CISC/CMPE 422/835:
Formal Methods in Software Engineering

Juergen Dingel

Fall 2022

Lecture:

- Specification vs implementation
- The power and utility of formal specifications
- Intro to Alloy

What is a specification?

- American Heritage Dictionary:

> *"A detailed, exact statement of particulars, especially a statement prescribing materials, dimensions, and quality of work for something to be built, installed, or manufactured"*

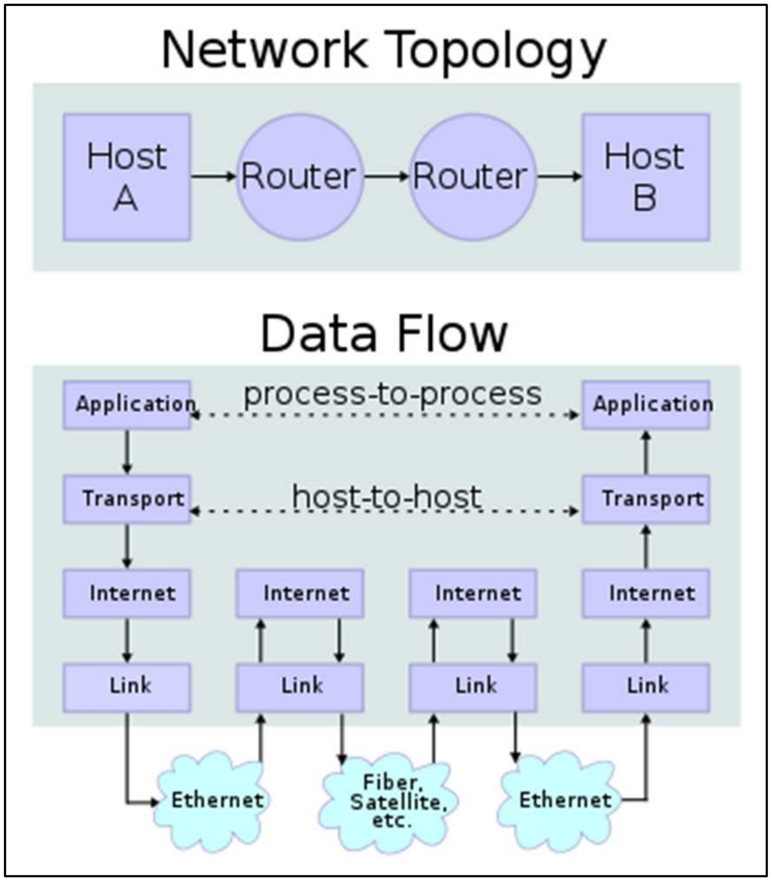
- For software, e.g.,
  - **input/output behaviour** of a system, component, or method
  - a **class invariant**
  - the **description of interactions** necessary for the execution of a protocol
  - **structure of and relationships** between objects
  - descriptions of **allowed resource consumption and expected performance**

Desirable features of specifications

- Correct
- As precise and detailed as necessary
- As abstract and unconstraining as possible

=> Declarative rather than operational

- what? vs how?



Example: Internet Protocol (IP)

DARPA. Internet Protocol Specification (RFC 791). Sept 1981. https://tools.ietf.org/html/rfc791

V.G. Cerf. In praise of under-specification? CACM 60(8):7-7. Aug 2017. https://dl.acm.org/citation.cfm?id=3110531

en.wikipedia.org/wiki/Internet_protocol_suite

Specifications vs implementations

- Specifications
  - *“as abstract as possible, as concrete as necessary”*
  - “declarative” rather than “operational”

  => may not be executable (efficiently)
- Implementation
  - executable
- The promise of Prolog

```
member(X, [X|_]).
member(X, [Y|Ys]) :- X=/=Y, member(X, Ys).
```

Specification languages

1. Non-formal: Natural language

- Pros
  - expressive
  - no/little training required
- Cons
  - often imprecise

> *"Aircraft that are non-friendly and have an unknown mission or the potential to enter restricted air-space within 5 minutes shall …"*

- limited opportunity for (automated) analysis due to its complexity (e.g., implicit context knowledge)

The (sometimes hidden) complexity of natural language

- E.g., informal descriptions of requirements may implicitly assume **context knowledge**:

Shoes must be worn!

