« Motivations for Static program Analysis »

Patrick Cousot

École normale supérieure
45 rue d'Ulm
75230 Paris cedex 05, France

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

VMCAI'05 Industrial Day

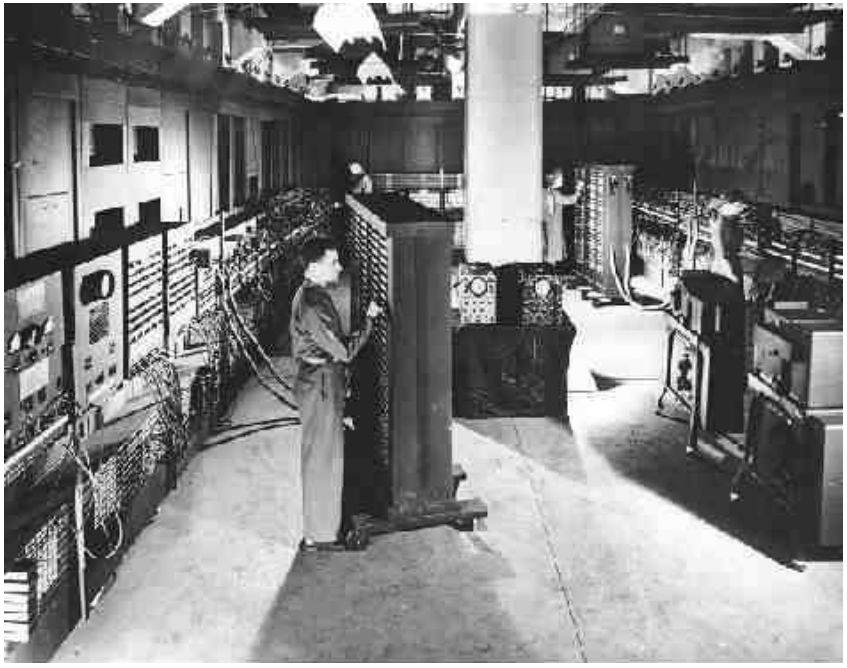
What is (or should be) the essential preoccupation of computer scientists?

What is (or should be) the essential preoccupation of computer scientists?

The production of reliable software, its maintenance and safe evolution year after year (up to 20 even 30 years).

Computer hardware change of scale

The 25 last years, computer hardware has seen its performances multiplied by 10^4 to $10^6/10^9$;



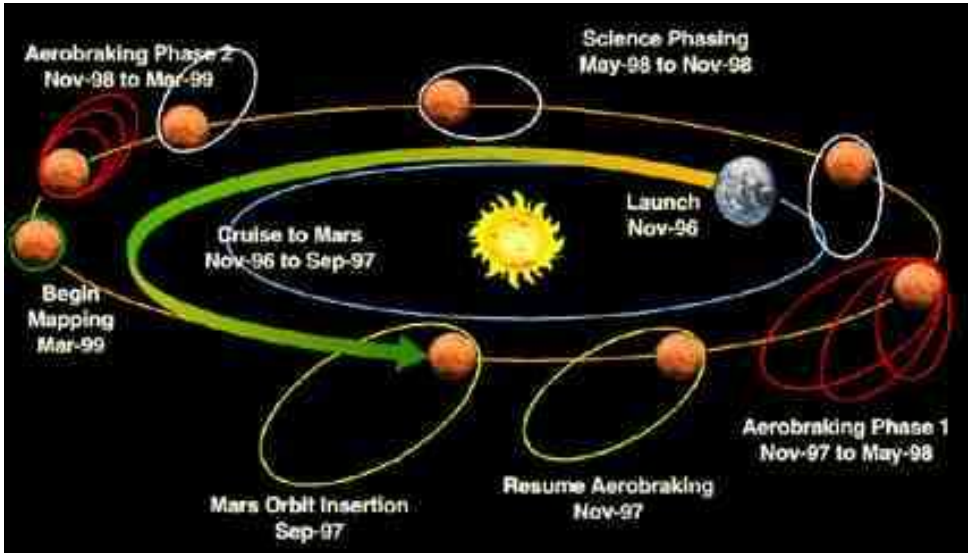
ENIAC (5000 flops)

