M. Weintraub and
F. Tip

TESTING STRATEGIES

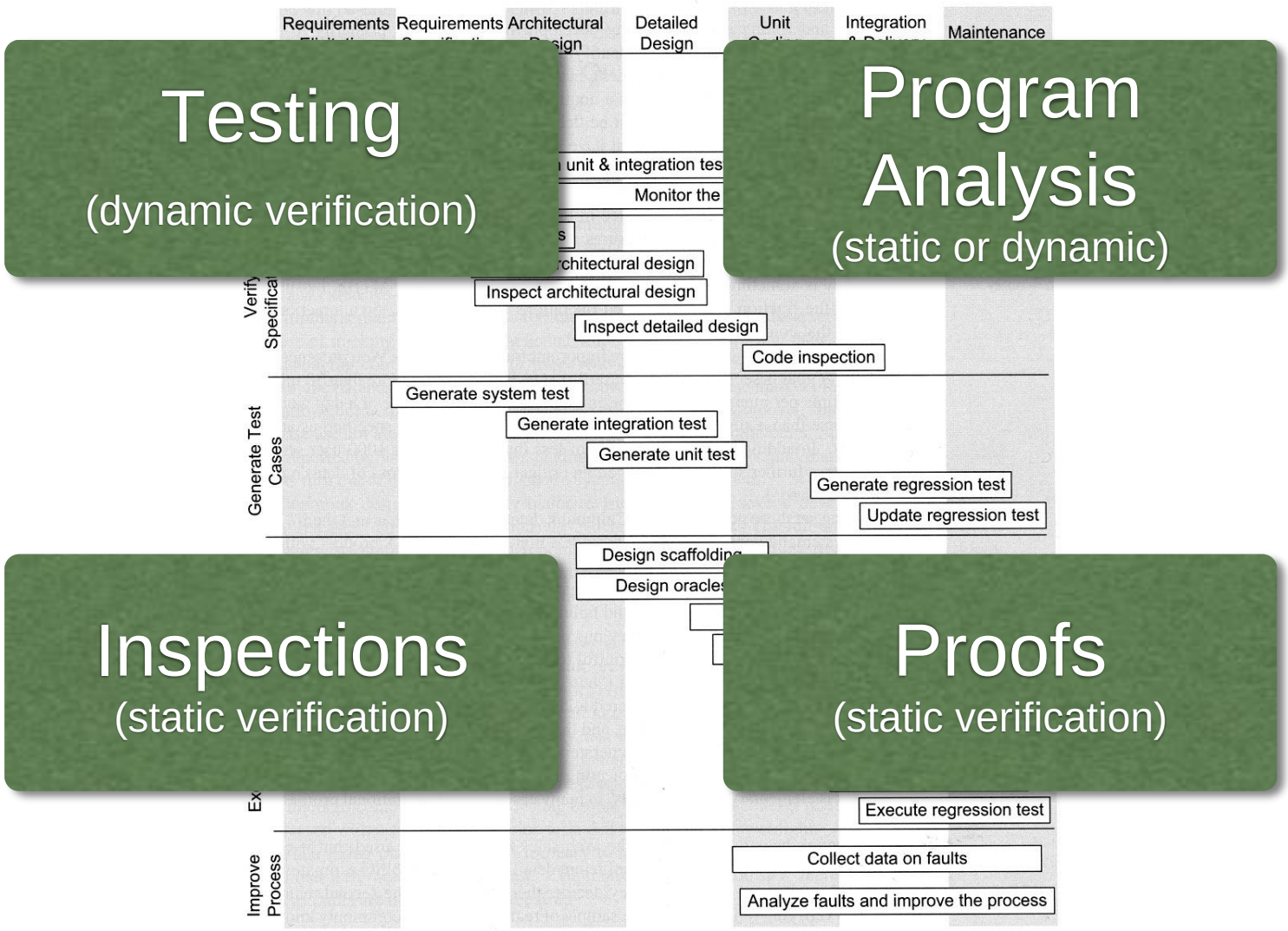
Thanks go to Andreas Zeller for allowing incorporation of his materials

Northeastern University
College of Computer and Information Science
440 Huntington Avenue . 202 West Village H . Boston, MA 02115 . T 617.373.2462 . ccis.northeastern.edu

TESTING

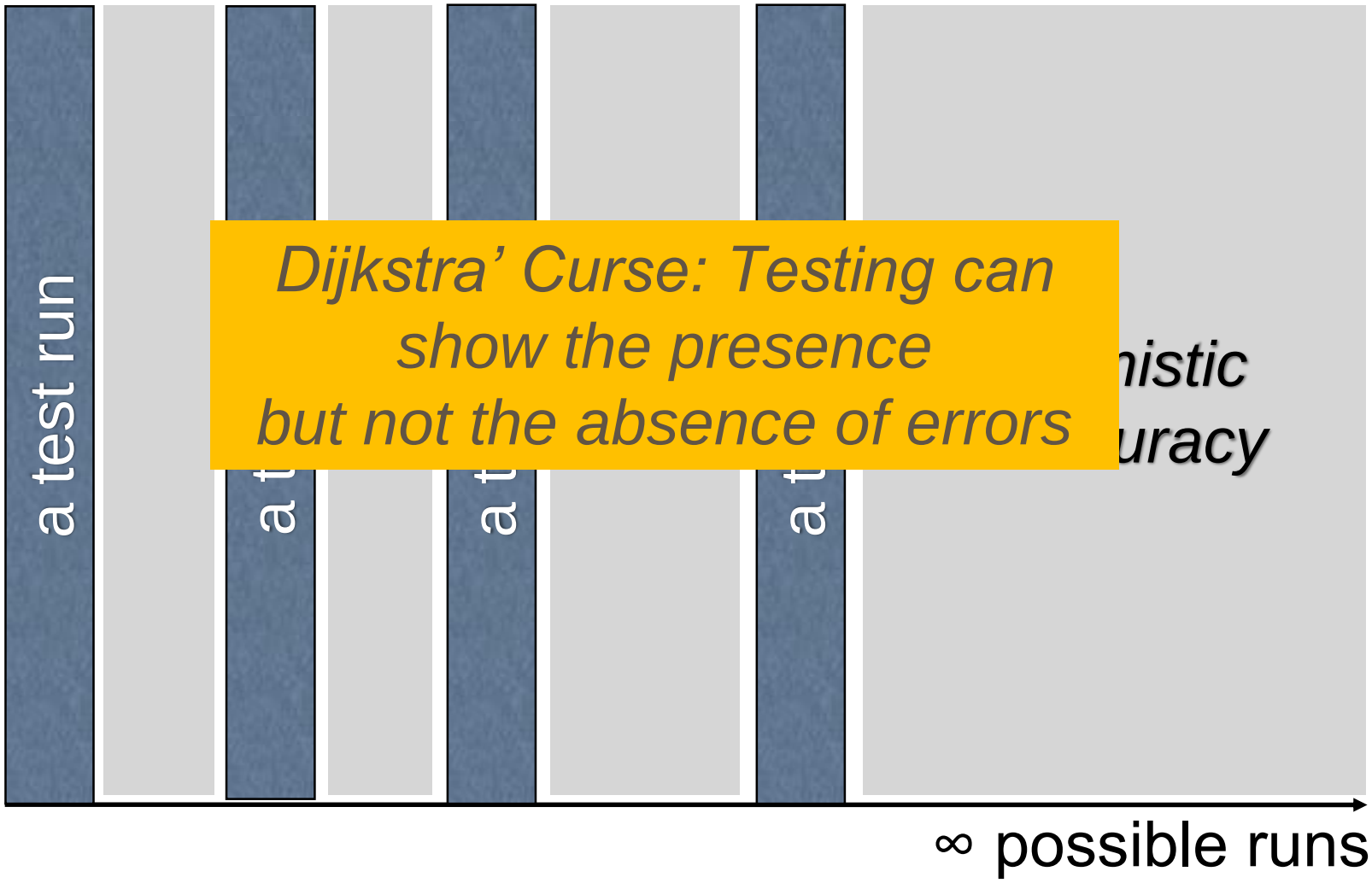
- **Testing**: a procedure intended to establish the quality, performance, or reliability of something, esp. before it is taken into widespread use

