TDDD04: Test plans and software defect taxonomies

Lena Buffoni lena.buffoni@liu.se

LINKÖPING UNIVERSITY

Lecture plan

- Test planning (ISO/IEC/IEEE 29119)
- Scripted vs Exploratory testing
- Defect and risk taxonomies (ISO/IEC/IEEE 1044)
- Test Driven Development

Scripted testing

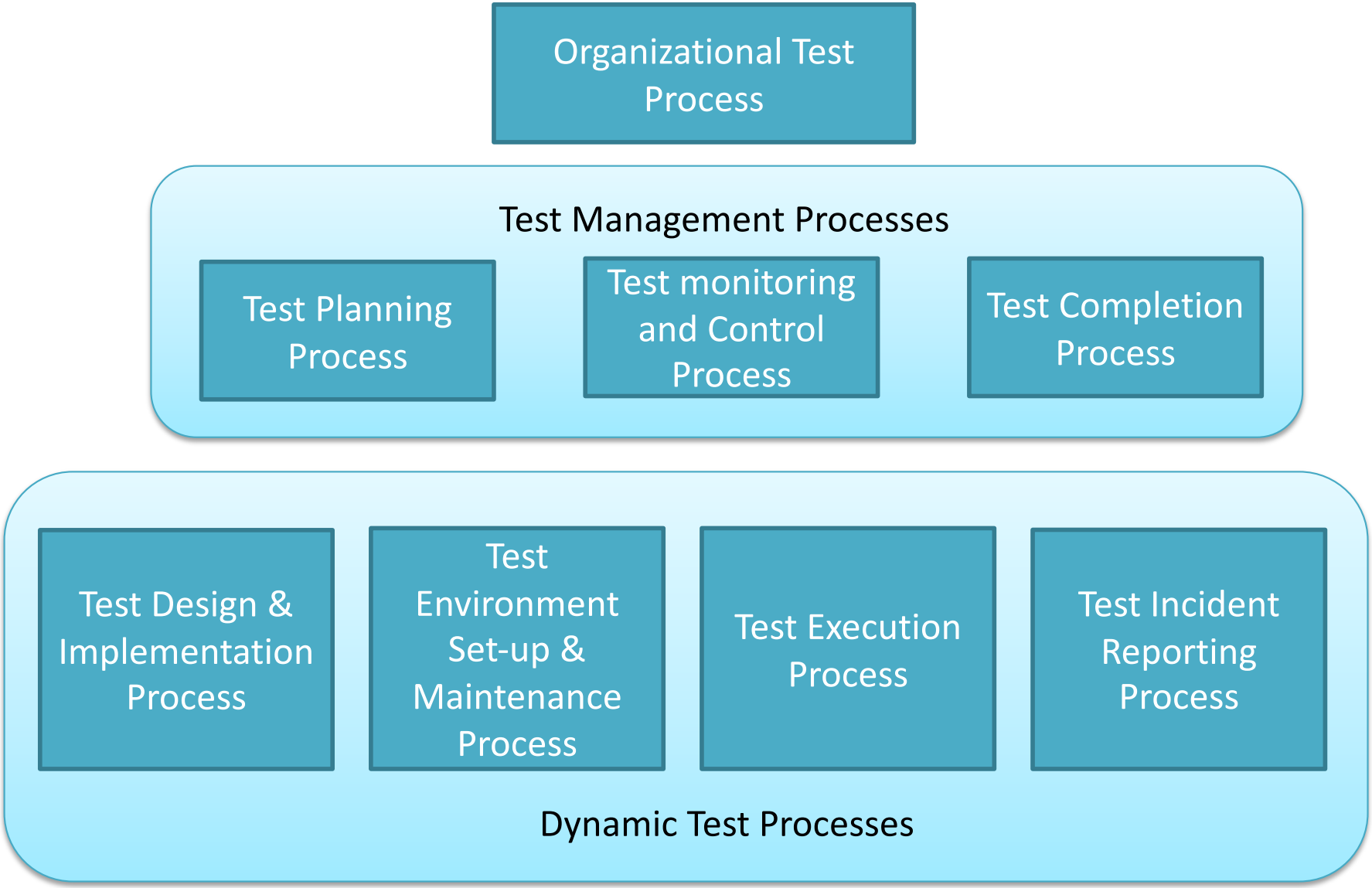
Testing performed based on a documented test script created from requirements, design and code

- Allow division of labor
- Tests can be easily understood and repeated
- Easier to automate tests
- Coverage can be easily defined and measured

