Teaching VDM
&
Teaching Formal Methods

Ana Paiva

apaiva@fe.up.pt **www.fe.up.pt/~apaiva**

Agenda

Our semester is structured in 13 lecture

So,

This talk has 13 sections

1st Lecture

Convince your students

that

Formal Methods

are important

(three different ways you may use for this purpose)

Why formal methods? (Facts)

- The **main source of bugs** is in the requirements specification phase - ambiguous and incomplete

- Formal methods are **unambiguous** - During the formal specification phase, the engineer **rigorously** defines a system using a modeling language

- Formal methods differ from other specification systems by their **heavy emphasis** on provability and correctness

- Once the model has been specified and verified, it is implemented by converting the specification into code (some times **automatically**)

1st Lecture

Prove

Formal Methods

are important

Use your own dog food!

Software is **bad** (P1).

Software differs from physical systems in at least two ways (P2):

software is **discontinuous** (P3),

and software is **complex** (P4).

Software is complex (P4), and complexity results in design flaws (P5); therefore, software has design **flaws** (P6).

Design flaws must be handled (P7). The three ways to handle design flaws are testing, design diversity, and fault avoidance (P8).

[Holloway C. Michael. 1997. Why Engineers should Consider Formal Methods. Technical Report. NASA Langley Technical Report Server.]

Use your own dog food!

Because software is discontinuous (P3), testing is inadequate (P9).

Also, because software is discontinuous (P3), design diversity is inadequate (P10).

Because there are only three ways to handle design flaws (P8), and the other two are inadequate (P9, P10), fault avoidance must be used to handle design flaws (P11).

Because formal methods are the most rigorous fault avoidance method (P12), and the greater the rigor, the more promising the method (P13), formal methods are the most promising fault avoidance method (P14).

Use your own dog food!

Because software has design flaws (P6), and design flaws must be handled (P7), and fault avoidance methods must be used to handle design flaws (P11), and formal methods are the most promising of these methods (P14), **software engineers should use appropriate formal methods (P15)**.

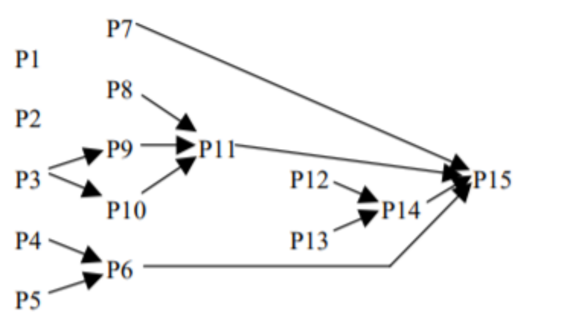


Figure 1: Structure of Typical Argument

Use your own dog food!

