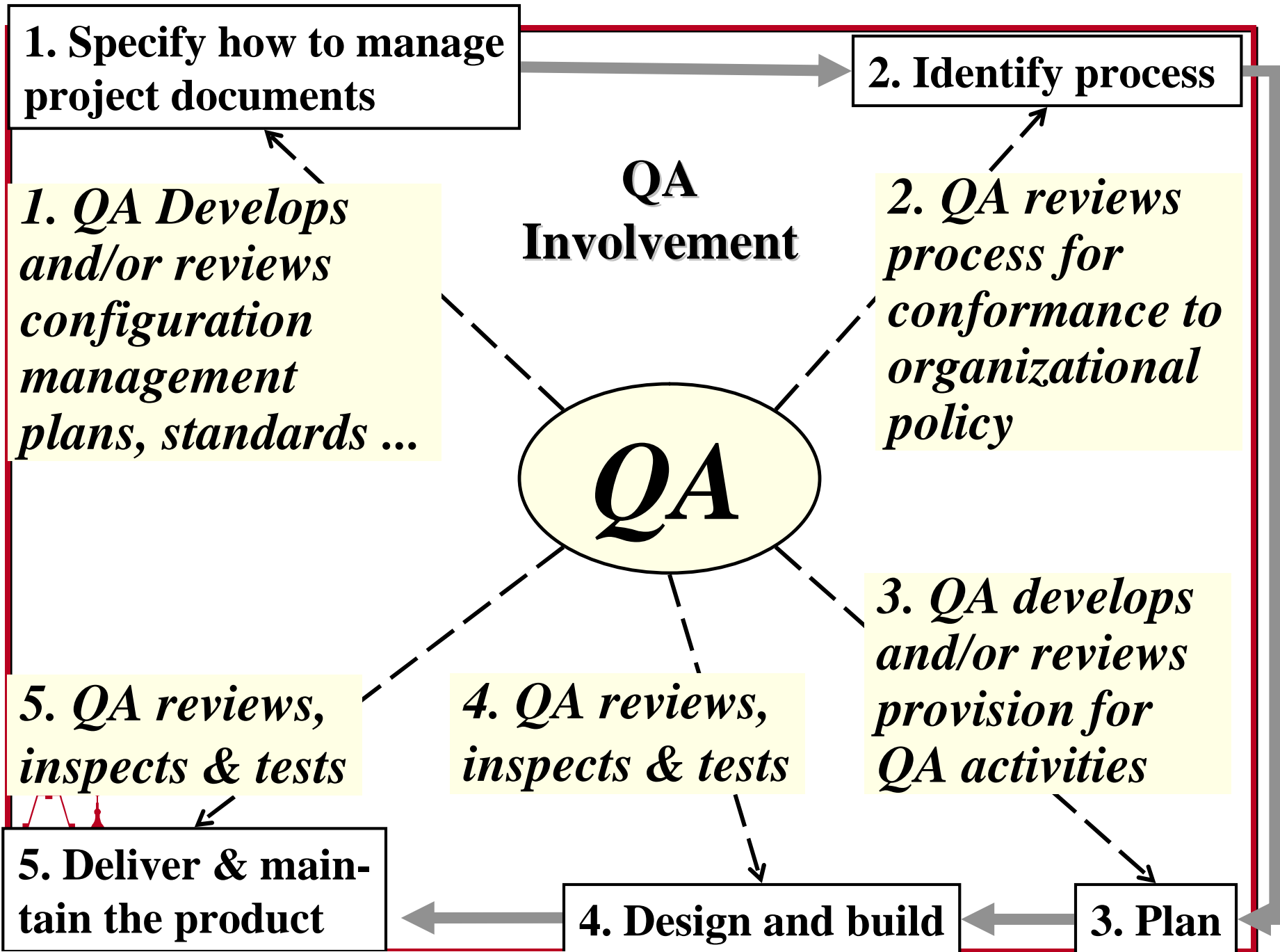


CS3733: Software Engineering

Lecture #10





Adapted from *Software Engineering: An Object-Oriented Perspective* by Eric J. Braude (Wiley 2001), with permission.

Principle of Inspection

***AUTHORS CAN USUALLY
REPAIR DEFECTS
THAT THEY RECOGNIZE***



Principle of Inspection

***AUTHORS CAN USUALLY
REPAIR DEFECTS
THAT THEY RECOGNIZE***

COROLLARY:

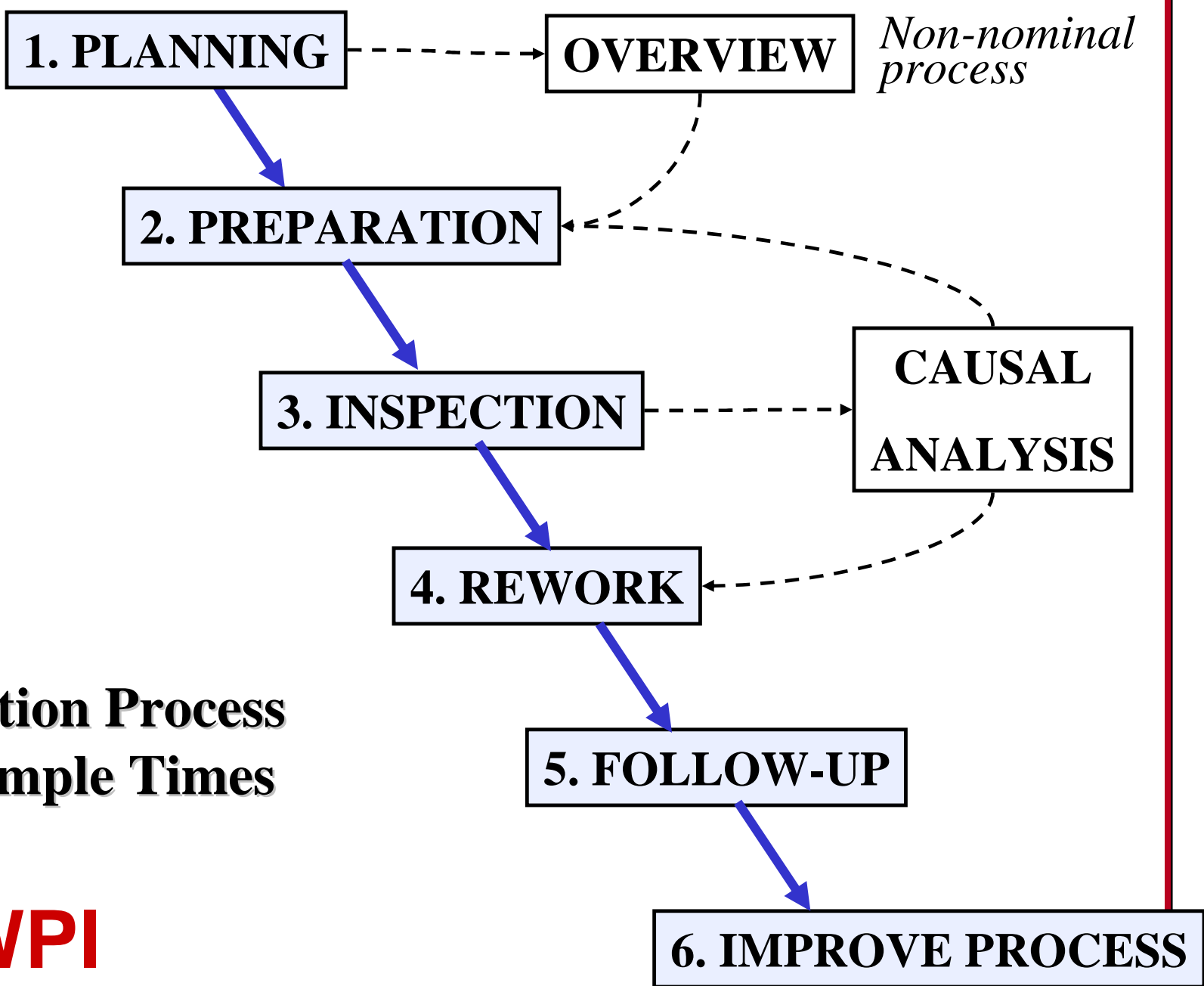
**Help authors to recognize defects
before they deliver their work.**

COROLLARY:

Have their peers seek defects.



