Here's a view of validating the correctness of software that takes into account the uncertainties in the real world that affect the behavior of AI and ML software.

CS445 / SE463 / ECE 451 / CS645

Software requirements specification & analysis

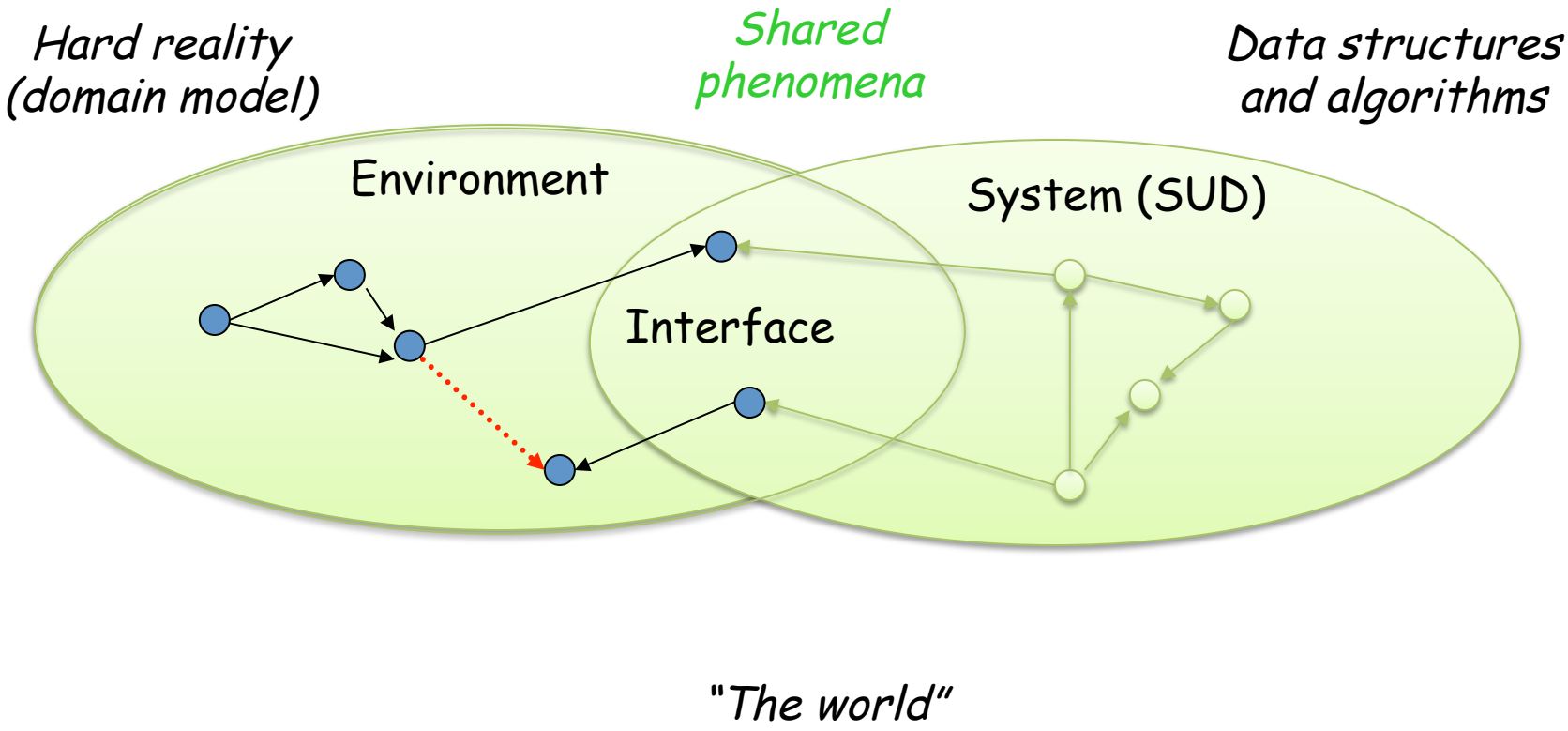
A reference model for requirements engineering