1st Lecture: Illustrate with examples

BASE DE DADOS FOI A MESMA

Nova empresa resolveu colocação de professores em seis dias

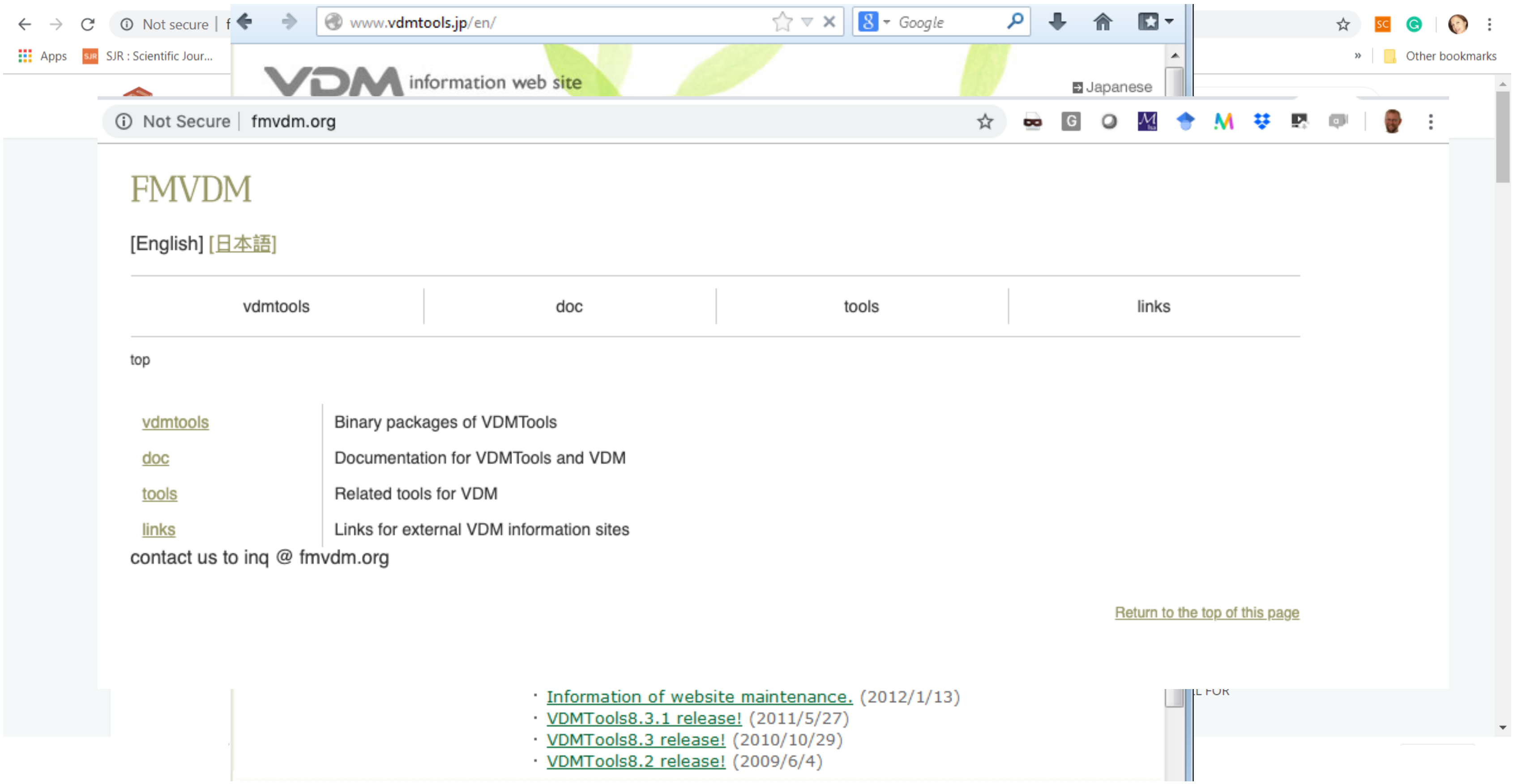
SOFIA RODRIGUES · 30 de Setembro de 2004, 9:35

0 PARTILHAS

(2004)

- The teacher placement problem was solved by a new computer solution designed in six days and executed in 30 minutes. The revelation was made by one of the five members of the ATX software team, an outside company hired by the ministry of education to "unlock" the unskilled teacher placement program created by Compta.
- From the same ministry database that contains all the faculty to be posted, ATX software has created a new algorithm, a computer solution, "thought out in full for six days and **based on very solid mathematical principles**," he said yesterday. computer engineer and author of the solution, Luis Andrade, during a press conference in Lisbon

Formal Methods Europe



2nd Lecture

Now that you have convinced them…

Start with the

Basics

Hoare Logic

- Hoare Logic forms the basis of all deductive verification techniques

- Named after **Tony Hoare**: inventor of **Quick Sort** (in 1960, when he was just 26), father of formal verification, 1980 Turing award winner

- Logic is also known as Floyd-Hoare logic: some ideas introduced by **Robert Floyd** in 1967 paper "Assigning Meaning to Programs"

Charles Antony Richard (Tony) Hoare

- A quote:
  - Computer programming is an **exact science** in that all the properties of a program and all the consequences of executing it in any given environment can, in principle, be found out from the text of the program itself by means of **purely deductive reasoning.**

Hoare Triple

$\{P\}\ S\ \{Q\}$ **Partial correctness**

or

$[P]\ S\ [Q]$ **Total correctness**

3rd Lecture

Let's have

some

fun

Let's start the fun!