Nominal process









Non-nominal process

Inspection Process & Example Times



Time/Costs per 100 LoC* -- one company's estimates

| | |
|------------------------------|---|
| Planning | 1 hr  (1 person) |
| <i>[Overview</i> | <i>1 hr</i>  <i>(3-5)]</i> |
| Preparation | 1 hr  (2-4 people) |
| Inspection meeting | 1 hr  (3-5 people) |
| Rework | 1 hr  (1 person) |
| <i>[Analysis</i> | <i>1 hr</i>  <i>(3-5)]</i> |
| <u>Total: approx.</u> | <u>7 - 21 person-hours</u> |



* lines of non-commented code

Hours Per Defect: One estimate

If defect found ...

| | ... at inspection time | ... at integration time |
|---------------------|-----------------------------------|------------------------------------|
| Hours to .. | | |
| .. detect | 0.7 to 2 | 0.2 to 10 |
| .. repair | 0.3 to 1.2 | 9+ |
| <u>Total</u> | <u>1.0 to 3.2</u> | <u>9.2 to 19+</u> |





Prepare For & Conduct Inspections

1. Build inspections into the project schedule

- plan to inspect all phases, starting with requirements
- allow for preparation (time consuming!) & meeting time

2. Prepare for collection of inspection data

- include # defects per work unit (e.g., KLOC), time spent
- develop forms: include *description*, *severity* and *type*
- decide who, where, how to store and use the metric data
 - default: appoint a single person to be responsible
 - failure to decide usually results in discarding the data

3. Assign roles to participants

- Three adequate (*author; moderator/recorder; reader*)
- Two far better than none (*author; inspector*)

4. Ensure every participant prepares

- bring defects pre-entered on forms to inspection meeting



IEEE 730-1989 Software Quality Assurance Plans Table of Contents

- 1. Purpose**
- 2. Referenced documents**
- 3. Management**
 - 3.1 Organization
 - 3.2 Tasks
 - 3.3 Responsibilities
- 4. Documentation**
 - 4.1 Purpose
 - 4.2 Minimum documentation requirements
 - 4.3 Other
- 5. Standards, practices, conventions and metrics**
 - 5.1 Purpose
 - 5.2 Content



~~IEEE 730-1989 Software Quality Assurance Plans Table of Contents~~

1. Purpose

2. Referenced documents

3. Management

3.1 Organization

3.2 Tasks

3.3 Responsibilities

4. Documentation

4.1 Purpose

4.2 Minimum documentation requirements

4.3 Other

5. Standards, practices, conventions and metrics

5.1 Purpose

5.2 Content



6. Reviews and audits

6.1 Purpose

6.2 Minimum requirements

6.2.1 Software requirements review

6.2.2 Preliminary design review

6.2.3 Critical design review

6.2.4 SVVP review

6.2.5 Functional audit

6.2.6 Physical audit

6.2.7 In-process audits

6.2.8 Managerial review

6.2.9 SCMP review

6.2.10 Post mortem review

6.3 Other

7. - 15. -- see next chapter

Meaning of “V&V” (Boehm)

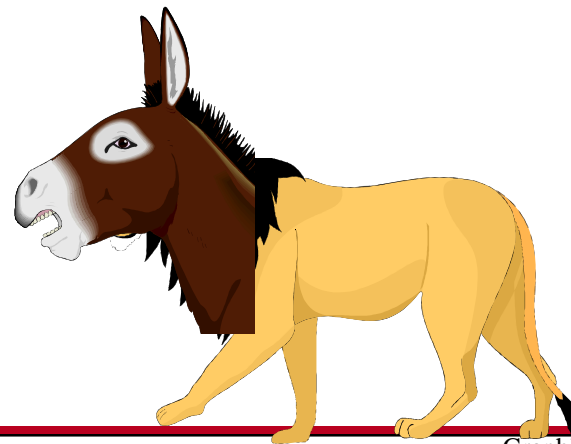


Verification:

are we building *the thing right?*

Validation:

are we building *the right thing?*



IEEE 1012-1986
Software Verification &
validation Plans Table of
Contents (Reaffirmed 1992)

- 1. Purpose**
- 2. Referenced documents**
- 3. Definitions**
- 4. V&V overview**
 - 4.1 Organization
 - 4.2 Master schedule
 - 4.3 Resource summary
 - 4.4 Responsibilities
 - 4.5 Tools, techniques & methodologies
- 5. Lifecycle V&V**
 - 5.1 Management of V&V
 - 5.2 Concept phase V&V
 - 5.3 Requirements phase V&V
 - 5.4 Design phase V&V
 - 5.5 Implementation phase V&V¹²



IEEE 1012-1986 Software Verification & validation Plans Table of Contents

(Reaffirmed 1992)

1. Purpose

2. Referenced documents

3. Definitions

4. V&V overview

4.1 Organization

4.2 Master schedule

4.3 Resource summary

4.4 Responsibilities

4.5 Tools, techniques & methodologies

5. Lifecycle V&V

5.1 Management of V&V

5.2 Concept phase V&V

5.3 Requirements phase V&V

5.4 Design phase V&V

5.5 Implementation phase V&V

5.3 Test phase V&V

5.4 Installation & checkout phase V&V

5.5 Operation & maintenance phase V&V

6. Software V&V reporting

6.1 Required reports

6.2 Optional reports

7. V&V administrative procedures

7.1 Anomaly reporting & resolution

7.2 Task iteration policy

7.3 Deviation policy

7.4 Standards, practices & conventions



Produce a Quality Product

1. Quantify your quality goals

minimum: *number of defects per KLOC*

team: *# defective requirements; # classes missing from design;
defects in testing; # defects found in operation.*

personal: *apply # defects to code, compile, unit test separately*

2. Build inspections and reviews into the schedule

(see scheduling, next chapter)

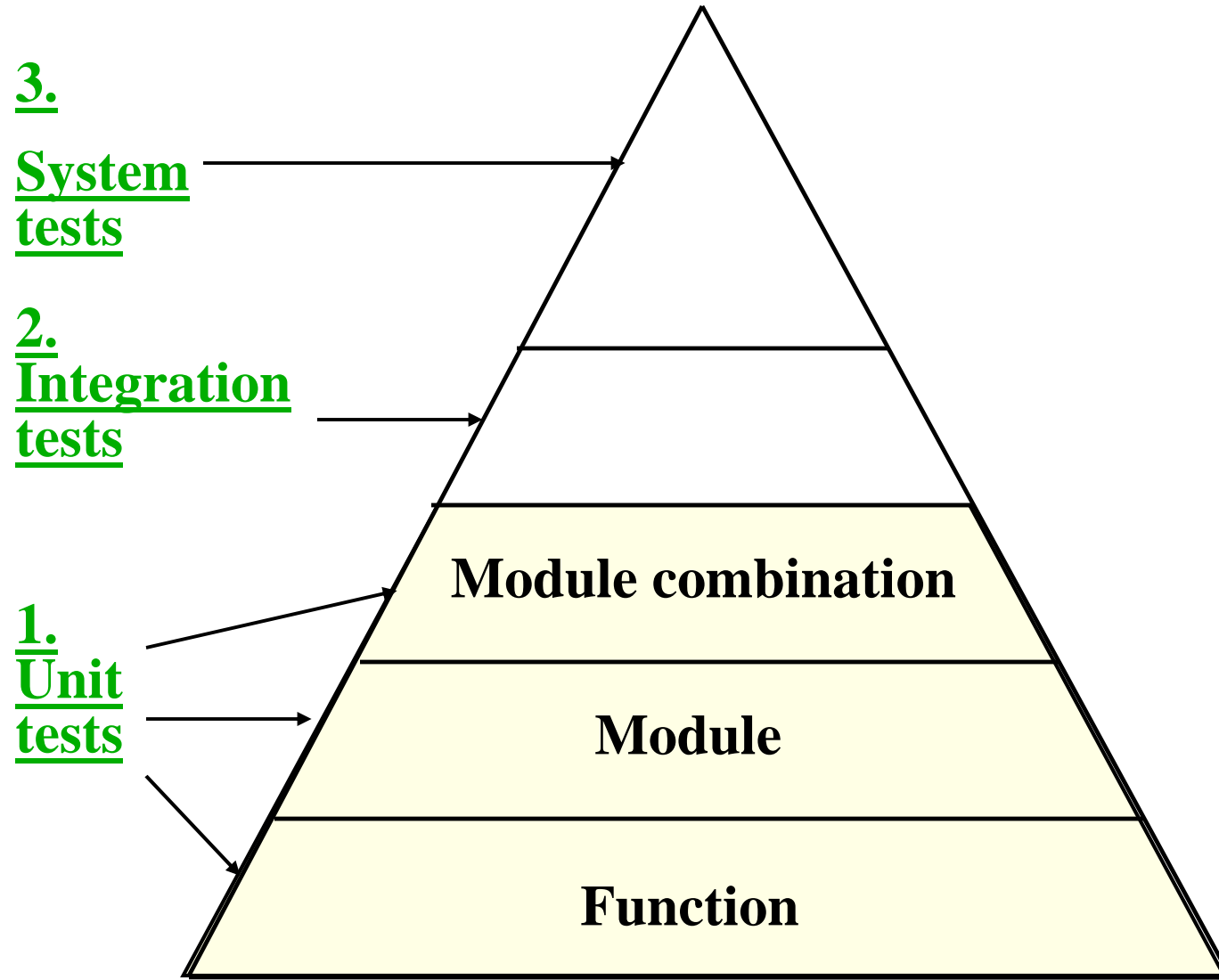
follow the inspection procedure (see figure 1.27 on page ??)