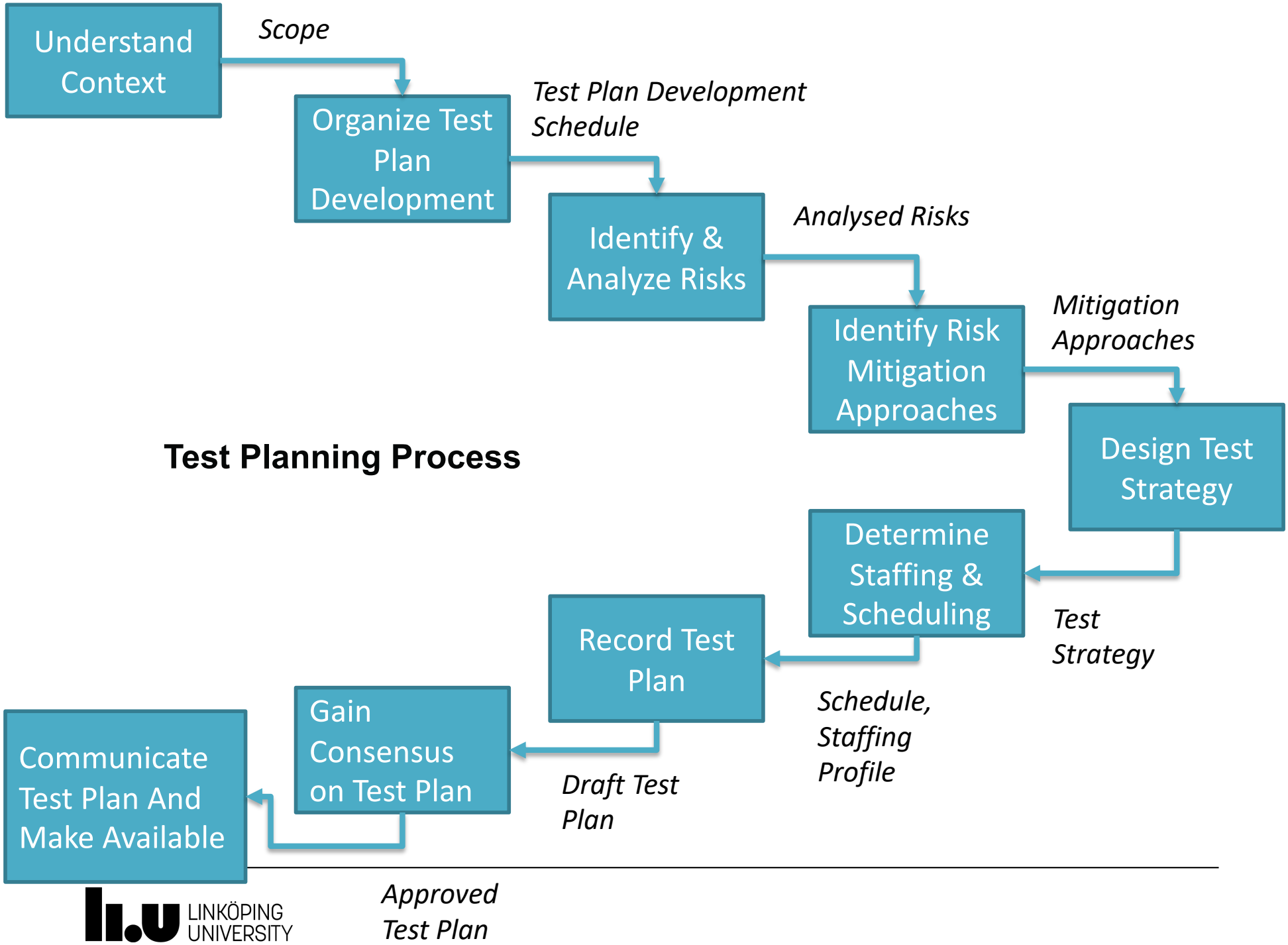
Overall/level test plan

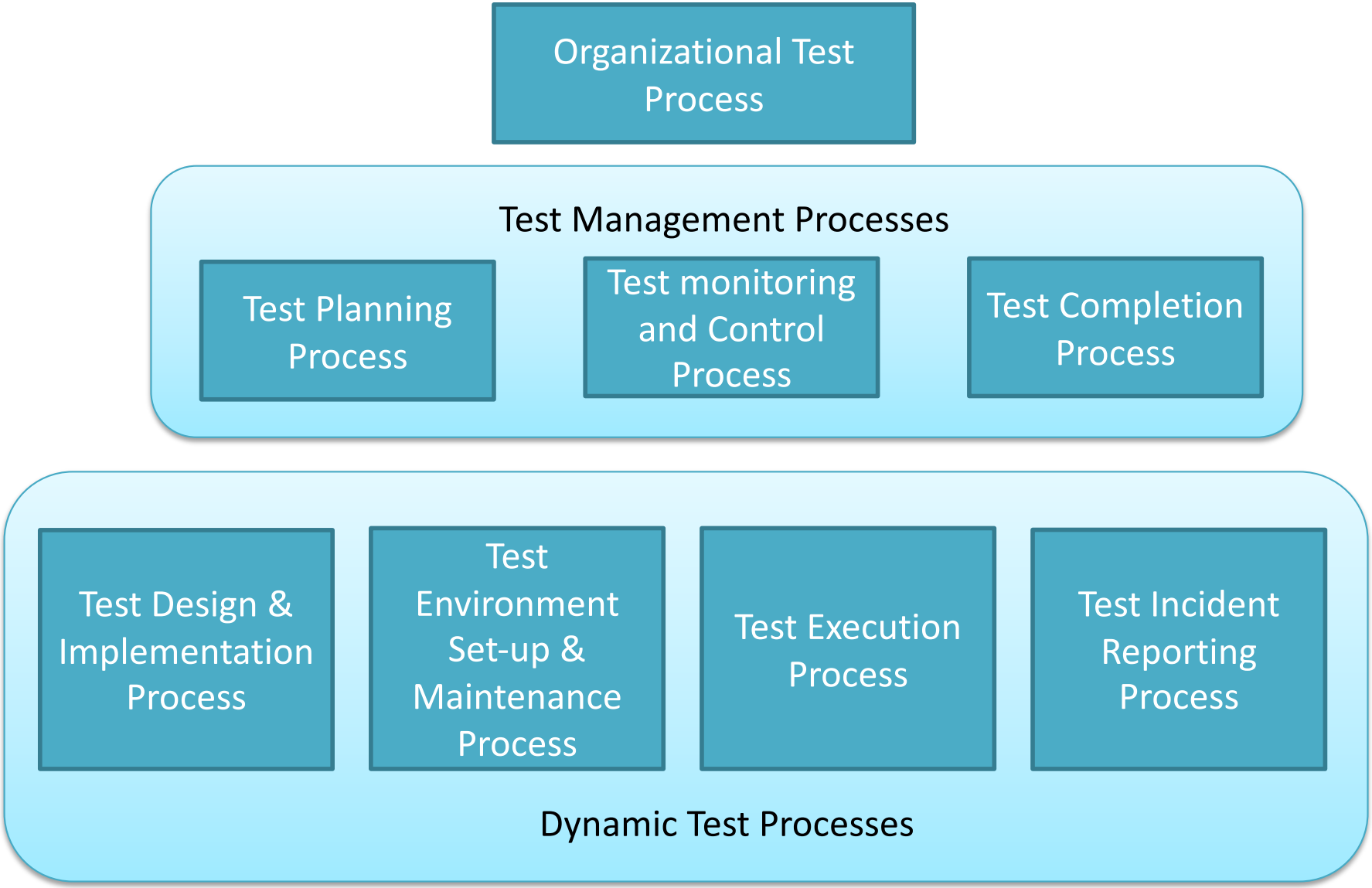
- Overall test plan: Global goals for the project

- Level test plan: goals at a specific testing level (unit, module, …)

