Accelerating Software Development

in Digital Therapeutics

Patrick Alff
Patrick.alff@voluntis.com

VOLUNTIS

voelter@acm.org
www.voelter.de
@markusvoelter **Markus Völter**

itemis voelter { ingenieurbüro für softwaretechnologie //

1

Software Medical Devices

and

Digital Therapeutics

SaMD Defined

Software intended to be used for one or more medical purposes that perform these purposes without being part of a hardware medical device.

International Medical Device Regulators Forum

Can run on general purpose computing platforms (PC, Smartphone).

For software to be an SaMD, it cannot just be an add-on or ”driver” to a hardware medical device.

An SaMD remains an SaMD even when used in combination with other products including medical devices.

Can have an interface to the outside world: to other medical devices, including hardware medical devices and other SaMD software, as well as general purpose software.

Mobile apps that meet the definition above are considered SaMD

DTx vs. SaMD?

SaMD is standalone medical software used for three different purposes:

Diagnostics
identification of an illness or examination of symptoms software

Software Accessory to a physical medical devices
i.e., a control app for an insulin pump

Therapeutics (DTx)
diabetes management, cancer treatments symptom management, substance abuse, PTSD, schizophrenia, respiratory & cardiac diseases

A DTx is an SaMD used for therapeutic purposes.

Three Classes

Class I

Stethoscope

Active side-effect management App

- Low risk devices: Do not support or sustain human life
- Relatively simple design
- QMS: 21 CFR Part 820
- Most are exempt of 510(k) clearance

Class II

X-Ray Machine

Insulin dose calculation App

- Medium risk to patient
- More complex in design
- QMS: 21 CFR Part 820 + Design Controls
- Most require 510(k) clearance

Class III

Heart Valve

Pacemaker

- Supports or sustains life, or high risk of injury
- Typically clinical trial to demonstrate efficacy and safety
- QMS: 21 CFR Part 820 + Design Controls + FDA inspection
- Usually requires a PMA (pre-market approval)

Digital Therapeutics Defined

DIGITAL THERAPEUTICS ALLIANCE

Digital Therapeutics (DTx) deliver evidence-based therapeutic interventions to patients that are driven by high quality software programs to prevent, manage, or treat a medical disorder or disease.

Used independently or in concert with **medications**, devices, or other therapies to optimize patient care and health outcomes.

Incorporate advanced **technology best practices** relating to design, clinical validation, usability, and data security.

Are validated by **regulatory bodies** as required to support product claims regarding risk, efficacy, and intended use.

Regulated vs. Unregulated

Digital Services	Adjunctive DTx	Drug replacement DTx
Support or improve patient habits or behavior May have evidence of clinical benefit	Adds value or improves clinical benefit of a drug	Software bears direct clinical benefit as standalone treatment or combined with drug
Not regulated (FDA/CE)	**Requires FDA clearance / CE-Mark**	**Approved and Regulated by the FDA / CE-Mark**

omada

Roche + mySugr

welldoc

VOLUNTIS

proteus DIGITAL HEALTH

PEAR THERAPEUTICS

AKILI LABS

Propeller

DIGITAL THERAPEUTICS ALLIANCE

DTx Market and Adoption

*The application of digital technology in mental health, in heart disease, and in many cancers **will revolutionize the industry.***

Eric Schmidt
Executive Chair of Google's parent Alphabet

Market analysts report between **21.0% and 30.7% CAGR** of the DTx industry, depending on method, geography and segments.

4.5 million patients are using DTx, and this will increase to 130 million by 2023. *)

73% of healthcare professionals believe that DTx will help patients with chronic diseases, such as diabetes or heart disease. ***)*

Over **150 firms** globally are now developing some form of DTx

*) Juniper Research study from 2018
**) Research Now 2015 Survey in US and UK

2

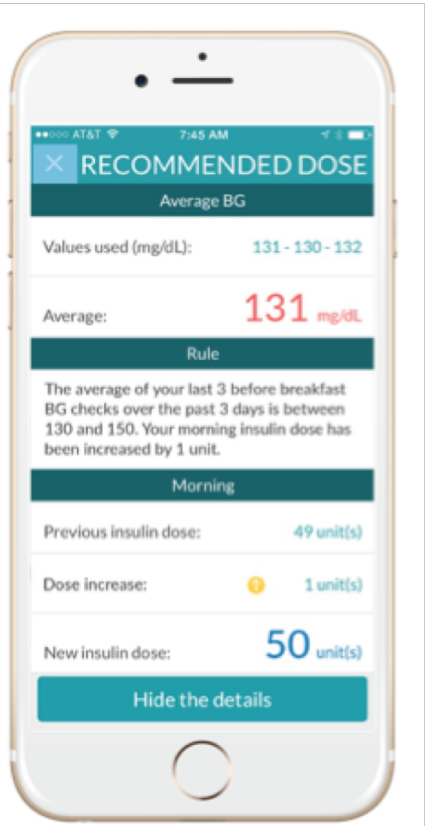
Voluntis‘ DTx Apps

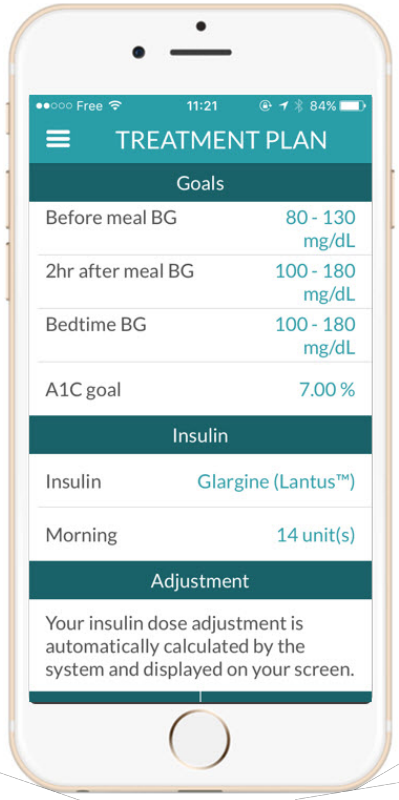
Context

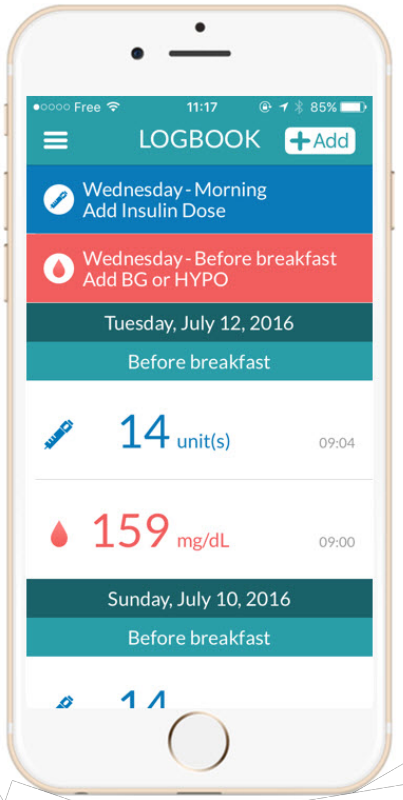
Mobile Apps that help patients w/ treatments

