What can Industry 4.0 learn from SE?

CIRP GA 2016
Guimarães, August 21-27
2016

José Nuno Oliveira
HASLAB/ Univ. Minho & INESC TEC
for i = 1 to 4 do {industry (i);}
Introduction

Software

Lessons learned

Turning point

(“L’enfant terrible” is born)
1st NATO Conference on **Software Engineering**, Darmstadt, October 1968
Meanwhile (50 years)

“Traditional” engineering principles apply to process but not so well to product — why?
“L’enfant terrible”

Hardware and other “traditional” industrial products fabricated according to the laws of physics.

Software not governed by the laws of physics:
• it does not weight / does not smell
• it does not warm up / cool down
• it is chemically neutral ...

Anthony Oettinger (ACM President, 1967):

“(…) the scientific, rigorous component of computing, is more like mathematics than it is like physics”
Software = mathematics in motion

Can one pretend that software production is not affected by its special nature and simply move on?

People have tried to do so, for 50 years, with little success.

Still Oettinger (already in 1967):

"It is a matter of complexity. Once you start putting thousands of these instructions together you create a monster which is unintelligible to anyone save its creator and, most of the time, unfortunately even to the creator."
**Industry 4.0 and software**

Industry 4.0 to rely on highly sophisticated software on an unprecedented scale.

Billions, not thousands, of lines of code required to

\[
\text{for all do } \{ \text{human := robot} \}
\]

Software correctness and robustness therefore essential.
What have we learned about software?

Software lives on **abstraction**:

"The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise." (E. Dijkstra)

From a Robot Programming Tutorial:

"The fundamental challenge of all robotics is this: It is impossible to ever know the true state of the environment. A robot can only guess the state of the real world based on measurements returned by its sensors."

We have developed a sound theory for (safe) guessing called **abstract interpretation** — widely used in program analysis tools nowadays.
Type oriented programming

Something we’ve also learned is how important types are.

Every computation, piece of data should have a formal type.

Types permit (automatic) checking before building.

*Doing software without types is like doing biology without a post-Linnaean taxonomy …*

*Beware: most of the software running today is (still) untyped or too weakly typed (!)*
Parametricity and scalability

We also learned to appreciate **generic** (parametric) programs which automatically **instantiate** to specific needs.

**Polymorphic** types do this - nice theory called **parametric polymorphism** (John Reynolds, CMU).

So nice that one derives **properties** of programs **before** even **writing them** — very helpful in **correctness** arguments.

Thanks to techniques like “lazy programming” our **generic** programs have also become **scalable**.
We have learned to better understand and take advantage of this ‘*quinta essentia*’ of *programming*.

Thanks to *D&C* our programs have become *parallel*. Think of *Google, cloud* computing, ...
Cyber-security

Surely the most critical problem ahead.

\[ \text{for some do } \{ \text{human := intruder} \} \]

We are learning how to use number theory and automata theory to build software that is provably secure.
Contract-oriented programming

We have also learned that, as in the regular functioning of any society, programming should be based on formal contracts validated using the underlying maths.

Contracts ensure safety and security essential to safety-critical equipment operation.
What can I4.0 learn from SE?

Level of **sophistication** and **safety** needed in **I4.0** incompatible with **ad hoc** software development.

**I4.0** should invest on **high-assurance**, **parametric** software **components** developed on a grand scale.

Opportunity for developing widely available, **certified cyber physical component** (CPC) libraries.