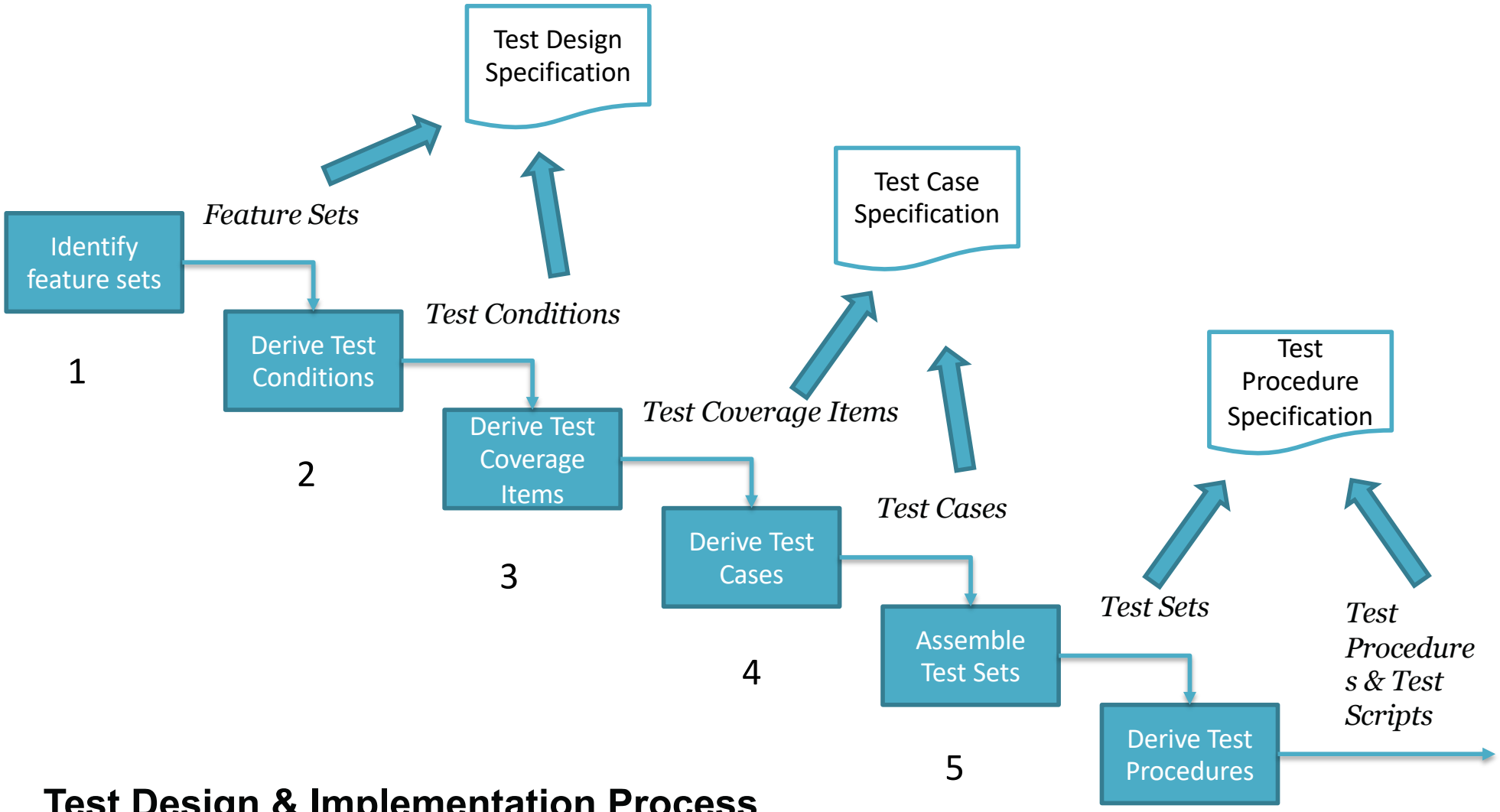


ISO/IEC/IEEE, "ISO/IEC/IEEE international standard for software and systems engineering – software testing part 2: Test processes," ISO/IEC/IEEE 29119-2:2013(E), pp. 1–68, Sept 2013.


Understand Context
Scope
Organize Test Plan Development
Test Plan Development Schedule
Identify & Analyze Risks
Analysed Risks
Identify Risk Mitigation Approaches
Mitigation Approaches
Design Test Strategy
Test Planning Process
Determine Staffing & Scheduling
Test Strategy
Record Test Plan
Schedule, Staffing Profile
Gain Consensus on Test Plan
Draft Test Plan
Communicate Test Plan And Make Available
Approved Test Plan



ISO/IEC/IEEE, "ISO/IEC/IEEE international standard for software and systems engineering – software testing part 2: Test processes," ISO/IEC/IEEE 29119-2:2013(E), pp. 1–68, Sept 2013.



Test Design & Implementation Process

Example

"The system shall accept insurance applicants over the age of 18 and under the age of 80 years on the day of application based on their input age in whole years; all others shall be rejected.

Accepted applicants of 70 and over shall receive a warning that in the event of a claim they shall pay an excess of $1000."

Derive test conditions

- Completion criterion?
  - "The Test Completion Criterion is that 100 % normal case coverage is achieved and all test cases must result in a "pass" status on execution."
- Valid input?
  Invalid input?

