Formal Methods in Software Engineering

2026/27

Formal Methods

- Rigorous/math-supported approaches to software development
- A staple of the computing curricula at UMinho
  - Algorithms and Complexity
  - Program Calculation
  - Computational Logic
  - …

Featured

Formal Methods Are Having a Moment — Why Your Development Team Should Care in 2026

column

DOI: 10.1145/3819084

BY FERNANDA GRACIOLLI AND NADA AMIN

You Don't Know Jack About Formal Verification

What can you confidently guarantee about your software?

So, why haven't you used it? Because, historically, the cost of writing proofs dwarfed the cost of writing the code. You would state an "obvious" property and then spend days coaxing a proof assistant into accepting it. The tools were slow, the process was painstakingly tedious and required Ph.D.-level skill, and this all produced error messages that were inscrutable. The guarantee was real, but the price was too high, which is why formal verification lived almost exclusively in avionics, chip design, nuclear systems, and cryptographic protocols—domains where a bug costs lives or fortunes. For everyone else, the reasonable conclusion was: Tests are good enough.

Now, Formal Verification Is AI's Problem

AI proposes implementation candidates, and a deterministic, mechanical process checks each candidate's correctness. If the proof is wrong, the verifier rejects it and AI tries again. Trust in AI is reduced (to the specifications) because the verifier is an external authority. The human stays in the loop for the part that requires judgment: deciding *which* properties are worth guaranteeing and system design. Meanwhile, the machine handles the labor of producing a verifiably correct implementation.

This changes who and what formal verification is for. It's no longer just for safety-critical systems with the budget for specialized proof engineers. It's for anyone who has a property worth proving.