Dogs must be carried!

What is the problem?

[M. Jackson. *Software Specifications and Requirements: a lexicon of practice, principles and prejudices*. Addison-Wesley, 1995.]

The (sometimes hidden) complexity of natural language

- Informal descriptions of requirements may implicitly assume **context knowledge**:

Shoes must be worn!

Dogs must be carried!

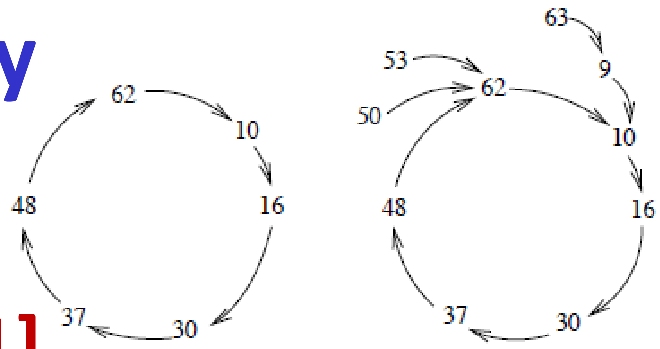
$$\forall x : \text{Person}.\ \exists y : \text{Shoes}.\ \text{Owns}(x,y) \wedge \text{Wears}(x,y)$$

$$\forall x : \text{Person}.\ \exists y : \text{Dog}.\ \text{Owns}(x,y) \wedge \text{Carries}(x,y)$$ ☹

$$\forall x : \text{Person}.\ \forall y : \text{Dog}.\ \text{Owns}(x,y) \rightarrow \text{Carries}(x,y)$$ ☺

Analyzing informal models
in a meaningful way typically impossible

The (sometimes hidden) complexity of software



Chord: Distributed hash table [Chord01]

[Chord01] Stoica, Morris, Karger, Kaashoek, Balakrishnan. *"Chord: A scalable peer-to-peer lookup service for Internet applications"*. SIGCOMM. 2001.

- *"3 features that distinguish Chord from many other peer-to-peer lookup protocols are its simplicity, provable correctness, and provable performance"*
- Papers present properties, invariants and proofs
- 4th most-cited paper in CS for years (CiteSeer)
- 2011 SIGCOMM Test-of-Time Award

"Unfortunately, the claim of correctness is not true. The original specification […] does not have eventual reachability, and not one of the seven properties claimed to be invariants […] is actually an invariant."

"For complex protocols such as Chord, there is every reason to use lightweight modeling as a design and documentation tool"

P. Zave. Various papers on www.pamelazave.com/chord.html

But, once a problem is formalized impressive things are possible

- Impress your friends by solving every Sudoku puzzle

```
 . . . | . . . | . 1 .
 4 . . | . . . | . . .
 . 2 . | . . . | . . .
-----------------------
 . . . | . 5 . | 4 . .
 . . 8 | . . . | 3 . .
 . . 1 | . 9 . | . . .
-----------------------
 3 . . | 4 . . | 2 . .
 . 5 . | 1 . . | . . .
 . . . | 8 . 6 | . . .
```

www.spass-prover.org

?

```
6,9,3|7,8,4|5,1,2,
4,8,7|5,1,2|9,3,6,
1,2,5|9,6,3|8,7,4,
------------------
9,3,2|6,[illegible],1|4,8,7,
5,6,8|2,4,7|3,9,1,
7,4,1|3,9,8|6,2,5,
------------------
3,1,9|4,7,5|2,6,8,
8,5,6|1,2,9|7,4,3,
2,7,4|8,3,6|1,5,9,
```

Note: This is not a toy!

