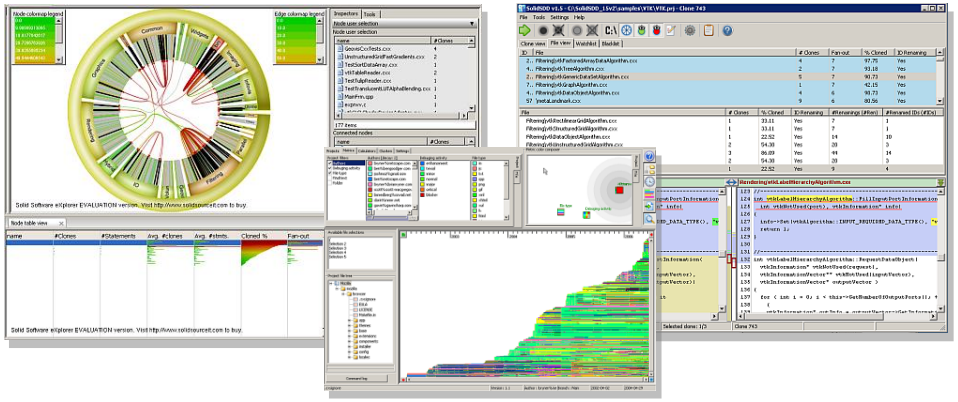
A Visual Analytics Toolset for Program Structure, Metrics, and Evolution Comprehension



Dennie Reniers, Lucian Voinea

SolidSource BV, Eindhoven
the Netherlands

Ozan Ersoy, Alexandru Telea

University of Groningen
the Netherlands

Software Visual Analytics

- integrates data mining, analysis, and interactive visualization for sense-making from large software systems

- data: structure, dependencies, metric, behavior, evolution
- tools: static analysis, fact extraction, repository mining
  graph, table, matrix, timeline visualizations
- tasks: sensemaking by iterative hypothesis creation, refinement, (in)validation

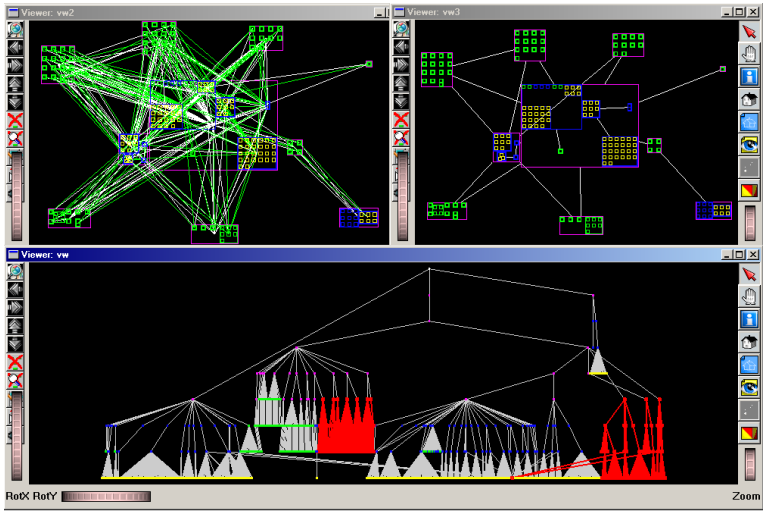
Visual analytics

✓ proven added value in many contexts (including software)

✗ hard to develop efficient and effective tools

- wide range of technologies
- scalability, usability, robustness, integration issues
- visualization is still not widely accepted in software engineering