RECALL FROM BEFORE – THESE ARE OUR TECHNIQUES FOR EVALUATING SOFTWARE

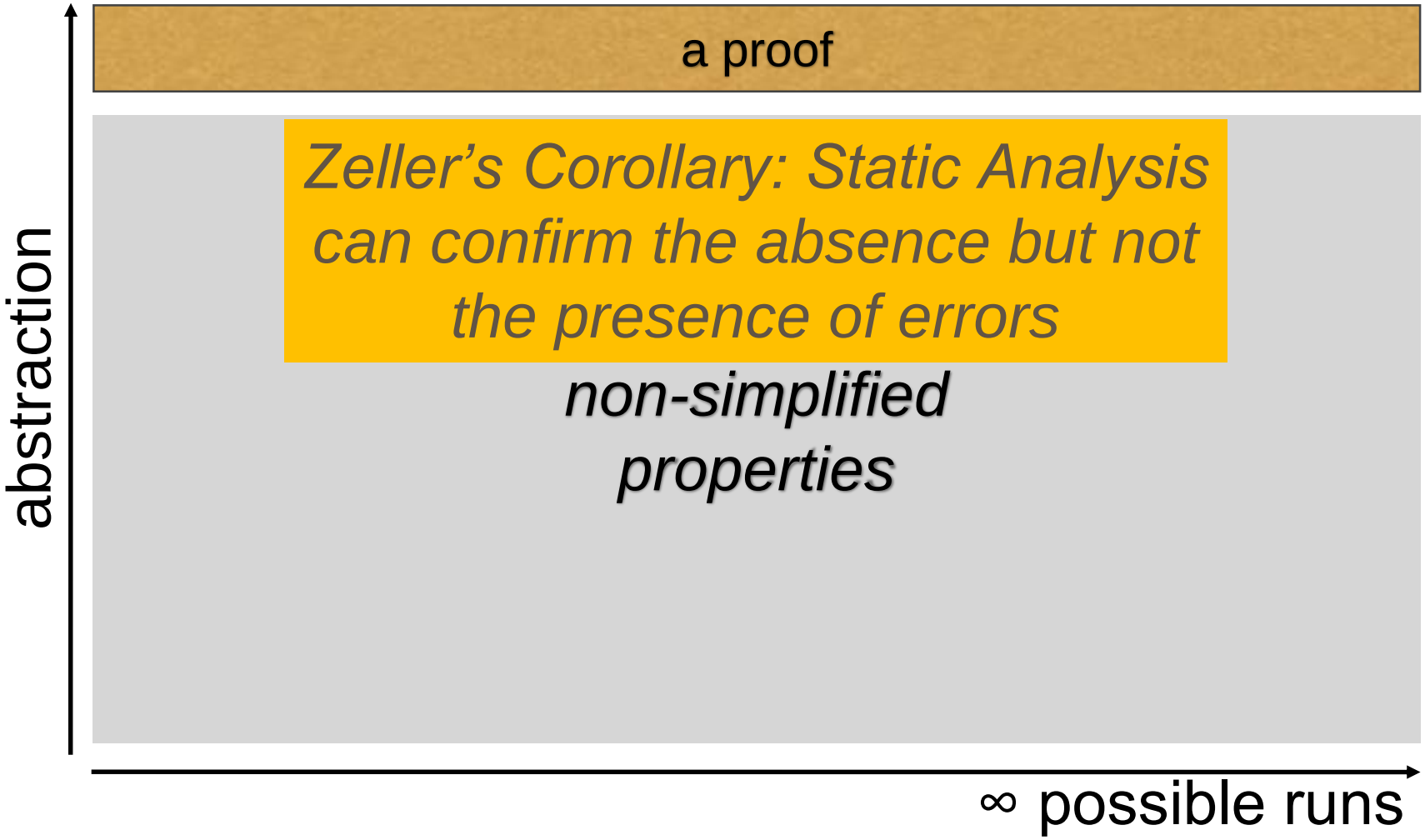


3

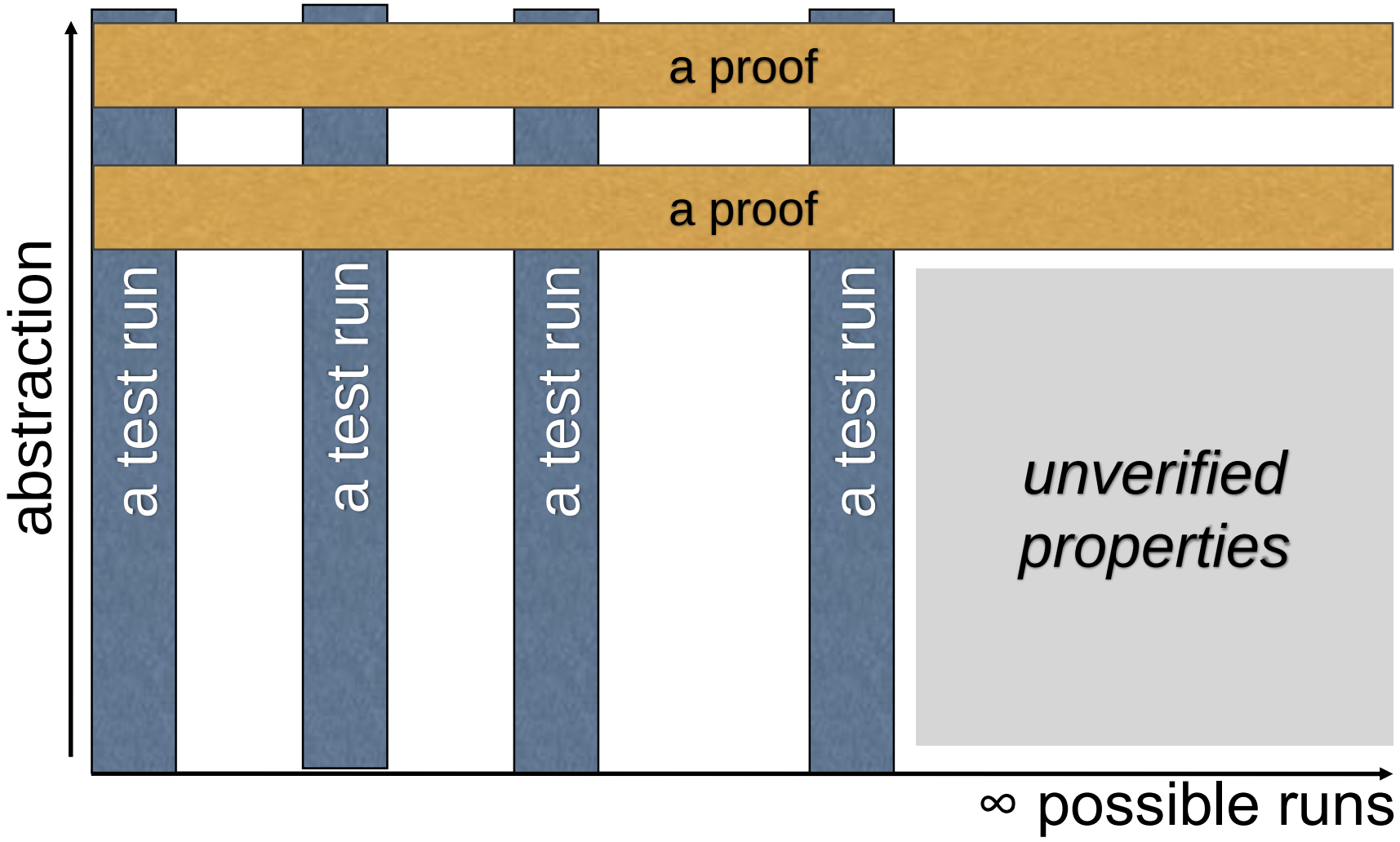
THE CURSE OF FUNCTIONAL TESTING



ITS STRUCTURAL TESTING COROLLARY



COMBINING METHODS



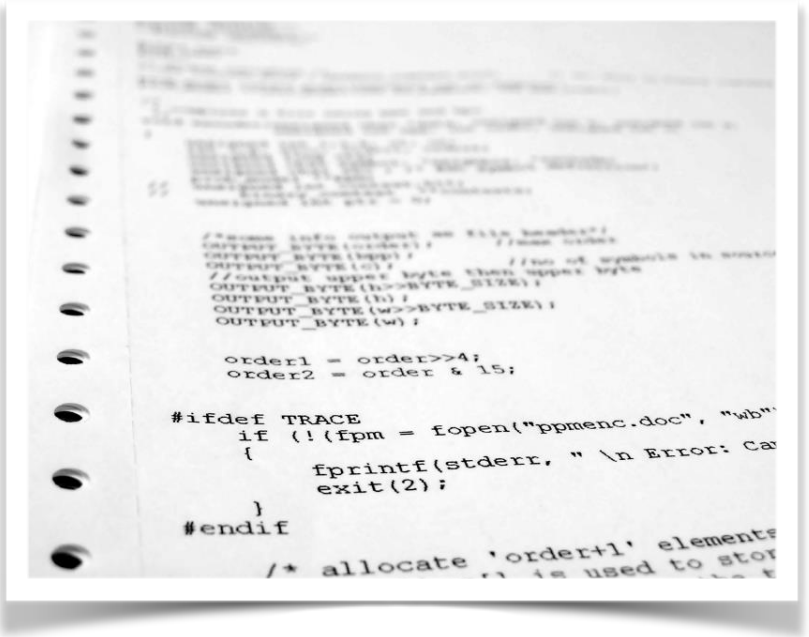
WHY IS SOFTWARE VERIFICATION HARD?

- Many different quality requirements
- Evolving (and deteriorating) structure
- Inherent non-linearity
- Uneven distribution of faults

WHY IS SOFTWARE VERIFICATION HARD?

- Many different quality requirements
- Evolving (and deteriorating) structure
- Inherent non-linearity
- Uneven distribution of faults

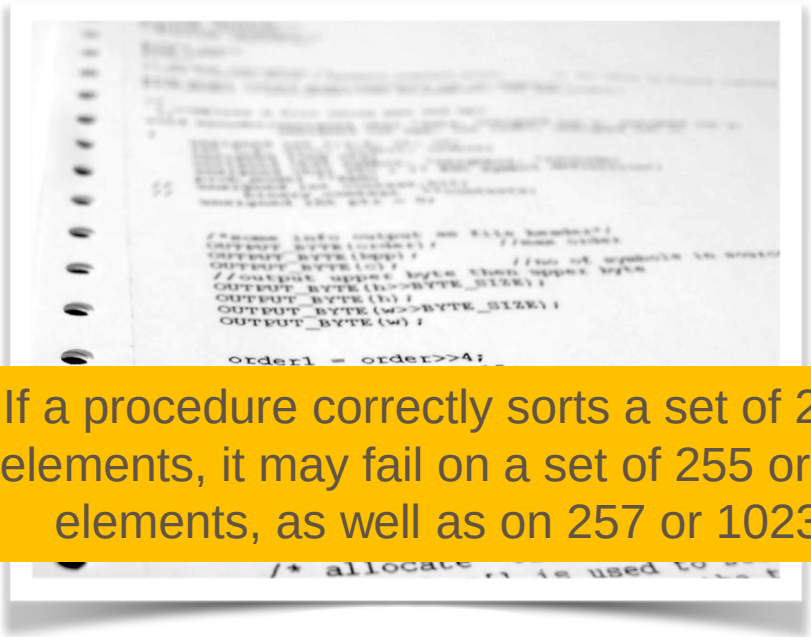
If an elevator can safely carry a load of 1000 kg, it can also safely carry any smaller load



WHY IS SOFTWARE VERIFICATION HARD?