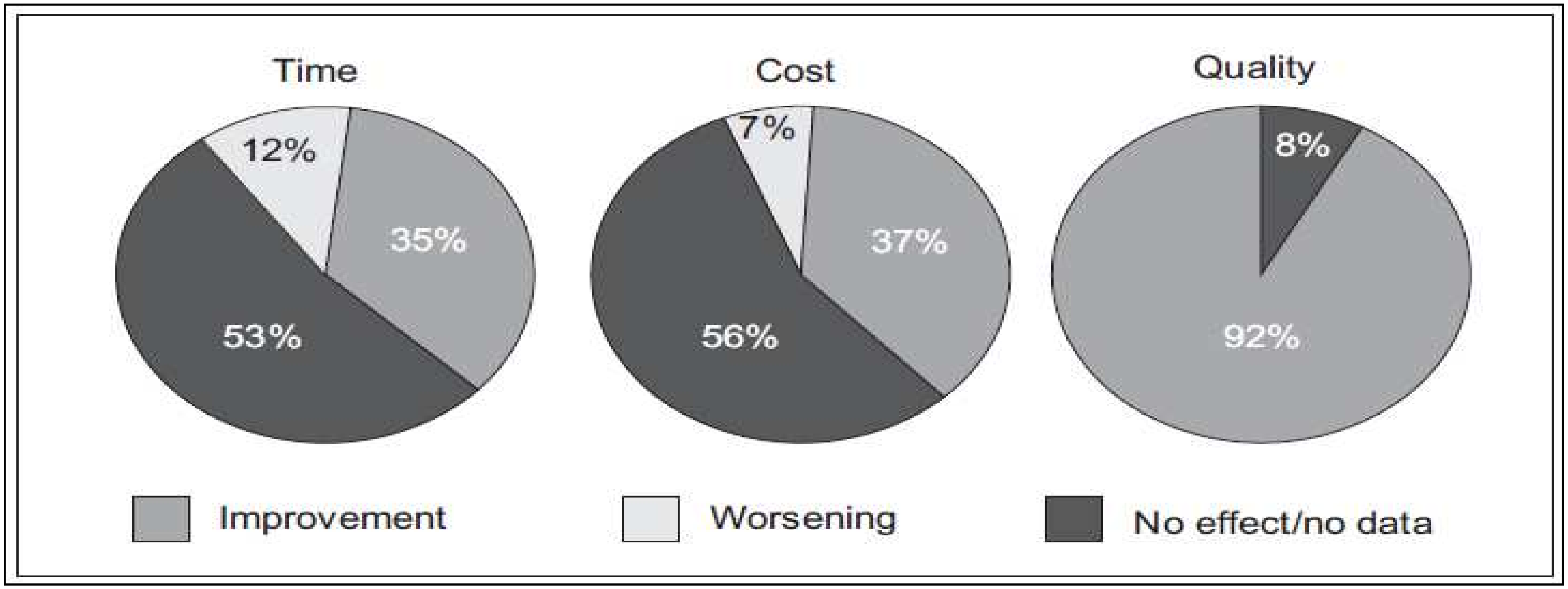
More than 10^{38} possibilities, i.e., size of state space > 10^{38}

Number of cells in human body: 10^{13}

Number of atoms in universe: 10^{80}

But, once a problem is formalized amazing things are possible (cont'd)

- Survey of 62 int'l FM projects
  - **Domains:** Real-time, distributed & parallel, transaction processing, high-data volume, control, services



