Computer Science
UNIVERSITY OF TORONTO

lecture 10: verification & validation

csc302h

winter 2013

administrative

- matplotlib issues? everyone should have built & run it by now (at least on cdf)
- should have identified your short-list for a2 and be in the process of selecting from it.
  - github project issue list:

    https://github.com/matplotlib/matplotlib/issues
- tutorial this week: git basic basics

Computer Science
UNIVERSITY OF TORONTO

recap from last time

- robustness analysis
  - bridge between use case and more technical things like sequence diagrams & code
  - skipped if you don't need it, diagrams erased when you are finished with them
  - used to:
    - analyze logic of a use case
    - ensure use cases are "robust" in that they really do represent the usage requirements
    - identify objects & responsibilities
    - visualize the things will build (i.e. code)
    - communicate (almost) technical stuff to stakeholders

Verification and Validation

Refresher: definitions of V&V

V&V strategies

- Modeling and Prototyping
- Inspection
- Formal Analysis
- (Testing)

Independent V&V

Quality Assurance

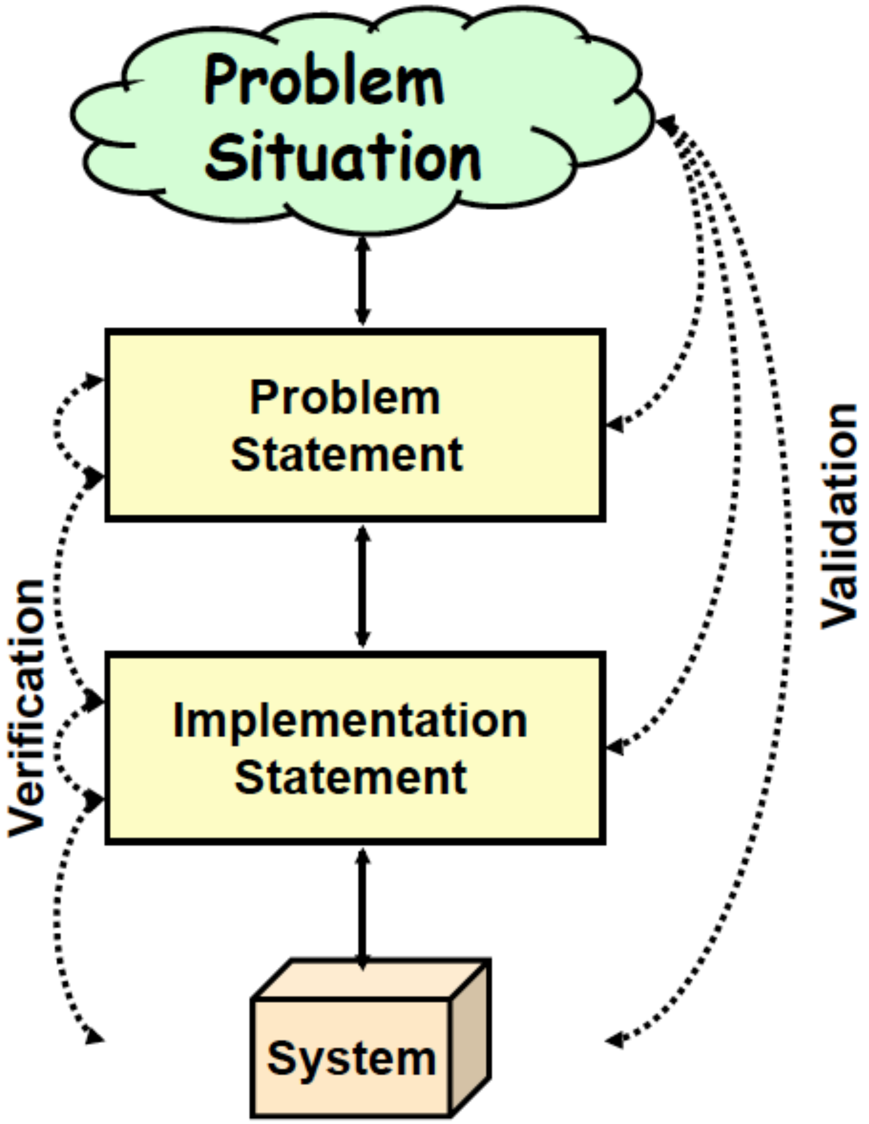
Refresher: V&V

Validation:

