

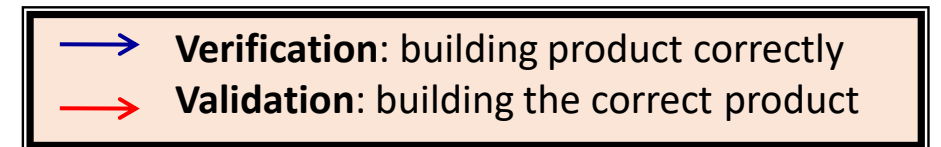
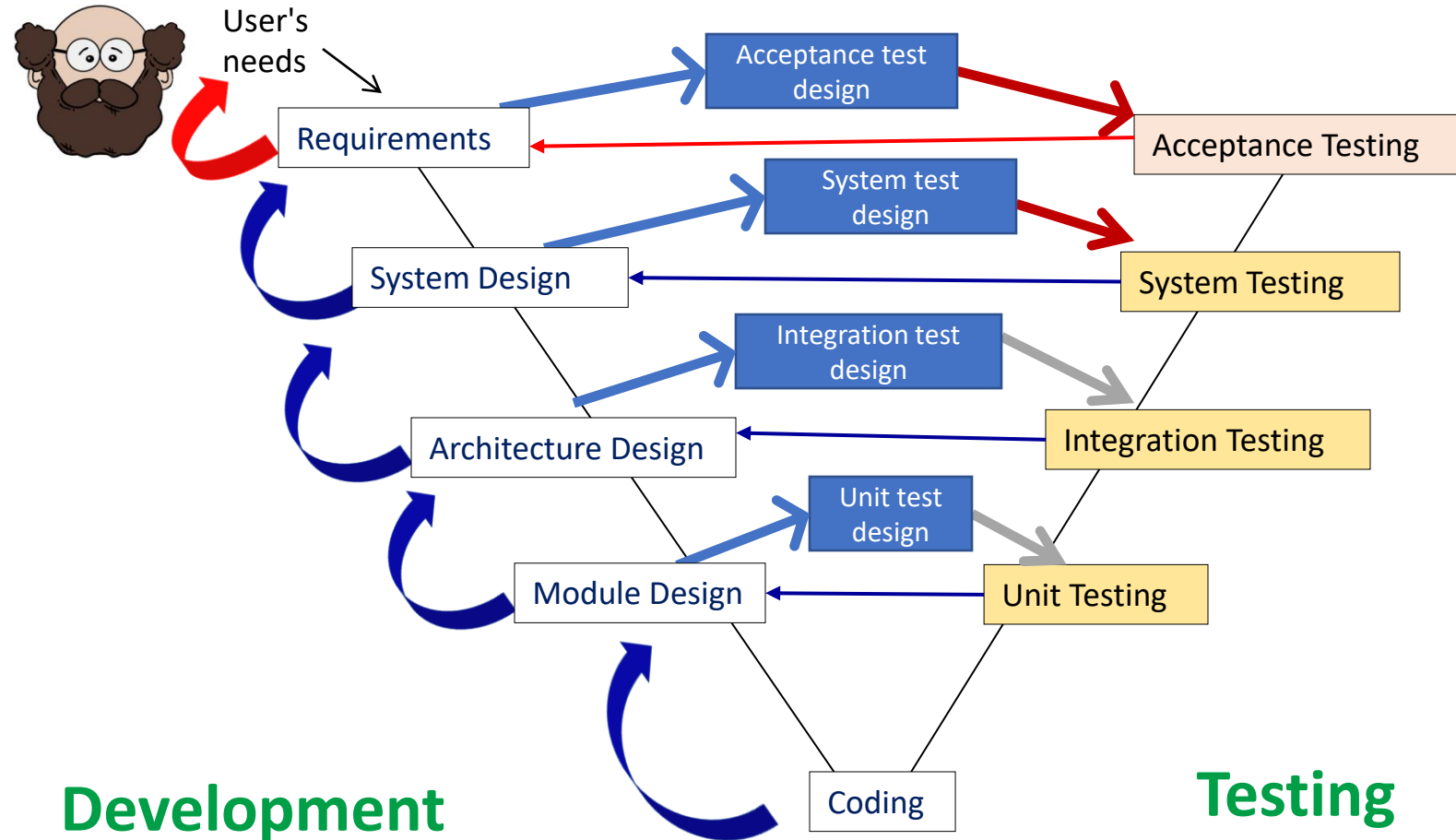
Software Testing (I)

What is software testing?

- Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test



The V Model



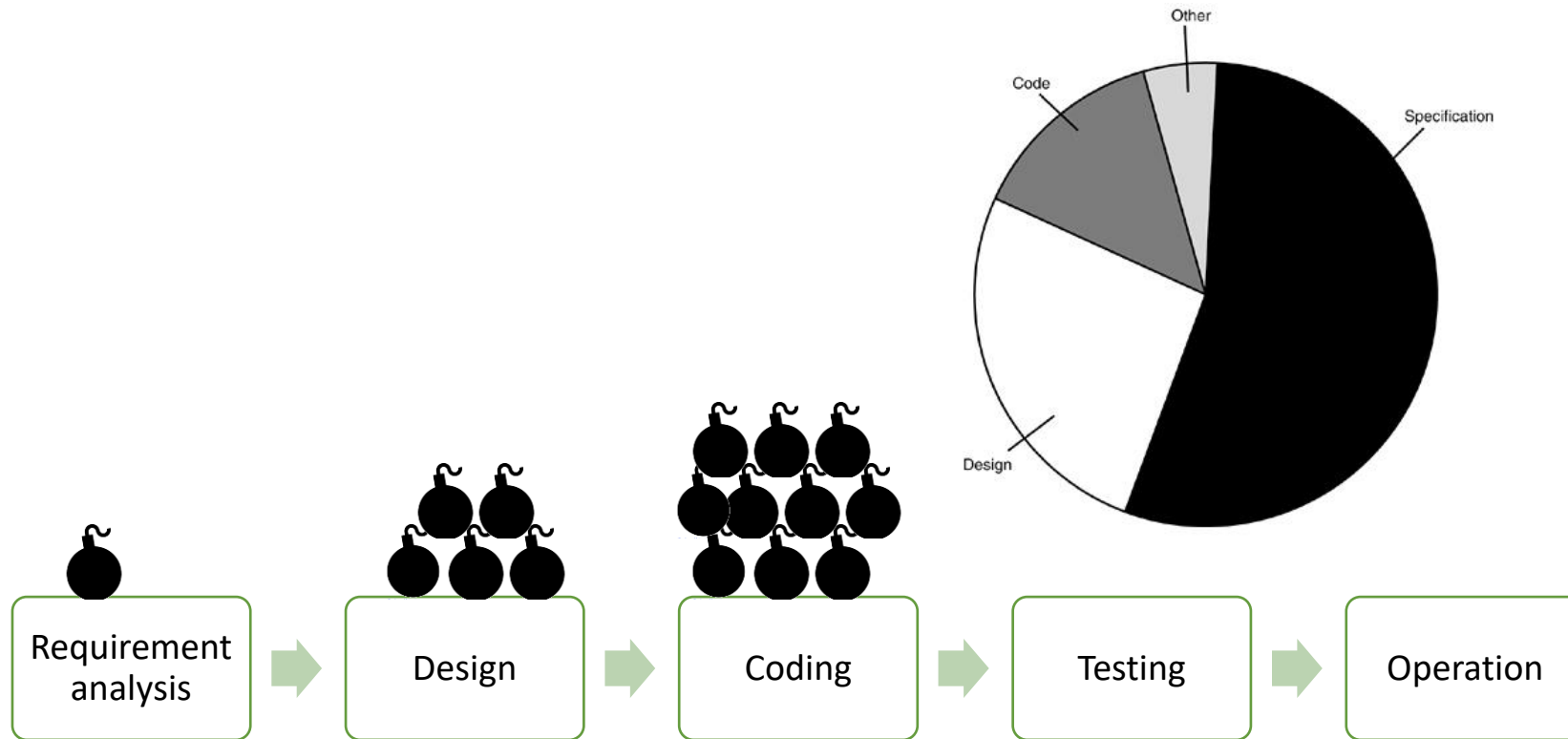
Dynamic testing vs. static testing

- Dynamic testing
 - Detect the presence of defects by running the program with a given set of test cases
- Static testing
 - Refers to testing something that's not running
 - E.g. Review the code and documentations (e.g. requirement specification, design, user manual, test plan and test cases) to locate errors

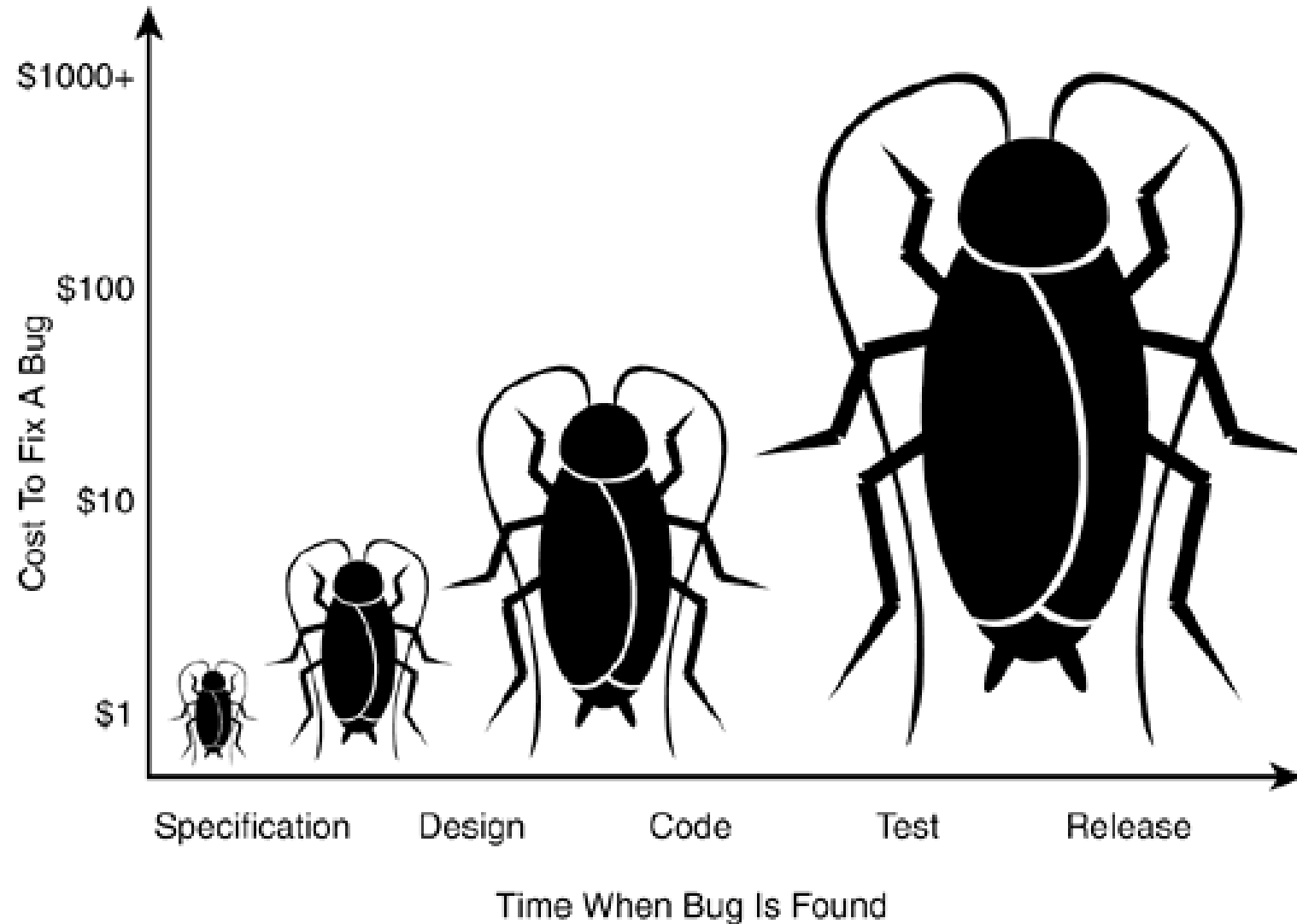
1/5 of the bugs (defects) are found during dynamic testing, remaining are found in code review!