What if we note the following: 40 <= Age <= 55 results in a discount message (unspecified in the description). How do we handle that?

Derive test coverage items

- Use-cases to cover

Derive test cases

- Select representatives from each class to achieve 100% use-case coverage

Assemble test sets

- What can be automated?
  What must be manually tested?

Derive test procedures

- Ordering of test cases based on exposure/dependencies
- Traceability

Limitations

- Very dependent on the quality of system requirements
- Inflexible, if some unusual behavior is detected, it will not be pursued
- Focus can shift to documentation

Exploratory testing

IEEE definition : “*a type of unscripted experience-based testing in which the tester spontaneously designs and executes tests based on the tester's existing relevant knowledge, prior exploration of the test item (including the results of previous tests), and heuristic "rules of thumb" regarding common software behaviors and types of failure*”

Example

- Create a mental model
- Define one or more tests to disprove the model
- Execute the tests and observe the outcome
- Evaluate the outcome against the model
- Repeat

Definitions

Schedule: an uninterrupted block of time devoted to testing (1-2 hours)

Charter: a guide defined before the testing session covering

- what to test
- available documentation
- test tactics
- risks involved

Useful when

- the next test case cannot be determined in advance and needs to be chosen based on previous experience
- it is necessary to provide rapid feedback on a products quality
- a defect is detected, to explore the scope and variations of the defect

Exploratory testing =/= random testing!

Limitations

- Does not prevent defects
- Incompatible with agile development
- Does not detect omission errors
- Can focus excessively on a particular area
- Hard to know when to stop testing

Fault classification

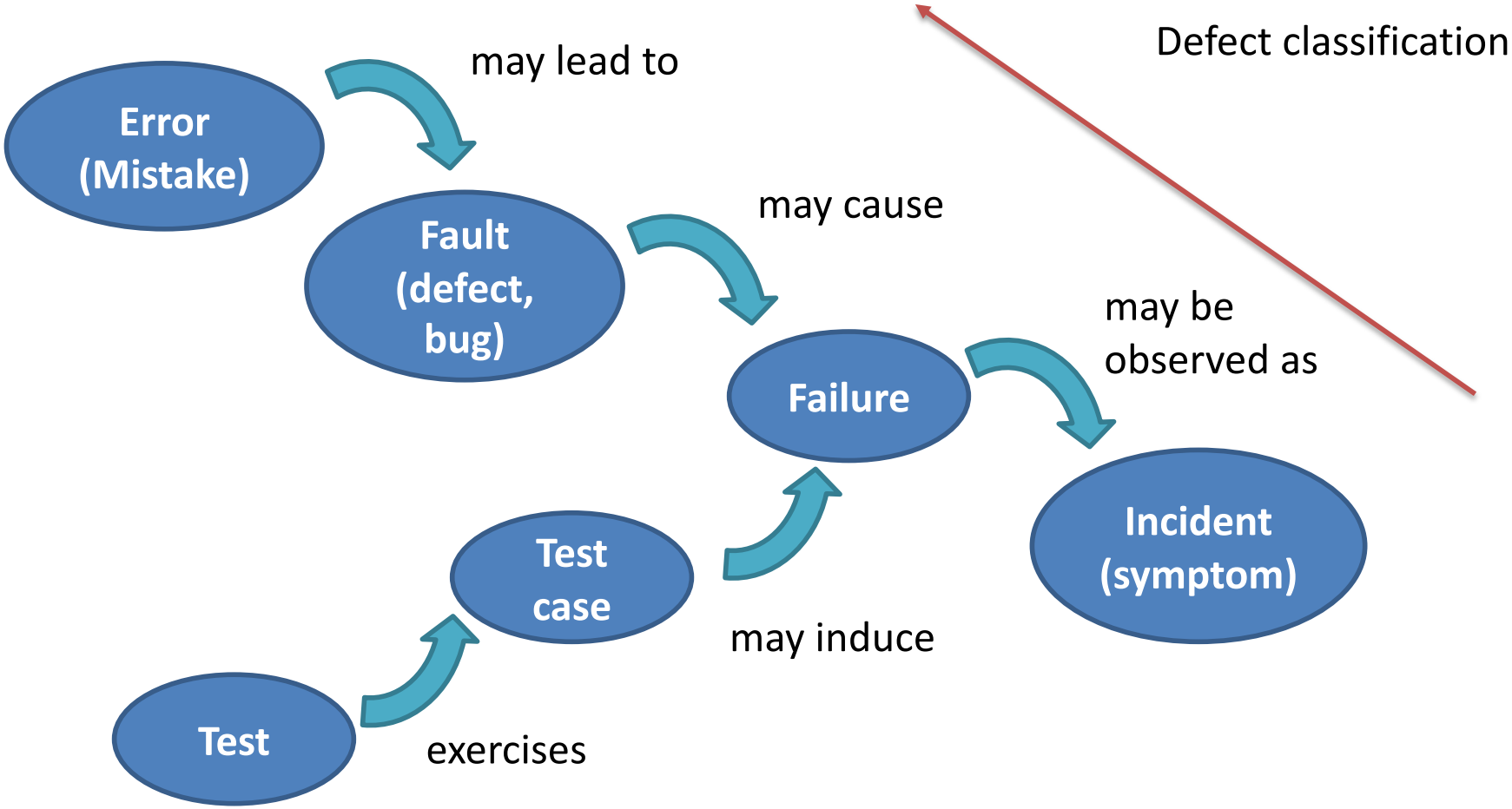
Software defect taxonomies: what kind is it?