Monitor side-effects and recommend actions

Manage dosage of medications







VOLUNTIS
Connected Therapeutics

Context

Mobile Apps that help patients w/ treatments

Monitor side-effects and recommend actions

Manage dosage of medications

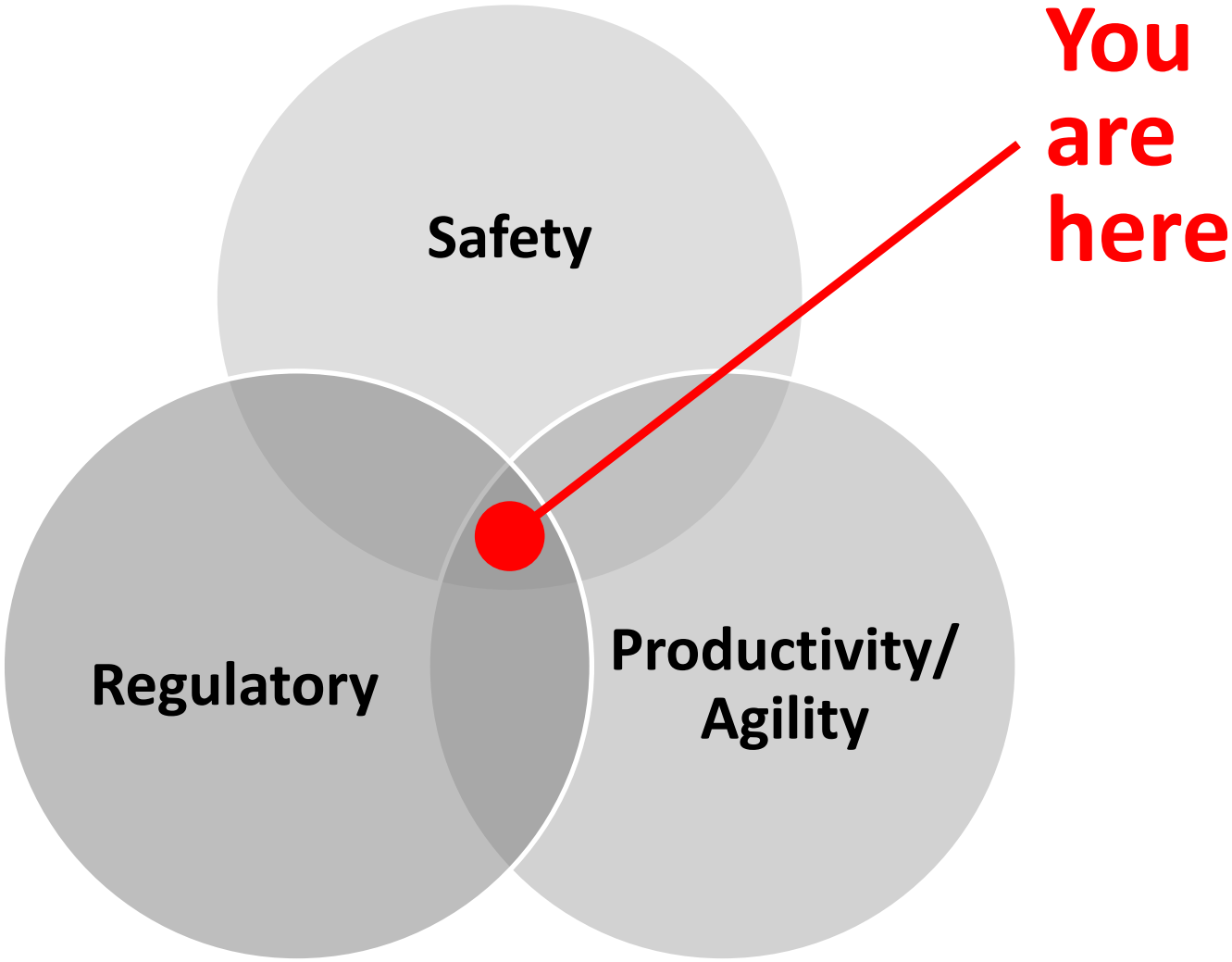
“Algorithms“ for recommendations and dosage at the core of these apps.

Safety-critical, since they could hurt patients.

Customer develops many different apps/algos like this, efficiency of algo development is key.

VOLUNTIS
Connected Therapeutics

Context



Why use DSLs in DTx

Improved Quality and Safety

Improved **collaboration** between software engineers and medical team.

Streamlined **validation** b/c of more understandable representation of the algorithms.

“Live” models enable **executing** the algorithms as part of the validation.

More trust in **tests** b/c of simplified review and increased (semantic) coverge.

Fewer implementation **errors** through increased automation.

Accelerated Go-to-Market

Iterating algorithm designs faster and earlier in the SDLC.

Parallel development medical algorithm and the rest of the application/infrastructure.

Reduced uncertainty during implementation phase because of immediate testing and simulation.

Generating required **documentation** automatically.

Accelerating **verification** and **validation** because of increased automation of tests.

Shortening algorithm **deployment time** because of interpreted execution – code is data!

Reduced R&D Cost

The **later** an error is found, the more **expensive** it is to fix.

Even more pronounced in healthcare!

Front-Loading the design and validation, avoid late-stage change requests when changes are expensive.

Expedite the inception, design and validation process with our medical boards.

Faster **turn around** times to modify and deploy medical algorithms once on the market.

Leverage pre-tested **components** in our SDLC.

4

Solution Approach

Solution Approach

Health care professionals directly „code“ algos, using a suitable language.

Avoids indirections through requirements docs.

Speed up dev significantly.

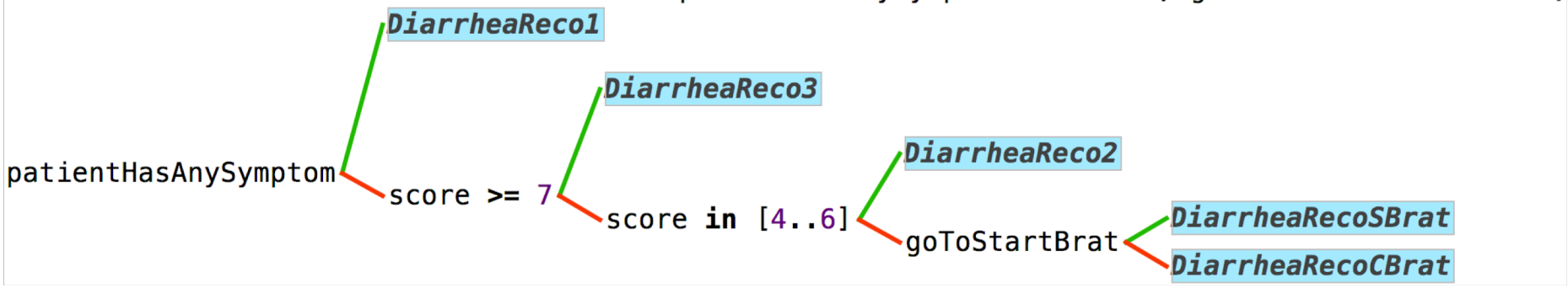
} Pretty typical DSL-based dev-approach.

Some Language Impressions I

```
decision table BpScoreDecisionTable(sys: bpRange, dia: bpRange) =
```

		dia					
		<= 50	[51..90]	[91..95]	[96..100]	[101..109]	>= 110
sys	<= 90	1	1	3	4	5	6
	[91..140]	2	2	3	4	5	6
	[141..150]	3	3	3	4	5	6
	[151..160]	4	4	4	4	5	6
	[161..179]	5	5	5	5	5	6
	>= 180	6	6	6	6	6	6

```
decision tree DiarrheaStoolsDecisionTree(score: DiarrheaStoolsOverBaseline,
                                         patientHasAnySymptom: boolean, goToStartBrat: boolean)
```




```
type temperature: number[36|42]{1}
type measuredTemp: number[35|43]{2}

Error: type number[32.55|39.99]{4} is not a subtype of number[36|42]{1}
val T_measured: measuredTemp = 42.22
val T_calibrated: temperature = T_measured * 0.93
```