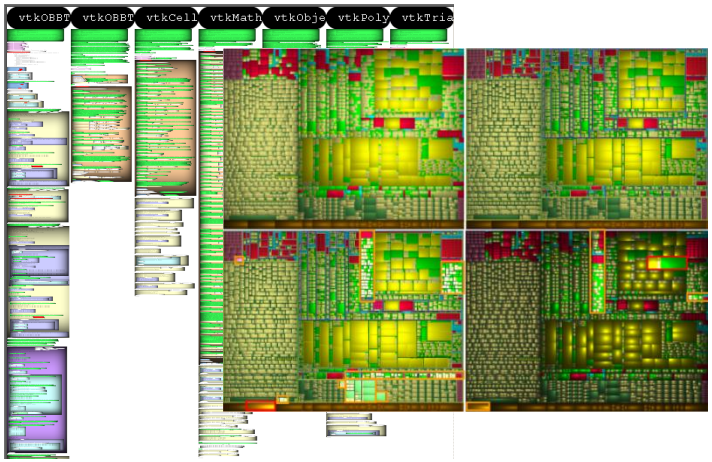
Visual Analytics Tooling



2002 SoftVision

SoftVision
- software architectures
- node-link 2D/3D layouts

Visual Analytics Tooling

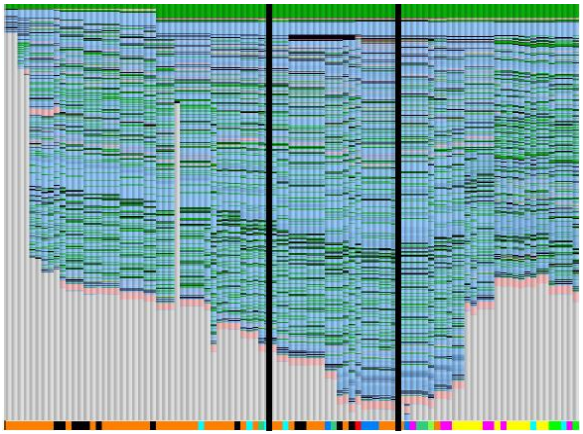


2004 VCN

VCN (Visual Code Navigator)

- code syntax structure
- dense pixel layouts
- gcc-based static analysis

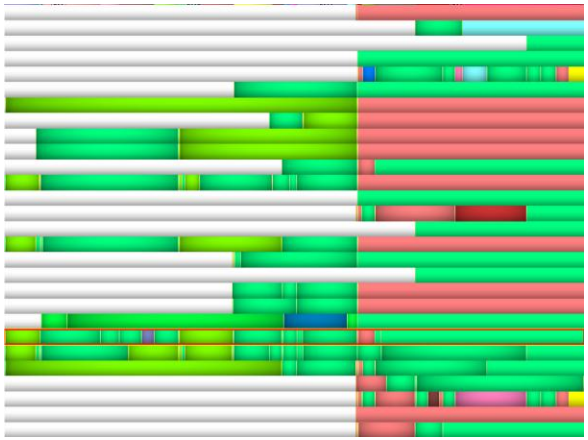
Visual Analytics Tooling



2005 CVSscan

CVSscan
- line-level code evolution
- dense pixel layouts
- CVS repositories

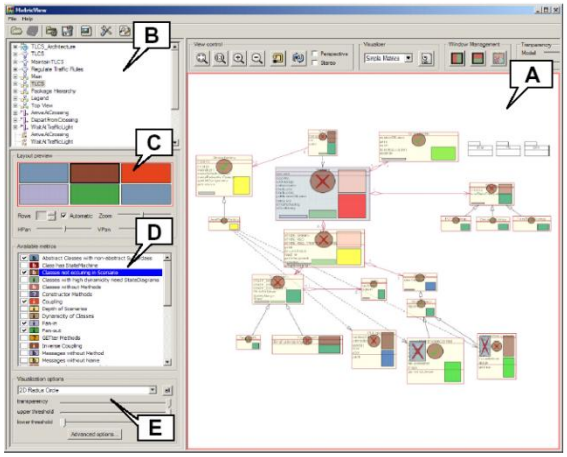
Visual Analytics Tooling



2006 CVSgrab

CVSgrab
- file-level code evolution
- dense pixel layouts
- CVS, SVN repositories