- Useful to guide test planning (e.g. have we covered all kinds of faults)
- Beizer (1984): Four-level classification
- Kaner et al. (1999): 400 different classifications

Severity classification: how bad is it?

- Important to **define** what each level means
- **Severity does not equal priority**
- Beizer (1984): mild, moderate, annoying, disturbing, serious, very serious, extreme, intolerable, catastrophic, infectious.
- ITIL (one possibility): severity 1, severity 2

Failure classification: Reminder



Fault vs defect in ISO/IEC/IEEE 1044

- A fault is an executed defect
- A defect can be found before it is executed
  (eg: by inspection)

Multiple failures can be caused by the same defect!

Taxonomies

A taxonomy is a classification of things into ordered groups or categories that indicate natural hierarchical relationships.

- Guide the test case design
- Understand the defects better
- Help determine coverage that test-cases are providing
- Can be created at different levels

Software level taxonomy: IEEE Standard Classification for Software Anomalies

Defect attributes

Attribute	Definition
Defect ID	Unique identifier for the defect.
Description	Description of what is missing, wrong, or unnecessary.
Status	Current state within defect report life cycle.
Asset	The software asset (product, component, module, etc.) containing the defect.
Artifact	The specific software work product containing the defect.
Version detected	Identification of the software version in which the defect was detected.
Version corrected	Identification of the software version in which the defect was corrected.
Priority	Ranking for processing assigned by the organization responsible for the evaluation, resolution, and closure of the defect relative to other reported defects.
Severity	The highest failure impact that the defect could (or did) cause, as determined by (from the perspective of) the organization responsible for software engineering.
Probability	Probability of recurring failure caused by this defect.
Effect	The class of requirement that is impacted by a failure caused by a defect.

Defect attributes - continued

Attribute	Definition
Type	A categorization based on the class of code within which the defect is found or the work product within which the defect is found.
Mode	A categorization based on whether the defect is due to incorrect implementation or representation, the addition of something that is not needed, or an omission.
Insertion activity	The activity during which the defect was injected/inserted (i.e., during which the artifact containing the defect originated).
Detection activity	The activity during which the defect was detected (i.e., inspection or testing).
Failure reference(s)	Identifier of the failure(s) caused by the defect.
Change reference	Identifier of the corrective change request initiated to correct the defect.
Disposition	Final disposition of defect report upon closure.

Effect: Examples

Effect	Functionality	Actual or potential cause of failure to correctly perform a required function (or implementation of a function that is not required), including any defect affecting data integrity.
Effect	Usability	Actual or potential cause of failure to meet usability (ease of use) requirements.
Effect	Security	Actual or potential cause of failure to meet security requirements, such as those for authentication, authorization, privacy/confidentiality, accountability (e.g., audit trail or event logging), and so on.
Effect	Performance	Actual or potential cause of failure to meet performance requirements (e.g., capacity, computational accuracy, response time, throughput, or availability).
Effect	Serviceability	Actual or potential cause of failure to meet requirements for reliability, maintainability, or supportability (e.g., complex design, undocumented code, ambiguous or incomplete error logging, etc.).
Effect	Other	Would/does not cause any of the above effects.

