Course Overview

Julio Mariño

Rigorous Software Development
EUROPEAN MASTER IN SOFTWARE ENGINEERING/ MASTER IN SOFTWARE AND SYSTEMS
Universidad Politécnica de Madrid/IMDEA Software

September 2017

rigorous?

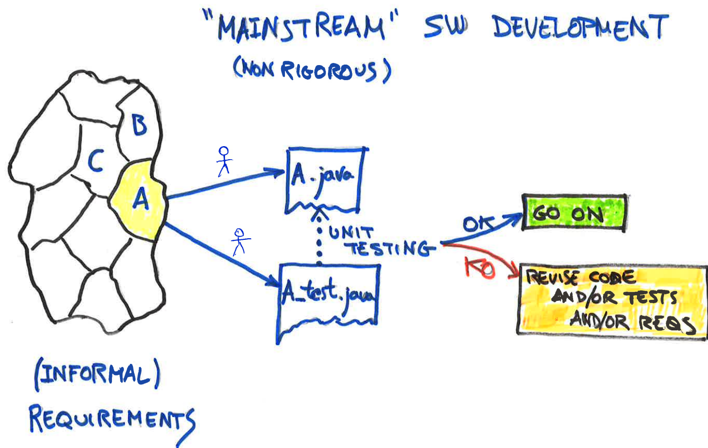
the game of the name

- Most software projects are validated by means of *testing*.
- As testing is, in general, nonexhaustive, it cannot prove by itself the absence of errors in software (it can reveal errors, of course).
- By rigorous we mean ways of developing software that can be used to *mathematically* rule out the possibility of errors.
- So, rigorous software development will make use of logic and arithmetic in order to build defect-free software. In other words, we will use *formal methods* to avoid errors.
- However, not every formal method is necessarily rigorous.

POLITÉCNICA

business as usual

software as it is commonly developed



business as usual

software as it is commonly developed

- The starting point is a set of informal (natural language) requirements that can be split into several components: A, B, etc.
- From these pieces programmers produce *implementation* files (`A.java`, etc.) and *unit test* files (`A_test.java`, etc.)
- If a test fails, we know that something went wrong. It can be the implementation (true positive), or the test code (false positive).
- As the testing code is generally not difficult to write, a false positive usually means that both programmers understood the requirements in different ways.
- Some common methodologies (test-driven development, etc.) use the unit tests as substitute for a more formal requirements specification.
- If the tests do not fail we cannot claim the implementation to be correct.

POLITÉCNICA

classical program verification

the oldest rigorous method

"CLASSICAL" FORMAL VERIFICATION

B

C

A

A.java

Ã

VERIF. ENGINE

(INFORMAL) REQUIREMENTS

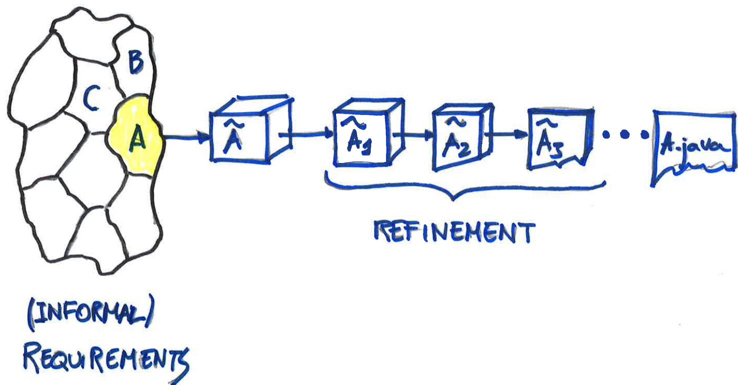
classical program verification

the oldest rigorous method

- We asume that somebody has already produced an implementation (`A.java`).
- In order to prove it correct, a formal specification ($\tilde{A}$) is produced by someone trained in formal logic.
- Then, it is necessary to build a formal proof that the implementation behaves as specified.
- This verification step is costly in terms of time, trained personnel and the machinery needed to halp in the process.
- We will devote the first weeks of our course to this topic.
- The logical framework in which specifications and proofs will be written will be Floyd-Hoare logic.
- The tool we will use to make this easier will be the *Dafny* system from Microsoft Research.
- There will be an exercise sheet to be handed in individually.

correctness by construction

do not verify: just do it right!



