Developing Software Rigorously: Introduction and Motivation[1]

Manuel Carro
manuel.carro@upm.es

Universidad Politécnica de Madrid &
IMDEA Software Institute

[1] Many slides borrowed from J. R. Abrial and M. Butler

Take notes

TECHNOLOGY

To Remember a Lecture Better, Take Notes by Hand

Students do worse on quizzes when they use keyboards in class.

Picture & headline © *The Atlantic*

https://www.theatlantic.com/technology/archive/2014/05/to-remember-a-lecture-better-take-notes-by-hand/361478/

I will make notes / slides available *after* the lectures

I will ask you to work during the lectures

Plan

- Three-hour lectures.
  - Three 50-minute sections with ten-minute breaks.
  - Worked well in previous years.
- Homework + term project (with presentation).
- Final exam for those who **choose not** to do HW + project.
- Hands-on lectures when possible.

Purpose of the course

- To give you some insights about modelling and formal reasoning
- To show how programs can be *correct by construction*
- To show that modelling can be made practical
- To illustrate this approach with many examples

Requirements

No formal requirements.
But I expect you to be familiar with a series of topics:

- You should have a fairly ample base in programming.
- You should have a working knowledge of first order logic.
- You should feel comfortable with rigorous / math reasoning.

What you will learn

By the end of the course you should be comfortable with:

- Modelling (versus programming).
- Abstraction and refinement.
- Some mathematical techniques used to reason about programs.
- Proving as a means to construct (provably) correct programs.
- Using tools to help in the above.

Software is omnipresent

Today's car: typically 100+ microprocessors, 100 M. lines of code, 20.000 programmer years.

Software is omnipresent

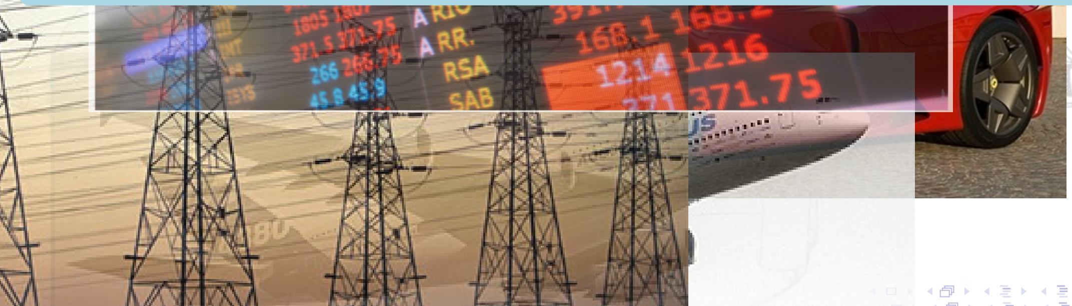
Plane: computers manage controls, calculate routes, ...

Software is omnipresent

Large interconnected systems: independent, isolated, automatic decision making (which must be globally correct).

Software is omnipresent

- Cell phones (from O.S. to compression algorithms to user interfaces).
- HDTV (routing, encoding and decoding), Netflix, . . .
- Buying and selling on the Internet (web interfaces, databases, encryption).
- Stock market (algorithmic trading, high frequency trading).
- Skype, Whatsapp, AirBnB, idealista, GroupOn, FB, Steam, Spotify, E-Banking, Google Maps / Waze, Uber / Lyft, Tesla, . . .



- Managed by extremely complex and *"intelligent"* software.
- All of them *critical* to a certain degree.
- Some **extremely** critical

- √ Managed by extremely complex and *"intelligent"* software.
- √ All of them *critical* to a certain degree.
- √ Some **extremely** critical

Overall challenge:

How to develop complex software, with resources that are **always limited**, ensuring that it will work correctly?

Growth in complexity and expectations

- Processes managed by computing systems increasingly complex.
- Same software is to run in several platforms.
- Computing systems interact more and more with each other.
- They should be increasingly autonomous.
- Reactive.

How far are we from giving reasonable guarantees?
(Only showing some types of problems)

Skype bug sends messages to unintended recipients.

How far are we from giving reasonable guarantees?

(Only showing some types of problems)

Skype bug sends messages to unintended recipients.

Apple's “in-app purchase” service for iOS bypassed by Russian hacker.

How far are we from giving reasonable guarantees?
(Only showing some types of problems)

Skype bug sends messages to unintended recipients.

Apple's "in-app purchase" service for iOS bypassed by Russian hacker.

German security experts find major flaw in credit card terminals.

How far are we from giving reasonable guarantees?
(Only showing some types of problems)

Skype bug sends messages to unintended recipients.

Apple's "in-app purchase" service for iOS bypassed by Russian hacker.

German security experts find major flaw in credit card terminals.

Defects leave critical military, industrial infrastructure open to hacks (Niagara Framework, linking 11+ million devices in 52 countries).

