Introduction Software Quality Assurance & Software Testing

Presented by : Lamhot

What is SQA?

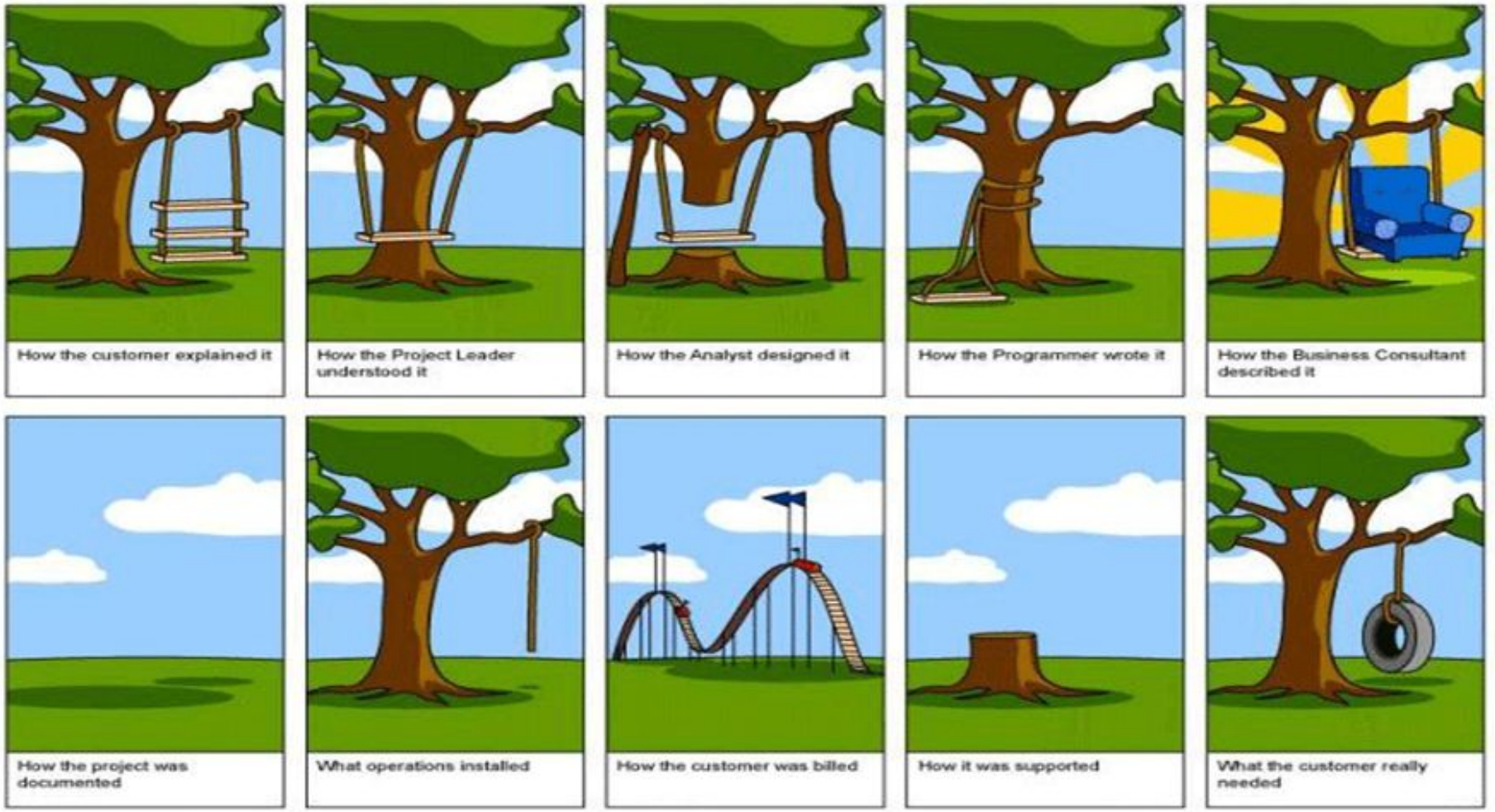
Involves the entire software development PROCESS:

- monitoring and improving the process
- making sure that any agreed-upon standards and procedures are followed
- ensuring that problems are found and dealt with.

Software Quality Assurance (SQA) is an umbrella activity that is applied throughout the software development process

Testing is the process of identifying **defects**, where a defect is any **variance between actual and expected results**. “A mistake in coding is called **Error**, error found by tester is called **Defect**, defect accepted by development team then it is called **bug**, build does not meet the requirements then it Is **Failure**.”

