Software Testing Preliminaries

Dr. Mark C. Paulk

SE 4367 – Software Testing, Verification, Validation, and Quality Assurance

Topics: Software Testing

<u>Part I: Preliminaries</u>

1. Software Testing

- **Humans, Errors, and Testing**
- **Software Quality**
- **Requirements, Behavior, and Correctness**
- **Correctness vs Reliability**
- ***Testing and Debugging***
- ***Test Metrics***
- ***Software and Hardware Testing***
- ***Testing and Verification***
- ***Defect Management***
- ***Test Generation Strategies***
- ***Static Testing***
- ***Model-Based Testing and Model Checking***
- ***Types of Testing***
- ***Saturation Effect***
- ***Principles of Testing***

The Purpose of Testing

What is the purpose of testing?

To demonstrate that the software works?

NO!

To find defects in the software?

YES!

There is always the possibility of latent defects…

ISO/IEC/IEEE 29119-4:2015 (Software Testing, Part 4: Test Techniques)

<u>Specification-based test design techniques</u>

- **equivalence partitioning**
- **classification tree method**
- **boundary value analysis**
- **syntax testing**
- **combinatorial test design techniques**
- **decision table testing**
- **cause-effect graphing**
- **state transition testing**
- **scenario testing**
- **random testing**

<u>Structure-based test design techniques</u>

- **statement testing**
- **branch testing**
- **decision testing**
- **branch condition testing**
- **branch condition combination testing**
- **modified condition decision coverage (MCDC) testing**
- **data flow testing**

<u>Experience-based test design techniques</u>

- **error guessing**

Software Testing Body of Knowledge

International Software Testing Qualifications Board (ISTQB)

- **Certified Tester Foundation Level Syllabus**
- **Advanced Level**
  - **Test Manager, Test Analyst, Technical Test Analyst**
- **Expert Level**
  - **Test Manager, Strategic Test Management, Operational Test Management, Managing the Test Team, Improving the Testing Process, Assessing Test Processes, Implementing Test Process Improvement**
- **Certified Tester Foundation Level Extension Syllabus Agile Tester**
- **www.istqb.org**

Also… textbooks, SWEBOK (and other BOKs), TPI, TMM, ISEB, ISO & IEEE standards, …

ISTQB® FOUNDATION LEVEL

ISTQB® – FOUNDATION LEVEL

Fundamentals of Testing	Testing Throughout the Software Life Cycle	Static Techniques	Testing Design Techniques	Test Management	Tool Support for Testing
Why is Testing Necessary?	Software Development Models	Static Techniques and the Test Process	The Test Development Process	Test Organization	Types of Test Tools
What is Testing?	Test Levels	Review Process	Categories of Test Design Techniques	Test Planning and Estimation	Effective use of Tools: Potential Benefits and Risks
Seven Testing Principles	Test Types	Static Analysis by Tools	Specification-based Techniques (black-box)	Test Progress Monitoring and Control	Introducing a Tool into an Organization
Fundamental Test Process	Maintenance Testing		Structure-based Techniques (white-box)	Configuration Management	
The Psychology of Testing			Experience-based Techniques	Risk and Testing	
Code of Ethics			Choosing Test Techniques	Incident Management	

Fundamental Testing Techniques

Black box testing

- **cover the requirements**
- **cover the equivalence classes**
- **cover the boundary values (3-point or 2-point)**

White box testing

- **cover the statements**
- **cover the decisions**
- **cover the conditions**

Verification & Validation

Verification

"Are we building the product right".

The software should conform to its specification.

Validation

"Are we building the right product".

The software should do what the user really requires.

Verification
(IEEE 12207:2017 6.4.9)

The purpose of the Verification process is to provide objective evidence that a system or system element fulfils its specified requirements and characteristics.

- **The Verification process identifies the anomalies (errors, defects, or faults) in any information item (e.g., system/software requirements or architecture description), implemented system elements, or life cycle processes using appropriate methods, techniques, standards or rules.**
- **This process provides the necessary information to determine resolution of identified anomalies.**

a) Constraints of verification that influence the requirements, architecture, or design are identified.
b) Any enabling systems or services needed for verification are available.
c) The system or system element is verified.
d) Data providing information for corrective actions is reported.
e) Objective evidence that the realized system fulfills the requirements, architecture and design is provided.
f) Verification results and anomalies are identified.
g) Traceability of the verified system elements is established.

Validation
(IEEE 12207:2017 6.4.11)

The purpose of the <u>Validation</u> process is to provide objective evidence that the system, when in use, <u>fulfils its business or mission objectives</u> and stakeholder requirements, <u>achieving its intended use in its intended operational environment</u>.

- **The objective of validating a system or system element is to acquire confidence in its ability to achieve its intended mission, or use, under specific operational conditions.**
- **Validation is ratified by stakeholders.**
- **This process provides the necessary information so that identified anomalies can be resolved by the appropriate technical process where the anomaly was created.**

a) Validation criteria for stakeholder requirements are defined.
b) The availability of services required by stakeholders is confirmed.
c) Constraints of validation that influence the requirements, architecture, or design are identified.
d) The system or system element is validated.
e) Any enabling systems or services needed for validation are available.
f) Validation results and anomalies are identified.
g) Objective evidence that the realized system or system element satisfies stakeholder needs is provided.
h) Traceability of the validated system elements is established.