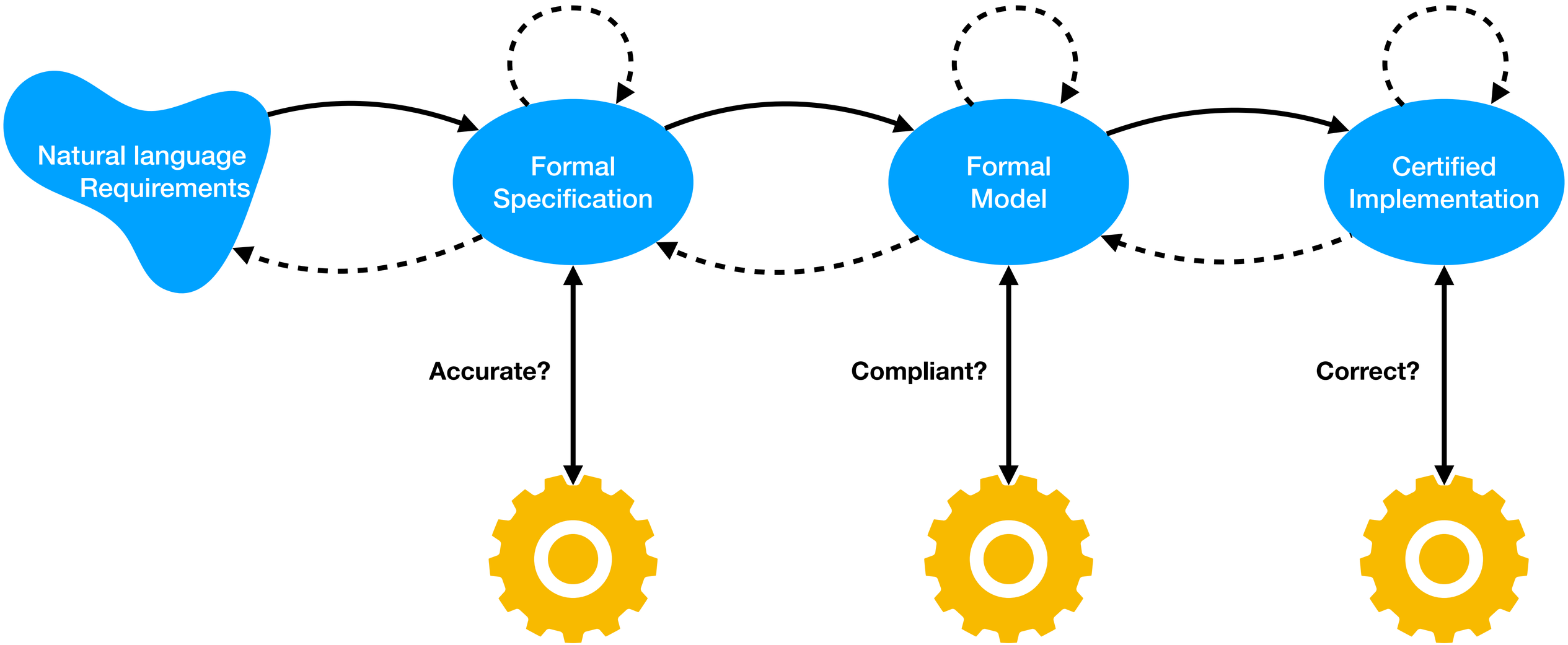
The Proof Is Only as Good as the Spec

Everything stated in this article so far assumes the spec is right. That's not a small assumption—it's the hardest part and the crux of formal verification. No tool can fully automate the design decisions *you* want your app to adopt. The verifier will ruthlessly enforce whatever you tell it to, but it can't tell you what to enforce. Brainstorming and planning with AI, along with spec-conflict audits during the formal spec-writing step, facilitates the design process.

Consider the example set forth in this article. The secrets-management bug is catchable only if someone has stated the subset invariant: Derived permissions must always match a subset of the granting permission's environments. If the spec had instead said something narrower—"finite permissions can contain only other finite permissions"—the proof would pass, but the guarantee wouldn't cover future permission representations. The invariant needs to be stated at the right level of abstraction, and the subsequent proof needs to prove it at the right level of depth. Getting both of these right requires a deep understanding of the domain, even if you never need to touch a formal spec or proof and your review is focused on the artifacts produced by the results of both.

Target audience

- Every software engineer should know a bit about FMs
  - MFES will teach you a bit more about FMs
- Some software engineers should know a lot about FMs
  - MFP will teach you a lot about FMs
- Numerous UMinho alumni in international careers as formal SW engineers


Natural language Requirements
Formal Specification
Formal Model
Certified Implementation
Accurate?
Compliant?
Correct?

Syllabus

- Logics for formal specification
- Automated proof techniques
- Formal software design with Alloy
- Deductive program verification with Why3

Lecturers

- Alcino Cunha (MAC)
  - alcino@di.uminho.pt, Ed7 1.18
- Jorge Sousa Pinto (JSP)
  - jsp@di.uminho.pt, Ed7 2.28
- Maria João Frade (MJF)
  - mjf@di.uminho.pt, Ed7 1.18
- Nuno Macedo (NM)
  - nmacedo@di.uminho.pt, Ed7 1.18