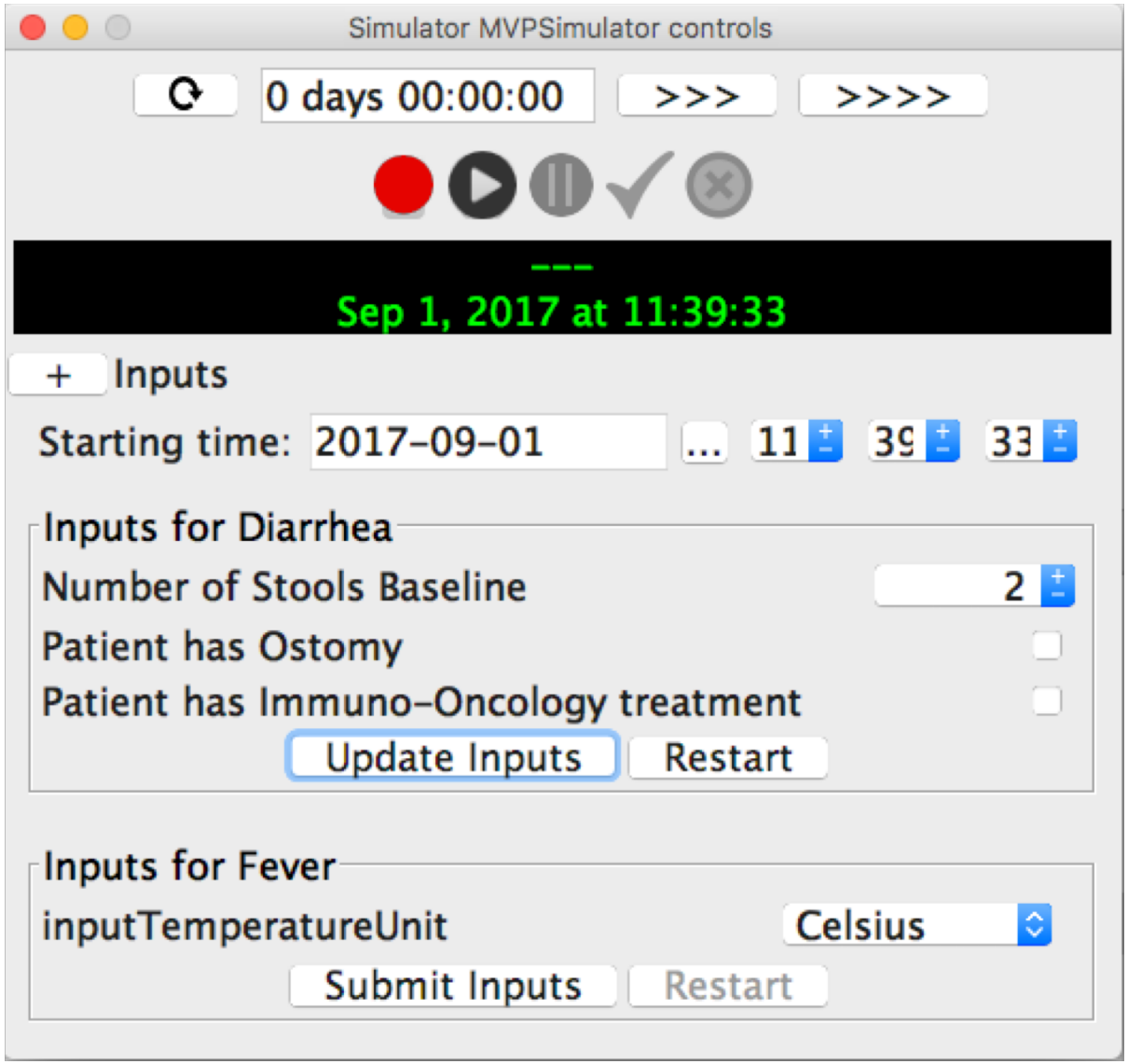
Some Language Impressions II

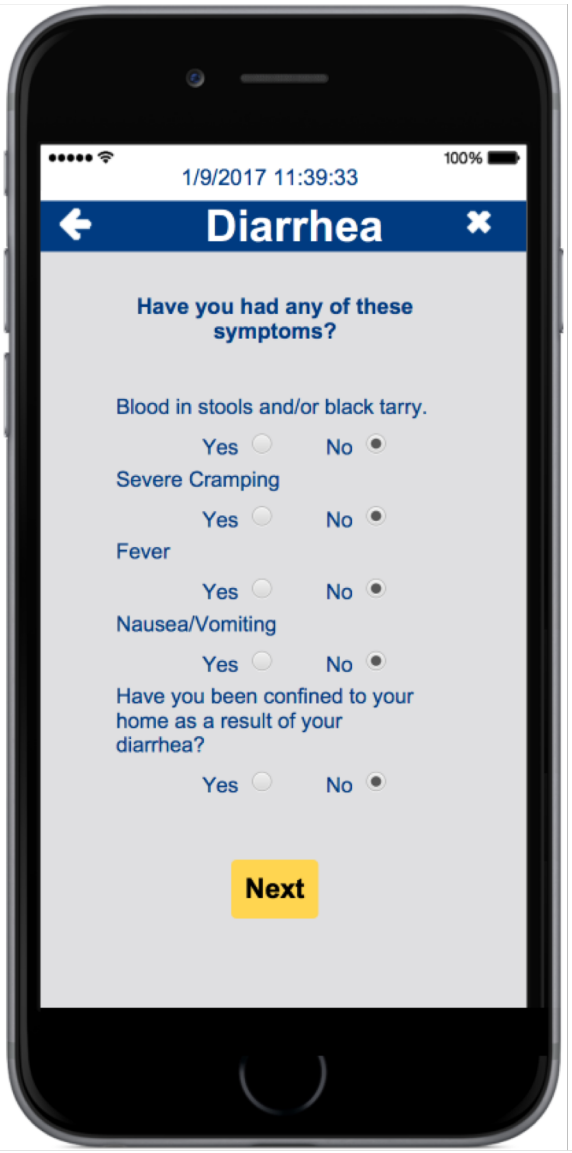
```
start: State_initialize
→A state State_initialize
  W on EventAskBP
    [true] ⇨ State_AskBG

←A state State_AskBG
  W on EventAddBGValue
    [true] ⇨ State_CheckHypoEvent

→A state State_CheckHypoEvent
    [hypoEvent] ⇨ State_AskReason
    [!hypoEvent] ⇨ State_StoreBGValue

state State_AskReason
  W on EventAskReason
    [true] ⇨ State_StoreBGValue

→A ←A state State_StoreBGValue
    [true] ⇨ State_AskBG
```

Some Language Impressions IV

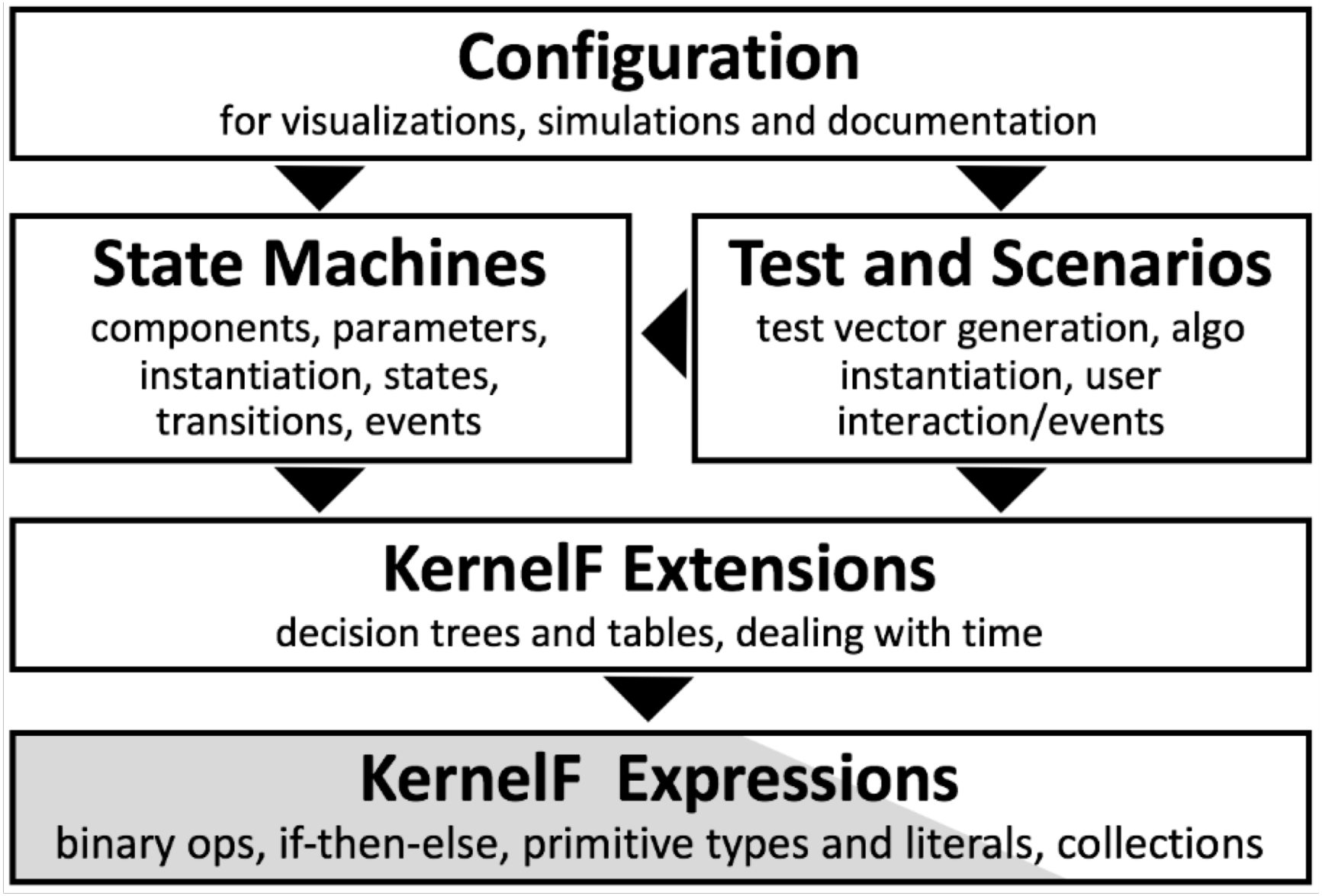
```
PASS
function test gradeStools
  given  7  expected  3
  given  6  expected  2
  given  5  expected  2
  given  4  expected  2

PASS
function test DiarrheaStoolsDecisionTree
  given  false, 1, true, false    expected   DiarrheaUSRecoLevel1Symptom
  given  false, 9, false, false   expected   DiarrheaUSRecoGrade3

PASS
function test checkScreeningQuestion
  given   answers to DiarrheaScreeningQuestionnaire{    expected  true
            dietarySupplements: false
            medication        : true
            hospitalized      : false
          }
```