3. Document your quality goals and procedures

use a documentation standard to avoid missing issues

SQAP (see case study for example); If time allows: SVVP

Testing: the Big Picture



3.

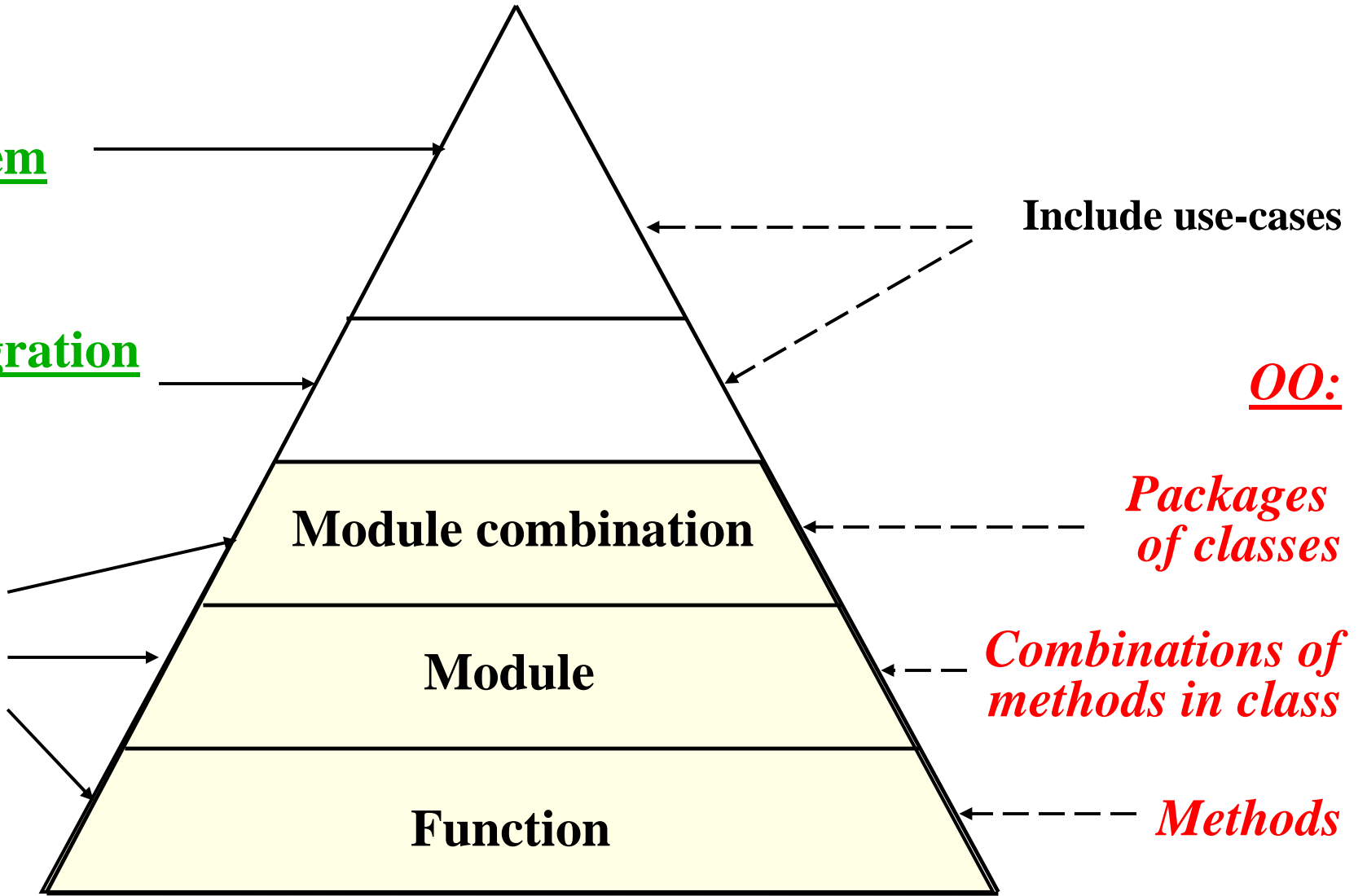
System tests

2.

Integration tests

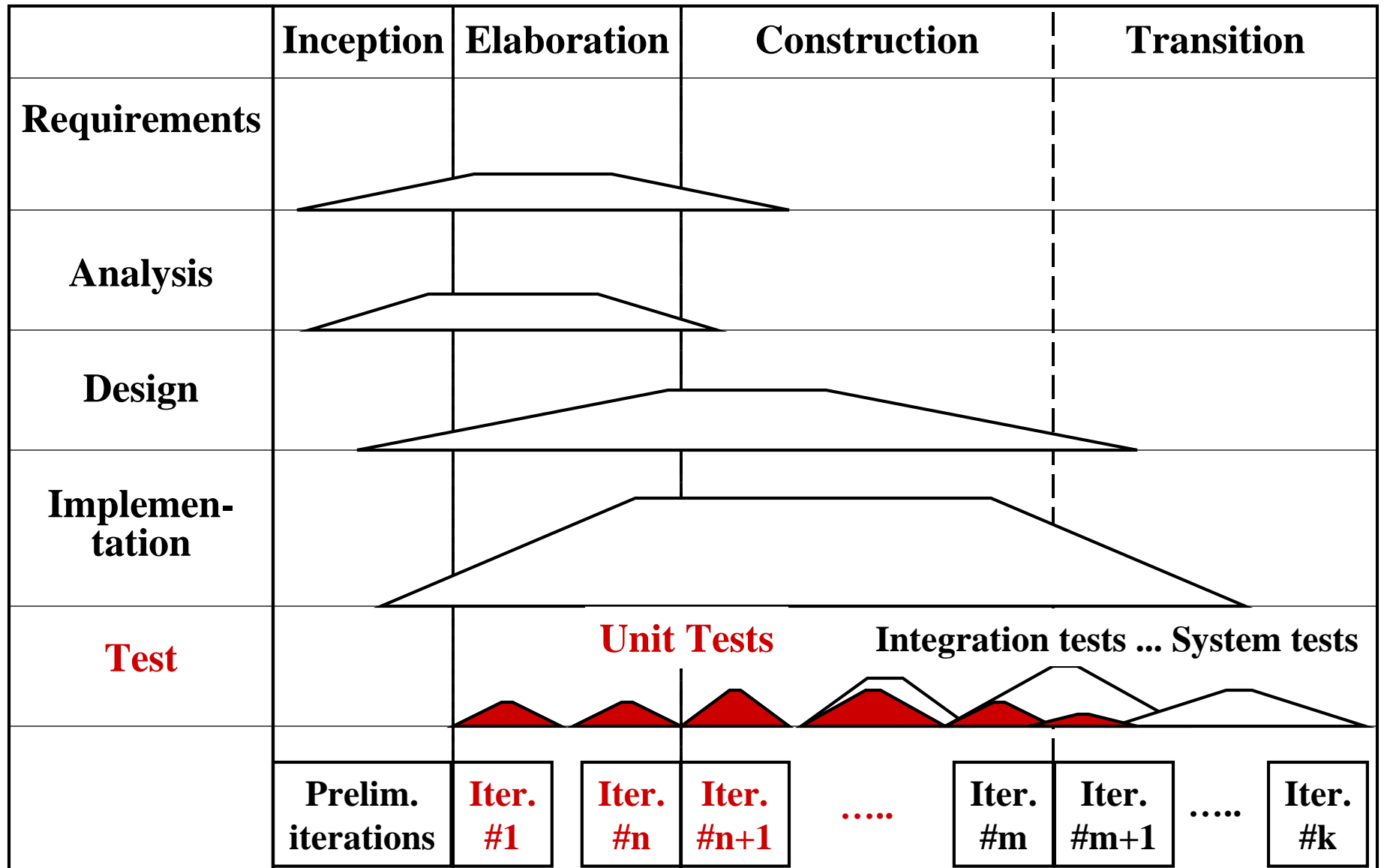
1.