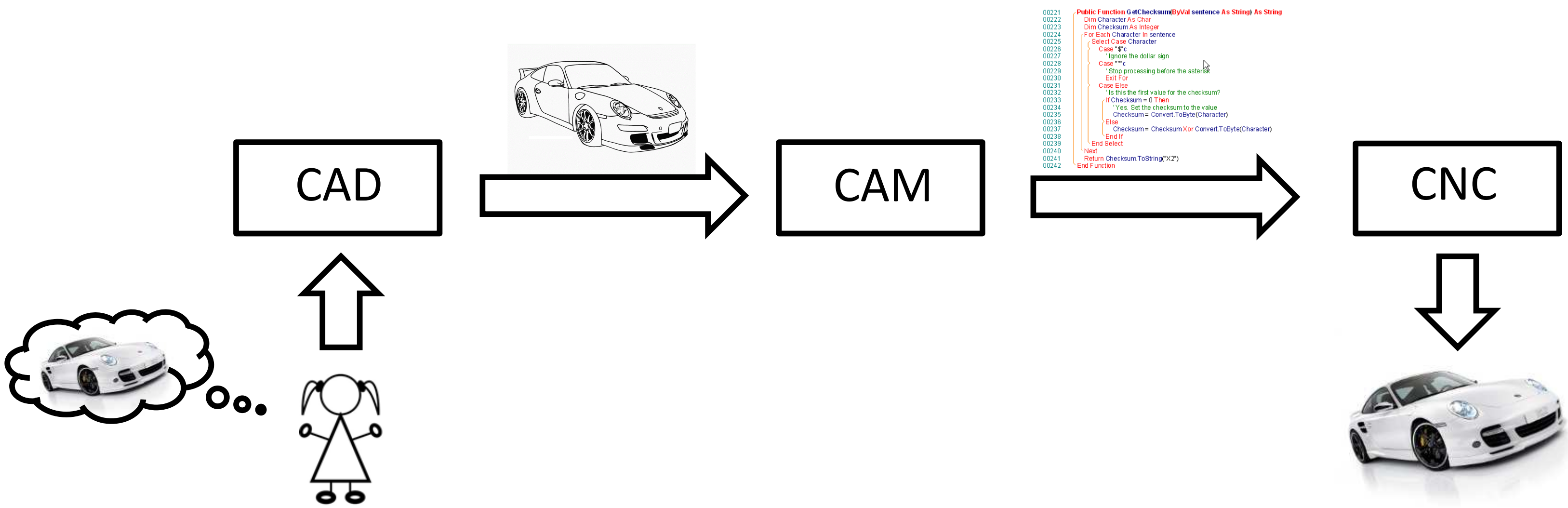
[Radio Technical Commission for Aeronautics (RTCA). DO-333: Formal Methods Supplement to DO-178C and DO-278A.

[Woodcock et al. Formal Methods: Practice and Experience. ACM Computing Surveys 41(4). 2009]

But, once a problem is formalized amazing things are possible (cont'd)

Mechanical design from about 1972: CAD/CAM

1. Create drawings w/ computer (CAD)
2. From drawing, computer automatically generates program to drive milling and CNC machines (CAM)