Intel/Sandia Teraflops System (10^{12} flops)

The information processing revolution

A scale of 10^6 is typical of a significant **revolution**:

- Energy: nuclear power station / Roman slave;
- Transportation: distance Earth — Mars / Paris — Toulouse



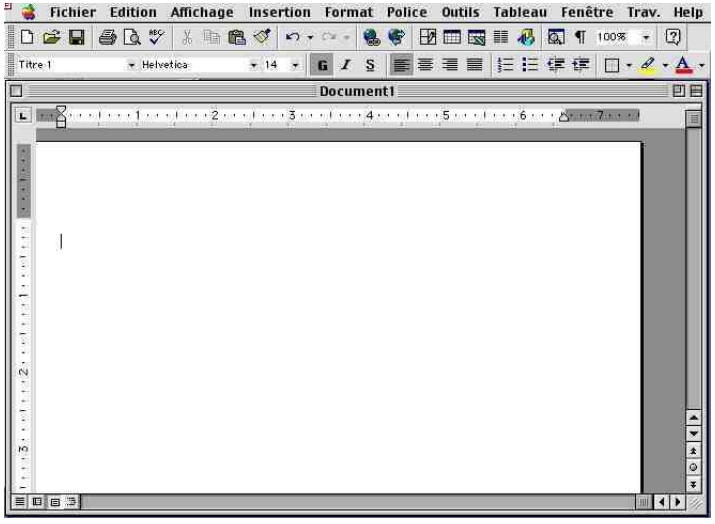
Computer software change of scale

– The size of the programs executed by these computers has grown up in similar proportions;

20 000 procedures;

400 files;

$>$ 15 years of development.

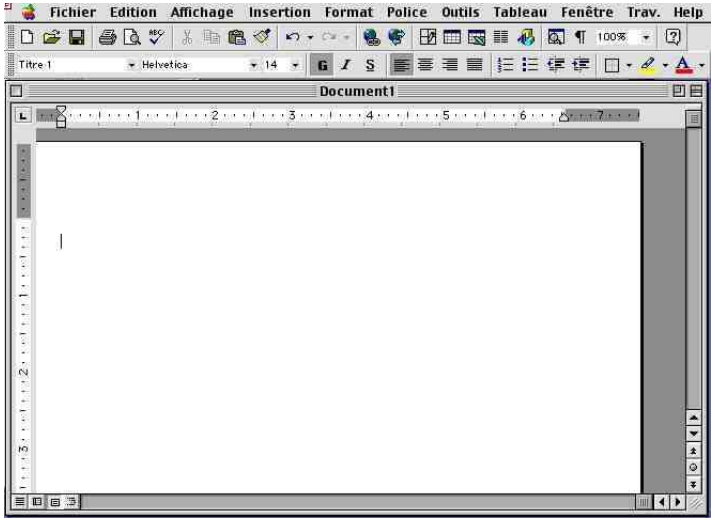


Computer software change of scale

– The size of the programs executed by these computers has grown up in similar proportions;

– **Example 1** (modern text editor for the general public):

- $>$ 1 700 000 lines of C [1];
- 20 000 procedures;
- 400 files;
- $>$ 15 years of development.

