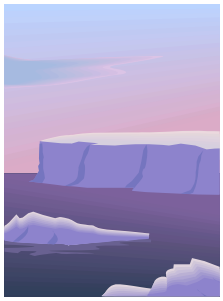
DEFECT-FLOW MODELS

PART 2: Defect Classification



Dr. Adam Trendowicz
adam.trendowicz@iese.fraunhofer.de

Martin Kowalczyk
martin.kowalczyk@iese.fraunhofer.de

Michael Kläs
michael.klaes@iese.fraunhofer.de

Ove Armbrust
ove.armbrust@iese.fraunhofer.de

Fraunhofer IESE

Fraunhofer IESE

Part 2: Defect Classification

- **Defect Classification Theory**
  - Terminology
  - Structures for Classification
- **Prominent Approaches**
  - IEEE Standard Classification Scheme
  - Hewlett-Packard (HP) Classification Scheme
  - Orthogonal Defect Classification (ODC)
- **Summary**

Basic idea of defect classification

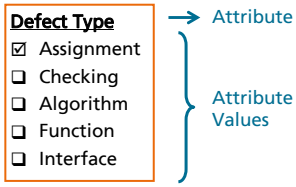
Basic Idea

- Capture semantics of a defect by classifying it according to a few attributes
  - e.g., what had to be fixed, how was the defect detected, etc.
- Analyze the actual distribution of attributes for a larger number of defects
- Analyze data together with developers to interpret data in order to
  - devise controlling actions or
  - make improvement suggestions

Defect classification and defect classification schemes

Defect Classification

- Measurement of one or several attributes (e.g., defect type) of a defect on defined scales (typically nominal or ordinal scale)



Defect Classification Scheme

- A set of attributes and corresponding attribute values for defect classification

Defect Qualifier = {missing, incorrect, extraneous}
Defect Type = {assignment, checking, algorithm, function, …}
Activity = {code inspection, unit test, integration test, …}

Example: classification data

ID	1	2	3
open date	01.01.2009	03.01.2009	11.1.2009
close date	02.01.2009	10.01.2009	13.01.2009
activity	inspection	unit test	system test
trigger	conform.	simple	SW config
impact	capability	usability	reliability
target	code	code	code
type	assign	checking	function
qualifier	misssing	missing	incorrect
source	in-house	in-house	outsourced
age	new	new	refixed

Part 2: Defect Classification

- **Defect Classification Theory**
  - Terminology
  - Structures for Classification
    - Attribute categorization scheme
    - Hierarchical attribute values
    - Hybrid scheme
- **Prominent Approaches**
  - IEEE Standard Classification Scheme
  - Hewlett-Packard (HP) Classification Scheme
  - Orthogonal Defect Classification (ODC)
- **Summary**

Structural alternatives for classification of attributes (1/3)

Attribute categorization scheme

- For each attribute, there is one set of attribute values
- Possible attribute values for attribute A are independent from values of attribute B
- Like table in relational database
- Example: ODC

Impact		Defect Type		Qualifier		Target	
Installability		Assignment/ Initialization		Missing	X	Code	X
Integrity/ Security		Checking		Incorrect		GUI	
Performance	X	Algorithm/ Method	X	Extraneous			
Maintenance		Function/ Class/ Object					
Serviceability		Timing/ Serialization					
Migration		Interface/ O-O Messages					
Documentation		Relationship					

Structural alternatives for classification of attributes (2/3)

Hierarchical attribute values

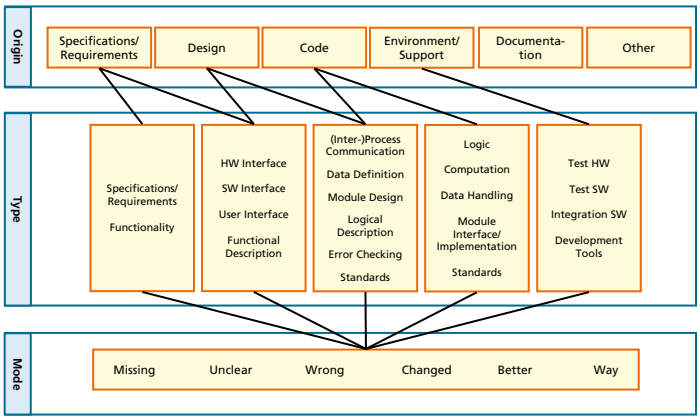
- Attribute values on level x can be refined on level x+1 (hierarchical dependency)
- Example: IEEE 1044

Attribute: Type - What type of defect/enhancement at the code level?			
Logic Problem	Forgotten cases or steps		
	Duplicate logic		
	Extreme conditions neglected		
	Unnecessary function		
	Misinterpretation		
	Missing condition test		
	Check wrong variable		
	Iterating loop incorrectly		
Computation Problem	Equation insufficient or incorrect	Missing computation	
		Operand incorrect	
		Operator incorrect	
		Parentheses used incorrectly	X
	Precision loss	Rounding or truncation fault	
		Mixed modes	
	Sign convention fault		

Structural alternatives for classification of Attributes (3/3)

Hybrid scheme

- Set of attribute values for attribute y depends on value of attribute x
- Example: HP scheme (origin determines type)



Part 2: Defect Classification

- **Defect Classification Theory**
  - Terminology
  - Structures for Classification
- **Prominent Approaches**
  - IEEE Standard Classification Scheme
  - Hewlett-Packard (HP) Classification Scheme
  - Orthogonal Defect Classification (ODC)
- **Summary**

Most important defect classification schemes

IEEE Standard Classification Scheme (for Software Anomalies)

- Purpose: support for defect tracking

Hewlett-Packard Classification Scheme

- Purpose: process improvement through reduction of defects over time

Orthogonal Defect Classification Scheme (ODC)

- Purpose: enable control of verification and validation activities, enable process improvement
- Developed at IBM, usage reported from IBM, Motorola, Cisco, …
- Many experience reports → quasi-standard