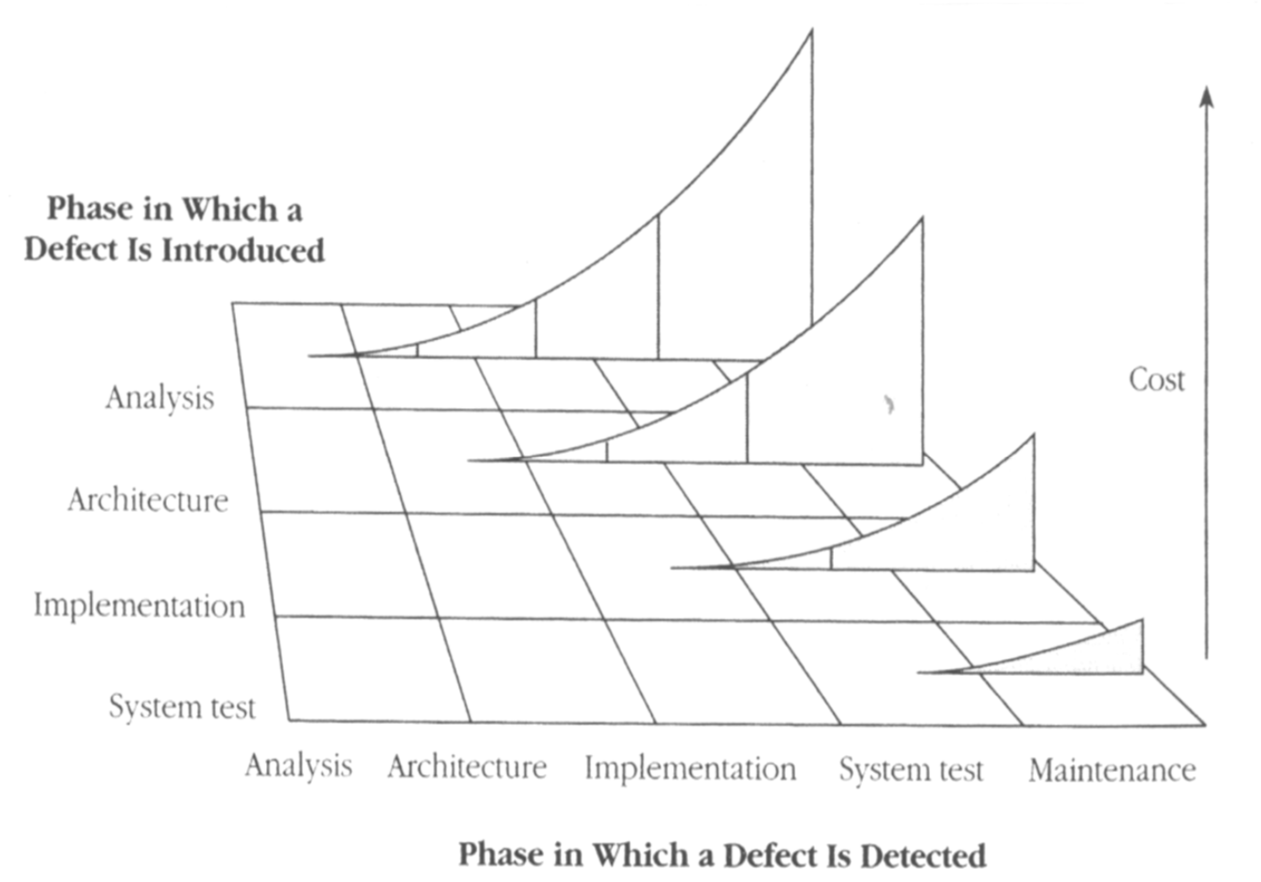


=> much better analysis capabilities and productivity

=> CAD/CAM has revolutionized manufacturing

Analysis of specifications

- Analysis of specifications for correctness, consistency, (un)desirable properties can pay off



Specification languages

1. Non-formal

- Natural language

2. Semi-formal

- UML

3. Formal

- precisely defined semantics
- mechanisms for abstraction, analysis, modularity, reuse
- Used for
  - safety-critical systems, but
  - not necessarily (e.g., state machines)
- Examples:
  - 1) Propositional and Predicate logic, 2) Alloy,
  - Z, B, VDM, …

Formal specification languages

1. Propositional and/or predicate logic

add: $(String \times Object \times \wp(String \times Object)) \rightarrow \wp(String \times Object)$ *such that*

$$
\begin{array}{l}
\forall\, d : \mathcal{P}(String \times Object).\, \forall\, d' : \mathcal{P}(String \times Object).\, \forall\, key : String.\, \forall\, val : Object. \\
\quad d' = add(key, val, d) \quad \leftrightarrow \\
\qquad ((\neg\ \exists\, v : Object.\langle key, v\rangle \in d) \quad \rightarrow \quad d' = d \cup \{\langle key, val\rangle\}) \wedge \\
\qquad (\exists\, v : Object.\langle key, v\rangle \in d. \quad \rightarrow \quad d' = d - \langle key, v\rangle \cup \langle key, val\rangle)
\end{array}
$$

- **Pros**
  - expressive, well-studied, formal, good tool and analysis support
- **Cons**
  - lack of modularity mechanisms
  - predicate logic is undecidable

2. Alloy

Alloy: What for?

1. Formal approach to describing structure and relationships between objects

But, why not use UML
(Class Diagrams & Object Diagrams)?

2. Analyze specifications automatically with respect to
   1. Correctness
   2. Consistency
   3. (Un-)desirable properties

Alloy: core ingredients

Alloy, the language:

- Declarative
- First-order logic + relational calculus
- *"Everything is a relation!"*

Alloy, the analysis:

- Automatic
- Satisfiability solving (SAT)

Alloy, the tool:

- Stable, usable, "light-weight"

Less is More

If Done Right

SAT

- Quintessential hard problem
  - First problem to be proven NP-complete [Cook 1971]
  - Lots of other common problems can be solved using SAT
- Hard, but not impossible
  - Heuristical SAT-solvers solve problems w/ ~1M variables, enough to deal w/ many practical problems
    - HW verification
      - E.g., circuit for z=x/y where x,y,z are

        128-bit floats: 2^{256} combinations
      - Non-solution: manual
      - Solution: random-constraint test gen.
    - SW verification
    - Planning, scheduling

From http://alloytools.org/tutorials/day-course