

Software Challenges for the Aerospace Industry and Research

Patrick Cousot

Jerome C. Hunsaker Visiting Professor
Department of Aeronautics and Astronautics, MIT

cousot@mit.edu www.mit.edu/~cousot

École normale supérieure, Paris

cousot@ens.fr www.di.ens.fr/~cousot

Workshop on Critical Research Areas in Aerospace Software
Aero. Astro. Dept., MIT, August 9th, 2005

The software problem is misunderstood

Cost-effective trust and high-confidence in software and systems is **not yet well understood** by

- industries that are new to the problem ¹
- the general public ².

⇒ No massive support for **research on software safety and security**

¹ think to desktop operating systems 15 years ago when no one would care about bugs or to some telephone software nowadays

² subject to the massive presence of software in its daily and anodyne acts but nevertheless unconscious and not well informed about the potential risks (however the success of open-source probably comes in part from the lack of confidence in software that no one really trust)

The software problem is maturing

- Mature software industries (as found in aerospace) understand that more cost-effective trust and confidence in the software products they build or buy will be the **main challenge in the forthcoming 10 years** ³
- Large software manufacturers anticipate that their public image and market share can be put in question by unanticipated **software catastrophes**
- The cost of potential catastrophes and hazards should be integrated not only at the company level but also at the level of the **Information Society**.

Software engineering

- Present-day software engineering is **almost exclusively manual**, with very few automated tools
- The immediate consequence of the growing size of software will be an **explosion of the cost to maintain high-quality objectives**
- Software engineering must emerge from its present strong dependence on manual activities

³ Recent catastrophes due to software failures are there to remind them this priority.

The trust and confidence grand challenge

To master, at a competitive cost, trust and confidence in

- **software production** i.e. requirements, specification, design, development, documentation, configuration, modification, quality assurance of source and binary;
- **software interaction** with humans and/or complex hardware interfaces to physical systems (themselves more and more developed by software);
- **software integration** in a larger, more sophisticated, large-scale, usually distributed system.

Quality assurance

- Quality assurance can no longer be entirely based on the **development process**;
- Should be **based on the product itself**: all the dynamic properties are already present in the text of the program;
- The current simulation, audit, review and test technologies, which will remain indispensable in the future, are rapidly **reaching their limits**
- Must therefore be complemented by **new design, checking, verification and certification tools** to get trust and confidence in specifications and software.

Tools for trust and high-confidence in software and systems

- Do not reduce to the problem of **bugs**;
- Covers the ability to prove **program properties** (any question about the execution of a piece of software/system going beyond the current practice of execution sampling (test, demos, etc)).
- These tools and automated techniques, to help reasoning about software execution, should be independent of applications but could be specialized on various aspects such as **functionality, interoperability, safety, responsiveness, reliability, security, scalability**, etc.

-
- Then, depending on the domain of applications and corresponding standards, end-users could **concentrate on some of these aspects and tools depending on the type of application** (e.g. functionality for business, safety for fly-by-wire, security for service applications) and could even **impose the use of such certification tools in future standards**.