Formal Methods Considered Normal

Janet Barnes and Angela Wallenburg

ABZ Conference, Southampton, 6 June 2018

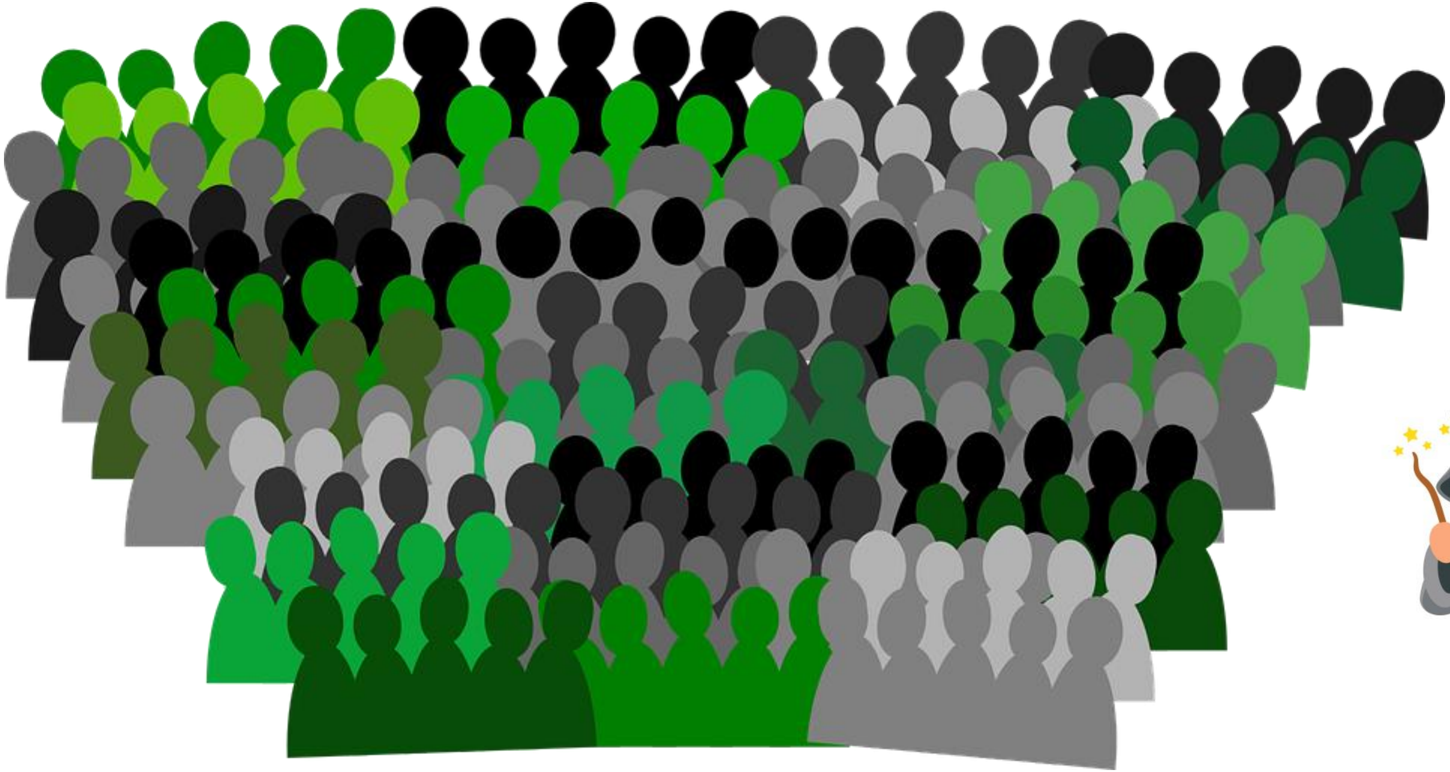
altran

Agenda

01.

Setting the Scene

Formal Methods in Industry: Always Applied by Expert Clique?



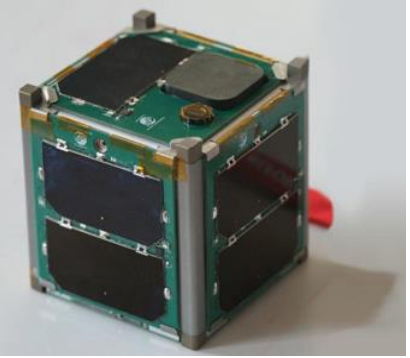
Some People Do Proof Every Day…

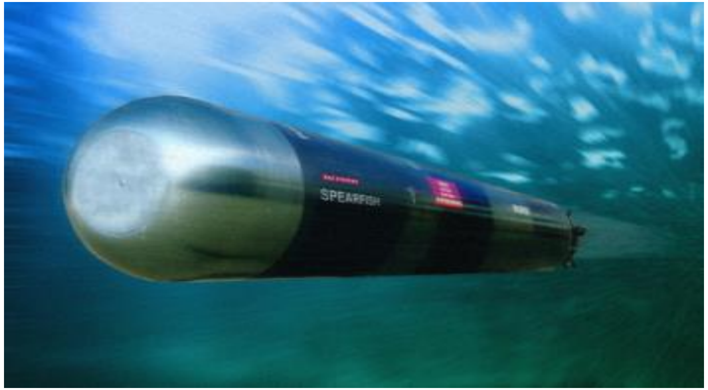
<photo collage of hordes of SPARK users>

… and are

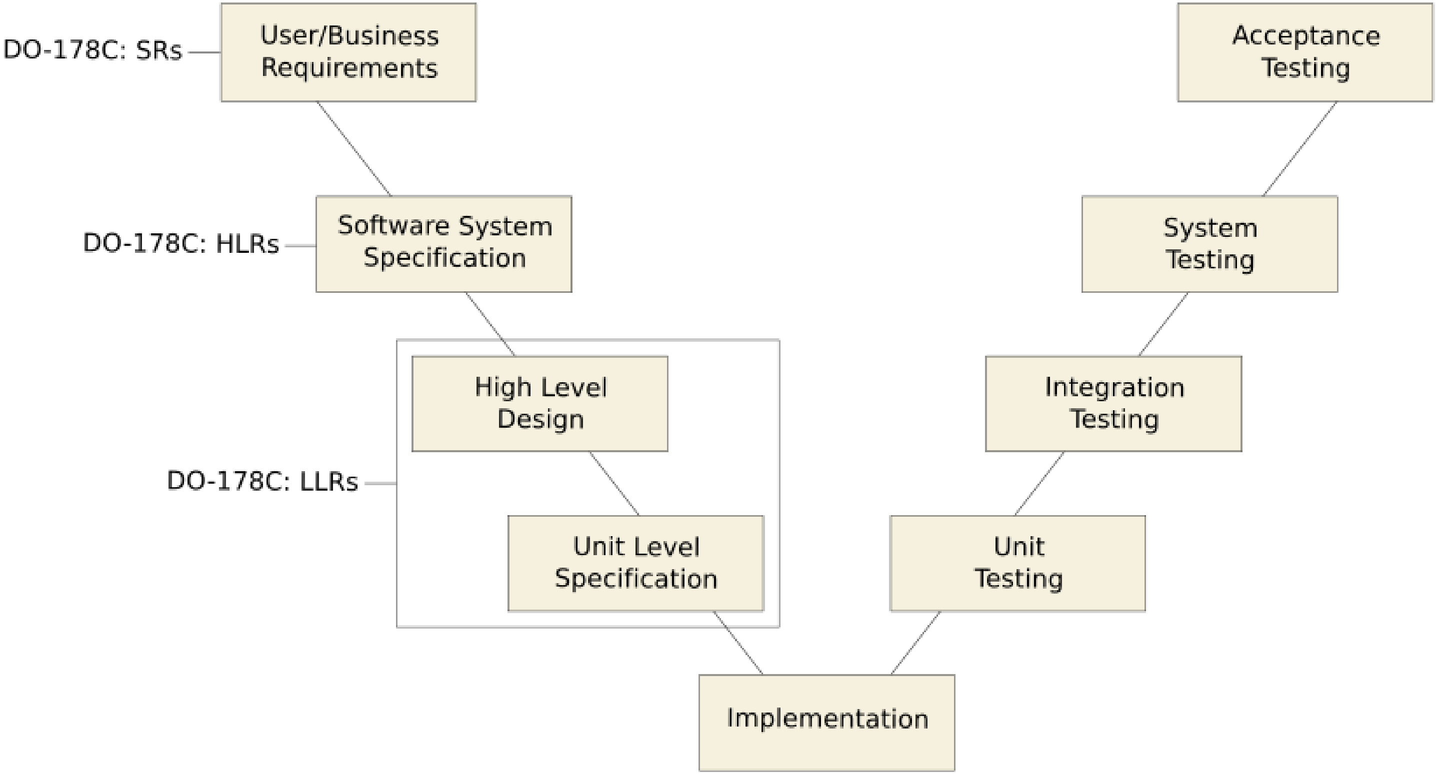
- aware of it
- using a principled (CbyC) approach
- not FM experts
- using SPARK