Decreasing “The Cycles of Miscommunication”



QA + Testing = good software

“the right things at the right time in the right way”
“makes sure the results of what you've done are what you expected”

Testing Terminologies

- **Error:** the amount by which the result in incorrect (formally)
- **Mistake / error**: a human action that produces an incorrect result
- **Fault**: an incorrect step, process, or data definition in a computer program
- **Failure**: an incorrect result as observed from the program output (the manifestation of faults after the program is run)
- **Wrong:** When requirements are implemented not in the right way. This defect is a variance from the given specification. It is Wrong!
- **Missing**: A requirement of the customer that was not fulfilled. This is a variance from the specifications, an indication that a specification was not implemented, or a requirement of the customer was not noted correctly.
- **Extra**: A requirement incorporated into the product that was not given by the end customer. This is always a variance from the specification, but may be an attribute desired by the user of the product. However, it is considered a defect because it's a variance from the existing requirements.

Testing Terminologies

- *A test case:* the input values (test data), expected output (produced by test oracle), the test objective
- A test suite: a group of test cases
- *A good test case*: has high chance to reveal errors
- *A successful test case*: test case that uncover new error

Test Suite & Test Cases

Test Case	Test Case Desc	Expectation
pd-001	discount product	should have discount badge original price & reduce price
pd-002	installment product (price < 500.000)	should have label (Nikmati Cicilan 0%)
pd-003	free shipping product	should have gratis biaya kirim tooltip
pd-004	wholesale product	should have wholesale tooltip
pd-005	seller has subscriber	should have subscriber badge & subscriber count on right sidebar pelapak
pd-006	premium seller	should have premium badge on right sidebar pelapak
pd-007	seller has average shipping time	should have waktu kirim tooltip (Waktu rata-rata yang diperlukan pelapak untuk mengirim pesanan yar
pd-008	seller with notes	should have catatan pelapak on product detail tab
pd-009	seller with review list	should have product rating badge, review tab & list
pd-010	seller with feedback list	should have pelapak feedback link, feedback tab & list
pd-011	minimum product info	should have social media icon, product detail tab (information, spesification, description) & Shipping
pd-012	installment product (price>=500.000)	should have Bank penyedia cicilan & Cicilan tanpa kartu kredit tooltip
pd-013	non active seller	should have "PELAPAK TIDAK AKTIF" label & /p/hobi-hiburan/sport/basket/1-jual-bola-basket
pd-014	draft product	should have "BARANG DRAFT" label & user should not able to buy/add to cart
pd-015	stock 0	should have "STOK HABIS" label & user should not able to buy/add to cart
pd-016	product has been deleted	should have "BARANG TELAH DIHAPUS" label & user should not able to buy/add to cart
pd-017	profuct unverify	should have "BELUM DIVERIVIKASI" label & user should not able to buy/add to cart
pd-018	user has been killed	should have "BELUM DIVERIVIKASI" label & user should not able to buy/add to cart
pd-019	user has been frozen	should have "BELUM DIVERIVIKASI" label & user should not able to buy/add to cart
pd-020	Seller has been closed	should have "LAPAK TUTUP" label & user should not able to buy/add to cart
pd-021	Seller has been reported	should have "MELANGGAR" label & user should not able to buy/add to cart

Test case specification components

- Test Case Specification Identifier
- Test Items
- Input Specifications
- Output Specifications
- Special Environmental Needs
- Special Procedural Requirements
- Inter case Dependencies

Test Procedure Specifications

- It specifies the steps required to execute a set of test cases.
- it specifies the steps necessary to analyze a software item in order to evaluate a set of features.
- Test Procedure Specification Identifier
- Purpose
- Specific Requirements
- Procedure Steps (Scenarios)

When testing is stopped ?

- No definite answer
- Possible answers:
- When testing resources are run out
- When the level of reliability as specified in SRS are reached
- When the tester is confident on the quality of the software

Approach to select test cases

- Black Box Testing: does not refer to the code of the program
- White Box Testing: refers to the code of the program under test
- Fault Based Testing: shows the absence of certain type of faults

Testing Design Strategies

- It needs to develop what we will call effective test cases for execution-based testing.
- By an effective test case we mean one that has a good possibility of revealing a defect
- These are called the black box (sometimes called functional or specification) and white box (sometimes called clear or glass box) test strategies

Using the Black Box Approach to Test Case Design

Black-box tests are derived from an understanding of the purpose of the code; knowledge on or about the actual internal program structure is not required when using this approach. The risk involved with this type of approach is that .hidden. (functions unknown to the tester) will not be tested and may not been even exercised.

