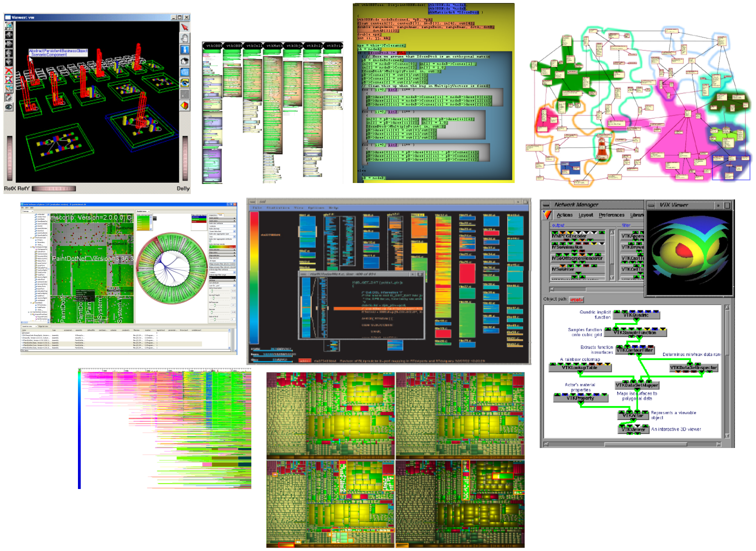
Software Visual Analytics for Maintenance

Example Solutions



prof. dr. Alexandru (Alex) Telea

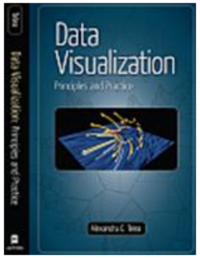
Department of Mathematics and Computer Science
University of Groningen, the Netherlands

Introduction

www.cs.rug.nl/~alext

SolidSource^{IT}

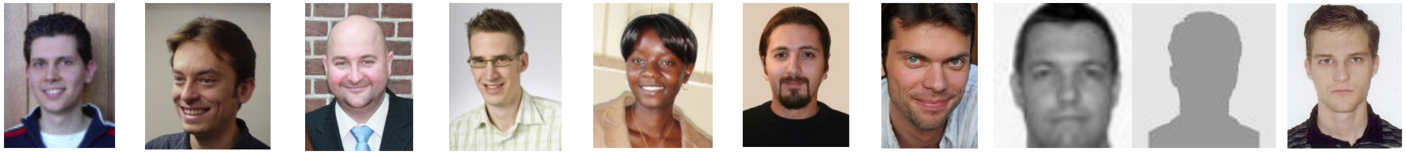
www.solidsourceit.com



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

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

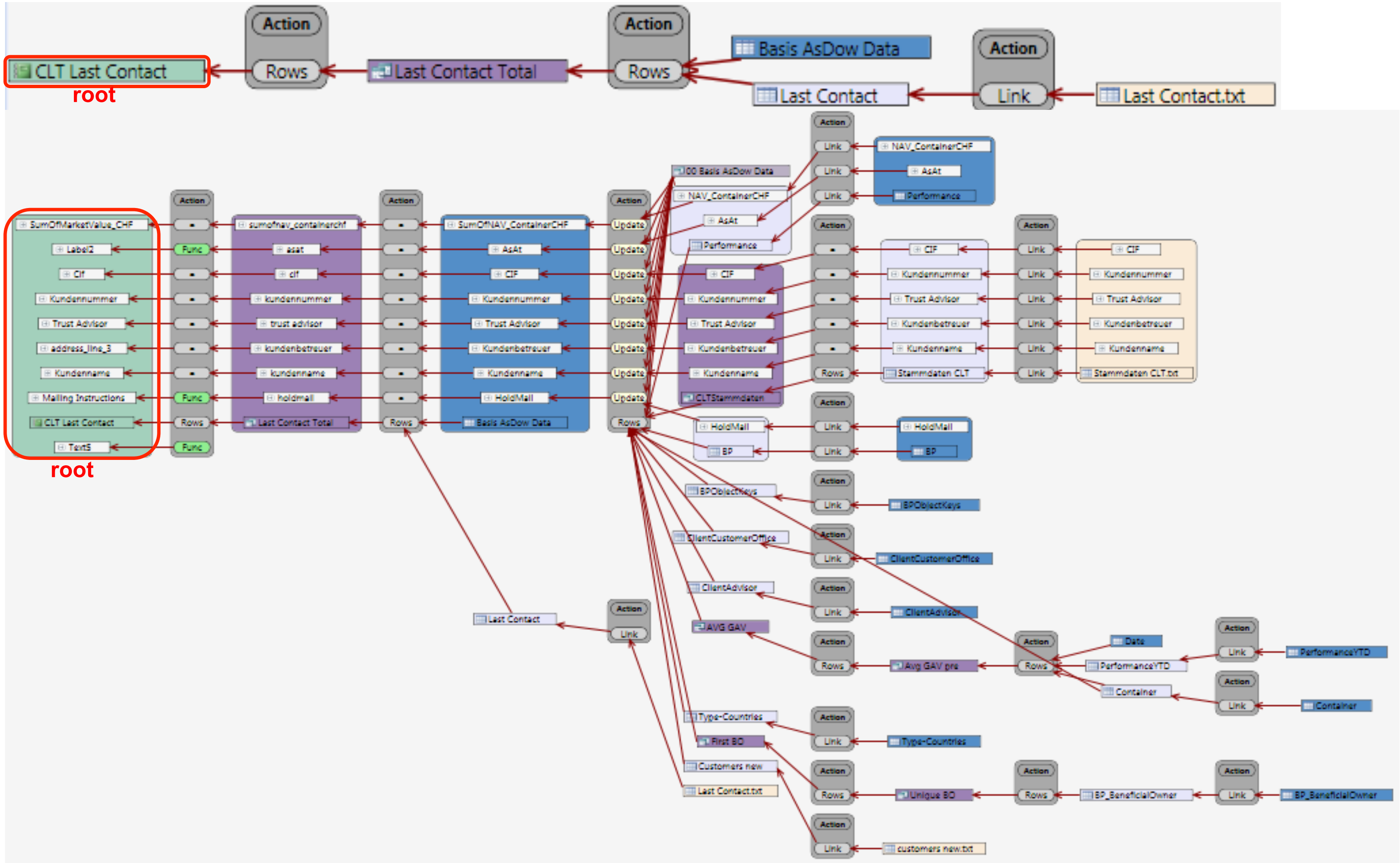
My PhD students…



My MSc students…

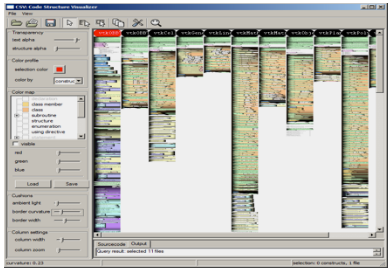
Avdo Hanjalic, Tijmen Klein, Johan v/d Geest, Mark Ettema, Daniel Kok, Karsten Westra, Yuri Meiburg, Hessel Hoogendorp, Liewe Kwakman, Madalina Florean, Bertjan Broeksema, Mark Stoetzer, Sergio Moreta, Kees van Koten, Frans Boerboom, Arjan Janssen, Freek Nossin, Matthijs van Eede, Martijn van Dortmont,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

Software Visualization?

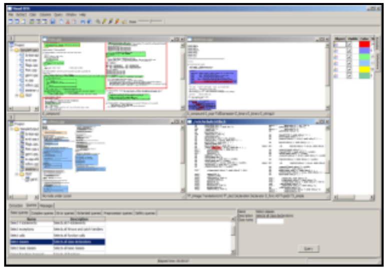


More examples: visualcomplexity.com

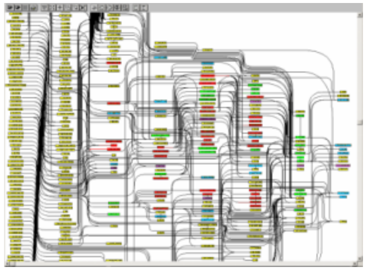
Software Visualization!



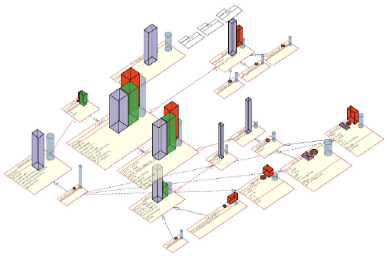
source code



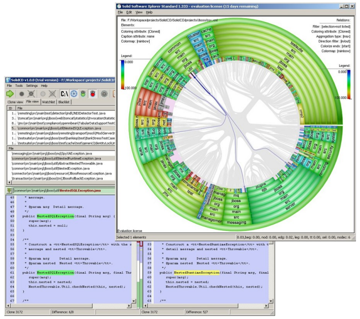
code quality



code dependencies



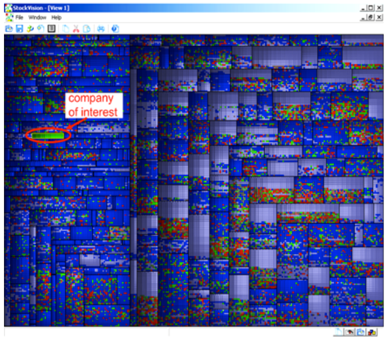
design and metrics



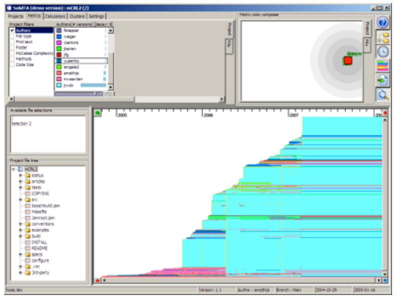
text duplication

How should we deal with **scale**?

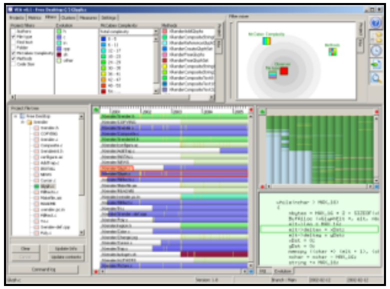
- simplified visualizations?
- continuous simplification?
- what to simplify exactly?
- reinvent wheel for each app?



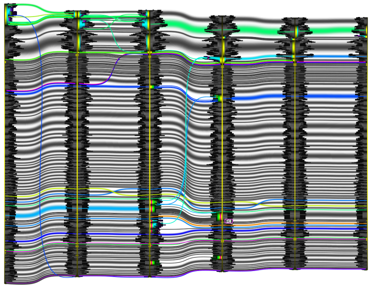
program dynamics



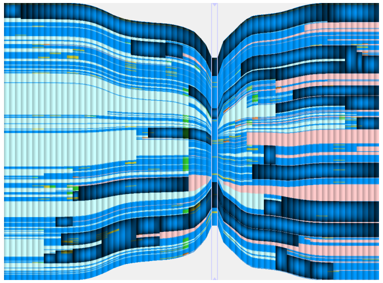
code repositories



evolution metrics



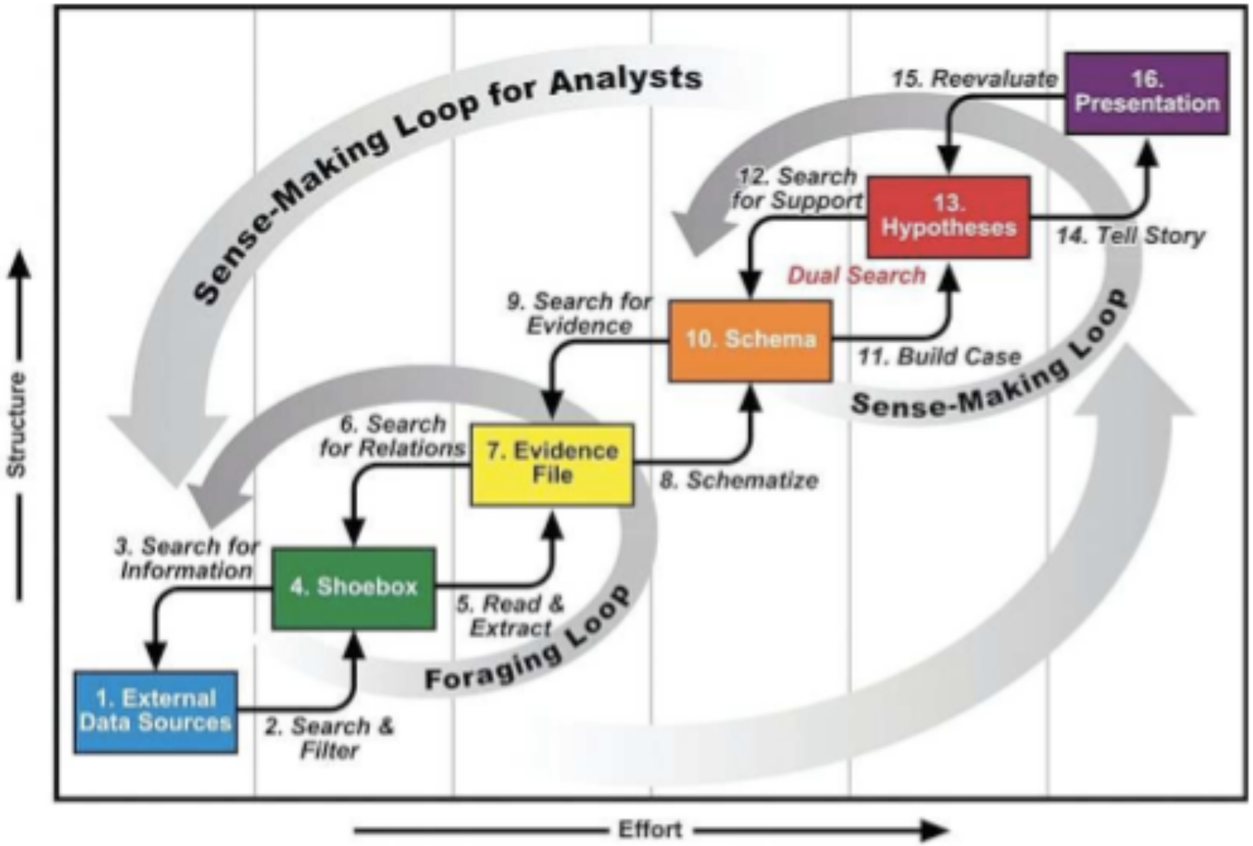
structure evolution



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 Visualization

Definition:

- “The static or animated 2D or 3D visual representation of information about software systems based on their structure, history, or behavior in order to help software engineering tasks” [Diehl, 2006]

Surveys:

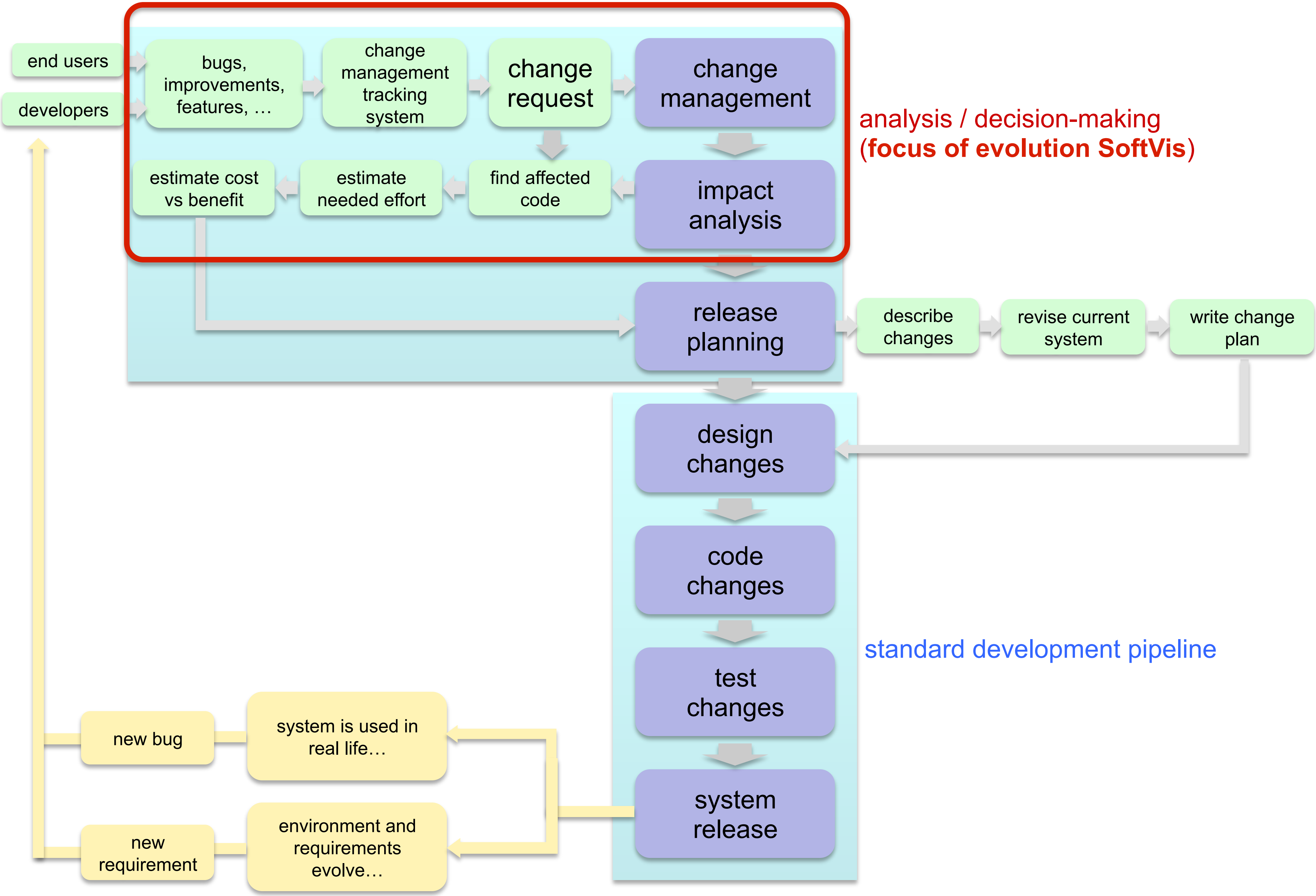
- IT industry: 457 billion $ (2013), **50% larger than in 2008** [www.infoedge.com]
- comparison: total US health care spending 2.5 trillion $ (2009) [www.usatoday.com/news/health]
- **80% of development costs** spent on maintenance [Standish’ 84, Corbi’ 99]
- **50% of this** is spent for understanding the software!

Practice:

- **40% engineers find SoftVis indispensable**, 42% find it not critical [Koschke ’ 02]

Goals: **reduce cost/time, increase quality and productivity!**

Visual Analytics in Software Maintenance



end users

developers

bugs, improvements, features, …

change management tracking system

change request

change management

analysis / decision-making
(**focus of evolution SoftVis**)

impact analysis

find affected code

estimate needed effort

estimate cost vs benefit

release planning

describe changes

revise current system

write change plan

design changes

code changes

test changes

system release

standard development pipeline

system is used in real life…

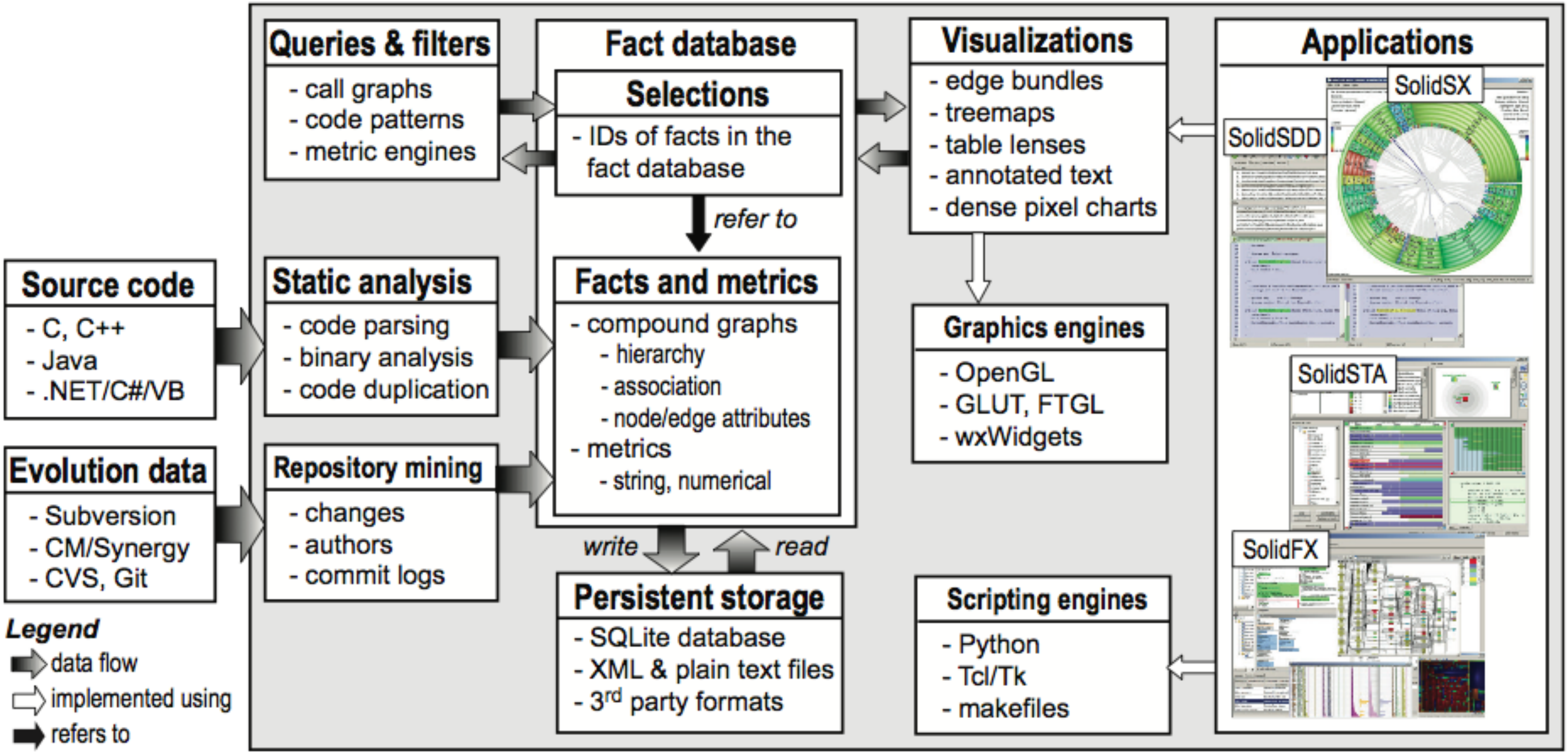
environment and requirements evolve…

new bug

new requirement

Techniques

Software Visual Analytics – Technical View



Source code
- C, C++
- Java
- .NET/C#/VB

Evolution data
- Subversion
- CM/Synergy
- CVS, Git

Queries & filters
- call graphs
- code patterns
- metric engines

Static analysis
- code parsing
- binary analysis
- code duplication

Repository mining
- changes
- authors
- commit logs

Fact database

Selections
- IDs of facts in the fact database

refer to

Facts and metrics
- compound graphs
  - hierarchy
  - association
  - node/edge attributes
- metrics
  - string, numerical

write *read*

Persistent storage
- SQLite database
- XML & plain text files
- 3^{rd} party formats

Visualizations
- edge bundles
- treemaps
- table lenses
- annotated text
- dense pixel charts

Graphics engines
- OpenGL
- GLUT, FTGL
- wxWidgets

Scripting engines
- Python
- Tcl/Tk
- makefiles

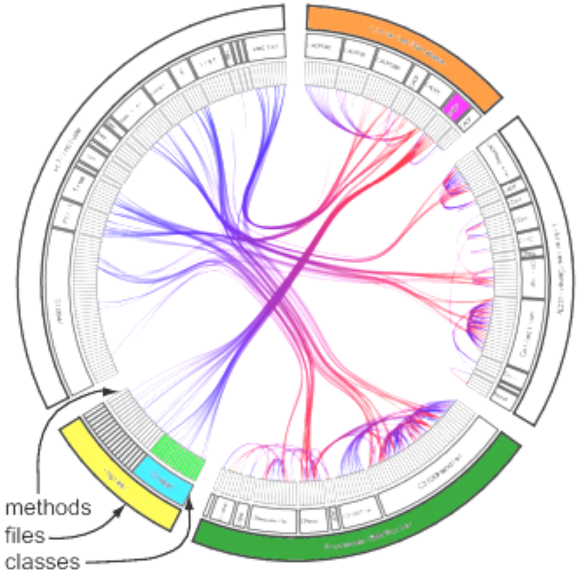
Applications

Legend
- ➡ data flow
- ⇨ implemented using
- ➡ refers to

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

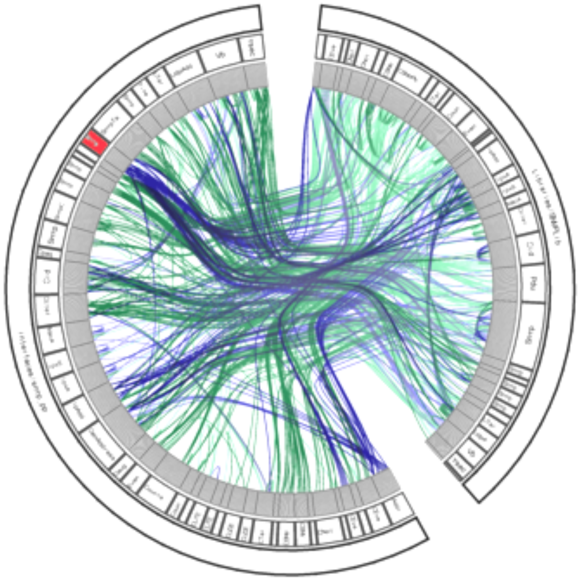
1. Assessing system modularity

Modular system



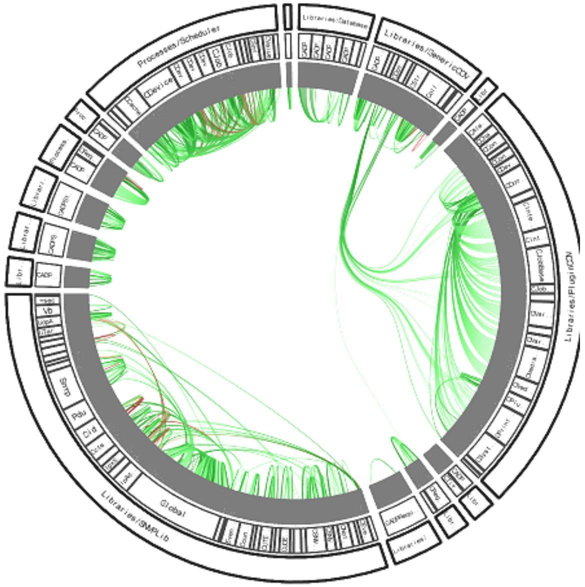
- blue = caller, red = called
- all functions in the yellow file call the purple class
- green file has many self-calls