Software V&V in IEEE 729:1983

Verification

- **The process of determining whether or not the products of a given phase of the software development cycle fulfill the requirements established during the <u>previous phase</u>.**

Validation

- **The process of evaluating software at the end of the software development process to ensure compliance with <u>software requirements</u>.**

Three Mechanisms for V&V

Testing

- **"test" is a generic word – see Weinberg's <u>Perfect Software and Other Illusions About Testing</u>**
  - **does (not) meet need**
  - **does (not) meet all requirements**
  - **(un)acceptable costs or constraints**
  - **grossly unreliable**
  - **poor performance**
- **by software testing, we typically assume dynamic execution of a program**
  - **compare the outcome expected with the actual outcome**
  - **black box, white box, unit, integration, system, regression, etc., testing**

Formal methods

- formal specifications
- model / property checking
- proofs of correctness
  - Correctness attempts to establish the program is error-free; testing attempts to find if there are any errors in the program.

Peer reviews

- A review of a software work product, following defined procedures, by peers of the producers of the product for the purpose of identifying defects and improvements. (Software CMM)
  - managers are not peers…
  - customers are not peers…

Generic V&V Techniques

- Technical reviews, management reviews, joint reviews
- Symbolic execution, program proving, proof of correctness, formal methods
- Anomaly analysis, syntactical checks
- Functional testing, black-box testing, equivalence partitioning, boundary value analysis
- Structural testing, white-box testing, basis path testing, condition testing, data flow testing, loop testing
- Unit testing
- Regression testing, daily build and smoke test
- Integration testing
- Random testing, adaptive perturbation testing, mutation testing, be-bugging
- Operational profile testing, stress testing, performance testing
- System testing, acceptance testing
- Peer reviews, structured walkthroughs, inspections, active design reviews, pair programming, …

Correctness vs Testing

Correctness is established via mathematical proofs of programs.

Proofs are precise but subject to human fallibility.

"Beware of bugs in the above code; I have only proved it correct, not tried it."
- Donald Knuth, 1977

Correctness attempts to establish the program is error-free; testing attempts to find if there are any errors in the program.

"Program testing can be used to show the presence of bugs, but never to show their absence!"
- Edsger Dijkstra, "Notes on Structured Programming," 1970

Quality

1) **the degree to which a system, component, or process meets specified requirements.**
2) **ability of a product, service, system, component, or process to meet customer or user needs, expectations, or requirements.**
3) **the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs.**
4) **conformity to user expectations, conformity to user requirements, customer satisfaction, reliability, and level of defects present.**
5) **the degree to which a set of inherent characteristics fulfils requirements.**
6) **the degree to which a system, component, or process meets customer or user needs or expectations.**

IEEE 24765: 2010 (Systems and Software Engineering – Vocabulary)

Also see R. Glass, "Defining Quality Intuitively," IEEE Software, May/June 1998.

Counting Defects

When programmers make mistakes, the defects injected are on the order of 10% of the SLOC.

- **over 100 defects / KSLOC from PSP**
- **the “normal” development process will remove about half of the defects before they are ever “officially” counted**

Defects will be distributed among requirements, design, code, ...