- You may know everything and want to **prove** it

$$\{P\}\ S\ \{Q\}$$

- You may not know everything and want to **find** it

$$\{?\}\ S\ \{Q\}$$

$$\{P\}\ S\ \{?\}$$

$$\{P\}\ ?\ \{Q\}$$

Let's start the fun!

Hoare Triple	Question	Technique
{P} S {Q}	S satisfies specification?	Inference Rules
{?} S {Q}	Which the precondition?	Weakest precondition
{P} S {?}	Which is the program?	Strongest post condition
{P} S {Q}	Which is the post condition?	Refinement

4th Lecture

Hopefully, at this class your students ask:

How can we do that?

4^{th} Lecture: How can we do that?

Nº	Instruction	Inference Rules
R1	**skip**	**{P} skip {P}**
R2	**Assignment**	**{P[E/x]} x := E {P(x)}**
R3	**Sequence or Composition**	**{P} S {Q} , {Q} T {R}** **{P} S; T {R}**
R4	**If**	**{P $\wedge$ C} S {Q} , {P $\wedge \neg$C} T {Q}** **{P} if C then S else T {Q}**
R5	**Cycle**	**I $\wedge$ C $\Rightarrow$ v$\in$N, {I $\wedge$ C $\wedge$ v=V} S {I $\wedge$ v<V}** **{I} while C do S {I $\wedge \neg$C}**
R6	Strengthening the precondition	**P'$\Rightarrow$ P, {P} S {Q}** **{P'} S {Q}**
R7	Weakening the postcondition	**{P} S {Q}, Q $\Rightarrow$ Q'** **{P} S {Q'}**
R8	Intermediate assertions	**{P$\wedge$A} assert A {P}**

…or Weakest precondition rules

4^{th} Lecture: How can we do that?

Nº	Name	Refinement Rules
1	Strengthen Post-condition Rule	If $Q' \Rightarrow Q$ then $Spec(P, S, Q) \sqsubseteq Spec(P, S, Q')$
2	Weaken Pre-condition Rule	If $P \Rightarrow P'$, then $Spec(P,S,Q) \sqsubseteq Spec(P',S,Q)$
3	Skip Rule	If $P \Rightarrow Q$ then $Spec(P,S,Q) \sqsubseteq Spec(P, skip, Q)$
4	Assignment Rule	if $P \Rightarrow Q[E/x]$ then $Spec(P, x{:}S, Q) \sqsubseteq Spec(P, x{:=}E, Q]$
5	Composition Rule	$\{P\}\ S\ \{Q\} \sqsubseteq \{P\}\ S_1\ \{M\};\ S_2\ \{Q\}$
6	Following Assignment Rule	$\{P\}S\{Q\} \sqsubseteq \{P\}\ S_1\ \{Q[E/x]\};\ x{:=}\ E\{Q\}$
7	Selection Rule	If $P \Rightarrow G_1 \vee G_2 \vee \ldots \vee G_n$, then $\{P\}\ S\ \{Q\} \sqsubseteq \{P\}$ if $G_1 \rightarrow \{G_1 \wedge P\}\ S_1\ \{Q\}$ [] $G_2 \rightarrow \{G_2 \wedge P\}\ S_2\ \{Q\}$ [] … [] $G_n \rightarrow \{G_n \wedge P\}\ S_n\ \{Q\}$ fi $\{Q\}$
8	Repetition Rule	Suppose $G = G_1 \vee G_2 \vee \ldots \vee G_n$, then $\{I\}\ S\ \{I \wedge \neg G\} \sqsubseteq \{I\}\ DO\ \{I \wedge \neg G\}$ where DO is do $G_1 \rightarrow \{I \wedge G_1 \wedge V = V_0\}\ S_1\ \{I \wedge (0 \leq V < V_0)\}$ [] … [] $G_n \rightarrow \{I \wedge G_n \wedge V = V_0\}\ S_n\ \{I \wedge (0 \leq V < V_0)\}$ od

5^{th} to 7^{th} Lecture

Exercises

Apply different rules

in

small examples

At this point… The students think…

- Oh, Ok….
- This works just for small toy examples

- It does not scale...

