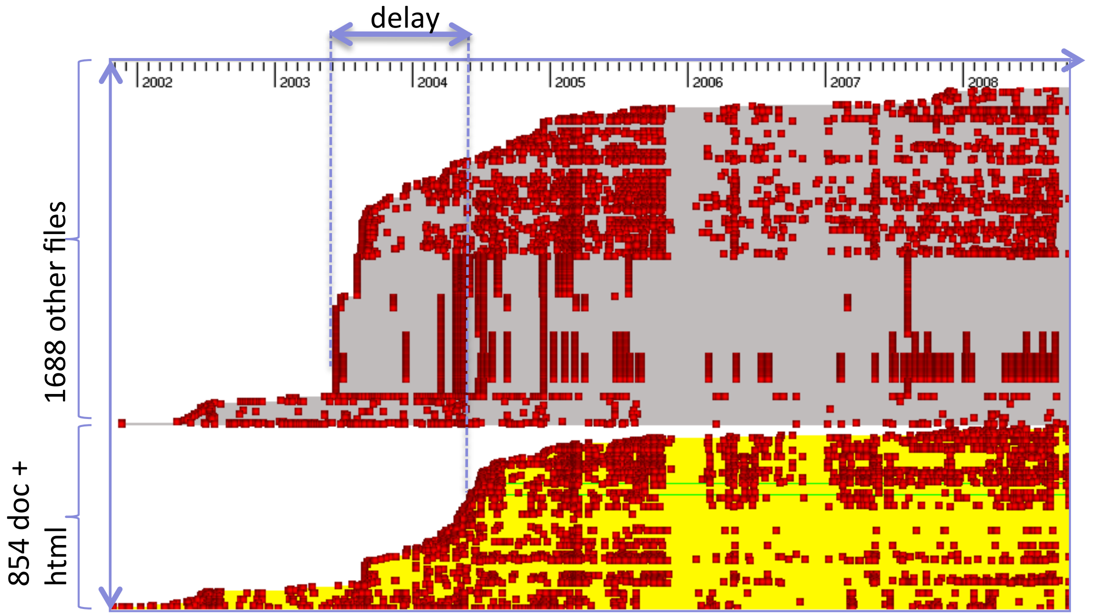


Internal duplication

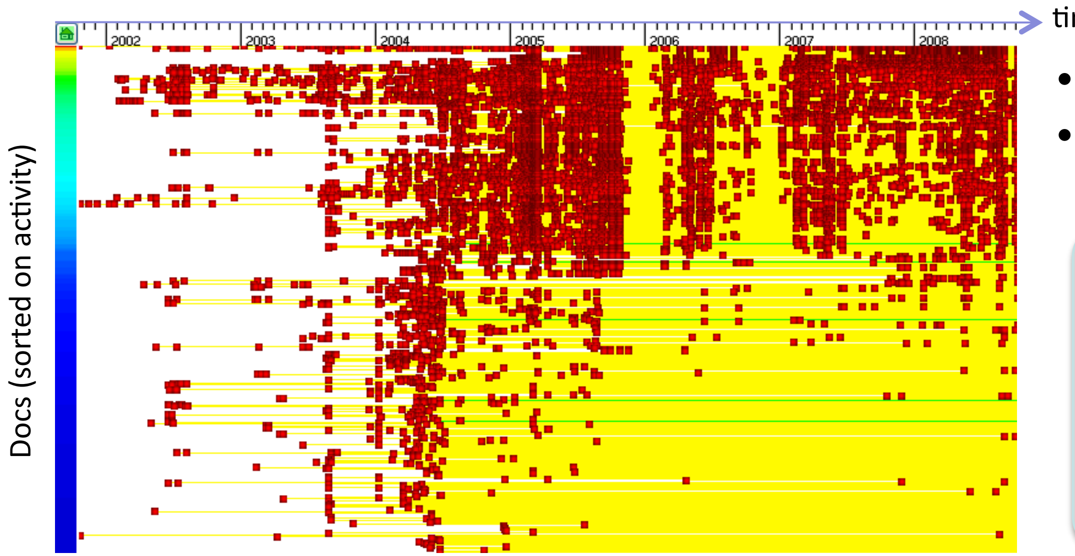
- color: #duplicated blocks within a file

Little external/internal duplication
Arguably **not a problem** for testing

Analysis 7: Documentation



- **30% of files** are documentation
- updated **regularly**
- grow **in sync** with rest of code base

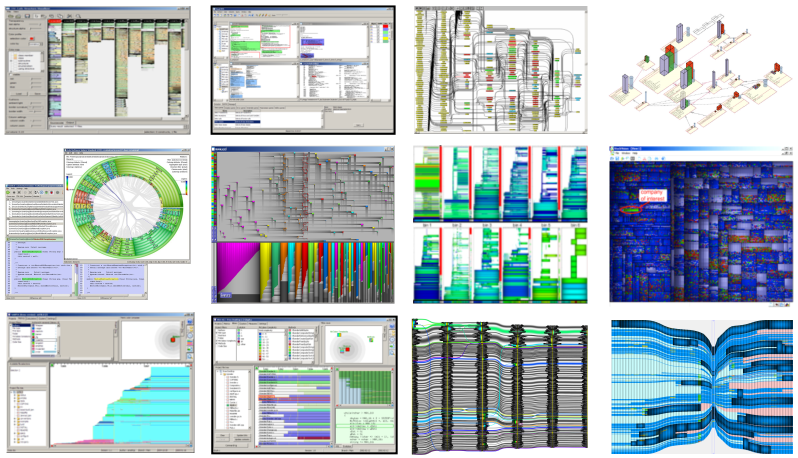


- **40% of docs** frequently updated
- rest seem to be **stale**

Code is **well documented**…
…so refactoring likely doable
Start from up-to-date docs

Conclusions - Software Visual Analytics in Testing

- **Provide insight in multidimensional correlations**
  - Program structure, dependencies, metrics, development/testing effort, documentation
  - Evolution of all these aspects in time
- **Added value**
  - Assess testing effort
  - Pinpoint hot-spots (where to invest the effort)
  - Make sense of all that 'big data'



Thank you for your interest!

Alex Telea
a.c.telea@rug.nl