- **there will be ripple effects from defects injected earlier in the process**

Defect Potentials

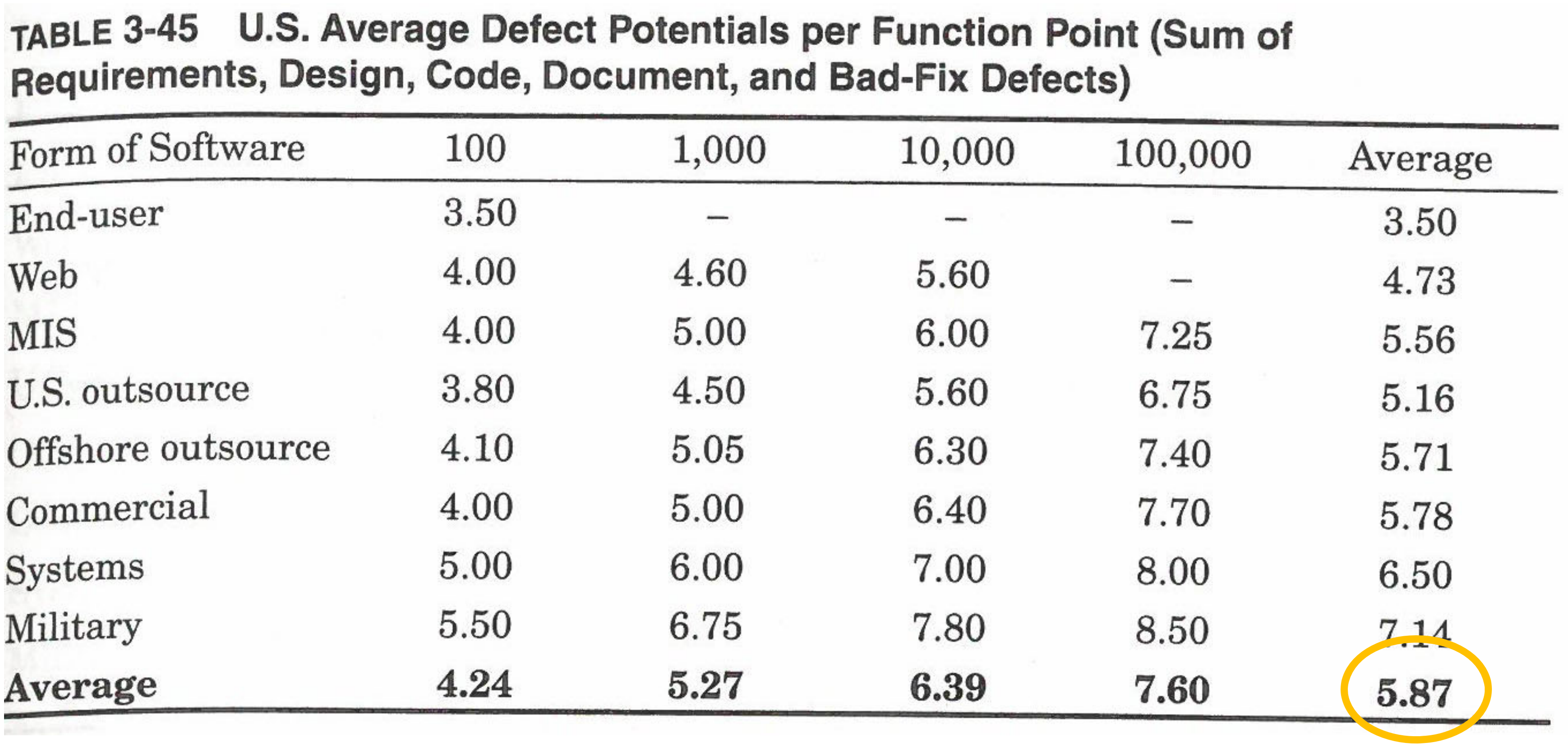


TABLE 3-45 U.S. Average Defect Potentials per Function Point (Sum of Requirements, Design, Code, Document, and Bad-Fix Defects)

Form of Software	100	1,000	10,000	100,000	Average
End-user	3.50	–	–	–	3.50
Web	4.00	4.60	5.60	–	4.73
MIS	4.00	5.00	6.00	7.25	5.56
U.S. outsource	3.80	4.50	5.60	6.75	5.16
Offshore outsource	4.10	5.05	6.30	7.40	5.71
Commercial	4.00	5.00	6.40	7.70	5.78
Systems	5.00	6.00	7.00	8.00	6.50
Military	5.50	6.75	7.80	8.50	7.14
Average	**4.24**	**5.27**	**6.39**	**7.60**	**5.87**

C. Jones, Applied Software Measurement, Third Edition, 2008.

- **information from projects that used formal design and code inspections plus multistage testing activities**
- **project size measured in function points (100, 1000, …) (x100 SLOC)**