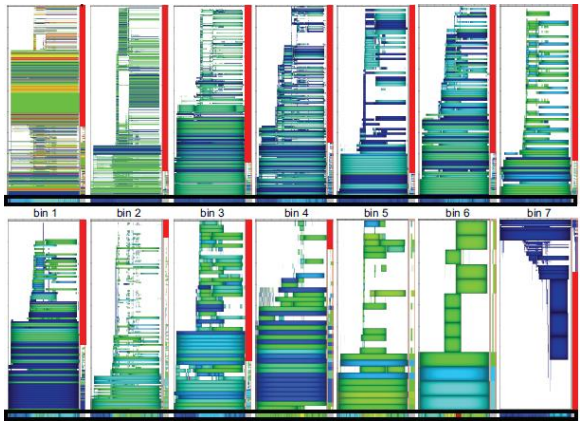
Visual Analytics Tooling



2006 MetricView

MetricView
- architectures and metrics
- UML layouts and glyphs
- XMI diagrams

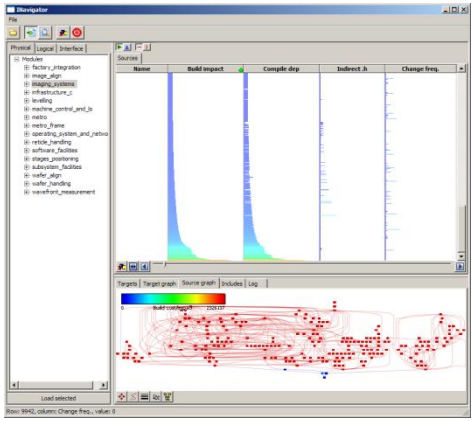
Visual Analytics Tooling



2007 MemoView

MemoView
- dynamic memory allocations
- dense pixel layouts
- third-party traces/logs

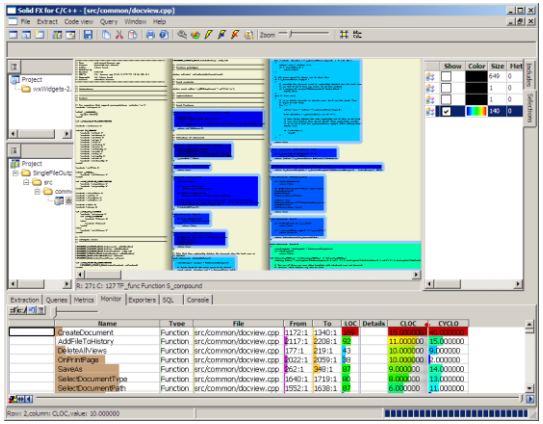
Visual Analytics Tooling



2007 SolidBA

SolidBA
- build dependencies
- table lenses, graphs
- C/C++ lightweight static analysis

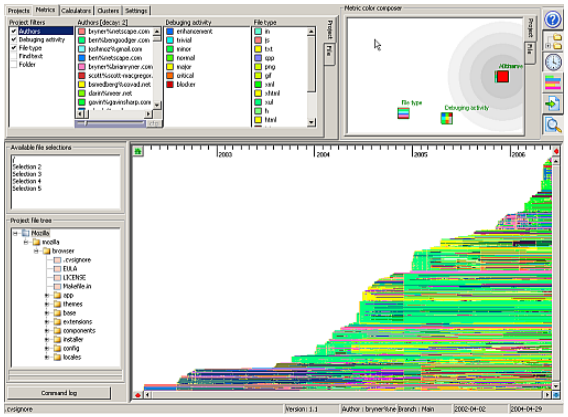
Visual Analytics Tooling



SolidFX
- code structure, metrics, dependencies
- dense pixel layouts, table lenses, UML diagrams
- C/C++ heavyweight static analysis