8^{th} Lecture

Exercises

Use the overall approach

in more

real examples

Challenges

- Students have other courses in parallel
- Find real complex examples
- Motivate the students
- Find good tools
- Make them conscious of Formal Methods advantages
- …

There no way to avoid that,
so,
balance the effort

Real, not too complex and fun

Overture?

Make them gather metrics
to be conscious
of the benefits

Real & not too complex

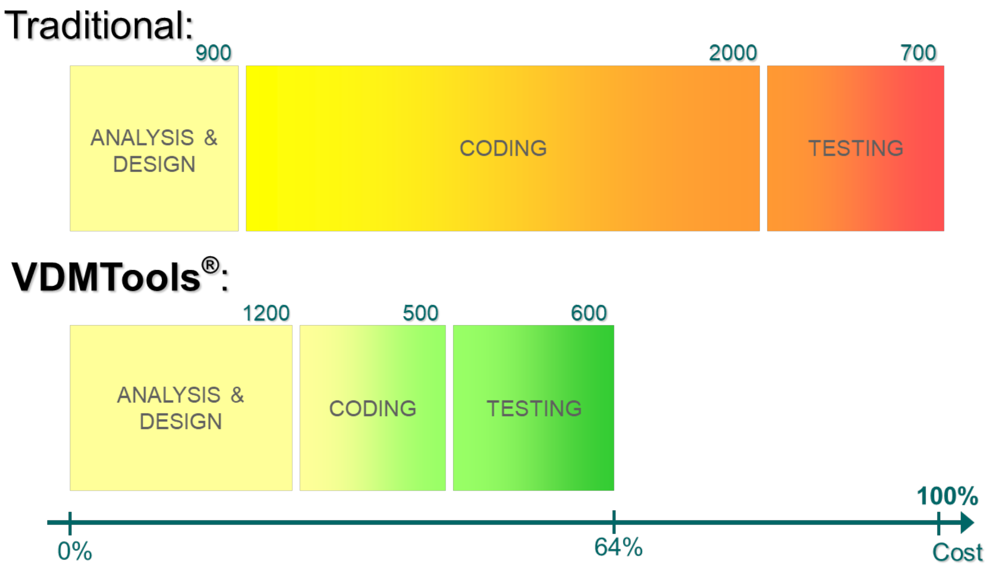
- I Already tried several different themes:
  - **Information systems**: system to manage the information of an University; System to manage a football championship, etc.
  - **Board games**: monopoly; chess, go, abalone, etc…
    - This worked quite well because they had to specify the rules and at the end they could play...
    - However I didn't ask for a GUI and an API interface to play a game is not very attractive…
  - So,
  - I combined the Formal Methods course with the Computer Graphics course so the students could have a graphical user interface connected with a background developed in VDM++

Good Tools

- Alloy Analyzer (for Alloy)
- Dafny (to prove program's correctness. Good thing: it is on the web)
- VDM tools (to illustrate the end-to-end process)
- Overture (to illustrate the end-to-end process)
  - Main problems
    - Code generator – several problems
    - Students need to look into the generated code to fix the problems and such code is not easy to read.

Conscious of the advantages

- Gather metrics
  - How much time did you spend with the specification?
  - How much time did you spend in testing the specification?
  - How much time did you spend generating the code?
  - How much time did you spend testing the code?



Conscious of the advantages

- Gather metrics
  - How confident are you about the quality of the work you are presenting to the teacher?
    - This is the best software I have developed
    - Teacher, please try the software as you want...

- At the end

- Make them think…
  - What do you think about this process?
  - What do you think about specifying contracts (pre and post conditions)?
  - Now, do you think you are a better developer?

- Usually they say YES

9th Lecture

How did I implement

this along the years?