Defect Removal Effectiveness *(Jones 2008)*

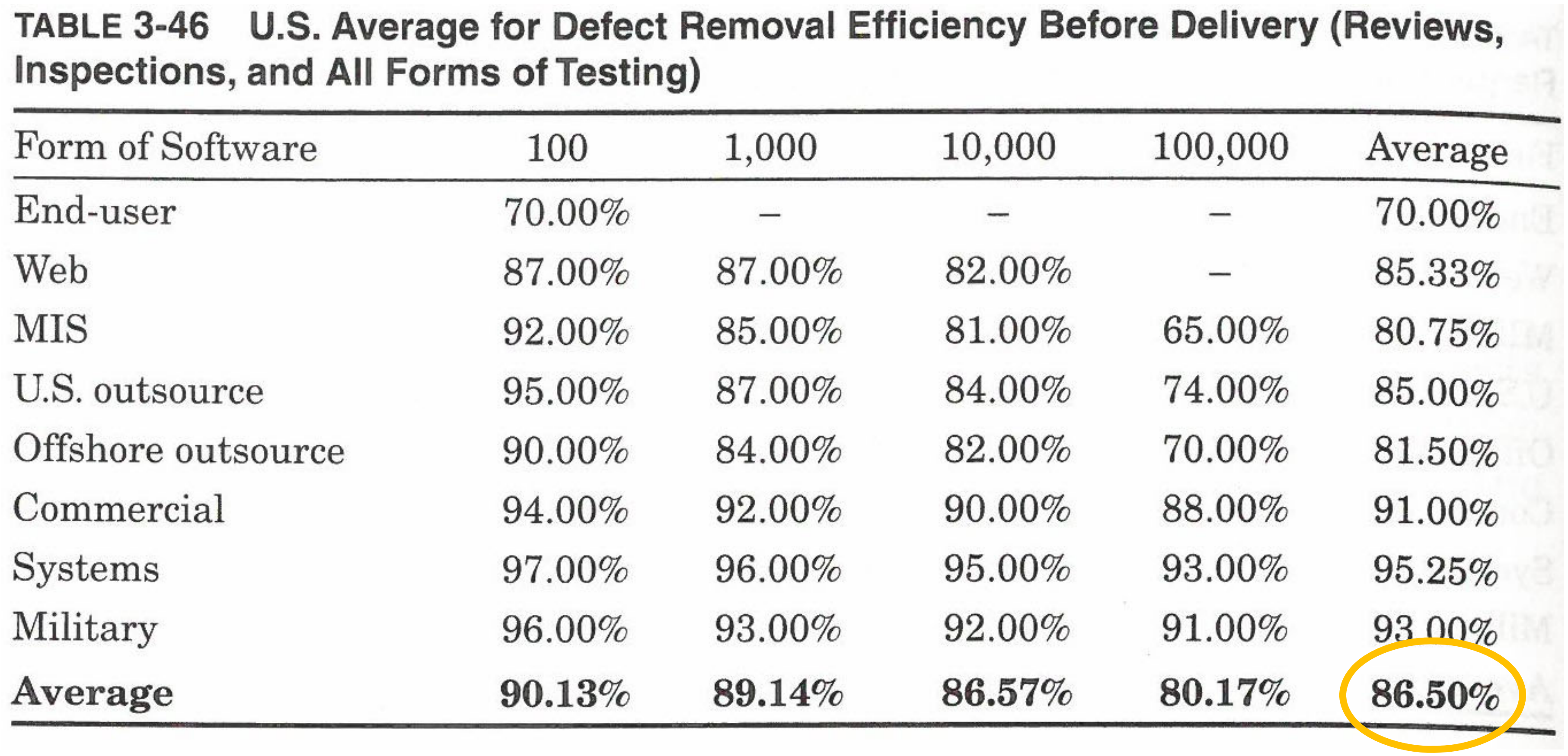


TABLE 3-46 U.S. Average for Defect Removal Efficiency Before Delivery (Reviews, Inspections, and All Forms of Testing)

Form of Software	100	1,000	10,000	100,000	Average
End-user	70.00%	–	–	–	70.00%
Web	87.00%	87.00%	82.00%	–	85.33%
MIS	92.00%	85.00%	81.00%	65.00%	80.75%
U.S. outsource	95.00%	87.00%	84.00%	74.00%	85.00%
Offshore outsource	90.00%	84.00%	82.00%	70.00%	81.50%
Commercial	94.00%	92.00%	90.00%	88.00%	91.00%
Systems	97.00%	96.00%	95.00%	93.00%	95.25%
Military	96.00%	93.00%	92.00%	91.00%	93.00%
Average	**90.13%**	**89.14%**	**86.57%**	**80.17%**	**86.50%**

Number of Delivered Defects *(Jones 2008)*

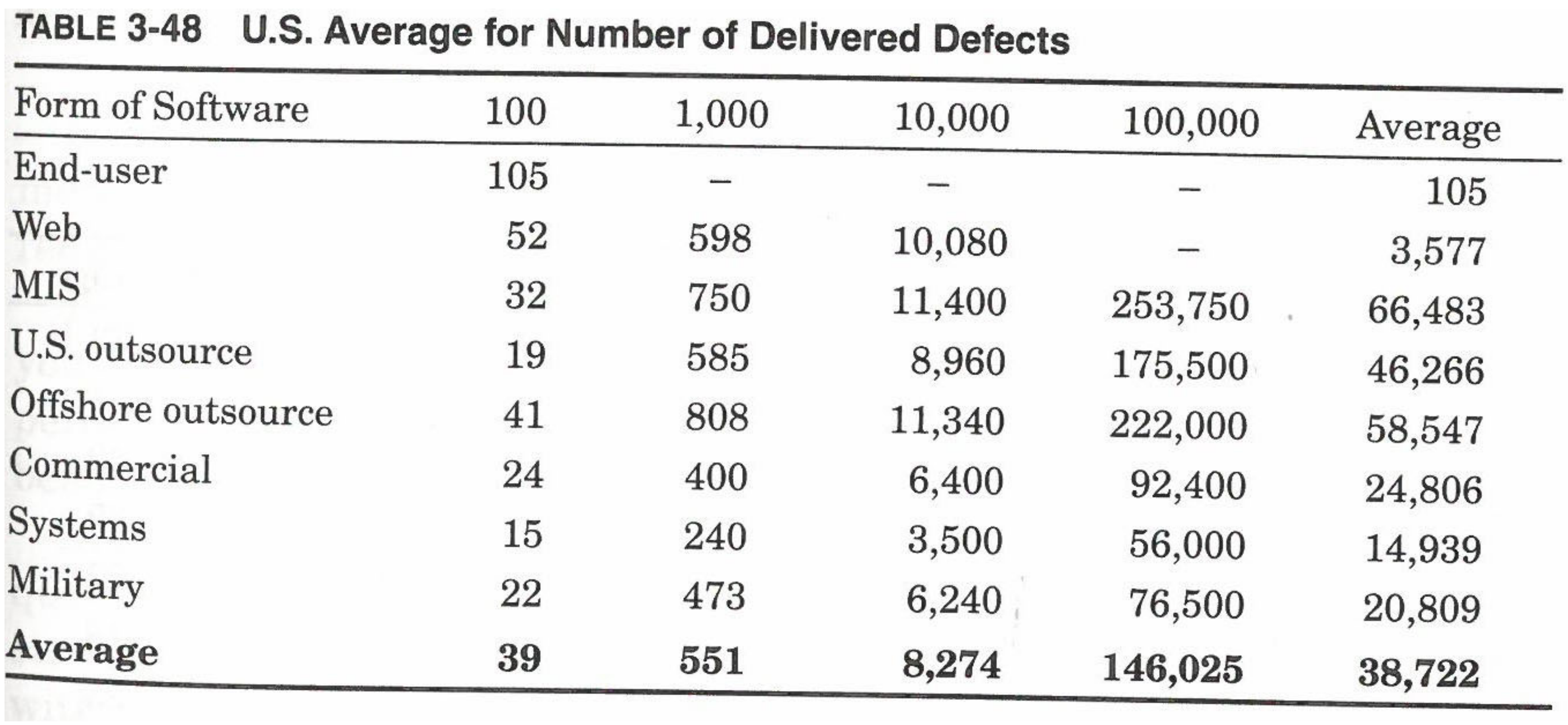
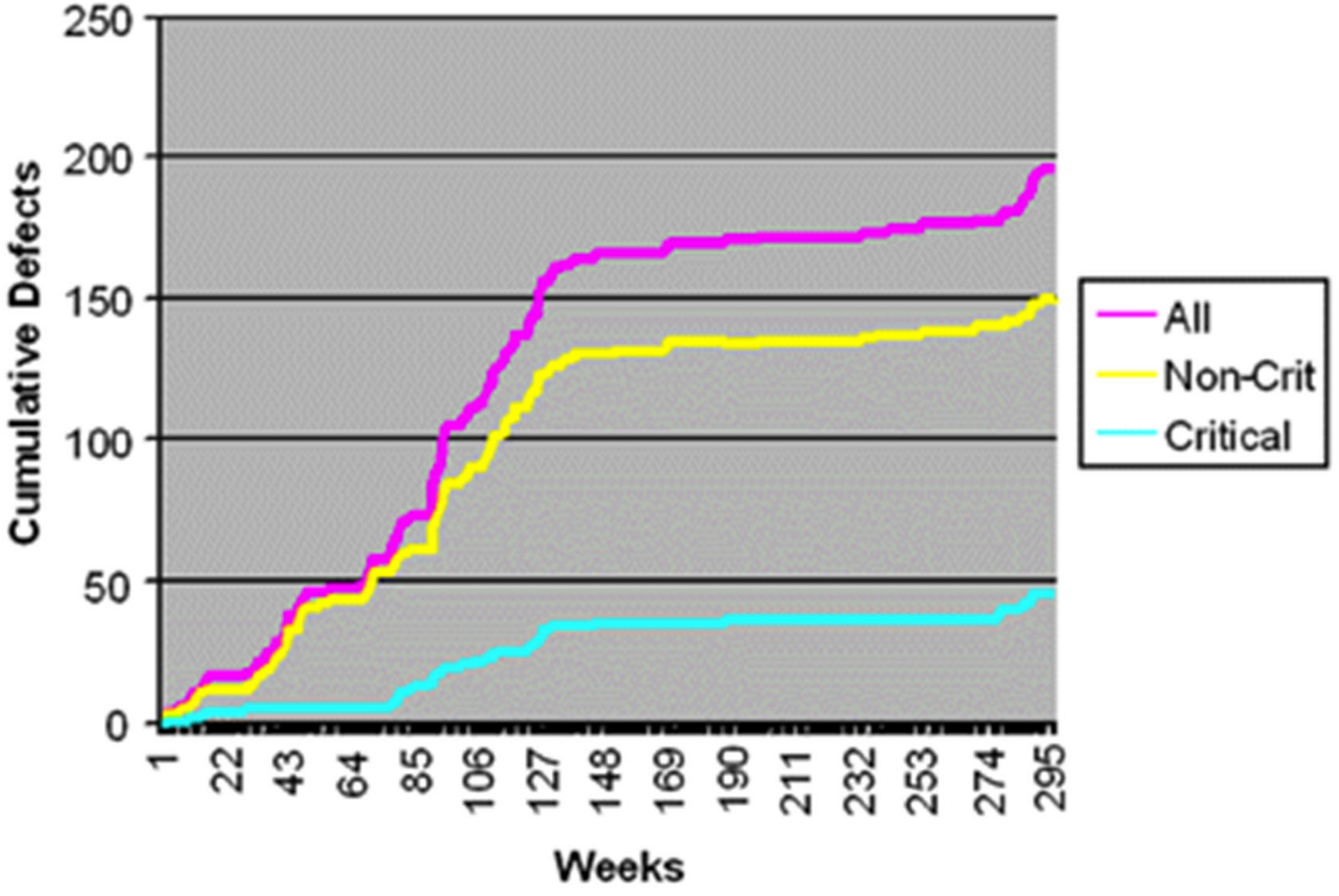