Sources of software defect

- Numerous studies have been performed to show that the main cause of software defect is specification and design

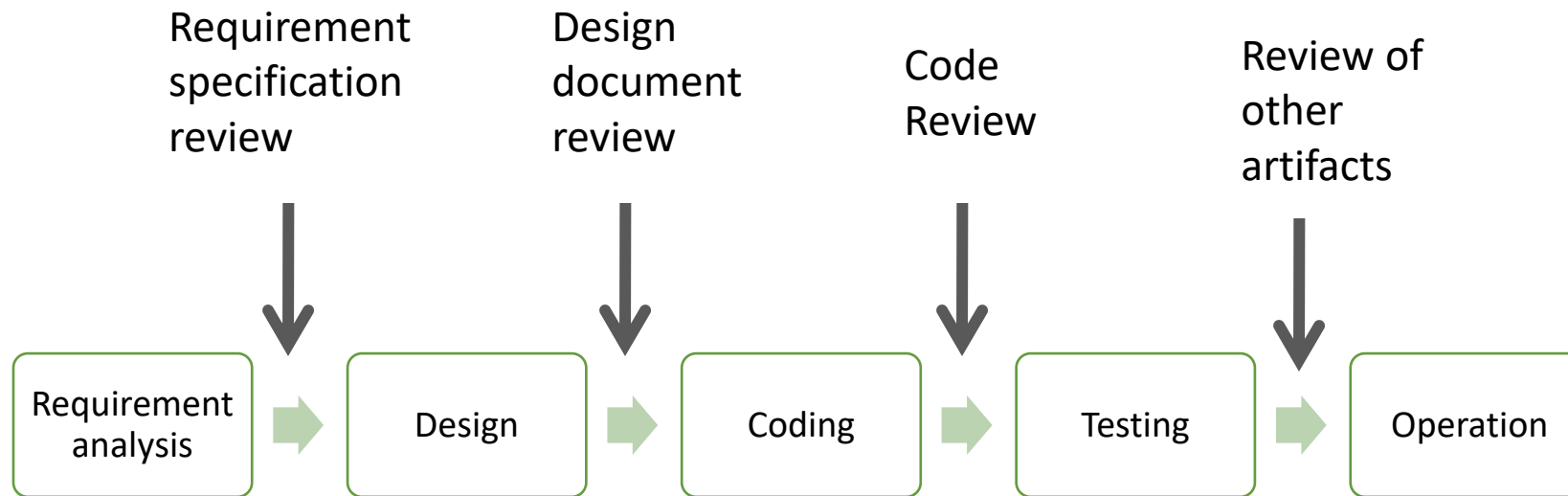


The cost to fix bugs

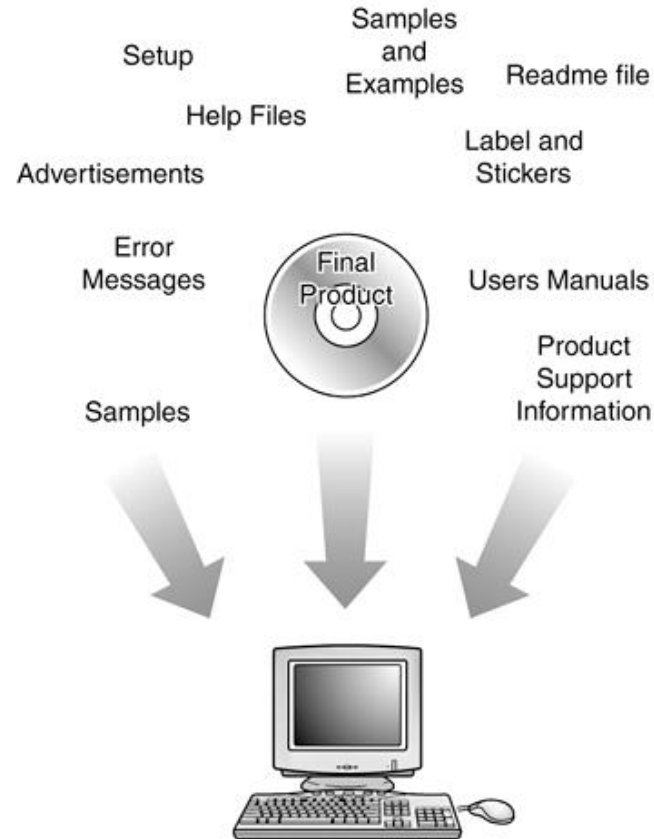


Static testing

- Static testing refers to testing something that's not running
 - Examples of activities
 - Review the code and documentations (e.g. requirement specification, design, user manual, test plan and test cases) to locate errors



What artifacts to be reviewed?



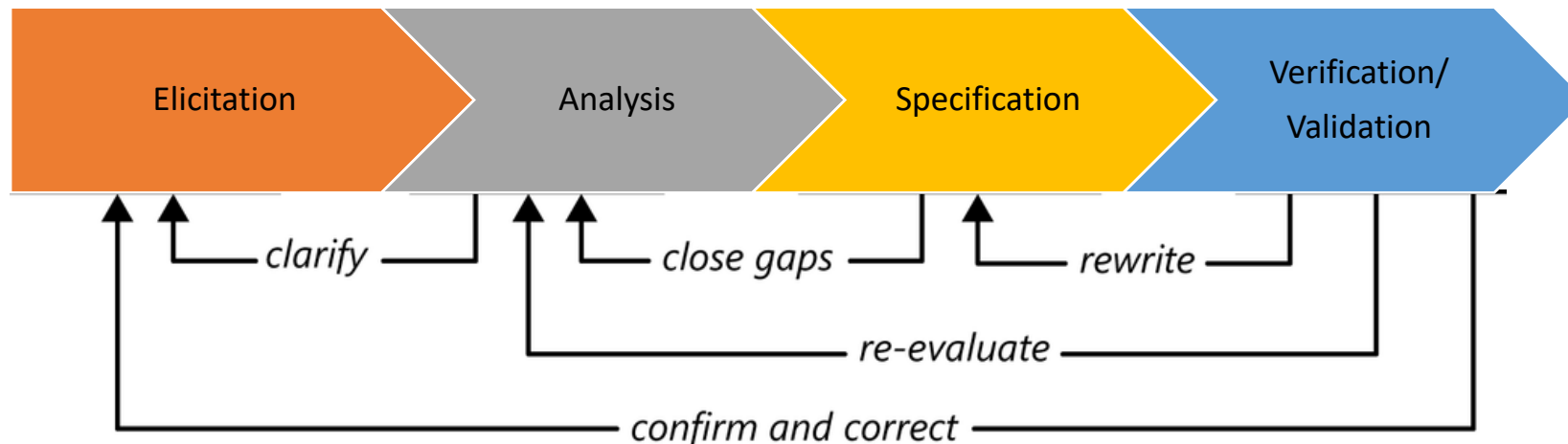
Non-software pieces may also have bugs and should be tested

Requirements development phases

- Elicitation
 - Working with individuals who represent each user class to understand their functionality needs and their quality expectations.
 - Activities: interviews, workshops, document analysis, prototyping, etc
- Analysis
 - Decomposing high-level requirements into an appropriate level of detail
 - Negotiating implementation priorities



- Specification
 - Translating the collected user needs into written requirements and diagrams suitable for comprehension, review, and use by their intended audiences.
- Validation/Verification
 - Reviewing the documented requirements to correct any problems
 - Developing acceptance tests and criteria to confirm that a product based on the requirements would meet customer needs and achieve the business objectives.