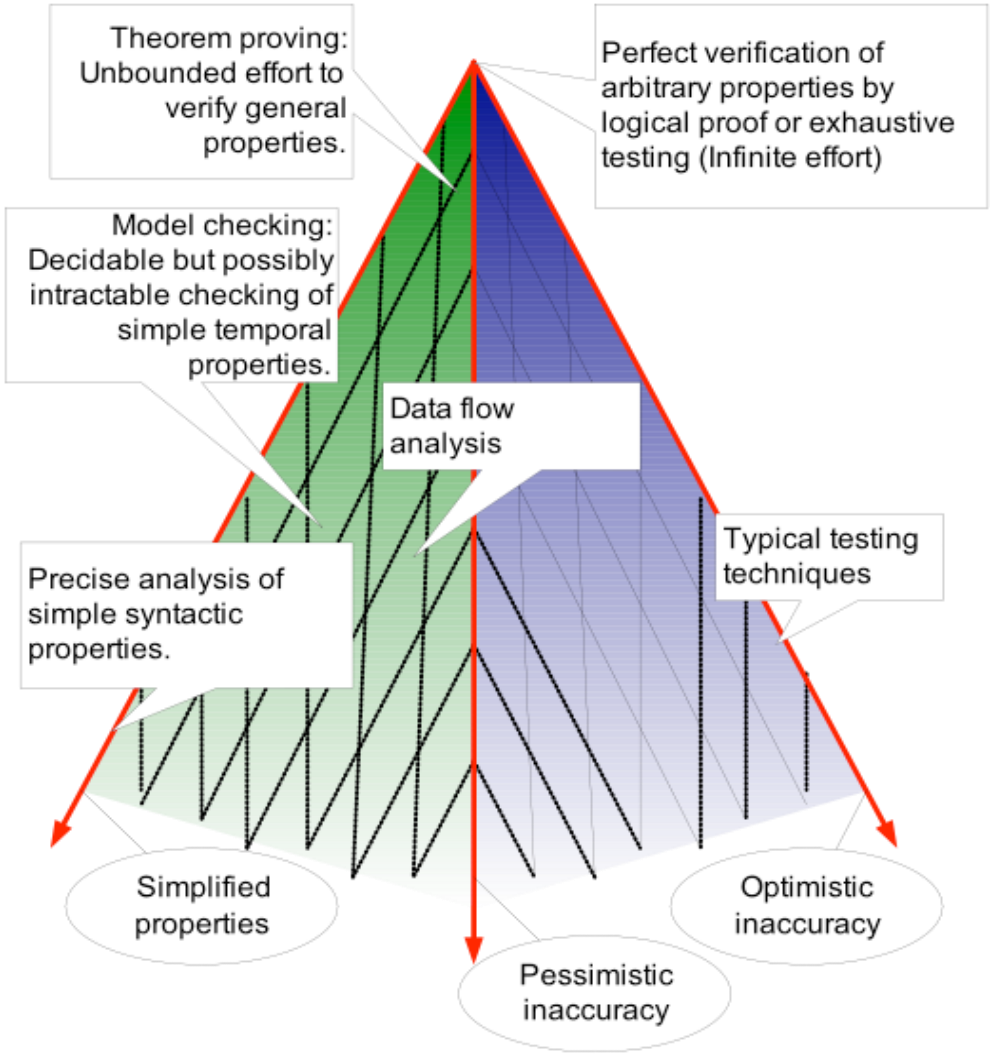
- Many different quality requirements
- Evolving (and deteriorating) structure
- Inherent non-linearity
- Uneven distribution of faults

If an elevator can safely carry a load of 1000 kg, it can also safely carry any smaller load



If a procedure correctly sorts a set of 256 elements, it may fail on a set of 255 or 53 elements, as well as on 257 or 1023

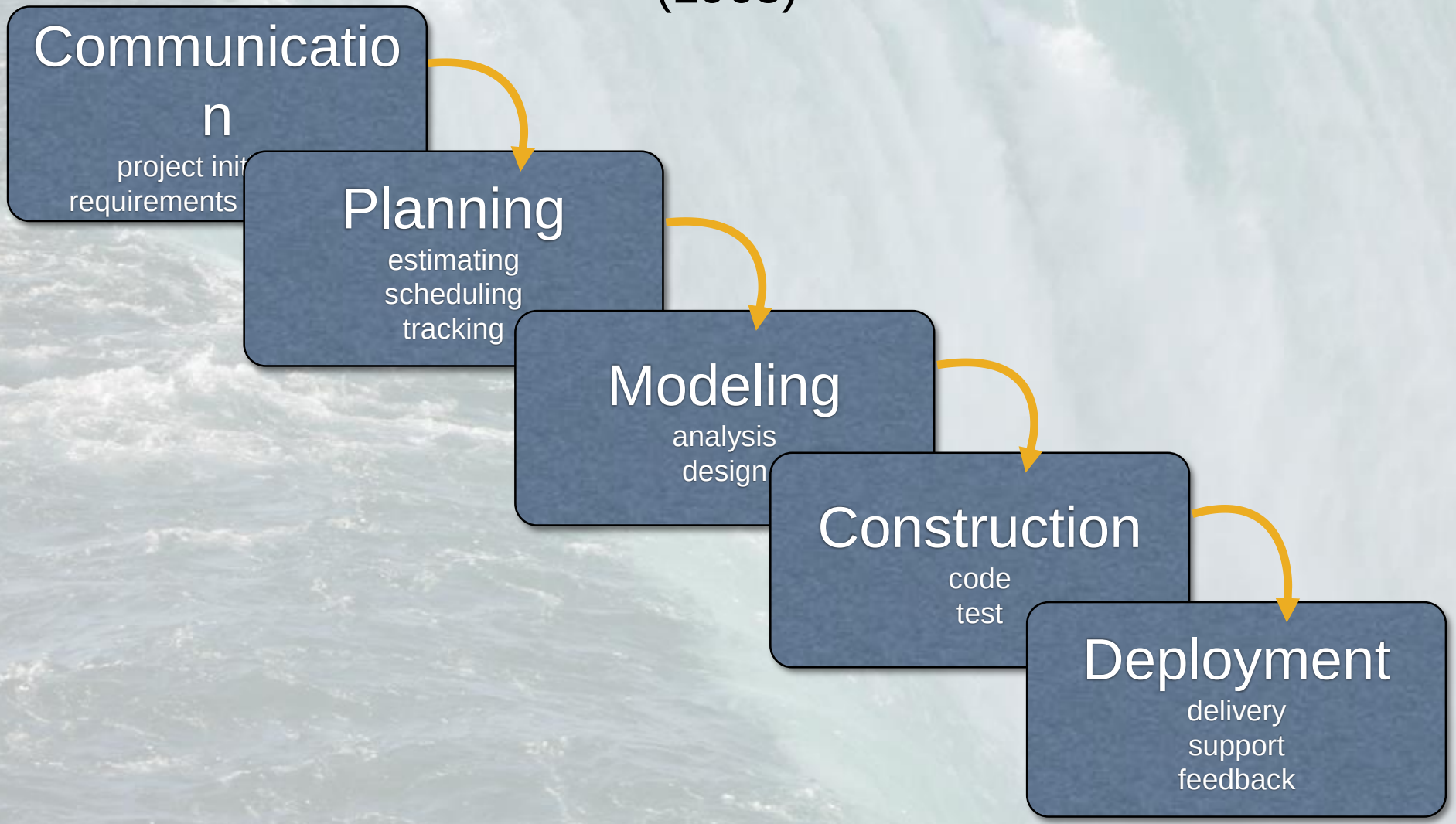
A TESTING PROGRAM INVOLVES TRADE-OFFS



- We can be inaccurate (optimistic or pessimistic)
- or we can simplify properties…
- but you cannot have it all!