Monolithic system



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

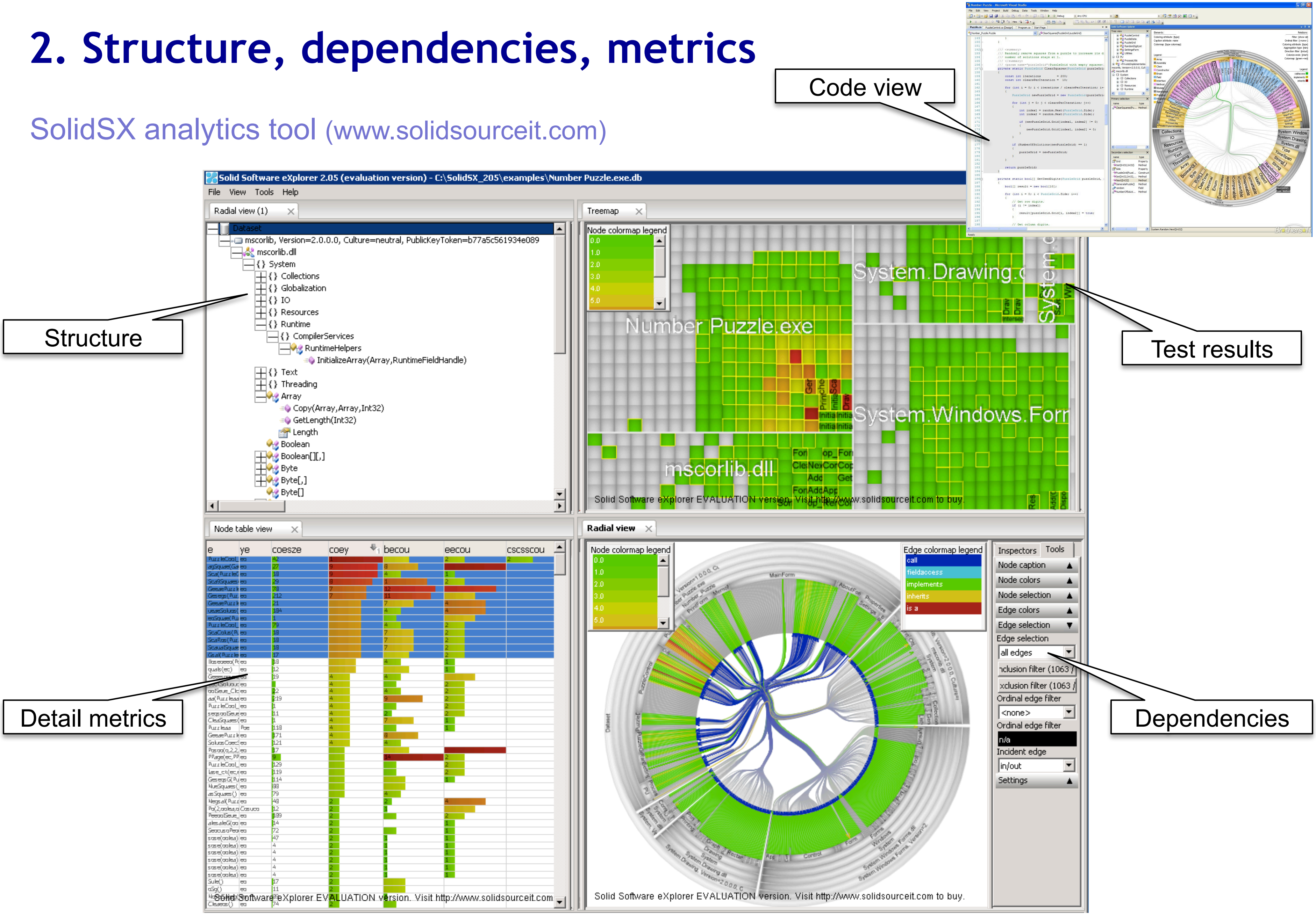
Decoupled system



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

2. Structure, dependencies, metrics

SolidSX analytics tool (www.solidsourceit.com)



Code view

Structure

Test results

Detail metrics

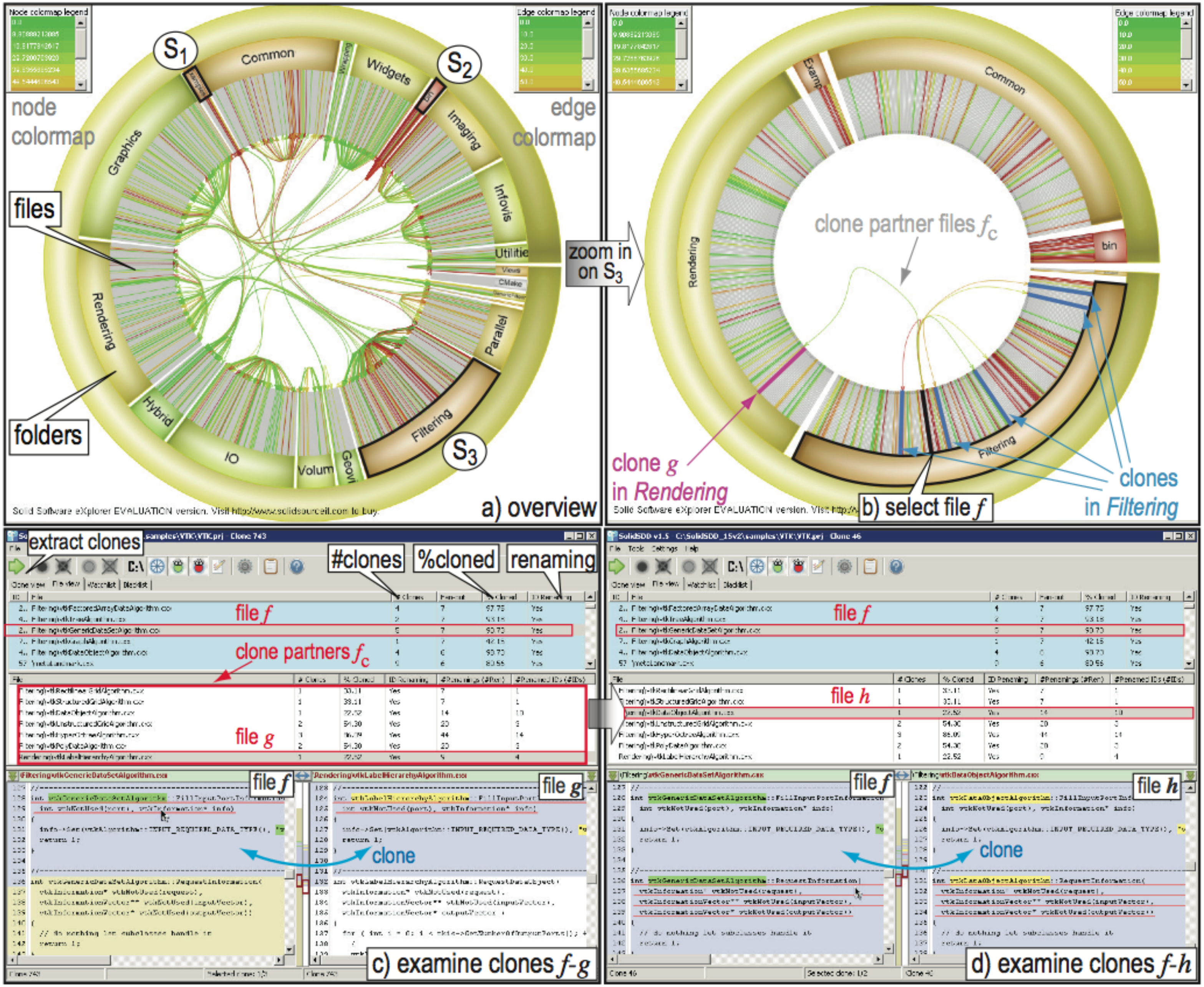
Dependencies

More information: www.cs.rug.nl/svcg/SoftVis/Dependencies

www.cs.rug.nl/svcg

3. Code duplication

SolidSDD tool (www.solidsourceit.com)

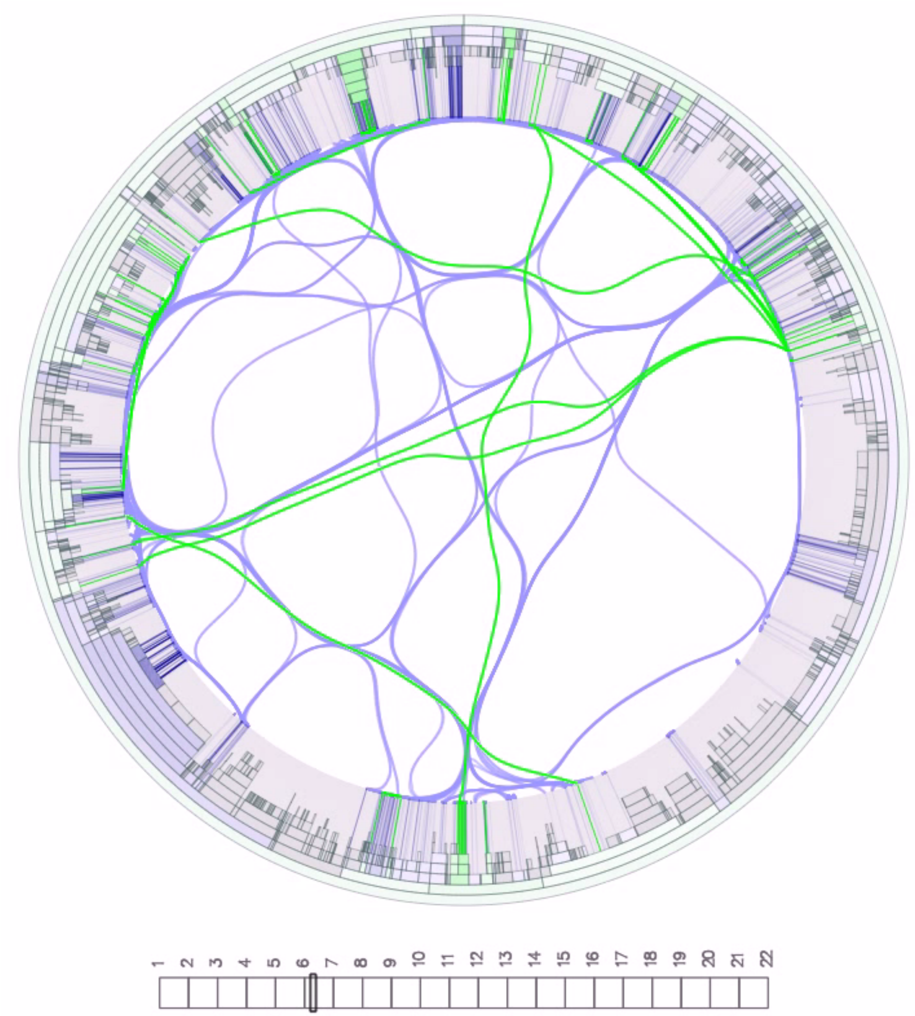


Color legend ☐ non-cloned code 🟨 clone partner not shown ☐ clone shown in both windows 🟩 identifier renamed (shown as 🟨 in right window)

4. Clone evolution

Questions

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

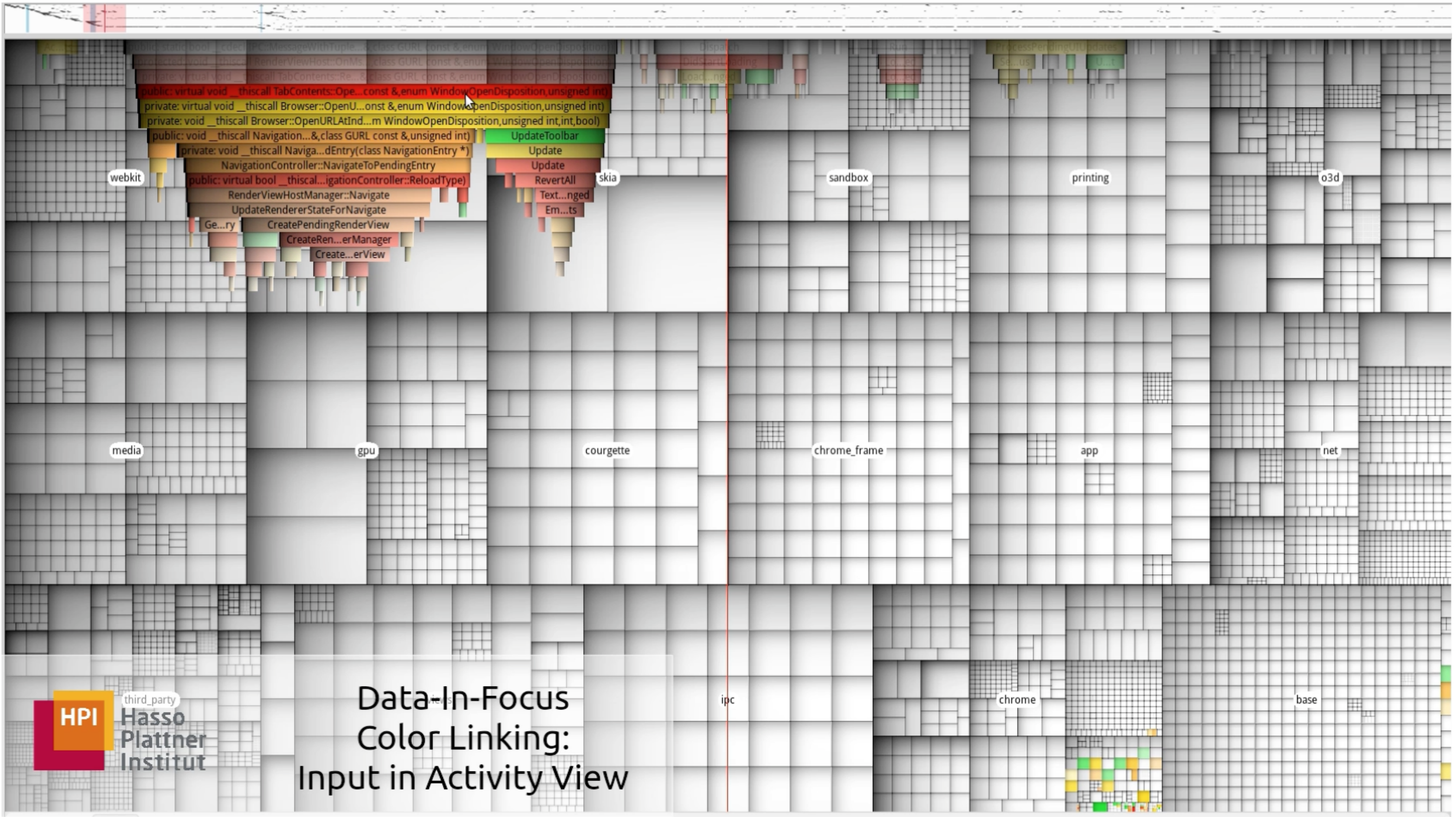


Evolution of clones in Mozilla Firefox (~55K clone relations, 3.5 MLOC C/C++)

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?



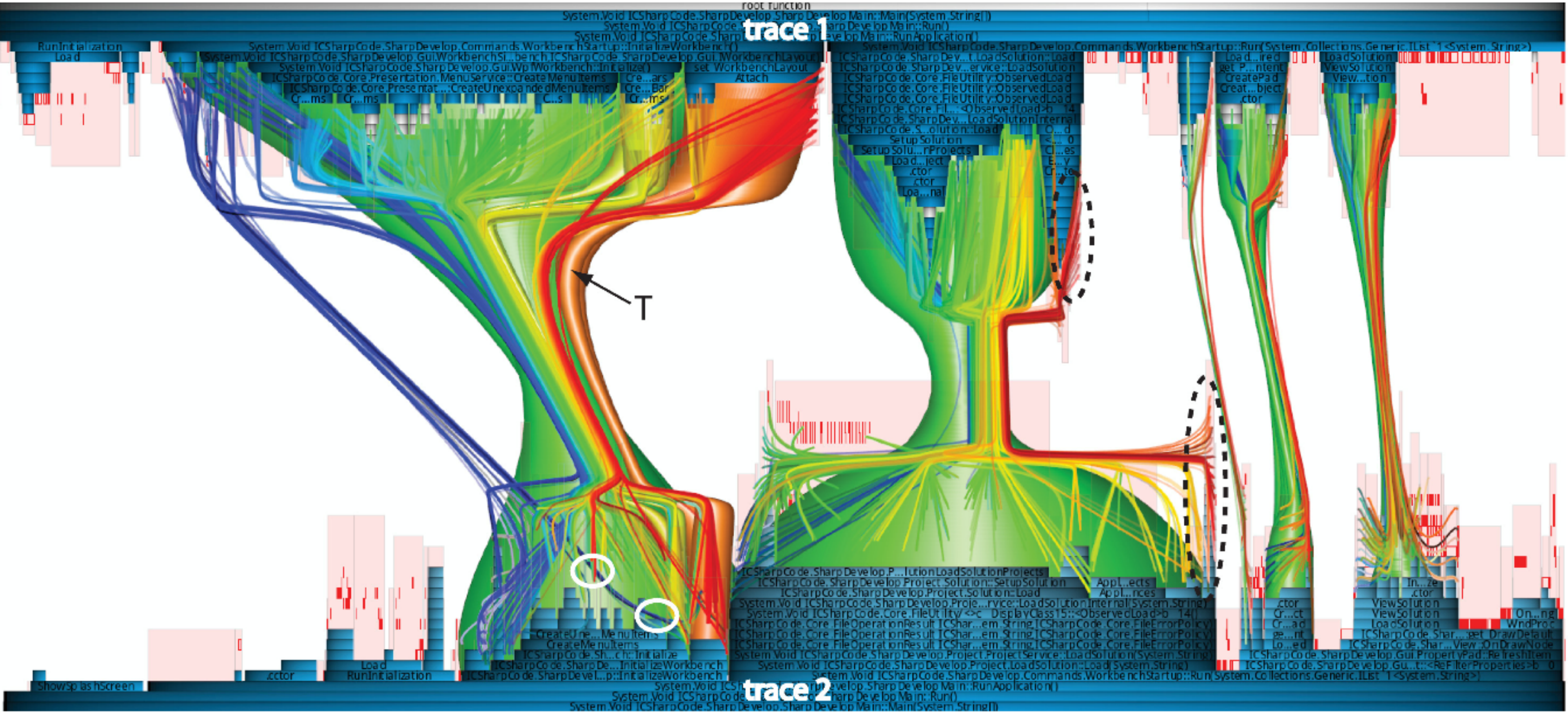
Code: Chrome browser (2.7 MLOC C/C++, 8900 files+folders)
Trace: 9000 calls to 914 functions

More details: www.cs.rug.nl/svcg/SoftVis/ViewFusion
Tool implementation: www.softwarediagnostics.com

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

More details: www.cs.rug.nl/svcg/SoftVis/TraceDiff

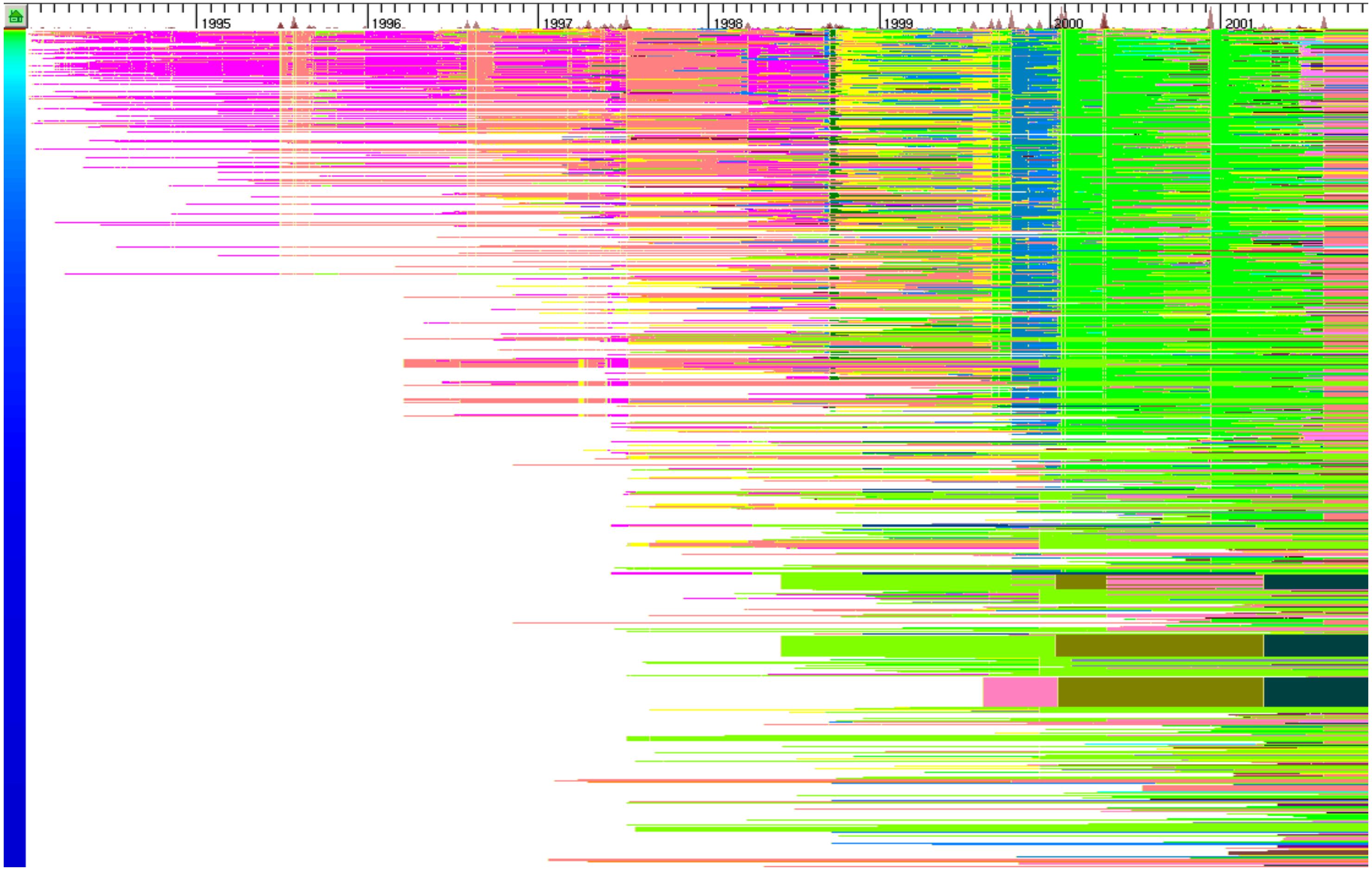
Tool implementation: www.softwarediagnostics.com

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)?

time (revision)



most active files

files sorted by activity

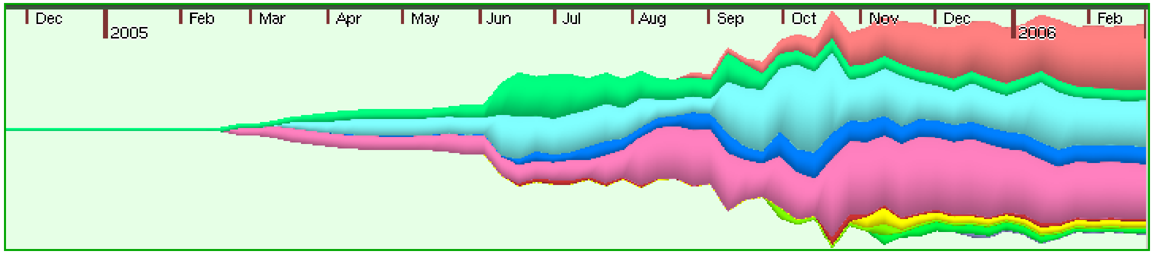
Tool implementation: www.cs.rug.nl/svcg/SoftVis/EvolVis

Analyzing developer effort

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

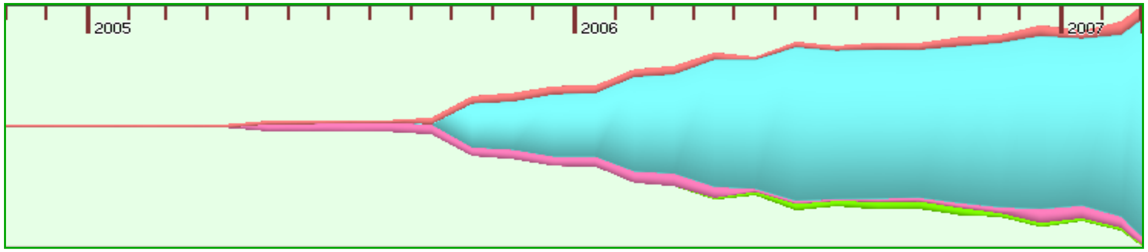
Project A (open-source)

- 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?!