Characteristics of a good Software Requirements Specifications (SRS)

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable

Ref: IEE Std 830-1998 - IEEE Recommended Practice for Software Requirements Specifications

Correct

- An SRS is correct if, and only if, every requirement stated is one that the software shall meet.
- Consistent with other project documentation and other applicable standards
- The customer or user can determine if the SRS correctly reflects the actual needs.

Unambiguous

- Ambiguous requirements may be interpreted in different ways by developers and users

Ambiguous terms	Ways to improve them
acceptable, adequate	Define what constitutes acceptability and how the system can judge this.
and/or	Specify whether you mean "and," "or," or "any combination of" so the reader doesn't have to guess.
as much as practicable	Don't leave it up to the developers to determine what's practicable. Make it a TBD and set a date to find out.
at least, at a minimum, not more than, not to exceed	Specify the minimum and maximum acceptable values.
best, greatest, most	State what level of achievement is desired and the minimum acceptable level of achievement.
between, from X to Y	Define whether the end points are included in the range.
depends on	Describe the nature of the dependency. Does another system provide input to this system, must other software be installed before your software can run, or does your system rely on another to perform some calculations or provide other services?
efficient	Define how efficiently the system uses resources, how quickly it performs specific operations, or how quickly users can perform certain tasks with the system.
fast, quick, rapid	Specify the minimum acceptable time in which the system performs some action.
flexible, versatile	Describe the ways in which the system must be able to adapt to changing operating conditions, platforms, or business needs.
i.e., e.g.	Many people are unclear about which of these means "that is" (i.e., meaning that the full list of items follows) and which means "for example" (e.g., meaning that just some examples follow). Use words in your native language, not confusing Latin abbreviations.
improved, better, faster, superior, higher quality	Quantify how much better or faster constitutes adequate improvement in a specific functional area or quality aspect.
including, including but not limited to, and so on, etc., such as, for instance	List all possible values or functions, not just examples, or refer the reader to the location of the full list. Otherwise, different readers might have different interpretations of what the whole set of items being referred to contains or where the list stops.
in most cases, generally, usually, almost always	Clarify when the stated conditions or scenarios do not apply and what happens then. Describe how either the user or the system can distinguish one case from the other.
match, equals, agree, the same	Define whether a text comparison is case sensitive and whether it means the phrase "contains," "starts with," or is "exact." For real numbers, specify the degree of precision in the comparison.
maximize, minimize, optimize	State the maximum and minimum acceptable values of some parameter.
normally, ideally	Identify abnormal or non-ideal conditions and describe how the system should behave in those situations.
optionally	Clarify whether this means a developer choice, a system choice, or a user choice.
probably, ought to, should	Will it or won't it?
reasonable, when necessary, where appropriate, if possible, as applicable	Explain how either the developer or the user can make this judgment.
robust	Define how the system is to handle exceptions and respond to unexpected operating conditions.

Ambiguous terms	Ways to improve them
seamless, transparent, graceful	What does "seamless" or "graceful" mean to the user? Translate the user's expectations into specific observable product characteristics.
several, some, many, few, multiple, numerous	State how many, or provide the minimum and maximum bounds of a range.
shouldn't, won't	Try to state requirements as positives, describing what the system will do.
state-of-the-art	Define what this phrase means to the stakeholder.
sufficient	Specify how much of something constitutes sufficiency.
support, enable	Define exactly what functions the system will perform that constitute "supporting" some capability.
user-friendly, simple, easy	Describe system characteristics that will satisfy the customer's usage needs and usability expectations.

Ref: Software requirements (3rd edition)

Verifiable/Testable

- A requirement is verifiable if, and only if, there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement.
- Non-verifiable requirements include statements such as “works well,” “user friendly,” and “shall usually happen.”

Example: Testable non-functional requirement

The system should be easy to use by medical staff and should be organized in such a way that user errors are minimized



Medical staff shall be able to use all the system functions after four hours of training. After this training, the average number of errors made by experienced users shall not exceed two per hour of system use.

Examples of Metrics for specifying non-functional requirements

Property	Measure
Speed	<ul style="list-style-type: none">• Processed transactions/second• User/event response time
Ease of use	<ul style="list-style-type: none">• Training time
Reliability	<ul style="list-style-type: none">• Mean time to failure• Probability of unavailability• Rate of failure occurrence• Availability
Robustness	<ul style="list-style-type: none">• Time to restart after failure• Percentage of events causing failure• Probability of data corruption on failure
Portability	<ul style="list-style-type: none">• Percentage of target dependent statements

Defect checklist for reviewing requirements documents

Completeness

- ☐ Do the requirements address all known customer or system needs?
- ☐ Is any needed information missing? If so, is it identified as TBD?
- ☐ Have algorithms intrinsic to the functional requirements been defined?
- ☐ Are all external hardware, software, and communication interfaces defined?
- ☐ Is the expected behavior documented for all anticipated error conditions?
- ☐ Do the requirements provide an adequate basis for design and test?
- ☐ Is the implementation priority of each requirement included?
- ☐ Is each requirement in scope for the project, release, or iteration?

Correctness

- ☐ Do any requirements conflict with or duplicate other requirements?
- ☐ Is each requirement written in clear, concise, unambiguous, grammatically correct language?
- ☐ Is each requirement verifiable by testing, demonstration, review, or analysis?
- ☐ Are any specified error messages clear and meaningful?
- ☐ Are all requirements actually requirements, not solutions or constraints?
- ☐ Are the requirements technically feasible and implementable within known constraints?

