A Framework for Testing and Analysis

Learning objectives for this slideset

- Be confident to identify dimensions and tradeoff between test and analysis activities
- Be confident to distinguish validation from verification activities
- Have the capability to identify common limitations and potentials of test and analysis methods

Verification and validation

- Validation:
  does the software system meets the user's real needs?

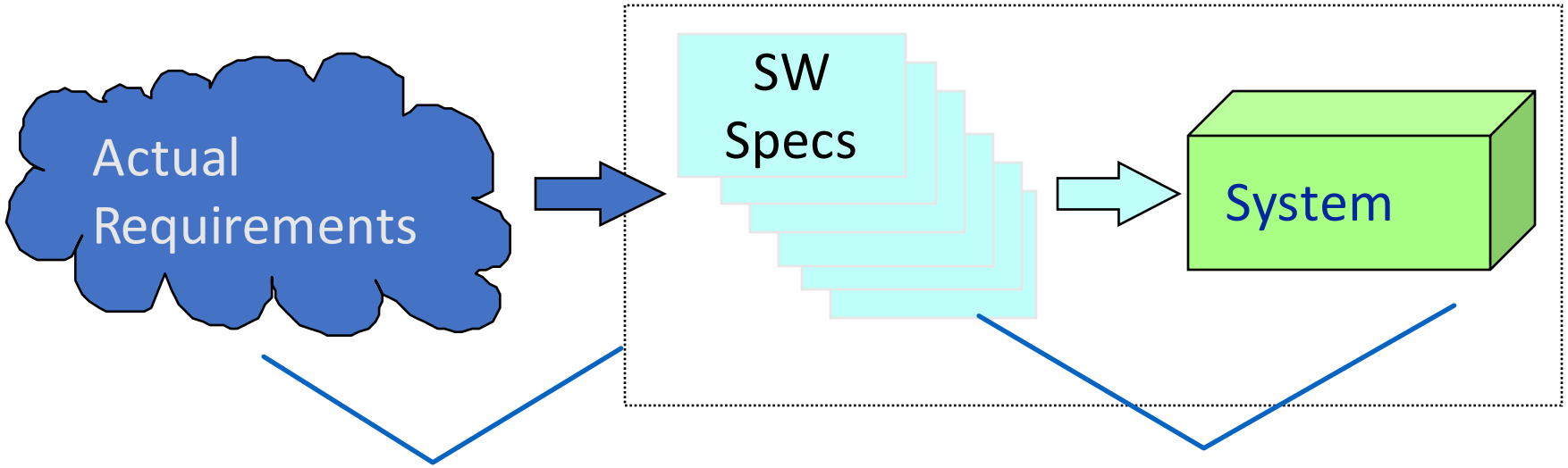
  are we building the right software?

  [This is connecting the technical system to the stakeholders worlds]
- Verification:
  does the software system meets the requirements specifications?

  are we building the software right?

  [This is about connecting the technical system to a more or less formal statements of requirement]