Unit tests



Unified Process

Jacobson *et al*:
USDP



Input
determined
by...

**Black-, Gray-,
& White-box Testing**

Result

... requirements

Black box

*Actual output
compared
with
required output*

*... requirements &
key design elements*

Gray box

*As for black-
and white box
testing*

*... design
elements*

White box

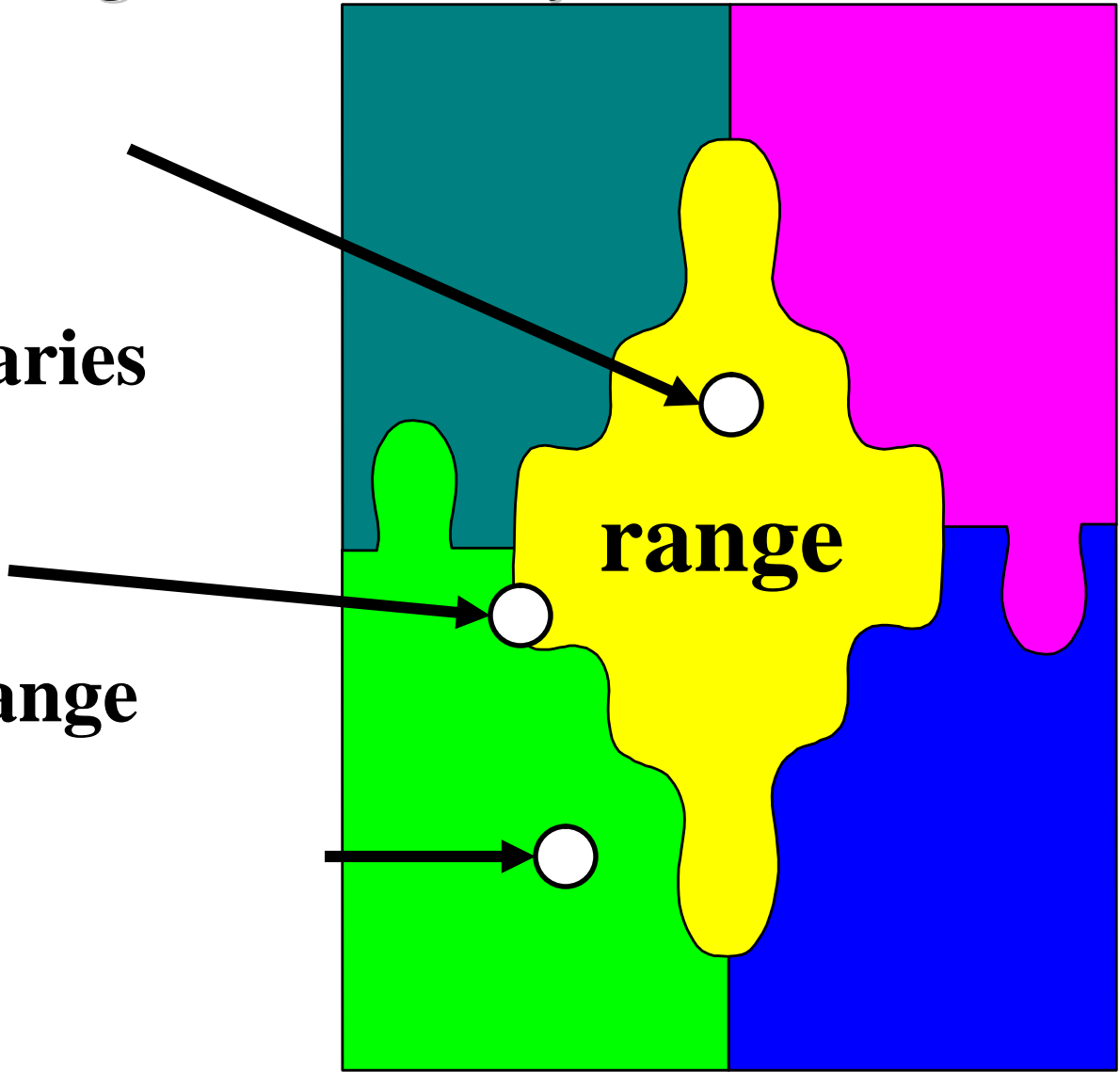
**Confirmation
of expected
behavior**

Testing Ranges: Elementary Cases

1. within range

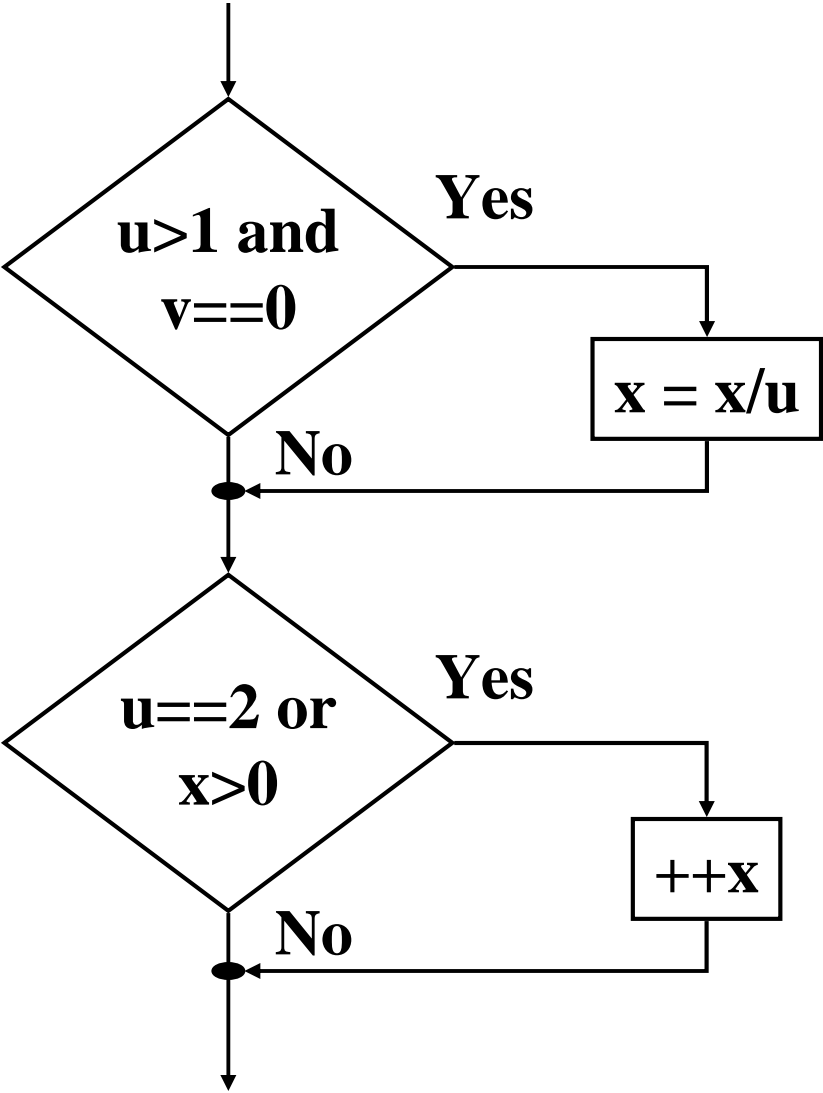
**2. at the boundaries
of the range**

**3. outside the range
("illegal")**



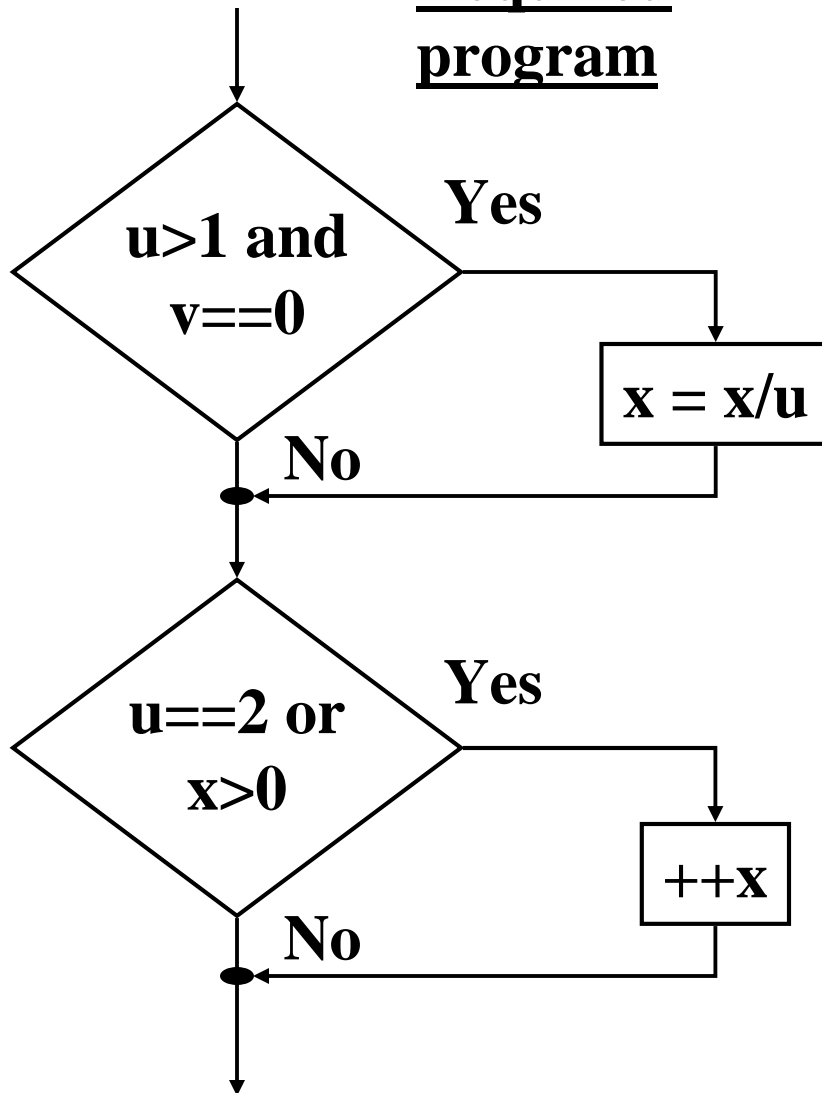
Covering Every Statement is Not Sufficient (Myers)

Required
program



Covering Every Statement is Not Sufficient (Myers)

Required
program



Code attempt to implement
flowchart

```
if( (u>1) && (v==0) ) (1)
```

```
    x = x/u; (2)
```

```
if( (u==2) || (x>3) ) (3)
```