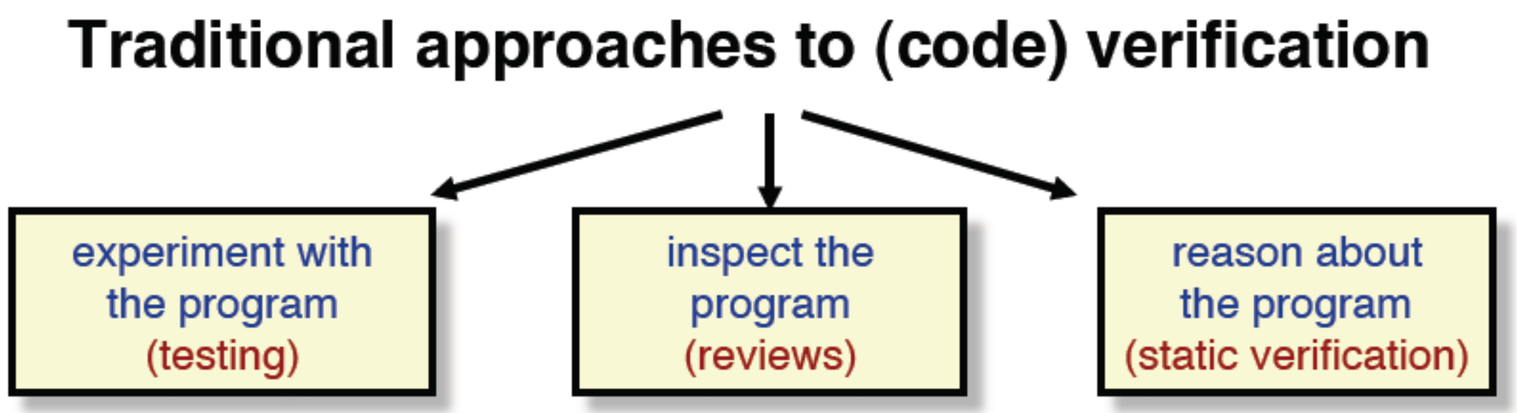
- "Are we building the right system?"
- Does our problem statement accurately capture the real problem?
- Did we account for the needs of all the stakeholders?

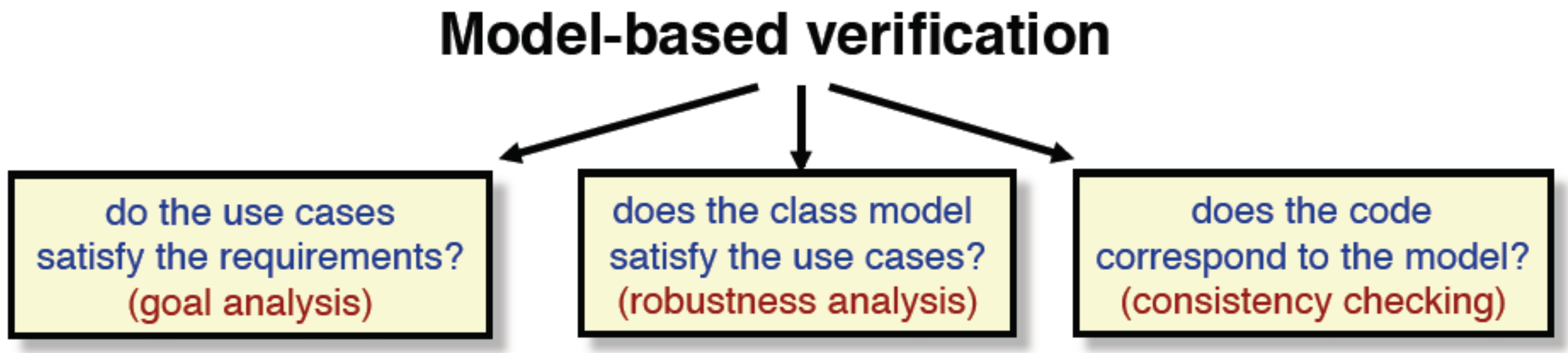
Verification:

- "Are we building the system right?"
- Does our design meet the spec?
- Does our implementation meet the spec?
- Does the delivered system do what we said it would do?
- Are our requirements models consistent with one another?