Equivalence Class Partitioning

Equivalence testing leverages the concept of "classes" of input conditions.

Example: Negotiation Feature

range	value	Max Nego %	Min Nego %	Discount Max	Discount Min
(a) 0-10k	10,000	20%	10.00%	2,000	1,000
(b) 10-50k	50,000	15%	6.00%	7,500	3,000
(c) 50-100k	100,000	12%	6.00%	12,000	6,000
(d) 100-250k	250,000	10%	4.00%	25,000	10,000
(e) 250k-500k	500,000	8%	3.00%	40,000	15,000
(f) 500-750k	750,000	8%	2.70%	60,000	20,000
(g) 750-1000k	1,000,000	7%	2.50%	70,000	25,000
(h) 1000-1500k	1,500,000	7%	2.40%	105,000	35,000
(i) 1500-3000k	3,000,000	6%	1.70%	180,000	50,000
(j) 3000-5000k	5,000,000	5%	1.30%	250,000	65,000
(k) >5000k	10,000,000	4%	0.75%	400,000	75,000

Test case design

- After the equivalence classes have been identified in this way, the next step in test case design is the development of the actual test cases.
- A good approach includes the following steps.
  1. Each equivalence class should be assigned a unique identifier. A simple integer is sufficient.
  2. Develop test cases for all valid equivalence classes until all have been covered by (included in) a test case. A given test case may cover more than one equivalence class.
  3. Develop test cases for all invalid equivalence classes until all have been covered individually. This is to insure that one invalid case does not mask the effect of another or prevent the execution of another.

Boundary Value Analysis

Boundary-value analysis is really a variant on Equivalence Partitioning but in this case the upper and lower end of the class and often values outside the valid range of the class are used for input into the test cases. For example, if the Class in "Numeric Month of the Year" then the Boundary-values could be 0, 1, 12, and 13.

Bugs : Error 500 Add / delete popular product (product without /p)			
No Test Case	**test case name**	**expect**	**result**
dtb_pp_01	product url with /p (full url)	success	success
dtb_pp_02	product url without /p (full url)	success	success
dtb_pp_03	include http://	success	success
dtb_pp_04	exclude htttp://	success	success
dtb_pp_05	exclude www	success	success
dtb_pp_06	no base url	success	success
dtb_pp_07	random string (invalid)	Flash message: k tidak valid atau tid	success
dtb_pp_08	empty/blank	Flash message: k tidak valid atau tid	500
dtb_pp_09	integer out if range (loremipsum)	Flash message: k tidak valid atau tid	500

Cause-and-Effect Graphing

- Cause-and-effect graphing is a technique that can be used to combine conditions and derive an effective set of test cases that may disclose inconsistencies in a specification.
- However, the specification must be transformed into a graph that resembles a digital logic circuit.
- The tester is not required to have a background in electronics, but he should have knowledge of Boolean logic.
- Developing the graph, especially for a complex module with many combinations of inputs, is difficult and time consuming. The graph must be converted to a decision table that the tester uses to develop test cases.

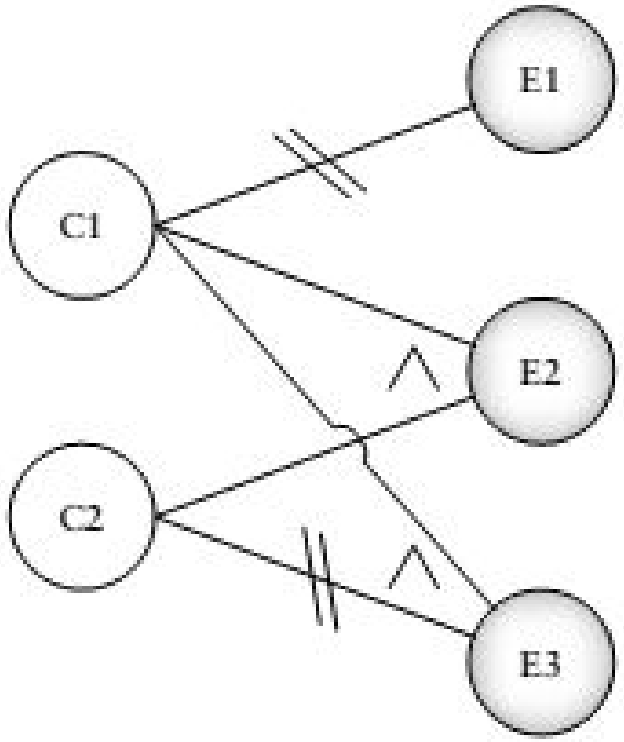
Example

Suppose we have a specification for a module that allows a user to perform a search for a character in an existing string. The specification states that the user must input the length of the string and the character to search for. If the string length is out-of-range an error message will appear. If the character appears in the string, its position will be reported. If the character is not in the string the message“not found” will be output.