Some Language Impressions V

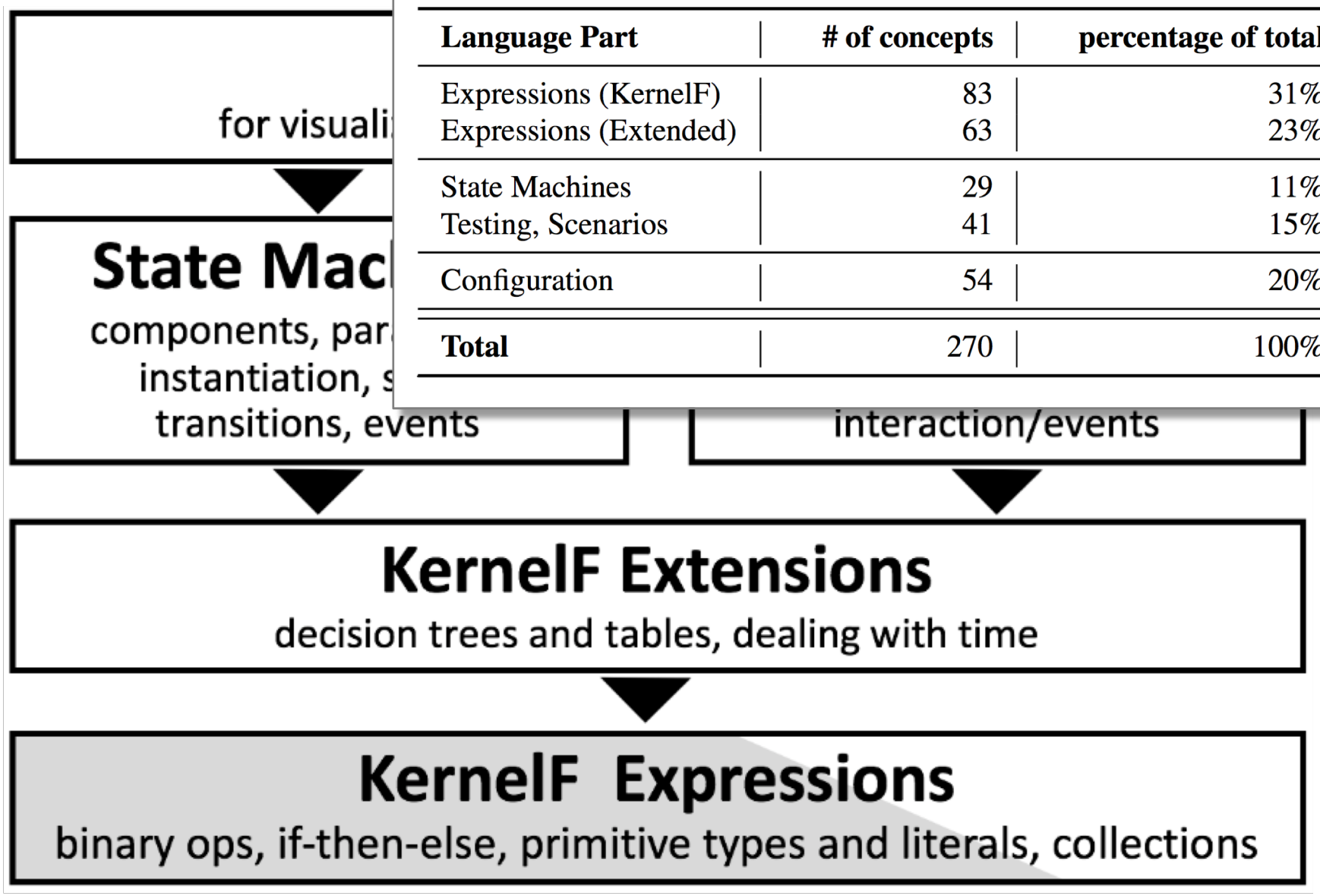




Languages Used



Languages Used



Language Part	# of concepts	percentage of total
Expressions (KernelF)	83	31%
Expressions (Extended)	63	23%
State Machines	29	11%
Testing, Scenarios	41	15%
Configuration	54	20%
Total	270	100%

Solution Approach

Health care professionals directly „code“ algos, using a suitable language.

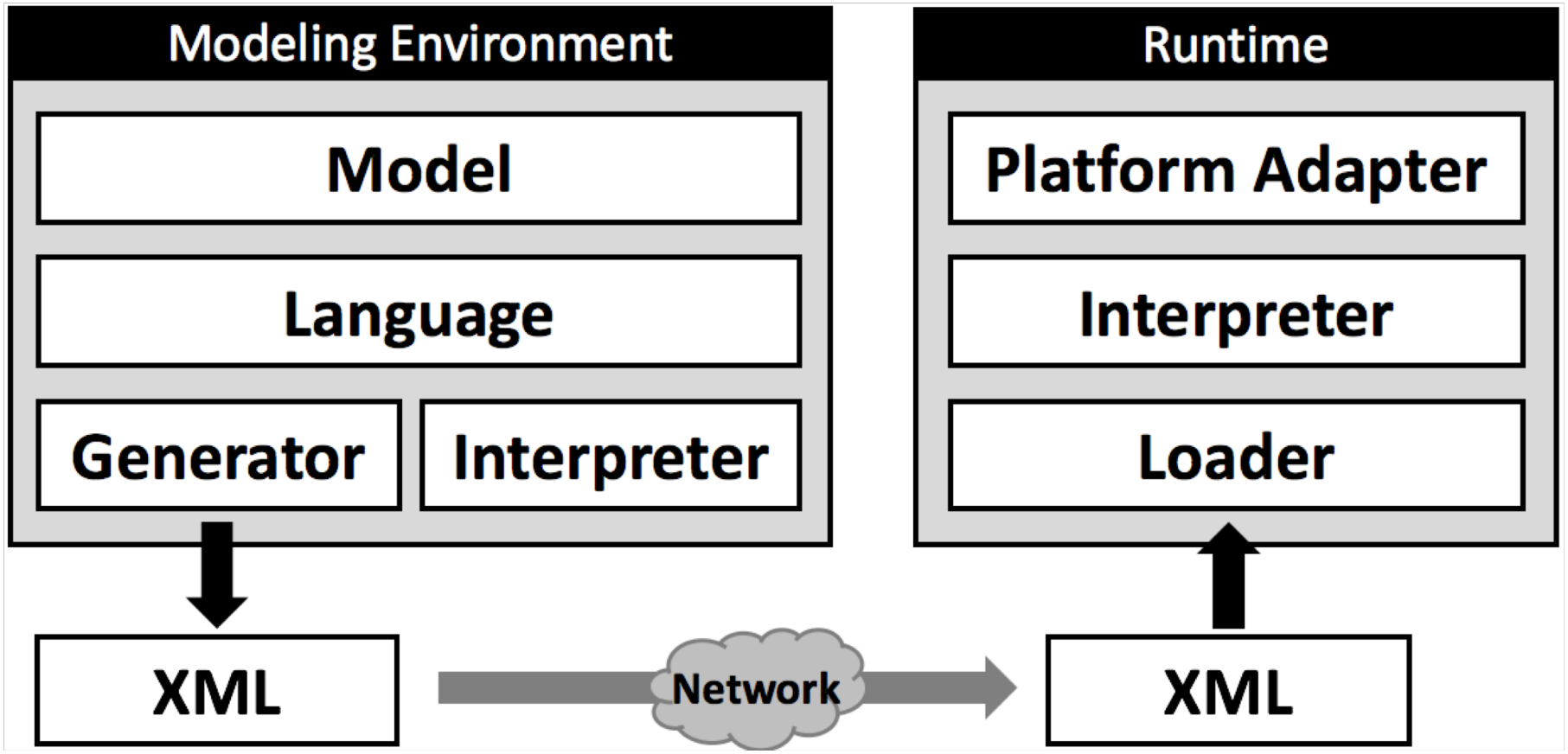
Avoids indirections through requirements docs.

Speed up dev significantly.

} Pretty typical DSL-based dev-approach.