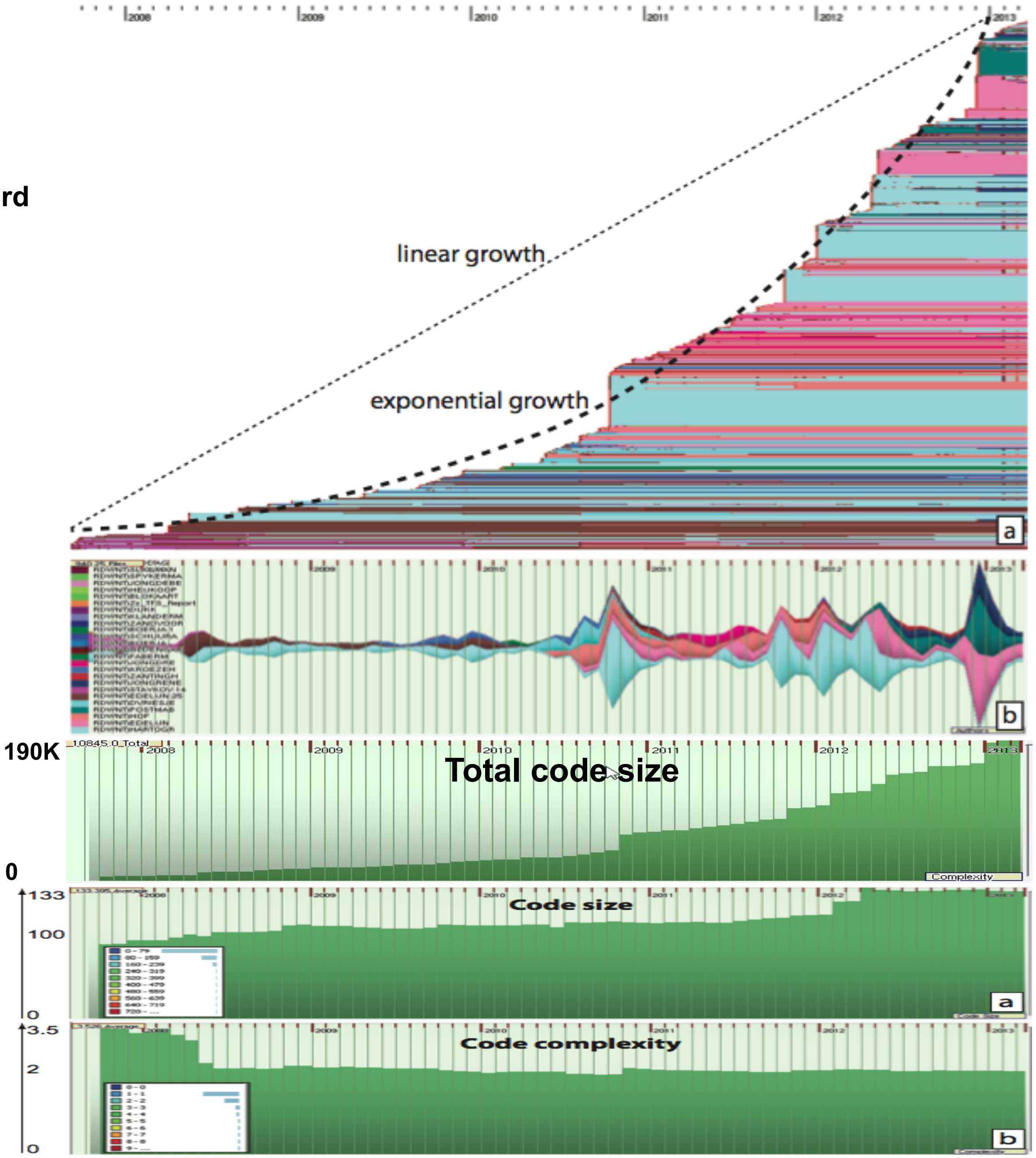


Tool implementation: www.cs.rug.nl/svcg/SoftVis/EvolVis

Correlating quality metrics

- C# code base
- 4 years, 190 KLOC

- Permanent quality monitoring dashboard solution



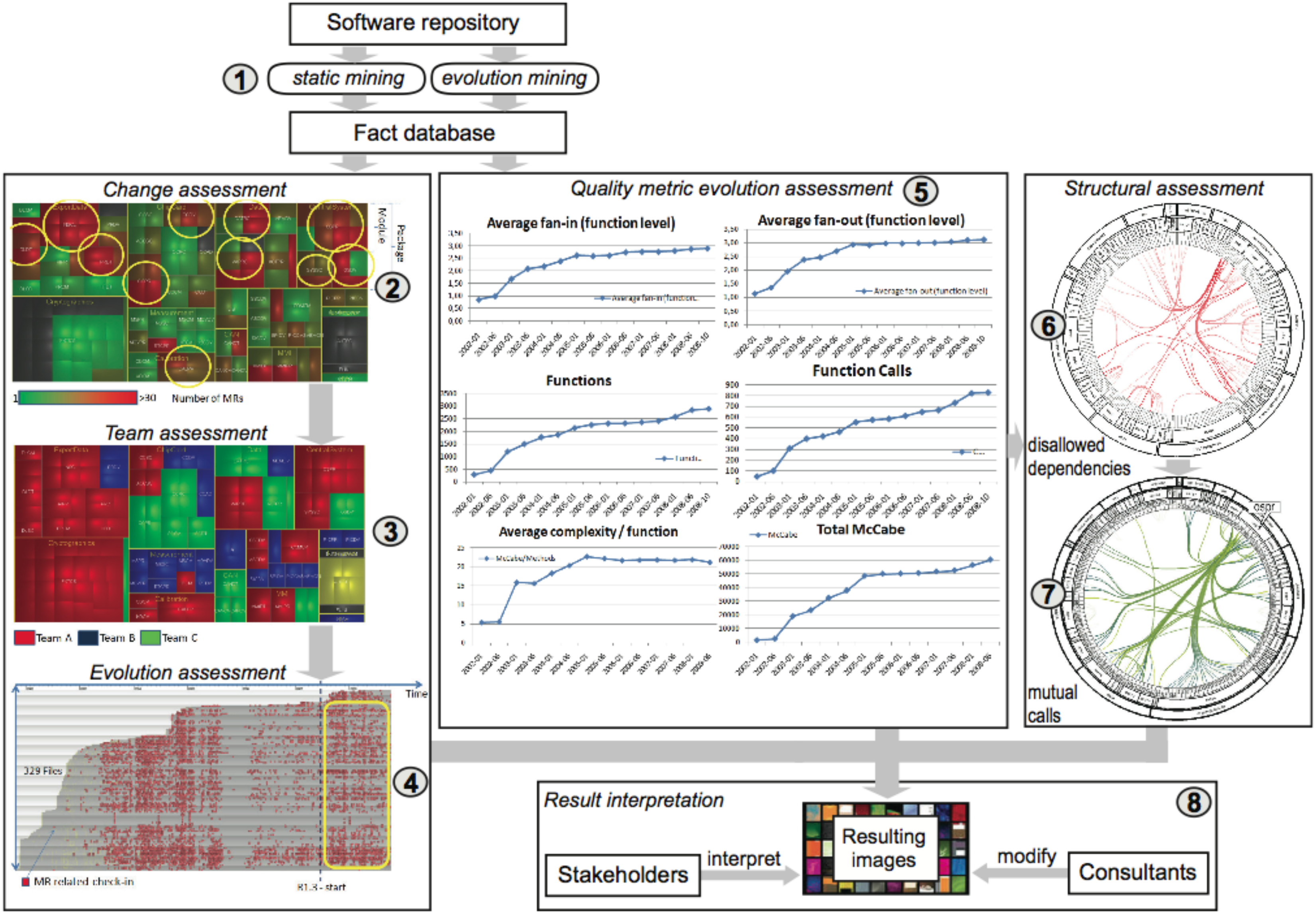
Tool implementation:
www.cs.rug.nl/svcg/SoftVis/EvolVis

Applications

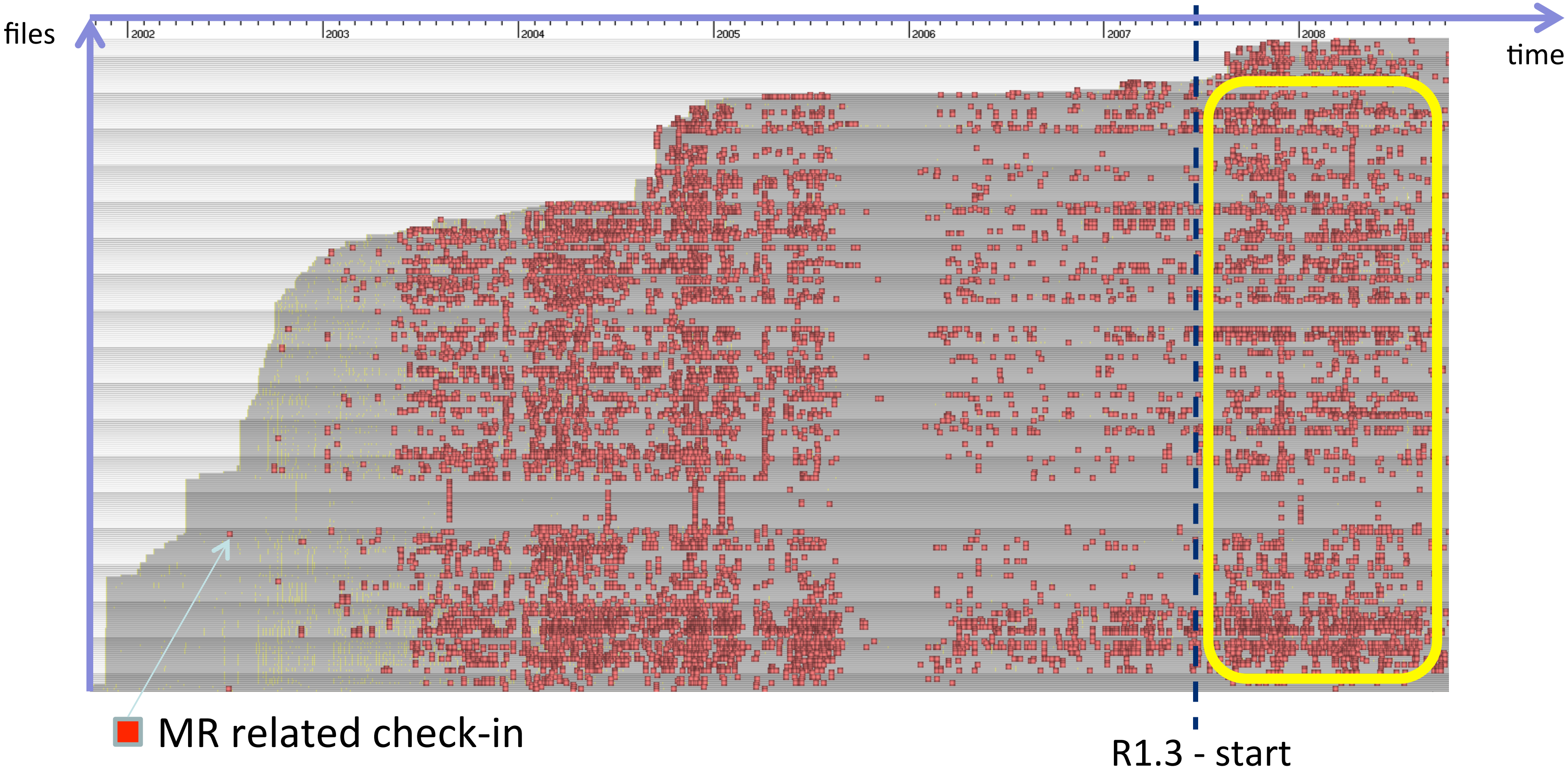
Application: Post-Mortem Assessment

Questions

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

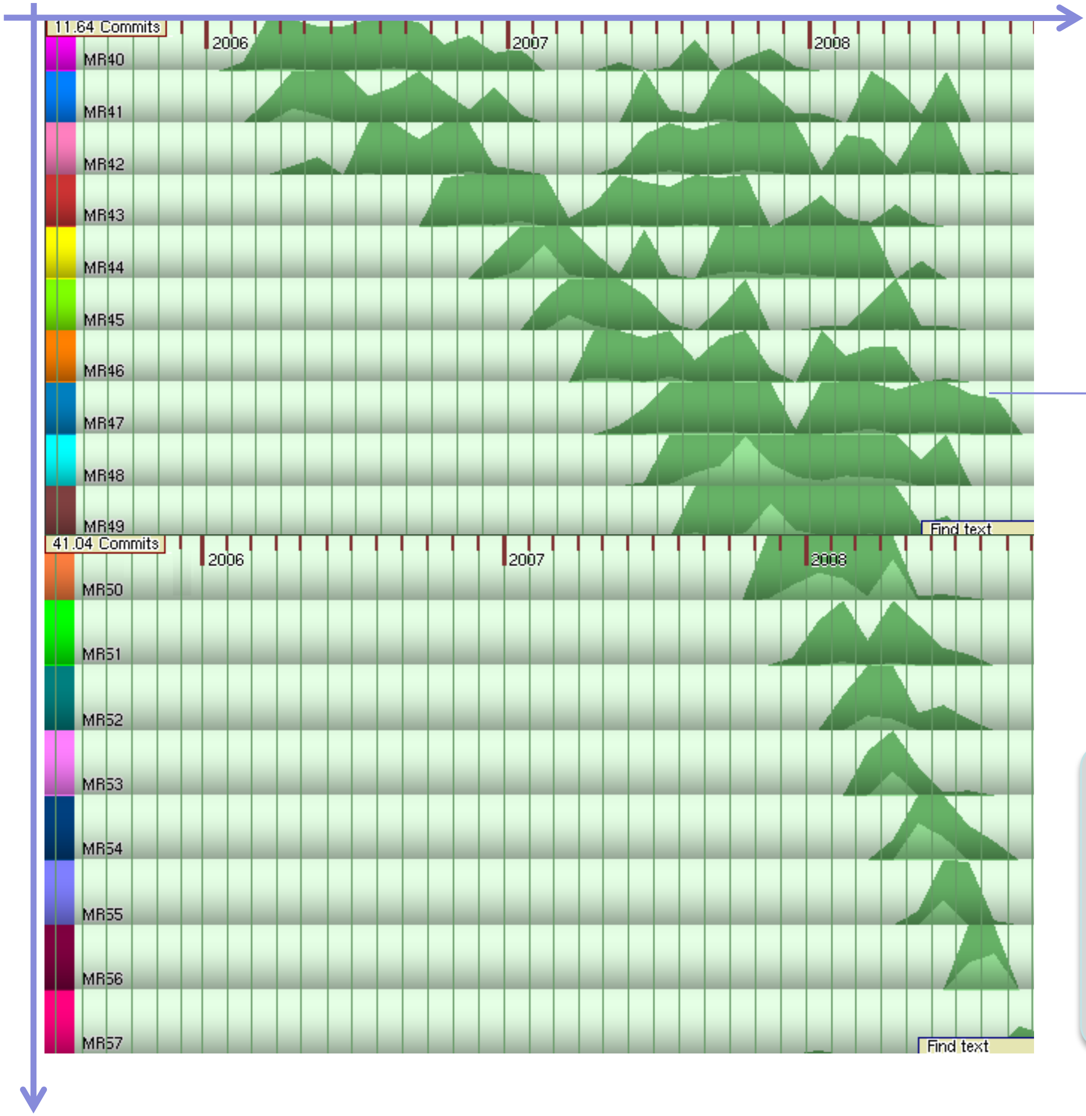


Analysis 1: Modification Request (MR) Lifetime



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

Requirements: MR Duration



time

graph: # commits referring to MRs within a given id range

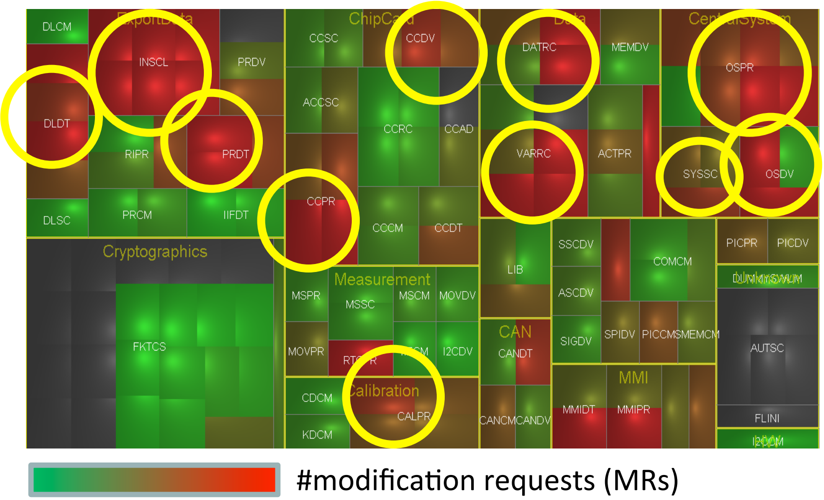
MR ids (1 bar=100 MRs)

In mid 2008, activity related to MRs from 2006 still takes place

Analysis 2: Team Code Ownership

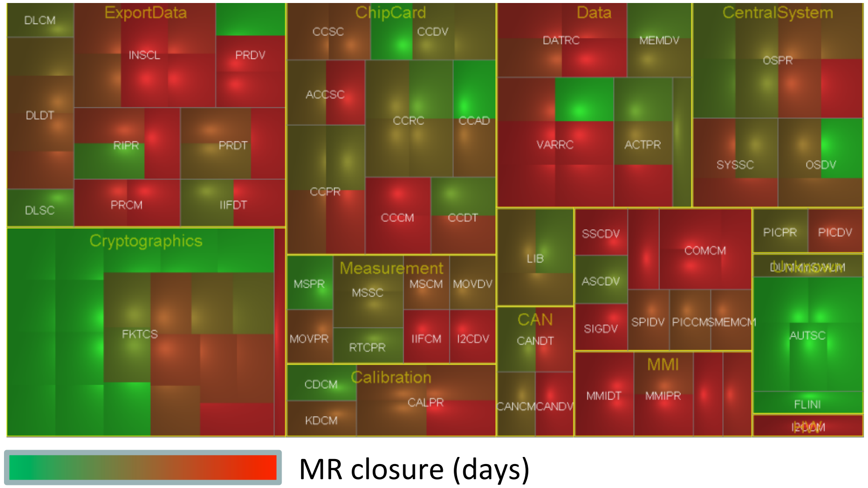
#developers

1 >8



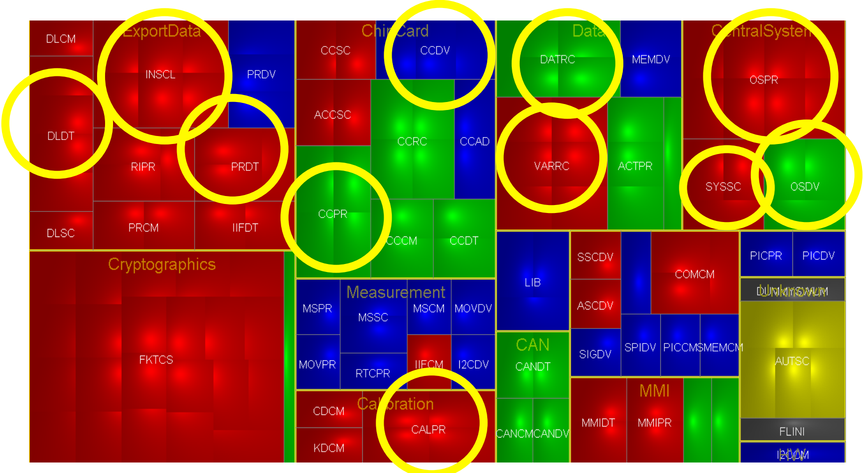
#modification requests (MRs)

1 >30



MR closure (days)

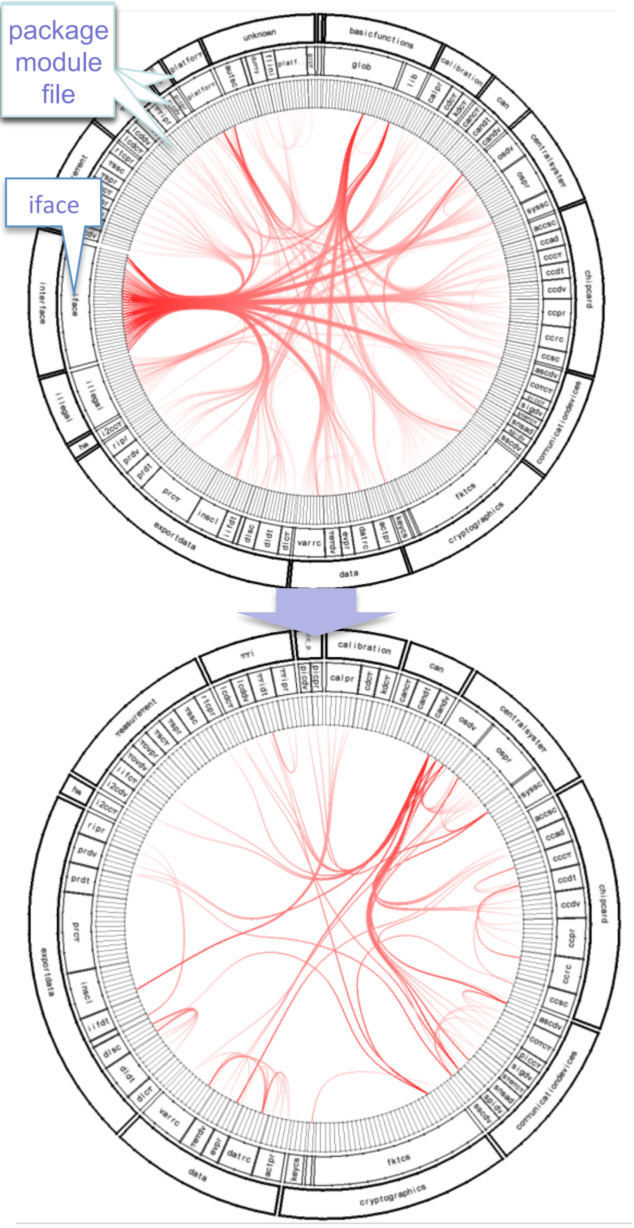
1 >90



🟥 team A 🟦 team B 🟩 team C

⚠ 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, …

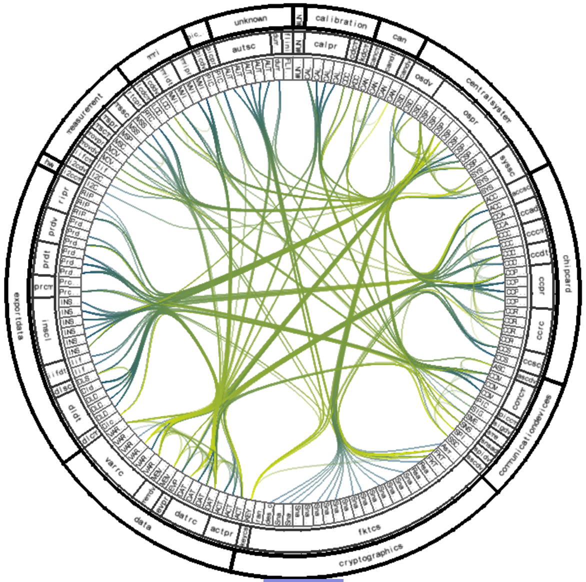
is used

Most dependencies occur via the iface, basicfunctions and platform packages

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

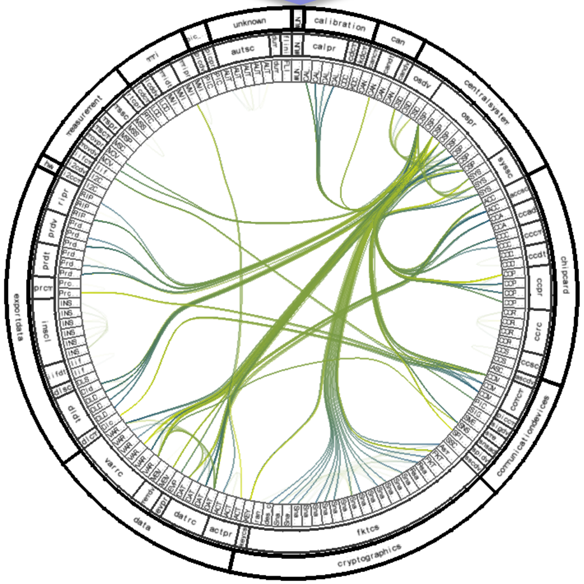
These are accesses that bypass established interfaces
There are several such accesses (bad)

Analysis 4: Code Call graph



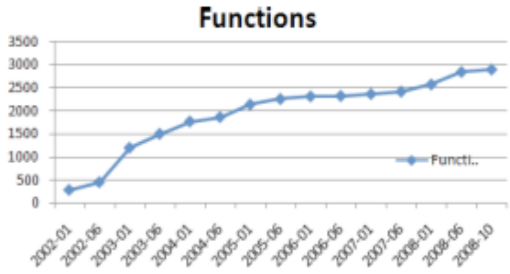
High coupling at package level
This image does not tell us very much

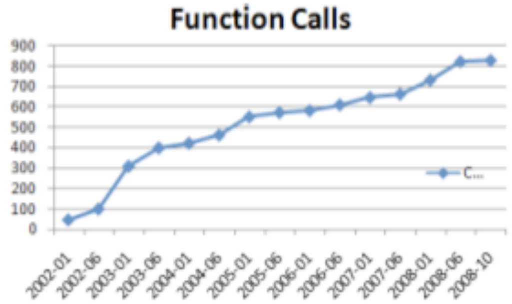
Select only modules which are *mutually call dependent*…
…to discover *layering violations*

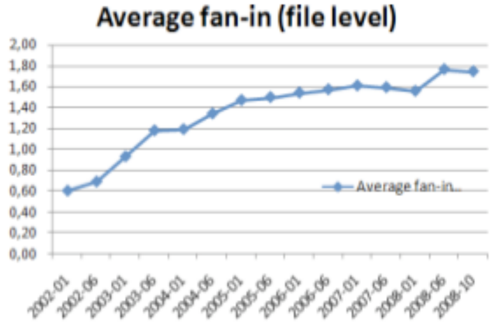


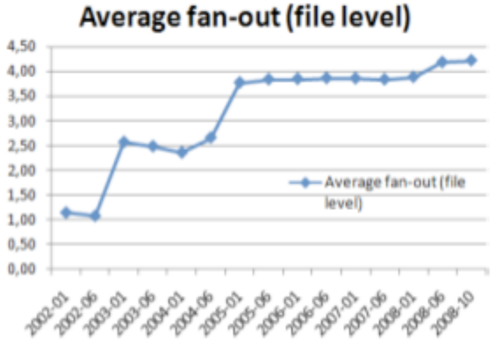
Not a strict layering in the system (as it should be)
Thus, the architecture is violated.