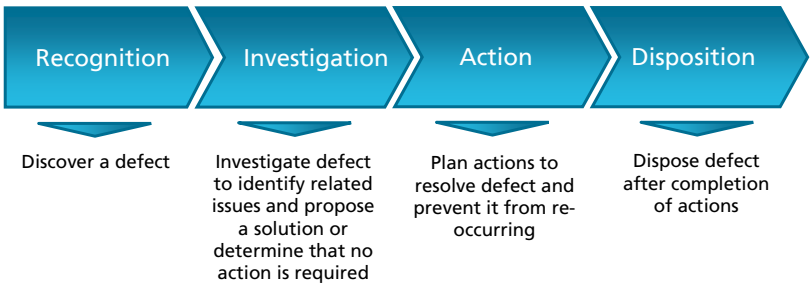
Introduction to these three schemes due to their high relevance

Part 2: Defect Classification

- **Defect Classification Theory**
  - Terminology
  - Structures for Classification
- **Prominent Approaches**
  - IEEE Standard Classification Scheme
  - Hewlett-Packard (HP) Classification Scheme
  - Orthogonal Defect Classification (ODC)
- **Summary**

IEEE standard classification scheme (standard 1044) [1]

The Classification Process Steps



Example: IEEE standard scheme

Recognition

Investigation

Action

Disposition

Attribute Name	Attribute Meaning
Project Activity	What were you doing when the defect occurred?
Project Phase	In which life-cycle phase is the product?
Suspected Cause	What do you think might be the cause?
Repeatability	Could you make the defect appear more than once?
Symptom	How did the defect manifest itself?
Product Status	What is the usability of the product with no changes?
Actual Cause	What caused the anomaly to occur?
Source	Where (part of the system and its documentation) was the defect's origin?
Type	What type of defect/enhancement at the code level?
Resolution	What to do to prevent the defect from happening again?
Corrective Action	What action to take to resolve the defect?
Severity	How bad was the defect in more objective engineering terms?
Priority	Rank the importance of resolving the defect?
Customer Value	How important is a fix to the customer?
Mission Safety	How bad was the defect wrt. project objectives or human well-being?
Project Schedule	Relative effect on the project schedule to fix?
Project Cost	Relative effect on the project budget to fix?
Project Risk	Risk associated with implementing a fix?
Project Qual./ Rel.	Impact on the product quality or reliability to make a fix?
Societal	Impact on society of implementing the fix?
Disposition	What actually happened to close the anomaly?

Hierarchical structure of IEEE standard scheme

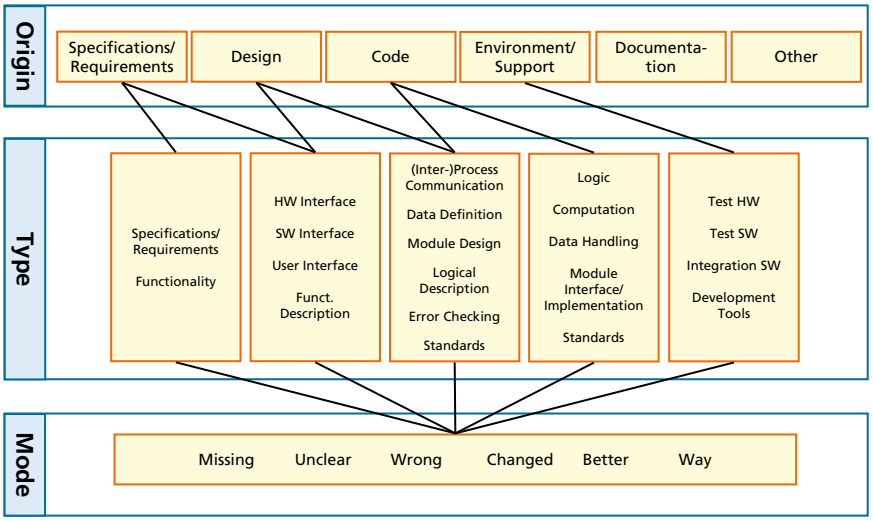
Type - What type of defect/enhancement at the code level?		
Logic Problem	Forgotten cases or steps	
	Duplicate logic	
	Extreme conditions neglected	
	Unnecessary function	
	Misinterpretation	
	Missing condition test	
	Check wrong variable	
	Iterating loop incorrectly	
Computation Problem	Equation insufficient or incorrect	Missing computation
		Operand incorrect
		Operator incorrect
		Parentheses used incorrectly
	Precision loss	Rounding or truncation fault
		Mixed modes
	Sign convention fault	

IEEE scheme is very exhaustive and thus application is quite complex!

Part 2: Defect Classification

- **Defect Classification Theory**
  - Terminology
  - Structures for Classification
- **Prominent Approaches**
  - IEEE Standard Classification Scheme
  - Hewlett-Packard (HP) Classification Scheme
  - Orthogonal Defect Classification (ODC)
- **Summary**

Hewlett-Packard (HP) scheme [2]



Attribute value: Origin

Origin - Where in the lifecycle phase does the defect have its origin?

Origin	Specifications / Requirements	**A mistake in the definition of the customer/target needs for a system or system component.** Such mistakes can be in functional requirements, performance requirements, interface requirements, design requirements, test requirements, development standards, etc.
	Design	**A mistake in the design of a system or system component**. Such mistakes can be in algorithms, control logic, data structures, database access, interface descriptions, etc.
	Code	**A mistake in the implementation** of a computer program. Such mistakes can be in product or test code, JCL, build files, etc.
	Environmental Support	**Defects that arise as a result of the system development and/or testing environment.** Such mistakes can be in the build/configuration process, the development/integration tools, the testing environment, etc.
	Documentation	**A mistake in any non-code material** delivered to a customer. Such mistakes can be in user manuals, installation instructions, data sheets, product demos, etc.
	Other	This classification should be used sparingly and when it is used, the defect should be very carefully and extensively described in associated documentation.

Attribute value: Type (specifications, requirements) (1/4)

Type (specifications, requirements) – What is the type of the defect?

Type	**Requirements/ Specifications**	The specification does not adequately describe the needs of the target users. Includes the effects of product strategy redirection and cases where functionality is increased to add market value.
	Functionality	Incorrect or incompatible product features.
	Hardware, Software, and User Interface	Incorrect specification of how the product will interact with its environment and/or users.
	Functional Description	Incorrect description of what the product does. Generally discovered during requirements or design inspection.

Attribute value: Type (design) (2/4)

Type (design) – What is the type of the defect?