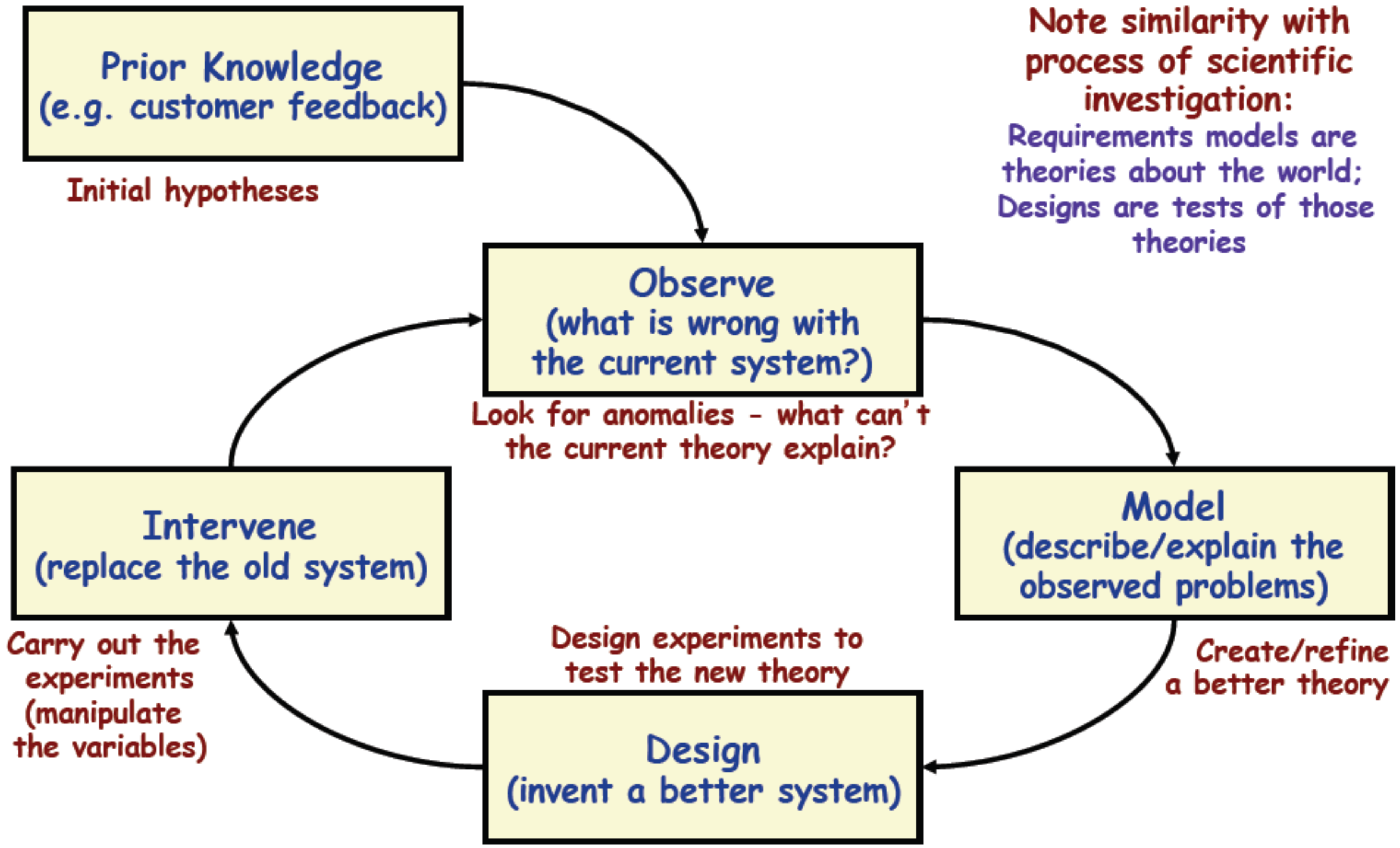


Verification


Traditional approaches to (code) verification
experiment with the program (testing)
inspect the program (reviews)
reason about the program (static verification)


Model-based verification
do the use cases satisfy the requirements? (goal analysis)
does the class model satisfy the use cases? (robustness analysis)
does the code correspond to the model? (consistency checking)