Result

- The input conditions, or causes are as follows:
  - C1: Positive integer from 1 to 80
  - C2: Character to search for is in string
- The output conditions, or effects are:
  - E1: Integer out of range
  - E2: Position of character in string
  - E3: Character not found
- The rules or relationships can be described as follows:
  - If C1 and C2, then E2.
  - If C1 and not C2, then E3.
  - If not C1, then E1.

Graph



Result

- If the existing string is“abcde,”then possible tests are the following:

Inputs	Length	Character to search for	Outputs
T1	5	c	3
T2	5	w	Not found
T3	90		Integer out of range

Decision Table

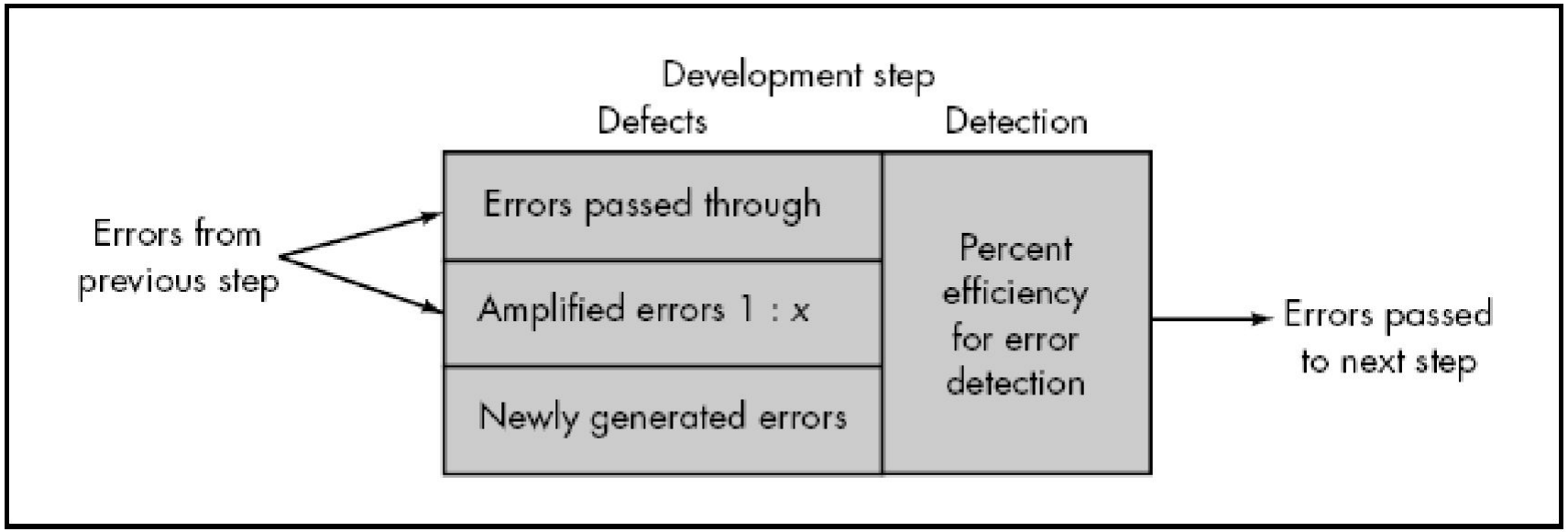
	T1	T2	T3
C1	1	1	0
C2	1	0	—
E1	0	0	1
E2	1	0	0
E3	0	1	0