Why We Do It



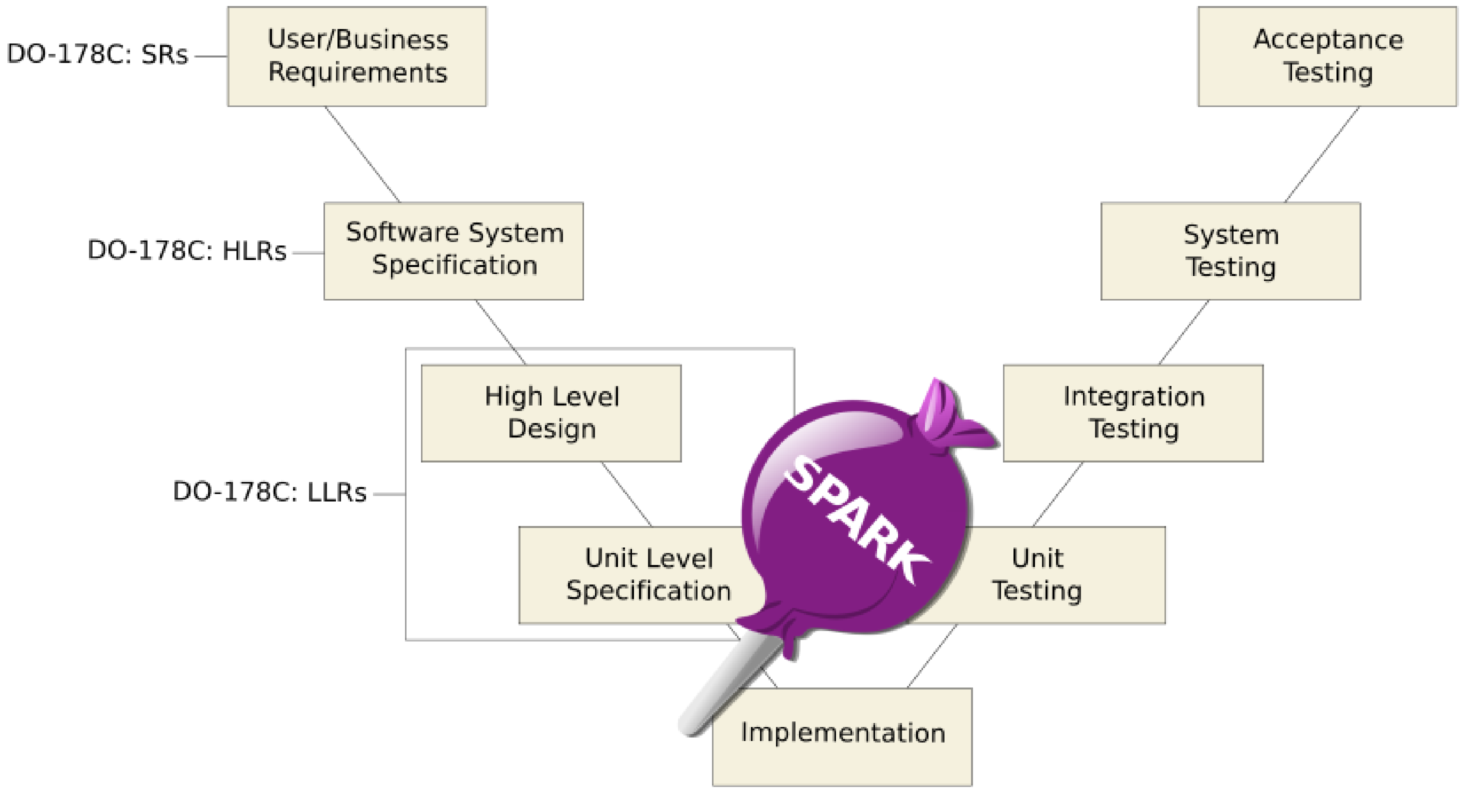


The Prevailing V-Model



altran

A Sweet Spot: SPARK



02.

SPARK – What has Worked and Why?

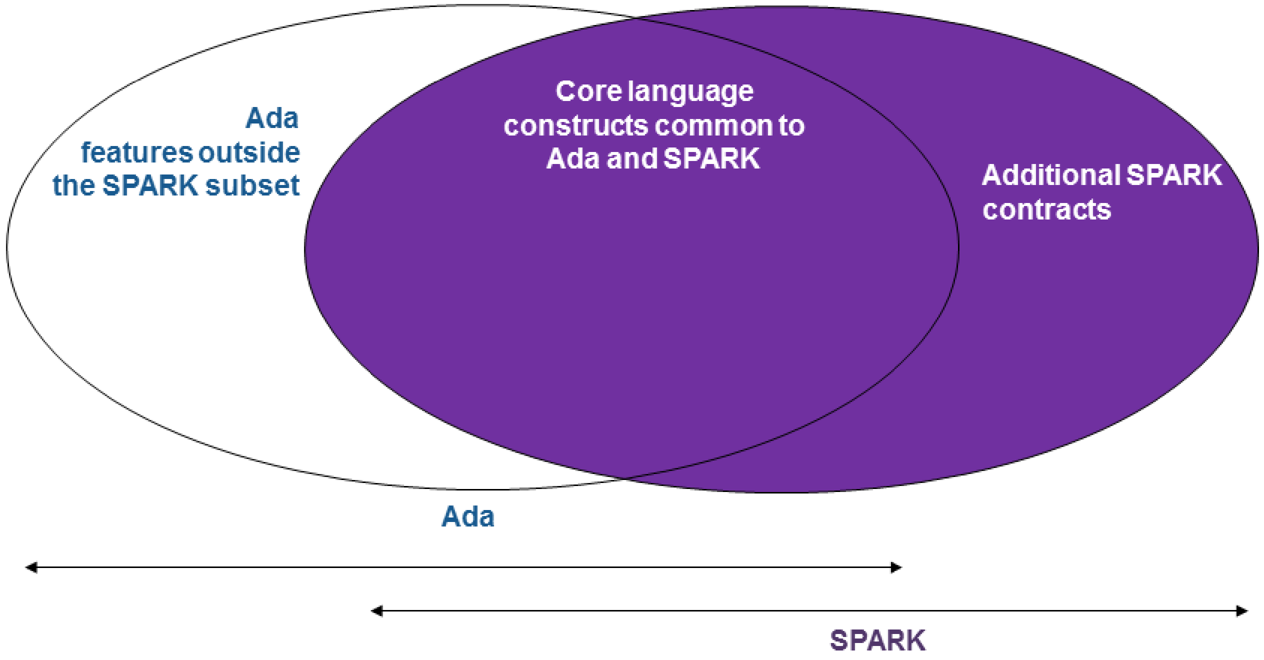
What is SPARK?

SPARK is...

- a language
- a set of tools
- a design approach

... for development of high-integrity applications.

SPARK – Analysable Subset of Ada



How We Feel about Types

Contract Example – An Observation about Types

```
procedure Sqrt (Input : in Integer; Res: out Natural)
  with
       Pre  => Input >= 0,
       Post => (Res * Res) <= Input and
               Input < (Res + 1) * (Res + 1);
```

What difference do types make?

Contract Example – An Observation about Types

```
procedure Sqrt (Input : in Integer; Res: out Natural)
  with
       Pre  => Input >= 0,
       Post => (Res * Res) <= Input and
               Input < (Res + 1) * (Res + 1);
```

With the help of types…

```
procedure Sqrt (Input : in Natural; Res: out Natural)
  with
       Post => (Res * Res) <= Input and
               Input < (Res + 1) * (Res + 1);
```

… less to write!