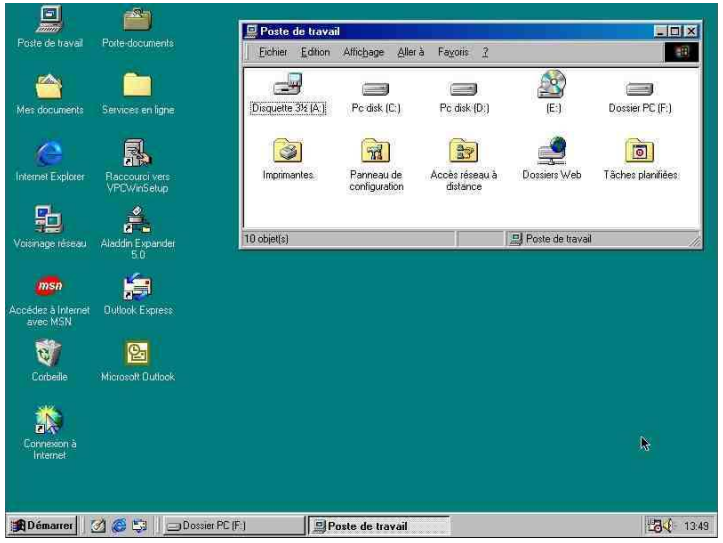


[1] full-time reading of the code (35 hours/week) would take at least 3 months!

Computer software change of scale (cont'd)

– **Example 2** (professional computer system):

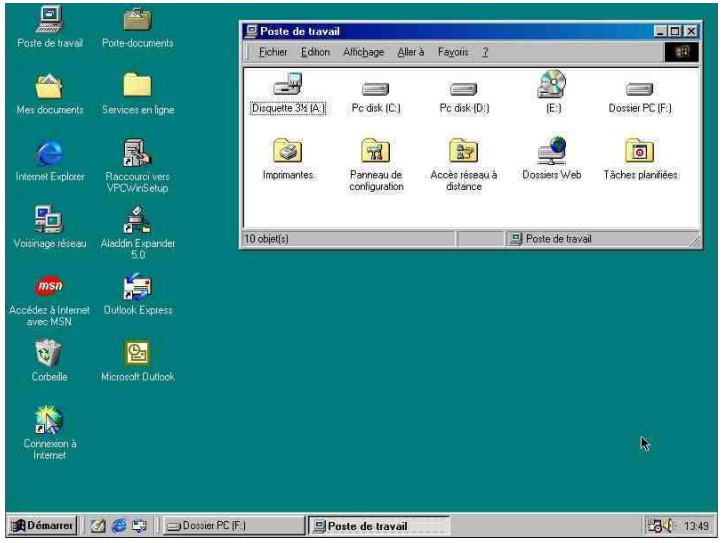
- 30 000 000 lines of code;



Computer software change of scale (cont'd)

– **Example 2** (professional computer system):

- 30 000 000 lines of code;
- 30 000 (known) bugs!



Bugs

– Software bugs
  - whether anticipated (Y2K bug)
  - or unforeseen (failure of the 5.01 flight of Ariane V launcher)

  are quite frequent;

Bugs 

– Software bugs
  - whether anticipated (Y2K bug)
  - or unforeseen (failure of the 5.01 flight of Ariane V launcher)

  are quite frequent;
– Bugs can be very difficult to discover in huge software;

Bugs

- Software bugs
  - whether anticipated (Y2K bug)
  - or unforeseen (failure of the 5.01 flight of Ariane V launcher)

  are quite frequent;
- Bugs can be very difficult to discover in huge software;
- Bugs can have catastrophic consequences either very costly or inadmissible (embedded software in transportation systems);

The estimated cost of an overflow

The estimated cost of an overflow

– **500 000 000 $**;

The estimated cost of an overflow

- **500 000 000 $**;
- Including indirect costs (delays, lost markets, etc):

2 000 000 000 $;

The estimated cost of an overflow

- **500 000 000 $**;
- Including indirect costs (delays, lost markets, etc):

  2 000 000 000 $;

- The financial results of Arianespace were **negative** in 2000, for the first time since 20 years.

Who cares?

– No one is legally responsible for bugs:

This software is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Who cares?

– No one is legally responsible for bugs:

> *This software is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.*

– So, no one cares about software verification

Who cares?

- No one is legally responsible for bugs:

  This software is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

- So, no one cares about software verification
- And even more, one can even make money out of bugs (customers buy the next version to get around bugs in software)

Why no one cares?

– Software designers don't care because there is no risk in writing bugged software

Why no one cares?

- Software designers don't care because there is no risk in writing bugged software
- The law/judges can never enforce more than what is offered by the state of the art

Why no one cares?

