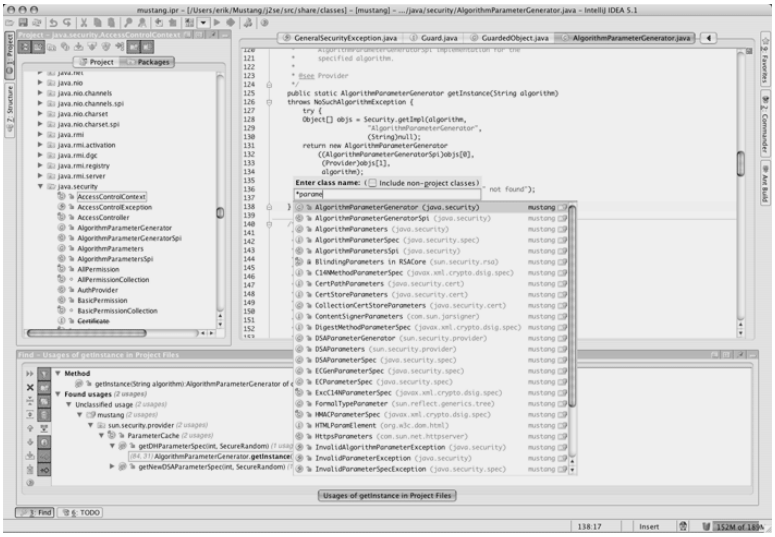
Software Visualization and Model Generation

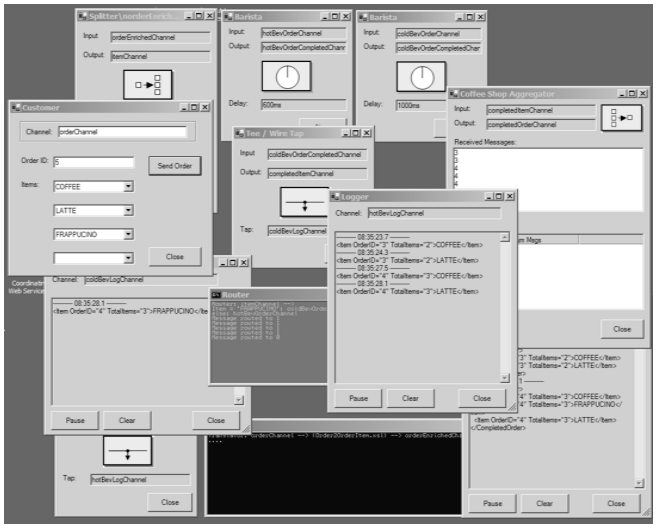
Erik Doernenburg
Software Developer
ThoughtWorks, Inc.

Gregor Hohpe
Software Engineer
Google, Inc.

Where are the most defects?



Which way do the messages flow?



A picture says more than 1000 words

- **The amount of information in current systems is beyond what we can handle**
- **Often we are only interested in a specific angle**
  - **Relationship between classes – not the entire source**
  - **Number of messages flowing – not the message content**
- **Humans are good at a spotting patterns in images…**

Being able to control large-scale systems is an illusion.
But we can observe what is happening…

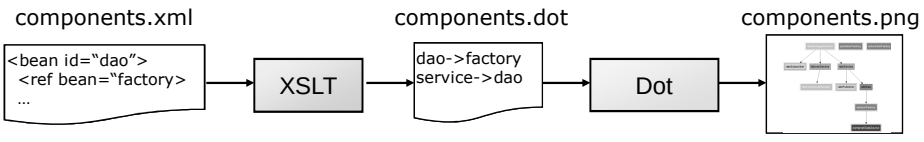
Where do we get the picture from?

- **Models created upfront convey a vision but usually don't reflect reality**
- **Generating a complete model for large systems is nearly impossible**
- **Systems evolve locally, often uncontrolled**
  - **Especially loosely coupled, dynamic systems (SOA)**
- **The best picture very much depends on the question you are trying to answer**