Type: Examples

Type	Data	Defect in data definition, initialization, mapping, access, or use, as found in a model, specification, or implementation.
		Examples: Variable not assigned initial value or flag not set Incorrect data type or column size Incorrect variable name used Valid range undefined Incorrect relationship cardinality in data model Missing or incorrect value in pick list
Type	Interface	Definition Defect in specification or implementation of an interface (e.g., between user and machine, between two internal software modules, between software module and database, between internal and external software components, between software and hardware, etc.).
		Examples: Incorrect module interface design or implementation Incorrect report layout (design or implementation) Incorrect or insufficient parameters passed Cryptic or unfamiliar label or message in user interface Incomplete or incorrect message sent or displayed Missing required field on data entry screen
Type	Logic	Defect in decision logic, branching, sequencing, or computational algorithm, as found in natural language specifications or in implementation language.
		Examples: Missing else clause Incorrect sequencing of operations Incorrect operator or operand in expression Missing logic to test for or respond to an error condition (e.g., return code, end of file, null value, etc.) Input value not compared with valid range Missing system response in sequence diagram

Failure classification

Failure ID	Unique identifier for the failure.
Status	Current state within failure report life cycle.
Title	Brief description of the failure for summary reporting purposes.
Description	Full description of the anomalous behavior and the conditions under which it occurred, including the sequence of events and/or user actions that preceded the failure.
Environment	Identification of the operating environment in which the failure was observed.
Configuration	Configuration details including relevant product and version identifiers.
Severity	As determined by (from the perspective of) the organization responsible for software engineering.
Analysis	Final results of causal analysis on conclusion of failure investigation.
Disposition	Final disposition of the failure report.

Failure classification - continued

Observed by	Person who observed the failure (and from whom additional detail can be obtained).
Opened by	Person who opened (submitted) the failure report.
Assigned to	Person or organization assigned to investigate the cause of the failure.
Closed by	Person who closed the failure report.
Date observed	Date/time the failure was observed.
Date opened	Date/time the failure report is opened (submitted).
Date closed	Date/time the failure report is closed and the final disposition is assigned.
Test reference	Identification of the specific test being conducted (if any) when the failure occurred.
Incident reference	Identification of the associated incident if the failure report was precipitated by a service desk or help desk call/contact.
Defect reference	Identification of the defect asserted to be the cause of the failure.
Failure reference	Identification of a related failure report.

Problem 1

Sue calls service desk and reports she cannot log in to timesheet system because the password field is missing from the login screen.

In this example, Sue has a problem in that she cannot log in, caused by a failure wherein the password field did not appear on the login screen, which was in turn caused by a defect inserted during coding of the Login.asp artifact.

Entity	Attribute	Problem 1
Failure	Failure ID	F080001
Failure	Status	Open
Failure	Title	Failure: missing password field
Failure	Description	The password field is missing from the login screen
Failure	Environment	Chicago-websrvr23
Failure	Configuration	TimeSheet v6.4
Failure	Severity	Critical
Failure	Analysis	Code error
Failure	Disposition	
Failure	Observed by	Sue
Failure	Opened by	Williams
Failure	Assigned to	
Failure	Closed by	
Failure	Date observed	April 1, 2008
Failure	Date opened	April 1, 2008
Failure	Date closed	
Failure	Test reference	N/A
Failure	Incident reference	S080002
Failure	Defect reference	D080234
Defect	Defect ID	D080234
Defect	Description	Password field not correctly implemented in code

Entity	Attribute	Problem 1
Defect	Status	Open
Defect	Asset	TS-srvr
Defect	Artifact	Login.asp
Defect	Version detected	V6.4
Defect	Version corrected	
Defect	Priority	
Defect	Severity	Critical
Defect	Probability	High
Defect	Effect	Functionality
Defect	Type	Interface
Defect	Mode	Missing
Defect	Insertion activity	Coding
Defect	Detection activity	Production
Defect	Failure reference	F080001
Defect	Change reference	C080049
Defect	Disposition	

Problem 2

Joe calls service desk and reports he cannot log in to timesheet system because the password field is missing from the login screen.

This example is similar to Problem 1 and serves to illustrate that two distinct failures (events) can be caused by a single defect (condition).

Entity	Attribute	
	ID	
	Title	
	Desctiption	
	Severity	
	Defect ID	
	Defect description	
	Probability	
	Effect	
	Type	
	Mode	

Problem 3

During customer qualification testing, Sam observed that the color of the font does not match the requirements document section 4.2.1.3.

This example illustrates the difference between the failure (appearance of incorrect color on screen) and the defect that caused it (incorrect data value assigned to a constant in the code).