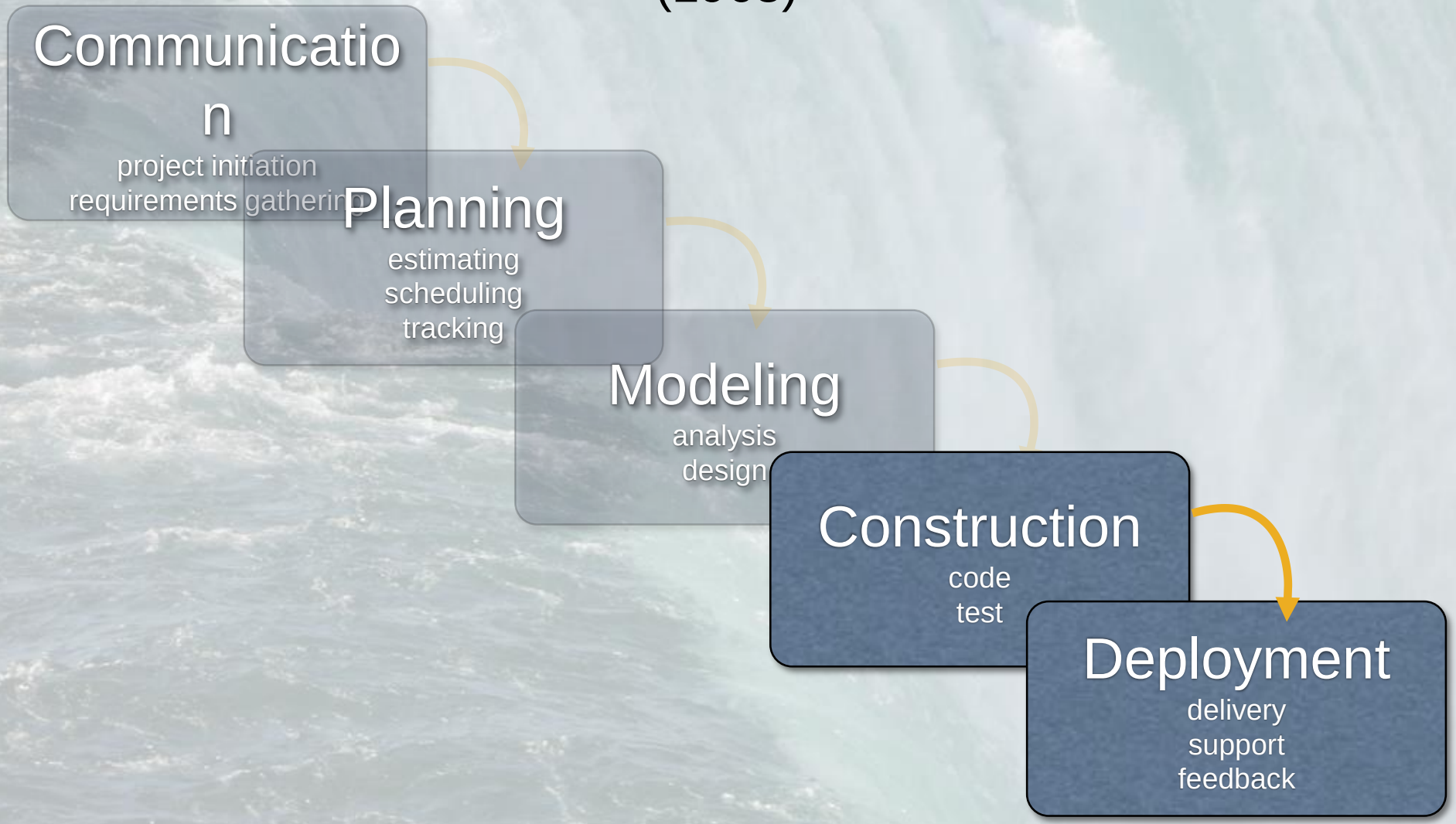
Waterfall Model

(1968)



Waterfall Model

(1968)



We built it!

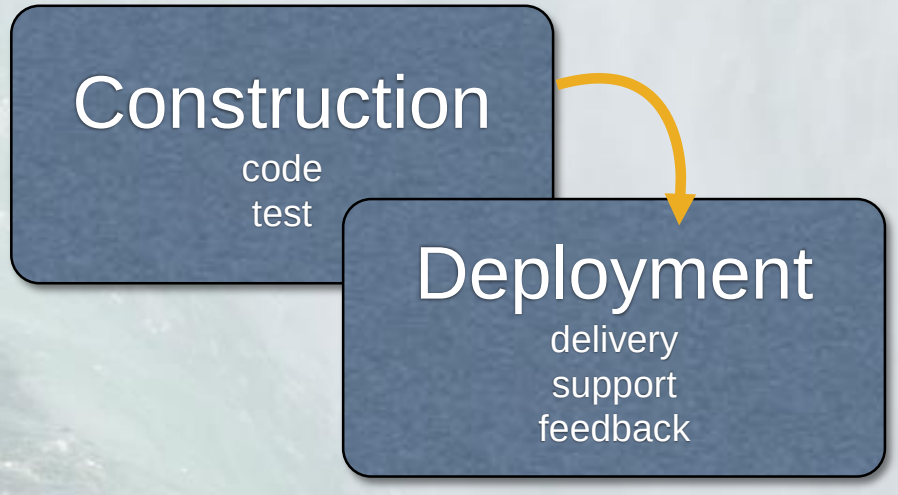
Shall we deploy it?

Waterfall Model

(1968)

Waterfall Model

(1968)



V & V

Validation

Ensuring that software has been built according to customer requirements

Are we building the right product or service?

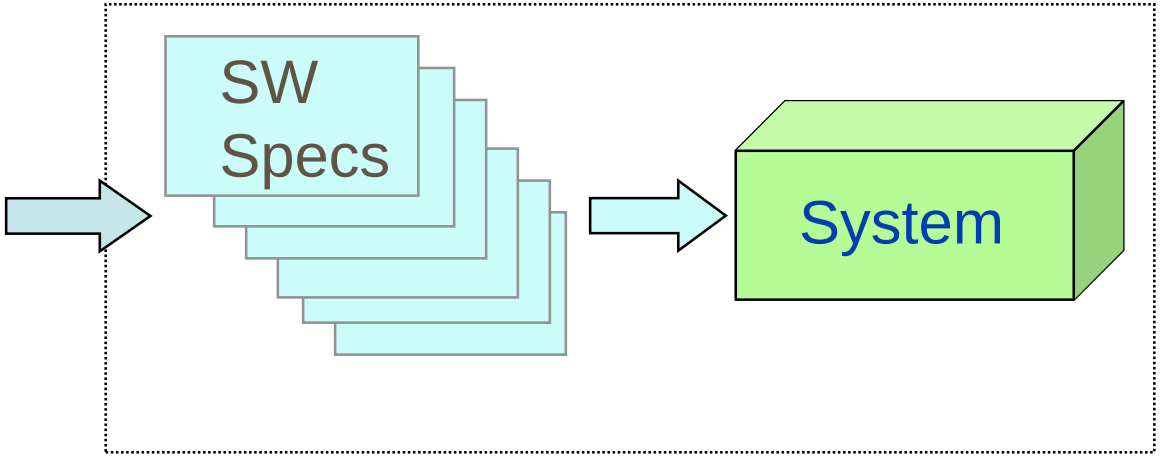
Verification

Ensuring that software correctly implements a specific function

Are we building the product or service right?

VALIDATION AND VERIFICATION

Actual Requirements



Validation

Involves usability testing, user feedback, & product trials

Verification

Includes testing, code inspections, static analysis, proofs

VALIDATION

“if a user presses a request button at floor i, an available elevator must arrive at floor i soon”

not verifiable, but validatable

VERIFICATION

“if a user presses a request button at floor i, an available elevator must arrive at floor i within 30 seconds”

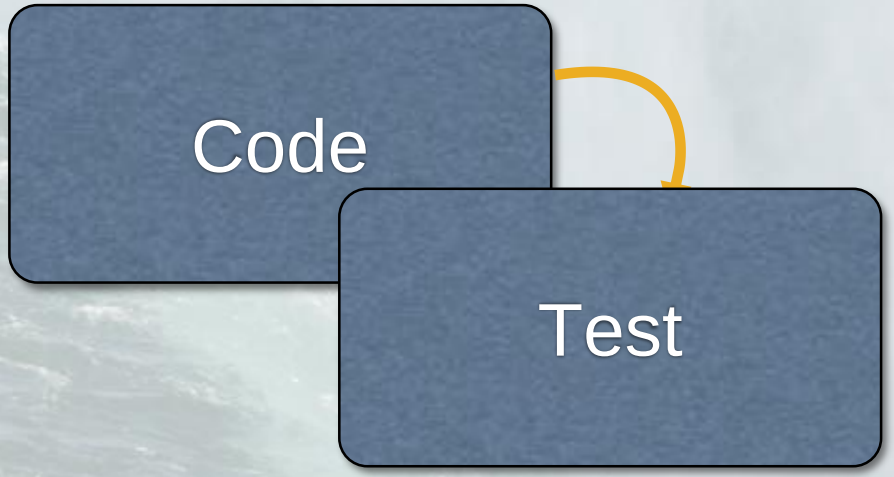
verifiable

CORE QUESTIONS

- When does V&V start? When is it done?
- Which techniques should be applied?
- How do we know a product is ready?
- How can we control the quality of successive releases?
- How can we improve development?