Other Methods

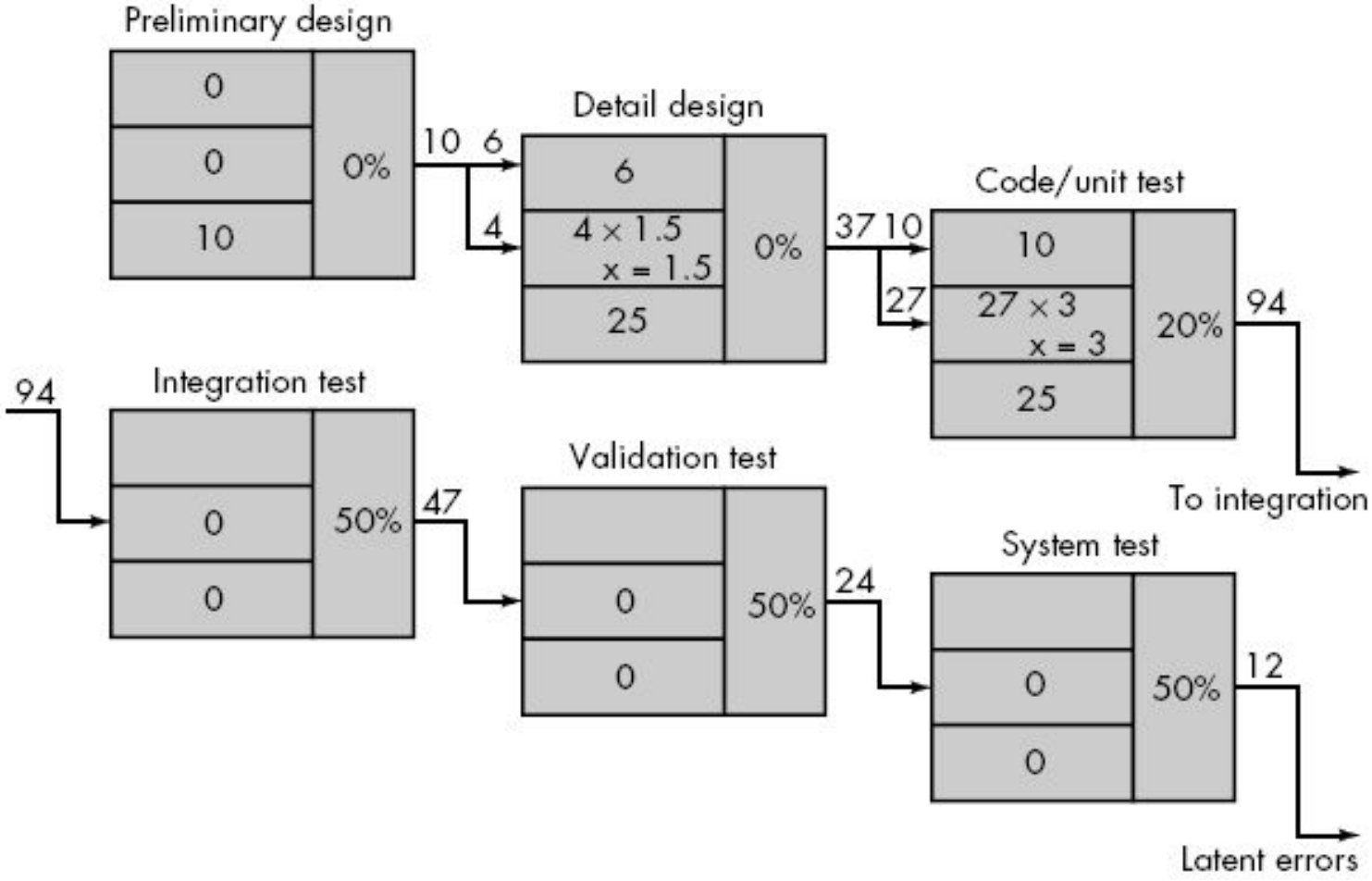
- State Transition Testing
  - It is based on the concepts of states and finite-state machines, and allows the tester to view the developing software in term of its states, transitions between states, and the inputs and events that trigger state changes.
  - This view gives the tester an additional opportunity to develop test cases to detect defects that may not be revealed using the input/output condition as well as cause-and-effect views presented by equivalence class partitioning and cause-and-effect graphing.
- Error Guessing
  - based on the tester's/developer's past experience with code similar to the code-under-test, and their intuition as to where defects may lurk in the code.
  - Code similarities may extend to the structure of the code, its domain, the design approach used, its complexity, and other factors.