Entity	Attribute	
	ID	
	Title	
	Desctiption	
	Severity	
	Defect ID	
	Defect description	
	Probability	
	Effect	
	Type	
	Mode	

Problem 4

During a peer review for software requirements for a new financial management system, Alice discovers that values are in the requirements as thousands of dollars instead of as millions of dollars.

This example illustrates classification of a defect detected directly, prior to any failure occurring.

Entity	Attribute	
	ID	
	Title	
	Desctiption	
	Severity	
	Defect ID	
	Defect description	
	Probability	
	Effect	
	Type	
	Mode	

Problem 5

Company A's battery ran out of power because there was no low-power warning. The design of a security system monitoring system did not include a warning for low battery power, despite the fact that this feature was specified in the requirements.

In this example, the defect was not detected until a failure occurred in a production environment.

Entity	Attribute	
	ID	
	Title	
	Desctiption	
	Severity	
	Defect ID	
	Defect description	
	Probability	
	Effect	
	Type	
	Mode	

Project level taxonomy: SEI Risk Identification Taxonomy

SEI Risk Identification Taxonomy

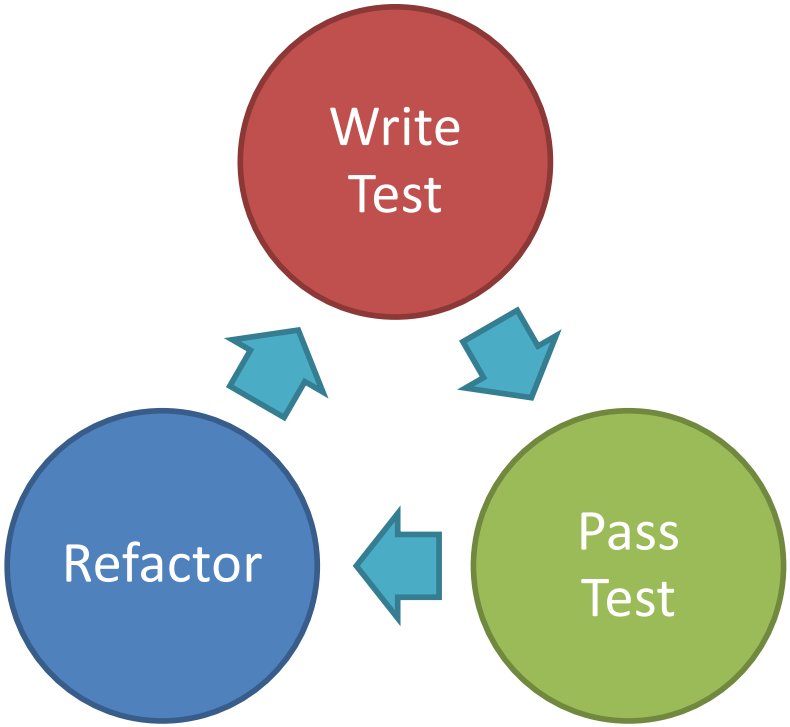
- Developed as a checklist to ensure all potential risks are covered

3 Main categories:

  - Product engineering
  - Development environment
  - Program constraints

Class	Element	Attribute
Product Engineering	Requirements	Stability
		Completeness
		Clarity
		Validity
		Feasibility
		Precedent
		Scale
	Design	Functionality
		Difficulty

Test-driven development



- Guided by a sequence of user stories from the customer/user
- Needs test framework support (eg: Junit)

NextDate:

User Stories

TEST

Input	Expected Output
Source Code	OK
15	Day = 15
15, 11	Day = 15 Month = 11

Code

1: the program compiles

```
Program NextDate
End NextDate
```

2: a day can be input and displayed

```
Program NextDate
        input int thisDay;
        print (“day =“ + thisDay);
End NextDate
```

2: a month can be input and displayed

```
Program NextDate
        input int thisDay;
        input int thisMonth;
        print (“day =“ + thisDay);
        print (“month =“ + thisMonth) ;
End NextDate
```

Pros and cons

+ working code
+ regression testing
+ easy fault isolation
+ test documented code

- code needs to be refactored
- can fail to detect deeper faults

Evaluating a test suite

- Number of tests?
- Number of passed tests?
- Cost/effort spent?
- Number of defects found?

Defect Detection Percentage = defects found by testing / total known defects

Thank you!

Questions?

LINKÖPING UNIVERSITY