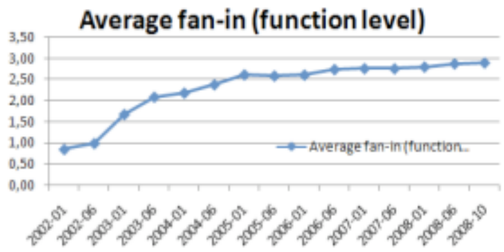
Analysis 5: Code Quality Metrics

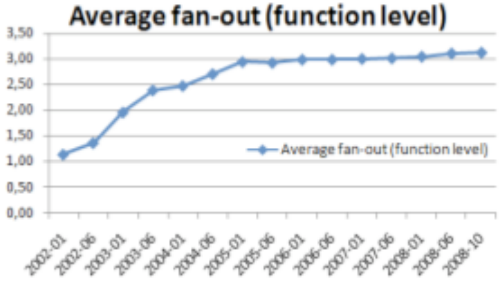

Functions

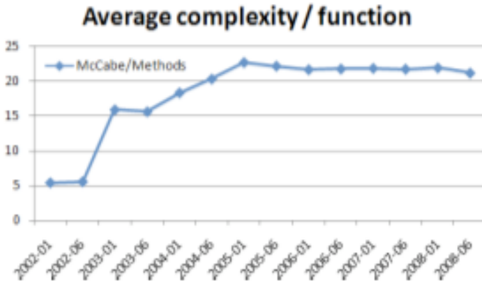

Function Calls

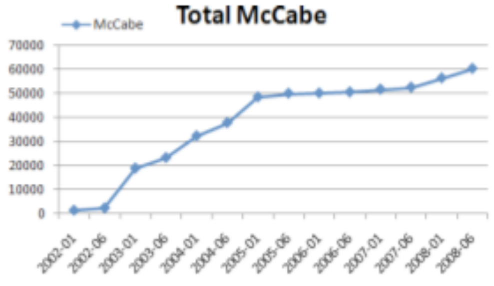

Average fan-in (file level)


Average fan-out (file level)


Average fan-in (function level)


Average fan-out (function level)

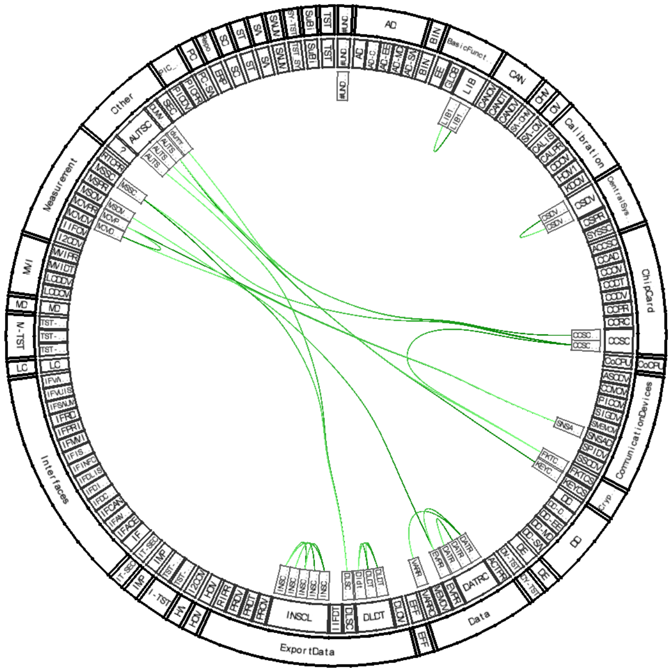

Average complexity / function


Total McCabe

Moderate code + dependency growth
- does not explain products problems

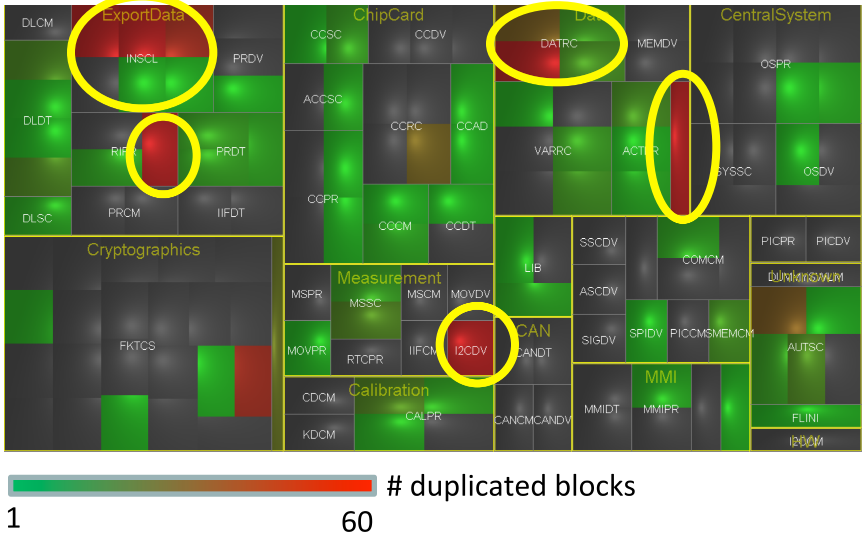
Average complexity/function > 20
Total complexity: up 20% in R1.3
- testing can be hard!
- **possible cause** of product's problems

Analysis 6: Code Duplication



External duplication

- show modules having similar code blocks of >25 LOC

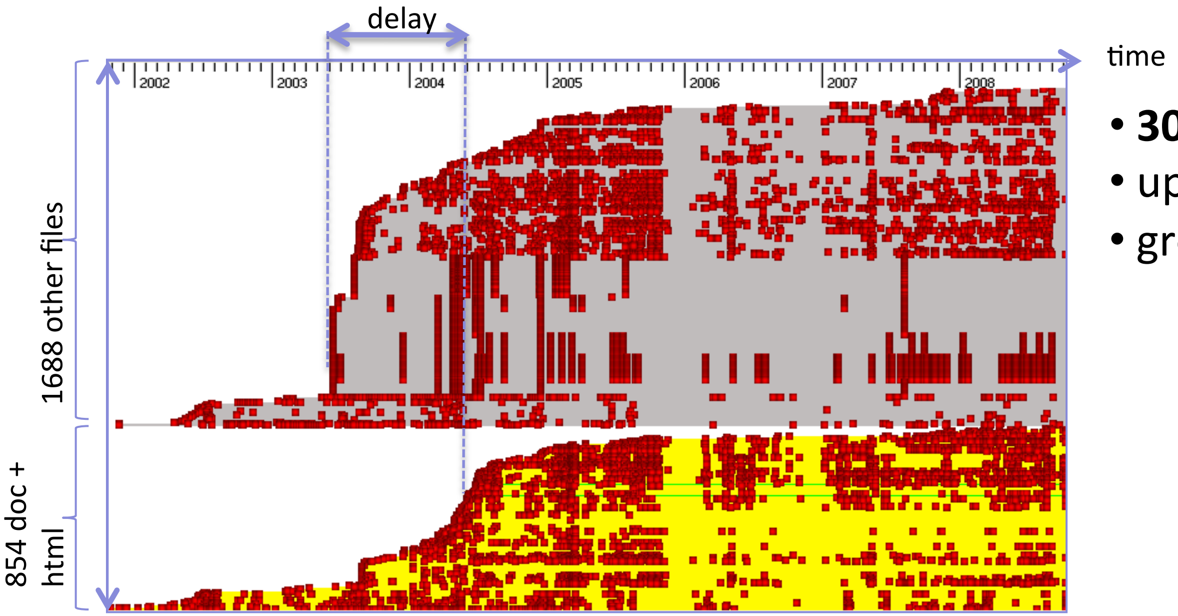


Internal duplication

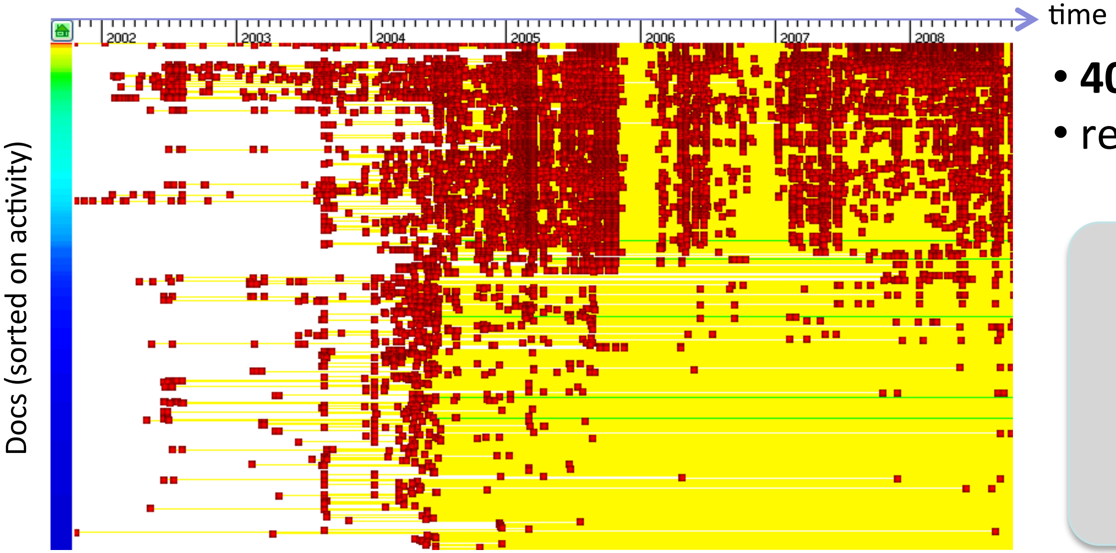
- color: #duplicated blocks within a file

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

Application: Database reverse engineering

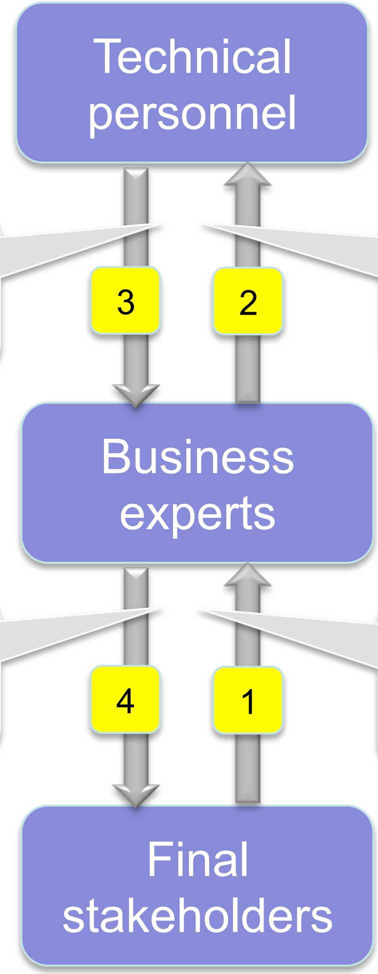
Context

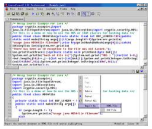
- clients: top-3 Swiss bank
- product: reporting system (2004-2012)
  - Oracle/SQL/MS Access databases
  - ~5000 tables, 60000 fields,
  - mix of TS-SQL, Visual Basic, MS Access
  - code needs **24-hour uptime**
- system was unmaintainable, at end

Questions

- how can we understand the **business logic**?
- how can we refactor the database design for better **maintenance**?

Stakeholders





Source code
(Access, SQL, Toad, Oracle, VB, …)

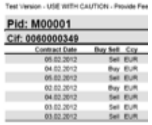
New report implementations
- how to **efficiently** communicate changes to business layer?

Business logic (BL) specifications
- how to **efficiently** translate BL Into technical details?

Business logic
(rules, conditions, limits, …)

New report implementations
- how to **efficiently** validate new reports?

Reporting requests
- business-level specifications
- **tight deadlines**



Test Version - USE WITH CAUTION - Provide Fee

Pid: M00001

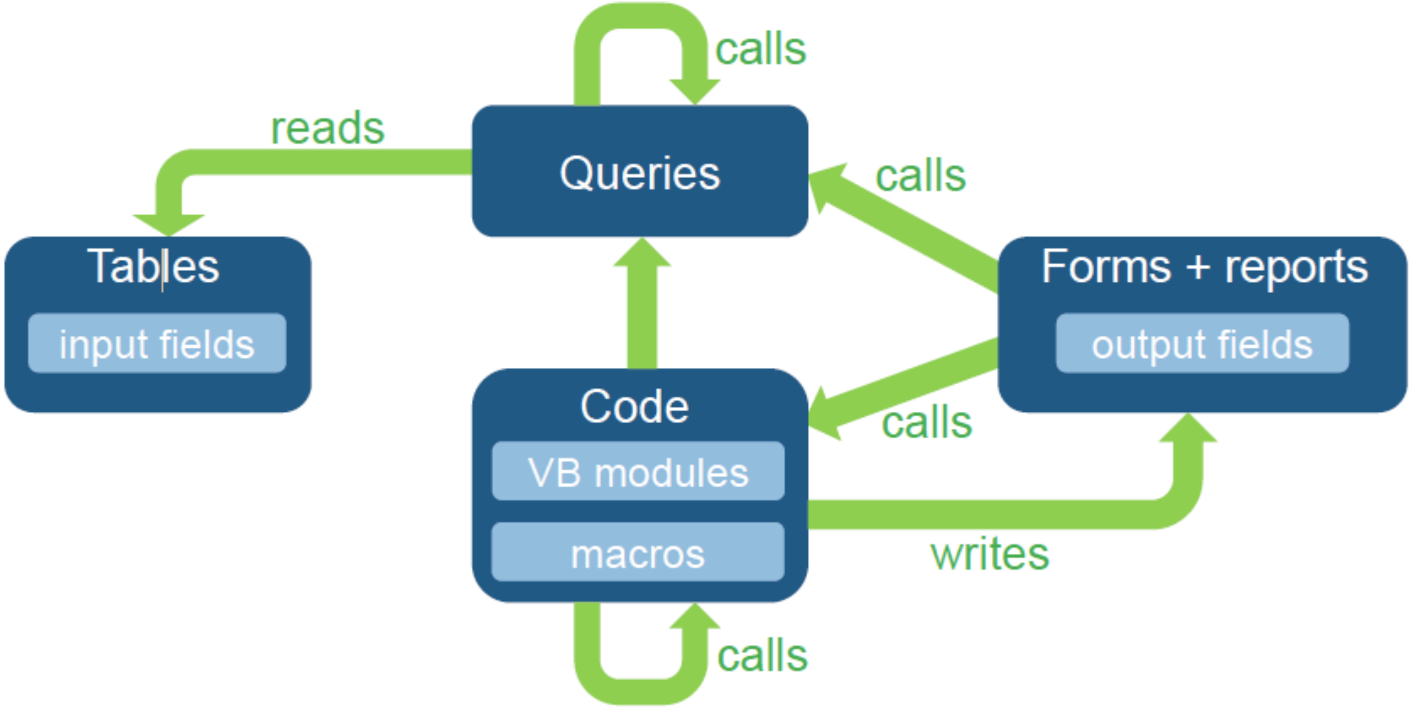
Cif: 0060000349

Contract Date	Buy Sell	Ccy
05.02.2012	Sell	EUR
04.02.2012	Buy	EUR
05.02.2012	Sell	EUR
02.02.2012	Buy	EUR
04.02.2012	Sell	EUR
03.02.2012	Sell	EUR
03.02.2012	Sell	EUR

Business reports
(the final facts & figures)

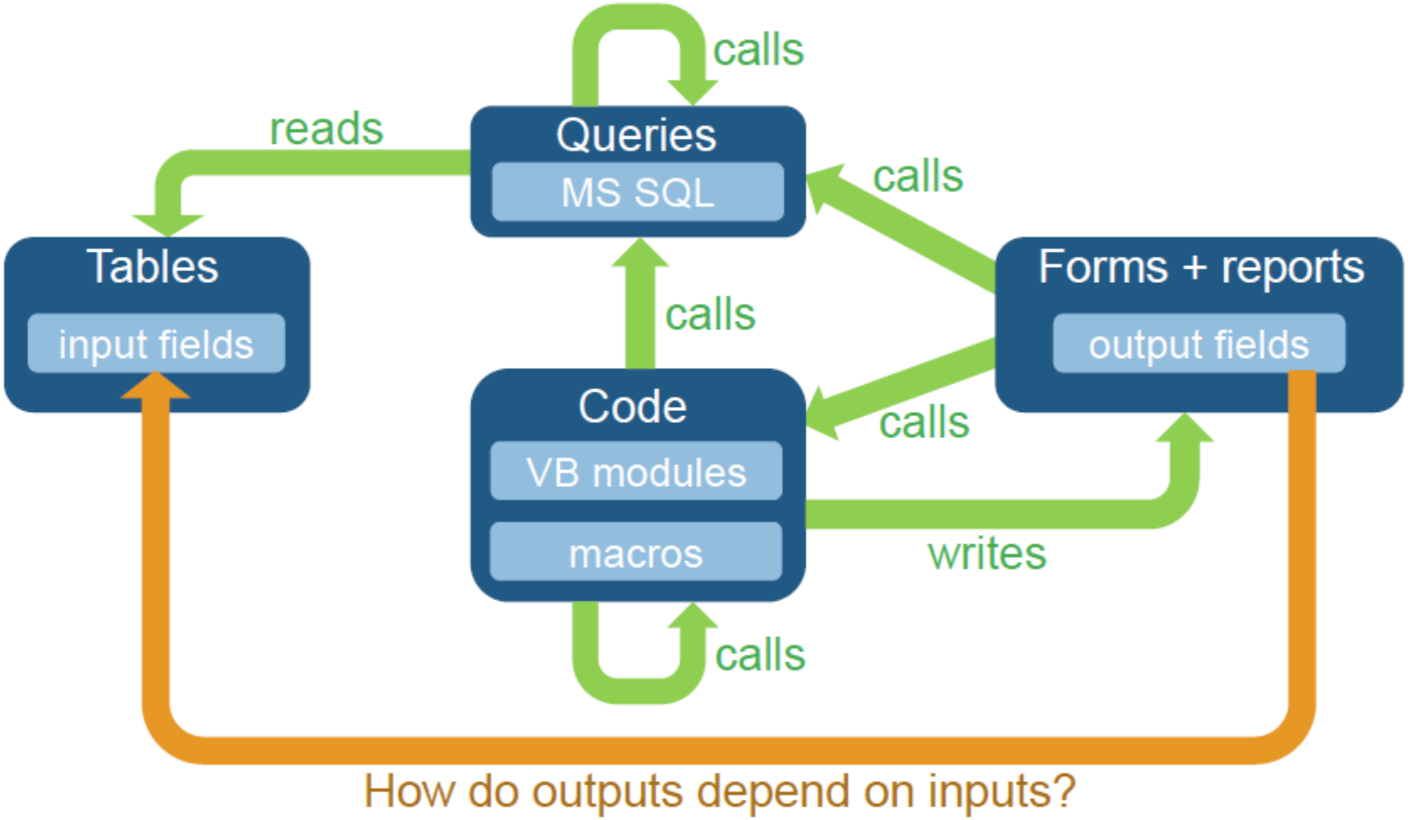
Problem Modeling

Data and control dependencies



Problem Modeling

Data and control dependencies



- one output to many input, one input to many output relations
- hard to find all relations purely statically – dependence on control flow (execution paths)

Problem Dimension

Entity	Measurements (AsDow)
databases processed	39
tables	1’258; **11’235** input fields
reports	66; 1’396 fields; ~21 fields/report; ~8 input/output relations (min.)
queries	3’617
VB code	52 modules; 271 functions; 24 KLOC
macros	22

Entity	Measurements (Reports)
databases processed	58
tables	1’977; **24’474** input fields
reports	351; 5’946 fields; ~17 fields/report; ~4 input/output relations (min.)
queries	2’010
VB code	69 modules; 186 functions; 17 KLOC
macros	110

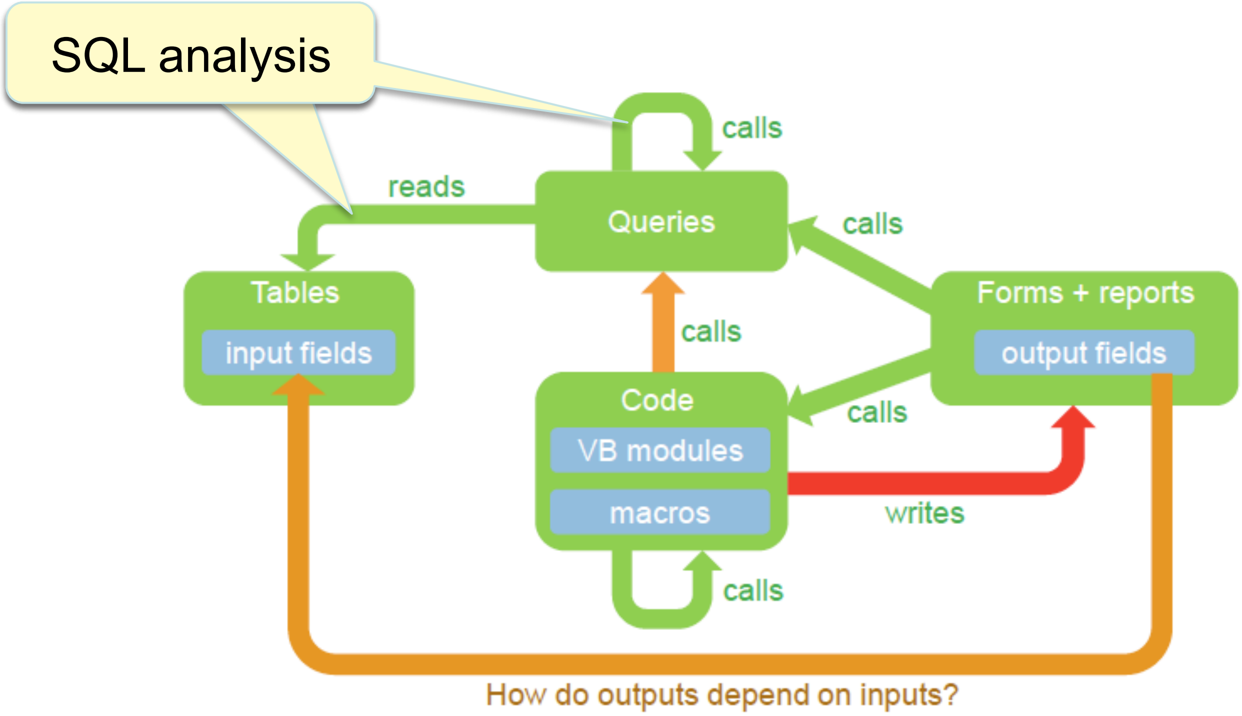
Input parameters

- 35709 input fields
- 5..10 processing steps per path
- assume input fan-out of 4 – very conservative
- total: 4*35709 = 142836 output-to-input field paths