Fall 2015

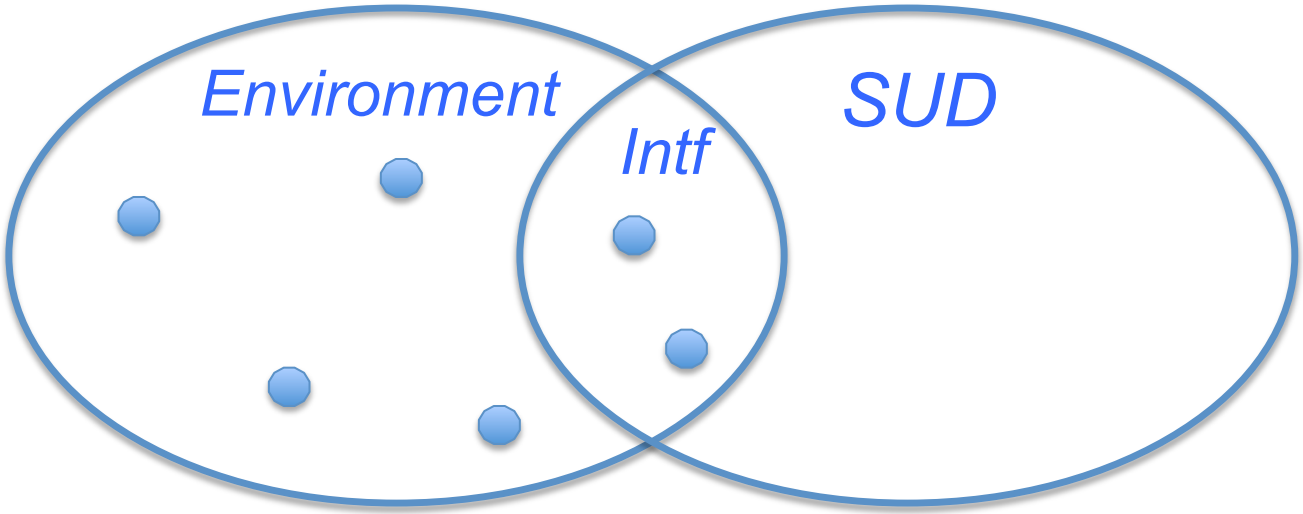
Mike Godfrey & Daniel Berry & Richard Trefler

RE Ref Model by Jackson, Zave, and Gunter*2

Reqs, specs, and programs



Reference model



R – Requirements live in ENV (incl. INTF)
S – Spec lives in INTF, describes behaviour of SUD
D – Domain knowledge lives in ENV (incl. INTF)

Reference model

- Thus, if we enlarge our model to include domain knowledge, then the following relationship must hold:

$$\mathbf{D, S \vdash R}$$

  - D is domain knowledge
  - S is the specification
  - R is the requirements

- The specification describes the behaviour of a system that realizes the requirements.
- The domain assumptions are needed to argue that any system that meets the specification (and that manipulates the interface phenomena) will satisfy the original requirements.

Zave-Jackson Validation Formula (ZJVF)

Uncertainty in "D, S ⊢ R"

- The formula D, S ⊢ R tries to be formal in the sense of describing what happens completely.
- One would expect computers and software and their combination to be formal in this sense.
- But, the real world intervenes to make this formula only a guideline and not an accurate, precise model.

Hidden: Uncertainty in "D, S ⊢ R"

- First, the real world *never* behaves as *any* model.
- Any model D is only an approximation.
- Generally, the simpler the model, the more of an approximation the model is, but the easier it is to prove things about the model.
- Modeling the real world accurately requires complexity to deal with all the weird exceptions.
- A mechanistic description generally has to be replaced by or tempered with a probabilistic model, e.g., 99.99% of drivers stop at a red light.

Hidden: Uncertainty in "D, S ⊢ R"

- At the lowest level, a CBS is mechanistic, e.g., a traffic light, the sqrt function, and can be modeled with a consistent S that is mechanistic, that always gives for any input the same answer that the CBS does.
- But floating point arithmetic is not the same as real numbers, and integer arithmetic suffers over & underflow.
- At higher levels, e.g., MS Word, an operating system, process control, etc., the CBS is so large that we cannot understand all of its code and all of its behavior. So, we begin to give probabilistic models of what the CBS does.

Hidden: Uncertainty in "D, S ⊢ R"

- All that applies to D, applies to R, because both are models of the real world, one as is, and the other as it is to be.
- R is always an approximation of what we want, because if we overlook something in the real world and it turns out to be relevant to the CBS's behavior, e.g., a gaggle of Canadian geese that fly near a jet engine, then R may not be correct.

Uncertainty in "D, S ⊢ R"

- The formula D, S ⊢ R tries to be formal in the sense of describing what happens completely.
- But, as we have seen, it cannot be completely formal because at least D and R have to describe the real world, which is not formal

What does this do to the hope of formally modeling computer systems?

Molecular Software

- Molecular SW, e.g., DNA, RNA, Proteins, Catalysts
- Molecules designed specifically to achieve a desired effect
- Molecule is shown empirically to behave as specified in S, with 99.95% certainty
- In this case, in D, S $\vdash$ R, also S is informal!

Same is true of AI and ML software, because it has an approximate model of how learning is done, and the data about the real world are inaccurate too!