- Software designers don't care because there is no risk in writing bugged software
- The law/judges can never enforce more than what is offered by the state of the art
- Automated software verification by formal methods is undecidable whence thought to be impossible

Why no one cares?

- Software designers don't care because there is no risk in writing bugged software
- The law/judges can never enforce more than what is offered by the state of the art
- Automated software verification by formal methods is undecidable whence thought to be impossible
- Whence the state of the art is that no one will ever be able to eliminate all bugs at a reasonable price

Why no one cares?

- Software designers don't care because there is no risk in writing bugged software
- The law/judges can never enforce more than what is offered by the state of the art
- Automated software verification by formal methods is undecidable whence thought to be impossible
- Whence the state of the art is that no one will ever be able to eliminate all bugs at a reasonable price
- And so no one ever bear any responsability

Current research results

– Research is presently changing the state of the art (e.g. ASTRÉE)

Current research results

- Research is presently changing the state of the art (e.g. ASTRÉE)
- We can check for the absence of large categories of bugs (may be not all of them but a significant portion of them)

Current research results

- Research is presently changing the state of the art (e.g. ASTRÉE)
- We can check for the absence of large categories of bugs (may be not all of them but a significant portion of them)
- The verification can be made automatically by mechanical tools

Current research results

- Research is presently changing the state of the art (e.g. ASTRÉE)
- We can check for the absence of large categories of bugs (may be not all of them but a significant portion of them)
- The verification can be made automatically by mechanical tools
- Some bugs can be found completely automatically, without any human intervention

The next step (5 years)

– If these tools are successful, their use can be enforced by quality norms

The next step (5 years)

- If these tools are successful, their use can be enforced by quality norms
- Professional have to conform to such norms (otherwise they are not credible)

The next step (5 years)

- If these tools are successful, their use can be enforced by quality norms
- Professional have to conform to such norms (otherwise they are not credible)
- Because of complete tool automaticity, no one can be discharged from the duty of applying such state of the art tools

The next step (5 years)

- If these tools are successful, their use can be enforced by quality norms
- Professional have to conform to such norms (otherwise they are not credible)
- Because of complete tool automaticity, no one can be discharged from the duty of applying such state of the art tools
- Third parties of confidence can check software a posteriori to trace back bugs and prove responsabilities

A foreseeable future (10 years)

– The real take-off of software verification must be enforced

A foreseeable future (10 years)

- The real take-off of software verification must be enforced
- Development costs arguments have shown to be ineffective

A foreseeable future (10 years)

- The real take-off of software verification must be enforced
- Development costs arguments have shown to be ineffective
- Norms/laws might be much more convincing

A foreseeable future (10 years)

- The real take-off of software verification must be enforced
- Development costs arguments have shown to be ineffective
- Norms/laws might be much more convincing
- This requires effectiveness and complete automation (to avoid acquittal based on human capacity limitations arguments)

Why will "partial software verification" ultimately succeed?

– The state of the art will change toward complete automation, at least for common categories of bugs

Why will "partial software verification" ultimately succeed?

- The state of the art will change toward complete automation, at least for common categories of bugs
- So responsabilities can be established (at least for automatically detectable bugs)

Why will "partial software verification" ultimately succeed?

- The state of the art will change toward complete automation, at least for common categories of bugs
- So responsabilities can be established (at least for automatically detectable bugs)
- Whence the law will change (by adjusting to the new state of the art)

Why will "partial software verification" ultimately succeed?

- The state of the art will change toward complete automation, at least for common categories of bugs
- So responsabilities can be established (at least for automatically detectable bugs)
- Whence the law will change (by adjusting to the new state of the art)
- To ensure at least partial software verification

Why will "partial software verification" ultimately succeed?

- The state of the art will change toward complete automation, at least for common categories of bugs
- So responsabilities can be established (at least for automatically detectable bugs)
- Whence the law will change (by adjusting to the new state of the art)
- To ensure at least partial software verification
- For the benefit of all of us

THE END

More references at URL www.di.ens.fr/~cousot.

THE END, THANK YOU

More references at URL www.di.ens.fr/~cousot.