TABLE 3-48 U.S. Average for Number of Delivered Defects

Form of Software	100	1,000	10,000	100,000	Average
End-user	105	–	–	–	105
Web	52	598	10,080	–	3,577
MIS	32	750	11,400	253,750	66,483
U.S. outsource	19	585	8,960	175,500	46,266
Offshore outsource	41	808	11,340	222,000	58,547
Commercial	24	400	6,400	92,400	24,806
Systems	15	240	3,500	56,000	14,939
Military	22	473	6,240	76,500	20,809
Average	**39**	**551**	**8,274**	**146,025**	**38,722**

Galileo System Test Defects



Galileo Testing

Mission: Jupiter Orbit
Launched: October 1989
Size of software: 22 KSLOC

Only one critical defect was found in the first 10 weeks of system testing.

- **five critical defects were found in the first 76 weeks of testing**
- **after nearly six years of testing, three critical defects were found in eight weeks**

Testing shows the presence of bugs, not the absence of bugs…

Subtle Distinctions

A person makes a mistake

Which becomes a fault (bug, defect)

Resulting at execution in a failure

Which is wrong with some degree of error

From the IEEE definition of fault tolerance…

Testing

Testing: **activity in which a system or component is executed**

- **under specified conditions,**
- **the results are observed or recorded, and**
- **an evaluation is made of some aspect of the system or component.**
    - ***IEEE Std 24765: 2010***

Testing environment (known, controlled, specified)

Program inputs

Program outputs compared with expected outcomes

Input Domain

The set of all possible inputs to a program P is known as the input domain, or input space, of P.

- **includes both valid and invalid inputs**

Testing a program on all possible inputs is known as exhaustive testing.

- **for a single 32-bit integer input, that is 2^{32} test cases**

A program is considered correct if it behaves as expected on each element of its input domain.