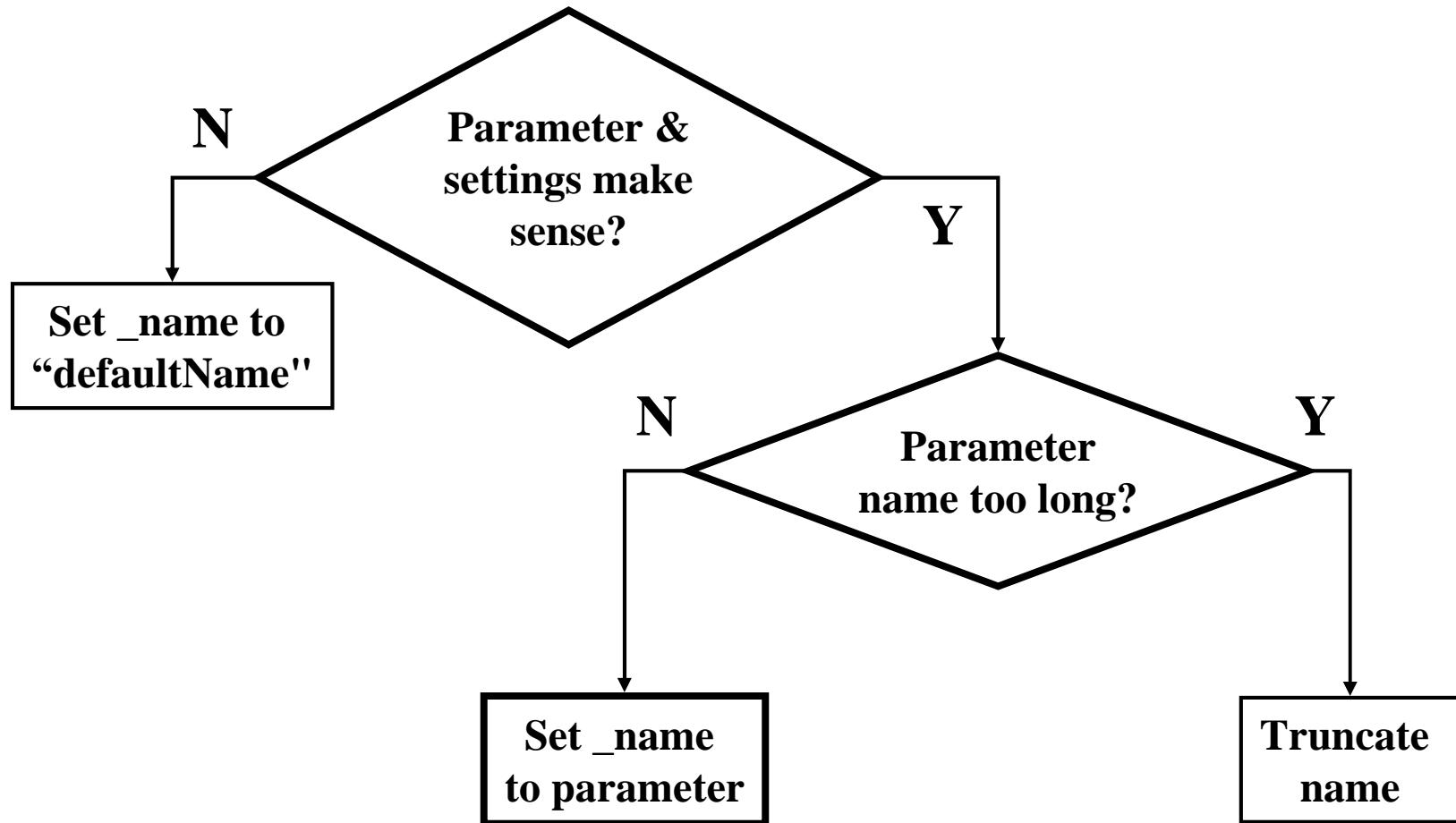
```
    ++x; (4)
```

u=2, v=0 and x=3

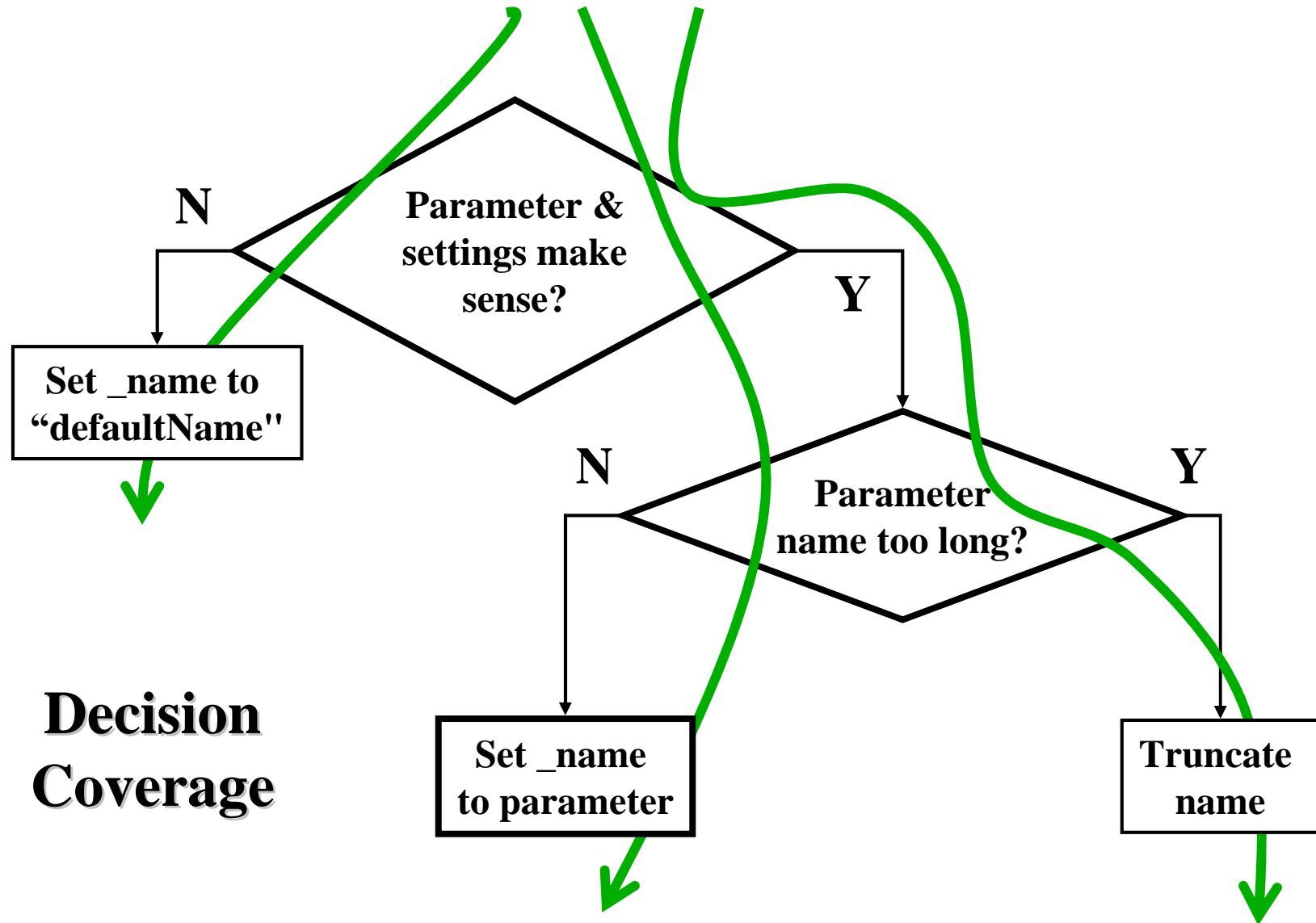
- executes every line (1) - (4)
- gives the correct output $x=2.5$