Cost impact of software defects

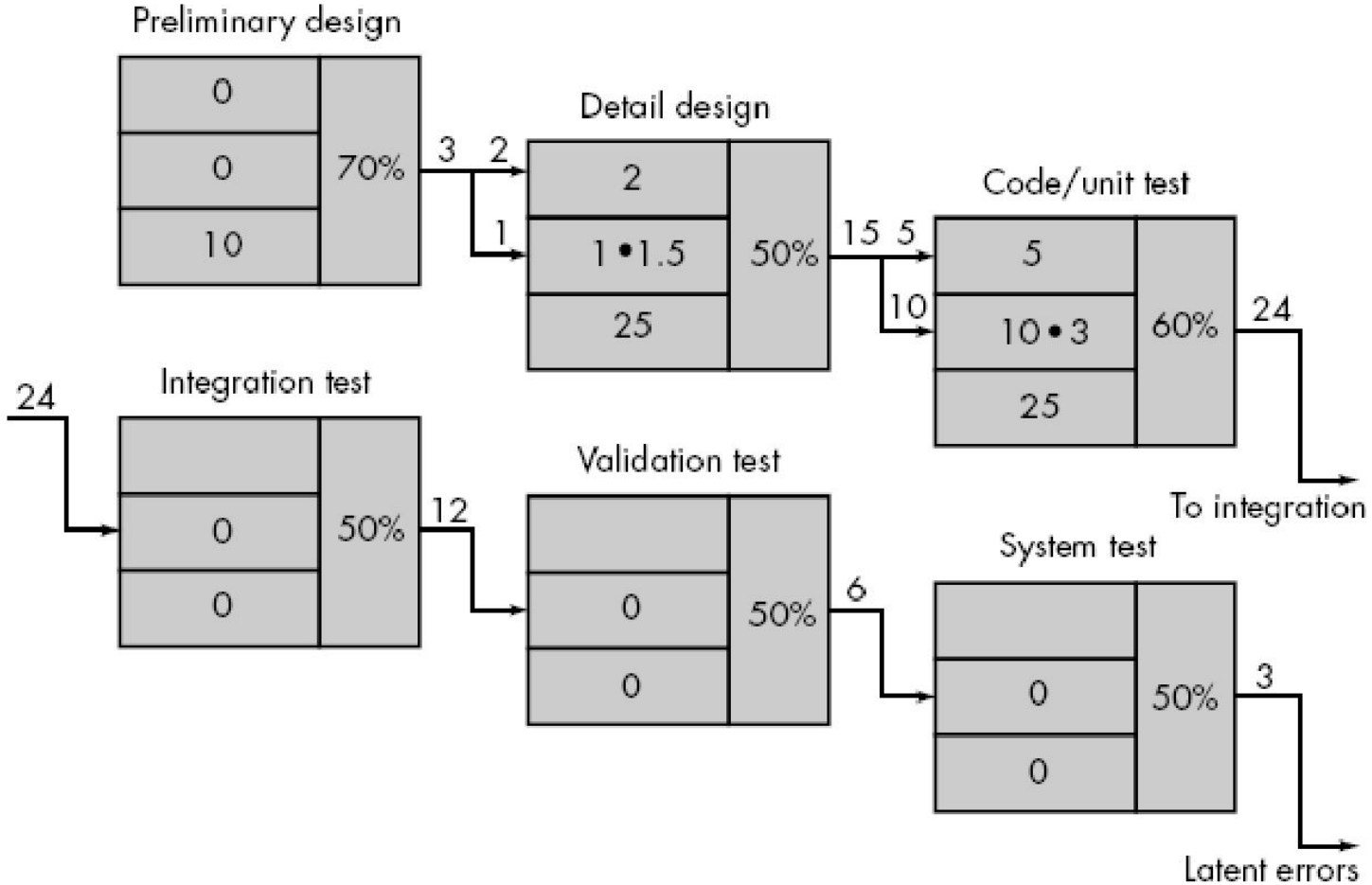
A defect amplification model (IBM Corporation, 1981) can be used to illustrate the generation and detection of errors during the preliminary design, detail design, and coding steps of the software engineering process.



Example of defect amplification for a software development process: no reviews



Example of defect amplification for a software development process: with reviews



References

- Pressman., Roger S., Software Engineering (Fifth Edition), McGraw-Hill International Editions, 2001
- IBM Corporation, “Implementing Software Inspections,” course notes, IBM Systems Sciences Institute, IBM Corporation, 1981.
- Jones, T.C., *Programming Productivity,* McGraw-Hill, 1986.
- ITD., Slide STQA IE 331315 and IE 431315, Arlinta C. Barus & Arnaldo M. Sinaga., 2014