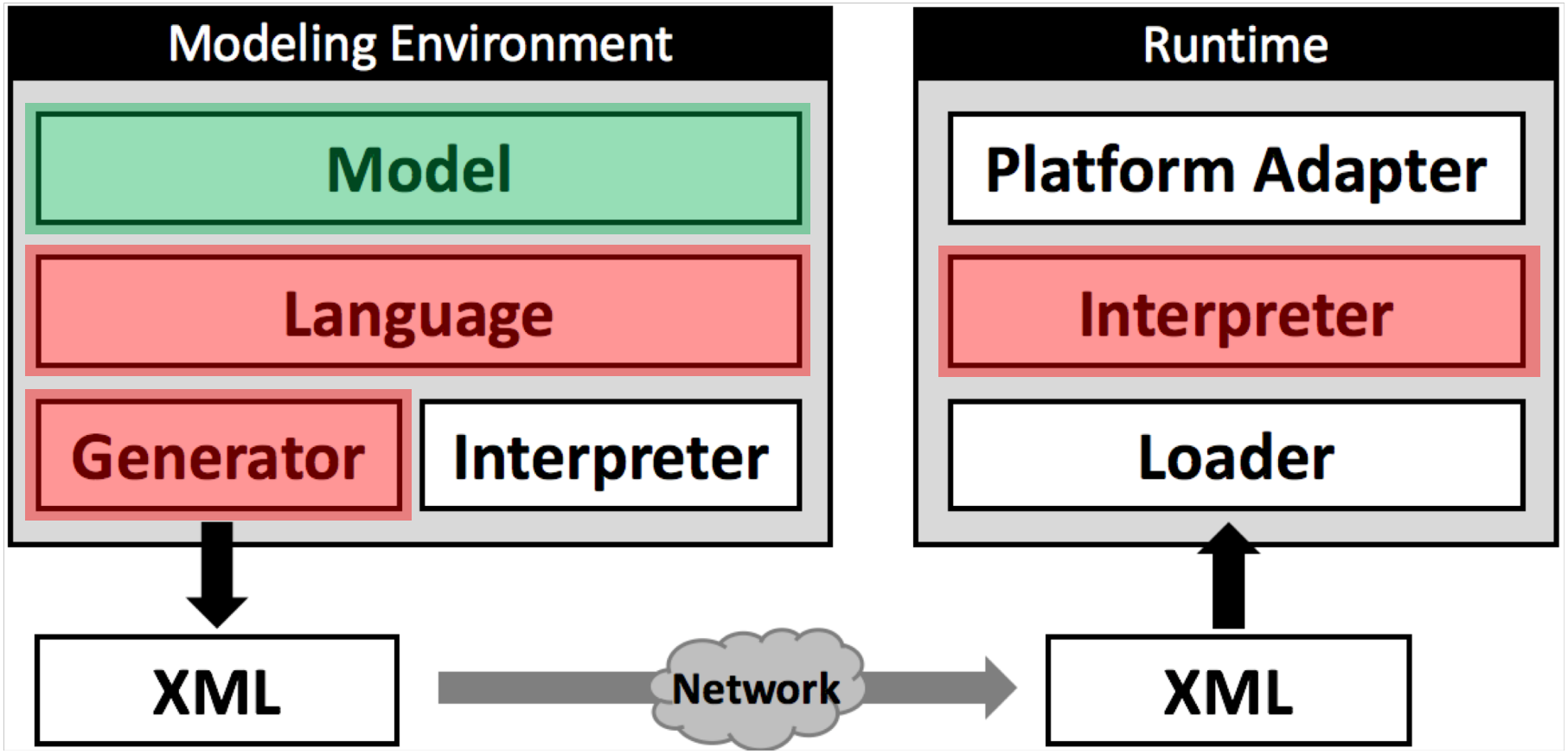
What good is all the abstraction if we cannot trust the translation to the implementation?

System Architecture



What good is all the abstraction if we cannot trust the translation to the implementation?

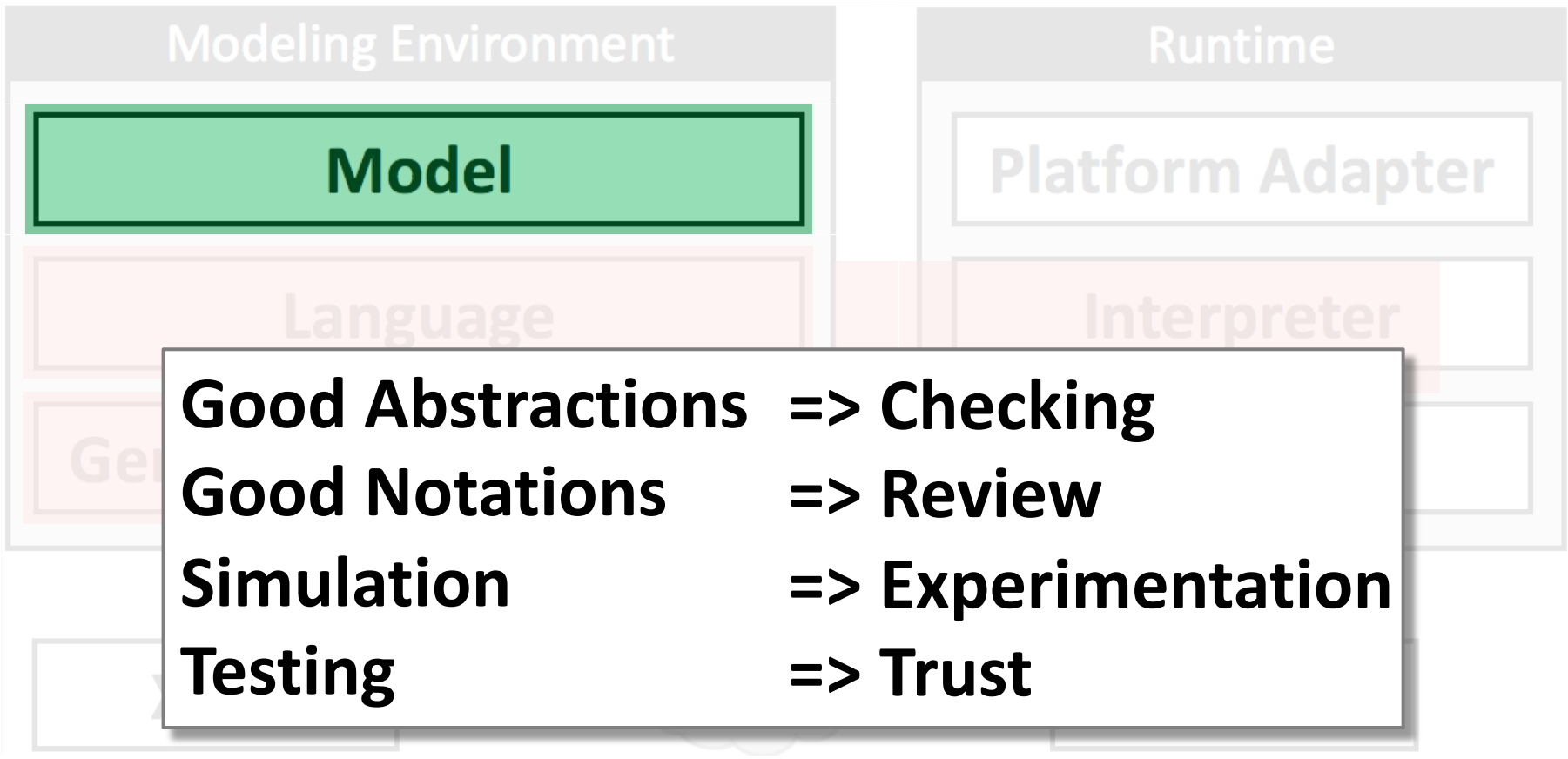
System Architecture & Safety Standards



Tools may introduce *additional systematic errors* if faulty.
Safety **standards** require reliable mitigation of such errors.

DO-178C EN50129 IEC62304 ISO26262

System Architecture & Safety Standards



Good Abstractions => Checking
Good Notations => Review
Simulation => Experimentation
Testing => Trust

Tools may introduce *additional systematic errors* if faulty.
Safety **standards** require reliable mitigation of such errors.