Quality Attributes

- ☐ Are all usability, performance, security, and safety objectives properly specified?
- ☐ Are other quality attributes documented and quantified, with the acceptable trade-offs specified?
- ☐ Are the time-critical functions identified and timing criteria specified for them?
- ☐ Have internationalization and localization issues been adequately addressed?
- ☐ Are all of the quality requirements measurable?

Organization and Traceability

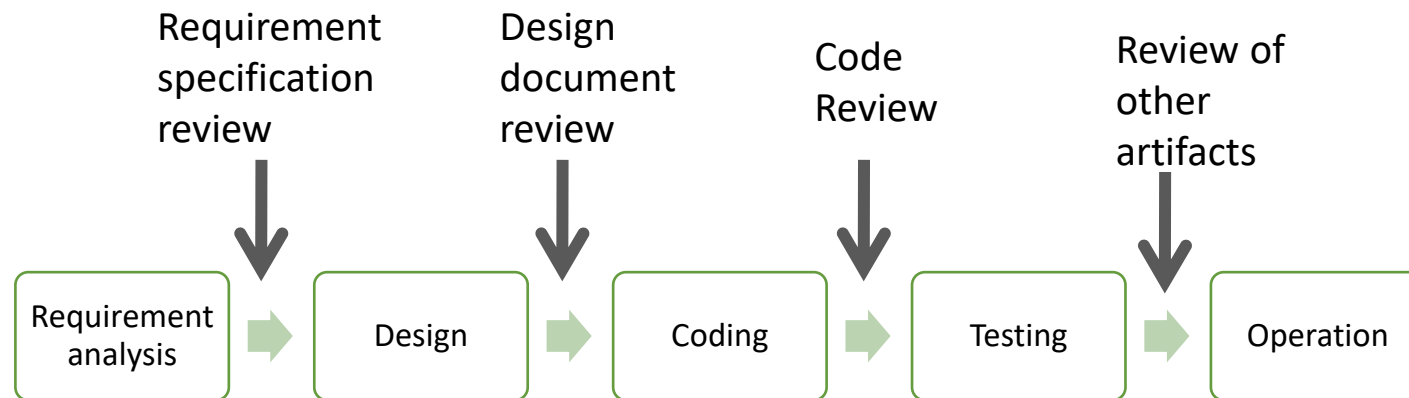
- ☐ Are the requirements organized in a logical and accessible way?
- ☐ Are all cross-references to other requirements and documents correct?
- ☐ Are all requirements written at a consistent and appropriate level of detail?
- ☐ Is each requirement uniquely and correctly labeled?
- ☐ Is each functional requirement traced back to its origin (e.g., system requirement, business rule)?

Other Issues

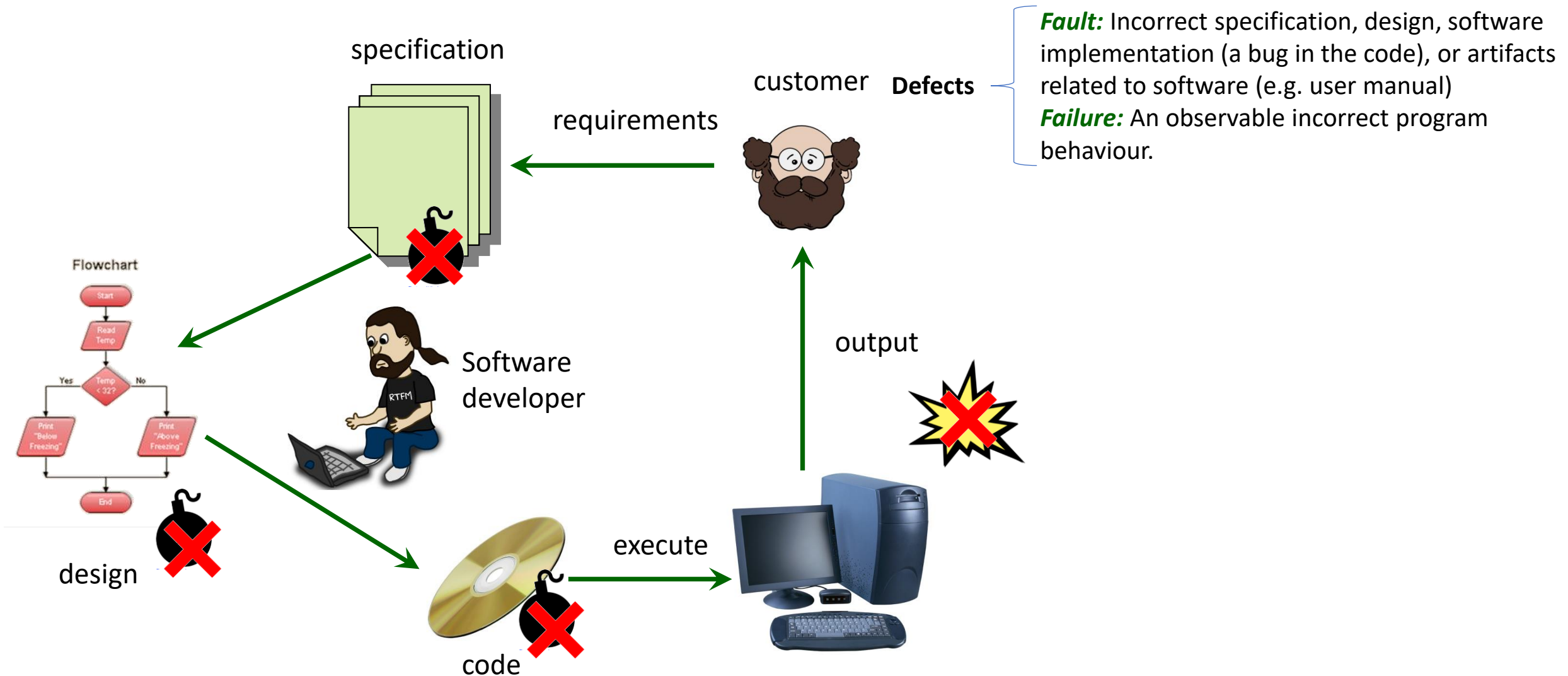
- ☐ Are any use cases or process flows missing?
- ☐ Are any alternative flows, exceptions, or other information missing from use cases?
- ☐ Are all of the business rules identified?
- ☐ Are there any missing visual models that would provide clarity or completeness?
- ☐ Are all necessary report specifications present and complete?