Waterfall Model

(1968)

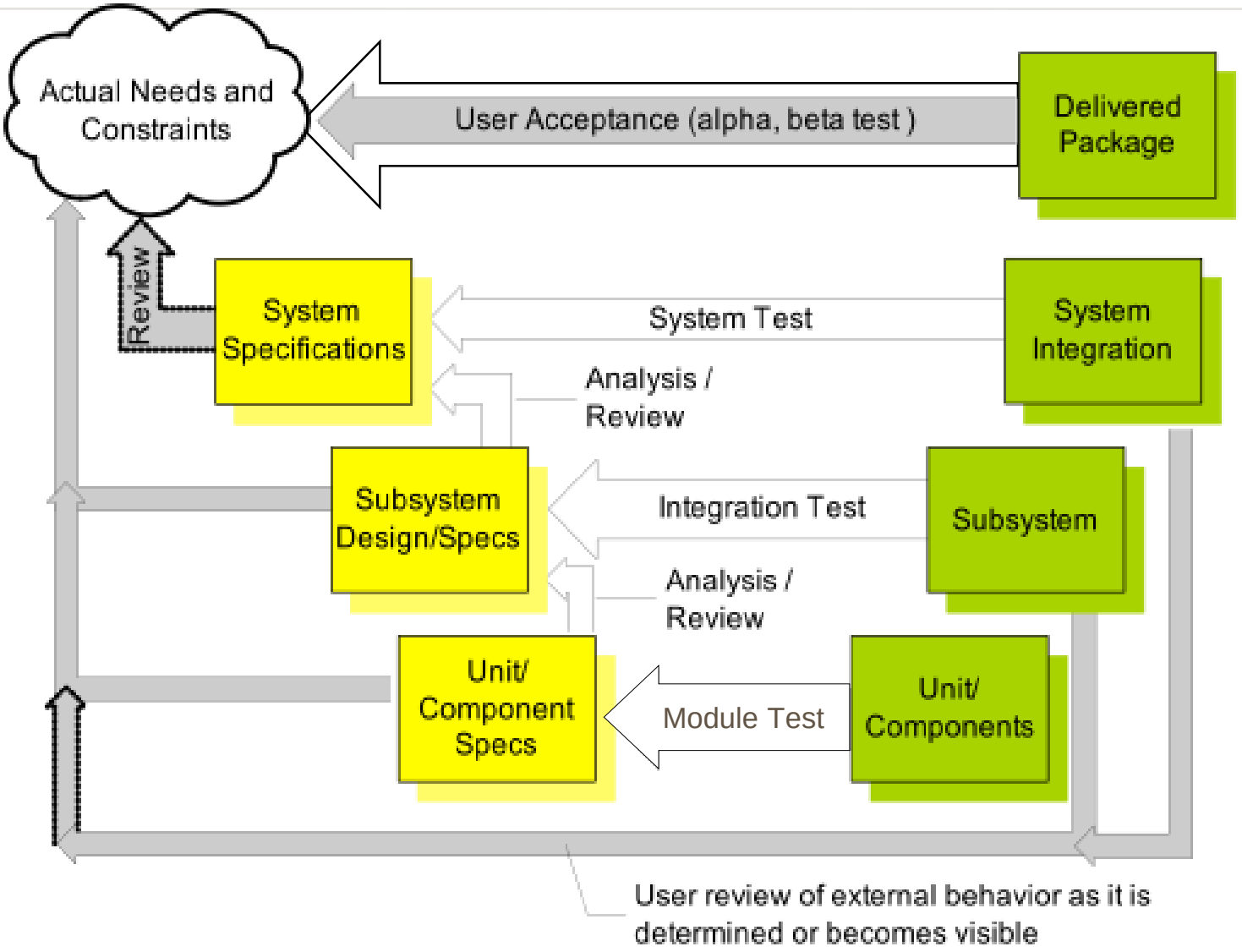


FIRST CODE, THEN TEST

1. Developers on software should do testing all
2. Software should be “toss over wall” to strangers who will test it mercilessly
3. Testers should get involved with the project only when testing is about to begin

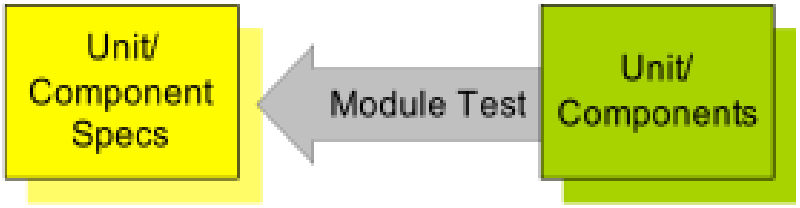
WRONG

V MODEL

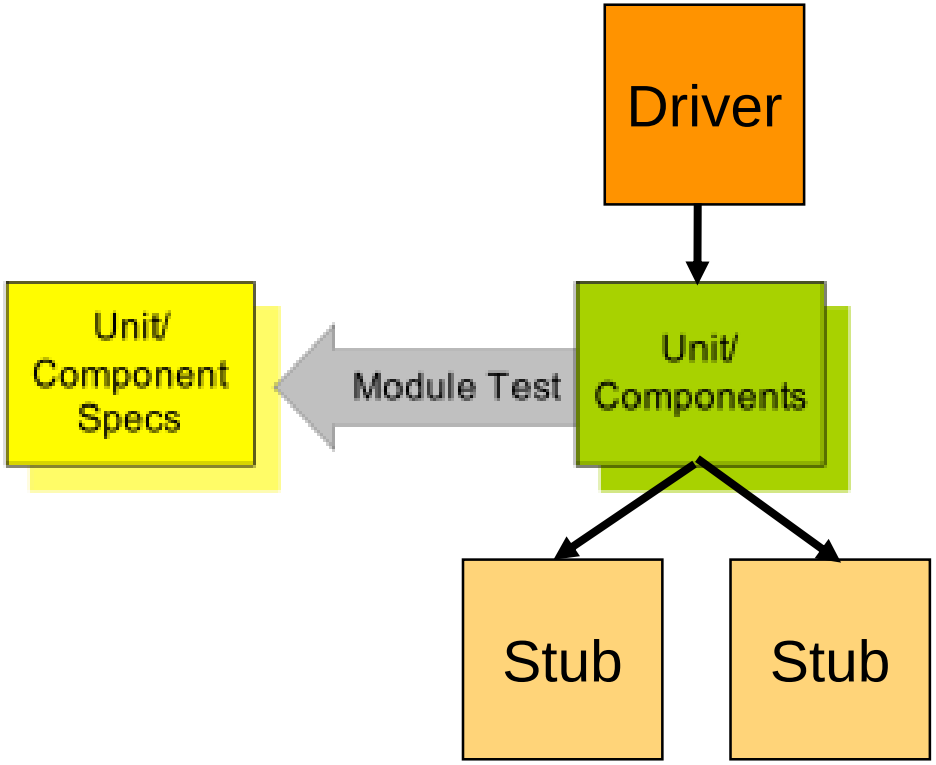

Actual Needs and Constraints
User Acceptance (alpha, beta test)
Delivered Package
Review
System Specifications
System Test
System Integration
Analysis / Review
Subsystem Design/Specs
Integration Test
Subsystem
Analysis / Review
Unit/ Component Specs
Module Test
Unit/ Components
User review of external behavior as it is determined or becomes visible

UNIT TESTS

- Aims to uncover errors at module boundaries
- Typically written by programmer herself
- Should be completely automatic (→ regression testing)



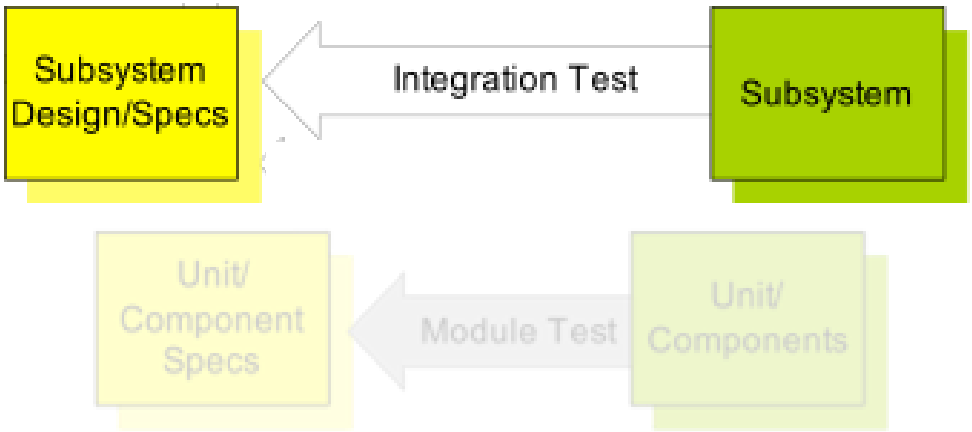
TESTING COMPONENTS: STUBS AND DRIVERS



- A *driver* exercises a module's functions
- A *stub* simulates not-yet-ready modules
- Frequently realized as *mock objects*

PUTTING THE PIECES TOGETHER: INTEGRATION TESTS

- General idea: *Constructing software while conducting tests*
- Options: Big Bang or Incremental Construction



BIG BANG APPROACH

All components are combined in advance

The entire program is tested as a whole

BIG BANG APPROACH

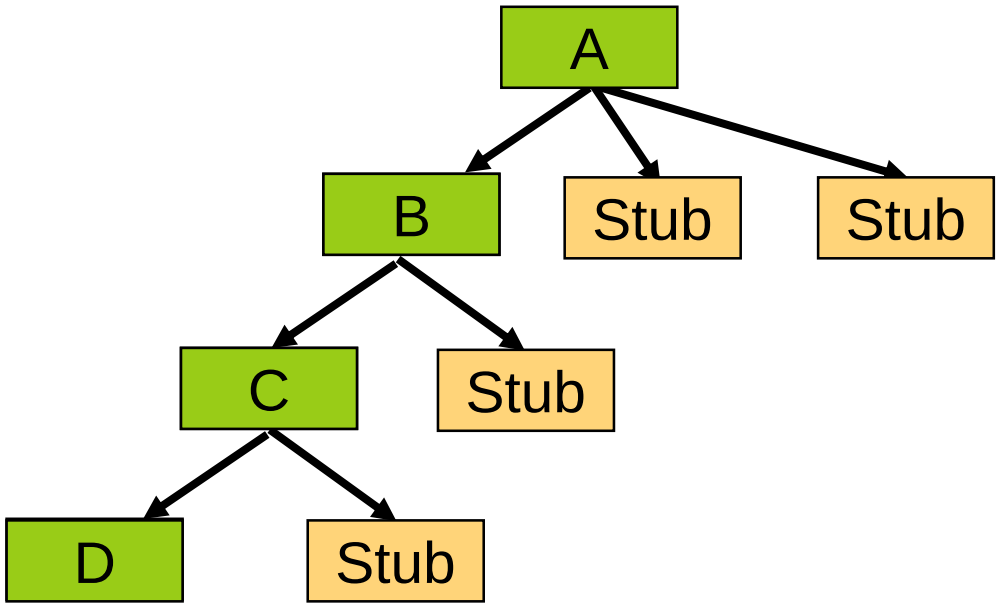
All components are combined in advance

The entire program is tested as a whole

CHAOS RESULTS!