Type	**Hardware, Software, and User Interface**	Problems with incorrect design of how the product will interact with its environment and/or users. E.g., incorrect use of libraries; design does not implement requirements; device capabilities overlooked or unused; design does not meet usability goals.
	Functional Description	Design does not effectively convey what the module/product should do. Generally discovered during design inspection or coding.
	Process or Inter-Process Communications	Incorrect interfaces and communications between processes within the product.
	Data Definition	Incorrect design of the data structures to be used in the module/product.
	Module Design	Problems with the (logic) flow and execution within processes.
	Logic Description	Design is incorrect in conveying the indented algorithm or logic flow. Generally, a defect found during design inspection or coding.
	Error Checking	Incorrect error condition checking.
	Standards	Design does not adhere to locally accepted design standards.

Attribute value: Type (code) (3/4)

Type (code) – What is the type of the defect?

Type	**Logic**	Forgotten cases or steps, duplicate logic, extreme conditions neglected, unnecessary functions, misinterpretation errors.
	Computation Problems	Equation insufficient or incorrect, precision loss, sign convention fault.
	Data Handling Problems	Initialized data incorrectly, accessed or stored data incorrectly, scaling of units of data incorrect, dimensioned data incorrectly, scope of data incorrect.
	Module Interface/ Implementation	Problems related to the calling of, parameter definition of, and termination of sub-processes. E.g., incorrect number of, or order of, subroutine arguments, ambiguous termination value for a function, or data types incorrect.
	Standards	Code does not adhere to locally accepted coding standard.

Attribute value: Type (environment, support) (4/4)

Type (environment, support) – What is the type of the defect?

Type	**Test Software**	Problems in software used to test the product's software capabilities. E.g., another application program, operating system, simulation software.
	Test Hardware	Problems with the HW used to run the test software. NOT the HW on which the product runs.
	Development Tools	Problems that are a result of development tools not behaving according to specifications or in a predictable manner.
	Integration Tools	Problems that result from integration software/ tools or processes.

Attribute value: Mode

Mode – Why did the defect occur?

Mode	**Missing**	Information was left out of a work product.
	Unclear	Information was misleading, ambiguous, or hard to understand.
	Wrong	The information in the work product was not correct.
	Changed	Changes in a work product caused changes to other work products.
	Better Way	There was a better way to do a work product, usually for better efficiency, performance, readability, maintainability, and supportability.

HP approach to causal analysis (1/2)

Step 0

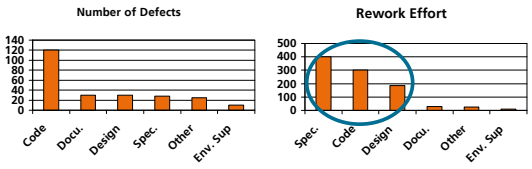
Data Collection and Validation

Rework-Effort: 66
Origin: Design **Type:** Data definition
Mode: Wrong **Module:** Disc_io
How could defect have been prevented/found earlier:
Design walkthrough, more complete test overage

Step 1

Create Pareto Chart for **Origin**

Create Pareto Chart for **Rework Effort** (per Origin)



Step 2

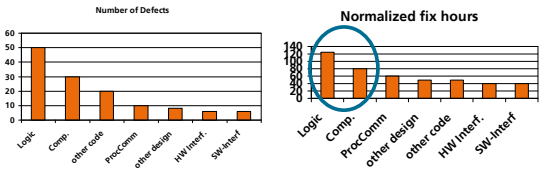
Compute **Average Rework Effort per Origin**

Average Fix hours
Spec: 14.25
Design: 6.25
Code: 2.5
Docu/Env/Other: 1

HP approach to causal analysis (2/2)

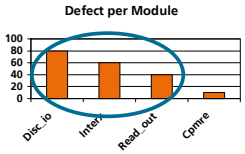
Step 3

For top 2-3 origins: Create Pareto Chart for **Defect Type** (multiplied by average fix times)



Step 4

Create Pareto Chart for Modules



Step 5

Review Defect Reports for the largest totals in Step 3 and 4 and summarize prevention proposals

How could defect have been prevented/found earlier:
Design walkthrough, more complete test coverage,

How could defect have been prevented/found earlier:
More timely data dictionary updates,

Focusing improvement actions based on Pareto chart

Select the largest column

- Frequently chosen, particularly appropriate for organizations with mature process control

Select easiest column

- Easy potential for solution or most confidence in solution
- Achieve early success for further motivation

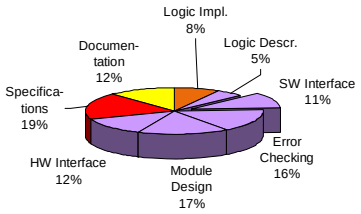
Select largest normalized column

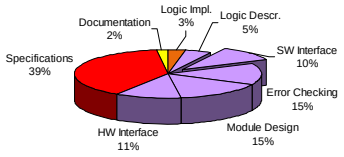
- Take into account the find-and-fix effort (where can we save most)

Select a combination of columns

- Select solution that can address several wedges
  - e.g., inspections for logic AND computation

Example of classification usage (1/3)

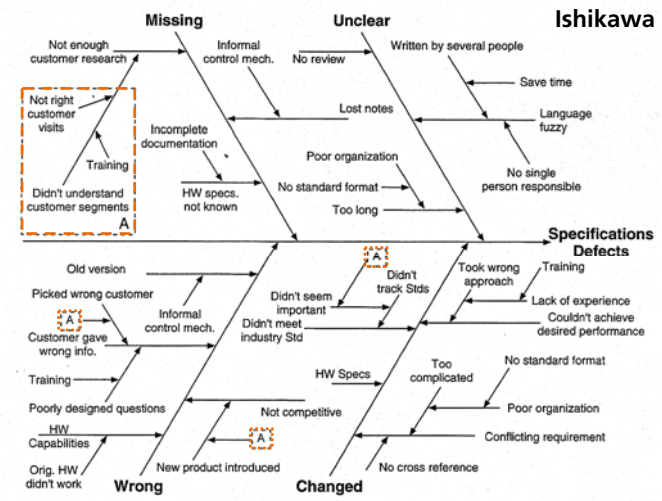




Detailed analysis based on classification

- Number of defects
  - No clear focus for action can be derived based solely on number of defects
  - Include effort for correction
- Number of defects normalized by correction effort
  - Specification defects offer most cost-saving potential
  - Further focus on specifications

Example of classification usage (2/3)



Example of classification usage (3/3)

Actions derived from analysis results

- Root cause analysis
  - Unclear understanding of customer segments
  - Lack of clearly assigned responsibilities
- Improvement actions
  - Marketing department sets up customer visits to learn more about customer needs
  - Assignment of configuration management responsibility to one person
  - Introduction of standard tool for version control

Contents: Defect Classification

- **Defect Classification Theory**
  - Terminology
  - Structures for Classification
- **Prominent Approaches**
  - IEEE Standard Classification Scheme
  - Hewlett-Packard (HP) Classification Scheme
  - Orthogonal Defect Classification (ODC)
- **Summary**

ODC scheme for design and code [3], [4]

Feedback to **Development** Process

- Defect Type
  - What had to be fixed?
- Defect Qualifier
  - Missing, extraneous or wrong?