Early testing

- Cheaper to fix the defect if it can be found earlier during the software development process.
 - Effort spent in rework of previous phases
 - One single defect in requirement phase may generate many defects in design and code (in multiple locations)
- To fix the defects early, testing should be performed to reveal the presence of defects as early as possible



Early defect detection and removal



Results in defect reduction in later phases in software development

Defect Removal Efficiency (DRE)

- A good testing process should
 - detect the defect as early as possible
 - increase the number of defects found by tester and decrease the number of defects found by customers
- DRE measures the effectiveness of a test process or a testing phase

Effectiveness of a test process

$$DRE = \frac{\text{no. of defects detected (and removed)}}{\text{total no. of defects}}$$

- We normally use the first year after product release to quantify the total defects.
- Example:
 - Suppose 1000 defects were detected during the development, and 100 defects were reported within the first year of the system release, then the total defect count is 1100.

Exercise

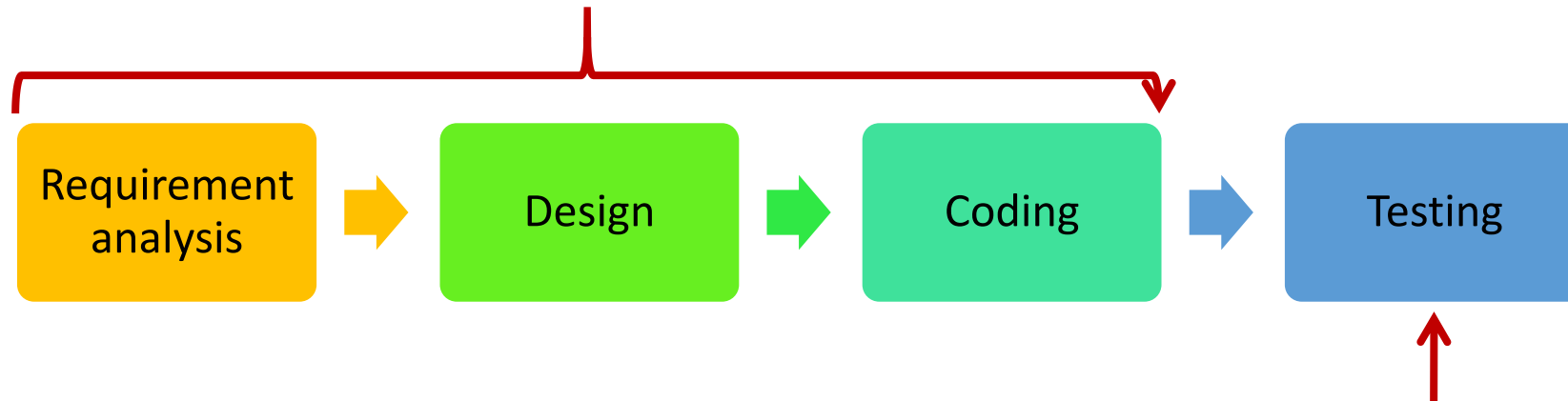
$$DRE = \frac{\text{no. of defects detected (and removed)}}{\text{total no. of defects}}$$

System	Defects found in testing	Number of defects found by users
A	30	70
B	20	30

- Which test team is more effective in finding defects?

Defect removal and injection in SDLC

These activities can inject defects, while the reviews and inspections at the end of the phase can remove defects.



The testing phase is mainly for defect removal. There is also a chance to introduce defects when the found defects are fixed incorrectly.

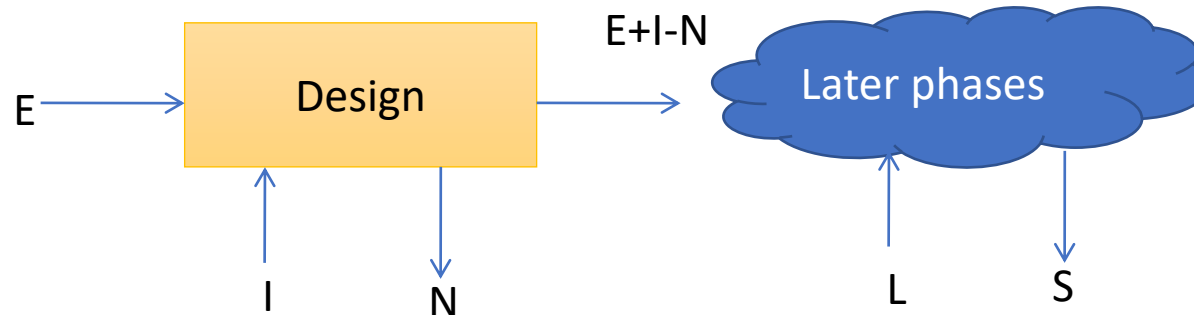
Formula for DRE (For a development phase)

- No. of defects found in design phase = 30
- No. of defects found in coding, testing, operation phases = 50
- Among the no. of defects found after the design phases, 40 of them are introduced after the design phase.