Test Case

A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.

- **IEEE 24765**

A pair consisting of test data to be <u>input</u> to the program and the <u>expected output</u>.

- **Mathur, pg 18**
- **do we usually write the expected output down as part of our test case?**

Test set – a collection of test cases.

Test Set and Test Procedure

Test set ***(IEEE 29119-1:2013)***

- **set of one or more test cases with a common constraint on their execution**
  - **EXAMPLE: A specific test environment, specialized domain knowledge or specific purpose.**

Test procedure ***(IEEE 29119-1:2013)***

- **sequence of test cases in execution order, and any associated actions that may be required to set up the initial preconditions and any wrap up activities post execution**
  - **Note 1 to entry: Test procedures include detailed instructions for how to run a set of one or more test cases selected to be run consecutively, including set up of common preconditions, and providing input and evaluating the actual result for each included test case.**

Functional (Black-Box) Testing

Functional testing

- **testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions**
- **testing conducted to evaluate the compliance of a system or component with specified functional requirements (IEEE 24765)**
  - **Synonymous with: specification-based, closed-box testing**

Specification-based testing

- **testing in which the principal test basis is the external inputs and outputs of the test item, commonly based on a specification, rather than its implementation in source code or executable software *(IEEE 29119-1:2013)***

Black-Box Measures

Have you covered all the requirements with at least one test case?

- **prerequisite: a requirements “specification” with clearly stated and labeled requirements**
- **how would “user stories” (from agile methods) fit with this prerequisite?**

Measure – percentage of requirements covered

Have you covered all combinations of (independent) requirements?

Other black-box testing measures:

- **percentage of equivalence classes covered**
- **percentage of boundary values covered**

Structural (White-Box) Testing

Structural testing – Testing that takes into account the internal mechanism of a system or component.

- **aka white-box testing, glass-box testing**

NOTE Types include branch testing, path testing, statement testing.

- **Branch testing – Testing designed to execute each outcome of each decision point in a computer program.**
- **Path testing – Testing designed to execute all or selected paths through a computer program.**
- **Statement testing – Testing designed to execute each statement of a computer program.**

IEEE 24765

White-Box Testing Measures

Have you covered every statement in the program with at least one test case?

- **don’t need to include syntactical markers, e.g., else, {, }, end**

Have you covered both branches of every decision in the program with at least one test case?

- **decisions include loops, ifs, switches**

Have you covered both branches of every condition in the program with at least one test case?

- **multiple conditions in compound predicates: AND, OR, …**

Measure – percentage of statements, decisions, conditions covered

Test Coverage Example

```
// Program P1
integer x, y;
input (x, y);
if (x > 0 && y > 0)
{
   y = y / x;
}
else
{
   y = x ** 2;
}
output (x, y);
end;
```

What is the statement coverage for $t_1 = (1, 2)$?

- **do not count syntactical markers, including comments, {, }, else, end**

The decision coverage?

The condition coverage?

```
~~1)    // Program P1~~
2)    integer x, y;
3)    input (x, y);
4)    if (x > 0 && y > 0)
~~5)    {~~
6)        y = y / x;
~~7)    }~~
~~8)    else~~
~~9)    {~~
10)       y = x ** 2;
~~11)   }~~
12)   output (x, y);
~~13)   end;~~
```

Statement coverage for T = {t_1 = (1, 2)}?

- Domain: {2, 3, 4, 6, 10, 12}
- t_1 traverses 2, 3, 4, 6, 12
- Statement Coverage = 5 / 6

Decision coverage for t_1?

- Domain: {4) if (x > 0 && y > 0)}
- t_1 traverses the true branch
- Decision Coverage = 0 / 1

Condition coverage for t_1?

- Domain: {4) x > 0, 4) y > 0}
- t_1 traverses the true branch of the two conditions
- Condition Coverage = 0 / 2

Enhancing Test Coverage

To achieve 100% statement, decision, and condition coverage, what test cases would you need to add to T?

For statement coverage, need to cover statement 10.

For decision coverage, need to cover the false branch of the decision at line 4.

For condition coverage, need to cover the false branches for x>0, y>0 in line 4.

```
~~1)    // Program P1~~
2)    integer x, y;
3)    input (x, y);
4)    if (x > 0 && y > 0)
~~5)    {~~
6)        y = y / x;
~~7)    }~~
~~8)    else~~
~~9)    {~~
10)       y = x ** 2;
~~11)   }~~
12)   output (x, y);
~~13)   end;~~
```

Statement coverage: add t_2 = (-1, 2) to t_1 = (1,2)

- **t_2 traverses 2, 3, 4, 10, 12**
- **Statement Coverage = 6 / 6**

Decision coverage:

- **t_2 traverses the false branch of the decision at line 4**
- **Decision Coverage = 1 / 1**

Condition coverage:

- **Conditions: 4) x > 0; 4) y > 0**
- **t_1, t_2 traverse the true and false branches of x>0**
- **t_1, t_2 traverse only the true branch of y>0**
- **add t_3 = (1,-2) to traverse the false branch of y>0**
- **Condition Coverage for T_2 = {t_1, t_2, t_3} = 2 / 2**

Questions About Test Coverage

Could we have generated a smaller (minimal) test set to have statement, decision, and condition coverage?

Does statement coverage guarantee decision coverage?

Does decision coverage guarantee statement coverage?

Does condition coverage guarantee decision coverage?