2008 SolidFX

Visual Analytics Tooling

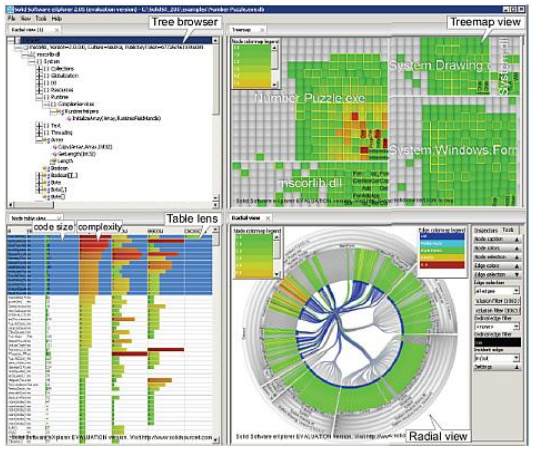


SolidSTA (Software Trend Analyzer)

- code evolution (line, file, project level)
- dense pixel layouts, table lenses, timelines
- CVS, SVN, Git, CM/Synergy repositories
- metric plug-ins

2008 SolidSTA

Visual Analytics Tooling

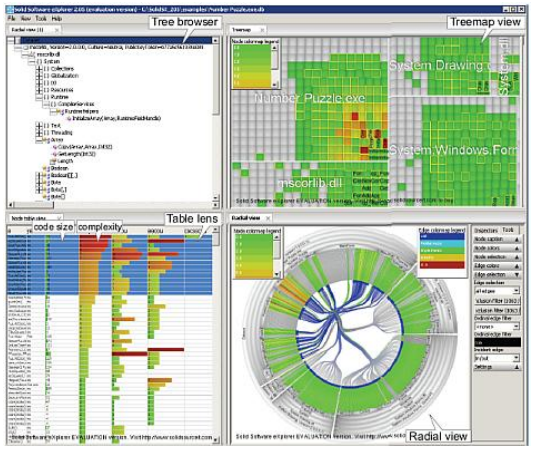


SolidSX (Software Explorer)

- code structure, dependencies, metrics
- bundled layouts, treemaps, table lenses
- C/C++, Java, .NET built-in analyzers
- Visual Studio integration

2009 SolidSX

Visual Analytics Tooling



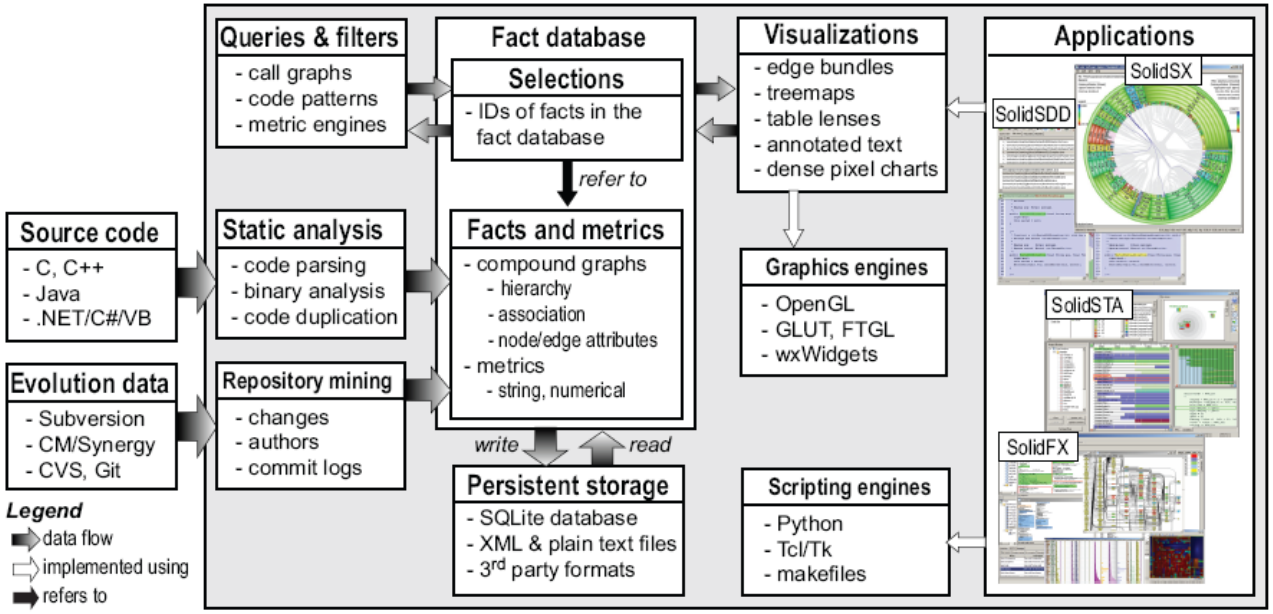
SolidSDD (Software Duplication Detector)