Manual effort needed for impact analysis

- 10 mins/path needed – very conservative
- result: 23806 hours ~ **15 person years!**

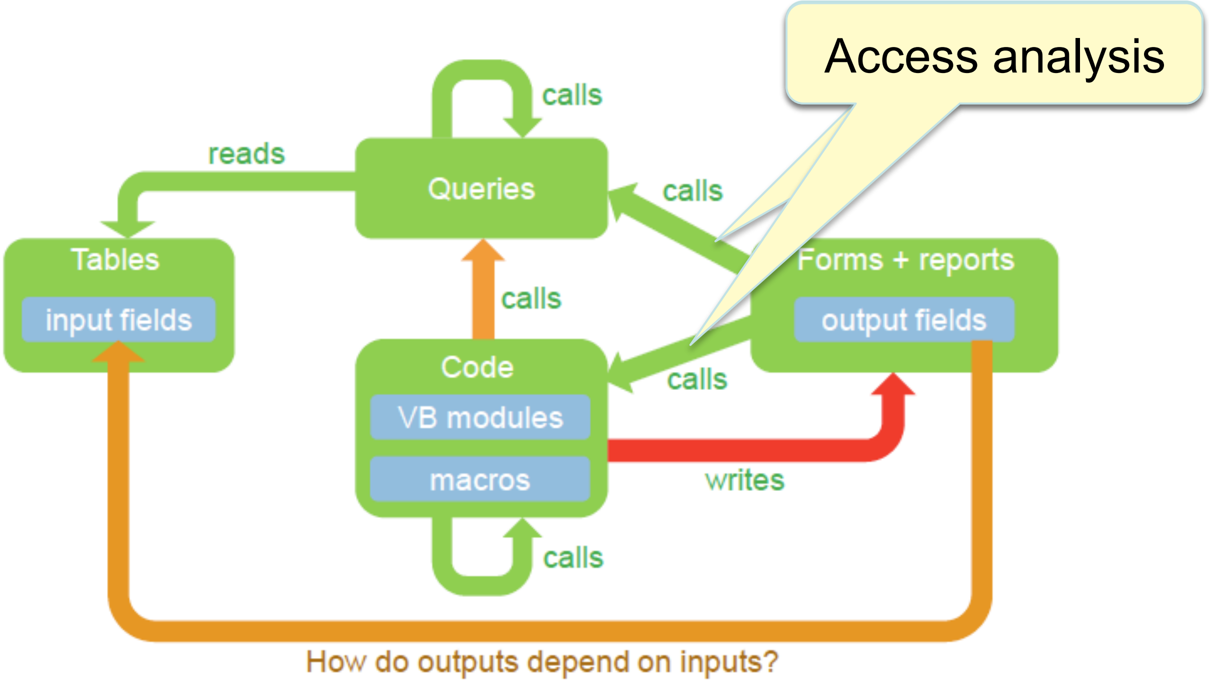
Solution - step 1



SQL analysis

- extract all SQL code (TS-SQL, Oracle, MS Access scripts)
- perform a syntax analysis on SQL (parsing)
- find all read table names and fields (columns)
  - use control flow analysis of SQL

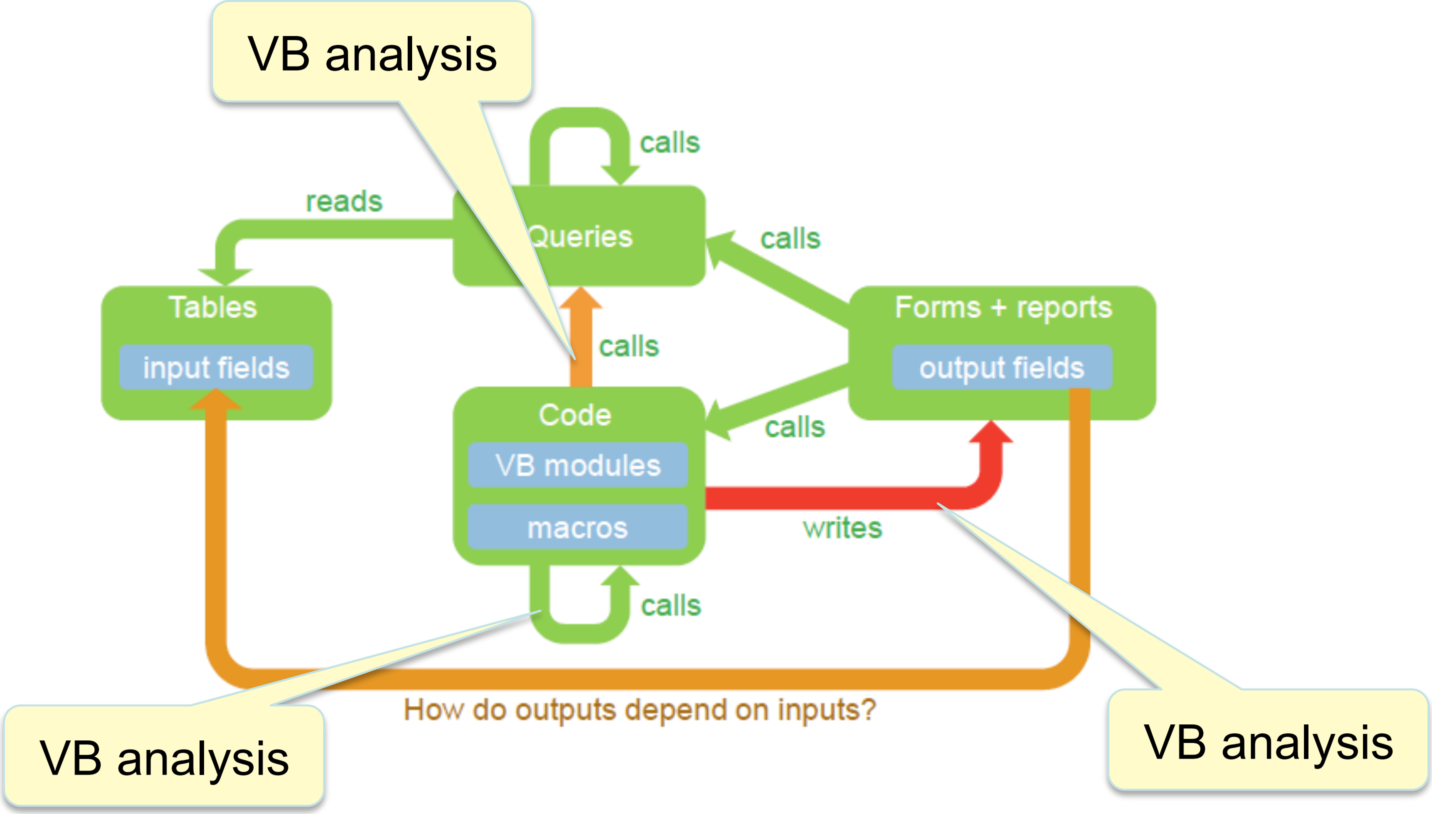
Solution - step 2



MS Access analysis

- extract all MS Access form (report) definitions
- identify fields of interest
- find SQL or VB code that these fields call (if any)

Solution - step 3



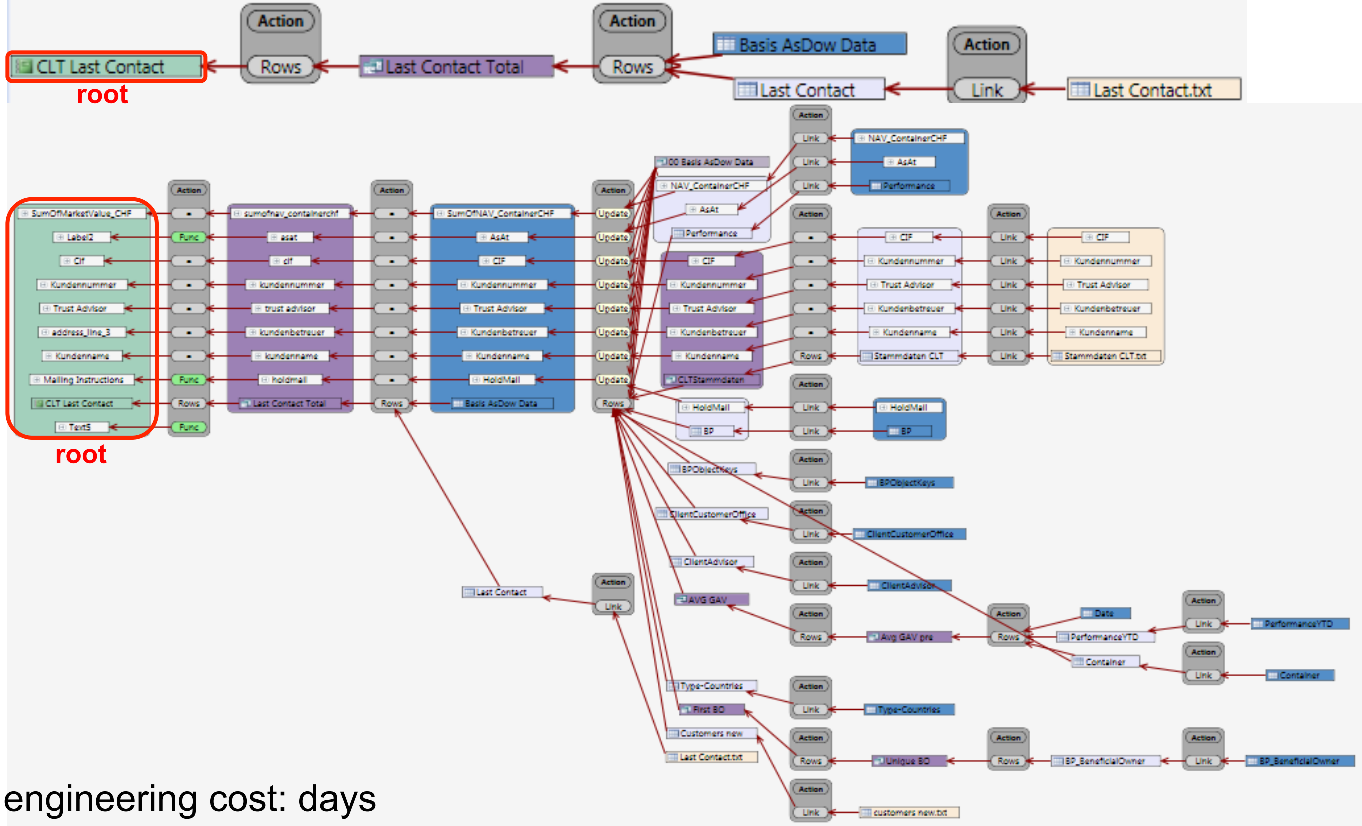
VB analysis

- extract all VB code from the Access reports
- perform a full **syntactic and control-flow analysis**
  - find VB code that writes to report fields — *easy*
  - find how that code is called (call analysis) — *relatively easy*
  - trace back values of output field names to SQL field names using **VB symbolic execution** — *very complex!*

Solution - putting it all together

Access Analyzer (SolidAA)

- end-to-end dataflow analysis across entire reporting platform
- answers question: “Where does this (report) data come from?”
- fully handles any MS Access / SQL database



Benefits

- reverse engineering cost: days
- learning cost: days
- solution cost: ~6 months
- client estimated savings: **~920 KEUR**

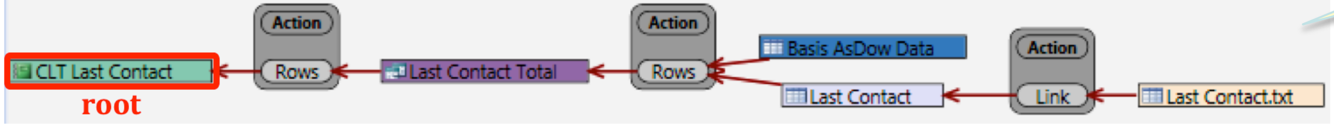
More details on SolidAA: www.soursourceit.com

Solution refinements

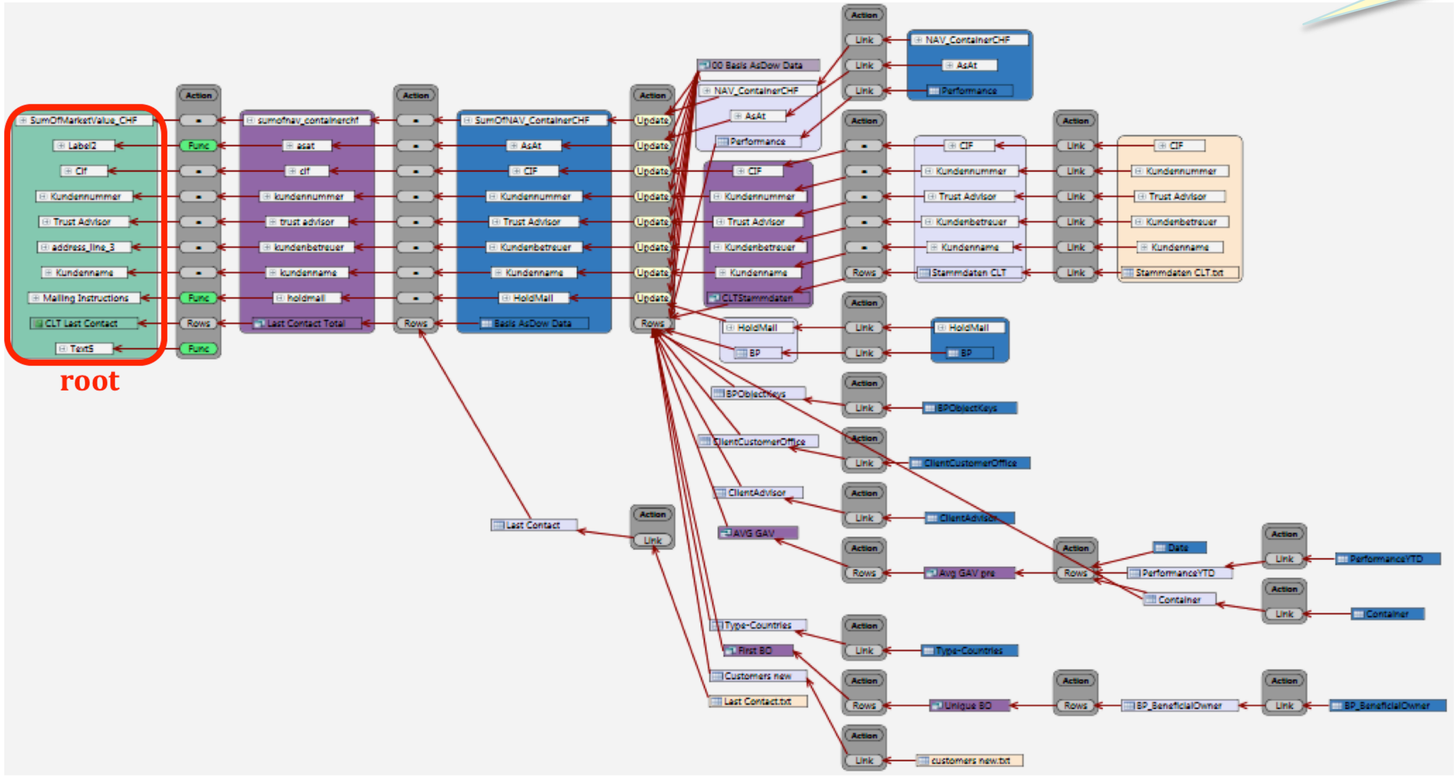
Level of detail

- show dependencies at form and table level only (overview)
- expand to show dependencies at form-field and table-field level (detail)
- detect "parallel paths" of data processing (desirable w.r.t. architecture)

Overview



Detail

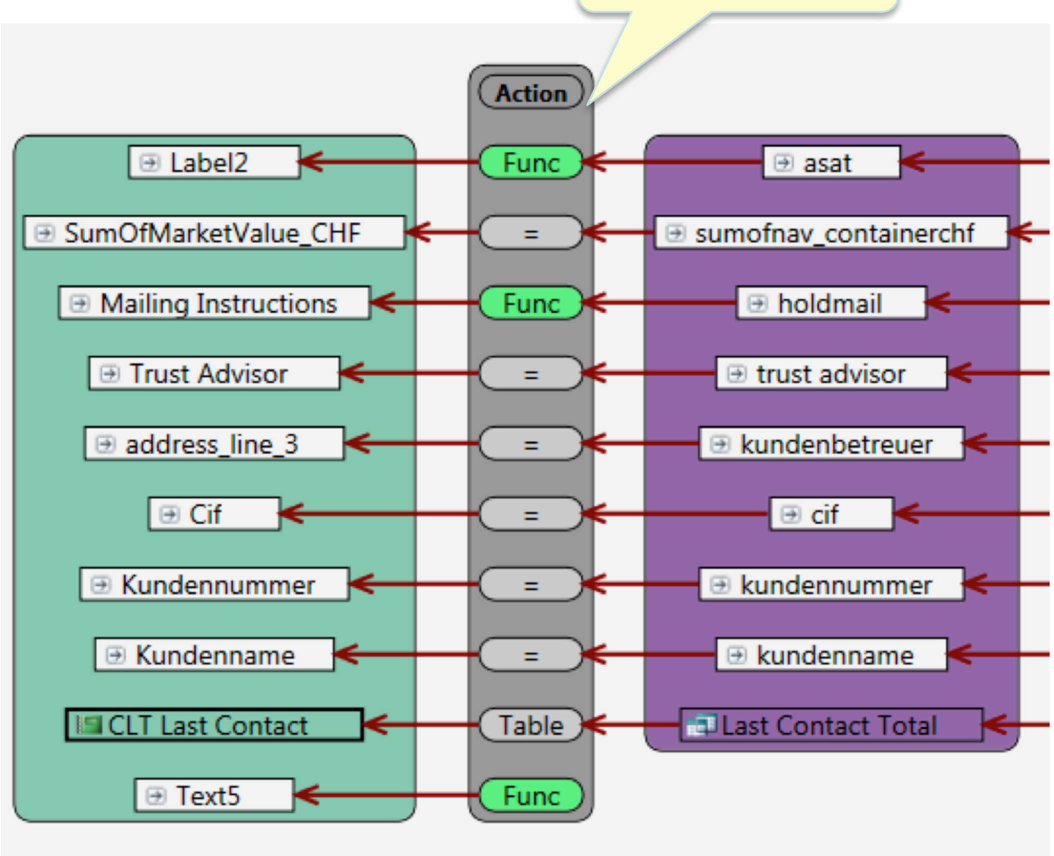


Solution refinements (cont.)

Level of detail (cont.)

- show how the data is **transformed** from input to output
  - copy, SQL function call, references (e.g. table links), …

Transformations

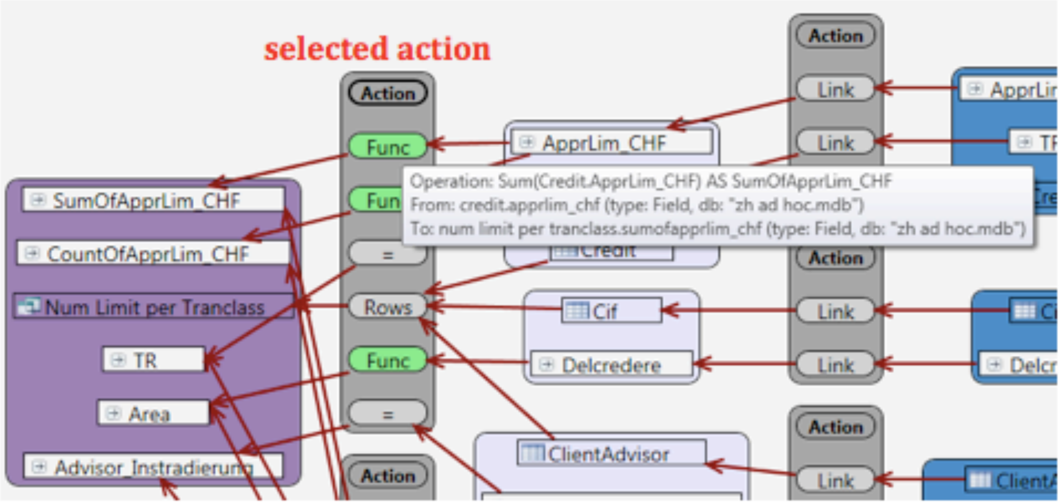


More details on SolidAA: www.soursourceit.com

Solution refinements (cont.)

Links to code (cont.)

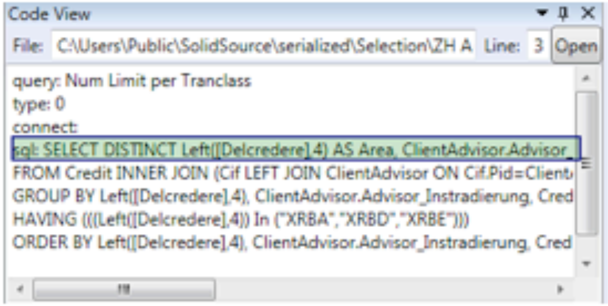
- show which bits of code (SQL, VB, Access, …) are responsible for each data link
- useful for fine-grained detail and understanding of the business logic



```
Operation: Sum(Credit.ApprLim_CHF) AS SumOfApprLim_CHF
From: credit.apprlim_chf (type: Field, db: "zh ad hoc.mdb")
To: num limit per tranclass.sumofapprlim_chf (type: Field, db: "zh ad hoc.mdb")
```

dependency view

action code



File: C:\Users\Public\SolidSource\serialized\Selection\ZH A Line: 3

```
query: Num Limit per Tranclass
type: 0
connect:
sql: SELECT DISTINCT Left([Delcredere],4) AS Area, ClientAdvisor.Advisor_
FROM Credit INNER JOIN (Cif LEFT JOIN ClientAdvisor ON Cif.Pid=Client
GROUP BY Left([Delcredere],4), ClientAdvisor.Advisor_Instradierung, Cred
HAVING (((Left([Delcredere],4)) In ("XRBA","XRBD","XRBE")))
ORDER BY Left([Delcredere],4), ClientAdvisor.Advisor_Instradierung, Cred
```

code view

More details on SolidAA: www.soursourceit.com

Application: Build Optimization

1. Context

- major embedded software company (NASDAQ 100)
- industrial 17.5 MLOC code base of C code
- modified daily by >500 developers worldwide

2. Problem

- high build time (>9 hours)
- modifying a header causes very long recompilations
- testing becomes very hard; perfective maintenance (refactoring) nearly impossible

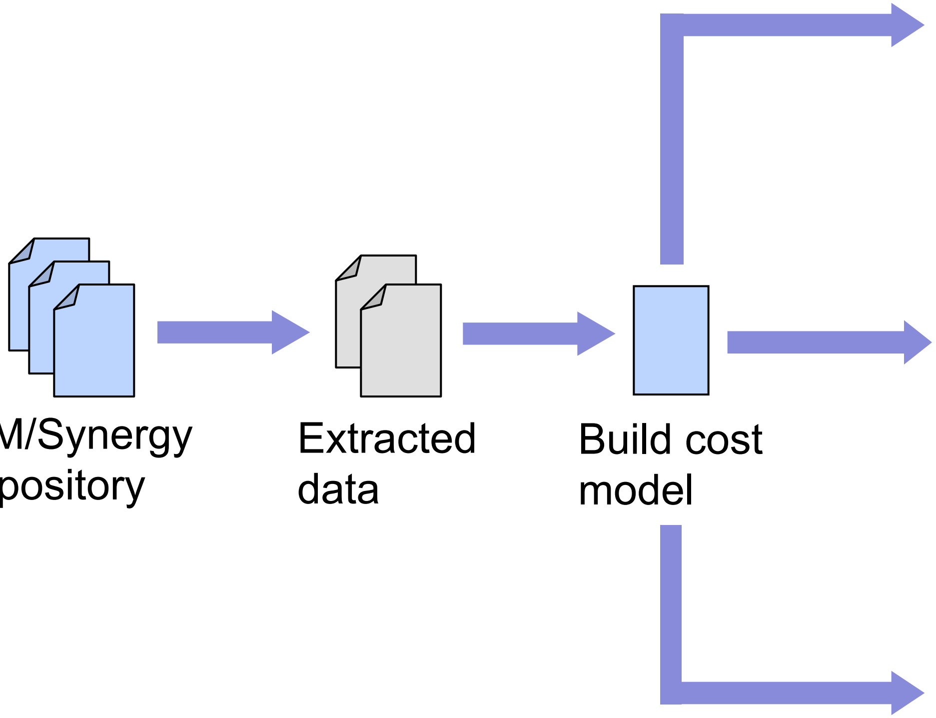
3. Questions

- why is the build time so long?
- what impact has a code change on the build time?
- how is a change impact spread over the entire code base?
- how to refactor the code to improve modularity and build time?

A. Telea, L. Voinea, Visual Software Analytics for the Build Optimization of Large-Scale Software Systems, *Comp. Statistics*, 2010

Build Optimization

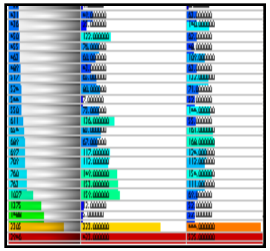
Three analyses – three tools in a unified toolset



CM/Synergy repository

Extracted data

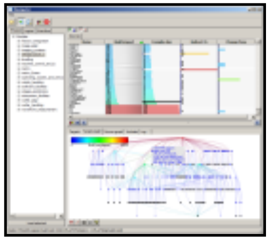
Build cost model



TableVision tool

Build process analysis

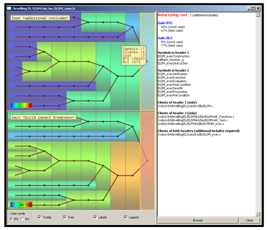
- why is the build slow?



