
# Reproducibility, Proofs and Domain Specific Languages

Patrik Jansson Chalmers University of Technology

2016-04-06

# Who am I?

- Patrik Jansson, Prof. of Comp. Sci., Chalmers, Sweden
  - Programming languages
  - Software technology
- Two EU projects about "Global Systems Science (GSS)"
  - CoeGSS.eu
  - GRACeFUL-project.eu
  - a "tweet-sized definition" (from 2013):

    *GSS is about developing systems, theories, languages and tools for computer-aided policy making with potentially global implications.*

# Why reproduce?

- Why is reproducibility a good thing?
  - trust

# Why reproduce?

- Why is reproducibility a good thing?
  - trust
  - correctness

# Why reproduce?

- Why is reproducibility a good thing?
  - trust
  - correctness
  - independent checking

# Why reproduce?

- Why is reproducibility a good thing?
  - trust
  - correctness
  - independent checking
- Terminology:
  - From "12 Rs, de Roure, 2010"

    *Reproducible - enough information for an independent experiment to reproduce the results.*

# Why reproduce?

- Why is reproducibility a good thing?
  - trust
  - correctness
  - independent checking
- Terminology:
  - From "12 Rs, de Roure, 2010"

    > *Reproducible - enough information for an independent experiment to reproduce the results.*

  - Let's call "Enough information [...]" a **specification**
  - of an experiment = **implementation**

## Specification, implementation and proof

- "Enough information [...]" a **specification**
- of an experiment = **implementation**

## Specification, implementation and proof

- "Enough information [...]" a **specification**
- of an experiment = **implementation**
- When is the implementation *correct* w.r.t. the spec.?
  - tradition science: *trust*
  - for maths and software: *proof*

## Specification, implementation and proof

- "Enough information [...]" a **specification**
- of an experiment = **implementation**
- When is the implementation *correct* w.r.t. the spec.?
  - tradition science: *trust*
  - for maths and software: *proof*
- If we have a formal system (a logic)
  - we may give a proof of correctness
  - (moving trust)

## Machine checked proof

- Dependent type theory
  - can express many specifications
  - and implementations
  - and proofs
- in the same formal system.

Proof assistants: Coq, Agda, Idris, ...

## Back to some of all the “Rs” (for software)

1. Repeatable: needs an implementation (source code or executable)

## Back to some of all the “Rs” (for software)

1. Repeatable: needs an implementation (source code or executable)
2. Reproducible: needs a specification

## Back to some of all the “Rs” (for software)

1. Repeatable: needs an implementation (source code or executable)
2. Reproducible: needs a specification
3. Reliable: strengthened by proofs

## Back to some of all the “Rs” (for software)

1. Repeatable: needs an implementation (source code or executable)
2. Reproducible: needs a specification
3. Reliable: strengthened by proofs
4. Reusable: helped by a language for combining experiments

## Back to some of all the “Rs” (for software)

1. Repeatable: needs an implementation (source code or executable)
2. Reproducible: needs a specification
3. Reliable: strengthened by proofs
4. Reusable: helped by a language for combining experiments

Risk: formal proofs improve 3. but damage 4.

## A short story about Dee and Foo

Actors: Dee the Domain Expert & Foo the Formaliser

Starting point: Dee has an experiment + informal description

Dee + Foo work on formalisation.

Happy ending?: formal spec. and computer proof of correctness

## A short story about Dee and Foo

Actors: Dee the Domain Expert & Foo the Formaliser

Starting point: Dee has an experiment + informal description

Dee + Foo work on formalisation.

Happy ending?: formal spec. and computer proof of correctness

Drama:

- Dee cannot understand the formal spec.
- Foo cannot understand the domain.

## A Domain Specific Language to the rescue

Develop a Domain Specific Language for expressing the specification

- Dee can use familiar terminology
- Foo understands enough to construct the proof

## A Domain Specific Language to the rescue

Develop a Domain Specific Language for expressing the specification

- Dee can use familiar terminology
- Foo understands enough to construct the proof

Or even better,

- Foo develops a library of proof components (like LEGO-bricks)
- now Dee can "assemble" the proof herself!

## A Domain Specific Language to the rescue

Develop a Domain Specific Language for expressing the specification

- Dee can use familiar terminology
- Foo understands enough to construct the proof

Or even better,

- Foo develops a library of proof components (like LEGO-bricks)
- now Dee can "assemble" the proof herself!

This is the ideal we strive for:

- not merely correct programs,
- nor even proven correct programs;
- we want proof done against a specification that is naturally expressed for a domain expert.

## Summary

Key points:

- experiment = implementation
- reproducibility requires also a *specification*
- *provability* is stronger than reproducibility
- dependently typed languages can express spec., impl. and proof
- formal specifications can introduce a "comprehension gap"
- Domain-specific languages (DSLs) can be used to bridge the gap

This is the ideal we strive for:

- not merely correct programs,
- nor even proven correct programs;
- we want proof done against a specification that is naturally expressed for a domain expert.