There are issues not discussed yet: infeasible decisions, dead code, short-circuit evaluation, etc.

```
~~1)    // Program P1~~
2)    integer x, y;
3)    input (x, y);
4)    if (x > 0 && y > 0)
~~5)    {~~
6)        y = y / x;
~~7)    }~~
~~8)    else~~
~~9)    {~~
10)       y = x ** 2;
~~11)   }~~
12)   output (x, y);
~~13)   end;~~
```

Could we have generated a smaller (minimal) test set to have statement, decision, and condition coverage?

T_2 = {t_1=<1,2>, t_2=<-1, 2>, t_3=<1,-2>}

Consider

T_3 = {t_1=<1,2>, t_2=<-1, -2>}

t_1: 2, 3, 4(tt), 6, 12
t_2: 2, 3, 4(ff), 10, 12

C_S = 6/6 = 100%
C_D = 1/1 = 100%
C_C = 2/2 = 100%

Does statement coverage guarantee decision coverage?

```
1) input (x);
2) if (x > 0)
3)    x = 2 * x;
4) output x;
```

Consider the coverage for T = {t_1=<1, EO=2>}

t_1: 1, 2(t), 3, 4

C_S = 4/4 = 100%

C_D = 0/1 = 0%

- **did not cover the false branch for line 2**

Does decision coverage guarantee statement coverage?

```
1) input (x);
2) if (x > 0)
3)    x = 2 * x;
4) output x;
```

Consider the coverage for T = {t_1=<1, EO=2>, t_2=<-1, EO=-1>}

t_1: 1, 2(t), 3, 4
t_2: 1, 2(f), 4

C_S = 4/4 = 100%

C_D = 1/1 = 100%

Intuitively, it seems that decision coverage would "subsume" statement coverage.

Does condition coverage guarantee decision coverage?

Consider coverage for
$T = \{t_1=<1, -1, EO=(2,-1)>,$
$t_2=<-1, 1, EO=(-2,1)>\}$

$C_C = 2 / 2 = 100\%$

- true/false for x>0
- true/false for y>0

$C_D = 0 / 1 = 0\%$

- true OR false = true

```
1) input (x, y);
2) if (x > 0 || y > 0)
3)     x = 2 * x;
4) else
5)     x = 3 * x;
6) output (x, y);
```

Another Test Coverage Example

```
integer x, n;
input (x, n);
for (i=1,n)
{
   if (i%2==0 || x < 0)
      x = x + 1;
   else
      x = x * 2;
}
output (x, n);
end;
```

What is the statement coverage for

T = {(1, 2), (-1,3)?

- **do not count syntactical markers, including comments, {, }, else, end**

The decision coverage?

The condition coverage?

Tracing Program Execution

Tracing / desk checks / hand execution / state transition table … a useful debugging technique

- **consistency minimizes confusion**

State vector for each step

- **state is values for each variable before a step is executed**
  - **exception: initializing & updating “for” loops**
  - **initialize before executing comparison**
  - **update at closing brace or before executing comparison**

Useful addition: values of conditions and/or decisions at decision steps

- **particularly helpful when variables involved in decision change during execution**

Tracing t_1=(1, 2)

```
1)   integer x, n;
2)   input (x, n);
3)   for (i=1,n)
4)   {
5)      if (i%2==0 || x < 0)
6)         x = x + 1;
7)      else
8)         x = x * 2;
9)   }
10)  output (x, n);
11)  end;
```

Step	x	n	i	
1	u	u	u	
2	u	u	u	
3 for	1	2	1	t
5 if	1	2	1	ff=f
8	1	2	1	
3 for	2	2	2	t
5 if	2	2	2	tf=t
6	2	2	2	
7 else	2	2	2	
3 for	3	2	3	f
10	3	2	u	

Another Notation for Tracing t_1=(1, 2)

```
1)   integer x, n;
2)   input (x, n);
3)   for (i=1,n)
4)   {
5)      if (i%2==0 || x < 0)
6)         x = x + 1;
7)      else
8)         x = x * 2;
9)   }
10)  output (x, n);
11)  end;
```

Step	x	n	i	
1				
2				
3	1	2	1	t
5				ff=f
8				
3	2	2	2	t
5				tf=t
6				
7	3	2	2	
3	3	2	3	f
10				
11				

Tracing t_2=(-1, 3)

```
1)   integer x, n;
2)   input (x, n);
3)   for (i=1,n)
4)   {
5)      if (i%2==0 || x < 0)
6)         x = x + 1;
7)      else
8)         x = x * 2;
9)   }
10)  output (x, n);
11)  end;
```

Step		x	n	i	
1		u	u	u	
2		u	u	u	
3	for	-1	3	1	t
5	if	-1	3	1	ft=t
6		-1	3	1	
7	else	0	3	1	
3	for	0	3	2	t
5	if	0	3	2	tf=t
6		0	3	2	
7	else	1	3	2	
3	for	1	3	3	t
5	if	1	3	3	ff=f
8		1	3	3	
3	for	2	3	4	f
10		2	3	u	
11		2	3	u	

```
1)    integer x, n;
2)    input (x, n);
3)    for (i=1,n)
~~4)    {~~
5)       if (i%2==0 || x < 0)
6)          x = x + 1;
~~7)       else~~
8)          x = x * 2;
~~9)    }~~
10)   output (x, n);
~~11)  end;~~
```

What is the statement coverage for

T = {(1, 2), (-1,3)?