- code clones
- C/C++, Java, C#
- integrated visualization

2010 SolidDD

Common Design

- multiple 'fact databases'
  - custom format: ASTs
  - SQLite: anything else (metrics, call graphs, evolution data, ...)
- selections:
  - sets of ID's in the fact databases
  - inputs and outputs for all operations
- operations:
  - connected in a dataflow model via selections



Common Visualizations

- small set of choice

annotated text

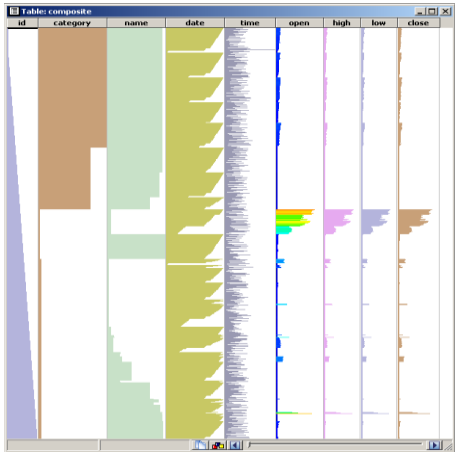
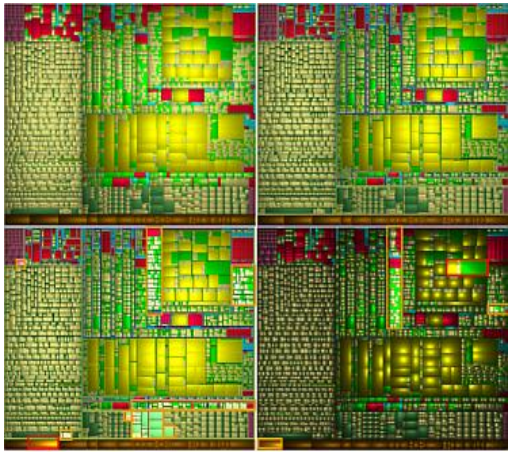
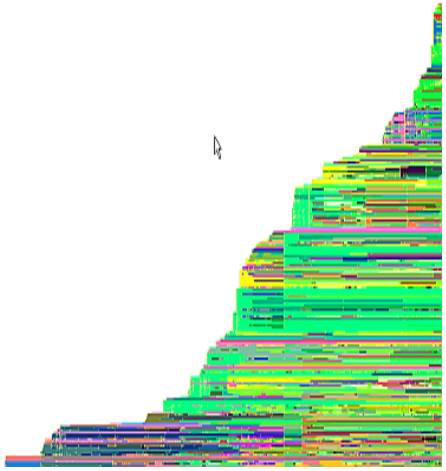


