# Building Software-Engineering Tools in Academia – A Tool Smith's Report –

*Dirk Beyer*
*Simon Fraser University*
*B.C., Canada*

# My Software Systems Experience

1. **BLAST**: Model Checking of Software
2. **CSIsat**: A Tool for LA+EUF Interpolation
3. **CrocoPat**: Simple and Efficient Relational Calculation
4. **CCVisu**: Automatic Software Decomposition
5. **CPAchecker**: Configurable Program Analysis
6. **EvolutionStoryboard**:
   Visualization of Software Structure Dynamics
7. **Chic**: Checking Software Interface Compatibility
8. **Rabbit**: Verification of Real-Time Systems

# Demo of two Examples

- CrocoPat
  - Component
  - Clear and simple interface: RSF, simple programming language, lib/ccommand-line tool
  - High quality implementation, code highly tuned for performance
    I really wanted to do this as perfect as possible.
    Crucial for correctness! (small defect → dramatic consequences)
- CCVisu
  - Component
  - Clear and simple interface: RSF, TXT or SVG or screen, lib/command-line tool
  - Started with prototypical impl. But result it is still really good.
    Relatively easy to test and validate.

# Holger's Questions and My Answers

- Should tool building remain a craft?
  → yes, it is even an art

- Should research prototypes be of commercial quality?
  → should commercial tools have academic quality?
  → prototypes are there as proof of concept
  → but we need them to reproduce results

- How to integrate/combine independently developed tools?
  → accept ideas as standard and conform (think RSF)

- What are the positive lessons learned in building tools?
  → very gentle users
  → effort in simplicity pays off

# Holger's Questions and Answers (2)

- What are the (recurring) pitfalls in tool building?
  → insufficient documentation, no tutorial provided
  → users not involved in development
  → not simple enough (needs to be ridiculously simple)
  → not the right license (use one non-free component …)
- What are the good practices and techniques?
  → publish tool on web, with data to reproduce results
  → good license (Apache or GPLv3)
  → try get other people involved
- Are there architectures and patterns for tool building?
  → have idea → implement → learn → redesign/refactor
  → converge to final solution
- How to compare or benchmark tools?
  → publish data, after some time somebody will factorize and republish as collection → benchmarks evolve naturally

# Lesson's Learned

- Basic:
  - small (code, binary, documentation)
  - simple (easy to apply, use)
  - easy to understand (simple concepts, short technical papers)
- Components:
  - make it component-based *and* be a component itself
  - provide a tutorial and reference manual
  - Is it easy to integrate into Rigi or SHriMP ?
- Language:
  - Java as language of choice (Bad experience with C++ and OCaml)
- Ownership:
  - active ownership of group leader essential (otherwise code will be gone with student)
  - understand, influence, contribute, use !
- License: (preferred Apache and GPLv3)
  - SW Free! (open source, allow modification and redistribution)