Along the years…

2006-2009	Introduction	**VDM**	Model Checking	Alloy	Algebraic specifications	Proofs (Hoare)
2009-2014	Introduction	Alloy	Model Checking	**VDM**	Proofs (Hoare)	
2015-2016	Introduction	Proofs (Hoare)	Model Checking	Alloy	**VDM**	
2016-2019	Introduction	Proofs (Hoare)	Proofs (Refinement)	Alloy	**VDM**	

VDM moved to the end of the semester

10^{th} Lecture

What

I have

learned

- Some students love it

- Some students hate it

- But, you should never give up…

- And also, …
- Yes, you should move the end-to-end process (VDM/Overture) to the end of the semester.

- You don't even need to teach them VDM!
- They have all the necessary knowledge to use it well!
- And it works very well. They use the method and the tools and they know why and how to use them.

11^{th} – 12^{th} Lectures

Warp up

Ten Principles to Teach Formal Methods

- **Principle 1**: The field of Formal Methods is too large to gain encyclopedic knowledge – choose representatives
  - Proofs (Hoare and Refinement Rules), Alloy, VDM
- **Principle 2**: Formal Methods are more than pure/poor Mathematics – focus on Engineering
  - Formal Methods **can be used with traditional as well as agile models**. Moreover, Formal Methods should not constitute separate phases or sprints, but should be rather integrated as part of the general validation activities. Thus, teaching Formal Methods should frequently resort to other topics in Software Engineering.

[“Teaching Formal Methods for Software Engineering – Ten Principles”, Antonio Cerone, Markus Roggenbach, Bernd-Holger Schlingloff, Gerardo Schneider, and Siraj Ahmed Shaikh]

Ten Principles to Teach Formal Methods

- **Principle 3**: Formal Methods need **tools** – make them available
  - Tools for simulation of behavior and visualization of state space or traces are essential to allow students to understand the behavior associated with their models: Dafny; Alloy Analyzer; Overture
- **Principle 4:** Modelling versus programming – work out the differences
  - Models of software systems are different from programming code as **programs are executable, while models can be executable**. A model is a purposeful abstraction of either an existing system or a system still to be built.

Ten Principles to Teach Formal Methods

- **Principle 5:** Tools teach the method – use them
  - Instead of tediously going through the semantics of each construct in a formal language, **allow the students to experiment** with an appropriate tool **to discover** the semantics for themselves
- **Principle 6:** Formal Methods need lab classes – create a stable platform
  - Labs can offer **hands-on** experience with Formal Methods tools and practical examples. Such teaching style appeals to the plug-and-play mindset of a student generation who loves to play with gadgets of all kinds

Ten Principles to Teach Formal Methods

- **Principle 7:** Formal Methods are best taught by examples – choose from a domain familiar to the target group
  - …a number of "**logic puzzles**" which have been popular since the middle of the 19th century. One can formalize and solve such puzzles…
  - e.g., Hanoi Tower Problem
- **Principle 8:** Each Formal Method consists of **syntax, semantics and algorithms - focus uniformly on these key ingredients**
  - A formal language is described by an unambiguous syntax and a Mathematical semantics. For a Formal Method (as opposed to a formal language) it is essential that there are some algorithms or procedures which describe what can be done with the syntactic objects in practice…

Ten Principles to Teach Formal Methods

- **Principle 9:** Formal Methods have several dimensions – use a **taxonomy**
  - In order to give students an orientation, it is important to provide a taxonomy. Formal Methods can, e.g., be **categorized according to the following dimensions: Application range, Underlying technology, Properties under concern, Usability**
- **Principle 10:** Formal Methods are fun – shout it out loud!
  - Psychology tells us that the human **learning capacity is highest when we enjoy** what we are doing.
  - **A strong motivator are also competitions**. There exist several competitions in the Formal Methods community, e.g., the VerifyThis Verification Competition, the Hardware Model Checking Competition, or the SAT competition.

13th Lecture

Students should know

“The sooner you start to code, the longer the program will take”

Think in advance! Specify! Use formal methods.

Teachers should know

“Learning is remembering what you are interested in”

Motivate the students! Make it fun! Use known and fun examples.

Teaching VDM
&
Teaching Formal Methods

?

Ana Paiva

apaiva@fe.up.pt **www.fe.up.pt/~apaiva**