For every failure, the entire program must be taken into account

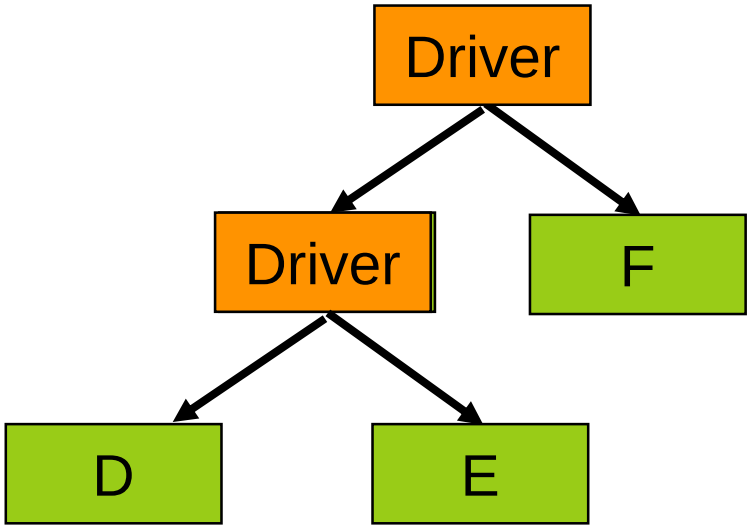
TOP-DOWN INTEGRATION



- Top module is tested with stubs (and then used as driver)
- Stubs are replaced one at a time ("depth first")
- As new modules are integrated, tests are re-run

Allows for early demonstration of capability

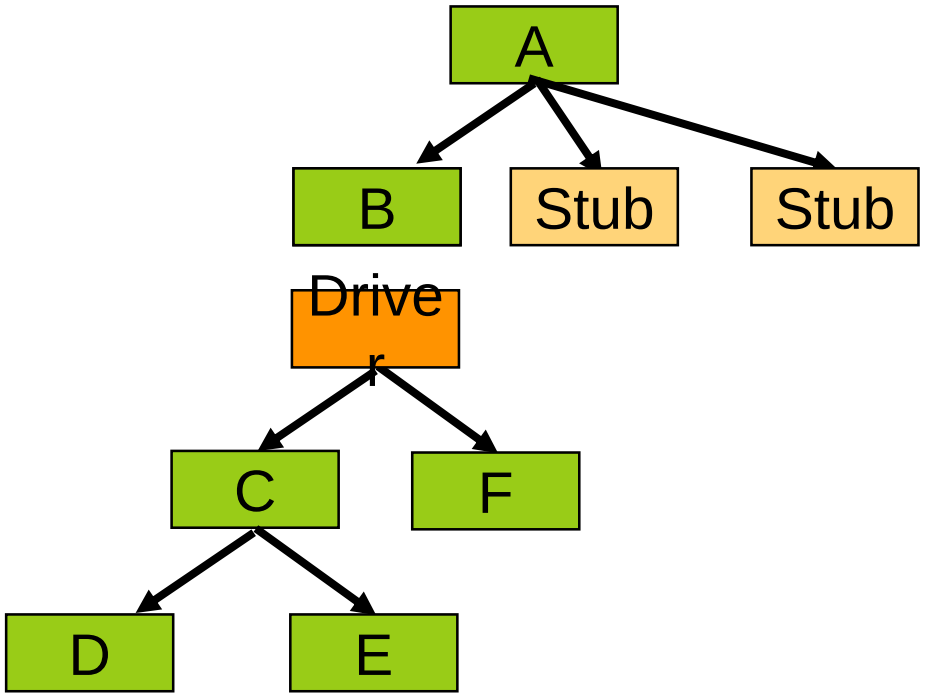
BOTTOM-UP INTEGRATION



- Bottom modules implemented first and combined into clusters
- Drivers are replaced one at a time
- Removes the need for complex stubs

Allows for early demonstration of capability

SANDWICH INTEGRATION



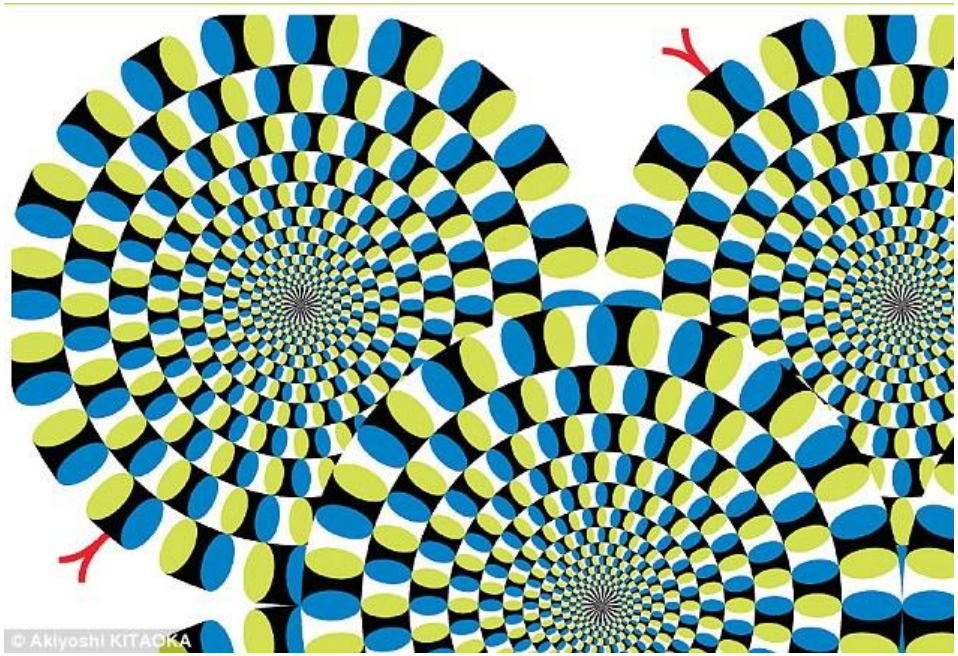
- Combines bottom-up and top-down integration
- Top modules tested with stubs, bottom modules with drivers

Combines the best of the two approaches

ONE DIFFERENCE FROM UNIT TESTING: EMERGENT BEHAVIOR

Some behaviors are only clear when components are put together

Usually this is identified after the fact, and causes test suites/cases to be refactored.



WHO TESTS THE SOFTWARE?

Developer

understands the system

but will test gently
driven by delivery

Independent Tester

must learn about system

will attempt to break it
driven by quality

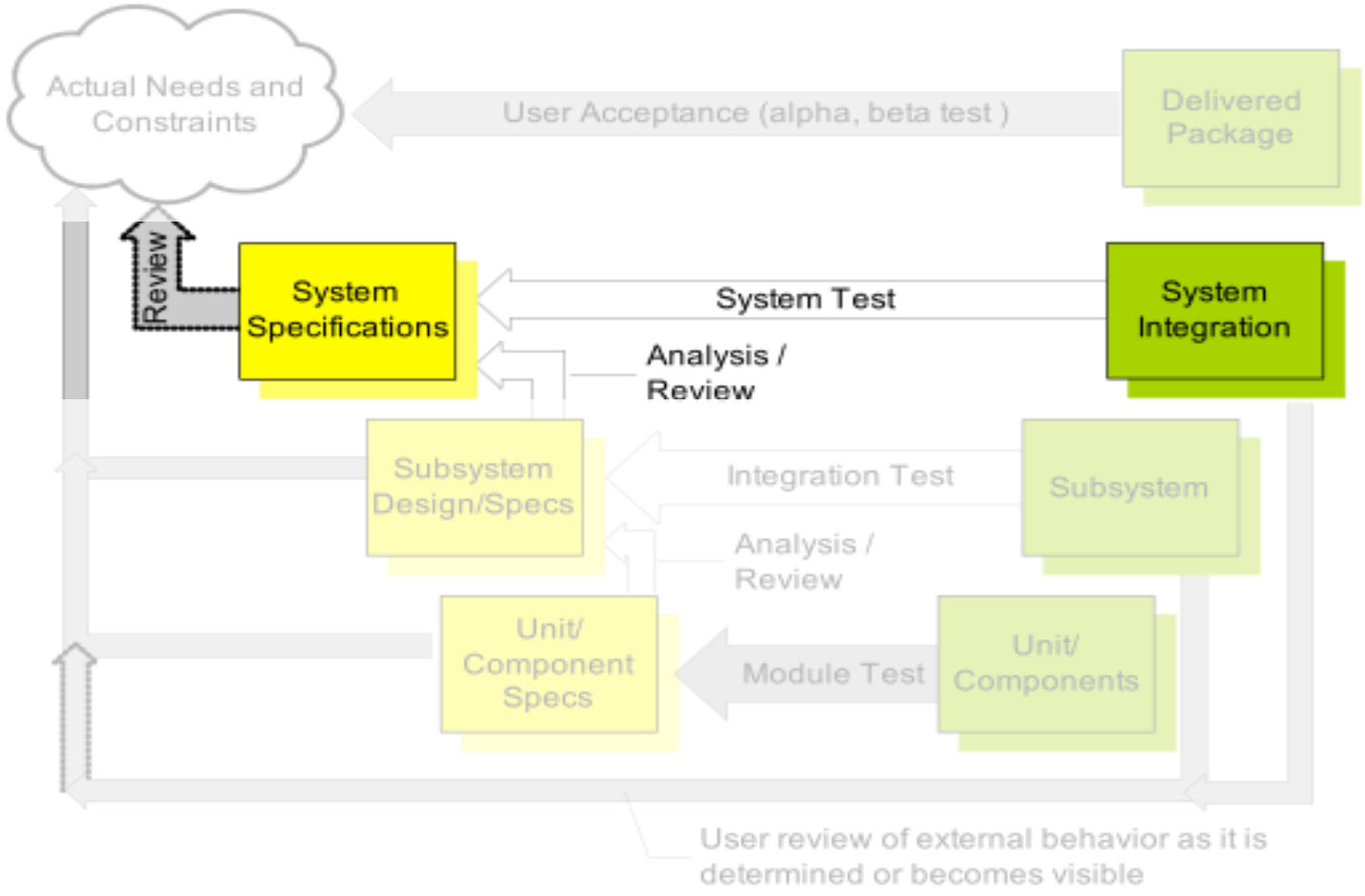
WEINBERG'S LAW

A developer is unsuited to test his or her code.