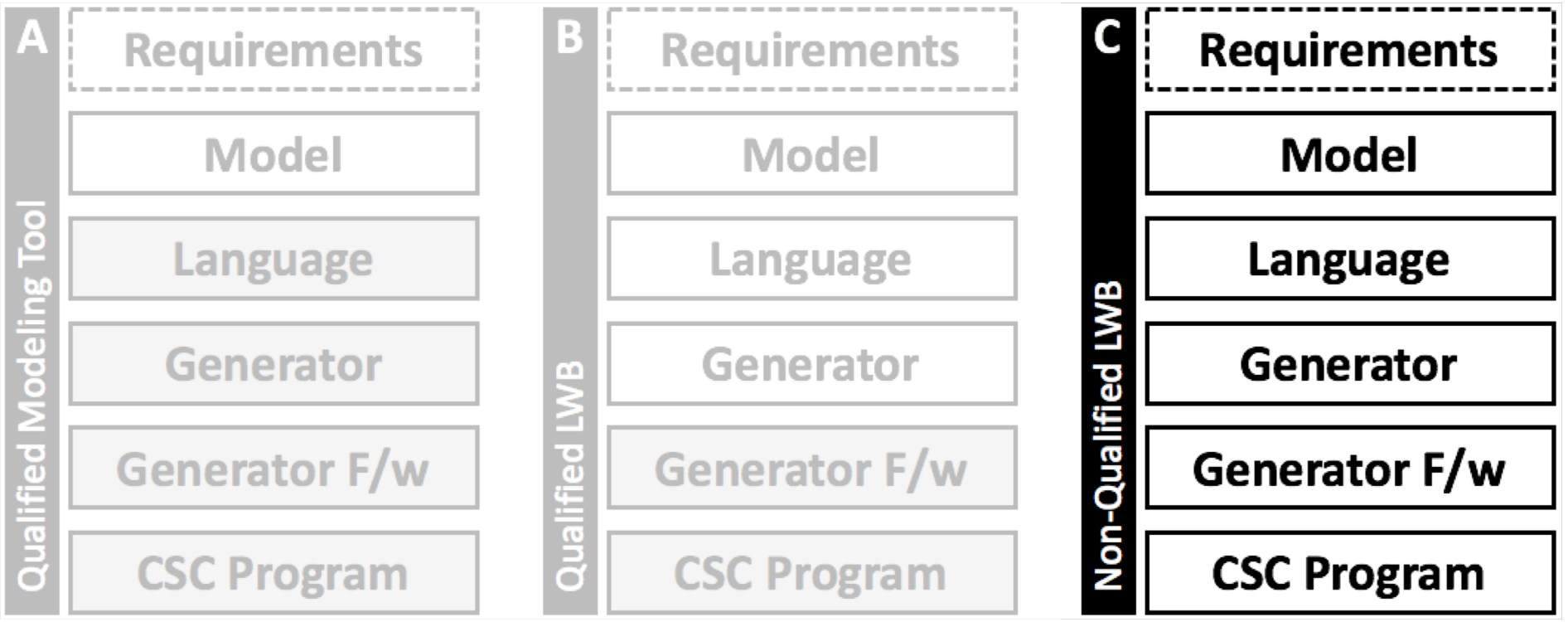
DO-178C EN50129 IEC62304 ISO26262

5

Safety

Unqualified Tools!

What good is all the abstraction if we cannot trust the translation to the implementation?



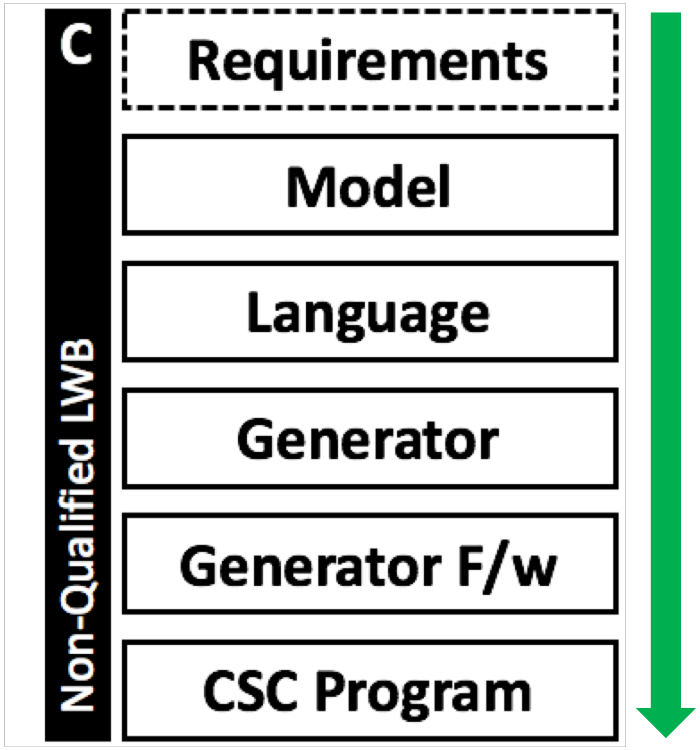
Unqualified Tools!

End-to-end testing required.
How to do this without exploding effort?

Automated Redundancy

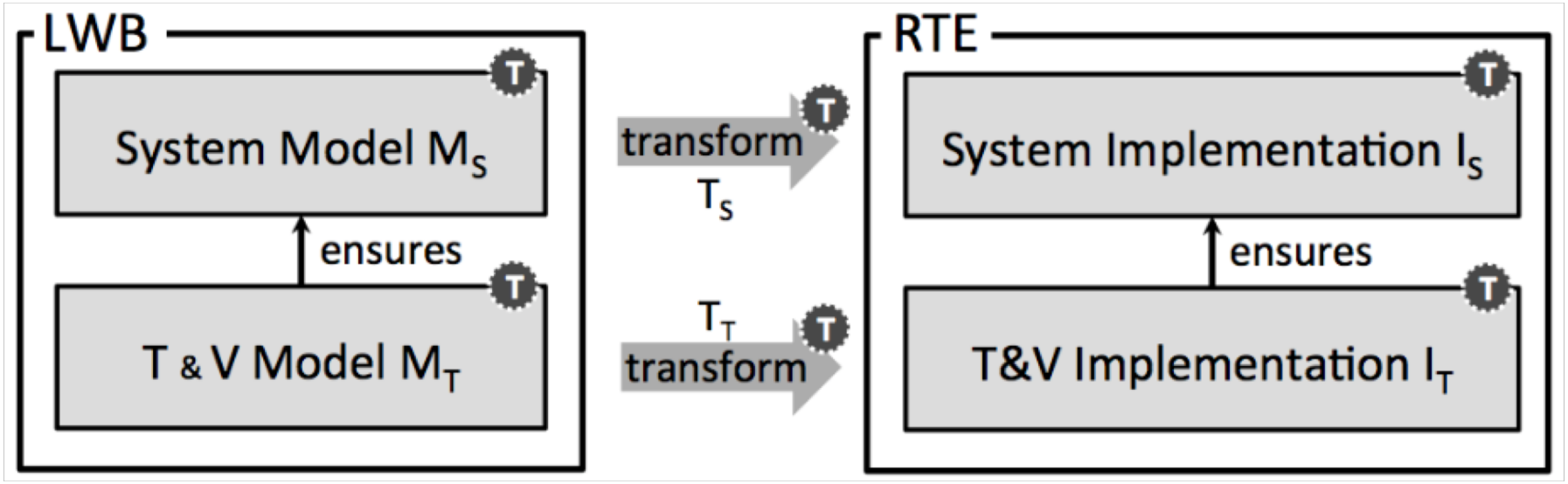
catch errors in redundant path
while reducing manual effort.