However, line (3) is wrong

Paths to be Checked



Paths to be Checked



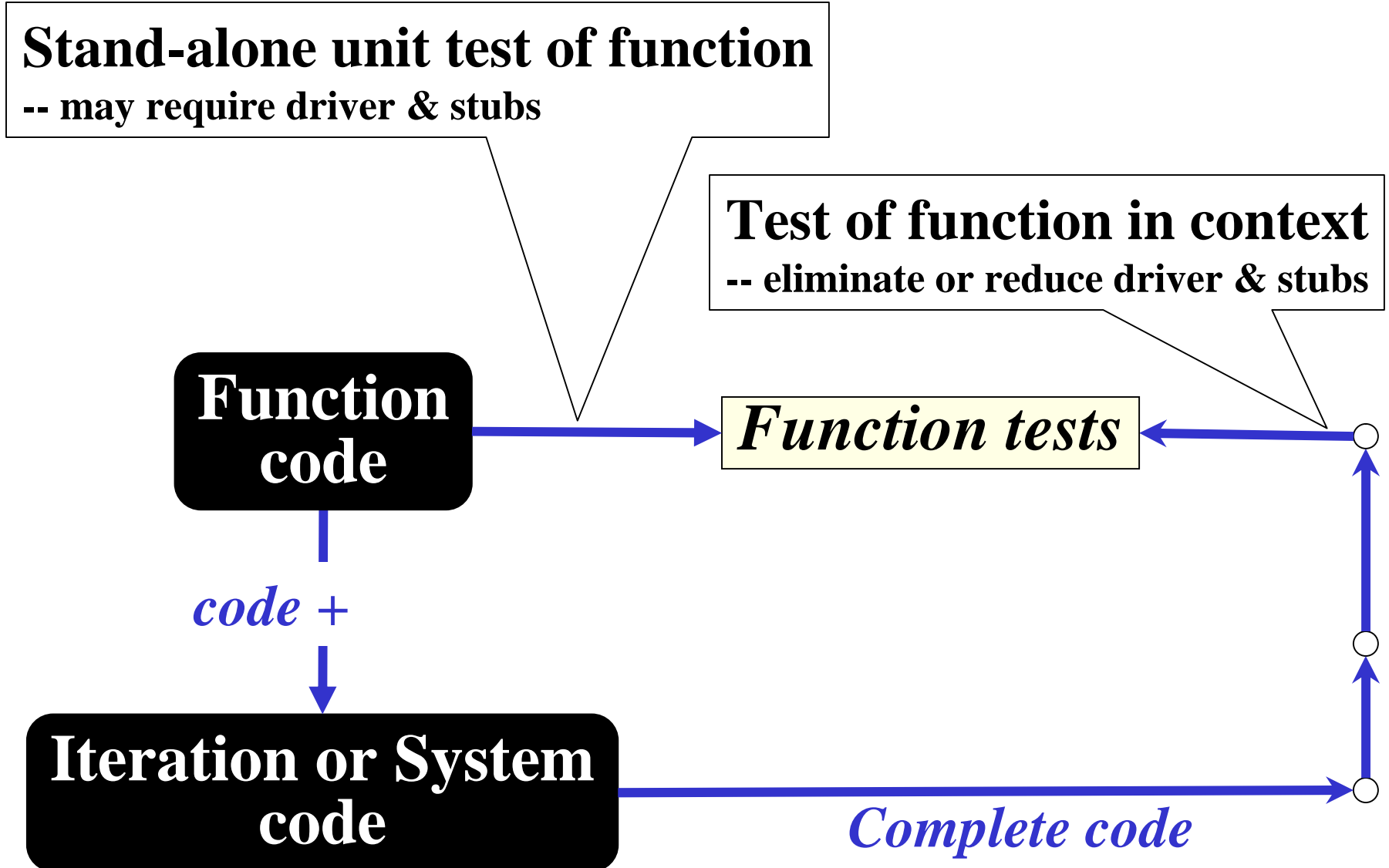
One way to Perform Method Testing (Humphrey) 1/2