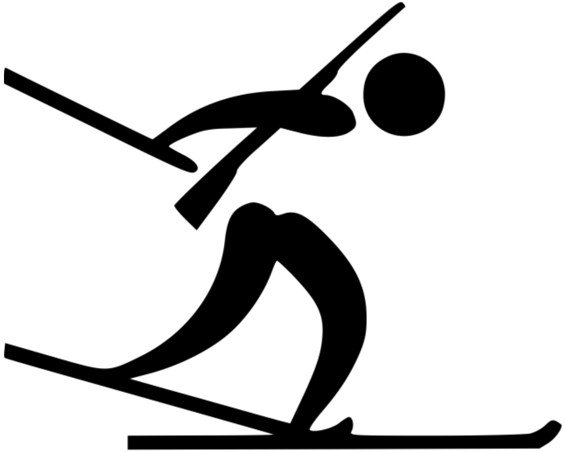
Mixing Test and Proof

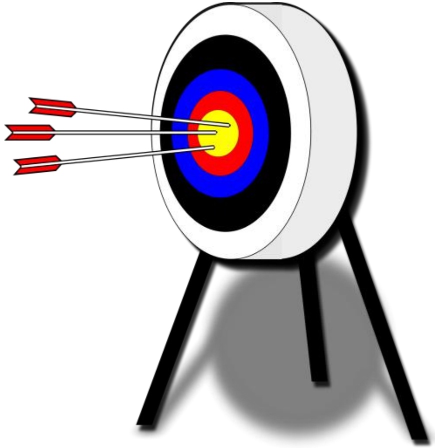
- Dynamic Semantics – contracts can be:
  - compiled
  - checked at run time
  - thought of as “pre-assert” and “post-assert”

- Static Semantics – contracts can be:
  - interpreted in logic
  - formal pre- and post- assertions of a Hoare triplet
  - checked exhaustively by a theorem prover

Number One Killer of FM Tools Uptake?

Alarm:
Off Target





!?

Semantics of Contracts – Overflows

- Contracts have the same semantics as in Ada:

```
procedure P (X, Y : in Positive; Z : out Positive)
  with Post => (if X + Y <= Positive'Last then Z = X + Y) and
               (if X + Y > Positive'Last then Z = Positive'Last);
```

X + Y could overflow and raise a run-time exception when the contract is executed

```
“warning: overflow check might fail”
```

Is this a false alarm in your context?

altran

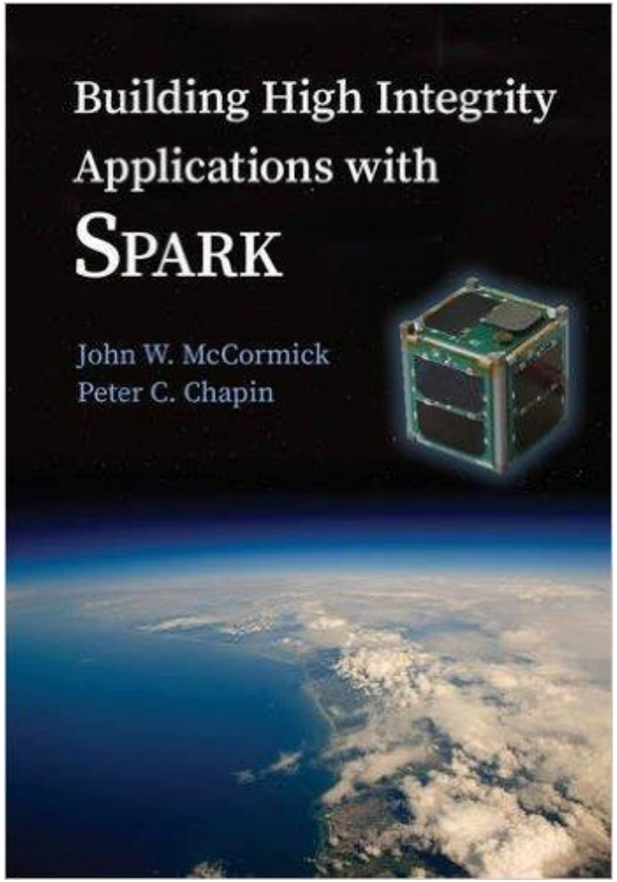
SPARK 2014 Design: Overflow Checking Modes

- Different user needs
  - run-time assertion checking for contracts on/off in deliverable
  - amount of proof activity, requirements on false alarm rate
- Customisable overflow checking mode
- Options
  1. strict Ada semantics for overflow checking
  2. minimized overflow checking
  3. eliminated - no possibility of overflow (mathematical semantics)
- Specified semantics is used both at run time and for proof

altran

SPARK - Teaching

- formal and sound
- contracts
- industrially used
- open source mature tools
- support for academic faculty
- code examples, labs, and sample answers
- excellent books: Chapin, McCormick 2015 (Barnes' book 3rd edition)
- Altran 5 day course for any and all



Consider teaching SPARK…

Lessons Learned in Research Productisation

- Successful case study?
- What about repeat usage?

Major investment.

Typical activities:

› Producing user-friendly wrapping software
› Patching theories (*80-20% rule for research work too*)
› Constructing cost-benefit arguments
› Pitching & Teaching
› Porting software
› Making resource control deterministic
› Test & build

03.

Current Large Scale Formal Specification

altran

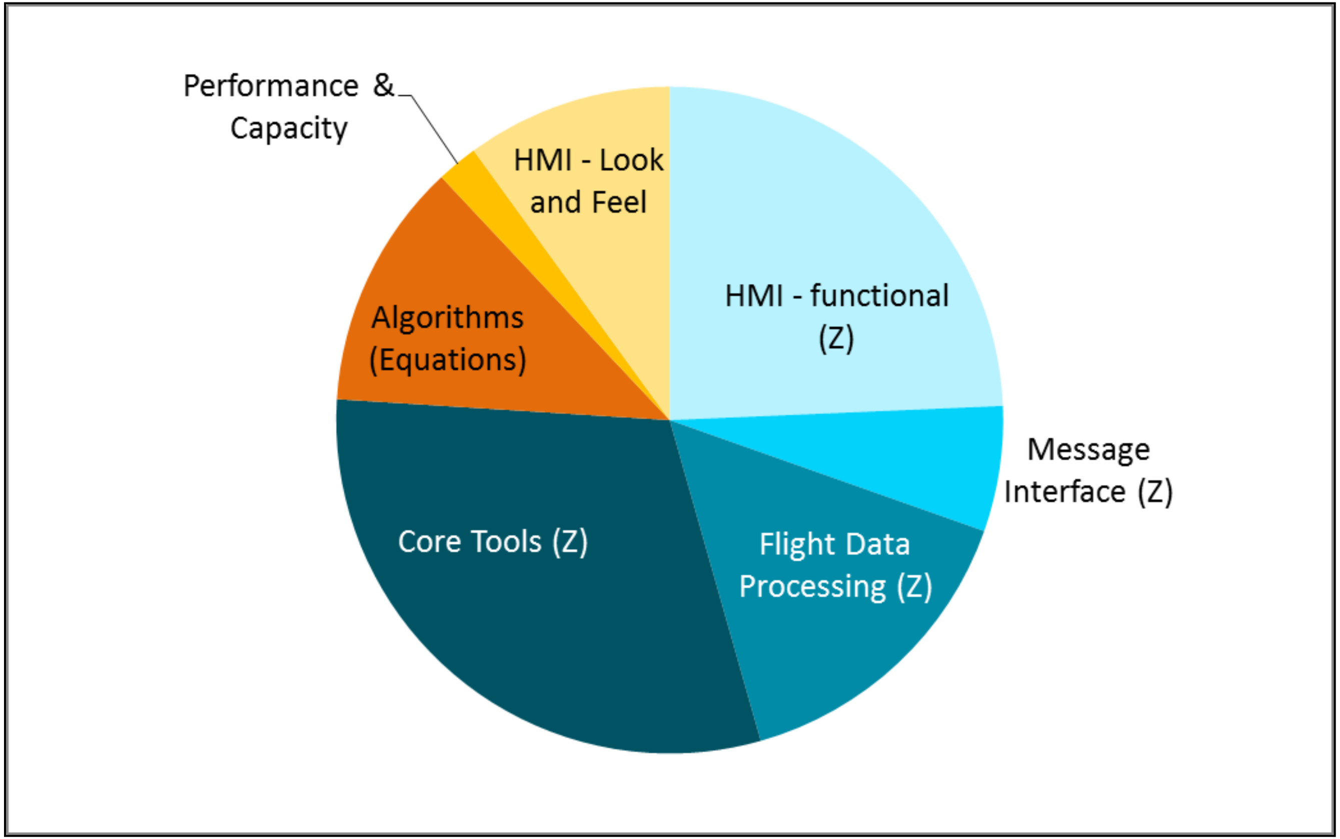
Case Study

- Air Traffic Control - iFACTS

What is iFACTS?