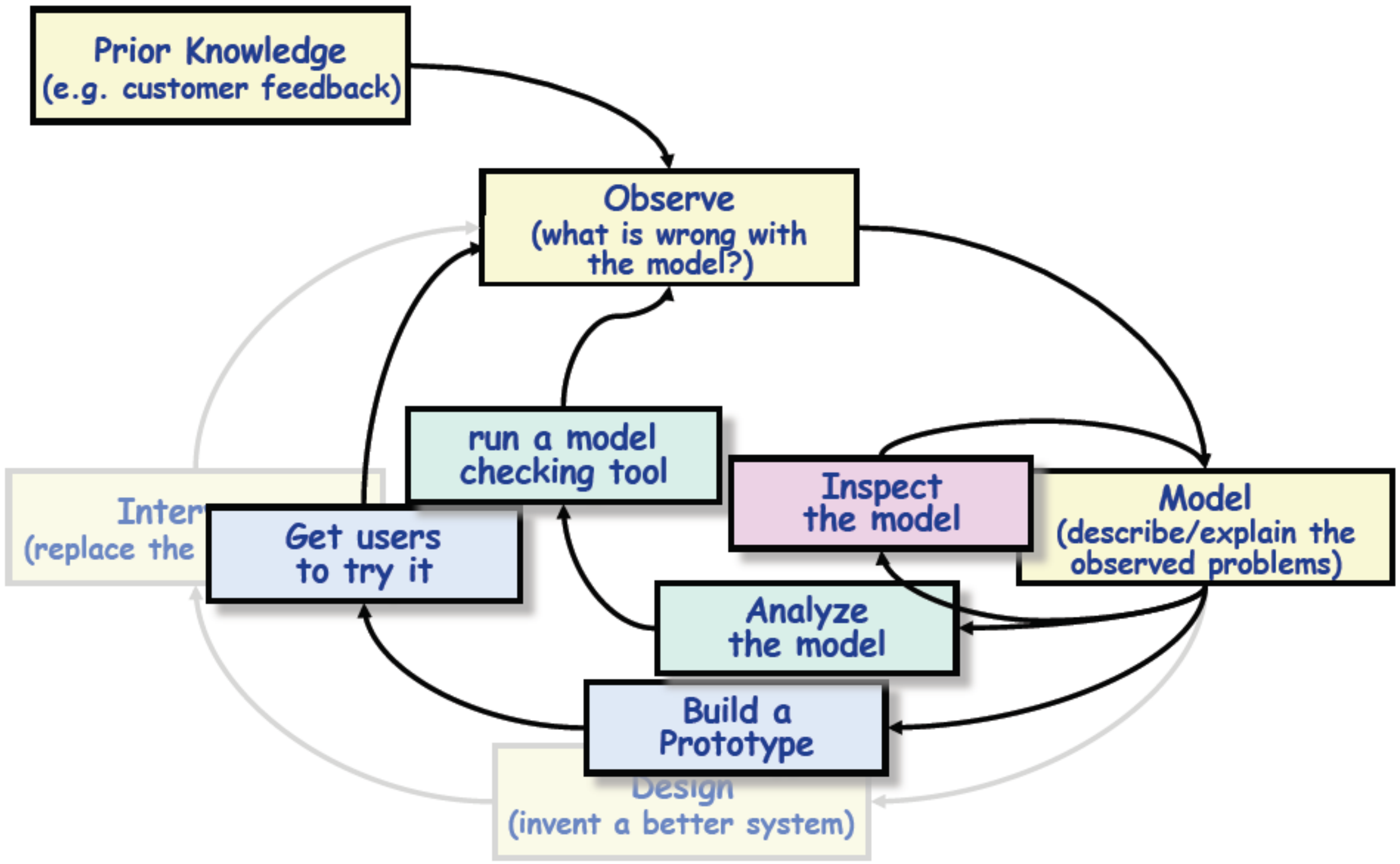
Understanding Validation


Prior Knowledge (e.g. customer feedback)
Initial hypotheses
Observe (what is wrong with the current system?)
Look for anomalies - what can't the current theory explain?
Model (describe/explain the observed problems)
Create/refine a better theory
Design (invent a better system)
Design experiments to test the new theory
Intervene (replace the old system)
Carry out the experiments (manipulate the variables)