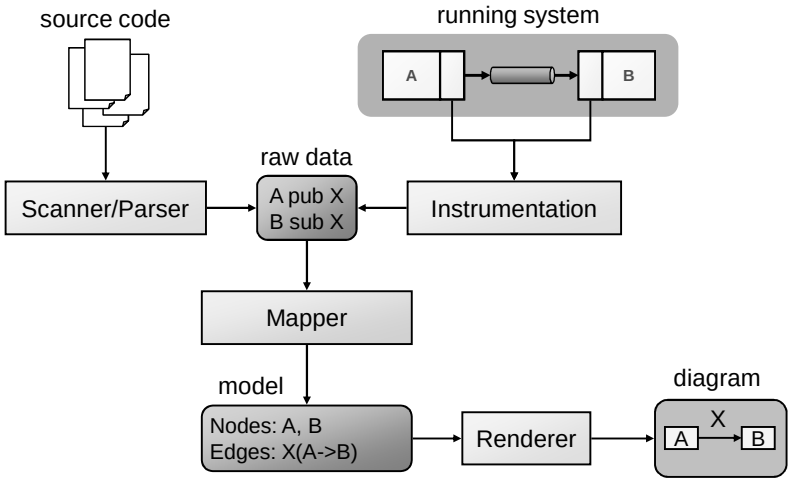
Need tools to create ad-hoc models more easily.

Example: Object Dependencies

- **Source: Spring bean configuration**
- **SpringViz, a small XSLT sheet, maps bean configuration to input for GraphViz Renderer**
- **Mapping/format hard-coded in style sheet**
- **Really simple but really useful**



Visualizing Software

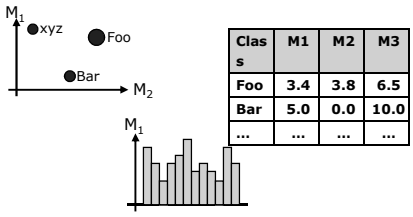


1. Select a meta-model

- **"A model that describes a model"**
  - **The elements a model can be composed of**
  - **How to combine these elements**
- **Example: meta-model for a class diagram**
  - **A class is a box with name, methods, fields,…**
  - **Available connectors: association, inheritance, aggregation…**
  - **Rules: no circles in inheritance etc.**
- **Sounds more scientific than it really is**
- **Usually pick from a few popular candidates**

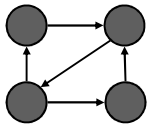
Common meta models

Metrics (Quantitative)

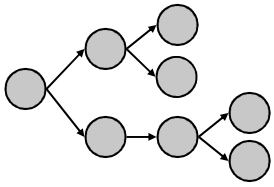


Class	M1	M2	M3
Foo	3.4	3.8	6.5
Bar	5.0	0.0	10.0
...	...	...	...

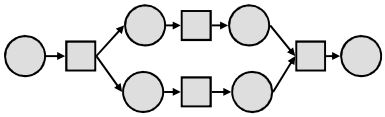
Directed Graph



Tree



Process Model (e.g. Petri Net)



2. Inspection / Instrumentation

Static Analysis

- **Inspect System Design**
- **Source code**
- **Configuration repository**
- **Scan / Parse into intermediate format**

Dynamic Analysis

- **Inspect Running System**
- **Profiling**
- **Listen to messages**
- **Log files**
- **Network sniffer**
- **Compiler decorator**

3. Mapping to the model

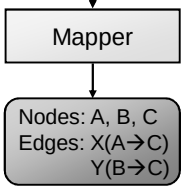
- **Map the gathered data onto the meta-model**

Senders

Comp.	Channel
A	X
B	Y

Receivers

Comp.	Channel
C	X
C	Y

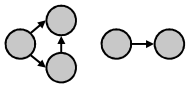


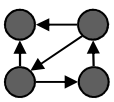
Graph Model

- **Example: Messaging System**
  - **Capture send / receive actions**
  - **Map onto directed graph**

4. Visualization / Validation

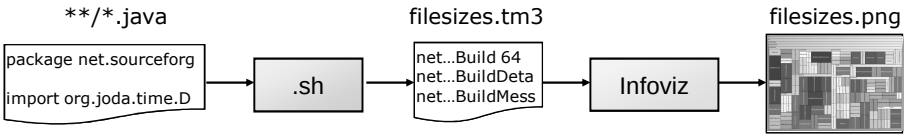
- **Graph rendering with GraphViz Dot**
  - **Automated graph layout tool**
  - **Takes textual input, produces graphics**
  - **Developed by AT&T, Common Public License**
- **Model validation**
  - **Do not simply observe but also verify & alert**
  - **Enforce rules or best practices**
  - **E.g., detect cycles, islands in a (dependency) graph**