- Tactical medium term support to controllers:
  - › **Trajectory Prediction**
  - › **Conflict Detection**
  - › **Flight Path Monitoring**
- Enhances existing ATC system:
  - › **Additional tools**
  - › **Display components**
- Improves airspace efficiency.

How much of iFACTS is Specified in Z?



altran

How big is the iFACTS Z specification?

- Initially developed by a team of **12 engineers**.
- Actively developed or maintained over **10 years**.
  - Supporting successful incorporation of major changes.
- The specification comprises over **40,000 lines of Z**.
- The key Z documents amount to over **3,000 pages**.

Why we choose Z to write specifications

- Z models can be **Abstract**
- Z schema notation allows **Structuring** of the specification through encapsulation, modularisation and composition.
- Z is **Expressive** with a large toolkit of operators
- Easy to combine English narrative with Z notation

$$
\begin{array}{|l}
\hline
\textit{Flight} \\
callsign : CALLSIGN \\
departure : FIX \\
destination : FIX \\
type : FlightType \\
\hline
departure \in knownFixes \\
\hline
\end{array}
$$

$$\lambda\, x : \mathbb{Z} \bullet x * x$$

$$TPRecognitionChange \fatsemi TPUpdateTrack$$

$$\Delta Flights$$

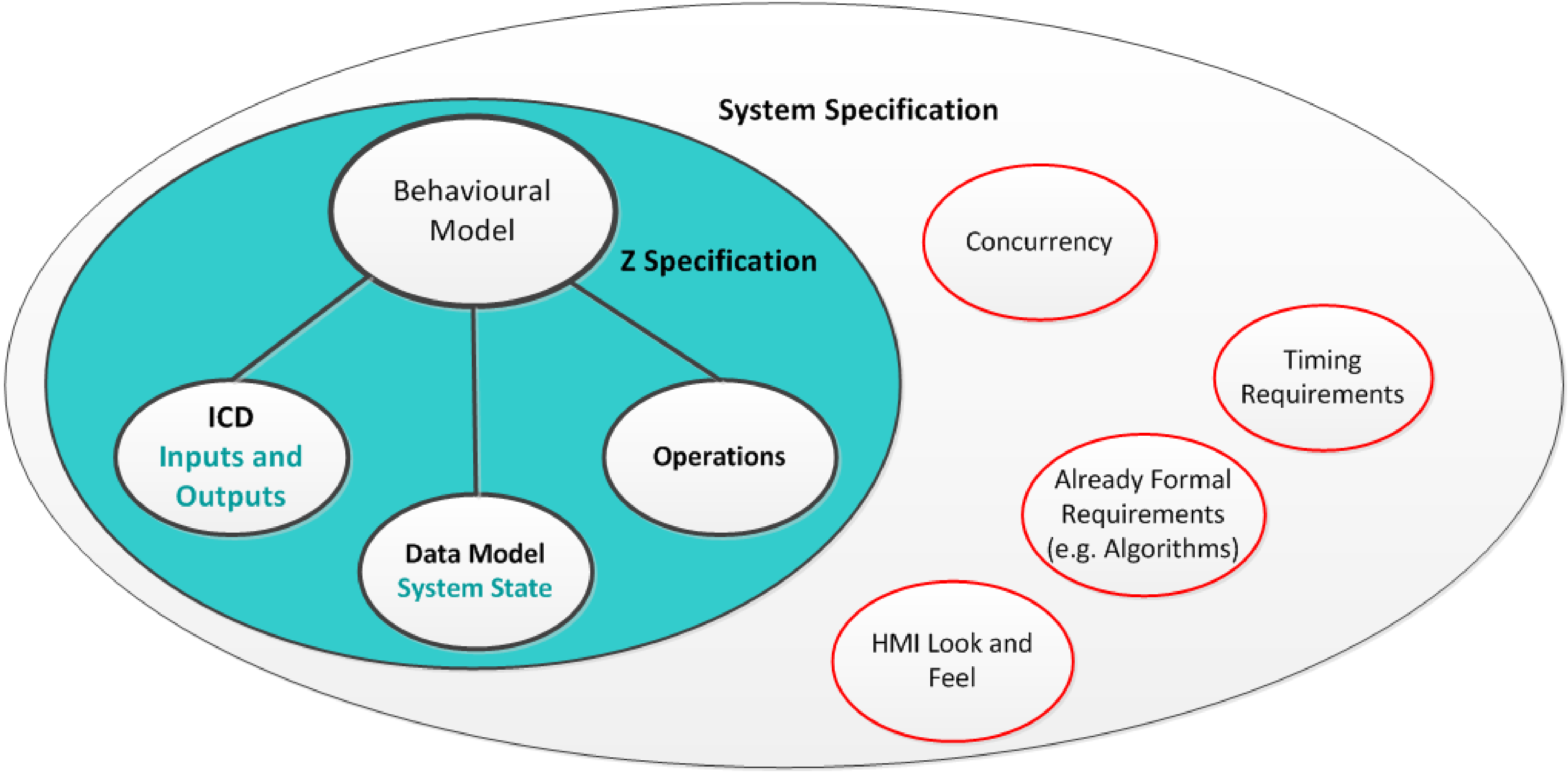
$$departure \in knownFixes$$

$$\Xi Tracks$$

$$\langle a, b \rangle$$

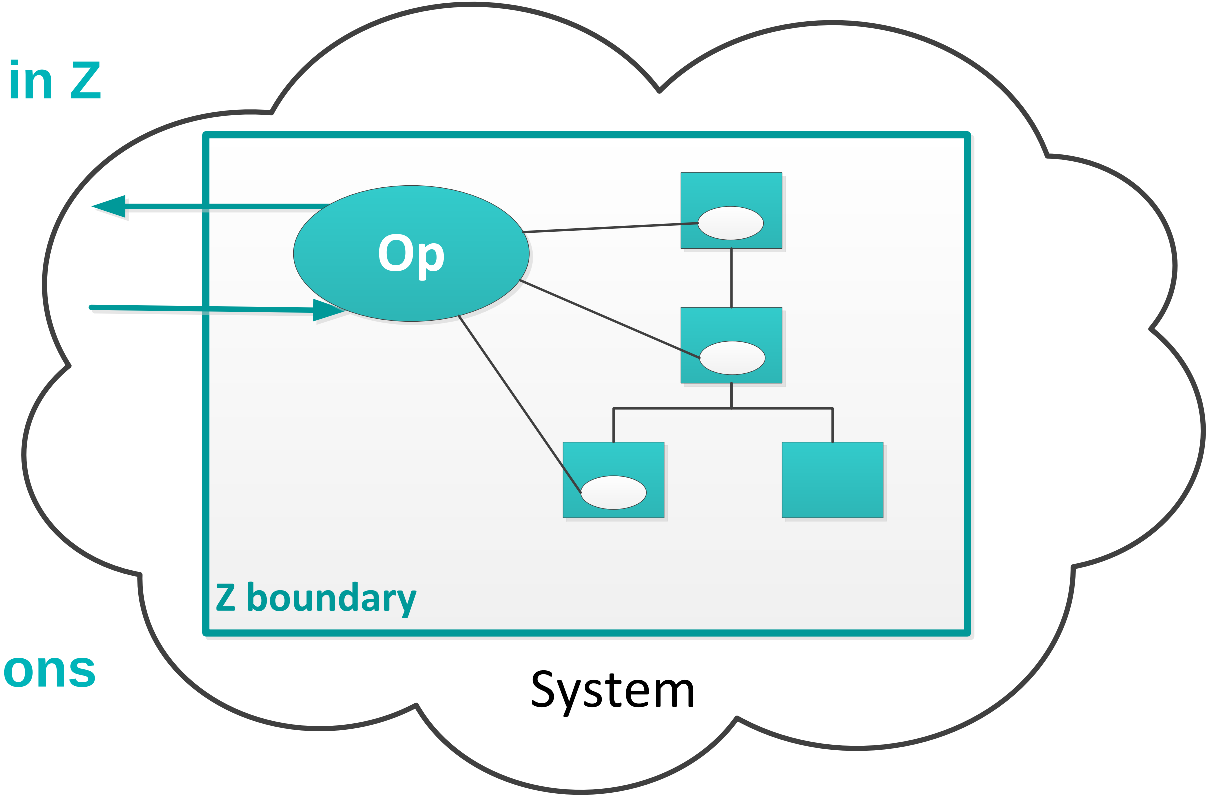
$$flightCallsign \llparenthesis knownFlights \rrparenthesis$$