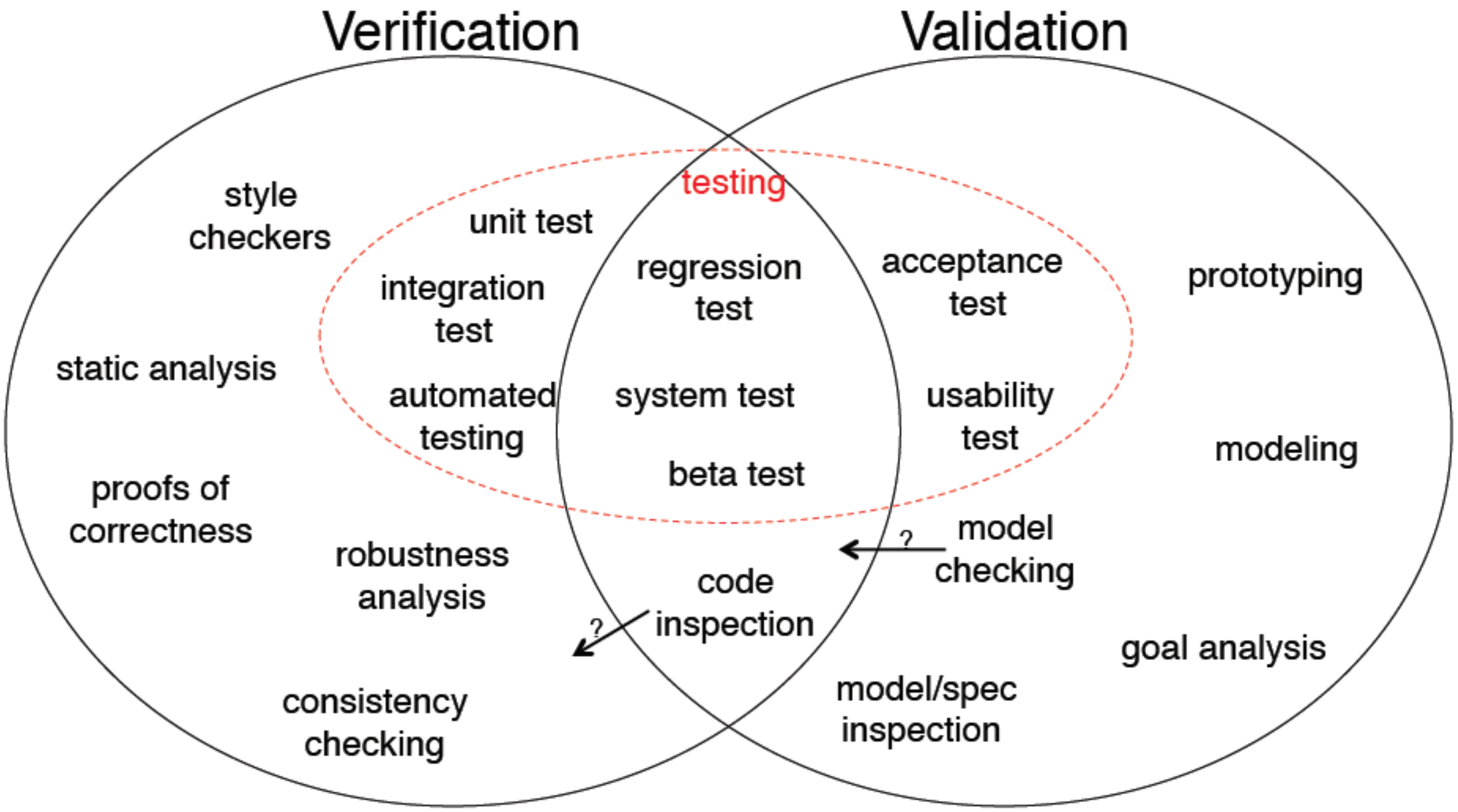
Note similarity with process of scientific investigation:

Requirements models are theories about the world; Designs are tests of those theories

Validation techniques



Choice of Techniques



Prototyping

Presentation Prototypes

Explain, demonstrate and inform – then throw away

e.g. used for proof of concept; explaining design features; etc.

Exploratory Prototypes

Used to determine problems, elicit needs, clarify goals, compare design options

Informal, unstructured and thrown away.

Breadboards or Experimental Prototypes

Explore technical feasibility, or test suitability of a technology, then thrown away