Validation and Verification

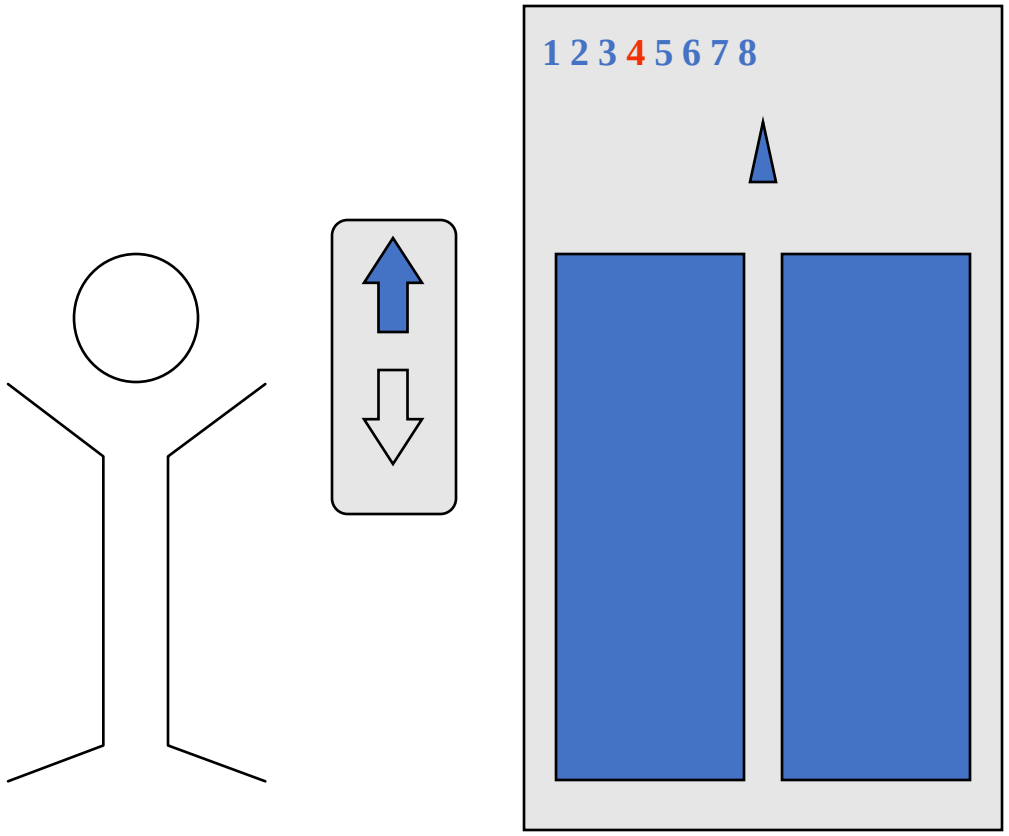


Includes usability testing, user feedback

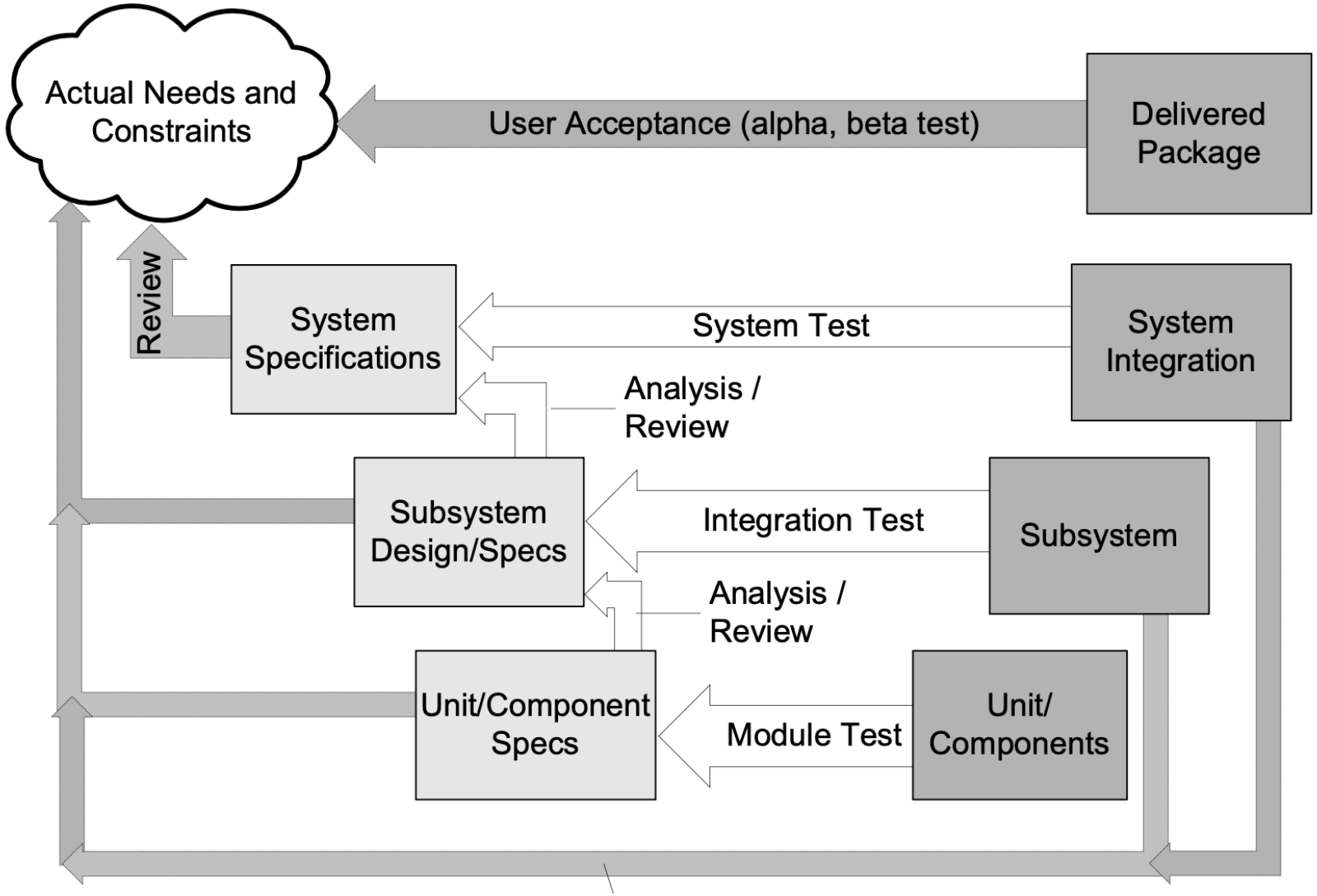
Includes testing, inspections, static analysis