How far are we from giving reasonable guarantees?
(Only showing some types of problems)

Skype bug sends messages to unintended recipients.

Apple's "in-app purchase" service for iOS bypassed by Russian hacker.

German security experts find major flaw in credit card terminals.

Defects leave critical military, industrial infrastructure open to hacks (Niagara Framework, linking 11+ million devices in 52 countries).

Hackers expose 453,000 credentials allegedly taken from Yahoo service.

How far are we from giving reasonable guarantees?
(Only showing some types of problems)

- Skype bug sends messages to unintended recipients.
- Apple's "in-app purchase" service for iOS bypassed by Russian hacker.
- German security experts find major flaw in credit card terminals.
- Defects leave critical military, industrial infrastructure open to hacks (Niagara Framework, linking 11+ million devices in 52 countries).
- Hackers expose 453,000 credentials allegedly taken from Yahoo service.
- Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).

How far are we from giving reasonable guarantees?
(Only showing some types of problems)

Skype bug sends messages to unintended recipients.

Apple's "in-app purchase" service for iOS bypassed by Russian hacker.

German security experts find major flaw in credit card terminals.

Defects leave critical military, industrial infrastructure open to hacks (Niagara Framework, linking 11+ million devices in 52 countries).

Hackers expose 453,000 credentials allegedly taken from Yahoo service.

Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).

Still infected, 300,000 PCs to lose Internet access.

How far are we from giving reasonable guarantees?
(Only showing some types of problems)

- Skype bug sends messages to unintended recipients.
- Apple's "in-app purchase" service for iOS bypassed by Russian hacker.
- German security experts find major flaw in credit card terminals.
- Defects leave critical military, industrial infrastructure open to hacks (Niagara Framework, linking 11+ million devices in 52 countries).
- Hackers expose 453,000 credentials allegedly taken from Yahoo service.
- Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).
- Still infected, 300,000 PCs to lose Internet access.
- Apple fixes App Store DRM error, crash-free downloads resume.

How far are we from giving reasonable guarantees?
(Only showing some types of problems)

- Skype bug sends messages to unintended recipients.
- Apple's "in-app purchase" service for iOS bypassed by Russian hacker.
- German security experts find major flaw in credit card terminals.
- Defects leave critical military, industrial infrastructure open to hacks (Niagara Framework, linking 11+ million devices in 52 countries).
- Hackers expose 453,000 credentials allegedly taken from Yahoo service.
- Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).
- Still infected, 300,000 PCs to lose Internet access.
- Apple fixes App Store DRM error, crash-free downloads resume.
- "Find and Call" app becomes first trojan to appear on iOS App Store.

How far are we from giving reasonable guarantees?

(Only showing some types of problems)

Skype bug sends messages to unintended recipients.

Apple's "in-app purchase" service for iOS bypassed by Russian hacker.

German security experts find major flaw in credit card terminals.

Defects leave critical military, industrial infrastructure open to hacks (Niagara Framework, linking 11+ million devices in 52 countries).

Hackers expose 453,000 credentials allegedly taken from Yahoo service.

Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).

Still infected, 300,000 PCs to lose Internet access.

Apple fixes App Store DRM error, crash-free downloads resume.

"Find and Call" app becomes first trojan to appear on iOS App Store.

iOS, Mac app crashes linked to botched FairPlay DRM.

How far are we from giving reasonable guarantees?
(Only showing some types of problems)

July 16, 2012: Skype bug sends messages to unintended recipients.

July 13, 2012: Apple's "in-app purchase" service for iOS bypassed by Russian hacker.

July 13, 2012: German security experts find major flaw in credit card terminals.

July 13, 2012: Defects leave critical military, industrial infrastructure open to hacks (Niagara Framework, linking 11+ million devices in 52 countries).

July 12, 2012: Hackers expose 453,000 credentials allegedly taken from Yahoo service.

July 12, 2012: Mountain Lion (Mac OS X version) sends some 64-bit Macs to sleep (related to graphics drivers).

July 7, 2012: Still infected, 300,000 PCs to lose Internet access.

July 6, 2012: Apple fixes App Store DRM error, crash-free downloads resume.

July 5, 2012: "Find and Call" app becomes first trojan to appear on iOS App Store.

July 5, 2012: iOS, Mac app crashes linked to botched FairPlay DRM.

Just two weeks

The Ariane 5 incident

Example: effect of a *single* integer overflow

The Ariane 5 incident

Example: effect of a *single* integer overflow

- June 4, 1996: After launch, the Ariane 5 rocket exploded.
- This was its maiden voyage.
- It flew for about 37 Sec only in Kourou's sky.
- No injury in the disaster.