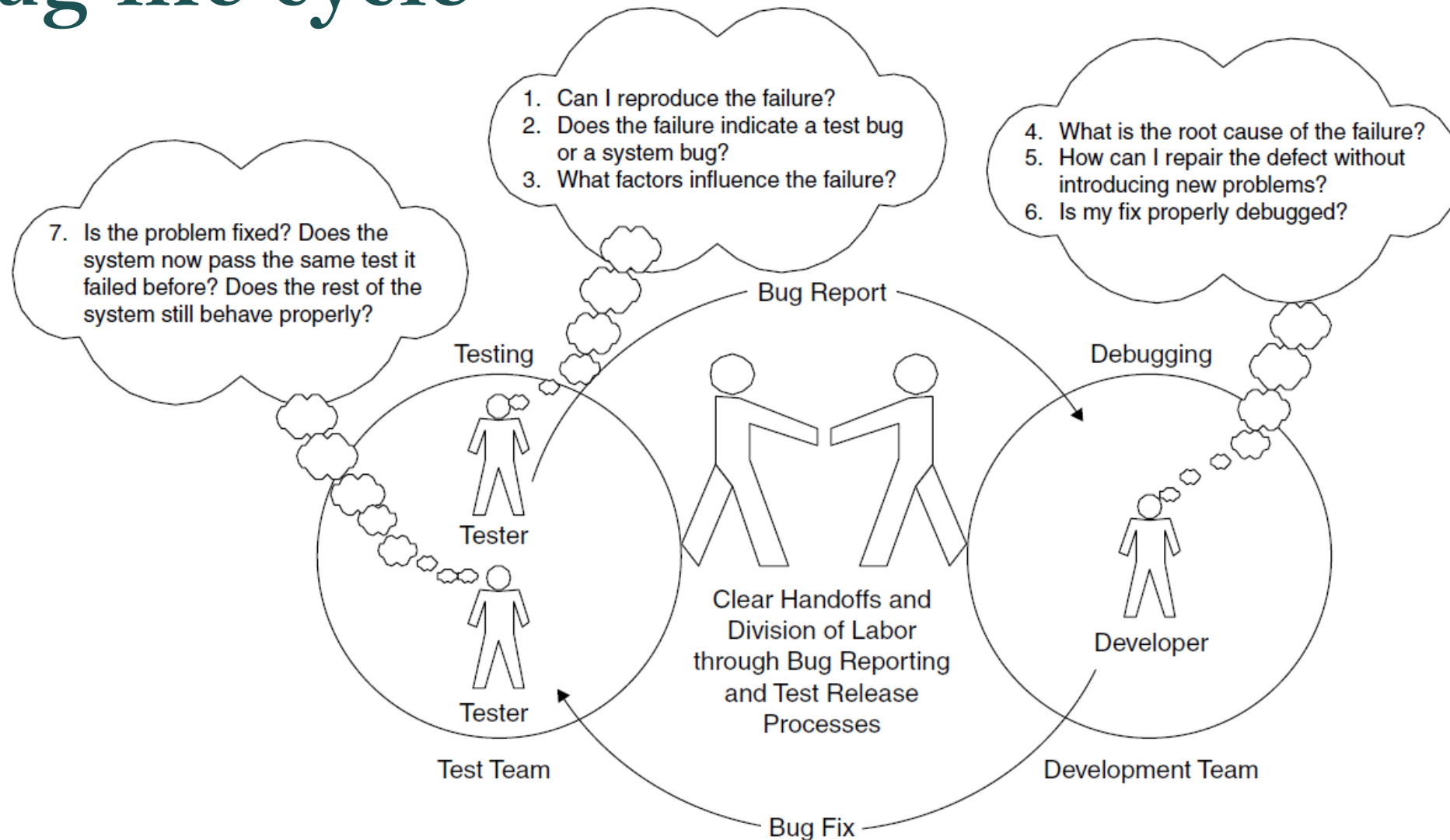
What is the DRE for the design phase?

$$\text{DRE} = N / (E + I) \times 100\%$$

- N is the number of defect removed by the development phase
- E is the number of defects existing on phase entry, and I is the number of defects