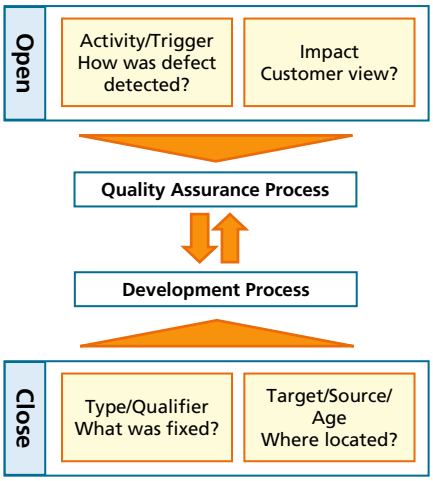
Feedback to **Quality Assurance** Activities

- Activity
  - When did you detect the defect?
- Trigger
  - How did you detect the defect?
- Impact
  - What would customers have noticed if defect had escaped into the field?

Feedback to **Product**

- Age
  - What is the history of the target (artifact)?
- Source
  - Who developed the target (artifact)?
- Target
  - What high-level entity was fixed?

Rationale behind ODC classification scheme



- **Opener Section** - Attributes can be classified when defect is detected
- **Closer Section** - Attributes can be classified when defect has been corrected
- Designed to capture as much information in few attributes and to perform useful analyses
- Applicable to all phases
- Consistent across products and projects
- Small number of orthogonal attribute values
- Time to classify: 45sec-2min per defect
- **Note:** Classical ODC attributes focus on late development phases (design, coding, and testing)

ODC attributes in detail

Attribute	Attribute Meaning	Attribute Values
Defect Type	What had to be fixed?	assignment, checking, algorithm, function, timing, interface, relationship
Defect Qualifier	How can the defect be qualified?	missing, incorrect, extraneous
Activity	When did you detect the defect?	design inspection, code inspection, unit test, integration test, system test
Trigger	How did you detect the defect?	Inspection Trigger: design conformance, logic/flow, lateral compatibility, backward compatibility, language dependency, concurrency, side effects, rare situation Unit Test Trigger System Test Trigger
Impact	What would customer have noticed if defect had escaped into field?	installability, serviceability, standard, integrity/security, migration, reliability, performance, documentation, requirements, maintenance, usability, accessibility, capability
Target	What high-level entity was fixed?	requirements, **design, code**, build/package/merge, information, user interface
Source	Who developed the target?	in-house, library, outsourced, ported
Age	What is the history of the target?	base, new, rewritten, re-fixed

Definition of the attribute "Defect Type" (1/2)

Defect Type - What had to be fixed?

Defect Type - Attribute Values		
	Assignment/ Initialization	**Value(s) assigned incorrectly or not assigned at all; but note that a fix involving multiple assignment corrections may be of the type Algorithm.** Examples: 1) Internal variable or variable within a control block did not have correct value, or did not have any value at all. 2) Initialization of parameters. 3) Resetting a variable's value. 4) The instance variable capturing a characteristic of an object (e.g., the color of a car) is omitted. 5) The instance variables that capture the state of an object are not correctly initialized.
	Checking	**Errors caused by missing or incorrect validation of parameters or data in conditional statements. It might be expected that a consequence of checking for a value would require additional code such as a do-while loop or branch. If the missing or incorrect check is the critical error, checking would still be the type chosen.** Examples:1) Value greater than 100 is not valid, but the check to make sure that the value was less than 100 was missing. 2) The conditional loop should have stopped on the ninth iteration. But it kept looping while the counter was <= 10.
	Algorithm/ Method	**Efficiency or correctness problems that affect the task and can be fixed by (re-)implementing an algorithm or local data structure without the need for requesting a design change. Problem in the procedure, template, or overloaded function that describes a service offered by an object.** Examples: 1) The low-level design called for the use of an algorithm that improves throughput over the link by delaying transmission of some messages, but the implementation transmitted all messages as soon as they arrived. The algorithm that delayed transmission was missing. 2) The algorithm for searching a chain of control blocks was corrected to use a linearly linked list instead of a circularly linked list. 3) The number and/or types of parameters of a method or an operation are specified incorrectly. 4) A method or an operation is not made public in the specification of a class.

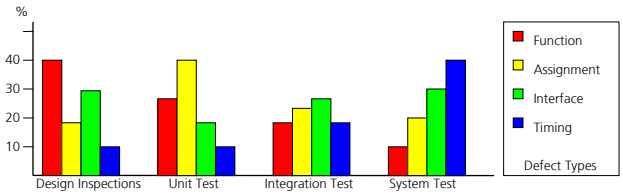
Definition of the attribute "Defect Type" (2/2)

Defect Type - What had to be fixed?

Defect Type - Attribute Values		
	Function/ Class/Object	**The error should require a formal design change, as it significantly affects capability, end-user interfaces, product interfaces, interface with hardware architecture, or global data structure(s). The error occurred when implementing the state and capabilities of a real or an abstract entity.** Examples:1) A database did not include a field for street address, although the requirements specified it. 2) A database included a field for postal zip code, but it was too small to contain international postal codes as specified in the requirements. 3) A C++ or SmallTalk class was omitted during system design.
	Timing/ Serialization	**Necessary serialization of shared resource was missing, the wrong resource was serialized, or the wrong serialization technique was employed.** Examples: 1) Serialization is missing when making updates to a shared control block. 2) A hierarchical locking scheme is in use, but the defective code failed to acquire the locks in the prescribed sequence.
	Interface/ O-O Messages	**Communication problems between: 1) modules 2) components 3) device drivers 4) objects 5) functions via 1) macros 2) call statements 3) control blocks 4) parameter lists** Examples: 1) A database implements both insertion and deletion functions, but the deletion interface was not made callable. 2) The interface specifies a pointer to a number, but the implementation is expecting a pointer to a character. 3) The OO message incorrectly specifies the name of a service. 4) The number and/or types of parameters of the OO message do not conform with the signature of the requested service.
	Relationship	**Problems related to associations among procedures, data structures, and objects. Such associations may be conditional.** Examples: 1) The structure of code/data in one place assumes a certain structure of code/data in another. Without appropriate consideration of their relationship, the program will not execute or it executes incorrectly. 2) The inheritance relationship between two classes is missing or specified incorrectly.