The story

- Normal behavior of the launcher for 36 Sec after lift-off
- Failure of both Inertial Reference Systems almost simultaneously
- Strong pivoting of the nozzles of the boosters and Vulcan engine
- Self-destruction at an altitude of 4000 m (1000 m from the pad)

More details

- Both inertial computers failed because of the overflow of one variable
- This caused a software exception that stopped these computers
- These computers sent post-mortem info through the bus
- Normally, main computer receives velocity info through the bus
- The main computer was confused and pivoted the nozzles

More details

- The faulty program was working correctly on Ariane 4
- The faulty program was not tested for A5 (since it worked for A4)
- But the velocity of Ariane 5 was far greater than that of Ariane 4
- That caused the overflow in one variable
- The faulty program happened to be useless for Ariane 5

Messages

- Clear, up to date, realistic requirements
- Relationship requirements / programs
- Proof that programs were built according to requirements

> *Note: we will not deal with* requirement engineering, *which is related and very interesting in itself.*

How?

How can we **ensure** that a program does what it is supposed to do?

How?

How can we **ensure** that a program does what it is supposed to do?

1. How do we state what is it supposed to do?

How?

How can we **ensure** that a program does what it is supposed to do?

1. How do we state what is it supposed to do?
2. How do we build the program?

How?

How can we **ensure** that a program does what it is supposed to do?

1. How do we state what is it supposed to do?
2. How do we build the program?
3. How do we prove that the program performs according to specifications?

How?

How can we **ensure** that a program does what it is supposed to do?

1. How do we state what is it supposed to do?
2. How do we build the program?
3. How do we prove that the program performs according to specifications?

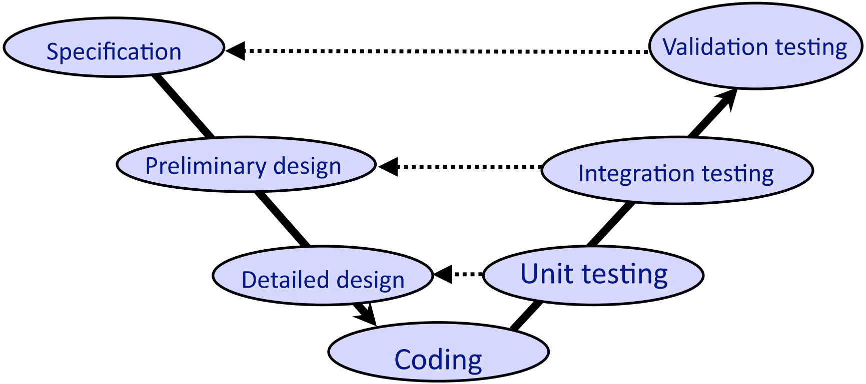
...in a way that is (a) dependable and (b) cost-effective?

Rate of error discovery

Cost of error fixes

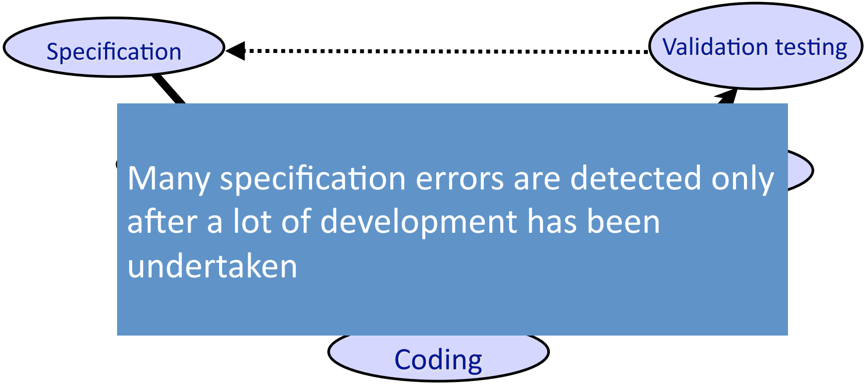
The V model

When are errors discovered in the V Model?



The V model

When are errors discovered in the V Model?



Many specification errors are detected only after a lot of development has been undertaken

Some sources of errors

- Lack of precision
  - Ambiguities
  - Inconsistencies
- Too much complexity
  - Complexity of requirements
  - Complexity of operating environment
  - Complexity of designs

Some sources of errors

- Lack of precision
  - Ambiguities
  - Inconsistencies
- Too much complexity
  - Complexity of requirements
  - Complexity of operating environment
  - Complexity of designs

Some preventive measures

- Early stage analysis
  - Precise descriptions of intent
  - Amenable to analysis by tools
  - Identify and fix ambiguities and inconsistencies as early as possible
- Mastering complexity
  - Encourage abstraction
  - Focus on what a system does
  - Early focus on key / critical features
  - Incremental analysis and design

Formal methods

- Rigorous techniques for formulation and analysis of systems
- They facilitate:
  - Clear specifications (contract)
  - Rigorous validation and verification

Formal methods

- Rigorous techniques for formulation and analysis of systems
- They facilitate:
  - Clear specifications (contract)
  - Rigorous validation and verification

If we do not capture precisely what a system ought to do,
there is little chance that we can decide whether it actually does it

Formal methods

- Rigorous techniques for formulation and analysis of systems
- They facilitate:
  - Clear specifications (contract)
  - Rigorous validation and verification

> *If we do not capture precisely what a system ought to do,*
> *there is little chance that we can decide whether it actually does it*

Deciding whether it does that it ought to do

Validation: Did we specify the right system?