INavigator tool

Dependency analysis

- how does a code change affect build time?



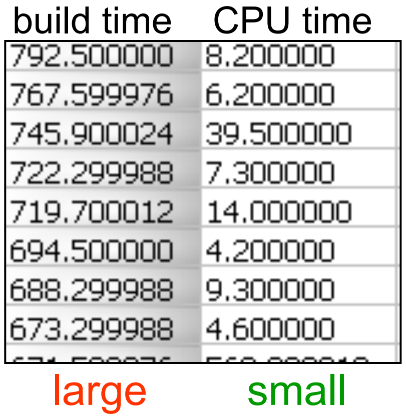
IRefactor tool

Refactoring analysis

- how to rewrite code to improve build time?

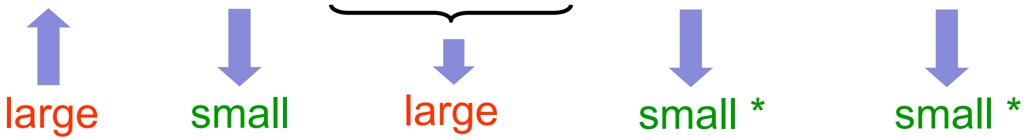
Question 1: Why is the build slow?

- measure build time using UNIX tools `time(x)`



build time	CPU time
792.500000	8.200000
767.599976	6.200000
745.900024	39.500000
722.299988	7.300000
719.700012	14.000000
694.500000	4.200000
688.299988	9.300000
673.299988	4.600000
large	small

- build time = CPU + I/O + network + paging + other processes



large

small

large

small *

small *

build time

CPU time

I/O time
+
network

build is I/O bound!

preventive actions

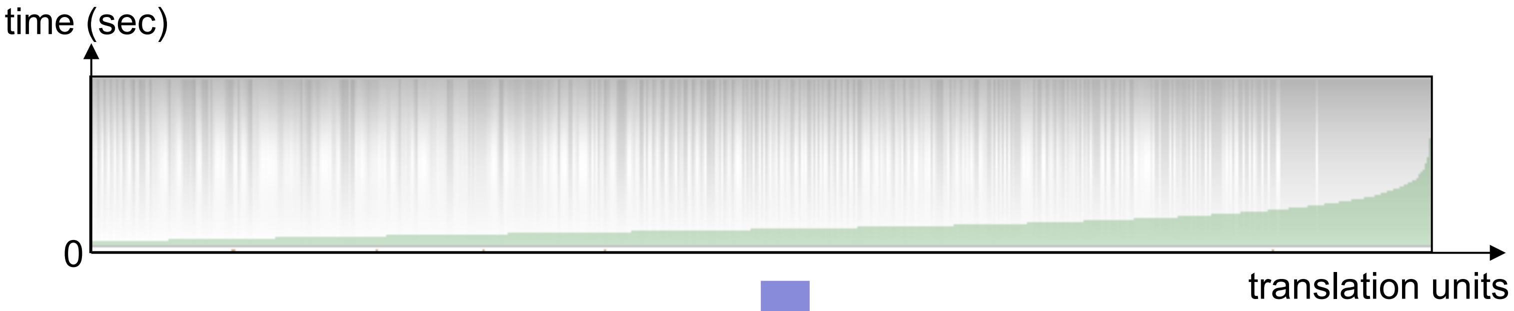
corrective actions

---

* assume no other CPU-intensive processes besides compilation

Question 1: First Steps

- simple histogram of build time



Build time depends significantly on the translation unit!

A useful build cost model must consider the per-unit build cost and not only the number of translation units

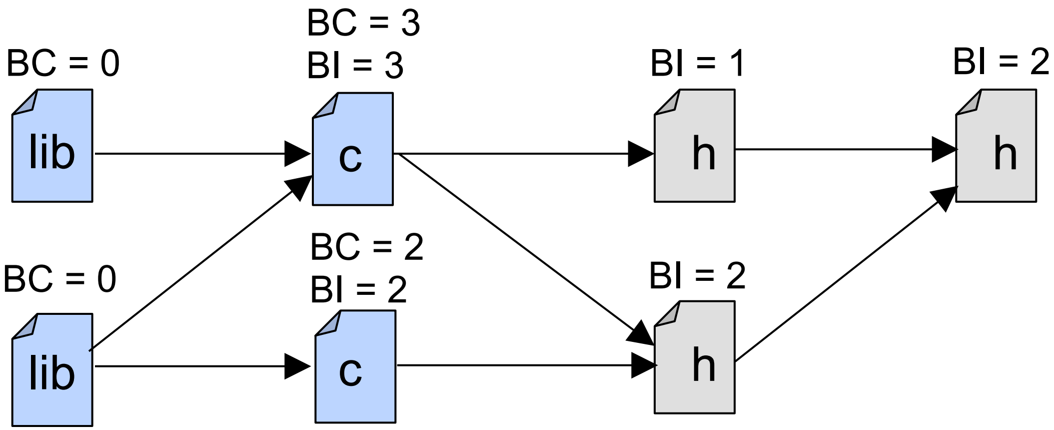
* assume no other CPU-intensive processes besides compilation

Build Cost Model - First Attempt

Build **cost**	"how much it costs to build a file"
• sources:	number of source LOC + (in)directly included headers
• binaries:	negligible (linking is cheap)
• headers:	zero (headers don't get compiled)

Build **impact**:	"how much it costs to rebuild the system when a file is modified"
• sources:	build cost of the source itself
• headers:	number of sources using that header

Example



---

* both application and system headers are considered

Build Cost Model - Validation

sorted on model’s impact

headers	model’s impact	time build measurements
AM/com/ext/inc/AM.h	3302	304316
DNDM/com/int/inc/DNDMxADMIN.h	6396	593680
DD/com/ext/inc/DDXA.h	6399	516282
IM/com/ext/inc/IM.h	6432	595957
LU/com/ext/inc/LUGB.h	6699	630971
UL/com/ext/inc/ULXAerr.h	7084	780792
HW/com/ext/inc/HW.h	7376	645346
SM/com/ext/inc/SMXA.h	9449	976110
SW/com/ext/inc/SW.h	9948	908233
LU/com/ext/inc/LU.h	17864	1682591
MI/MITF/int/inc/MIFREV.h	18380	1273473
DD/com/ext/inc/DDXA_types.h	18380	1273474
ER/com/ext/inc/ERXAerror_types.h	18380	1273473
GW/com/ext/inc/GWXA.h	18380	1273473
MO/MPPC/int/inc/MO.h	18380	1273473
ZS/com/int/inc/ZSUM_umf_internal.h	18380	1273474
DM/com/int/tst/DMTP.h	18380	1273473
MO/com/ext/inc/MOERR.h	18380	1273473
CN/com/ext/inc/CNDD.h	18380	1273473
CN/com/ext/inc/CN.h	36760	2546947
MI/MIHP/ext/inc/PCHPIF.h	36760	2546947
PL/com/ext/inc/PLXA.h	36760	2546947
MG/MGTAB/ext/inc/MGTAB.h	36760	2546947
ER/ERLO/ext/inc/ERXA.h	36760	2546948
MI/com/ext/inc/MI.h	36760	2546947
MI/MIHP/ext/inc/PCHP.h	36760	2546947
OO/com/ext/inc/OOXA.h	55140	3820421
MG/com/ext/inc/MG.h	55140	3820421
CN/com/ext/inc/CNXA.h	55140	3820421
TH/com/ext/inc/THXA.h	55140	3820421

low-impact headers

12th highest-impact header (reality) classified as 21st (model)

high-impact headers

- model is close to reality but not perfect
- deviations are important!

Build Cost Model - Refinement

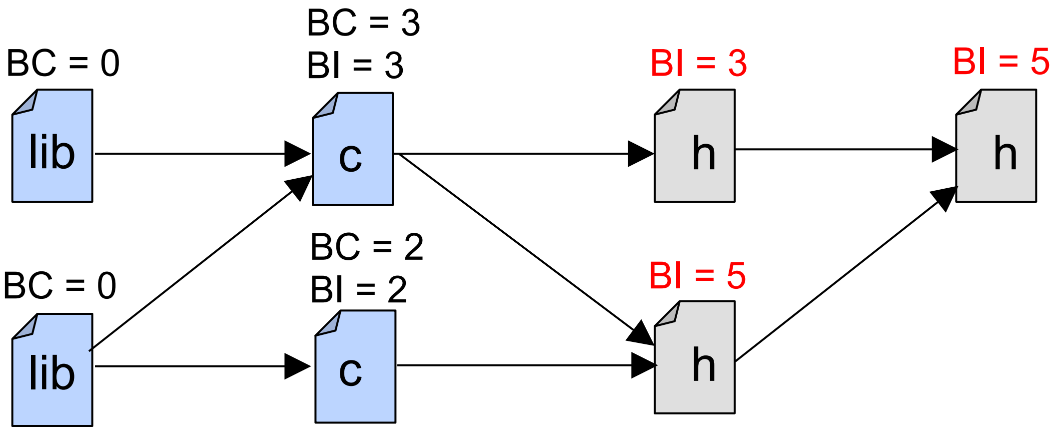
Build **cost** “how much it costs to build a file”

- sources: number of (in)directly included headers
- binaries: negligible (linking is cheap)
- headers: zero (headers don’t get compiled)

Build **impact**: “how much it costs to rebuild the system when a file is modified”

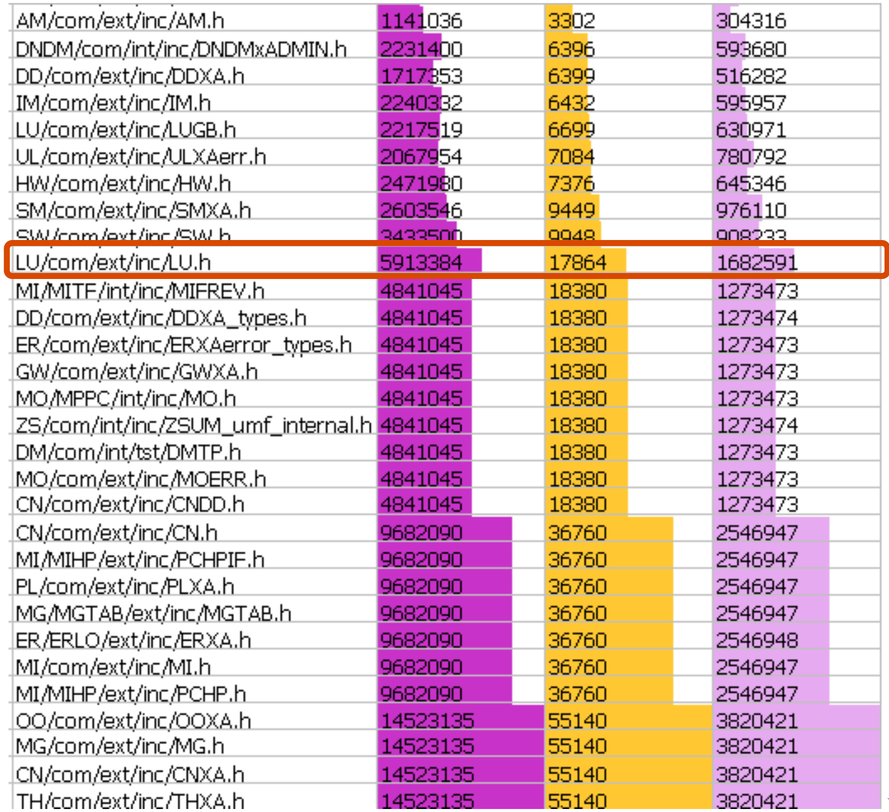
- sources: the build cost of the source itself
- headers: sum of build costs of all sources including header (in)directly

Example



Build Cost Model 2 - Validation

sorted on
first model's impact



headers	refined model's impact	first model's impact	time build measurements
AM/com/ext/inc/AM.h	1141036	3302	304316
DNDM/com/int/inc/DNDMxADMIN.h	2231400	6396	593680
DD/com/ext/inc/DDXA.h	1717353	6399	516282
IM/com/ext/inc/IM.h	2240332	6432	595957
LU/com/ext/inc/LUGB.h	2217519	6699	630971
UL/com/ext/inc/ULXAerr.h	2067954	7084	780792
HW/com/ext/inc/HW.h	2471980	7376	645346
SM/com/ext/inc/SMXA.h	2603546	9449	976110
SW/com/ext/inc/SW.h	3433500	9948	908233
LU/com/ext/inc/LU.h	5913384	17864	1682591
MI/MITF/int/inc/MIFREV.h	4841045	18380	1273473
DD/com/ext/inc/DDXA_types.h	4841045	18380	1273474
ER/com/ext/inc/ERXAerror_types.h	4841045	18380	1273473
GW/com/ext/inc/GWXA.h	4841045	18380	1273473
MO/MPPC/int/inc/MO.h	4841045	18380	1273473
ZS/com/int/inc/ZSUM_umf_internal.h	4841045	18380	1273474
DM/com/int/tst/DMTP.h	4841045	18380	1273473
MO/com/ext/inc/MOERR.h	4841045	18380	1273473
CN/com/ext/inc/CNDD.h	4841045	18380	1273473
CN/com/ext/inc/CN.h	9682090	36760	2546947
MI/MIHP/ext/inc/PCHPIF.h	9682090	36760	2546947
PL/com/ext/inc/PLXA.h	9682090	36760	2546947
MG/MGTAB/ext/inc/MGTAB.h	9682090	36760	2546947
ER/ERLO/ext/inc/ERXA.h	9682090	36760	2546948
MI/com/ext/inc/MI.h	9682090	36760	2546947
MI/MIHP/ext/inc/PCHP.h	9682090	36760	2546947
OO/com/ext/inc/OOXA.h	14523135	55140	3820421
MG/com/ext/inc/MG.h	14523135	55140	3820421
CN/com/ext/inc/CNXA.h	14523135	55140	3820421
TH/com/ext/inc/THXA.h	14523135	55140	3820421

low-impact headers

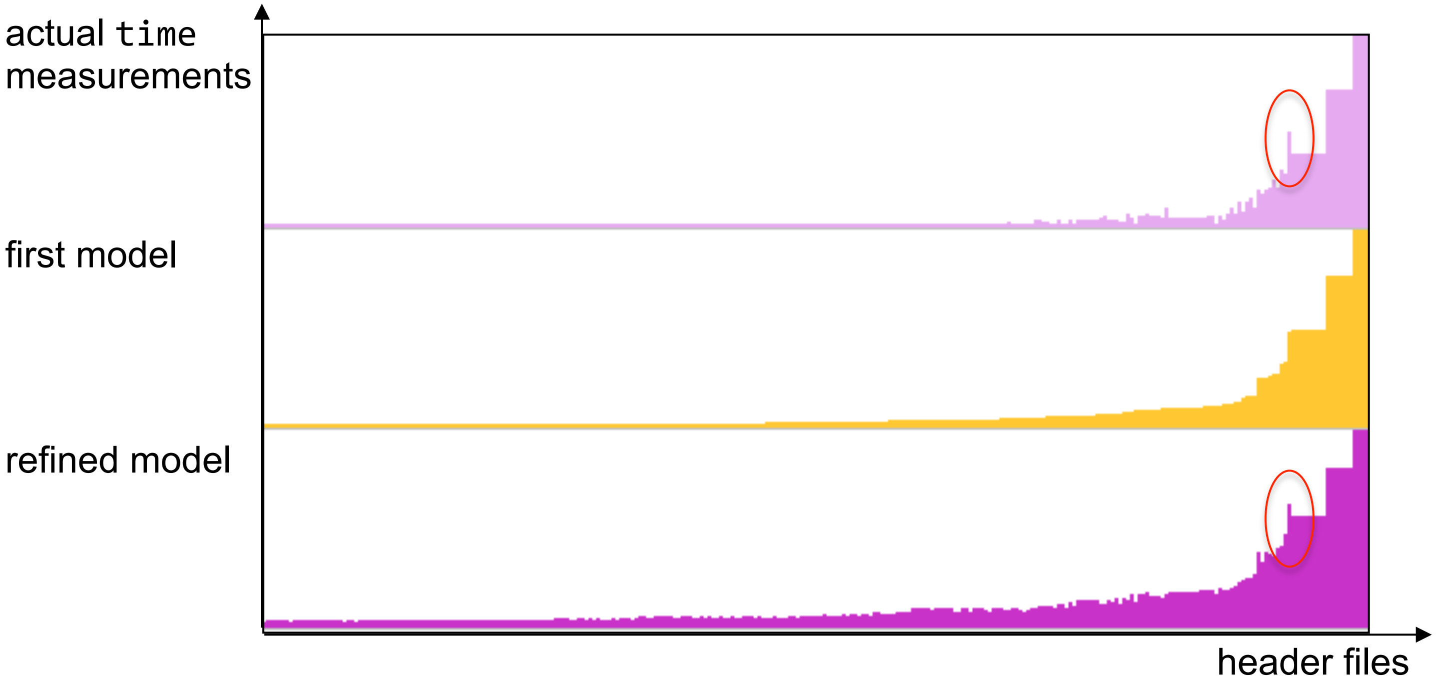
refined model classifies outlier correctly

high-impact headers

- refined model delivers same header-order (in terms of impact) as actual measurements

Build Cost Model 2 - Validation

Let' s look at the whole picture

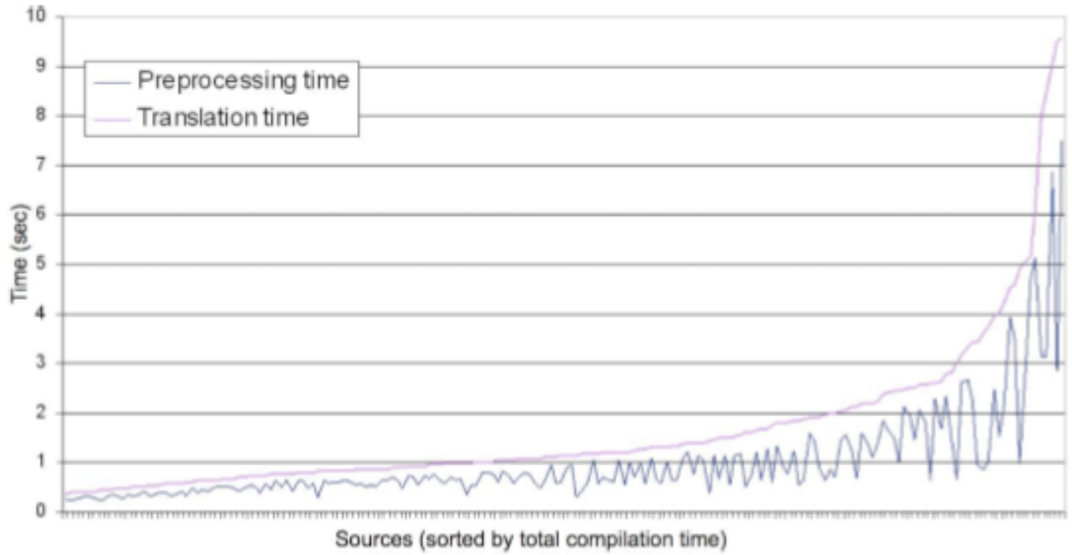


- refined model nicely matches reality, including subtle 'outliers'
- why is this so? (see next slide)

Build Cost Model 2 - Validation

Analyze deeper:

- compilation cost dominated by I/O (preprocessing headers)
- I/O cost dominated by file opening/closing on this platform
- hence the justification of impact = # totally opened headers



Conclusions

To reduce build time, we should:

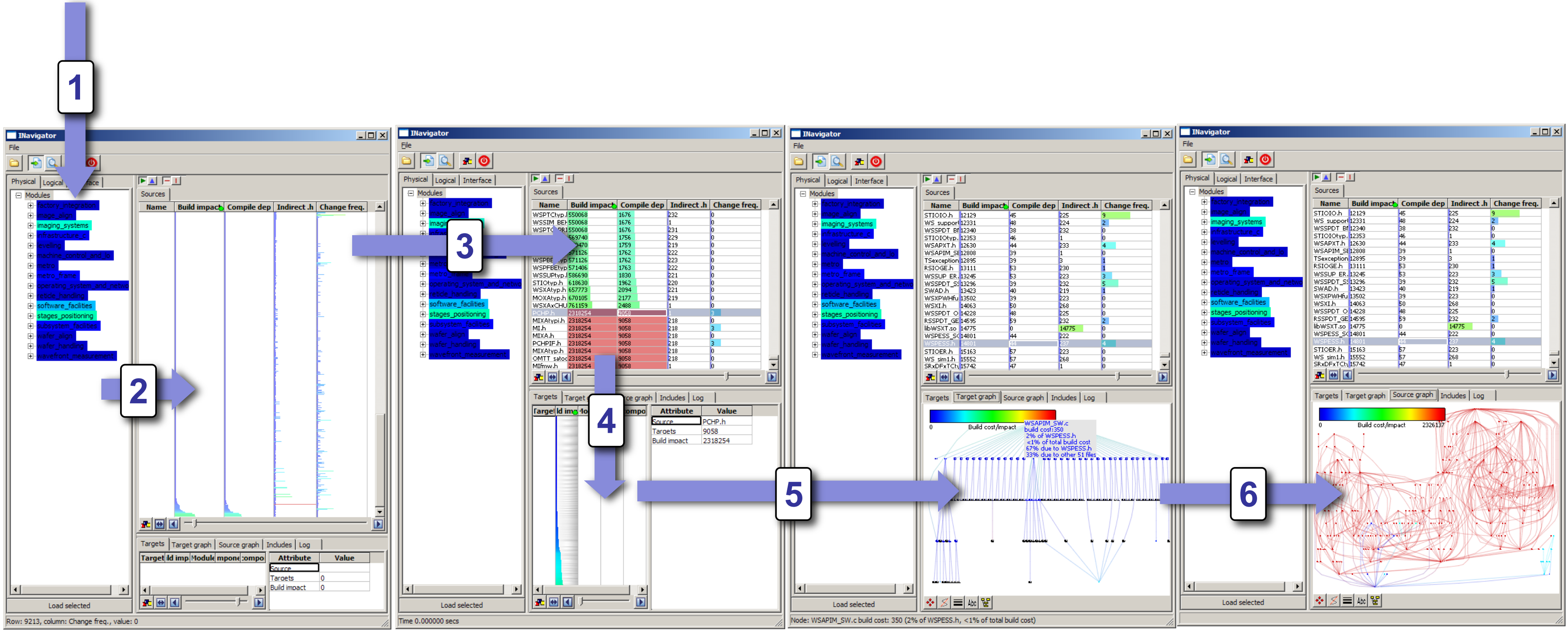
- either massively accelerate network → highly costly / complex ✗
- reduce per-header build impact → header impact analysis ✓
- reduce impact of change on build time → header refactoring ✓

Question 2: System-wide impact analysis

1. Find subsystems are expensive to build
2. For a subsystem, find headers have high build impact
3. Zoom in to highest impact headers

4,5. For a high-impact header, see how its impact spreads over sources

6. For a header, see its cost breakdown over its include-set



Row: 9213, column: Change freq., value: 0

Time 0.000000 secs

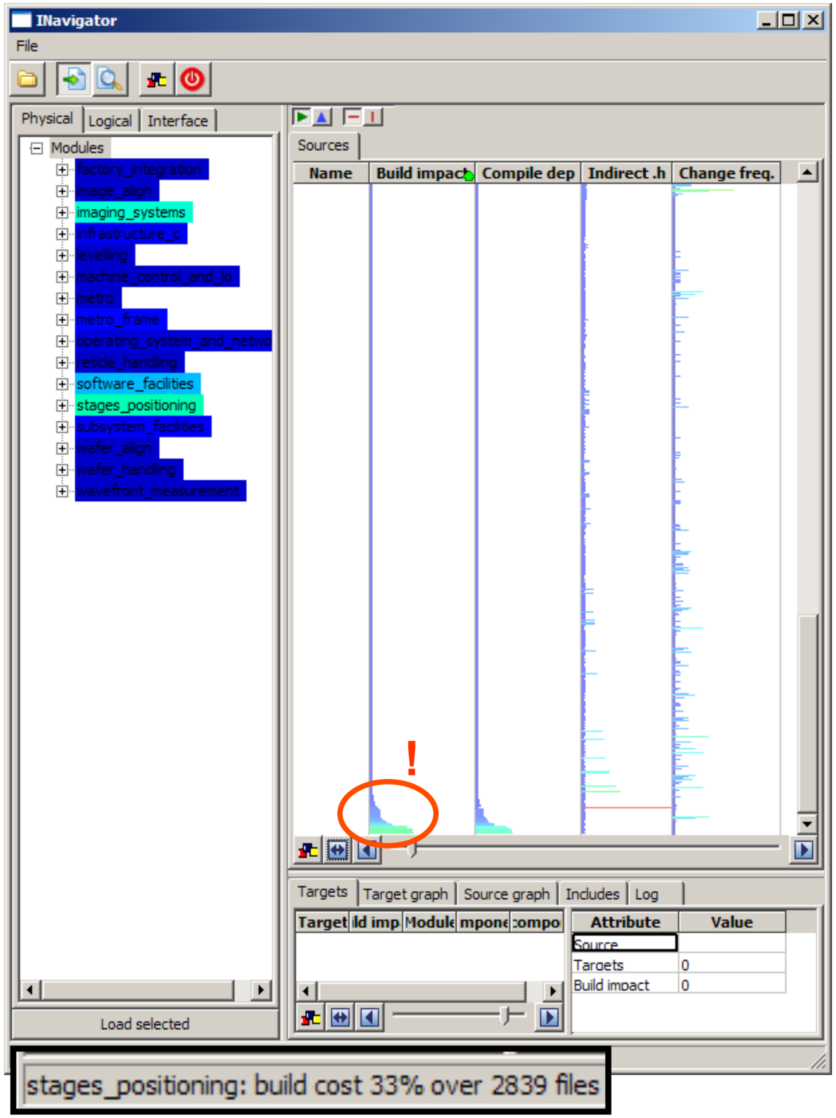
Node: WSAPIM_SW.c build cost: 350 (2% of WSPESS.h, <1% of total build cost)

More details on SolidBA: www.soursourceit.com

www.cs.rug.nl/svcg

university of groningen

Subsystem-level impact analysis



Attribute	Value
Source	
Targets	0
Build impact	0

stages_positioning: build cost 33% over 2839 files

Method

- color system tree by cost (blue=low, red=high)
- select desired subsystem
- right panel shows build impact for each header / source in that subsystem

Findings

- most headers have a low build impact
- however, a few have a very high impact
- touching those incurs a high build cost!