Rationale behind the ODC attribute “Defect Type”

- Characteristic defect type profiles are different for every activity of the development process
  - Unique ‘signature’ for each attribute value (i.e., values change over time)
  - Unique distribution for each detection activity



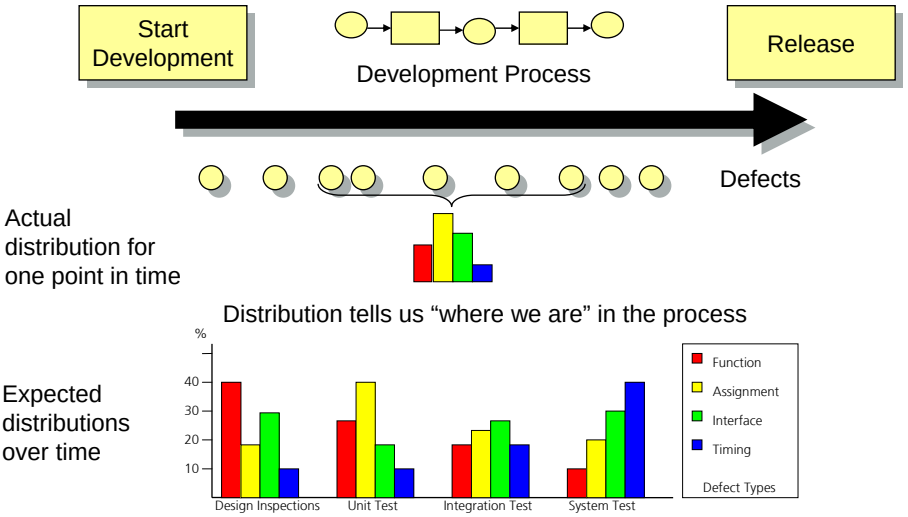
- Ideally, the attribute values should span the space of all possibilities they describe
  - Empirical approach: requires experience, many pilot projects to determine the appropriate values for an attribute

ODC attribute “Defect Type”

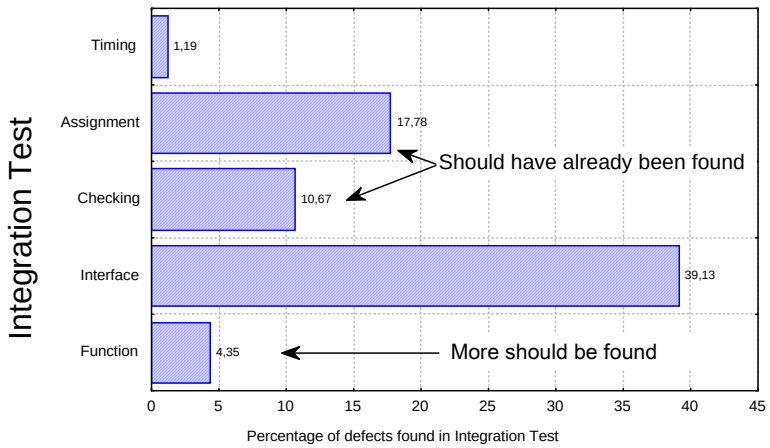
- **Developer** fixing the defect determines the defect type according to the **nature of the change**
- Each **defect type** can be **associated with** a software development phase

Defect Type	HLD	LLD	Code	UT	IT	ST
Function: requires formal design change	X					
Assignment: few LoC changed (e.g., initialization)			X	X		
Interface: interaction with other components, modules		X			X	
Algorithm: errors wrt. efficiency, correctness problems		X	X	X		
Checking: no proper validation of data, loop conditions		X	X	X		
Timing/Serialization: problem with shared resources		X				X
Relationship: associations among procedures, data structures	X					

Measuring progress with Defect Type



Example: Analyze Defect Type (1/2)



Conclusion - **Product entered integration test too early**

Example: Analyze Defect Type (2/2)

Result

- Too few defects associated with integration test
- Too many defects associated with unit test
- Product not mature enough for integration test

Cause

- Unit test completed too early

Implication

- Integration test is hampered by defects that it is not supposed to find

Action

- Go back to unit testing

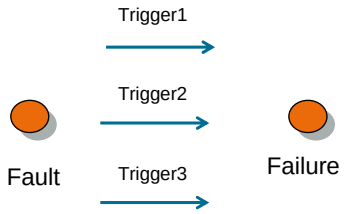
Validation

- During repeated unit test: more defects associated with unit test should be detected

ODC attributes in detail

Attribute	Attribute Meaning	Attribute Values
Defect Type	What had to be fixed?	assignment, checking, algorithm, function, timing, interface, relationship
Defect Qualifier	How can the defect be qualified?	missing, incorrect, extraneous
Activity	When did you detect the defect?	design inspection, code inspection, unit test, integration test, system test
Trigger	How did you detect the defect?	Inspection Trigger: design conformance, logic/flow, lateral compatibility, backward compatibility, language dependency, concurrency, side effects, rare situation Unit Test Trigger System Test Trigger
Impact	What would customer have noticed if defect had escaped into field?	installability, serviceability, standard, integrity/security, migration, reliability, performance, documentation, requirements, maintenance, usability, accessibility, capability
Target	What high-level entity was fixed?	requirements, **design, code**, build/package/merge, information, user interface
Source	Who developed the target?	in-house, library, outsourced, ported
Age	What is the history of the target?	base, new, rewritten, re-fixed

ODC attribute trigger



- **A trigger activates and/or discovers defects**. It is the catalyst that helps a defect to surface.
- Inspector or Tester classifies according to the condition that allows a defect to surface.
  - What did you think about?
  - Why did you write the test case?
- Knowing the best triggers for specific defect types enables earlier detection.