Typically no user/customer involvement

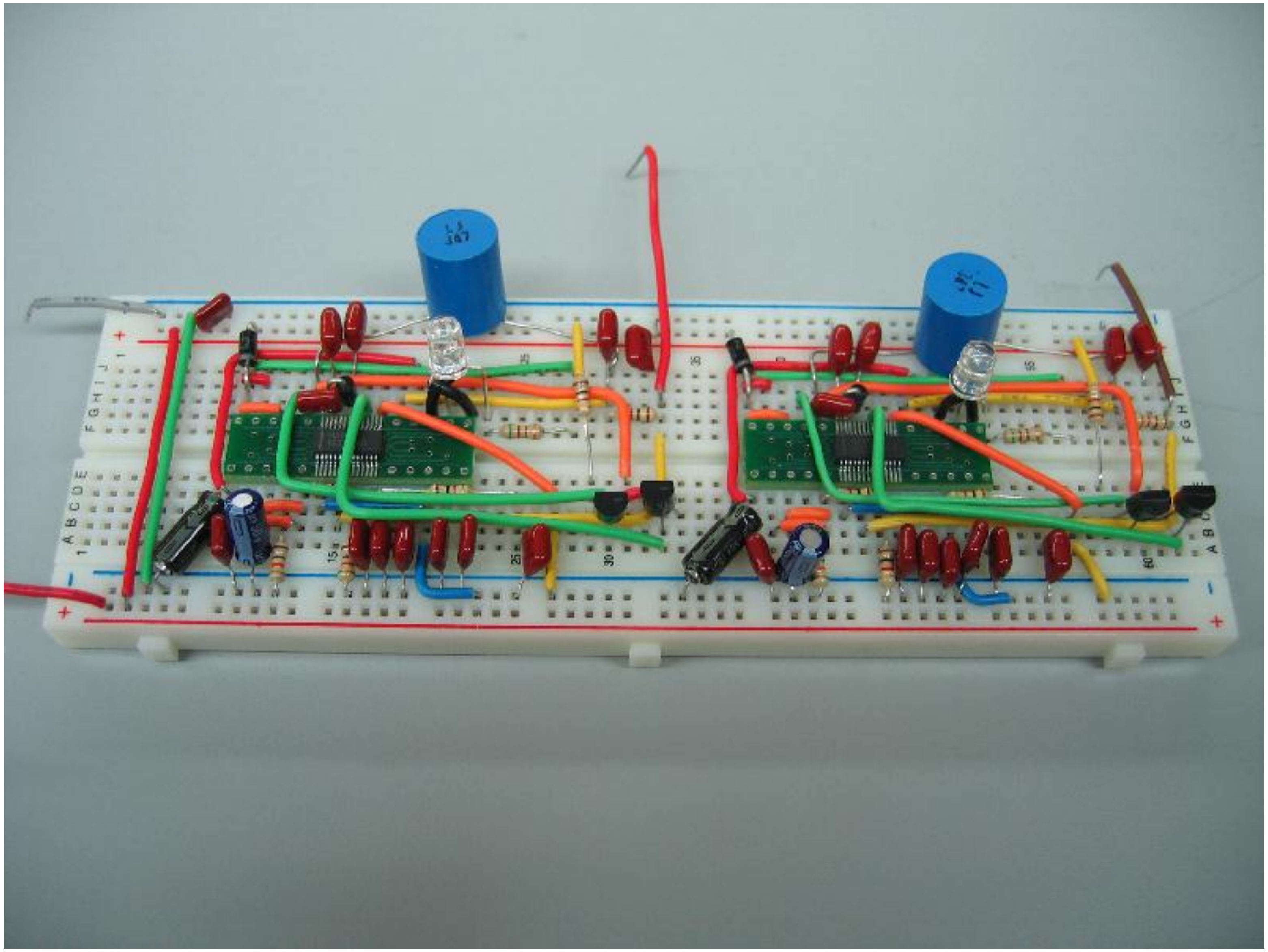
Evolutionary

(a.k.a. "operational prototypes", "pilot systems"):

Development seen as continuous process of adapting the system

"prototype" is an early deliverable, to be continually improved.

h/w breadboard == s/w spike



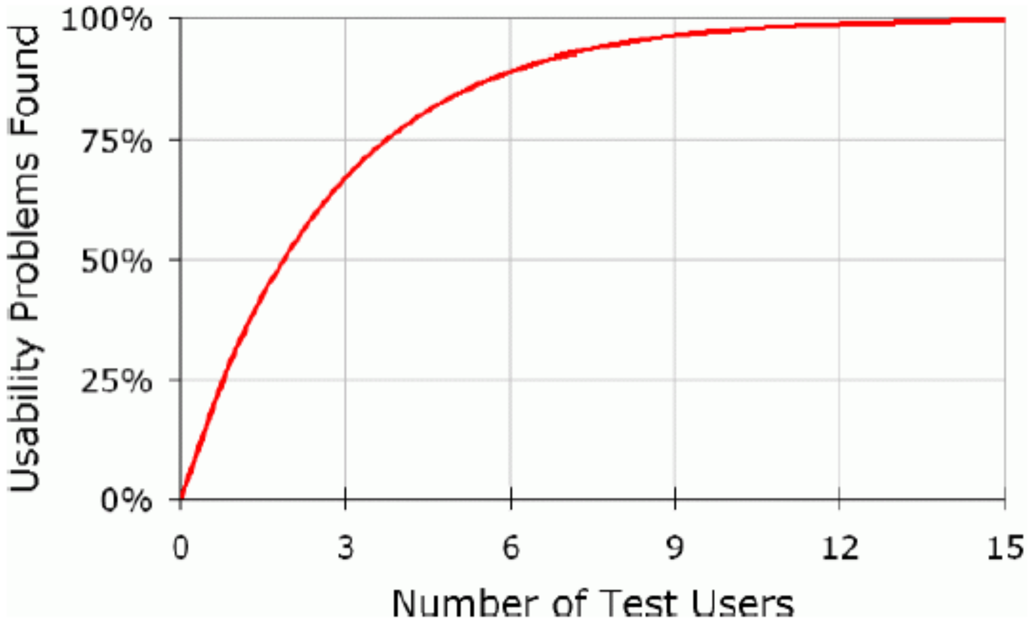
Usability Testing

Real users try out the system (or prototype)

- Choose representative tasks
- Choose representative users
- Observe what problems they encounter