+ specific risk mitigations



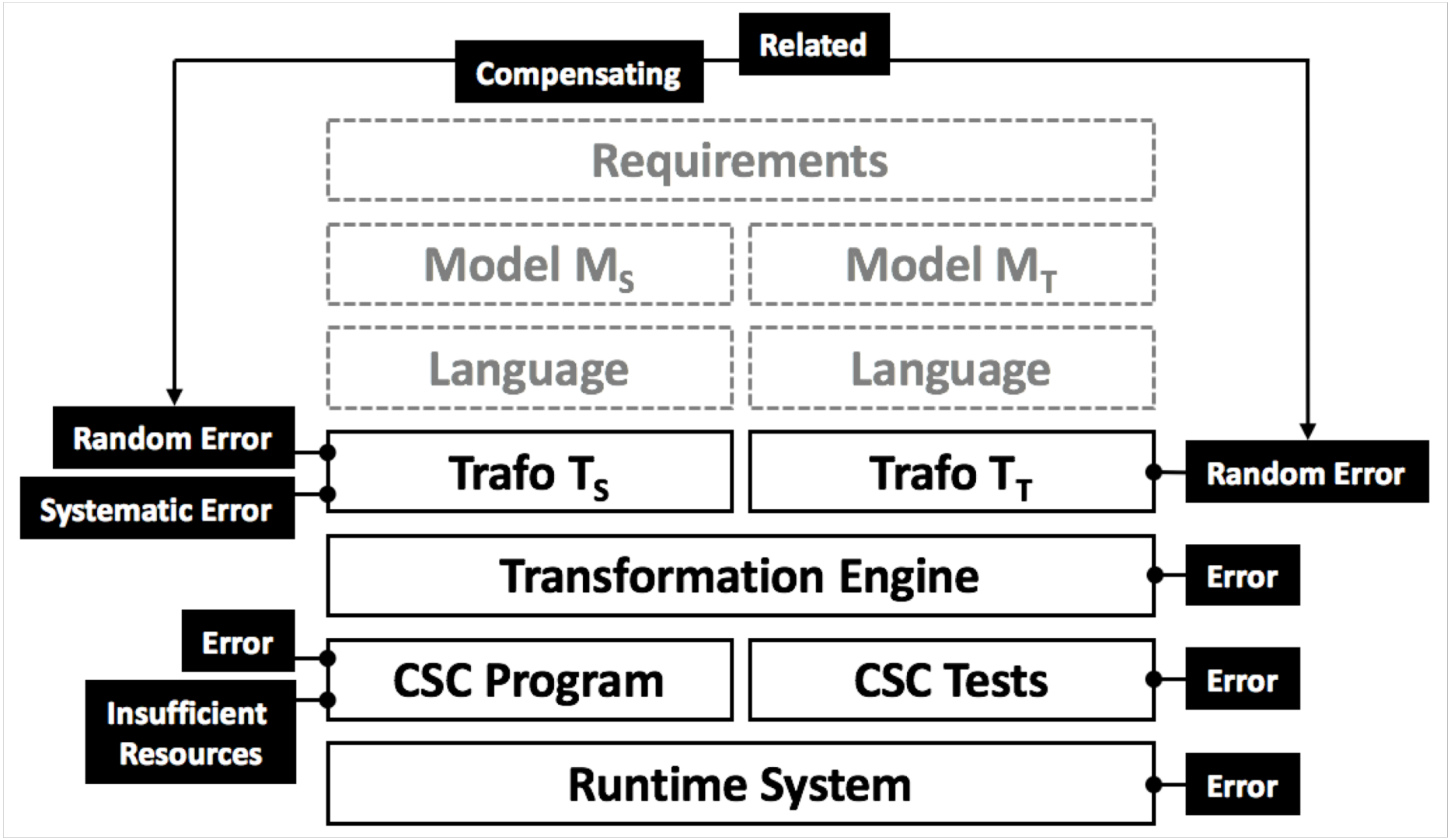
Modeling Architecture

Model the Algo/System with the DSL and also model the tests/verification. Then translate both and execute on the level of the implementation.

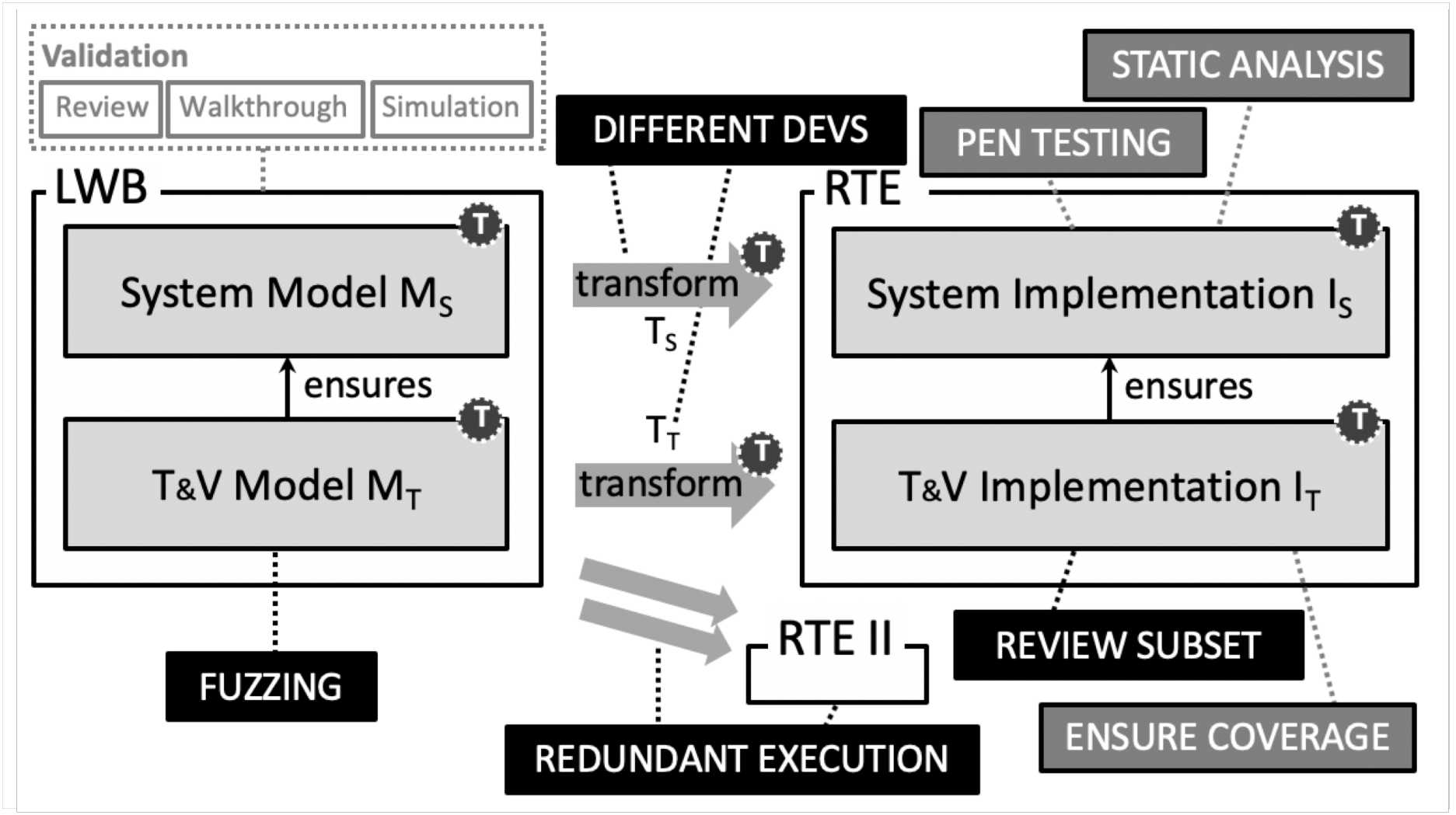


+ Risk Analysis + Mitigations

Risk Analysis



Mitigations – Safe Modeling Architecture



Mitigations – Safe Modeling Architecture

use **redundant execution** on two execution engines
use **different developers** for the two trafos
review a subset of the generated code
clearly define and **QA the DSL**
to use **fuzzing** on the tests
ensure high **coverage** for the tests
run the **tests** on the final **device**
perform **static analysis** on the generated code
perform **penetration testing** on the final system
and use **architectural safety mechanisms**.

Mitigations – Safe Modeling Architecture

use **redundant execution** on two execution engines
use **different developers** for the two trafos
review a subset of the generated code
clearly define and **QA the DSL**

---

only these specific to DSL use

to use **fuzzing** on the tests
ensure high **coverage** for the tests
run the **tests** on the final **device**
perform **static analysis** on the generated code
perform **penetration testing** on the final system
and use **architectural safety mechanisms**.

Mitigations – Safe Modeling Architecture

use **redundant** execution on two execution engines

- **C++ interpreter on device**
- **In-IDE Java Interpreter**

Lots of overhead? Not really.

Validation: the in-IDE interpreter is used for interactive testing, exploration, understanding, simulation. HCP‘s single-most appreciated use of the models!

Verification: addresses unrelated but compensating, as well as related errors in the transformations. Does not rely on trafo engine, so finds error in it. It‘s also simple (!fast), so acts as a specification.

Test Stats and other Numbers

Two reference Algos, 305 test cases for Bluejay, 297 for Greenjay, plus lower-level tests for decision tables and trees

100% line coverage regarding language structure, Java interpreter and C++ interpreter

Validation Effort Reduction from **50 PD to 15 PD**

Test Setup Effort reduced by a **factor of 20**

Shortened Turnaround for req -> impl -> write tests -> execute tests b/c of much better tool integration

„Tremendous Speedup“ for changes to algo *after* it has been validated – automatic reexecution of everything.

