



Debugging Auto-Generated Code with Source Specification in Exploratory Modeling

Tomohiro Oda
Keijiro Araki
Peter Gorm Larsen



Agenda

- Exploratory Modeling and ViennaTalk
- Automated Code Generator as Animation Engine
- Challenges in Debugging VDM Specification on Auto-Generated Code
- Design : Traceability
- Demo
- Summary

Exploratory Modeling

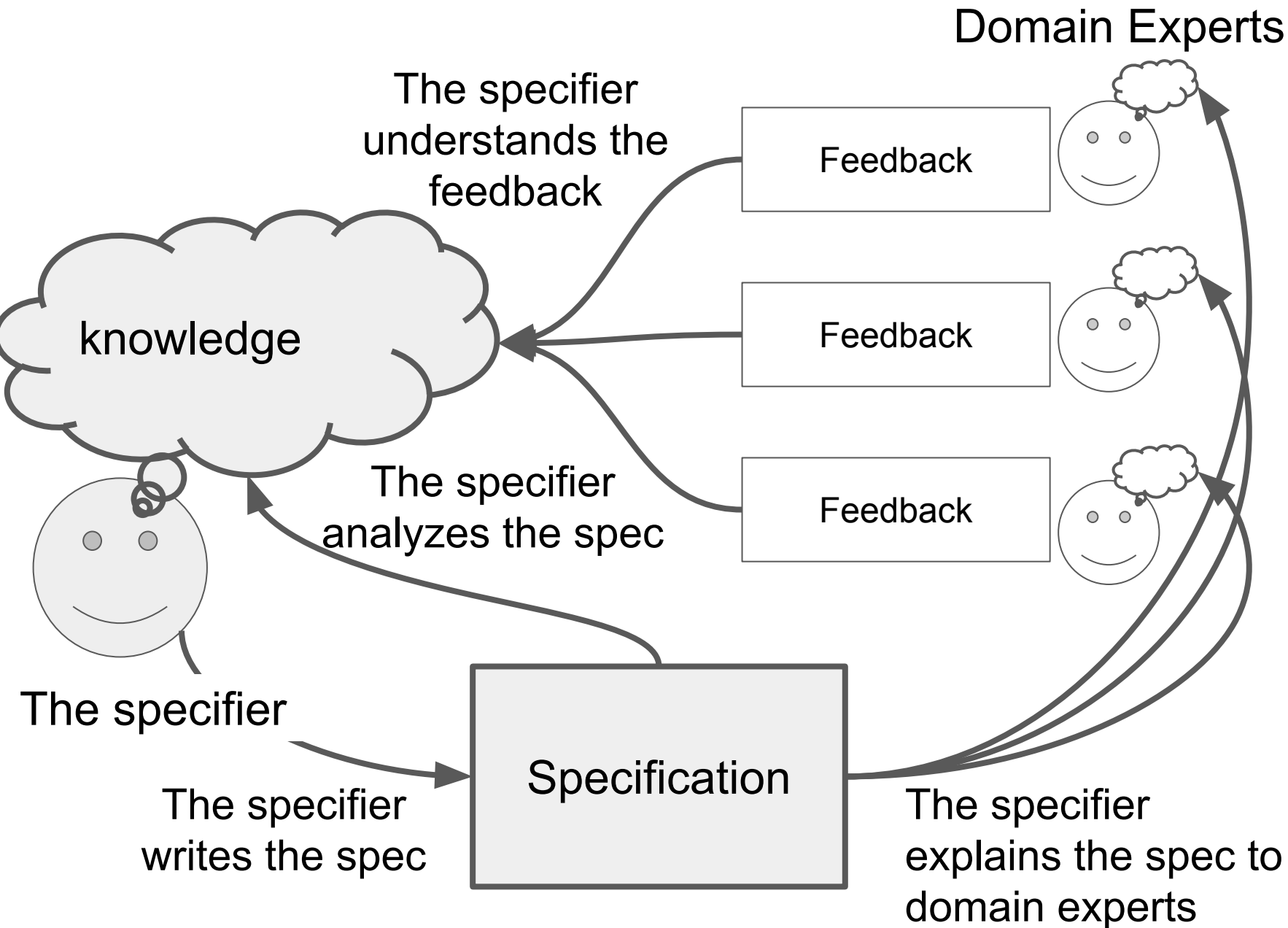
Exploratory modeling is to produce a specification, which is