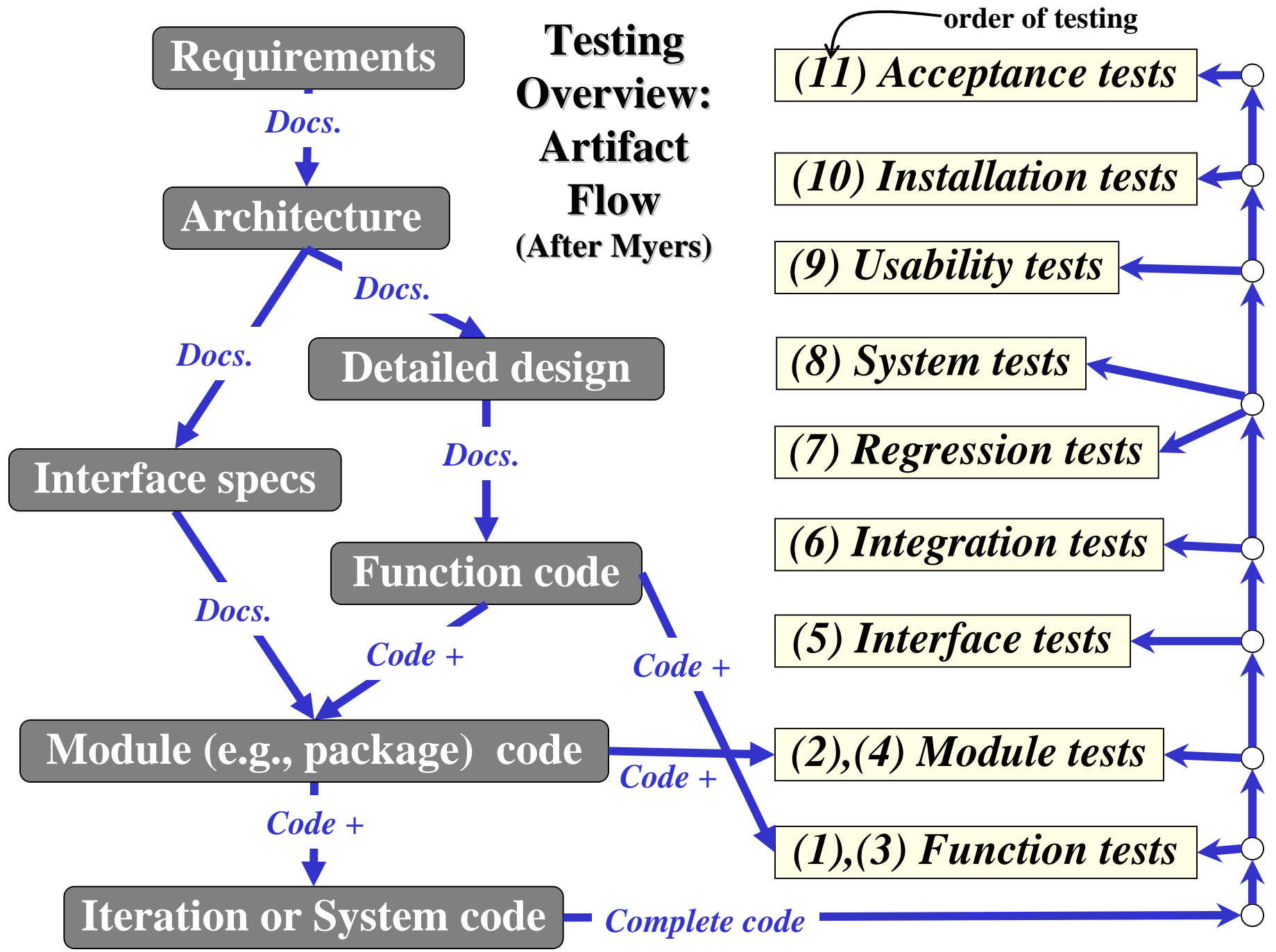
1. Verify operation at **normal parameter** values
(a black box test based on the unit's requirements)
2. Verify operation at **limit parameter** values
(black box)
3. Verify operation **outside parameter** values
(black box)
4. Ensure that **all instructions** execute
(statement coverage)
5. Check all **paths**, including both sides of all branches
(decision coverage)
6. Check the **use of all called objects**
7. Verify the handling of all **data structures**
8. Verify the handling of all **files**

One way to Perform Method Testing (Humphrey) 2/2

9. Check **normal** termination of all **loops**
(part of a correctness proof)
10. Check **abnormal** termination of all **loops**
11. Check **normal** termination of all **recursions**
12. Check **abnormal** termination of all **recursions**
13. Verify the **handling** of all **error** conditions
14. Check **timing** and **synchronization**
15. Verify all **hardware dependencies**

Testing Units in Context





Requirements

Docs.

Architecture

Docs.

Detailed design

Docs.

Interface specs

Docs.

Function code

Docs.

Module (e.g., package) code

Code +

Code +

(2),(4) Module tests

Code +

(1),(3) Function tests

Code +

Iteration or System code

Complete code

order of testing

(11) Acceptance tests

(10) Installation tests

(9) Usability tests

(8) System tests

(7) Regression tests

(6) Integration tests

(5) Interface tests

Requirements

Tested
against
requirements
("validation")

(11) *Acceptance tests**

(10) *Installation tests*‡*

(9) *Usability tests*‡*

(8) *System tests*‡*

(7) *Regression tests**

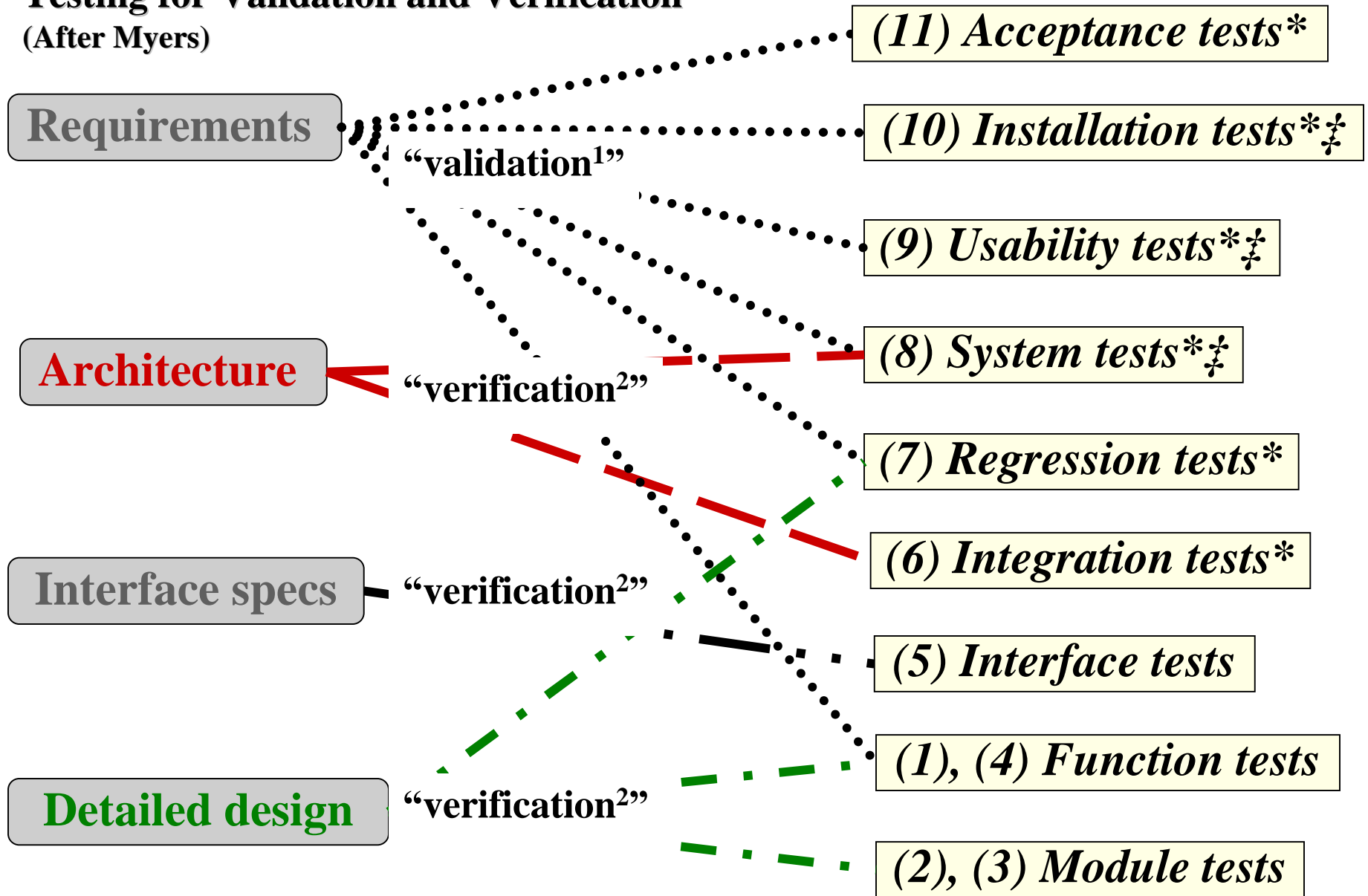
(1), (4) *Function tests*

Testing for Validation and Verification (After Myers)

Includes... : *use-cases
‡performance testing

Testing for Validation and Verification

(After Myers)