How we use Z in specifications



How we specify systems in Z

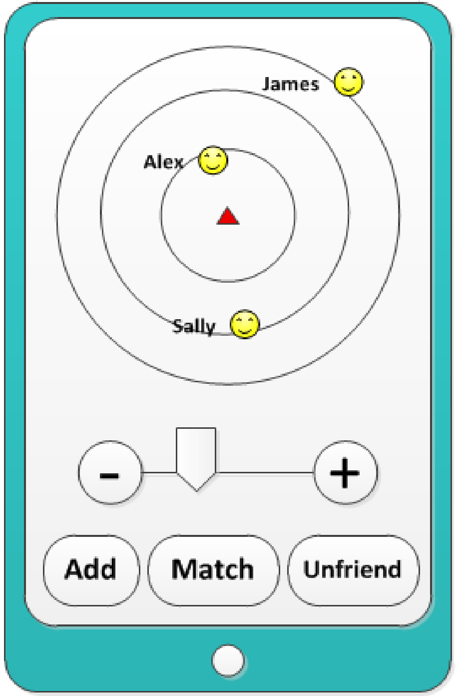
- **Z boundary** identified
- Structured **data model**
- Abstract **messages**
- System **operations**
- Detailed **partial operations**

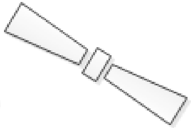


- For large systems it is important to **structure** the specification

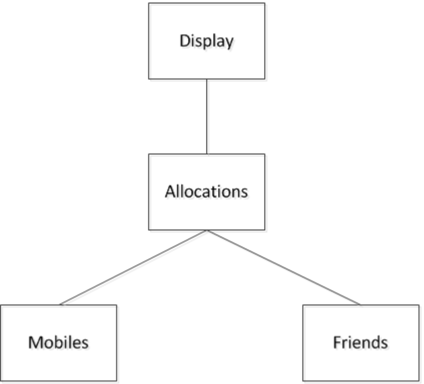
How we specify systems in Z –The boundary

- Right level of **abstraction**
- Drawing the line at the **display**
- What **messages** cross the Z boundary?





How we specify systems in Z – Data Model



- **Hierarchy** of packages
- package **static characteristics**
  - › **primary** state
  - › **derived** state and derivation
  - › **invariants** on the state
  - › **initial** state

MyApp

$Display$
$Allocations$
$Friends$
$Mobiles$

AllocationsDecs

$AllocationsEntitiesDecs$
$AllocationsAssociationsDecs$

AllocationsEntitiesDecs

$myMobile : \mathbb{F} MOBILE$

AllocationsAssociationsDecs

$mobileAllocation : MOBILE \rightarrowtail\!\!\!\!\!\!\!+\;\; FRIEND$

AllocationsDerivedAssociations

$AllocationsAssociations$
$Mobiles$
$Friends$
$activeMobileFriend : MOBILE \rightarrowtail\!\!\!\!\!\!\!+\;\; FRIEND$

$activeMobileFriend = mobiles \lhd mobileAllocation$

Allocations

$AllocationsDerivedAssociations$

$\# myMobile \leqslant 1$
$myMobile \subseteq \mathrm{dom}\ mobileState$
$\mathrm{ran}\ mobileAllocation \subseteq friends$

InitialAllocations

$Allocations$

$myMobile = \varnothing$
$mobileAllocation = \varnothing$

How we specify systems in Z – Operations

- Dynamic behaviour of the whole system
  - response to a **single stimulus**
  - models **outputs**
  - state changes as **post conditions** …

    … postponed to partial operations

GPSUpdate

$\Delta MyApp$

$gps? : GPSMsg$

$RefreshDisp$

$\Xi AllocationsDecs$

$\Xi Friends$

$UpdatePositionMob$

UpdatePositionMob

$\Delta Mobiles$

$gps? : GPSMsg$

RefreshDisp

$\Delta Display$

How we specify systems in Z – Partial Operations

- **behaviour** of one package
- the **detail** of the change

  …as post conditions on the package state

UpdatePositionMob

$$
\begin{array}{l}
\Delta Mobiles \\
gps? : GPSMsg \\
\hline
mobileState' = \\
\quad mobileState \\
\qquad \oplus \{Mobile \mid number = gps?.mobId \\
\qquad\qquad\qquad \land gpsLocation = gps?.position \\
\qquad\quad \bullet number \mapsto \theta Mobile\}
\end{array}
$$

RefreshDisp

$$
\begin{array}{l}
\Delta Display \\
\hline
\Xi DisplayPrimaryDecs
\end{array}
$$

altran

How we specify systems in Z – Schema types

- Heavy use to **aggregate** properties of an entity together.
- Encapsulation is enforced **stylistically**.

Friend

$ident : FRIEND$

$firstname : NAME$

$Surname : NAME$

$knownAs : NAME$

$birthday : Date$

FriendsDecs

$FriendsEntitiesDecs$

FriendsEntitiesDecs

$friendState : FRIEND \pinj Friend$

$friends : \mathbb{P}\ FRIEND$

FriendsEntities

$FriendsEntitiesDecs$

$\forall f : \operatorname{dom} friendState \bullet (friendState\ f).ident = f$

Friends

$FriendsEntities$

InitialFriends

$Friends$

$friends = \varnothing$

AddFriend

$\Delta MyApp$

$newfriend? : FriendData$

$\Xi DisplayDecs$

$\Xi AllocationsDecs$

$AddFriendFr$

$\Xi Mobiles$

How we specify systems in Z – Condition Hierarchies

- We apply a **divide and conquer** approach using predicate schemas
  - Schemas as guards.
  - Partial operations are decomposed into fragments.
  - Schema composition to “do A” and then “do B”

$$
\begin{array}{l}
\textbf{DisplayScreenCentre} \\
\hline
\mathit{DisplayDerivedDecs} \\
\mathit{Allocations} \\
\hline
\mathit{MyMobileExists} \Rightarrow \\
\quad (\exists\, mm : MOBILE \bullet \{mm\} = myMobile \\
\qquad \land\ screenCentre = (mobileState\ mm).gpsLocation) \\
\neg \mathit{MyMobileExists} \Rightarrow \\
screenCentre = defaultPosn \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textbf{MyMobileExists} \\
\hline
\mathit{Allocations} \\
\hline
myMobile \neq \varnothing \\
\hline
\end{array}
$$

Verification of Z specifications

- We always
  - **type check** our specifications.
  - **review** our specifications against the requirements.

- We may also prove the following:
  - **existence** of an initial state.
  - **precondition** of operations
  - **properties** hold of our system

- Our specifications are often too big to realistically apply these proofs.

Our experience – what is successful?

- Implementing from a Z specification.
- Developers and testers agree.
- Training software engineers to read Z.
- Requirements ambiguities found early.
- Z specification is maintainable.