Defect lifecycle

Error → Fault → Activation →Failure

Triggers for different activities

Inspection	Unit Test	System Test
■ Backward Compatibility ■ Lateral Compatibility ■ Design Conformance ■ Logic Flow ■ Side Effects ■ Documentation ■ Rare Situation	■ Coverage ■ Sequencing ■ Interaction ■ Variation ■ Simple Path ■ Complex Path	■ Recovery ■ Start / Restart ■ Workload / Stress ■ HW Configuration ■ SW Configuration ■ Normal Mode (Block Test)

Example: typical Unit Test triggers (1/3)

What was the purpose of the test case (white box)?		
Attribute Value	**Definition**	**Example**
Simple Path	**The test case was motivated by the knowledge of specific branches in the code and not by the external knowledge of the functionality**. This trigger would not typically be selected for field-reported defects, unless the customer is very knowledgeable of the code and design internals, and specifically invokes a specific path (as is sometimes the case when the customer is a business partner or vendor).	Tried executing the "default" path of a case statement, but since it didn't exist, the test failed.
Complex Path	**In white-/gray-box testing, the test case that found the defect was executing some contrived combinations of code paths. In other words, the tester attempted to invoke the execution of several branches under several different conditions.** This trigger would only be selected for field-reported defects under the same circumstances as those described under Simple Path.	Path failed because one part of the subroutine released memory that subsequently was used in another part of that subroutine.

Example: typical Unit Test triggers (2/3)

What was the purpose of the test case (black box)?		
Attribute Value	**Definition**	**Example**
Coverage	**During black-box testing, the test case that found the defect was a straightforward attempt to exercise code** for a single function, using no parameters or a single set of parameters.	■ The tester tried to delete a city from the database but it couldn't be deleted. ■ Every latch set gets messages 'MSGISG101141' Formatting incomplete. Code='10'. This message is issued incorrectly because an error condition does not really exist.
Variation	**During black-box testing, the test case that found the defect was a straightforward attempt to exercise code for a single function but using a variety of inputs and parameters**. These might include invalid parameters, extreme values, boundary conditions, and combinations of parameters.	■ When the tester tried to add one more character than the maximum allowable, the software hung. ■ If an assignment of an operation to a work position is made, and on the operation details screen no work rule is specified, upon saving details, the client application crashes.

Example: typical Unit Test triggers (3/3)

What was the purpose of the test case (black box)?		
Attribute Value	**Definition**	**Example**
Sequencing	**During black-box testing, the test case that found the defect executed multiple functions in a very specific sequence.** This trigger is only chosen when each function executes successfully when run independently, but fails in this specific sequence. It may also be possible to execute a different sequence successfully.	■ The test case first added a record, then deleted it, and finally tried to add it again but got the message "You cannot add a record that already exists." ■ The "+" key was pressed twice and the program crashed.
Interaction	**During black-box testing, the test case that found the defect initiated an interaction among two or more bodies of code.** This trigger is only chosen when each function executes successfully when run independently, but fails in this specific combination. The interaction was more involved than a simple serial sequence of the executions.	Created a UWIP of type 'Serialized Unit'. Saved the UWIP without assigning a serial to the part. Reopened the UWIP from the UWIP distance view, went to parts and selected 'Edit'. Now the list of available serials comes up empty. Can't assign any serial to that part anymore.

Example: typical System Test triggers (1/2)

What was the purpose of the test case?		
Attribute Value	**Definition**	**Example**
Workload/ Stress	**The system is operating at or near some resource limit, either upper or lower**. These resource limits can be created by means of a variety of mechanisms, including running small or large loads, running a few or many products at a time, letting the system run for an extended period of time.	■ When the system is idle for 10 minutes, it hangs. ■ ISGGRP00 should not hold lock for so long that it causes the rest of the complex to hang. After processing a certain number of requests, it should release and then re-obtain the lock in order to give other units of work, specifically ring processing, a chance to execute.
Recovery/ Exception	**The system is being tested with the intent of invoking an exception handler or some type of recovery code**. The defect would not have surfaced if some earlier exception had not caused exception or recovery processing to be invoked. From a field perspective, this trigger would be selected if the defect is in the system's or product's ability to recover from a failure, not the failure itself.	■ ISGQSCAN recovery routine ISGQSCNR converts unexpected error in ISGQSCAN to ABEND09A and RCA220, instead of ABEND322. ■ After an abend, a DISPLAY GRS Contention command is entered, the MASID/MTCB issuer has JOBNAME/*UNKNOWN in the MSGISG020I output. The bit RIBESDIV has been set in an invalid manner.
Startup/ Restart	**The system or subsystem was being initialized or restarted** following some earlier shutdown or complete system or subsystem failure.	■ After pulling the plug on the CPU, the software was not able to start up again until some files left in one of the directories were cleaned out.

Example: typical System Test triggers (2/2)

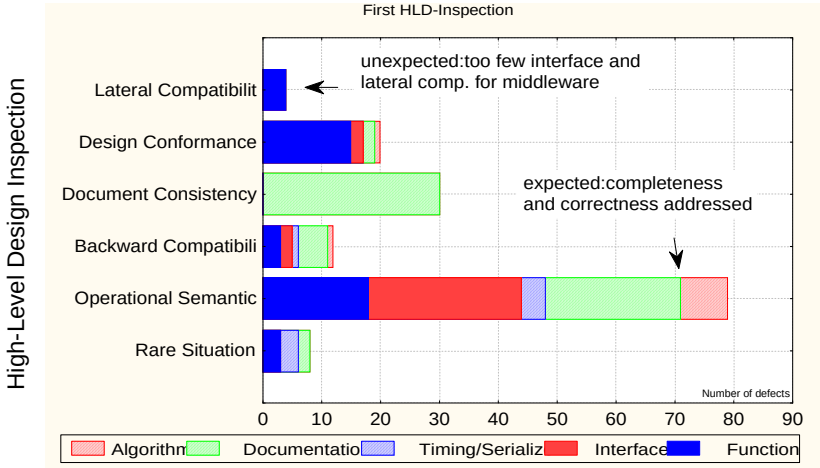
What was the purpose of the test case?		
Attribute Value	**Definition**	**Example**
Hardware Configuration	The system is being **tested to ensure functions execute correctly under specific hardware configurations.**	Found a defect when sending any job to a particular brand printer.
Software Configuration	The system is being tested to **ensure functions execute correctly under specific software configurations.**	TRYJOIN does not work in a non-sysplex environment.
Blocked Test (previously Normal Mode)	**The product is operating well within resource limits and the defect surfaced while attempting to execute a system test scenario.** This trigger would be used when the scenarios could not be run because there are basic problems that prevent their execution. This trigger must not be used in customer-reported defects.	During system test, wanted to check for workload stress by printing 1000 jobs. However, when Print was clicked, nothing happened. No screen appeared to prompt for input.