Coverage Analysis


```
assessment: StructuralCoverage
query:      structural test coverage {...}
sorted: ☒    must be ok: ☐    hide ok ones: ☐
last updated: Feb 28, 2017 (9 months ago) by markusvoelter
```

---

org.iets3.core.expr.base

TupleValue	Covered. N=2, V=12 H=3..3
SomeValExpr	Covered. N=11, V=197 H=6..9
LogicalImpliesExpression	Covered. N=4, V=10 H=2..3
ErrorLiteral	Covered. N=27, V=198 H=0..9
ErrorExpression	Covered. N=9, V=24 H=1..3
RangeTarget	Covered. N=13, V=104 H=4..7
TupleAccessExpr	Covered. N=4, V=10 H=2..3
SomeExpression	Covered. N=21, V=252 H=4..9
NoneExpression	Covered. N=21, V=58 H=1..6
CastExpression	Covered. N=3, V=12 H=3..6
AsSetOp	Covered. N=5, V=23 H=3..8
SimpleSortOp	Covered. N=5, V=22 H=3..5
ForeachOp	Missing.
OldMemberRef	Covered. N=1, V=32 H=8..8
RecordMember	Partial. Missing: [type_old, type_old]
RecordLiteral	Partial. Missing: [type_old]

Test Generation

kernelf
functional • embeddable • extensible

For everything that can be seen as taking a list of arguments, those can be synthesized.

```
test case TestFunctions [success] {
  vectors function add  -> producer: random 25
          results: true
```

	a	a2	b	c	s	res	status
0: valid	3	-3.06	false	BLUE	""	3	ok
1: valid	2	2.74	false	BLUE	"M/Yh-0I/ac"	2	ok
2: valid	1	0.22	false	BLUE	""	1.22	ok
3: invalid input	4	-0.45	true	BLUE	"7l:6h7sg!afLmULGU8wtI00H9"		not executed
4: invalid input	1	1.38	false	RED	"Mtoa7J66uuWTye2f2-fLhD§hj8C2K"		not executed
5: invalid input	1	-2.63	false	BLUE	"n66r7E (f0J$aQMj§"		not executed

```
vectors function plus -> producer: eqclass
        results: true
```

	a	b	c	res	status
0: valid	0	-10	GREEN	-10	ok
1: valid	0	-10	BLUE	-10	ok
2: valid	0	10	GREEN	10	ok
3: valid	0	10	BLUE	10	ok
4: valid	0	-1	GREEN	-1	ok

	a	b	status
0: valid	7	2	no expected value given; actual was 14
1: valid	1	8	no expected value given; actual was 8
2: valid	5	2	[POST] res == a * b
3: valid	7	8	no expected value given; actual was 56
4: valid	5	6	[POST] res == a * b
5: valid	8	0	[PRE] b > 0
6: valid	5	8	[POST] res == a * b

Mutation / Fuzzing

kernelf
functional • embeddable • extensible

For a set of tests that all succeed, if after a change to the program they still do, this is a problem.

```
fun doodle(a: number[1|5]) = if true then a else a * 1

fun add(a: number[1|5]) = alt [ a > 3      => a + 1
                              [ otherwise => doodle(a) ]

test case TestFunctions [incomplete] {
  vectors function add                    -> producer: random 30
          results: true                      { 30 entries }
          mutator: # of mutations 20
                   keep all:      false
                     -> ParensExpression
                     -> NumberLiteral
                     -> LogicalNotExpression
                     -> ParensExpression
                     -> ParensExpression
}
```

```
fun add(a: number[1|5]) = alt [ ((a) + 1) > 3 => a + 1
                              [ a
                              [ otherwise     => doodle(a) ]

fun doodle(a: number[1|5]) = if true then a else a * 2
                                                    1

fun doodle(a: number[1|5]) = if true then a else ((a * 1) + 1)
                                                  a * 1

fun doodle(a: number[1|5]) = if !(true) then a else a * 1
                                true
```

6

Did it work?

Overall Benefits

Faster iterations with medical team to design algo.

Acceleration of the medical algo **validation**, and thus overall acceleration of mobile App development.

Reduced overall DTx concept to market **times.**

Medical algo implementation **independent of target** OS (iOS, Android, windows) with identical behavior.

Distribution to mobile Apps outside of Google PlayStore / Apple AppStore.

Quality Improvements I

Simulation reduces inception errors and algorithm design flaws.

Simulation projects medical team as patient or healthcare practitioner.

Simulation helps with vigilance case investigations using real world data, which is a huge safety advantage for SaMD!

Encapsulation of medical device features and segregation from unregulated features drastically reduced defect ratio in SaMD.

Quality Improvements II

We implemented a true **test** pyramid

We were able to improve the algo test **coverage**

We saw increased App **stability** due to DSL + RTE

Algorithm coding **errors** were reduced compared to previous GPL approaches

DSL made the algo code **readable** to engineers who are not coders

Better alignment of implementation with medical **requirements**

DSL Development

People with DSL-building **experience** are not easy to find.

Language engineers are **not** DTx experts and vice versa.

Adopting the **MPS** projectional editor took some convincing power.

It took a few **iterations** before the DSL met our expectations.

Keeping the DSL **lean** and avoiding unnecessary concepts was a challenge e.g., hierarchical state machines
measurement units and conversions

Aligning the DSL concepts and design decisions with mindset and experience of the **test and validation team** took time.

Adoption of the new Approach

It took **time and money** to develop the DSL, the RTE, simulator, test & regulatory strategy.

Finding and **training** medical algorithm designers to be DSL users wasn't that easy.

Convincing **regulatory** affairs and **quality** assurance about the DSL + RTE approach took some time.

The runtime driving the App put the coding paradigm upside-down and didn't convince all mobile **developers.**

Integrating the runtime with our **existing technology stack** and framework was time consuming.

Prepare to invest into your future!

Features vs. Speed

Our goal was to reduce the algo design times and go-to-market times:

Instead, we saw that our medical team and our algo designers used the fast iterations and the simulator to finetune the algo, dealing with more exceptions and special cases, which caused the complexity index of the algos to skyrocket.

Risk:

This would have made the DTx behavior in real life less predictable.

Mitigation:

Simplify these over-engineered algos.

Introduce an algo complexity index.

Instruct teams to keep the algo design simple by not exceeding the maximum complexity index.

A nice problem to have!

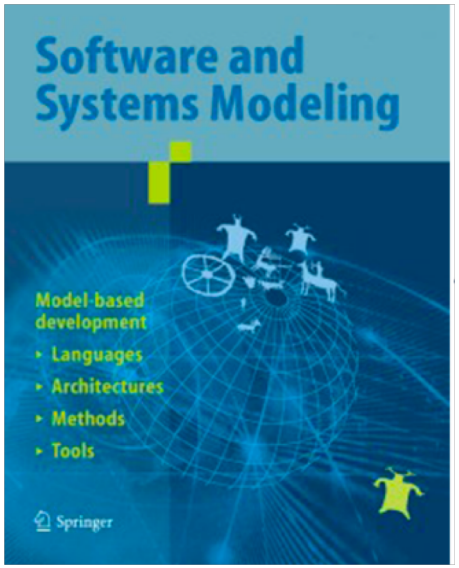
Wish vs. Reality

Some people still believe that the DSL should be a **do-it-yourself graphical tool** for medical doctors. I am afraid that's still science-fiction, and this might get some resistance from the FDA…

To make this world a better place, let's have medical doctors deal with the health of their patients and the engineers deal with technology. **Efficiently!**

… and use DSL models as a means of efficient communication and experimentation.

WRAP UP



Software & Systems Modeling
pp 1–24 | Cite as

Using language workbenches and domain-specific languages for safety-critical software development

Authors | Authors and affiliations

Markus Voelter, Bernd Kolb, Klaus Birken, Federico Tomassetti, Patrick Alff, Laurent Wiart, Andreas Wortmann, Arne Nordmann

Regular Paper
First Online: 17 May 2018

Abstract

Language workbenches support the efficient creation, integration, and use of domain-specific languages. Typically, they execute models by code generation to programming language code. This can lead to increased productivity and higher quality. However, in safety-/mission-critical environments, generated code may not be considered trustworthy, because of the lack of trust in the generation mechanisms. This makes it harder to justify the use of language workbenches in such an environment. In this paper, we demonstrate an approach to use such tools in critical environments. We argue that models created with domain-specific languages are easier to validate and that the additional risk resulting from the transformation to code can be mitigated by a suitably designed transformation and verification architecture. We validate the approach with an industrial case study from the healthcare domain. We also discuss the degree to which the approach is appropriate for critical software in space, automotive, and robotics systems.

http://voelter.de/data/pub/MPS-in-Safety-1.0.pdf

Overall Efficiency and Quality
goals were achieved as expected.

Developing a good DSL is Work
both conceptually and technically.

The Safety Mechanisms Work -
overall error rate much lower than in manual coding

Formality and Fancy Notations
as supported by MPS are useful for validation

A Model is not enough -
analysis, simulation, testing, debugging is crucial.

Yes, we‘d do it again.