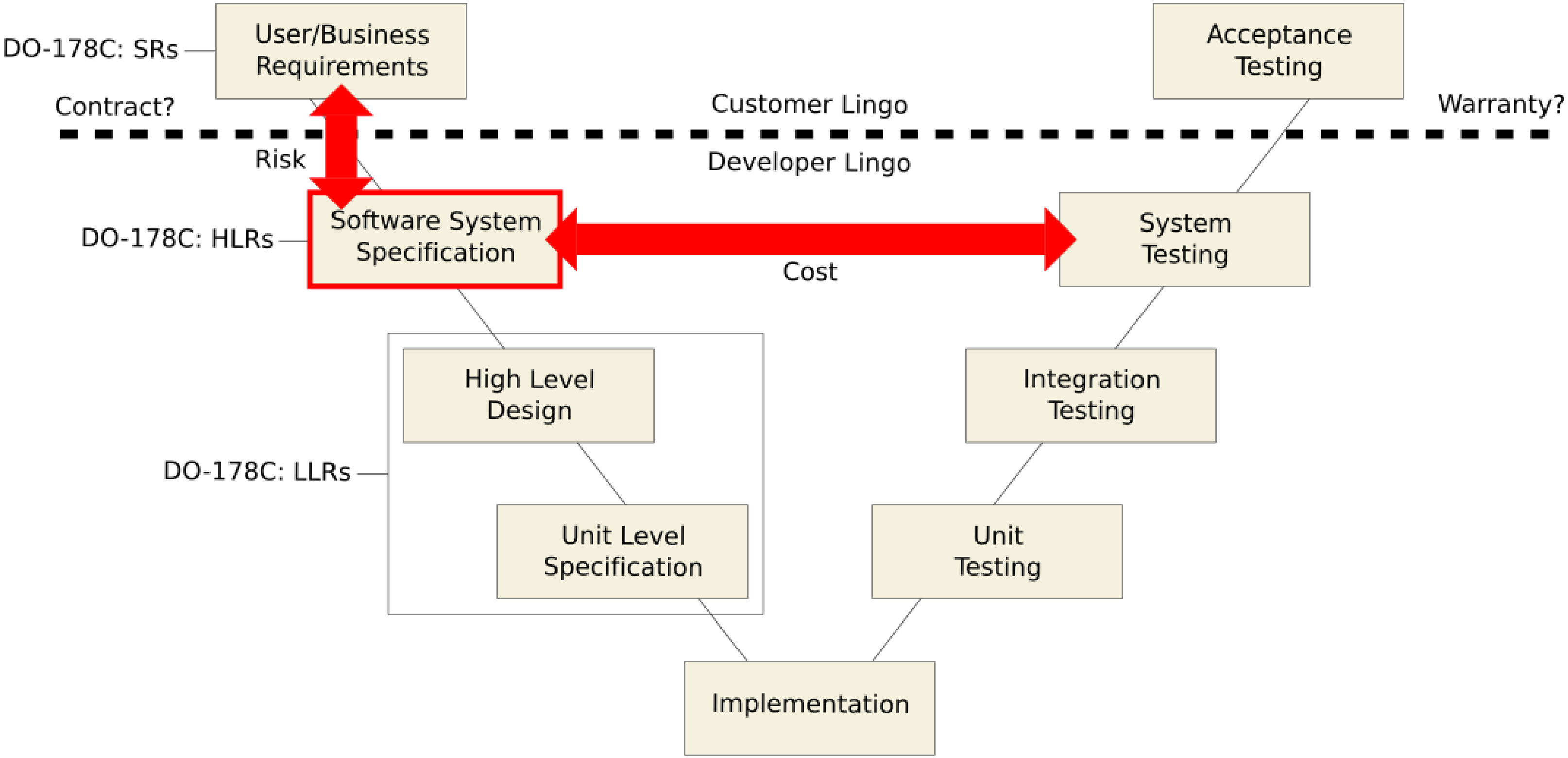
Our experience – what is difficult?

- Finding good Z specification authors
- Overcoming the language gap
- Justifying the up-front investment

04.

Looking Forward

Specification Improvement Motivation



SECT-AIR Goal

"To deliver a step-change improvement in the <u>affordability</u> of aerospace software. This is required to secure and develop the UK as a world leader in critical and complex systems development and enable UK aerospace to build new products."

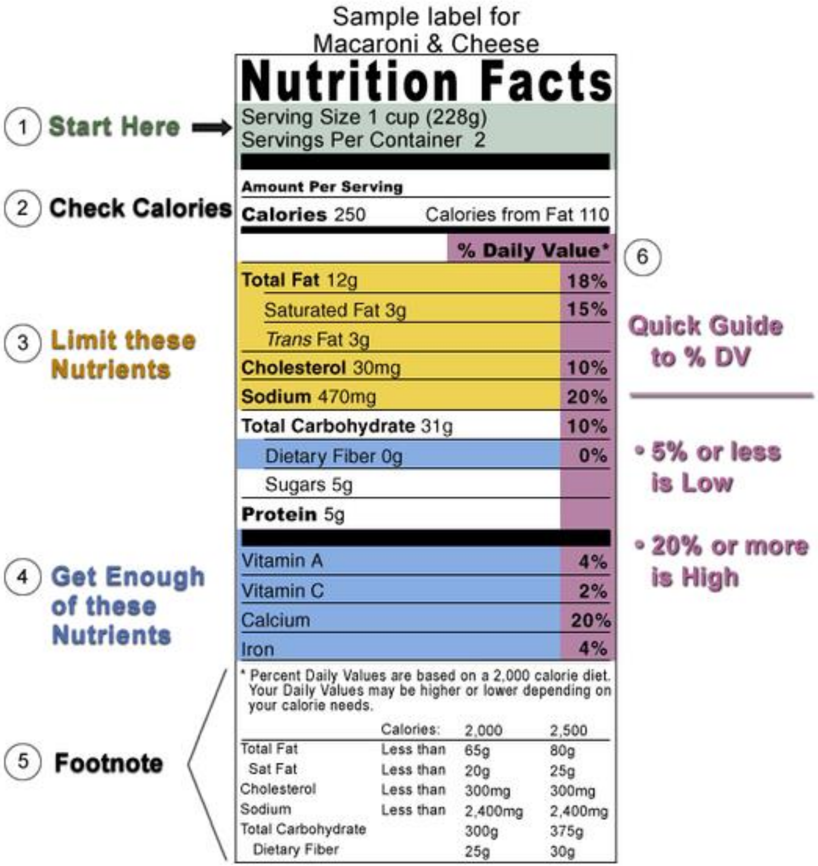
Which Specification Technology to Choose?

Wait... let us consider semantics first!

Ontology

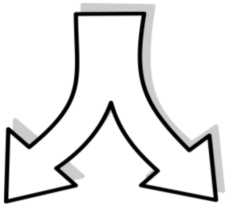
Focusing on *semantics*.

- Domain
- Expressiveness
- Executable or not
- Non-determinism
  - Ex: throttle is in range $t_{min} \dots t_{max}$
- Abstraction mechanisms
- Validation possibilities
- Tool processing possibilities