EVERYONE IS A TESTER!

- **Experienced Outsiders and Clients**
  - Good for finding gaps missed by developers, especially domain specific items
- **Inexperienced Users**
  - Good for illuminating other, perhaps unintended uses/errors
- **Mother Nature**
  - Crashes tend to happen during an important client/customer demo…

SYSTEM TESTING



SPECIAL KINDS OF SYSTEM TESTING

- **Recovery testing**
  forces the software to fail in a variety of ways and verifies that recovery is properly performed

- **Security testing**
  verifies that protection mechanisms built into a system will, in fact, protect it from improper penetration

- **Stress testing**
  executes a system in a manner that demands resources in abnormal quantity, frequency, or volume

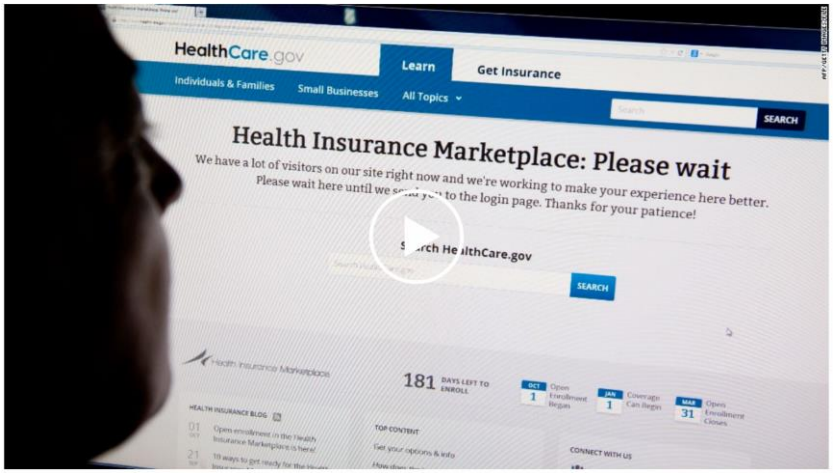
- **Performance testing**
  test the run-time performance of software within the context of an integrated system

PERFORMANCE TESTING

Measures a system's capacity to process a specific load over a specific time-span, usually:

1. *number of concurrent users*
2. *specific number of concurrent transactions*

Involves defining and running operational profiles that reflect expected use



TYPES OF PERFORMANCE TESTING

1. Load

   Aims to assess compliance with non-functional requirements

2. Stress

   Identifies system capacity limits

3. Spike

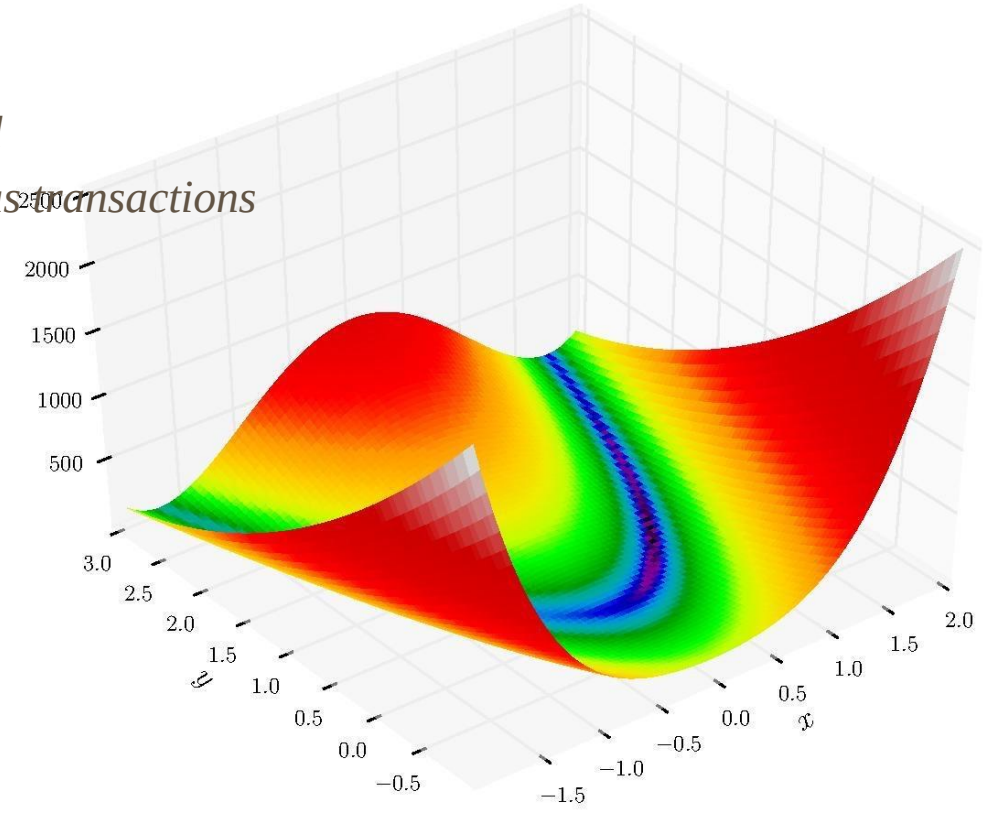
   Testing involving rapid swings in load

4. Endurance (or Soak)

   Continuous operation at a given load

MANY OPTIMIZATIONS ARE POSSIBLE

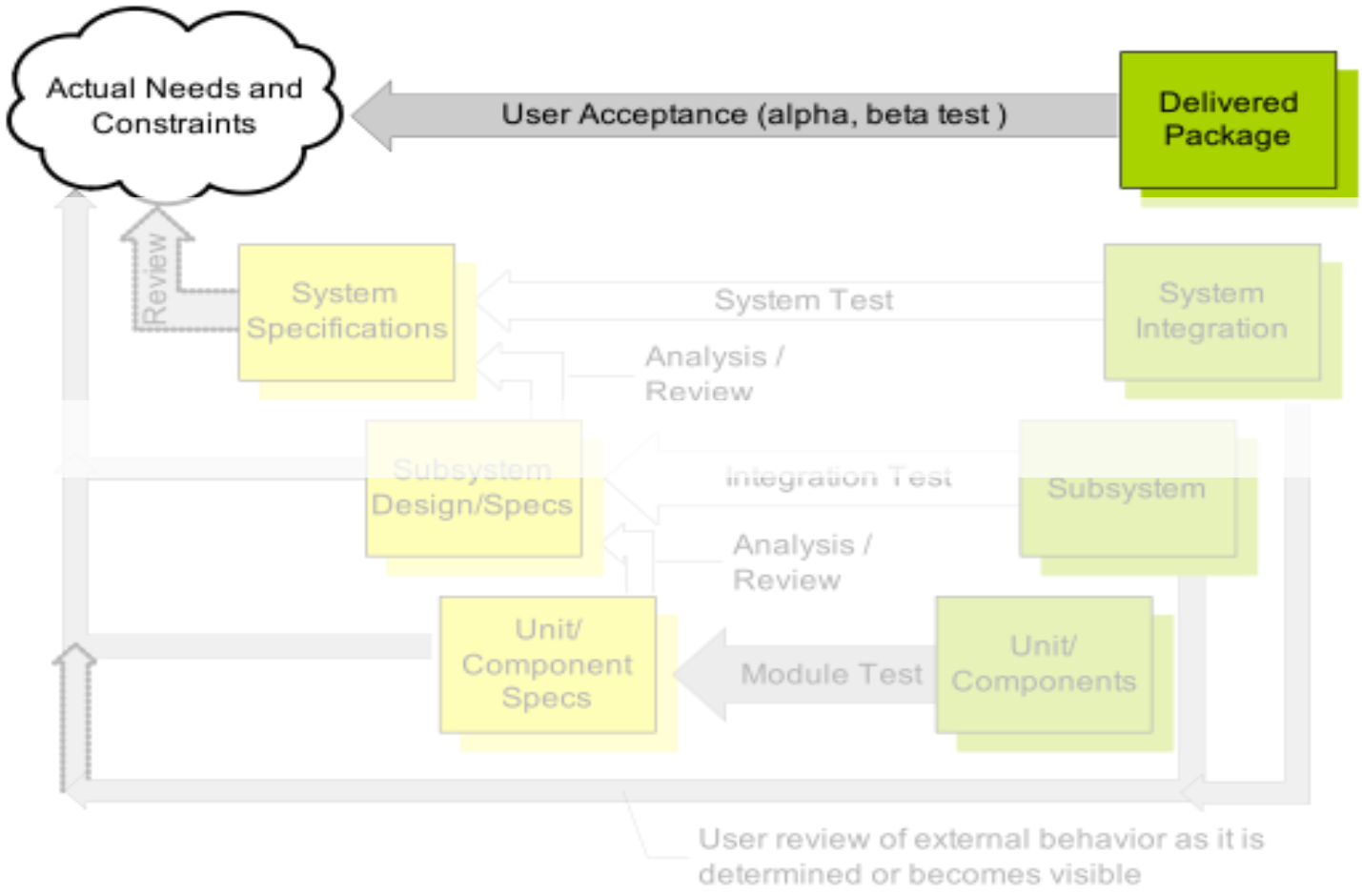
- For Throughput or Concurrency?
  - *Getting the most data processed*
  - *Greatest number of simultaneous transactions*
- For Server response time?
- For Service request round-trip time?
- For Server utilization?
- For End-User Experience?
- For Cost?