Wednesday

9h	T (MAC + JSP) CP2 0.05		
10h			
11h	TP1 (JSP) CP2 0.01	TP2 (MJF) CP3 0.06	TP3 (NM) CP1 2.26
12h			

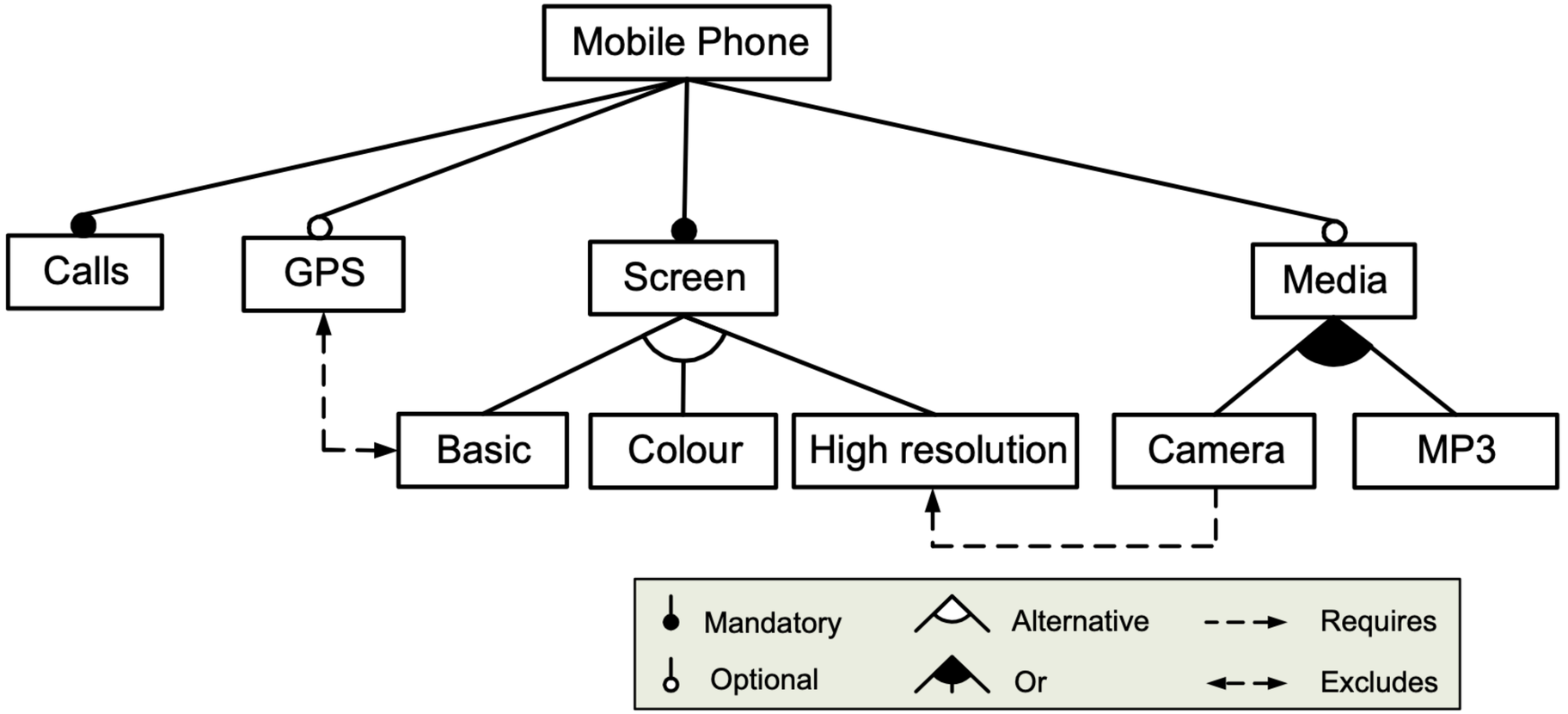
Assessment

- Continuous assessment
  - Practical exercises (20%) - 13 Oct, 3 Nov, 17 Nov, 22 Dec (blackboard)
  - Written test (80%) - 6 Jan
- Assessment by examination
  - Written exam (100%) - 27 Jan
- Final grades above 18 require developing a small project/challenge