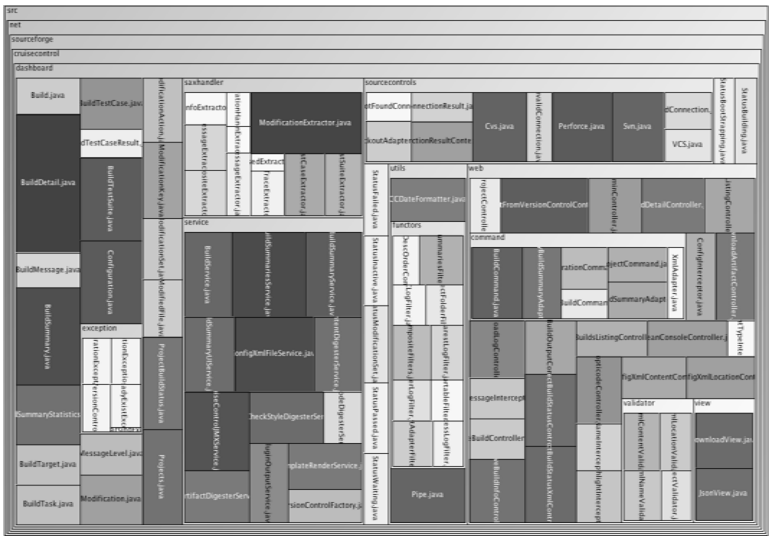


Example: Code Organisation

- **Static analysis: File metrics**
- **Shell script outputs file sizes in simple tree metric format (tm3)**
- **InfoViz used to visualize/render**
- **InfoViz is an interactive tool**



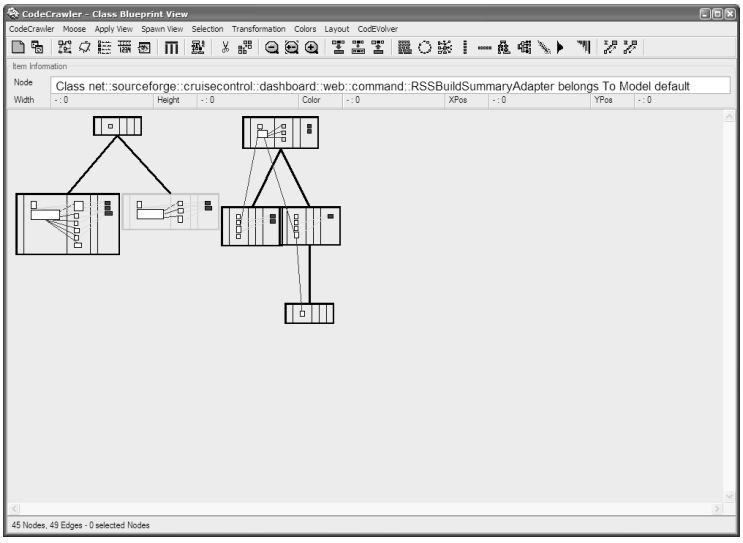
Package Tree View



Example: System Complexity

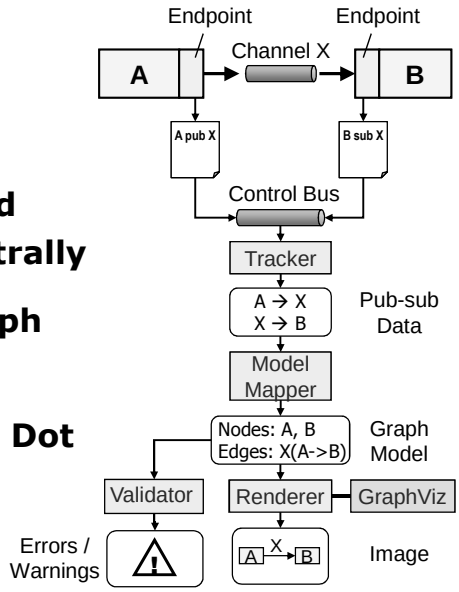
- **Static analysis: Source code analysis**
- **Code Crawler imports XMI and calculates metrics**
  - **NOA, NOM, WLOC**
- **Renders polymetric System Complexity view**
  - **Width, height, color used for metrics**
  - **Position used for tree layout of inheritance**
- **Goal of this view is to classify inheritance hierarchies**
  - **Subsystems**
  - **Large stand-alone classes**
- **Can use other views to understand inner workings of specific hierarchies**