➡ because they are used in many sources
➡ because they include many headers

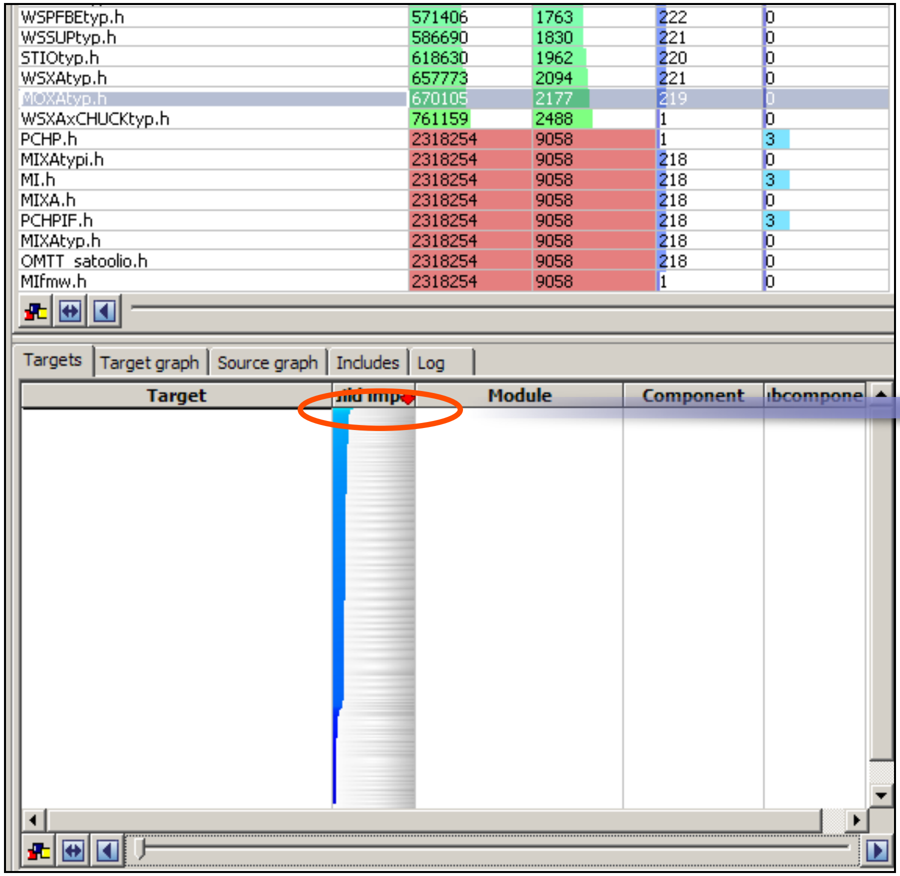
More details on SolidBA: www.soursourceit.com

Question 3: How to reduce the build cost?

- OK, we have a high-impact header h: how easy it to reduce that impact?
- visualize the build cost distribution of h over the sources which use it

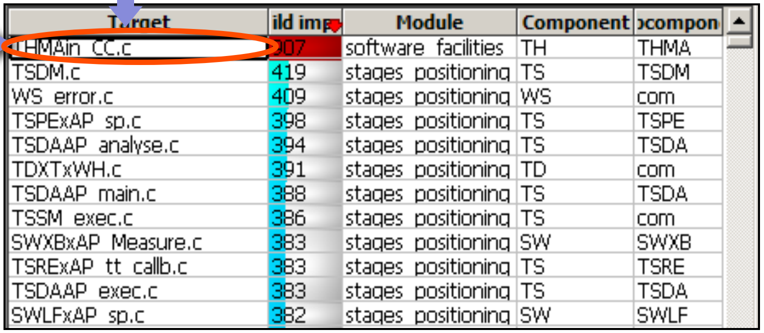
Case 1: easy refactoring

- build cost spread unevenly over the targets including selected header h
- to decrease cost due to h, we only need to change a few targets



WSPFBEtyp.h	571406	1763	222	0
WSSUPtyp.h	586690	1830	221	0
STIOtyp.h	618630	1962	220	0
WSXAtyp.h	657773	2094	221	0
MOXAtyp.h	670105	2177	219	0
WSXAxCHUCKtyp.h	761159	2488	1	0
PCHP.h	2318254	9058	1	3
MIXAtypi.h	2318254	9058	218	0
MI.h	2318254	9058	218	3
MIXA.h	2318254	9058	218	0
PCHPIF.h	2318254	9058	218	3
MIXAtyp.h	2318254	9058	218	0
OMTT satoolio.h	2318254	9058	218	0
MIfmw.h	2318254	9058	1	0

build impact of h is located mainly in one single place!

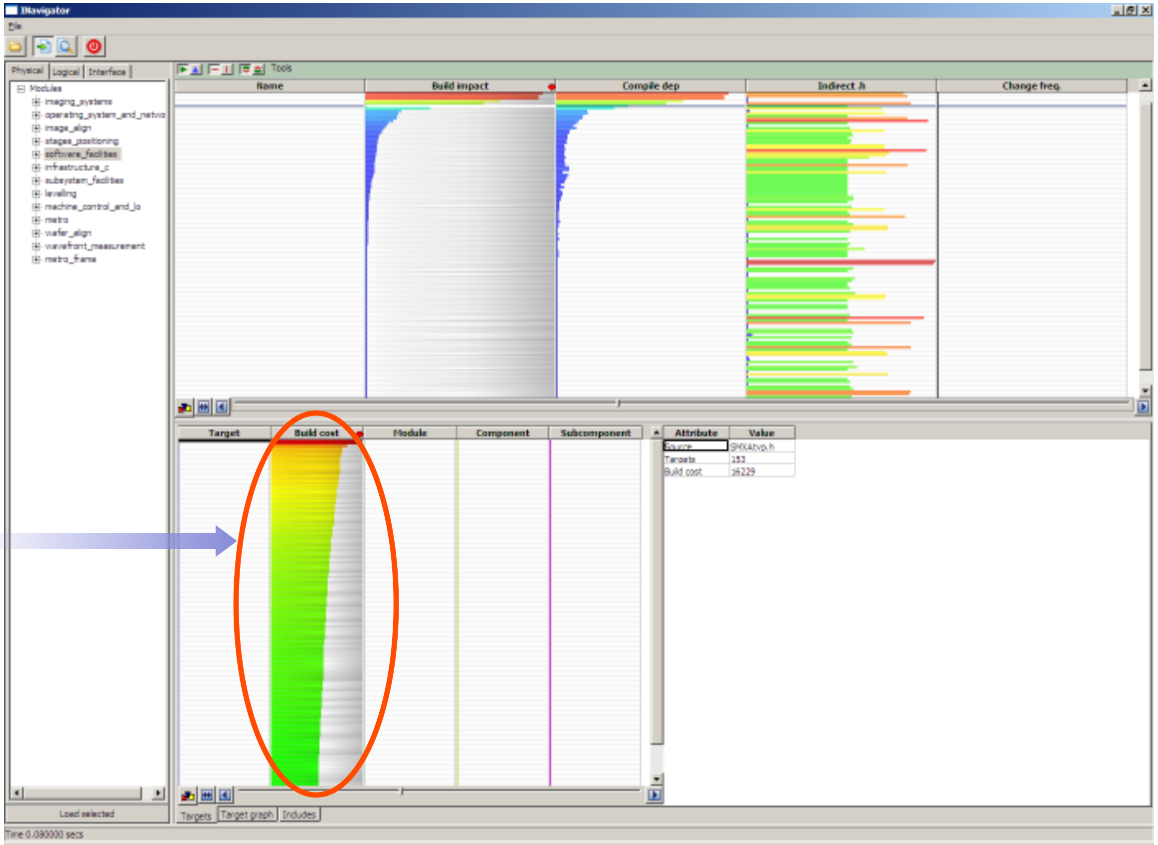


Target	ild imp	Module	Component	bcompon
THMAin CC.c	907	software facilities	TH	THMA
TSDM.c	419	stages positioning	TS	TSDM
WS error.c	409	stages positioning	WS	com
TSPExAP sp.c	398	stages positioning	TS	TSPE
TSDAAP analyse.c	394	stages positioning	TS	TSDA
TDXTxWH.c	391	stages positioning	TD	com
TSDAAP main.c	388	stages positioning	TS	TSDA
TSSM exec.c	386	stages positioning	TS	com
SWXBxAP Measure.c	383	stages positioning	SW	SWXB
TSRExAP tt callb.c	383	stages positioning	TS	TSRE
TSDAAP exec.c	383	stages positioning	TS	TSDA
SWLFxAP sp.c	382	stages positioning	SW	SWLF

Refactoring analysis

Case 1: difficult refactoring

- build cost spread evenly over the targets including selected header h
- to decrease cost due to h, we need to change almost all targets



Attribute	Value
Source	SMXAtvp.h
Targets	153
Build cost	16229

selected high-impact header

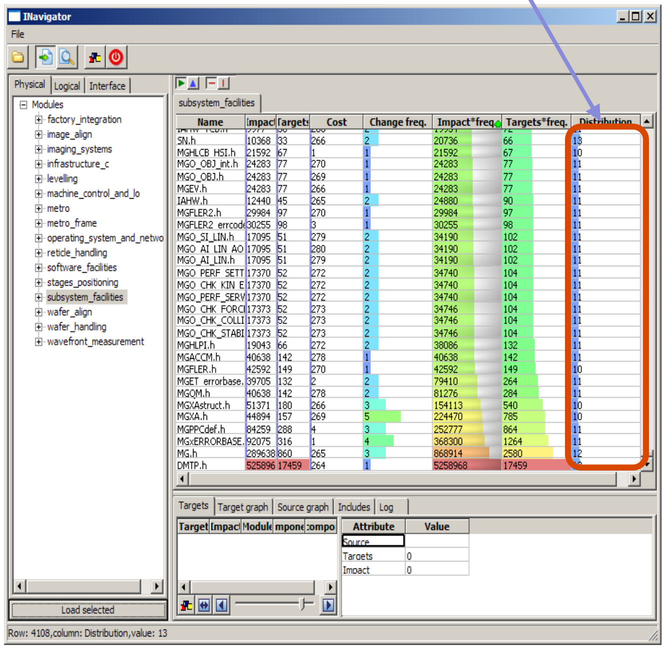
build impact of h

Refactoring analysis - Refinement

- not all headers change equally often (e.g. system headers)
- new metrics:
  - build impact * change frequency
  - impact distribution: impact (%) of a header contained in the 10% most expensive of its targets
- easy & quick to use

flat, low distribution (~15%): impact is spread uniformly over all targets. Hence, we cannot improve by refactoring a *few* targets

skewed distribution (50%): half of impact is concentrated in 10% most expensive targets. Hence, refactoring these is an interesting option



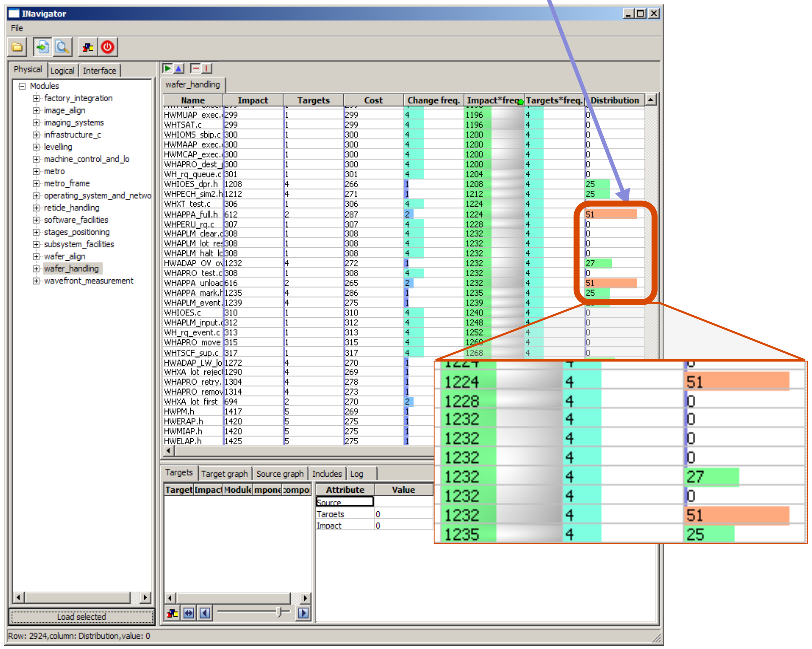
Name	Impact	Targets	Cost	Change freq.	Impact*freq.	Targets*freq.	Distribution
SN.h	10368	33	266	2	20736	66	13
MGHLCB HSI.h	21592	67	1	1	21592	67	10
MGO_OBJ_int.h	24283	77	270	1	24283	77	11
MGO_OBJ.h	24283	77	269	1	24283	77	11
MGEV.h	24283	77	266	1	24283	77	11
IAHW.h	12440	45	265	2	24880	90	11
MGFLER2.h	29984	97	270	1	29984	97	11
MGFLER2 errcode	30255	98	3	1	30255	98	11
MGO_SI_LIN.h	17095	51	279	2	34190	102	11
MGO AI LIN AO	17095	51	280	2	34190	102	11
MGO_AI_LIN.h	17095	51	279	2	34190	102	11
MGO PERF SETT	17370	52	272	2	34740	104	11
MGO CHK KIN E	17370	52	272	2	34740	104	11
MGO_PERF_SERV	17370	52	272	2	34740	104	11
MGO CHK FORCI	17373	52	273	2	34746	104	11
MGO_CHK_COLLI	17373	52	273	2	34746	104	11
MGO_CHK_STABI	17373	52	273	2	34746	104	11
MGHLPI.h	19043	66	272	2	38086	132	11
MGACCM.h	40638	142	278	1	40638	142	11
MGFLER.h	42592	149	270	1	42592	149	10
MGET errorbase.	39705	132	2	2	79410	264	11
MGQM.h	40638	142	278	2	81276	284	11
MGXAstruct.h	51371	180	266	3	154113	540	10
MGXA.h	44894	157	269	5	224470	785	10
MGPPCdef.h	84259	288	4	3	252777	864	11
MGxERRORBASE.	92075	316	1	4	368300	1264	11
MG.h	289638	860	265	3	868914	2580	12
DMTP.h	525896	17459	264	1	5258968	17459	

Target	Impact	Module	mpone	compo

Attribute	Value
Source	
Targets	0
Impact	0

Row: 4108,column: Distribution,value: 13

sorted by impact*change



Name	Impact	Targets	Cost	Change freq.	Impact*freq.	Targets*freq.	Distribution
HWMUAP exec.	299	1	299	4	1196	4	0
WHTSAT.c	299	1	299	4	1196	4	0
WHIOMS sbip.c	300	1	300	4	1200	4	0
HWMAAP exec.	300	1	300	4	1200	4	0
HWMCAP_exec.	300	1	300	4	1200	4	0
WHAPRO_dest_	300	1	300	4	1200	4	0
WH_rq_queue.c	301	1	301	4	1204	4	0
WHIOES_dpr.h	1208	4	266	1	1208	4	25
WHPECH_sim2.h	1212	4	271	1	1212	4	25
WHXT test.c	306	1	306	4	1224	4	
WHAPPA_full.h	612	2	287	2	1224	4	51
WHPERU_rq.c	307	1	307	4	1228	4	0
WHAPLM clear.c	308	1	308	4	1232	4	0
WHAPLM lot res	308	1	308	4	1232	4	0
WHAPLM halt lc	308	1	308	4	1232	4	0
HWADAP OV ov	1232	4	272	1	1232	4	27
WHAPRO test.c	308	1	308	4	1232	4	0
WHAPPA unload	616	2	265	2	1232	4	51
WHAPPA mark.h	1235	4	286	1	1235	4	25
WHAPLM_event.	1239	4	275	1	1239	4	
WHIOES.c	310	1	310	4	1240	4	0
WHAPLM_input.c	312	1	312	4	1248	4	0
WH_rq_event.c	313	1	313	4	1252	4	0
WHAPRO move	315	1	315	4	1260	4	0
WHTSCF_sup.c	317	1	317	4	1268	4	0
HWADAP_LW_lo	1272	4	270	1			
WHXA lot reject	1290	4	269	1			
WHAPRO retry.	1304	4	278	1			
WHAPRO remov	1314	4	273	1			
WHXA lot first	694	2	270	2			
HWPM.h	1417	5	269	1			
HWERAP.h	1420	5	275	1			
HWMIAP.h	1420	5	275	1			
HWELAP.h	1425	5	275	1			

Impact*freq.	Targets*freq.	Distribution
1224	4	51
1228	4	0
1232	4	0
1232	4	0
1232	4	0
1232	4	27
1232	4	0
1232	4	51
1235	4	25

Target	Impact	Module	mpone	compo

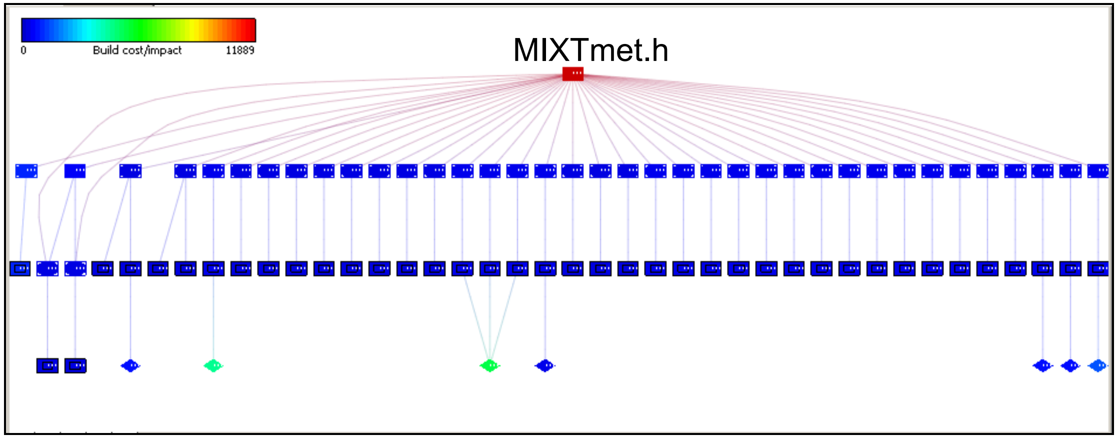
Attribute	Value
Source	
Targets	0
Impact	0

Row: 2924,column: Distribution,value: 0

Refactoring support

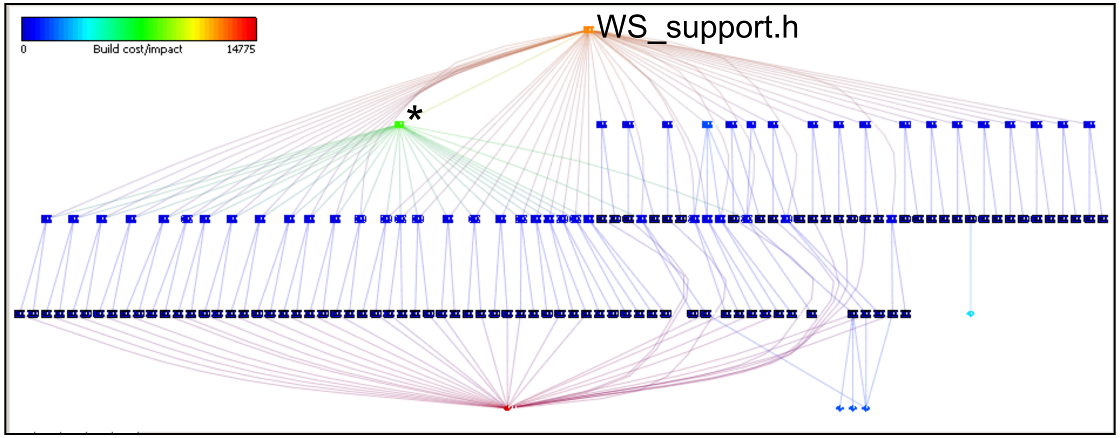
- OK, we found a high-impact header; how to decide a refactoring plan?
- show dependencies header → clients using hierarchical DAG layout

Example 1: MIXTmet.h, used by 38 sources, high impact



- build impact due to direct header inclusion
- hard to decrease via refactoring

Example 2: WS_support.h, used by 48 sources, high impact



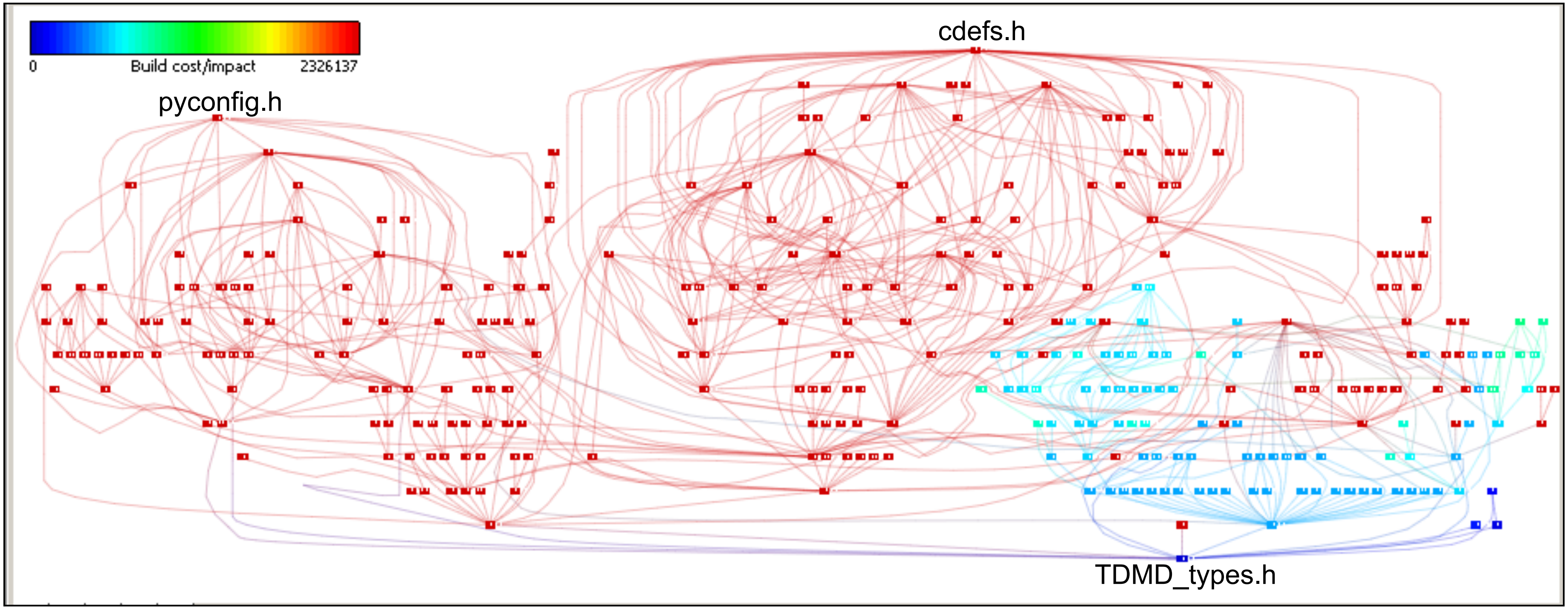
- build impact channeled via one intermediate header: WS_sim1_support.h (*)
- simpler refactoring may be possible

Refactoring support (2)

- say we want to include a header: is this potentially expensive?
- show header's own include graph colored by build impact

Example: TDMD_types.h, used by 30 sources

- not a high-impact header itself
- but it includes high-impact headers!
- hence using this header introduces potentially expensive changes



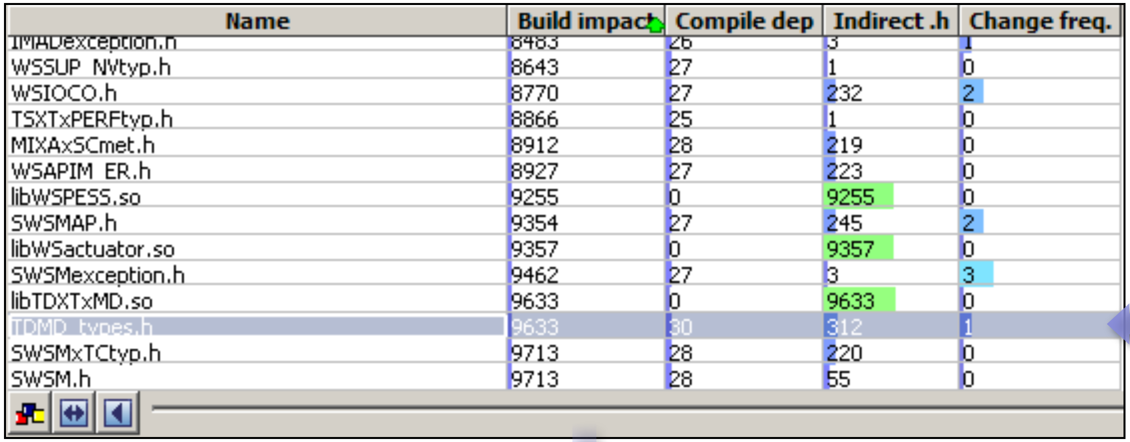
More details on SolidBA: www.soursourceit.com

Refactoring support (3)

How much costlier becomes the system build if we add an #include?