Example of triggers: Inspection Trigger

Each trigger value can be associated with a certain skill level of people

Inspection Trigger	Novice	Product experience	Cross-product experience	Expert
Design Conformance (compare design with specification)	X	X	X	X
Understanding Details (of structure, operation)		X	X	X
Backward Compatibility (to previous version of product)		X		
Lateral Compatibility (to other systems/services)			X	
Concurrency (simultaneous use of resources)				X
Logic Flow (operational semantics in question)		X		
Side Effects (potential impact on another function/product)		X	X	
Documentation (code comments, user guides, manuals)		X	X	X
Rare Situation (unusual sets of circumstances)				X

Trigger application: evaluate inspection effectiveness (1/3)



Trigger application: evaluate inspection effectiveness (2/3)

Result

- Too few defects associated with lateral compatibility and interface

Cause

- Inspection team consisted mainly of inexperienced inspectors; thus, compatibility issues were not considered adequately

Implication

- Many crucial defects still remain, causing existing customer applications to fail

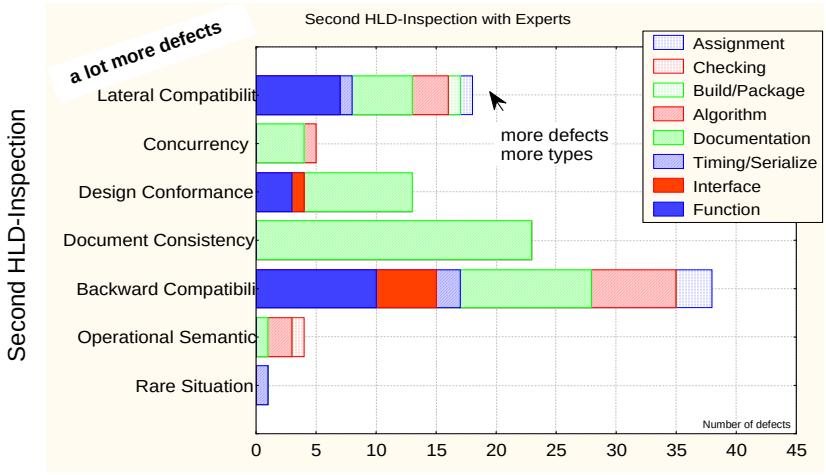
Action

- Re-inspect with more experienced inspectors concentrating on deficiencies

Validation

- → see next slide

Trigger application: evaluate inspection effectiveness (3/3)



ODC attributes in detail

Attribute	Attribute Meaning	Attribute Values
Defect Type	What had to be fixed?	assignment, checking, algorithm, function, timing, interface, relationship
Defect Qualifier	How can the defect be qualified?	missing, incorrect, extraneous
Activity	When did you detect the defect?	design inspection, code inspection, unit test, integration test, system test
Trigger	How did you detect the defect?	Inspection Trigger: design conformance, logic/flow, lateral compatibility, backward compatibility, language dependency, concurrency, side effects, rare situation Unit Test Trigger System Test Trigger
Impact	What would customer have noticed if defect had escaped into field?	installability, serviceability, standard, integrity/security, migration, reliability, performance, documentation, requirements, maintenance, usability, accessibility, capability
Target	What high-level entity was fixed?	requirements, **design, code**, build/package/merge, information, user interface
Source	Who developed the target?	in-house, library, outsourced, ported
Age	What is the history of the target?	base, new, rewritten, re-fixed

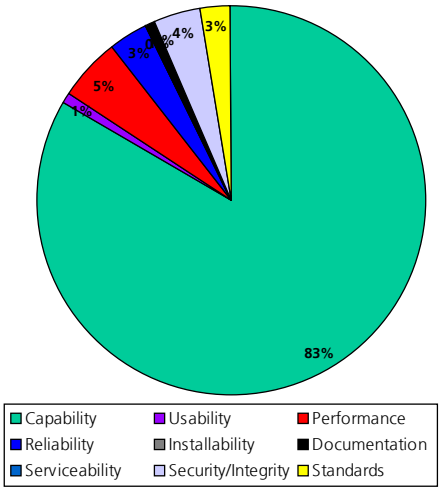
ODC attribute impact (1/2)

What would customer have noticed if defect had escaped into field?		
Attribute Value	Definition	Example
Installability	The ability of the customer to prepare and place the software in position for use. (Does not include Usability.)	During automated installation, got an error message saying installation failed because a file was missing.
Serviceability	The ability to diagnose failures easily and quickly, with minimal impact on the customer.	The diagnostics software number error messages rather than indicating where the problem actually occurred.
Standards	The degree to which the software complies with established pertinent standards.	
Integrity/Security	The protection of systems, programs, and data from inadvertent or malicious destruction, alteration, or disclosure.	Logged in as Read Only, Profiles enabled. Was able to save changes from the System Component Assignment Panel. Was also able to delete a component.
Migration	The ease of upgrading to a current release, particularly in terms of the impact on existing customer data and operations.	Co-requisite information with regard to other products is not made available to customers.
Reliability	The ability of the software to consistently perform its intended function without unplanned interruption.	While invoking modem software, system crashed and had to be rebooted.
Performance	The speed of the software as perceived by the customer and the customer's end users, in terms of their ability to perform their tasks.	A module should not hold a local lock for so long that it causes the rest of the complex to hang.