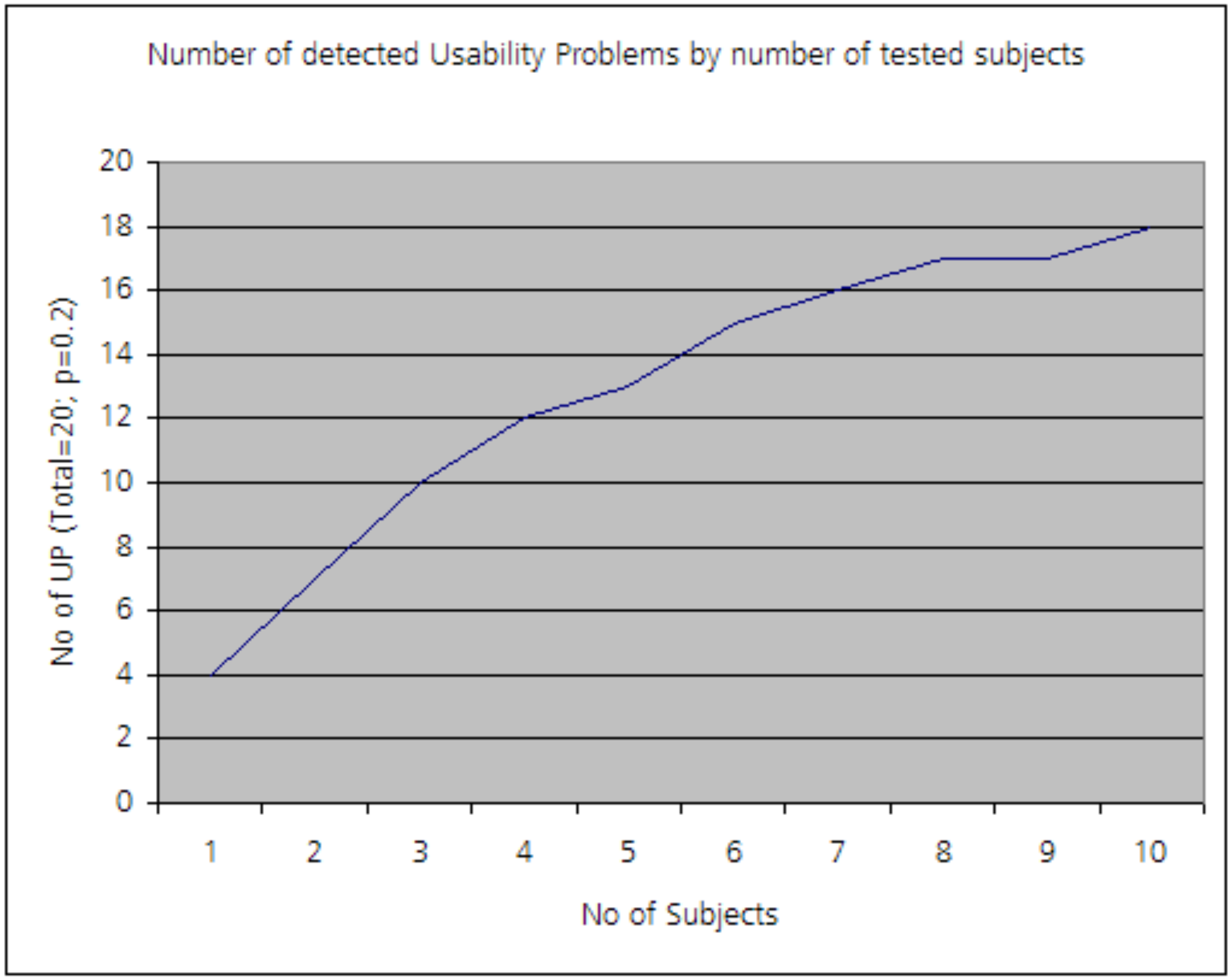
How many users?

- 3-5 users gives best return on investment



Neilson’s equation

$$U = 1 - (1 - p)^n$$



Model Analysis

Verification

- "Is the model well-formed?"
- Are the parts of the model consistent with one another?

Validation:

- 'What if' questions:
  - reasoning about the consequences of particular requirements;
  - reasoning about the effect of possible changes
  - "will the system ever do the following..."
- Formal challenges:
  - "if the model is correct then the following property should hold..."
- Animation of the model on small examples
- State exploration
  - E.g. use model checking to find traces that satisfy some property

UML Consistency Checking

Use Case Diagrams

- Does each use case have a user?
  - Does each user have at least one use case?
- Is each use case documented?
  - Using sequence diagrams or equivalent

Class Diagrams

- Does the class diagram capture all the classes mentioned in other diagrams?
- Does every class have methods to get/set its attributes?