Verification or validation depends on the specification

- Unverifiable (but validatable) spec: ... if a user presses a request button at floor i, an available elevator must arrive at floor i soon...
- Verifiable spec: ... if a user presses a request button at floor i, an available elevator must arrive at floor i within 30 seconds...
- Are there problems with this approach? How might you re-frame the original requirement?



Validation and Verification Activities



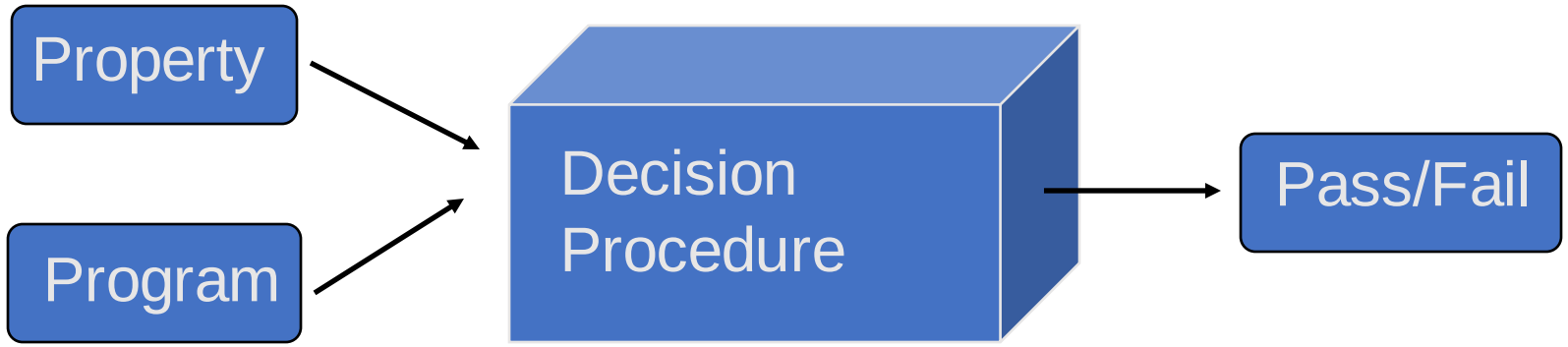
This is based on the "classical" V-model of development and illustrates the roles of validation and verification in that model. There are other possibilities, but this illustrates many of them.

validation

verification

Adapted by Stuart Anderson from (c) 2007 Mauro Pezzè & Michal Young

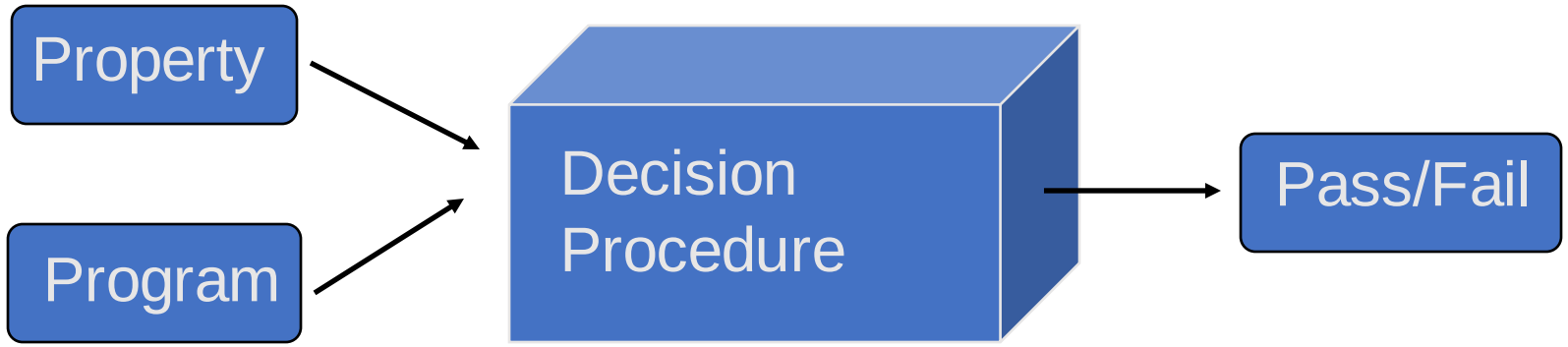
You can't ~~always~~ *ever* get what you want