- valid,
- feasible and
- full-featured,

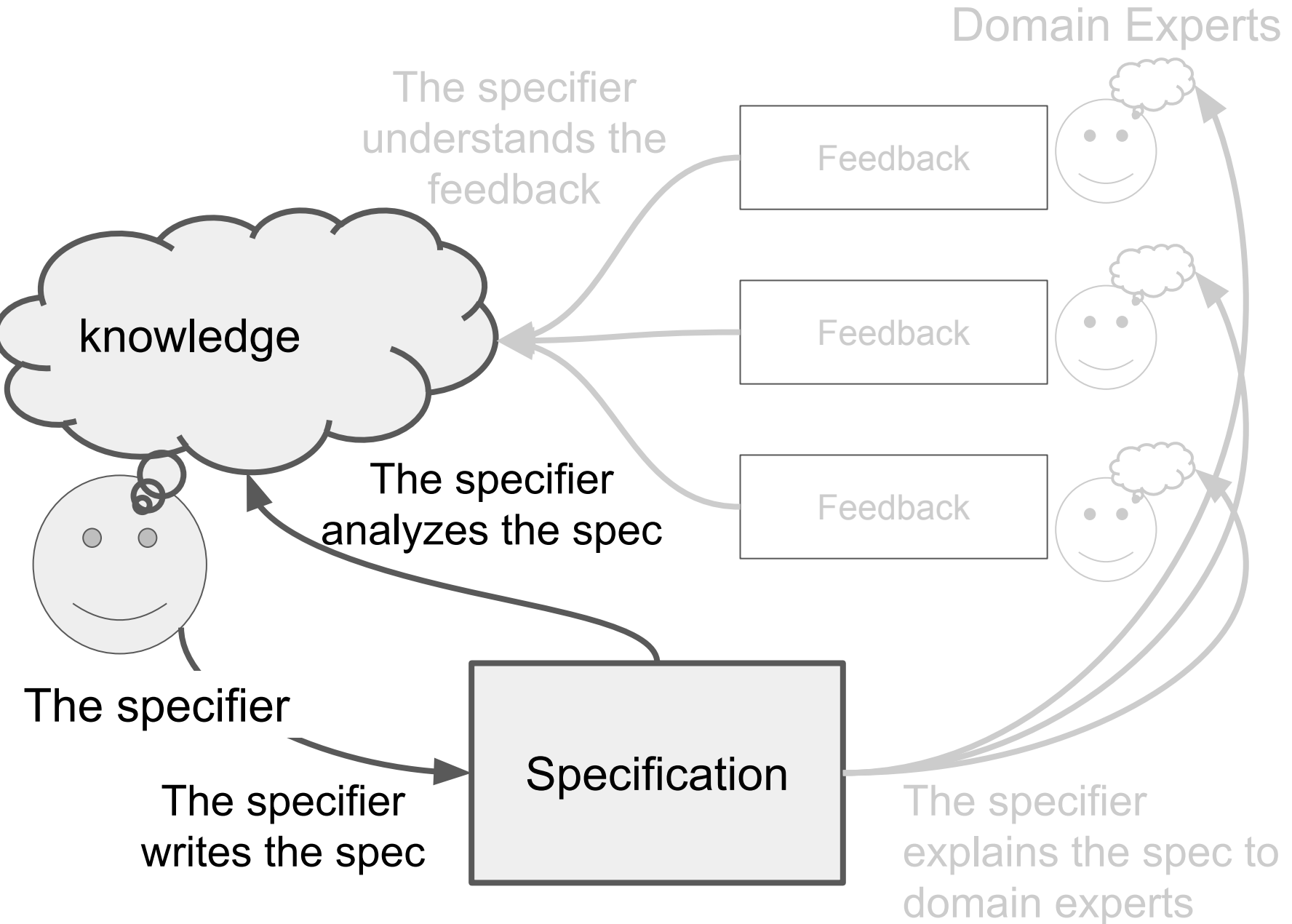
followed by rigorous specification, which is

- totally defined,
- sound,
- verifiable and
- maintainable.