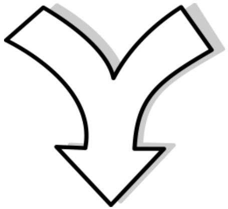


Two strands of work: Evaluation & Step Change in Existing Process

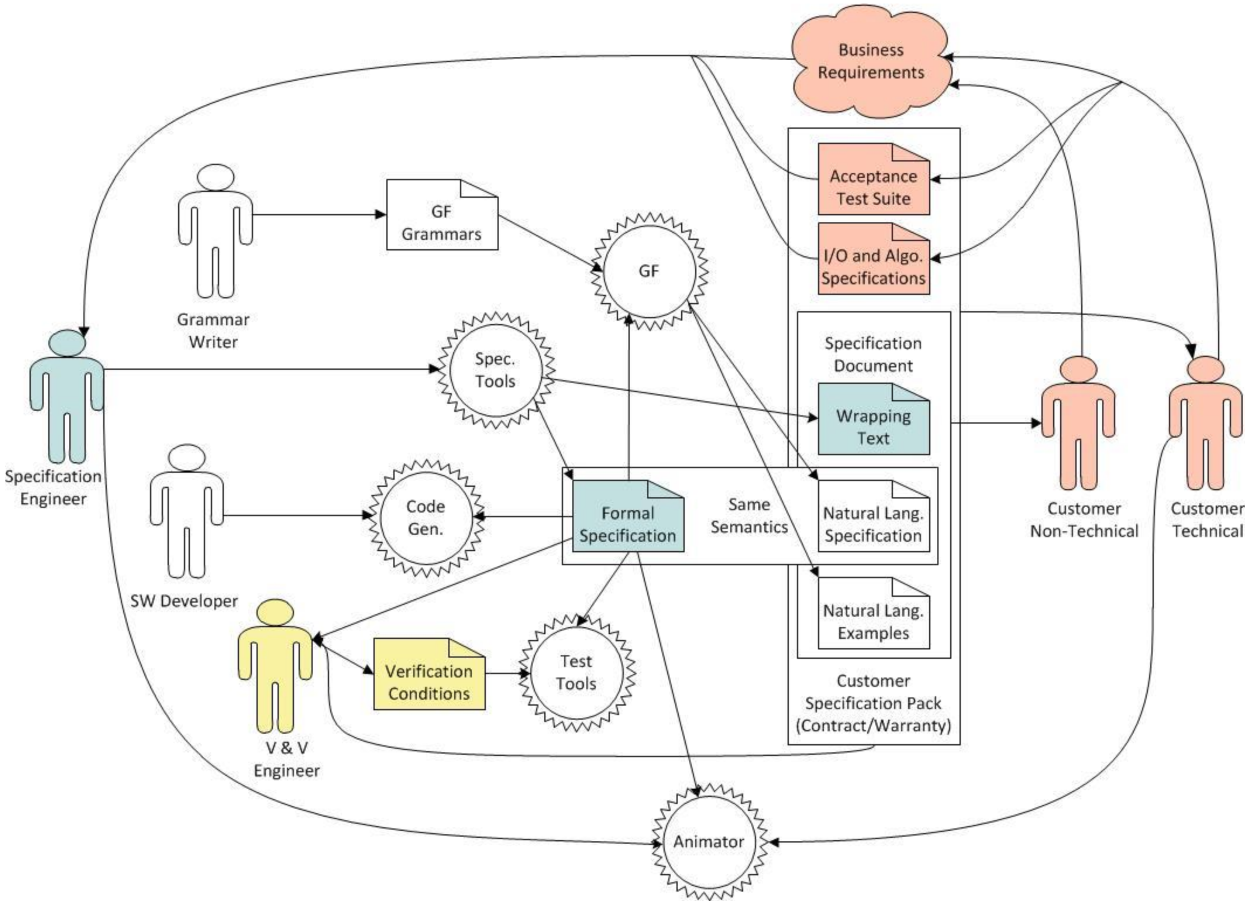
1. Evaluation of ABZ languages and tools
   - No silver bullet!
   - Challenging evaluation questions: records, editors, tool maturity, structuring, refinement
   - Longer term evaluation and community effort



2. Step change in risk and cost improvement
   - Multilingual specifications
   - Natural language
   - Automatically generated V&V monitors
   - Z and fuzz based



Specification Solution Architecture



13 Years Ago in Another Verification Tool Building Team...

An example OCL specification for verification of a JavaCard program in the KeY proof system (Johannison 2005):

```
context OwnerPIN
def: let tryCounter = self.triesLeft->at(1)

context OwnerPIN::check(pin: Sequence(Integer),
          offset: Integer, length: Integer): Boolean
post: self.tryCounter = 0 implies result = false
post: (self.tryCounter > 0 and pin <> null and offset >= 0 and length >= 0
          and offset+length <= pin->size()
          and Util.arrayCompare(self.pin, 0, pin, offset, length) = 0
      ) implies (result = true and self.isValidated() and tryCounter = maxTries)
post: (self.tryCounter > 0 and not (pin <> null and offset >= 0 and length >= 0
             and offset+length <= pin->size()
             and Util.arrayCompare(self.pin, 0, pin, offset, length) = 0)
      ) implies (not self.isValidated() and self.tryCounter = tryCounter@pre-1 and
          (( not excThrown(java::lang::Exception) and result = false)
           or excThrown(java::lang::NullPointerException)
           or excThrown(java::lang::ArrayIndexOutOfBoundsException)))
```

Fig. 1. OCL specification from the Java Card API

First Translation Attempt

In Fig. 2 we show the translation of the OCL specification produced by an earlier version of our system. The English text is basically correct, but it is clumsy and very hard to read.

for the class OwnerPIN introduce the following definition : the tryCounter is defined as the element at index 1 of the triesLeft of the ownerPIN for the operation check (pin : Seq(Integer) , offset : Integer , length : Integer) : Boolean of the class javacard::framework::OwnerPIN the following holds : the following postconditions should hold : (*) if the tryCounter of the ownerPIN is equal to 0 , the result is equal to false (*) if the tryCounter of the ownerPIN is greater than 0 and pin is not equal to null and offset is at least 0 and length is at least 0 and offset plus length is at most the size of pin and the query arrayCompare (the pin of the ownerPIN , 0 , pin , offset , length) to Util is equal to 0 , the result is equal to true and the query isValidated ( ) holds for the ownerPIN and the tryCounter of the ownerPIN is equal to the maxTries of the ownerPIN (*) if the tryCounter of the ownerPIN is greater than 0 and it is not the case that pin is not equal to null and offset is at least 0 and length is at least 0 and offset plus length is at most the size of pin and the query arrayCompare (the pin of the ownerPIN , 0 , pin , offset , length) to Util is equal to 0 , it is not the case that the query isValidated ( ) holds for the ownerPIN and the tryCounter of the ownerPIN is equal to the tryCounter of the ownerPIN at the beginning of the Operation minus 1 and it is not the case that an exception is thrown and the result is equal to false or a nullPointerException is thrown or an arrayIndexOutofBoundsException is thrown

Fig. 2. Translation of OCL specification (before)

Translation to English of the OCL specification (Johannison 2005)

ALTRAN

Improved Translation

for the class **OwnerPIN** introduce the following definition :

– the try counter is defined as the element at index 1 of the `triesLeft` attribute

for the operation **check (pin : Sequence(Integer) , offset : Integer , length : Integer) : Boolean** of the class **javacard::framework::OwnerPIN** ,
the following post-conditions should hold :

– if the try counter is equal to 0 then this implies that the result is equal to false
– if the following conditions are true
  - the try counter is greater than 0
  - *pin* is not equal to null
  - *offset* is at least 0
  - *length* is at least 0
  - *offset* plus *length* is at most the size of *pin*
  - the query `arrayCompare ( the pin , 0 ,` *`pin`* `,` *`offset`* `,` *`length`* `)`[1] on `Util` is equal to 0

  then this implies that the following conditions are true
  - the result is equal to true
  - this owner PIN is validated
  - the try counter is equal to the maximum number of tries
– if the try counter is greater than 0 and at least one of the following conditions is not true
  - *pin* is not equal to null
  - *offset* is at least 0
  - *length* is at least 0
  - *offset* plus *length* is at most the size of *pin*
  - the query `arrayCompare ( the pin , 0 ,` *`pin`* `,` *`offset`* `,` *`length`* `)`[2] on `Util` is equal to 0

  then this implies that the following conditions are true
  - this owner PIN is not validated
  - the try counter is equal to the previous value of the try counter minus 1
  - at least one of the following conditions is true
    * an exception is not thrown and the result is equal to false
    * a null pointer exception is thrown
    * an array index out of bounds exception is thrown

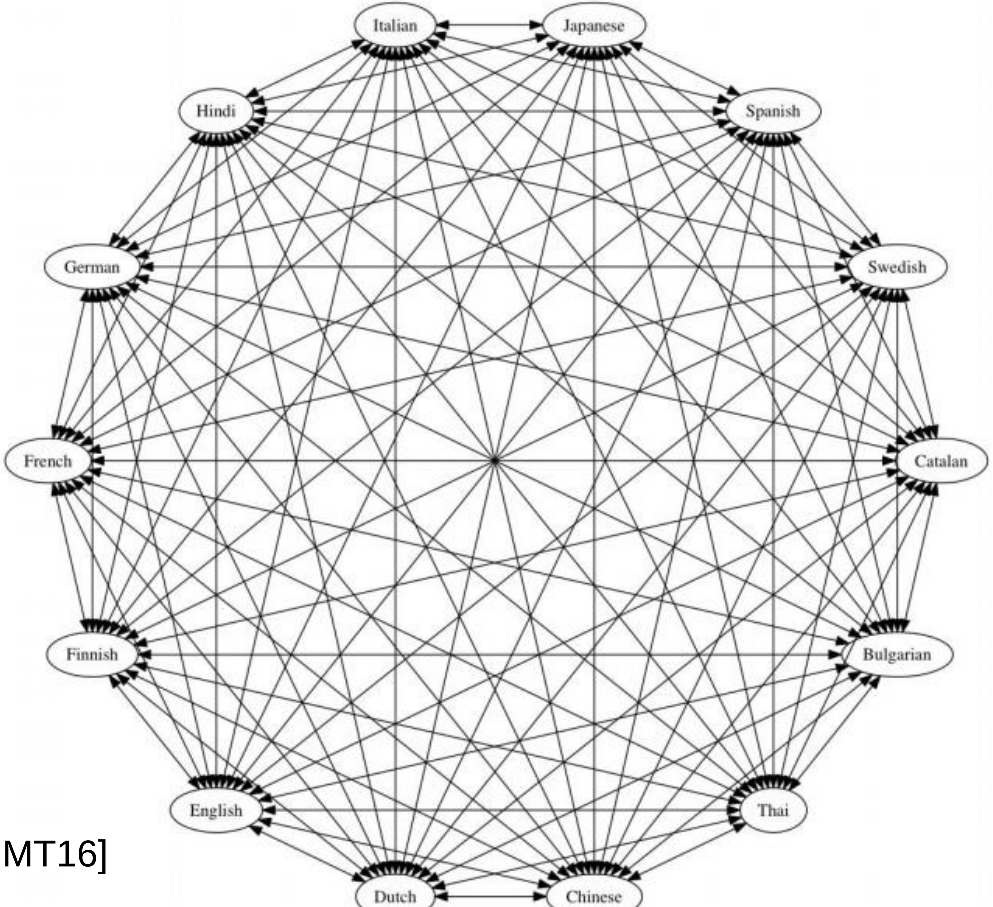
Trade-Offs in Natural Language Translation Techniques