Correctness properties are undecidable

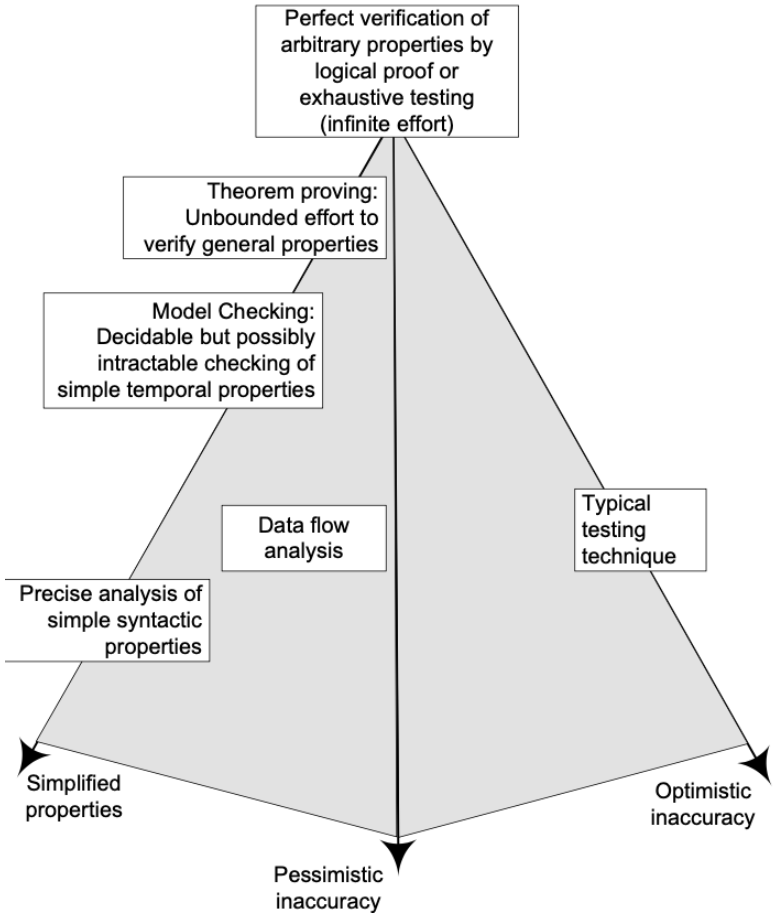
the halting problem can be embedded in almost every property of interest

You **can** get what you want sometimes



Restricting the form of properties and programs can allow you to find a decision procedure BUT in general you can't so in some domains there are such restrictions that allow this kind of setup.

Getting what you need …



- optimistic inaccuracy: we may accept some programs that do not possess the property (i.e., it may not detect all violations).
  - testing
- pessimistic inaccuracy: it is not guaranteed to accept a program even if the program does possess the property being analyzed
  - automated program analysis techniques
- simplified properties: reduce the degree of freedom for simplifying the property to check

Simplifying the situation

Original Situation

- Unrestricted use of language features that can result in unbounded looping of the code.
- In general it is quite possible to produce incomprehensible code by using the features of any modern programming language in an indisciplined manner.

Simplified Situation

- Impose restrictions: e.g. SPARK Ada: *Handling of exceptions is not permitted. Exception handling gives raise to numerous interprocedural control-flow paths. Formal verification of programs with exception handlers requires tracking properties along all those paths, which is not doable precisely without a lot of manual work. But raising exceptions is allowed (see Raising Exceptions and Other Error Signaling Mechanisms).*
- See: https://docs.adacore.com/spark2014-docs/html/ug/en/source/language_restrictions.html

Some Terminology

- **Safe**: A safe analysis has no optimistic inaccuracy, i.e., it accepts only correct programs.
- **Sound**: An analysis of a program P with respect to a formula F is sound if the analysis returns true only when the program does satisfy the formula.
- **Complete**: An analysis of a program P with respect to a formula F is complete if the analysis always returns true when the program actually does satisfy the formula.

Summary

- Many interesting properties are undecidable, thus in general we cannot count on tools that work without human intervention (but we often accept “approximately” correct programs)
- Assessing program qualities comprises two complementary sets of activities: validation (does the software do what it is supposed to do?) and verification (does the system behave as specified?)
- There is no single technique for all purposes: test designers need to select a suitable combination of techniques