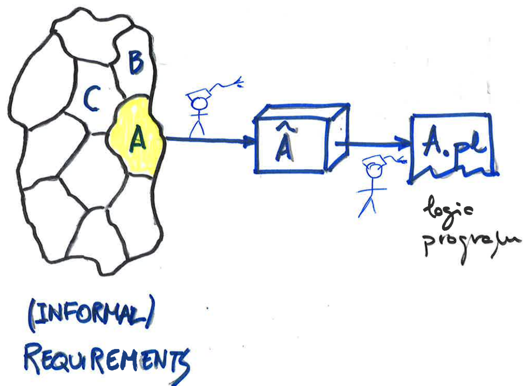
correctness by construction

do not verify: just do it right!

- Program verification is not scalable: it tends to be tremendously expensive when applied to large systems.
- Correctness by construction is a radical departure from program verification that tries to attack the scalability problem: instead of trying to match an existing implementation with a formal specification, the code is *derived* from the specifications so that correctness is ensured from the beginning.
- The specifications must be transformed (refined) in several steps, so that each step makes the formalism closer to an executable code while preserving the meaning of the previous statement.
- The core part of the course is the development of a use case using this approach, in groups of three students.
- The methodology used will be the B-method by J.R. Abrial, the language will be Event-B and the helper tool will be *Rodin*.
- Abrial gained popularity after generating the Ada code for the (driverless) line 14 of the Paris underground using the B-method.

POLITÉCNICA

executable specifications



executable specifications

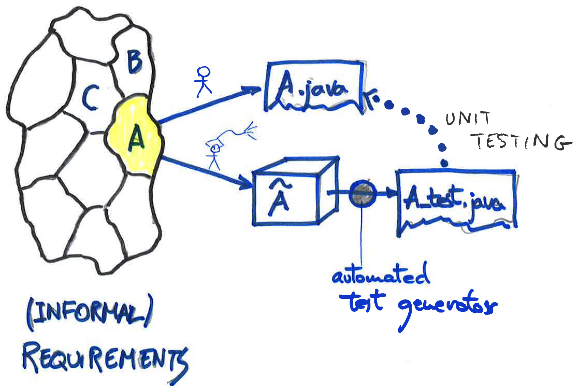
best of the two worlds?

- Executable specifications can be seen as an extreme case of correctness by construction where refinements only takes one step, although they actually appeared first.
- Executable specifications rely on declarative programming languages. In these languages programs admit a dual reading: as executable instructions and as theories (i.e. sets of formulas).
- Unsurprisingly, they can be unsatisfactory in both sides: inefficient in terms of execution speed and less expressive than the full logic used to write the specification in the first place.
- However, for prototyping, and in certain application areas, they can be priceless.
- We devote some lectures to this topic, using Prolog or Maude as declarative language.

POLITÉCNICA

property based testing

not really rigorous, but very helpful in practice



property based testing

not really rigorous, but very helpful in practice

- Finally, property-based-testing gives an unexpected twist to unit testing.
- It looks like test-driven development, but the second programmer does not write the unit tests code. Instead, she produces a formal specification and *generates* the testing code from them, using a very simple tool.
- This is not a rigorous method, as the validation is still provided by nonexhaustive testing, but companies are less reluctant to adopt it because it is *nonintrusive*: the other programmer is going to have the desired tests and in the same time or less. Besides, the generator can generate larger and better tests, and the specifications are reusable.
- This is an example of the so-called lightweight formal methods.

POLITÉCNICA