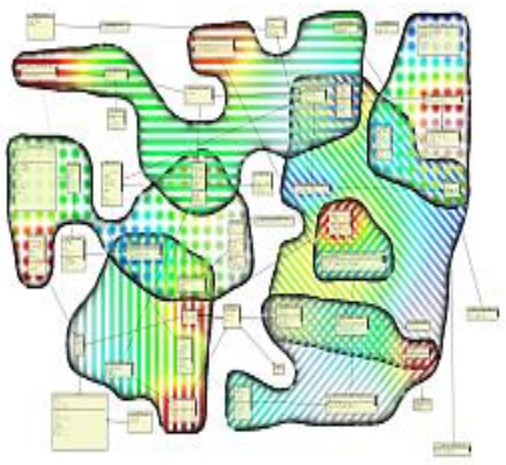
table lens



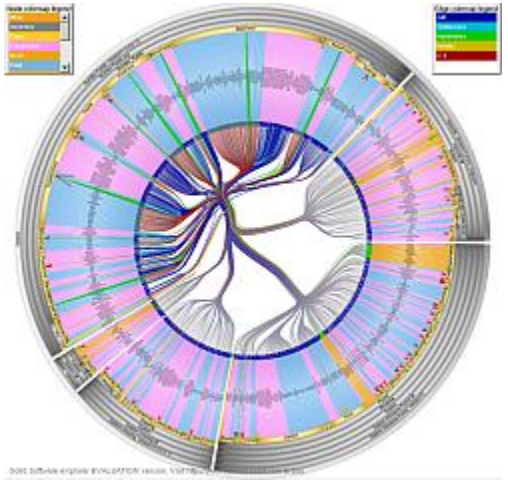
treemaps



timelines



UML diagrams



bundled layouts

Demonstration

SolidSX

- visual exploration of software structure, dependencies, metrics
- integrated static analyzers for C/C++, .NET/C#, Java
- open SQL/XML data formats
- pluggable metric engines
- Visual Studio integration

SolidSDD

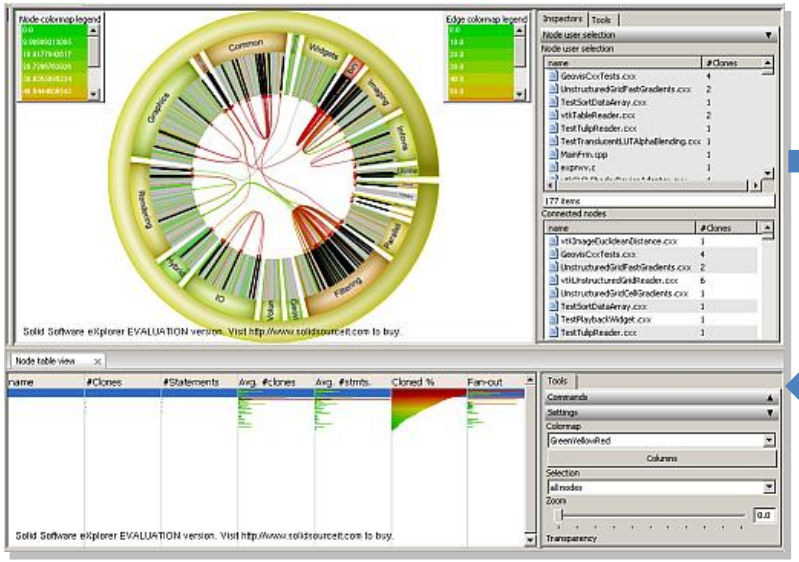
- visual exploration of code duplication (clones)
- integrated clone detector for C/C++, C#, Java
- open SQL data formats
- integrated with SolidSX