1. Statistical methods
   - Consumer-oriented
   - Wide coverage
   - Imprecise
2. Rule-based methods
   - Producer-oriented
   - Grammar-based
   - Restricted
3. Ad-hoc methods
   - We are used to writing parsers and linearisers… can't be that difficult?

Grammatical Framework

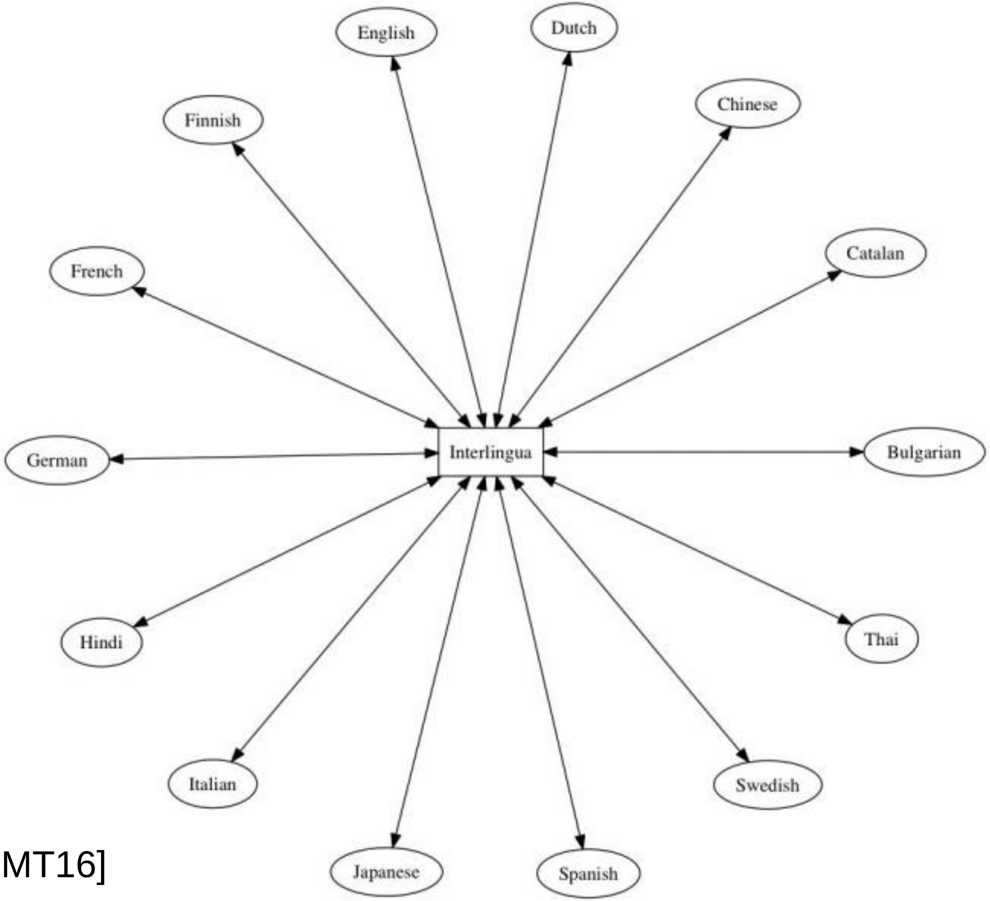
- Multilingual grammar formalism
- Based on type theory and functional programming
- Multilingual grammar = abstract syntax + concrete syntaxes
- Parsing: from string to abstract syntax
- Linearization: from abstract syntax to string
- Translation = parsing followed by linearization
- Abstract syntax is interlingua

Grammatical Framework Mission…



Aarne Ranta [FINMT16]

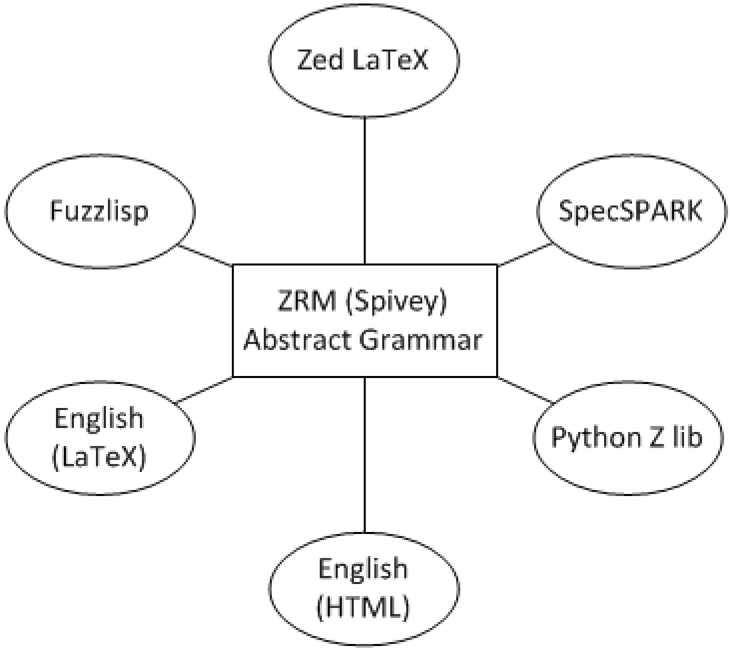
...is Grammar Engineering Without Tears



Aarne Ranta [FINMT16]

Multilingual Specifications – Progress and Future Work

- Abstract core grammar (GF) close to Spivey’s ZRM
- Concrete grammars:
  - Fuzzlisp
  - LaTeX
  - English
  - V&V Monitors in Python/SpecSPARK



Some Concluding Remarks

- Correctness by Construction is still important and hot
  - Keep teaching and improving “text-book style” formal verification

- How do we recruit and train specifiers?
  - How could we make ABZ courses for lawyers, linguists, chemical engineers, astrophysicists…?

05.

Resources

SPARK Resources & Getting Started

- SPARK 2014: http://www.spark-2014.org/
- GAP - GNAT Academic Program
  - Open-source, GPL release of SPARK tools
  - https://www.adacore.com/academia
  - Support from SPARK team for faculty
- Getting Started
  - Download the tools: http://libre.adacore.com/download/
  - User Guide: http://docs.adacore.com/spark2014-docs/html/ug/, chapter 5, SPARK tutorial, is a good start
  - SPARK 2014 Reference Manual: http://docs.adacore.com/spark2014-docs/html/lrm

Some More References

- Tokeneer case study: https://www.adacore.com/tokeneer
- Grammatical Framework: https://www.grammaticalframework.org/
- Digital Grammars: http://www.digitalgrammars.com/
- A. Ranta's FINMT 2016 presentation: https://blogs.helsinki.fi/language-technology/files/2016/09/FINMT2016-aarne-ranta.pdf
- Fuzz typechecker for Z: http://spivey.oriel.ox.ac.uk/corner/Fuzz_typechecker_for_Z
- High Integrity Agile, our take: https://cacm.acm.org/magazines/2017/10/221329-what-can-agile-methods-bring-to-high-integrity-software-development/abstract

altran