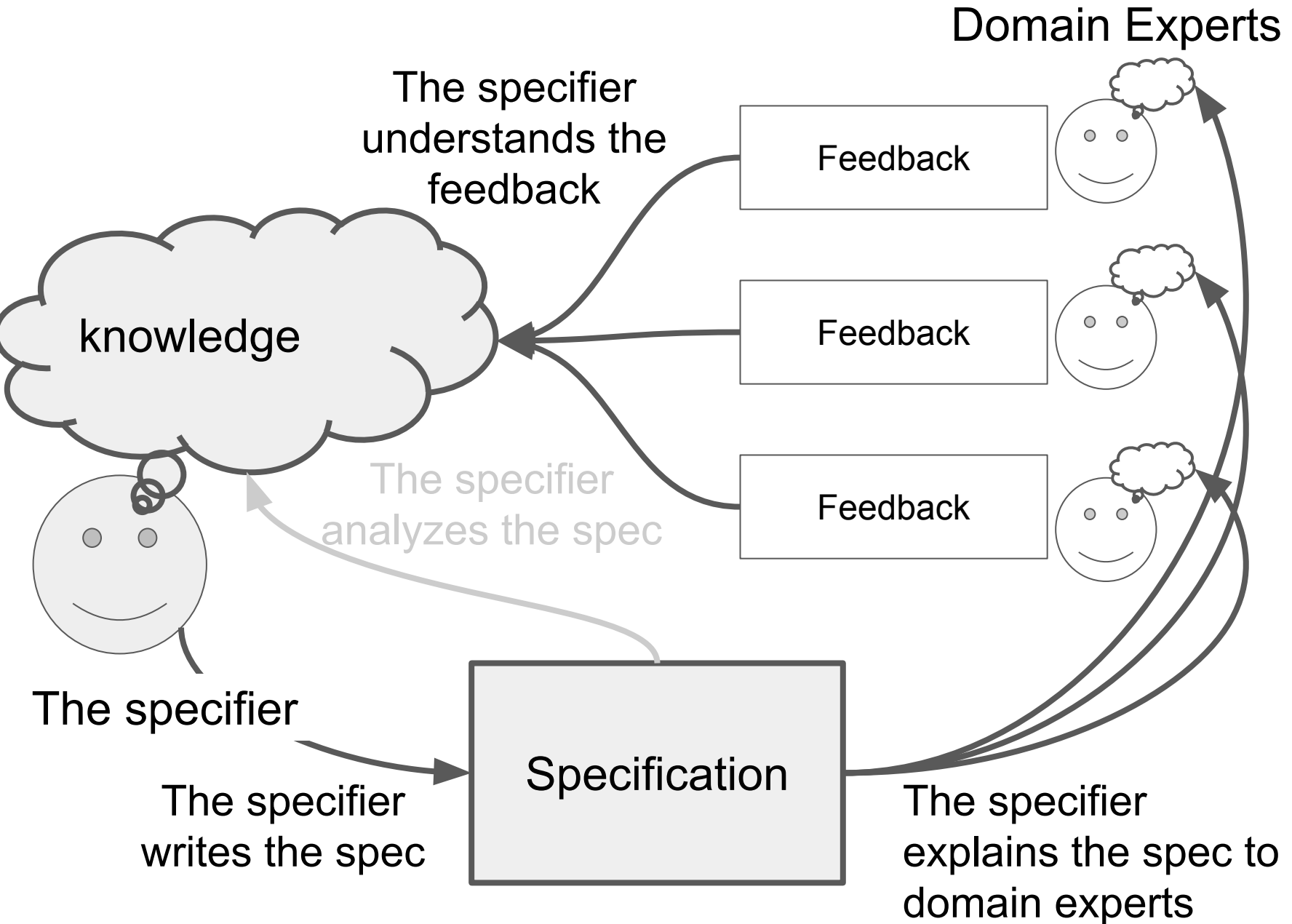
Workflow of Exploratory Modeling



The Cycle of Specifying : Individual's task



The Cycle of Learning : Collaborative task



Requirement of Code Generator as Animation Engine for Exploratory Modeling

- **Performance**
 - Tweak free : No need for "tuning" the spec
 - Feasibility : Closer to the production code
- **Interactivity**
 - Liveness : Fixing spec in action
 - UI : non-formalist friendly
 - Connectivity : networking, legacy libraries
- **Debuggability**
 - Finding : To be aware of unexpected behaviour
 - Locating : To spot the cause of the behaviour
 - Modifying : To fix the spec if necessary
 - Testing : To ensure the spec means as intended