Sequence Diagrams

- Is each class in the class diagram?
- Can each message be sent?
  - Is there an association connecting sender and receiver classes on the class diagram?
  - Is there a method call in the sending class for each sent message?
  - Is there a method call in the receiving class for each received message?

Model Checkers

Automatically check properties (expressed in Temporal Logic)

temporal logic adds modal operators to FOPL:

$\Box p$	p is true now and always (in the future)
$\Diamond p$	p is true eventually (in the future)
$\Box(p \Rightarrow \Diamond q)$	whenever p occurs, it's always (eventually) followed by a q

The model may be:

- of the program itself (each statement is a 'state')
- an abstraction of the program
- a model of the specification
- a model of the requirements

A Model Checker searches all paths in the state space

- ...with lots of techniques for reducing the size of the search
- Model checking does not guarantee correctness...
  - it only tells you about the properties you ask about
  - it may not be able to search the entire state space (too big!)
- ...but is good at finding many safety, liveness and concurrency problems

Inspections…

"Management reviews"

- E.g. preliminary design review (PDR), critical design review (CDR), …
- Used to provide confidence that the design is sound
- Audience: management and sponsors (customers)

"Walkthroughs" ≈ scientific peer review

- developer technique (usually informal)
- used by development teams to improve quality of product
- focus is on understanding design choices and finding defects

"(Fagan) Inspections"

- a process management tool (always formal)
- used to improve quality of the development process
- collect defect data to analyze the quality of the process
- written output is important
- major role in training junior staff and transferring expertise

Why use inspection?

Inspections are very effective

- *Code* inspections are better than testing for finding defects
- For *Models* and *Specs*, it ensures domain experts carefully review them

Key ideas:

- Preparation: reviewers inspect individually first
- Collection meeting: reviewers meet to merge their defect lists
- Note each defect, but don’t spend time trying to fix it
- The meeting plays an important role:
  - Reviewers learn from one another when they compare their lists
  - Additional defects are uncovered
- Defect profiles from inspection are important for process improvement

Wide choice of inspection techniques:

- What roles to use in the meeting?
- How to structure the meeting?
- What kind of checklist to use?

Structuring the inspection

Checklist

- uses a checklist of questions/issues
- review structured by issue on the list

Walkthough

- one person presents the product step-by-step
- review is structured by the product

Round Robin

- each reviewer in turn gets to raise an issue
- review is structured by the review team

Speed Review

- each reviewer gets 3 minutes to review a chunk, then passes to the next person
- good for assessing comprehensibility!

Benefits of formal inspection

Source: *Adapted from Blum, 1992, Freedman and Weinberg, 1990, & notes from Philip Johnson.*

For applications programming:

- more effective than testing
- most reviewed programs run correctly first time
- compare: 10-50 attempts for test/debug approach

Data from large projects

- error reduction by a factor of 5; (10 in some reported cases)
- improvement in productivity: 14% to 25%
- percentage of errors found by inspection: 58% to 82%
- cost reduction of 50%-80% for V&V (even including cost of inspection)

Effects on staff competence:

- increased morale, reduced turnover
- better estimation and scheduling (more knowledge about defect profiles)
- better management recognition of staff ability

Role for Independent V&V?

V&V performed by a separate contractor

- Independent V&V fulfills the need for an independent technical opinion.
- Cost between 5% and 15% of development costs
- NASA Studies show up to fivefold return on investment:
  - Errors found earlier, cheaper to fix, cheaper to re-test
  - Clearer specifications
  - Developer more likely to use best practices

Three types of independence:

- Managerial Independence:
  - separate responsibility from that of developing the software
  - can decide when and where to focus the V&V effort
- Financial Independence:
  - Costed and funded separately
  - No risk of diverting resources when the going gets tough
- Technical Independence:
  - Different personnel, to avoid analyst bias
  - Use of different tools and techniques

summary

- v&v is not just for software, but we care only about the s/w customizations to general v&v
- terms are often used interchangeably, or confused
- in software, v&v ≈ quality control, or quality assurance

summary (2)

- validation (a.k.a. high-level testing) is usually preformed by doing “dynamic testing”
  - unit tests (less so, more for verification), integration tests, system tests, (user) acceptance tests
- validation: “are we building the correct thing?”
- verification: “are we building the thing correctly?”
- test cases are written for verification & run for validation.
- range of activities:
  - mission critical: may use formal methods (proofs)
  - latest fart-app, probably not so much :)

summary (3)

- according to cmm & ieee-std-610:

  "verification is the process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase."

  "validation is the process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements."

Computer Science
UNIVERSITY OF TORONTO

next time…

software testing