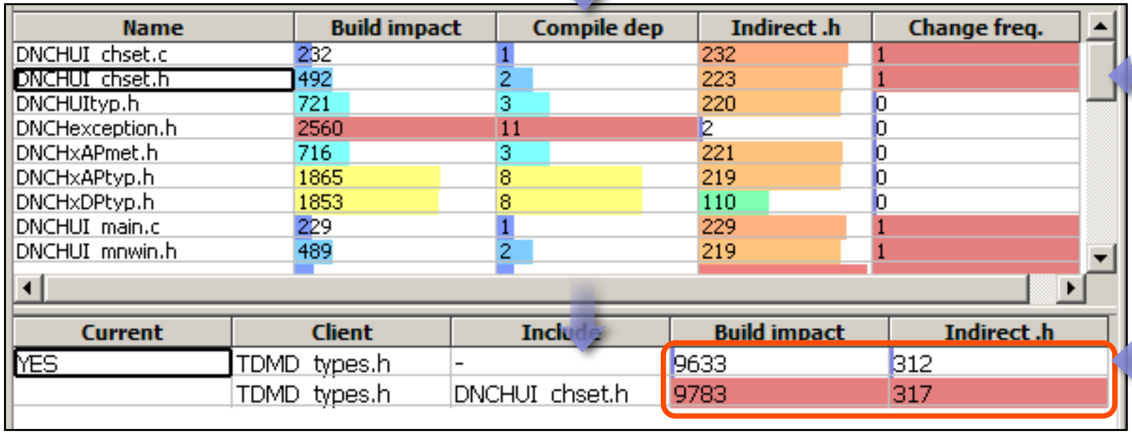
- select a "source" header – the one in which we want to #include
- select a "destination" header – the one to be #included
- show the build cost increase

Example: What if we #include DNCHUI_chset.h in TDMD_types.h?



Name	Build impact	Compile dep	Indirect .h	Change freq.
IMADexception.h	8483	26	3	1
WSSUP NVtyp.h	8643	27	1	0
WSIOCO.h	8770	27	232	2
TSXTxPERFtyp.h	8866	25	1	0
MIXAxSCmet.h	8912	28	219	0
WSAPIM ER.h	8927	27	223	0
libWSPESS.so	9255	0	9255	0
SWSMAP.h	9354	27	245	2
libWSactuator.so	9357	0	9357	0
SWSMexception.h	9462	27	3	3
libTDXTxMD.so	9633	0	9633	0
TDMD types.h	9633	30	312	1
SWSMxTCtyp.h	9713	28	220	0
SWSM.h	9713	28	55	0

source



Name	Build impact	Compile dep	Indirect .h	Change freq.
DNCHUI chset.c	232	1	232	1
DNCHUI chset.h	492	2	223	1
DNCHUItyp.h	721	3	220	0
DNCHexception.h	2560	11	2	0
DNCHxAPmet.h	716	3	221	0
DNCHxAPtyp.h	1865	8	219	0
DNCHxDPtyp.h	1853	8	110	0
DNCHUI main.c	229	1	229	1
DNCHUI mnwin.h	489	2	219	1

Current	Client	Include	Build impact	Indirect .h
YES	TDMD types.h	-	9633	312
	TDMD types.h	DNCHUI chset.h	9783	317

target

build impact increases from 9633 to 9783, i.e. **1.5%**

Refactoring support (4)

- previous methods OK for manual header-by-header refactoring only

How to refactor a large system?

- system $S = \{f_i\}_i$, $S = Headers \cup Sources$
- header $h_i \in Headers = \{s_j\}_j$, $s_j \in Symbols$ (function declarations, variables, types, macros, …)
- include relations

  $inc : S \rightarrow \mathcal{P}(Headers), inc(f) = \{h_i\} \leftrightarrow f$ includes h_i
- symbol use relations

  $use : Symbols \rightarrow \mathcal{P}(Headers), use(s) = \{h_i\} \leftrightarrow s$ is used by h_i
  - in typical systems, not all symbols $s_j \in h$ in a header are used together

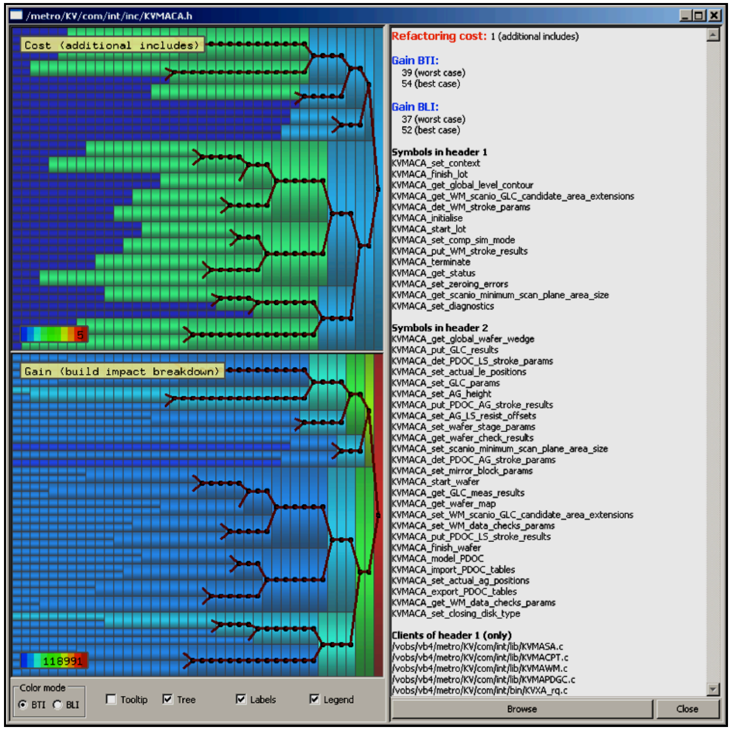
Automatic refactoring idea

- find high-impact header h (see last slides)
- split h into h_1 , h_2 ; $h_1 \cup h_2 = h$ by putting symbols used together in same h_i
- recursively split h_1 , h_2
- replace $inc(h)$ by $inc(h_1)$ and/or $inc(h_2)$

Refactoring support (4)

- intuitively: put symbols **often used together** in same header
- include newly created headers instead of original ‘monolithic’ ones
- why is this good
  - decrease build costs (by decreasing the included code size)
  - decrease build impact (by decreasing the number of included headers)

The IRefactor analysis tool



Color mode ● BTI ○ BLI

☐ Tooltip ☑ Tree ☑ Labels ☑ Legend

Refactoring cost: 1 (additional includes)

Gain BTI:
39 (worst case)
54 (best case)

Gain BLI:
37 (worst case)
52 (best case)

Symbols in header 1
KVMACA_set_context
KVMACA_finish_lot
KVMACA_get_global_level_contour
KVMACA_get_WM_scanio_GLC_candidate_area_extensions
KVMACA_det_WM_stroke_params
KVMACA_initialise
KVMACA_start_lot
KVMACA_set_comp_sim_mode
KVMACA_put_WM_stroke_results
KVMACA_terminate
KVMACA_get_status
KVMACA_set_zeroing_errors
KVMACA_get_scanio_minimum_scan_plane_area_size
KVMACA_set_diagnostics

Symbols in header 2
KVMACA_get_global_wafer_wedge
KVMACA_put_GLC_results
KVMACA_det_PDOC_LS_stroke_params
KVMACA_set_actual_le_positions
KVMACA_set_GLC_params
KVMACA_set_AG_height
KVMACA_put_PDOC_AG_stroke_results
KVMACA_set_AG_LS_resist_offsets
KVMACA_set_wafer_stage_params
KVMACA_get_wafer_check_results
KVMACA_set_scanio_minimum_scan_plane_area_size
KVMACA_det_PDOC_AG_stroke_params
KVMACA_set_mirror_block_params
KVMACA_start_wafer
KVMACA_get_GLC_meas_results
KVMACA_get_wafer_map
KVMACA_set_WM_scanio_GLC_candidate_area_extensions
KVMACA_set_WM_data_checks_params
KVMACA_put_PDOC_LS_stroke_results
KVMACA_finish_wafer
KVMACA_model_PDOC
KVMACA_import_PDOC_tables
KVMACA_set_actual_ag_positions
KVMACA_export_PDOC_tables
KVMACA_get_WM_data_checks_params
KVMACA_set_closing_disk_type

Clients of header 1 (only)
/vobs/vb4/metro/KV/com/int/lib/KVMASA.c
/vobs/vb4/metro/KV/com/int/lib/KVMACPT.c
/vobs/vb4/metro/KV/com/int/lib/KVMAWM.c
/vobs/vb4/metro/KV/com/int/lib/KVMAPDGC.c
/vobs/vb4/metro/KV/com/int/bin/KVXA_rq.c

- suggests refactoring possibilities and shows gained build impact

Refactoring visualization

0 🟦🟦🟩🟨🟧 ≥5

Refactoring cost

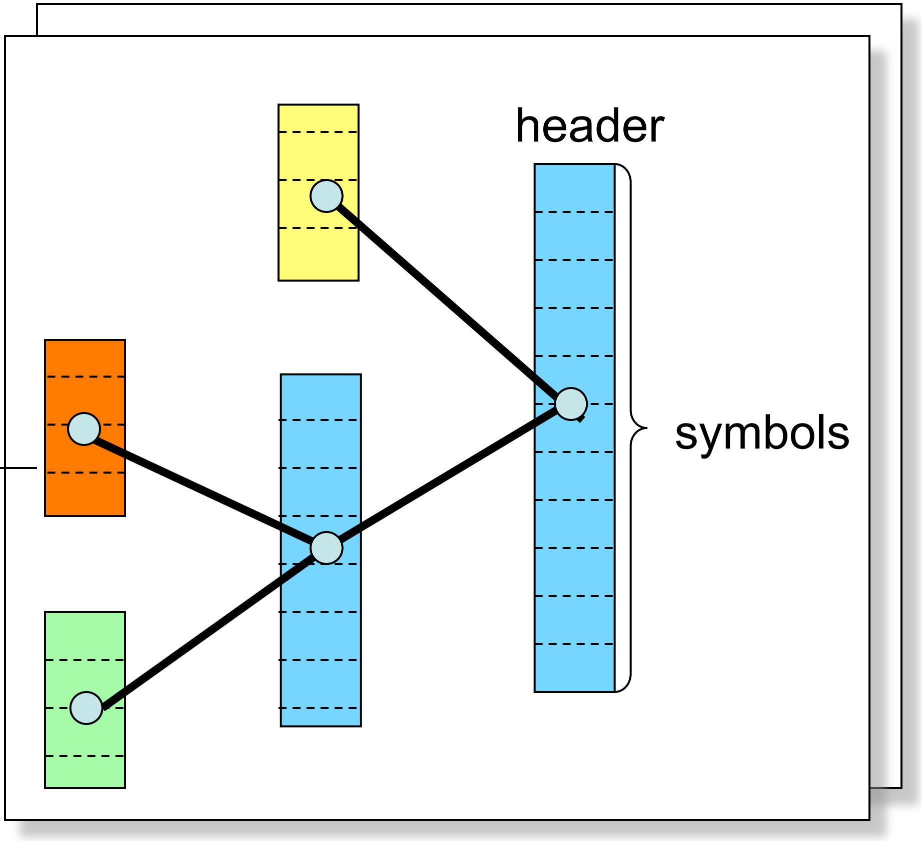
(how many files must include both headers after refactoring)

2 colors

Build impact

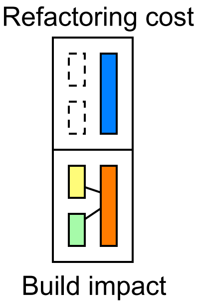
(how much we saved In build time)

min 🟦🟦🟩🟨🟧 max



Best refactoring candidates:

- low refactoring cost
- high build impact parents
- low build impact children



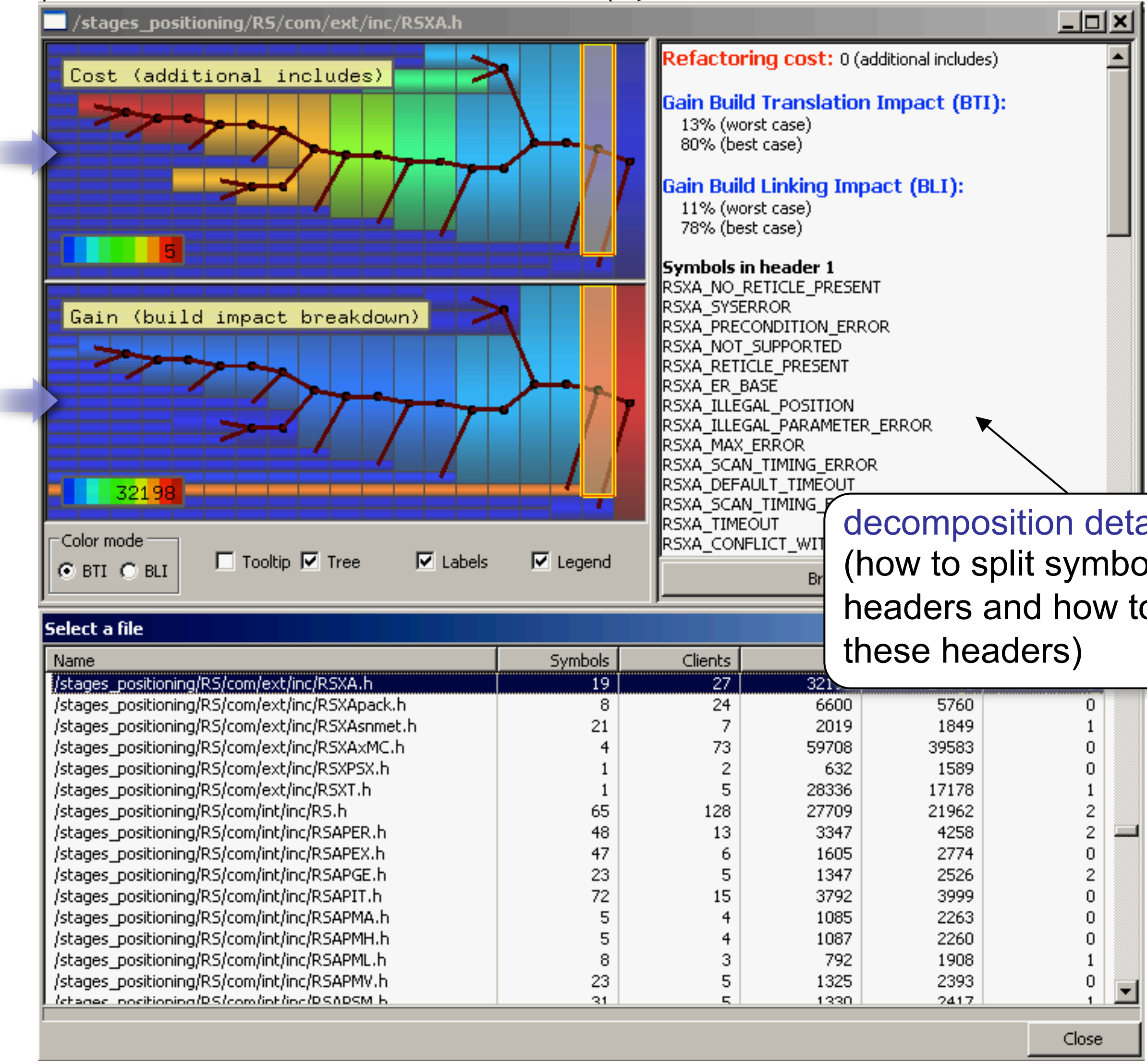
Refactoring visualization

suggested decomposition levels

header under analysis

Color: **refactoring cost**
(how many additional headers)

Color: **refactoring benefit**
(% build impact reduction)



Refactoring cost: 0 (additional includes)

Gain Build Translation Impact (BTI):
13% (worst case)
80% (best case)

Gain Build Linking Impact (BLI):
11% (worst case)
78% (best case)

Symbols in header 1
RSXA_NO_RETICLE_PRESENT
RSXA_SYSERROR
RSXA_PRECONDITION_ERROR
RSXA_NOT_SUPPORTED
RSXA_RETICLE_PRESENT
RSXA_ER_BASE
RSXA_ILLEGAL_POSITION
RSXA_ILLEGAL_PARAMETER_ERROR
RSXA_MAX_ERROR
RSXA_SCAN_TIMING_ERROR
RSXA_DEFAULT_TIMEOUT
RSXA_SCAN_TIMING_
RSXA_TIMEOUT
RSXA_CONFLICT_WIT

Color mode: ● BTI ○ BLI

☐ Tooltip ☑ Tree ☑ Labels ☑ Legend

Select a file

Name	Symbols	Clients			
/stages_positioning/RS/com/ext/inc/RSXA.h	19	27	321		
/stages_positioning/RS/com/ext/inc/RSXApack.h	8	24	6600	5760	0
/stages_positioning/RS/com/ext/inc/RSXAsnmet.h	21	7	2019	1849	1
/stages_positioning/RS/com/ext/inc/RSXAxMC.h	4	73	59708	39583	0
/stages_positioning/RS/com/ext/inc/RSXPSX.h	1	2	632	1589	0
/stages_positioning/RS/com/ext/inc/RSXT.h	1	5	28336	17178	1
/stages_positioning/RS/com/int/inc/RS.h	65	128	27709	21962	2
/stages_positioning/RS/com/int/inc/RSAPER.h	48	13	3347	4258	2
/stages_positioning/RS/com/int/inc/RSAPEX.h	47	6	1605	2774	0
/stages_positioning/RS/com/int/inc/RSAPGE.h	23	5	1347	2526	2
/stages_positioning/RS/com/int/inc/RSAPIT.h	72	15	3792	3999	0
/stages_positioning/RS/com/int/inc/RSAPMA.h	5	4	1085	2263	0
/stages_positioning/RS/com/int/inc/RSAPMH.h	5	4	1087	2260	0
/stages_positioning/RS/com/int/inc/RSAPML.h	8	3	792	1908	1
/stages_positioning/RS/com/int/inc/RSAPMV.h	23	5	1325	2393	0
/stages_positioning/RS/com/int/inc/RSAPSM.h	31	5	1330	2417	1

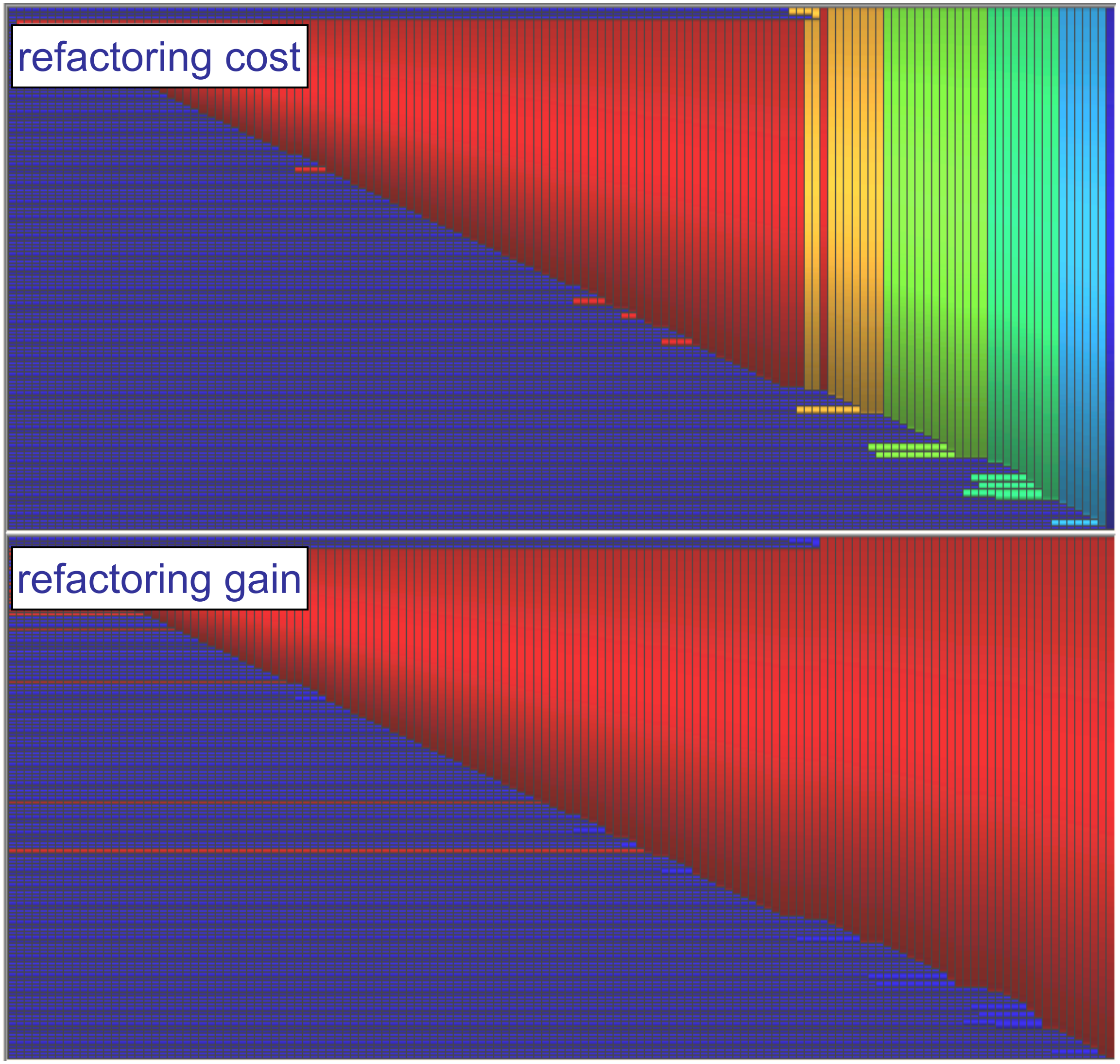
decomposition details:
(how to split symbols in smaller headers and how to #include these headers)

More details on SolidBA: www.soursourceit.com

Refactoring visualization

Example of bad candidate for header refactoring

As we gain benefits, we also increase costs



Collaborations & Customers

The Industry…

PHILIPS
sense and simplicity

ASML

Continental

PERVASIVE

NXP

making a success of life
momentum

NOKIA
Connecting People

LELY
innovators in agriculture

SBO
SOFTWARE BENCHMARKING

SolidSource IT

Research/Academia…

TU/e Technische Universiteit Eindhoven University of Technology

USP Universidade de São Paulo
BRASIL

UNICAMP
UNIVERSIDADE ESTADUAL DE CAMPINAS

IBM IBM Research

dgac DSNA
DTI Direction de la Technique et de l'Innovation

Laboratoire Bordelais de Recherche en Informatique

…Open Source

OpenCV

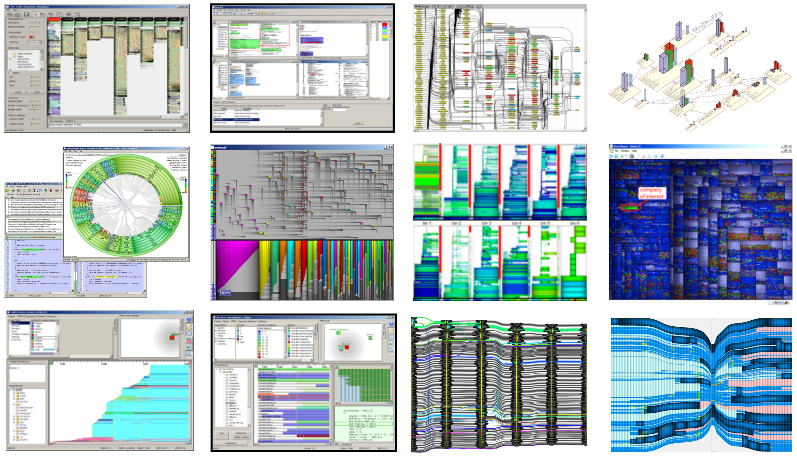
KOFFICE

mozilla

Conclusions - Software Visual Analytics

- **Provide insight in multidimensional correlations**
  - Program architecture, dependencies, metrics, development/testing effort, requirements, documentation, databases
  - Evolution of all these aspects in time

- **Added value**
  - Make the entire chain requirements..design..code visible and accountable
  - Assess software quality
  - Pinpoint hot-spots (where to invest effort)
  - Make sense of all that ‘big data’



Thank you for your interest!

Alex Telea
a.c.telea@rug.nl