SECURITY TESTING

- **Confidentiality**
  - Information protection from unauthorized access or disclosure
- **Integrity**
  - Information protection from unauthorized modification or destruction
- **Availability**
  - System protection from unauthorized disruption

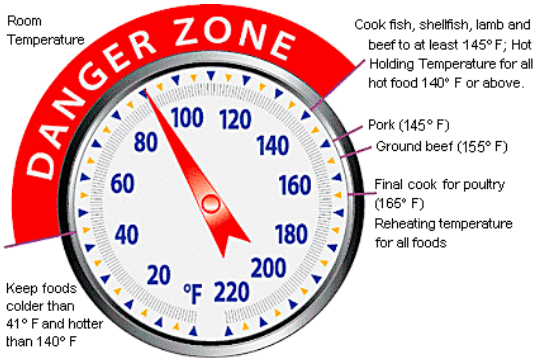
ACCEPTANCE TESTING



ACCEPTANCE TESTING

- Acceptance testing checks whether contractual requirements are met
- May be incremental
  - Alpha / Beta
- Work is over when acceptance testing is done

HOW DO WE KNOW WHEN A PRODUCT IS READY?

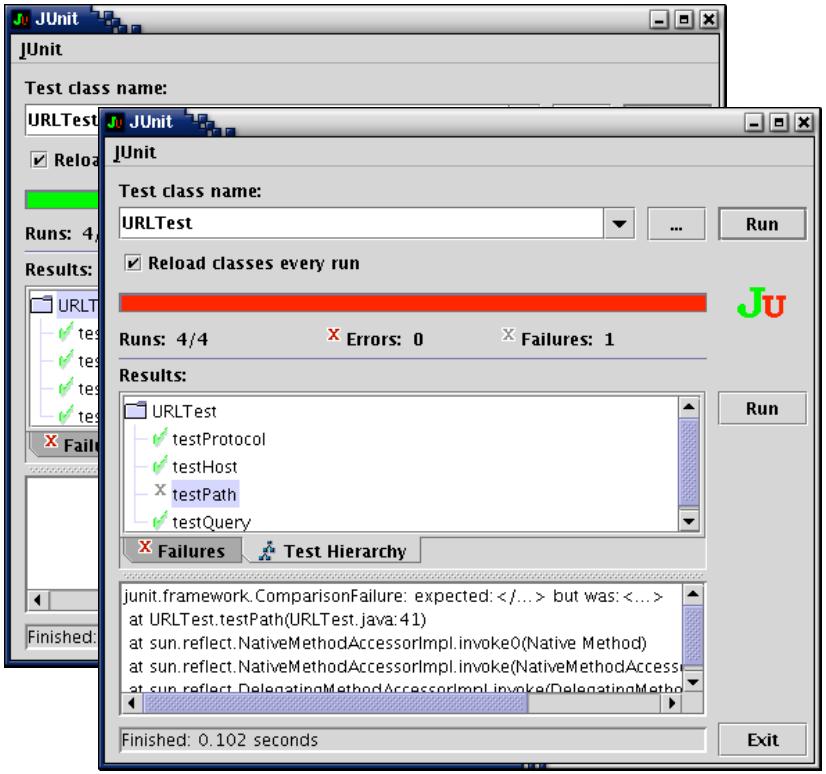


- Let the customer test it :-)
- We're out of time…
- Relative to a theoretically sound and experi… validated statistical model, we have done sufficient testi… with 95% confidence that the probability of 1,000 CPU … failure-free operation is ≥ 0.995.

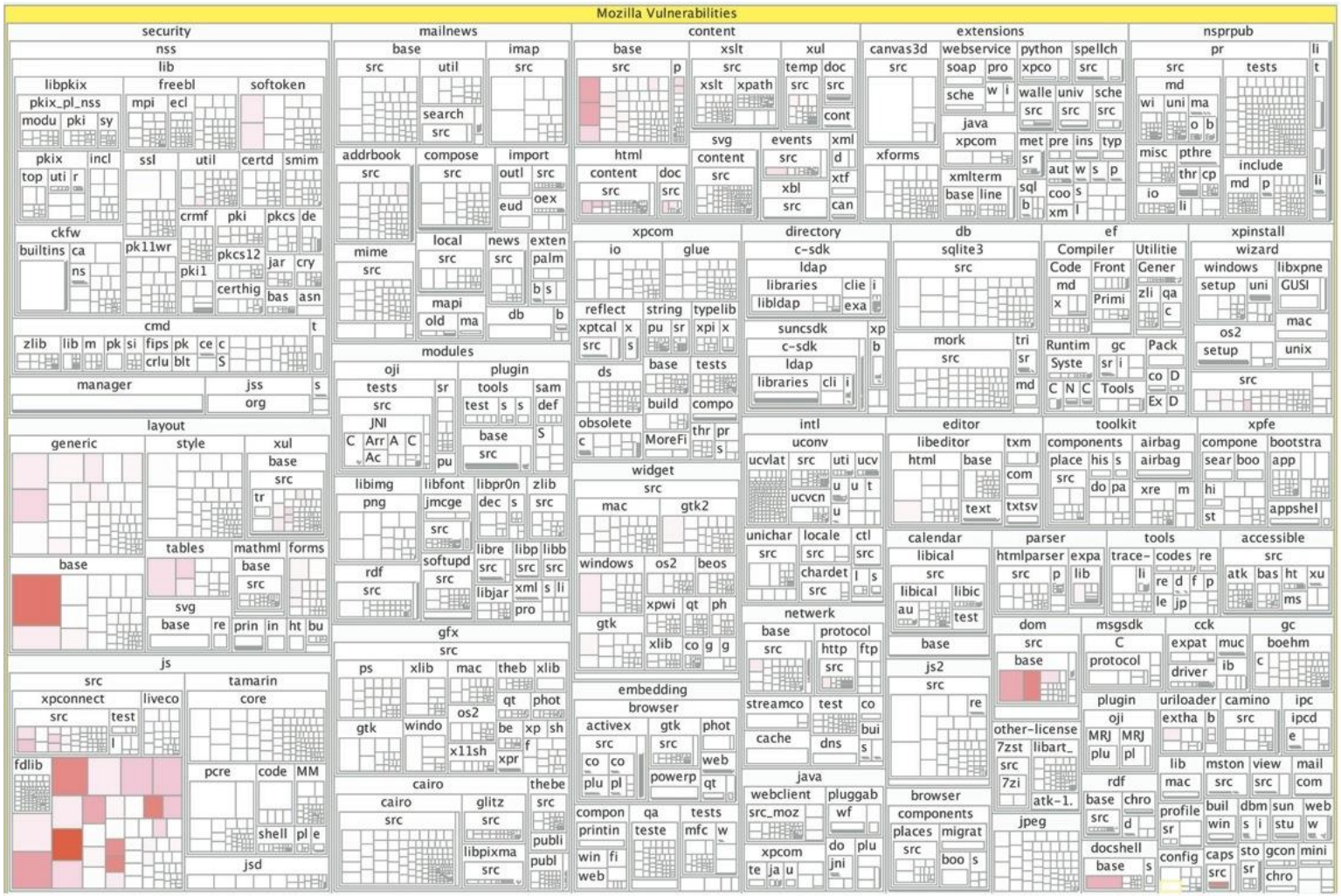
This is the type of argument we aim for.

REGRESSION TESTS

- set up automated tests
  - using, e.g., JUnit
- ideally, run regression tests after each change
- if running the tests takes too long:
  - prioritize and run a subset
  - apply regression test selection to determine tests that are impacted by a set of changes



COLLECTING DATA



REMEMBER PARETO'S LAW

Approximately 80% of defects come from 20% of modules

CORE QUESTIONS

- When does V&V start? When is it done?
- Which techniques should be applied?
- How do we know a product is ready?
- How can we control the quality of successive releases?
- How can we improve development?

BEST PRACTICES

- Specify requirements in a quantifiable manner
- State testing objectives explicitly
- Understand the users of the software and develop a profile for each user category
- Develop a testing plan that emphasizes “rapid cycle testing”

BEST PRACTICES

- Build “robust” software that is designed to test itself
- Use effective formal technical reviews as a filter prior to testing
- Conduct formal technical reviews to assess the test strategy and test cases themselves
- Develop a continuous improvement approach for the testing process

DESIGN FOR TESTING

- OO design principles also improve testing
  - Encapsulation leads to good unit tests
- Provide diagnostic methods
  - Primarily used for debugging, but may also be useful as regular methods
- Assertions are great helpers for testing
  - Test cases may be derived automatically