Applications

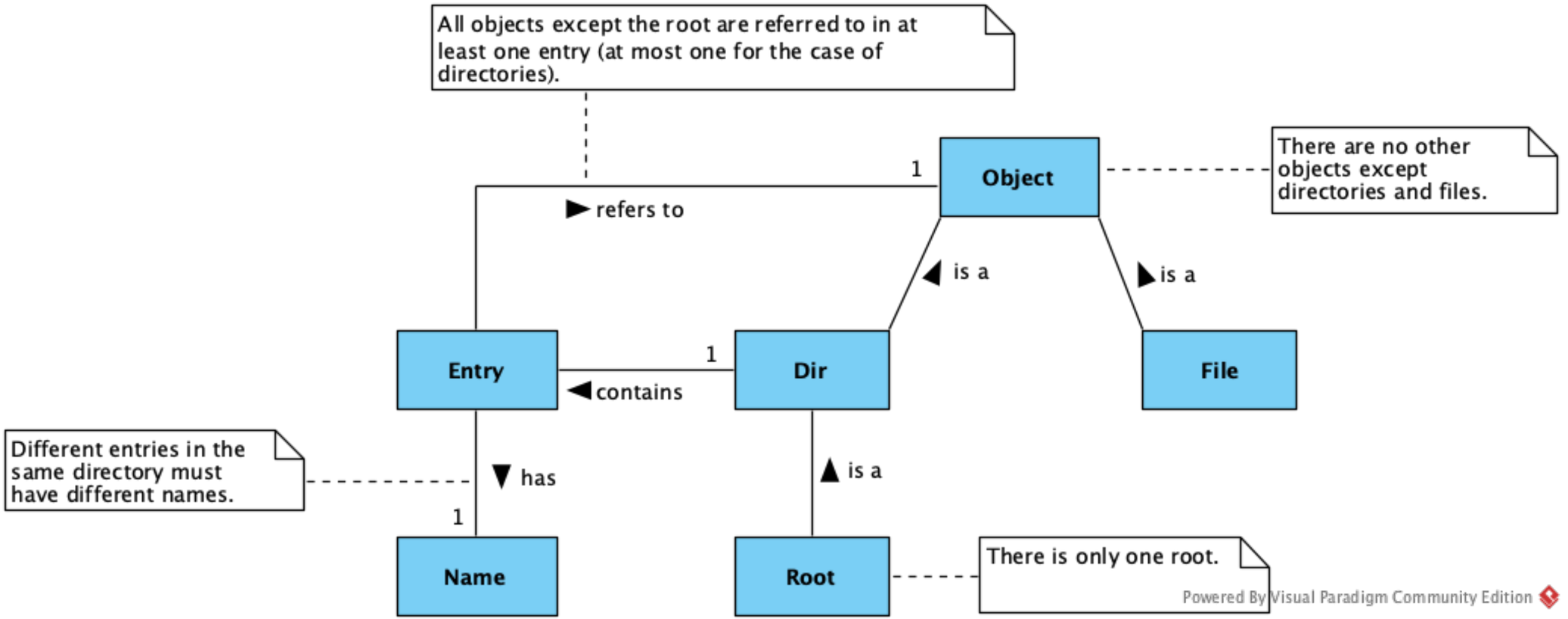
- Variability modelling
- Domain modelling
- Data-structure design
- App design
- Program testing
- Progam verification