- Answered informally: *did we build the right system?*

Verification: Does the finished product satisfy the specification?

- Can be answered formally: *did we build the system right?*

Specifications and the real world?

How can specifications be used?

- Use specifications to build tests (generation of tests based on specifications).
- Use specifications to check that a program computes what it should (static analysis, verification, model checking).
- Use specifications to compute (functional / logic / equational programming).
- Use specifications to drive the generation of a program (correctness by construction, automatic code generation).

How can guarantees be given?

- Enlightened management: of little help.
- Convincing arguments beyond any reasonable doubt:
  - Formal basis.
  - Proofs based on rigorous methods.
- Carefully prove that programs will behave as expected.

How can guarantees be given?

- Enlightened management: of little help.
- Convincing arguments beyond any reasonable doubt:
  - Formal basis.
  - Proofs based on rigorous methods.
- Carefully prove that programs will behave as expected.
- For **every** single program?

It's too difficult for humans to do!

- Mechanization, automation
- Computer-assisted software development
  - Correctness by construction
  - Automatic analysis
  - Verification (model checking, deductive verification)
  - Automated testing

  ...to ensure relevant properties hold.
- Many properties generic (e.g., termination, if necessary).
- Others specific (e.g., what some program is expected to do).
- Difficult!

“Simple” properties and “simple” code

- How easy is it to decide whether a program terminates or not?

```
input n;

while n > 1 do
    if n mod 2 = 0 then
        n:= n / 2
    else
        n:= 3*n + 1
    end if
end while
```

- Will it finish for **any** input value `n`?

“Simple” properties and “simple” code

- How easy is it to decide whether a program terminates or not?

```
input n;

while n > 1 do
    if n mod 2 = 0 then
        n:= n / 2
    else
        n:= 3*n + 1
    end if
end while
```

- Will it finish for **any** input value `n`?
- Sometimes we cannot prove a property because:
  - It is difficult to prove.
  - It is false.
  - It is undecidable.

A specification example

```
procedure WhatDoIDo(A: Array)
  repeat
    swapped := false
    for i := 1 to length(A) - 1 do
      if A[i-1] > A[i] then
        swap(A[i-1], A[i])
        swapped := true
      end if
    end for
  until not swapped
end procedure
```

- What does this program do?

A specification example

```
procedure WhatDoIDo(A: Array)
   repeat
     swapped := false
     for i := 1 to length(A) - 1 do
       if A[i-1] > A[i] then
         swap(A[i-1], A[i])
         swapped := true
       end if
     end for
   until not swapped
end procedure
```

- What does this program do?
- Can you specify (using FOL) the property that characterizes a sorted array?

A specification example

```
procedure WhatDoIDo(A: Array)
   repeat
     swapped := false
     for i := 1 to length(A) - 1 do
       if A[i-1] > A[i] then
         swap(A[i-1], A[i])
         swapped := true
       end if
     end for
   until not swapped
end procedure
```

- What does this program do?
- Can you specify (using FOL) the property that characterizes a sorted array?
- Can we prove that, after executing the code above, array A is sorted?

A specification example

```
procedure WhatDoIDo(A: Array)
  repeat
    swapped := false
    for i := 1 to length(A) - 1 do
      if A[i-1] > A[i] then
        swap(A[i-1], A[i])
        swapped := true
      end if
    end for
  until not swapped
end procedure
```

- What does this program do?
- Can you specify (using FOL) the property that characterizes a sorted array?
- Can we prove that, after executing the code above, array A is sorted?
- Can we use specifications to derive a correct sorting program?

Jean-Raymond Abrial.
Faultless systems: Yes we can!
IEEE Computer, 42(9):30–36, 2009.

Jean-Raymond Abrial.
Modeling in Event-B - System and Software Engineering.
Cambridge University Press, 2010.

Michael Huth and Mark Ryan.
Logic in Computer Science: Modelling and Reasoning About Systems.
Cambridge University Press, New York, NY, USA, 2004.

Lawrence C. Paulson.
Logic and Proof.
Lecture notes, U. of Cambridge, available at
https://www.cl.cam.ac.uk/teaching/2122/LogicProof/logic-notes.pdf, last acccessed on Feb 9, 2022.