- **do not count syntactical markers, including comments, {, }, else, end**

Domain: {1, 2, 3, 5, 6, 8, 10}

t_1: **1, 2, 3(t), 5(ff), 8,**
3(t), 5(tf), 6, 3(f), 10

t_2: **1, 2, 3(t), 5(ft), 6,**
3(t), 5(tf), 6,
3(t), 5(ff), 8, 3(f), 10

{~~1~~, ~~2~~, ~~3~~, ~~5~~, ~~6~~, ~~8~~, ~~10~~}

C_S = 7 / 7 = 100%

```
1)    integer x, n;
2)    input (x, n);
3)    for (i=1,n)
4)    {
5)       if (i%2==0 || x < 0)
6)          x = x + 1;
7)       else
8)          x = x * 2;
9)    }
10)   output (x, n);
11)   end;
```

What is the decision coverage for
T = {(1, 2), (-1,3)?

Domain: {3, 5}

t_1:	**3(t), 3(t), 3(f)**
	5(f), 5(t)
t_2:	**3(t), 3(t), 3(t), 3(f)**
	5(t), 5(t), 5(f)

Decision 3 is covered t & f
Decision 5 is covered t & f

C_D = 2 / 2 = 100%

```
1)   integer x, n;
2)   input (x, n);
3)   for (i=1,n)
4)   {
5)      if (i%2==0 || x < 0)
6)         x = x + 1;
7)      else
8)         x = x * 2;
9)   }
10)  output (x, n);
11)  end;
```

What is the condition coverage for T = {(1, 2), (-1,3)?

Domain: {3) i<=n,
5) i%2==0,
5) x<0}

t_1: 3(t), 3(t), 3(f)
5(ff), 5(tf)
t_2: 3(t), 3(t), 3(t), 3(f)
5(ft), 5(tf), 5(ff)

Condition 3) i≤n is covered
Condition 5) i%2==0 is covered
Condition 5) x<0 is covered

C_C = 3 / 3 = 100%

Levels of Testing

Unit testing

Integration testing

Product (system) testing

Acceptance testing

Alpha release

Beta release

Unit vs Integration Testing

Unit testing

- **testing of individual routines and modules by the developer or an independent tester**
- **a test of individual programs or modules in order to ensure that there are no analysis or programming errors**

Integration testing

- **testing in which software components, hardware components, or both are combined and tested to evaluate the interaction among them**

(ISO/IEC/IEEE 24765)

System vs Acceptance Testing

System testing

- **testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements**

Acceptance testing

- **testing conducted to determine whether a system satisfies its acceptance criteria and <u>to enable the customer to determine whether to accept the system</u>**
- **formal testing conducted to enable a user, customer, or other authorized entity to determine whether to accept a system or component.**

(ISO/IEC/IEEE 24765)

Regression Testing

Selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements.

Testing required to determine that a change to a system component has not adversely affected functionality, reliability or performance and has not introduced additional defects.

IEEE 24765

Developer and Tester as Two Roles

Distinct but complementary roles

- **developers have a conflict of interest in testing**

A developer may assume the role of tester at some points in the software process

- **developers frequently do unit testing**

Testers may be “black hats”

- **an independent testing group**
- **distinct testing skills: white box and black box**
- **make no assumptions about the software**
- **actively try to break the software**

Peer Reviews

A review of a software work product, following defined procedures, by peers of the producers of the product for the purpose of identifying defects and improvements.

- **Software CMM**

Review of work products performed by peers during development of the work products to identify defects for removal.

- **IEEE 24765**

Walkthroughs

Structured walkthrough – A systematic examination of the requirements, design, or implementation of a system, or any part of it, by qualified personnel.

IEEE 24765

See Weinberg's The Psychology of Computer Programming for a discussion of walkthroughs and egoless programming.

Inspections

1) A visual examination of a software product to detect and identify software anomalies, including errors and deviations from standards and specifications.
2) A static analysis technique that relies on visual examination of development products to detect errors, violations of development standards, and other problems
3) Technique examining or measuring to verify whether an activity, component, product, result, or service conforms to specified requirements.

NOTE Inspections are peer examinations led by impartial facilitators who are trained in inspection techniques. Determination of remedial or investigative action for an anomaly is a mandatory element of a software inspection, although the solution should not be determined in the inspection meeting. Types include code inspection; design inspection

IEEE 24765

Formal Methods

A technique for expressing requirements in a manner that allows the requirements to be studied mathematically.

Formal methods allow sets of requirements to be examined for completeness, consistency, and equivalency to another requirement set.

Formal methods result in formal specifications.

EIA 731.1: 2002 (Systems Engineering Capability Model)

- **definition focused on formal specifications but not proofs of correctness**

Quality Assurance

All the planned and systematic activities implemented within the quality system, and demonstrated as needed, to provide adequate confidence that an entity will fulfill requirements for quality. (ISO 12207)

Note 1 - There are both internal and external purposes for quality assurance: a) Internal quality assurance: within an organization, quality assurance provides confidence to management; b) External quality assurance: in contractual situations, quality assurance provides confidence to the customer or others.

Note 2 - Some quality control and quality assurance actions are interrelated.

Note 3 - Unless requirements for quality fully reflect the needs of the user, quality assurance may not provide adequate confidence.

Summary – Things to Remember

Purpose of testing

Testing vs correctness

Fault tolerance terminology – subtle distinctions

Black box vs white box testing

Statement, decision, condition coverage

Kinds of testing – unit, integration, system, acceptance, regression

Conflict of interest for developers

Questions and Answers