Note 1: Tested against requirements

Note 2: Tested against documents indicated

Includes... : *use-cases
‡performance testing

Lessons Learned (Kaner, Bach, Pettichord)

- All tests are an attempt to answer some question
- A requirement is a quality or condition that matters to someone who matters
- In the end all you have is an impression of the product
- Use heuristics to quickly generate test ideas
 - Boundaries
 - Each error message
 - Different configurations than the programmer's



Lessons Learned (Kaner, Bach, Pettichord)

- Confusion is a test tool
- To test you must explore
- Exploring involves a lot of thinking

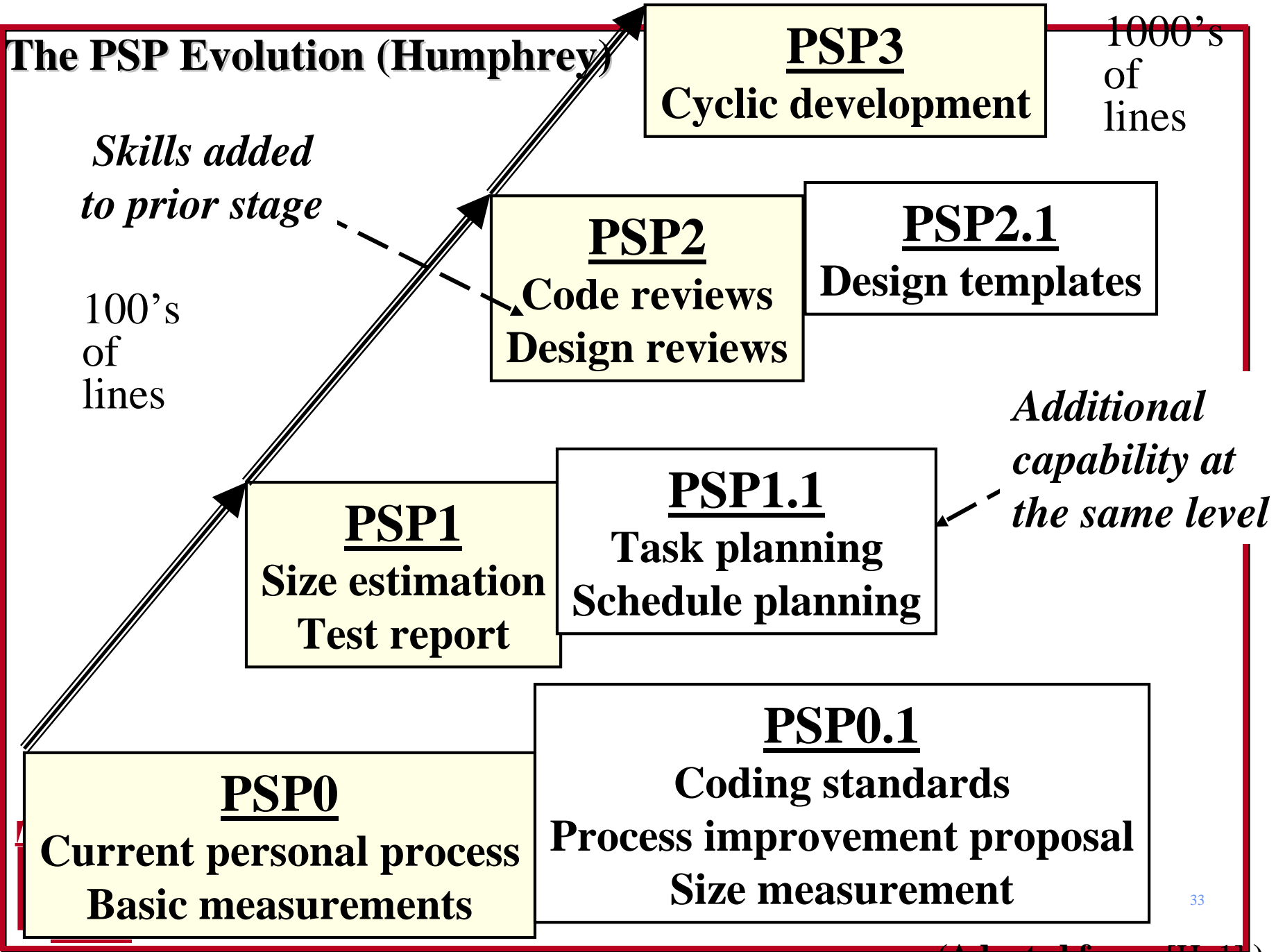


Exploratory Testing

- *Systematic* “testing on the fly”
- Output
 - Bugs
 - Test ideas
- Write everything down



The PSP Evolution (Humphrey)



(Adapted from [Hu1])

TSP Objectives 1 (Humphrey)



➤ Build self-directed teams

- 3-20 engineers
- establish *own* goals
- establish *own* process and plans
- track work

➤ Show managers how to manage teams

- coach
- motivate
- sustain peak performance



TSP Objectives 2 (Humphrey)

- **Accelerate CMM improvement**
 - **make CMM 5 “normal”**
- **“Provide improvement guidelines to high-maturity organizations”**
- **“Facilitate university teaching of industrial-grade teams”**



The Capability Maturity Model (CMM)



1. Initial (Software Engineering Institute)

Process:

**undefined,
ad hoc**

Result:

outcome depends on individuals

Lacking: **any reasonable process**



1. INITIAL Process undefined, ad hoc, depends on individuals

2. Repeatable (Software Engineering Institute)

Process

**tracks documents, cost, schedule,
functionality (after fact)**

Result

repeatable only on similar projects

Lacking: **complete process**



2. REPEATABLE Basic project management to track cost & schedule, repeatable on similar projects

3. Defined (Software Engineering Institute)

Process

**documented,
standardized,
tailorable**

Result

consistency

Lacking: **predictable outcomes**



3. DEFINED Consistent: Documented, standardized, tailorable

4. Managed (Software Engineering Institute)

Process

detailed measurement; control

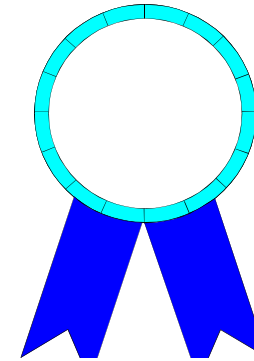
Result

**process and products with
quantified quality predictability**

Lacking **mechanism for process improvement**

4. MANAGED Predictable: process & products measured

5 Optimized (Software Engineering Institute)

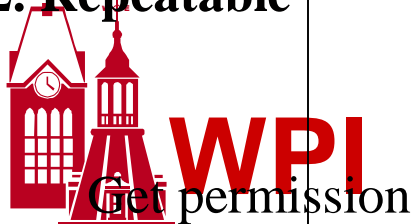


Process

**Continual process improvement
through quantitative feedback;
Extensible scope
Innovative ideas and technologies**



| <u>Level</u> | <u>Focus</u> | <u>Key Process Area</u> | <u>PSP</u> | <u>TSP</u> |
|--|---------------------------|---|--|--|
| | | | | |
| <div style="border: 1px solid black; padding: 10px; width: fit-content; margin: auto;"> Relating PSP, TSP & CMM (Humphrey) </div> | | | | |
| | | | | |
| 2. Repeatable | Project management | Requirements management Software project planning Software project tracking Software quality assurance Software configuration management Software subcontract management | X X | X X X X |



| CMM Level | Focus | Key Process Area | PSP | TSP |
|----------------------|---------------------------------------|--|-----|-----|
| 5. Optimizing | Continuous process improvement | Defect prevention | X | X |
| | | Technology change management | X | X |
| | | Process change management | X | X |
| 4. Managed | Product & process quality | Quantitative process management | X | X |
| | | Software quality management | X | X |
| 3. Defined | Engineering process | Organization process focus | X | X |
| | | Organization process definition | X | X |
| | | Training program | | |
| | | Integrated software management | X | X |
| | | Software product engineering | X | X |
| | | Inter-group coordination | | X |
| | | Peer reviews | X | X |
| 2. Repeatable | Project management | Requirements management | | X |
| | | Software project planning | X | X |
| | | Software project tracking | X | X |
| | | Software quality assurance | | X |
| | | Software configuration management | | X |
| | | Software subcontract management | | |

**Relating
PSP, TSP &
CMM
(Humphrey)**