ODC attribute impact (2/2)

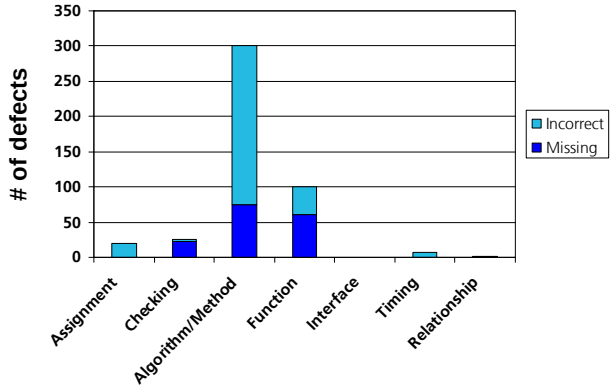
What would customer have noticed if defect had escaped into field?		
Attribute Value	Definition	Example
Documentation	The degree to which the publication aids provided for understanding the structure and intended uses of the software are correct and complete.	MSGISG015I RCAAE78 is not documented in the system messages manual.
Requirements	A customer expectation with regard to capability, which was not known, understood, or prioritized as a requirement for the current product or release.	Customer needs the software to have the ability to take input either from the keyboard, mouse, OR flat files.
Maintenance	The ease of applying preventive or corrective fixes to the software.	Fixes cannot be applied due to a bad medium.
Usability	The degree to which the software and publication aids enable the product to be easily understood and conveniently employed by its end user.	When running several jobs in system test, the system was flooded with messages. They scrolled by so quickly they couldn't be read.
Accessibility	Ensuring that successful access to information and use of information technology is provided to people who have disabilities.	No reading support available.
Capability	The ability of the software to perform its intended functions, and satisfy KNOWN requirements, where the customer is not impacted in any of the previous categories.	When save was clicked on, nothing happened.

Example: Improve customer satisfaction through impact (1/2)



- Analyze the attribute **Impact** to improve customer satisfaction
- Impact analysis shows major problem: **deficiencies with functionality**

Example: Improve customer satisfaction through impact (2/2)



- Defect-type analysis shows:
  - >50% missing function hints at **incomplete design or requirements**
  - Algorithm defects hint at **poor low-level design**

Example: Usage of attributes Trigger and Impact for System Test (1/5)

Defects can be everywhere in the defect space.

- How can we plan/control system test?

Trigger and Impact can be used to partition the defect space.

Example: Usage of attributes Trigger and Impact for System Test (2/5)

Attributes Trigger and Impact partition defect space

	Reliability	Performance	Integrity	Installabil.	Migration	Standards	Servicabil.	Usability	Capability	Documen.
Recovery										
Start/ Restart										
Workload/ Stress										
HW/SW Config.										
Normal Mode										

Example: Usage of attributes Trigger and Impact for System Test (3/5)

Identification of relative "defectiveness" of each partition

	Reliability	Performance	Integrity	Installabil.	Migration	Standards	Servicabil.	Usability	Capability	Documen.
Recovery	1					1		1	2	
Start/ Restart									1	
Workload/ Stress										
HW/SW Config.									9	
Normal Mode	5	1		2					8	

Mapping of defects within the defect space

Example: Usage of attributes Trigger and Impact for System Test (4/5)

Classification of test cases according to Trigger and Impact

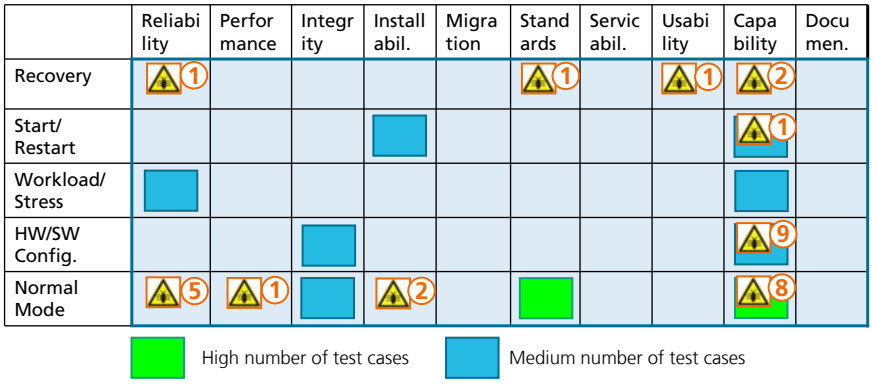
	Reliability	Performance	Integrity	Installabil.	Migration	Standards	Servicabil.	Usability	Capability	Documen.
Recovery										
Start/ Restart				2					12	
Workload/ Stress	4								2	
HW/SW Config.			2						24	
Normal Mode			1			68			174	

High number of test cases Medium number of test cases

Identification of gaps in testing

Example: Usage of attributes Trigger and Impact for System Test (5/5)

Combination of test cases and defects enables focused action



- Identify areas where more testing is needed
- Identify saturated areas – inefficient testing

Part 2: Defect Classification

- **Defect Classification Theory**
  - Terminology
  - Structures for Classification
- **Prominent Approaches**
  - IEEE Standard Classification Scheme
  - Hewlett-Packard (HP) Classification Scheme
  - Orthogonal Defect Classification (ODC)
- **Summary**

Summary

Defect Classification Theory

- Basic terminology
- Structures for classification (categorization, hierarchical, hybrid)

Most Prominent Approaches

- IEEE 1044 Standard Classification Scheme
  - Defect classification scheme compliant to a standard
- Hewlett-Packard (HP) Classification Scheme
  - Focus on improvement of development process by reducing the number of defects
  - Three descriptors for each defect: origin, type of defect, and mode
- Orthogonal Defect Classification (ODC)
  - Can also be used to improve development process by reducing the number of defects
  - Original purpose is to give project teams feedback on the progress of the current project
  - Highest industry acceptance

References

[1] Institute of Electrical and Electronics Engineers, IEEE Standard Classification for Software Anomalies, IEEE Std. 1044-1993, 1994.

[2] Robert B. Grady, Practical Software Metrics for Project Management and Process Improvement. Prentice Hall, 1992.

[3] Ram Chillarege, Inderpal S. Bhandari, Jarir K. Chaar, Michael J. Halliday, Diane S. Moebus, Bonnie K. Ray, and Man-Yuen Wong, Orthogonal defect classification -- A concept for in-process measurements, IEEE Transactions on Software Engineering, vol. 18, pp. 943--956, Nov. 1992

[4] IBM, Center for Software Engineering
http://www.research.ibm.com/softeng/ODC/ODC.HTM