Variability modelling

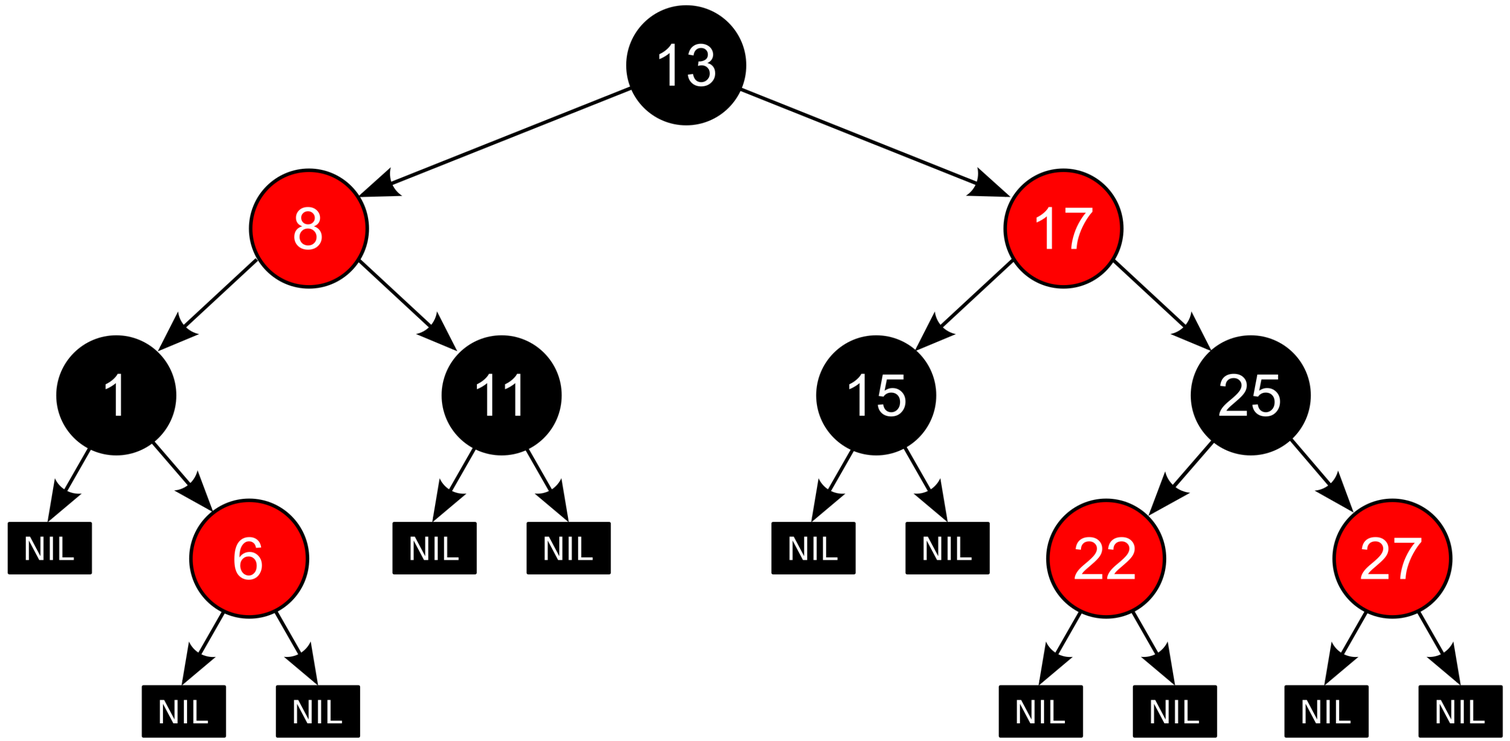


https://doi.org/10.1016/j.is.2010.01.001