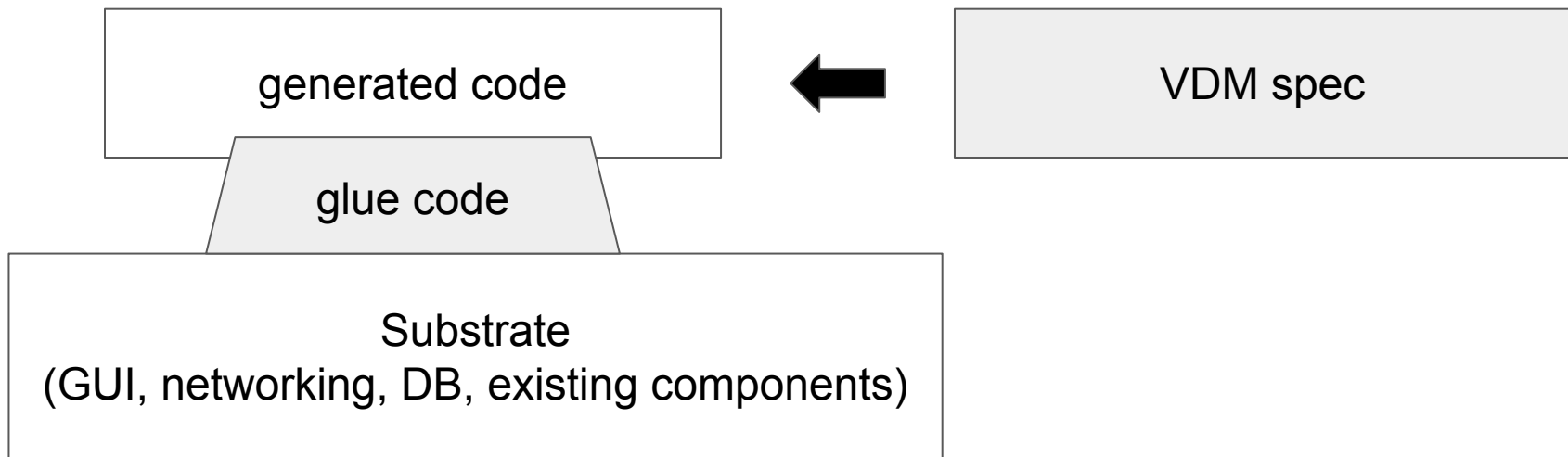
Challenges of Code Generator as Animation Engine for Exploratory Modeling

- Performance
 - Tweak free : No need for "tuning" the spec
 - Feasibility : Closer to the production code
- Interactivity
 - Liveness : Fixing spec in action
 - UI : non-formalist friendly
 - Connectivity : networking, existing components
- **Debuggability**
 - Finding : To be aware of unexpected behaviour
 - **Locating : To spot the cause of the behaviour**
 - **Modifying : To fix the spec if necessary**
 - Testing : To ensure the spec means as intended

challenges of debugging auto-generated code

Artifacts to observe

Artifacts to fix



1. Finding an issue
2. Locating the cause



2. Locating the cause
3. Modifying the spec

4. Testing the new code

VDMDebugger

Smalltalk steps

VDM steps

call stack

Smalltalk code

VDM spec

variables

The screenshot displays the VDMDebugger interface with several key components:

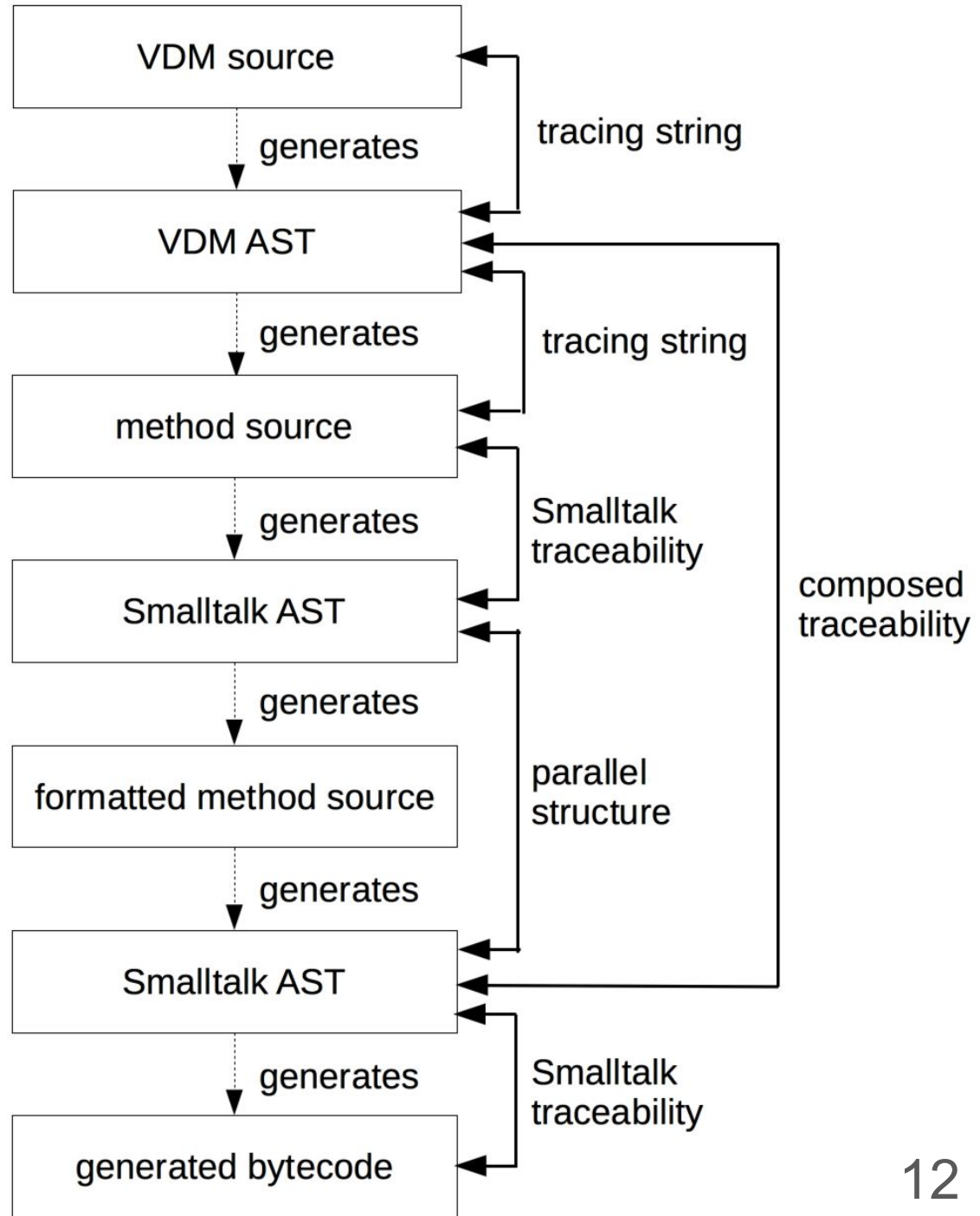
- Stack:** A list of frames showing the current execution context. The top frame is labeled 'inc:'.
- Source:** A window showing Smalltalk code for the `inc` method. The code includes type annotations and error handling. A callout points to the `count` variable in the code.
- VDM spec:** A window showing the VDM specification for the `inc` method, including preconditions, postconditions, and the method body.
- Variables:** A table at the bottom showing the current state of variables.

Type	Variable	Value
implicit	self	a Counter
attribute	count	compose Counter of count and end

demo

traceability

from each bytecode
to substring of the spec



Summary

Done:

- Bytecode to VDM source traceability
- Step execution in granularity of VDM and Smalltalk

Todo:

- Live modification to VDM source on VDMDebugger
- VDMPad-like diagram presentation of VDM values