CodeCrawler screenshot

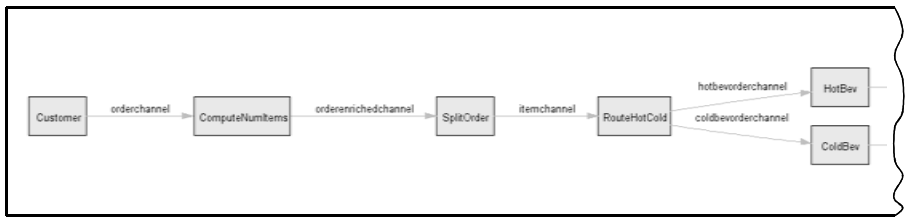


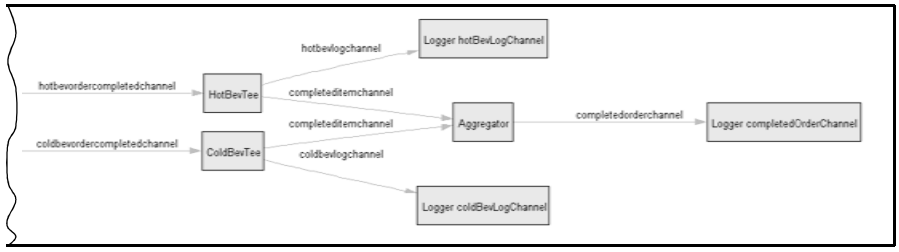
Example: Messaging

- **Dynamic Analysis: Instrument Message Sender and Receiver**
- **Collect publication and subscription data centrally**
- **Map to a Directed Graph model**
- **Render with GraphViz Dot**
- **Validate against rules**

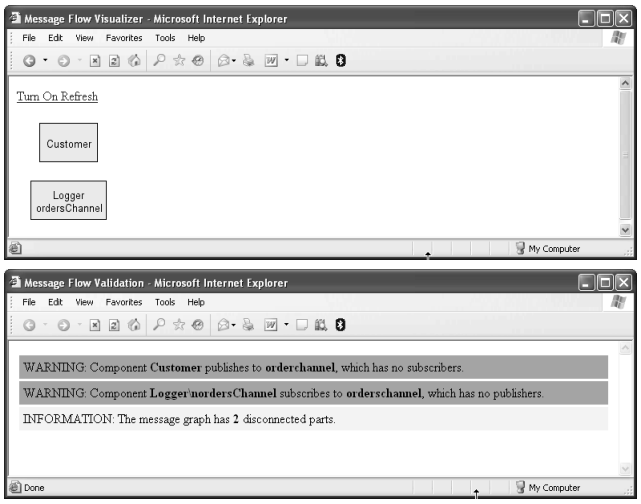


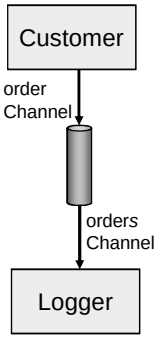
Message Flow Graph





Message Flow Validation





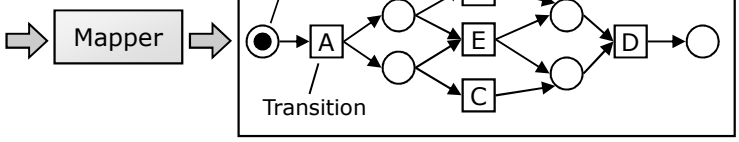
Example: Process Mining

- **A system performs a series of activities for each case (process instance)**
- **Dynamic analysis: log files of system activities**
- **Goal: create a process model from running system**
- **Meta-model: Petri Net**
- **Variety of algorithms and heuristics, e.g. α-Algorithm (Wil v.d. Aalst)**

Process Mining Example

Case	Activity
1	A
2	A
3	A
3	B
1	B
1	C
2	C
4	A
2	B
2	D
5	A
4	C
1	D
3	C
3	D
4	B
5	E
5	D
4	D

Log File



Process Model

What's next?

- **Technique applies to many levels**
  - **Single module source code**
  - **Multiple modules**
  - **Whole systems**
- **Diagrams hard to scale to huge systems**
  - **Interactive zoom / drill-down**
  - **Trends and outliers over time (e.g. source repository)**
- **Systems become larger, effects happen over time**
  - **Aggregation is key**
  - **Provide a "1,000 ft view"**

Resources

- **Tools**
  - **http://www.graphviz.org (Dot)**
  - **http://www.samoht.com/wiki/wiki.pl?SpringViz**
  - **http://www.eaipatterns.com (Messaging Visualization)**
- **Wil v.d. Aalst: Process Aware Information Systems, Wiley, 2005**
  - **www.processmining.org (Process Mining Tool)**
- **Michele Lanza's work (CodeCrawler and more)**
  - **http://www.inf.unisi.ch/faculty/lanza**
- **Software Visualization by Stephan Diehl**

Questions