Demo

- VTK code base (3500 C/C++ files, 2.5 MLOC)
- clone detection: 5 minutes

Demonstration

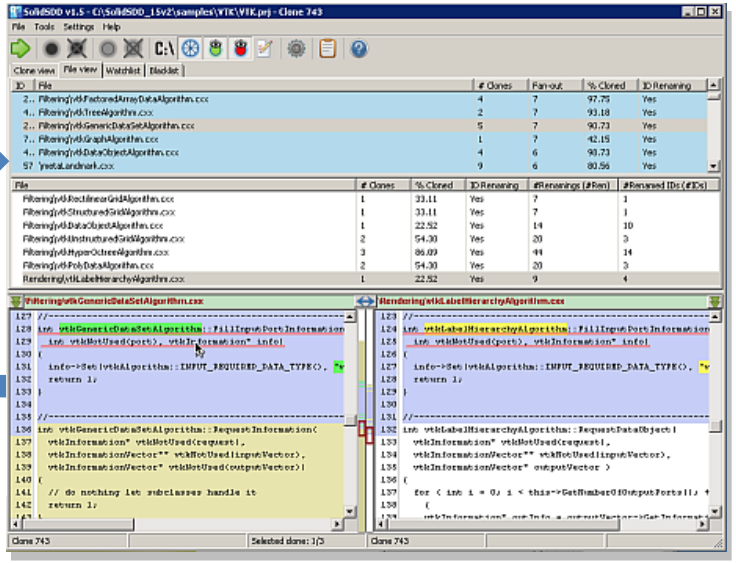
SolidSX visualization tool



selected code

clone database

SolidSDD clone extractor



data: SQLite
events: sockets

Implementation

Efficiency

- required to handle code bases of MLOC-size
- all core visualization/analyses implemented in C/C++ with OpenGL 1.1

Uniformity

- single GUI toolkit / look-and-feel: wxWidgets (Qt: equally good alternative)

Flexibility

- scripting layer: Python (smooth integration with C/C++)

Genericity

- use simple attribute-entity-relationship (AER) data model – wherever possible
- persistent storage and queries: SQLite – wherever possible
- XML: thanks but no thanks (does not scale for MLOC-size AER graphs!)

Toolchain integration

- data: shared on-disk SQLite databases
- events: socket-based communication
- very flexible: integrate with no programming (!)

Discussion

1. Should academic tools be of commercial quality?

- tool = carrier for testing new method? Polished implementation = waste
- tool = proof-point for a method's effectiveness? Good implementation = vital!

2. How to integrate and combine independently developed tools?

- several levels
  - dataflow: read/write databases in common formats (SQL, XMI, FAMIX, ...)
  - shared database: single format (Eclipse CDT, Intellisense)
  - common APIs: best but hardest (Mondrian, CodeCrawler, SolidSX)

3. What are lessons learned and pitfalls building tools?

- 2D vs 3D: software engineers do *not* (yet) accept 3D visualizations!
- interaction: minimalist design = best (hide rest under advanced options)
- scalability: vital for acceptance; dense pixel visualizations are best
- integration: **most crucial acceptance factor**

4. Are there any useful tool building patterns for tools?

- architecture: dataflow + shared database = most useful composition pattern
- visualizations: dense-pixel layouts are **best** since **scalable**
- heavyweight-vs-lightweight analysis:
  - both are useful and arguably required
  - simple database model favors their integration

5. What are effective techniques to improve the quality of academic tools?

- **usability** is the single most important factor to optimize
- problem: ‘value model‘ for academic work does not match the one of end-users!

6. How to compare/benchmark tools?

- lab studies: good for **technically** testing a new visualization/interaction method
- class studies: **biased**, as indicated by literature (see paper references)
- field studies: best, but **hardest**; extra effort often not compensated in academic model!
- ‘insight‘ is hard to quantify – the value is in the eyes of the beholder
- side-by-side tool comparisons: **good compromise**

7. What languages are best suited for building tools?

- core: C/C++ for absolute performance
- database: SQLite (fast, small, simple, portable)
- graphics: OpenGL 1.1 only
- scripting: Python (Tcl/Tk or Smalltalk possible, but harder)

Conclusions

Visual software analytics

- **effective:** added-value in solving real problems
- **hard:** lots of implementation/optimization effort
- **challenging:** the only way forward for software visualization (we believe)

See it yourself

www.cs.rug.nl/SoftVis: Free academic software visual analytics tools!
www.solidsourceit.com: Free-trial commercial-grade visual analytics tools!

Thank you!