Software Research: Where do we go from here?

Benjamin C. Pierce
University of Pennsylvania

SDP Workshop, 18-19 April 2001

We're not doing so badly…

- **Present-day software engineering is actually astonishingly successful. Yes, we complain about cost, bugs, etc., but we are routinely building and using software systems of a size and complexity that could scarcely have been believed 20 years ago.**

- **How did we do this?**

By applying *lots and lots* of good ideas

But we can't stop here!

- **Does this mean we have "solved the software crisis"?**

  Of course not: there was never just one software crisis.

- **As our capabilities grow, so do our appetites**

Some new problem areas

- **Dynamically Assembled Software**
  - **Plug-ins and friends**
  - **Software leasing**
  - **Mobile agents**
- **Worldwide file sharing**
  - **Massive replication**
- **Interfacing security infrastructure(s) and application programs**
- **Concurrent / distributed programming “for the masses, by the masses”**

How Formal?

- **Formal methods**
  - **Hoare logic**
  - **Relational calculus**
  - **Process calculi**

Impractical for most applications, but rich source of powerful concepts (invariants, pre-/post-conditions, rely/guarantee, serializability, etc.

- **"Lightweight" formal methods**
  - **Type systems**
  - **Model checking**
  - **Run-time monitoring**

Big bang for buck

- **Pseudo-formal methods**
  - **UML**

Surprisingly effective…

95% of benefit is obtained by *attempting* to formalize software designs; *succeeding* not especially important!

- **Informal methods**
  - **Design patterns**
  - **Extreme programming**

Also very effective… "Mining the expertise of good programmers and managers"

Lightweight Formal Methods

Formal methods will never make a difference until they can be used by people that do not understand them. *---Tom Melham*

- **Types**
  - **Capture simple "contracts" between components and their environments**
  - **Conformance is checked automatically every time the program is compiled**
  - **Challenge: Can we engineer powerful type systems to make them useful in a highly dynamic environment? (Cf. proof-carrying code)**
- **Model checking**
  - **Very helpful in expositing flaws in hardware designs**
  - **Challenge: Can we do something similar for software?**
- **Run-time monitoring**
  - **Identify components that *actually* misbehave in real time, instead of detecting those that *might* in advance**
- **Need more techniques like these!**

Research Agenda

I. **Nurture foundations**

   Moderate number of small projects (1-2 PI)

II. **Encourage experimentation**

   Many small-to-medium-sized efforts (1-5 PI)

III. **Stress-test promising ideas**

   Small number of larger projects (4-15 PI)

Good ideas in software often take a *long* time to mature.

- E.g. garbage collection…
  - Invented in the late ‘50s
  - Gained widespread acceptance in the mid ‘90s!