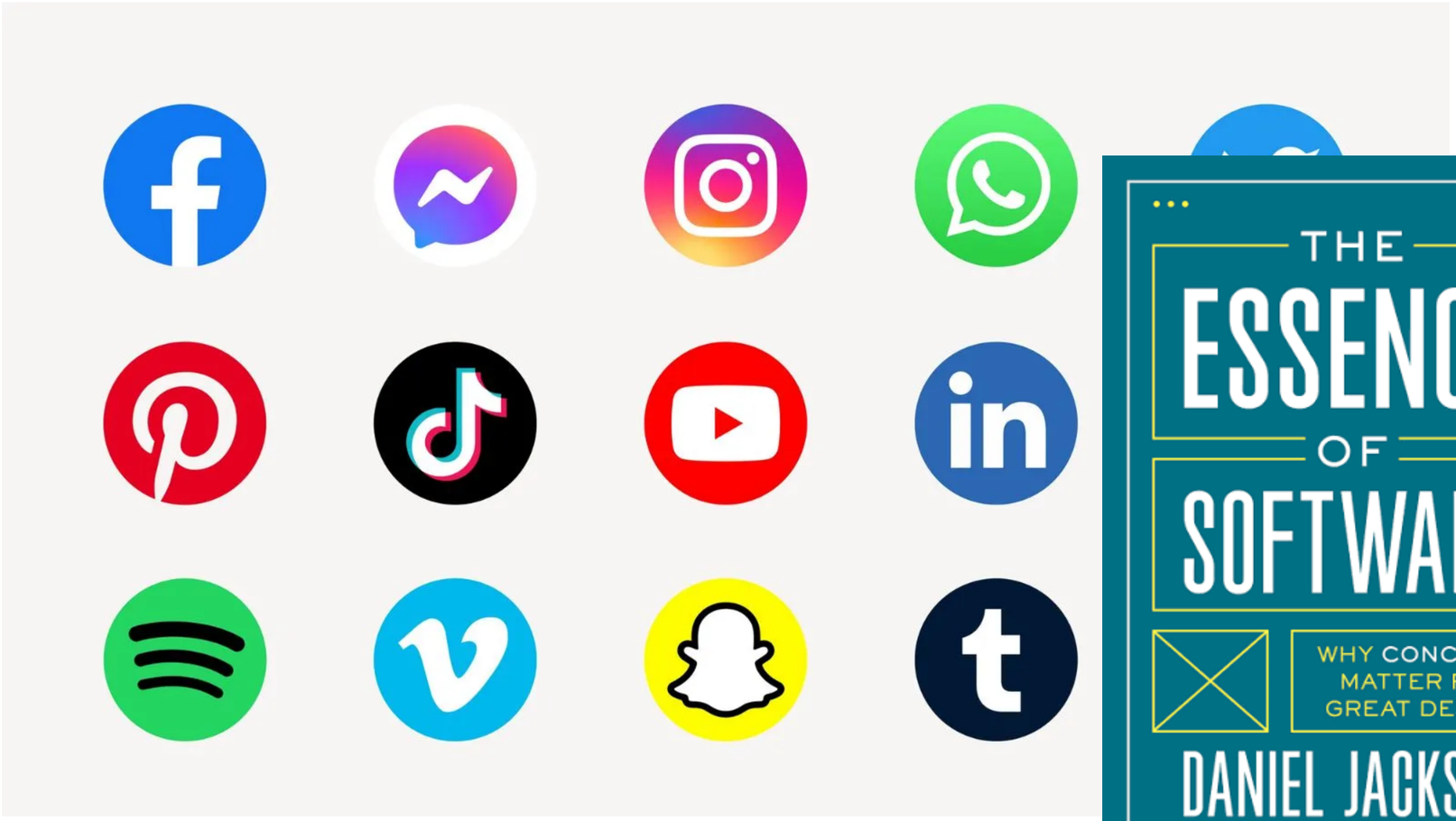
Domain modelling



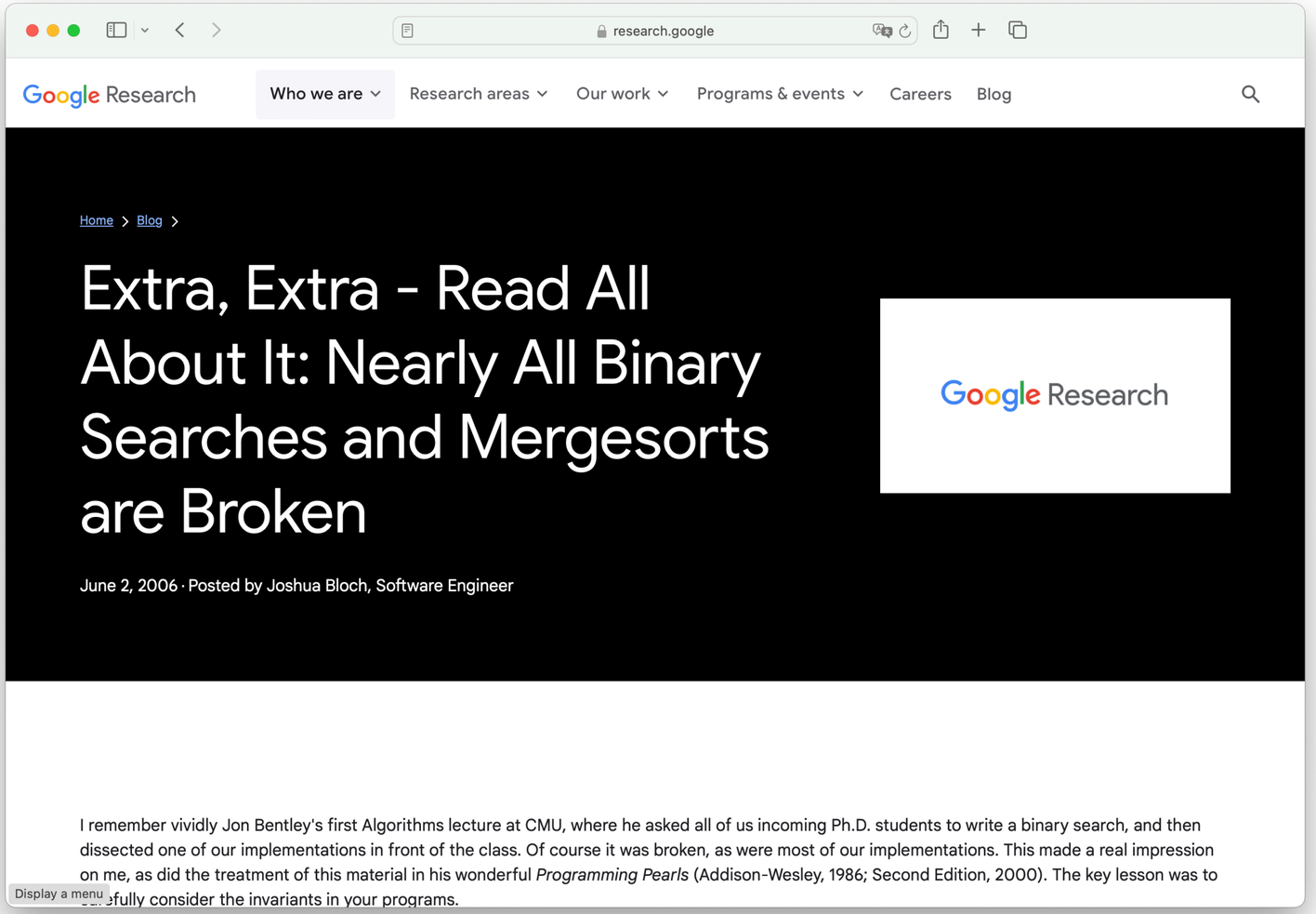
Data-structure design



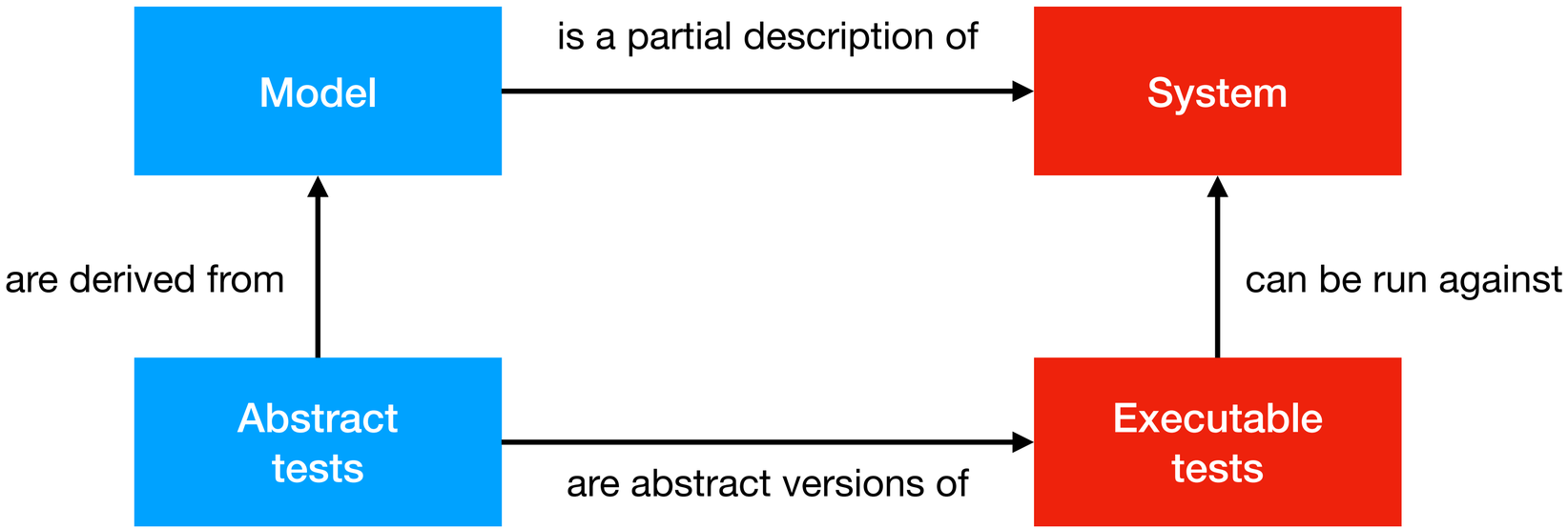
App design with *concepts*



Program verification



Program testing



https://haslab.github.io/MFES/