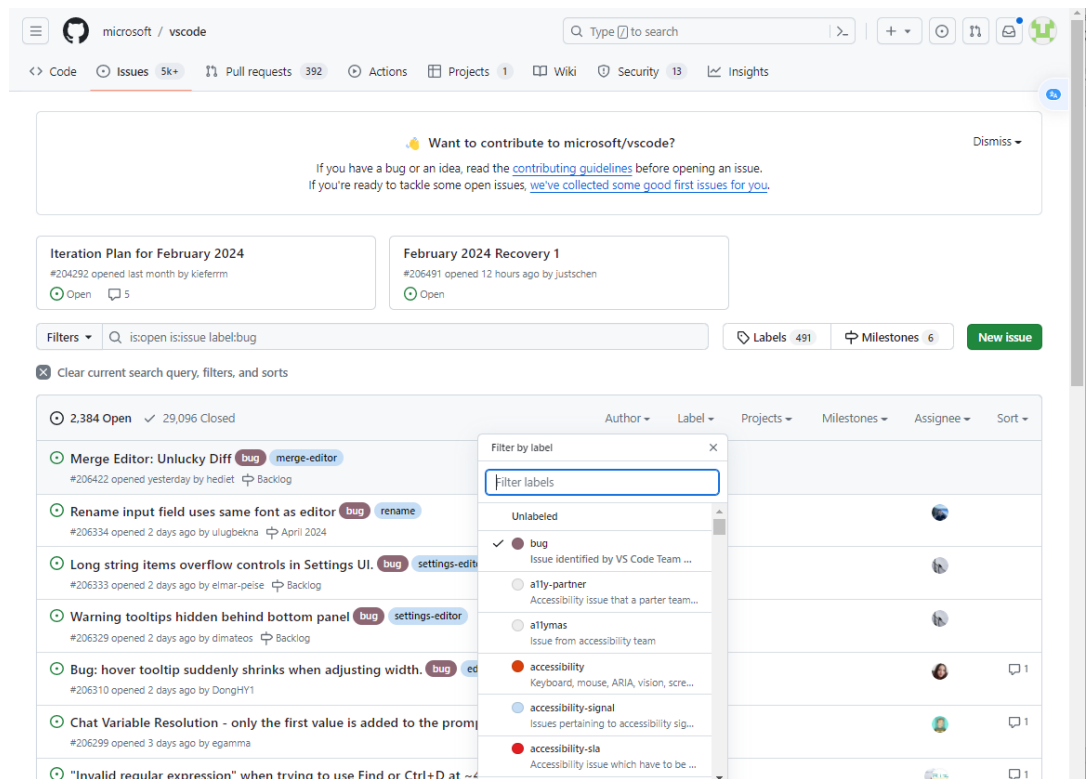


Questions, players, and handoffs in the bug life cycle

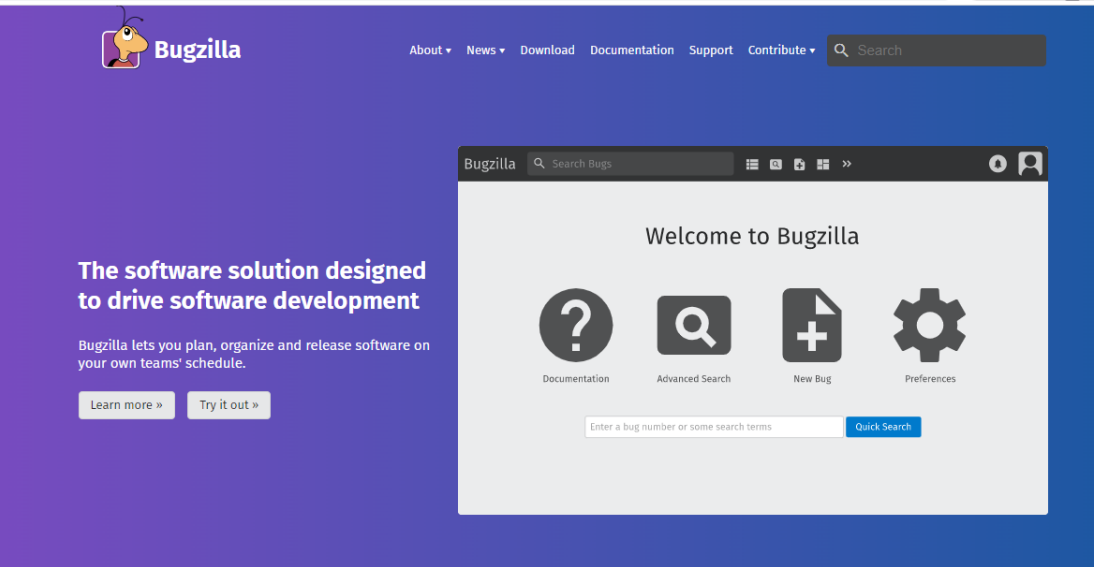


Bug tracking systems

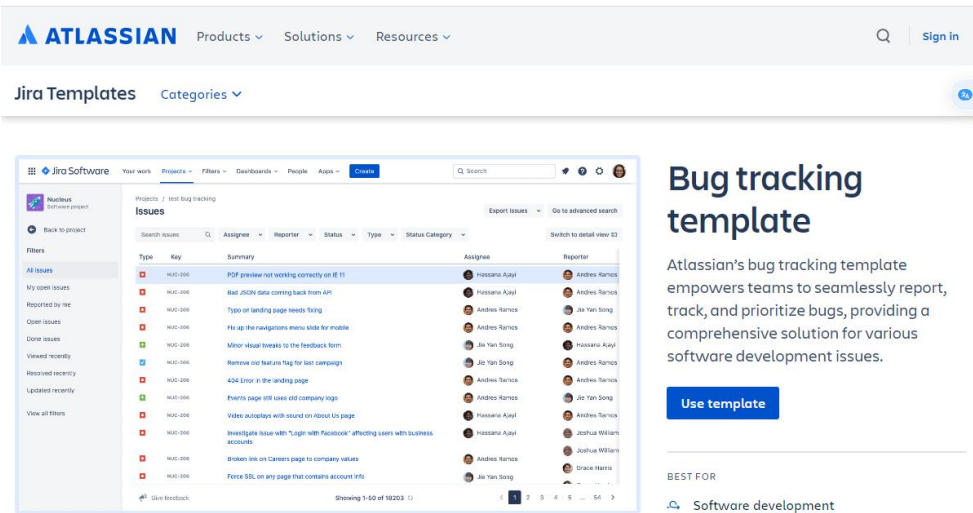
GitHub Issues



Bugzilla



Jira



Bug tracking template

Atlassian's bug tracking template empowers teams to seamlessly report, track, and prioritize bugs, providing a comprehensive solution for various software development issues.

Use template

BEST FOR
Software development

KEY FEATURES
Bug tracking
Progress tracking
Sprint analytics tools

Fields in Bugzilla bug report

Apache OpenOffice (AOO) Bugzilla – Issue 128594 Pasting vector graphics formats (WMF, EMF, SVG) inserts a bitmap instead Last modified: 2024-02-23 19:05:17 UTC

[Home](#) | [New](#) | [Browse](#) | [Search](#) | [Search](#) | [\[2\]](#) | [Reports](#) | [Requests](#) | [Help](#) | [Log In](#) | [Forgot Password](#)

Issue List: (1 of 14) [First](#) [Last](#) [Prev](#) [Next](#) [Show last search results](#)

Issue 128594 - Pasting vector graphics formats (WMF, EMF, SVG) inserts a bitmap instead

[Status:](#) CONFIRMED
[Alias:](#) None
[Product:](#) Draw
[Component:](#) open-import ([show other issues](#))
[Version:](#) 4.2.0-dev
[Hardware:](#) All All
[Importance:](#) P5 (lowest) Normal ([vote](#))
[Target Milestone:](#) ---
[Assignee:](#) AOO issues mailing list
[QA Contact:](#)
[URL:](#)
[Keywords:](#)
[Depends on:](#)
[Blocks:](#)

[Reported:](#) 2024-02-23 19:05 UTC by damjan
[Modified:](#) 2024-02-23 19:05 UTC ([History](#))
[CC List:](#) 0 users
[See Also:](#)
[Issue Type:](#) DEFECT
[Latest Confirmation in:](#) ---
[Developer Difficulty:](#) ---

Attachments

[Add an attachment](#) (proposed patch, testcase, etc.)

Note

You need to [log in](#) before you can comment on or make changes to this issue.

damjan 2024-02-23 19:05:17 UTC [Description](#)

In Inkscape, draw a circle, and copy it to the clipboard.

On the latest trunk version of OpenOffice (after commit dbea4404ca2c86a3a74294e262e6584c3e450818), paste it into Draw.

It will paste the circle. Clicking it to select it displays "Bitmap with transparency selected". Enlarging it by dragging the edges makes it blurry, because it didn't paste as a circle that can be re-drawn at a large size, but as bitmap pixels.

Special paste, with Ctrl+Shift+V, also allows pasting as "GDI metafile". Choosing that option also pastes successfully. Clicking the circle to select it display "Metafile selected". Enlarging it by dragging the edges also makes it blurry. Saving this image as SVG shows the SVG contains a bitmap, not the description of a circle.

WMF and EMF have the same problem as SVG (they're harder to test, you have to use xclip with special MIME types specific to OpenOffice).

However saving the circle from Inkscape into an SVG file, and opening the SVG file with Draw, does produce a real circle, that still looks good when resized, and can be saved and exported and copied out as a real circle.

What goes wrong when pasting, that converts all vector graphics into raster graphics?

- Severity
 - Priority
 - Status indicates the general health of a bug.
 - Resolution indicates what happened to this bug.
 - Assigned to – the person in charge of resolving the bug
- Importance of a bug

https://bz.apache.org/ooo/show_bug.cgi?id=128594

Severity

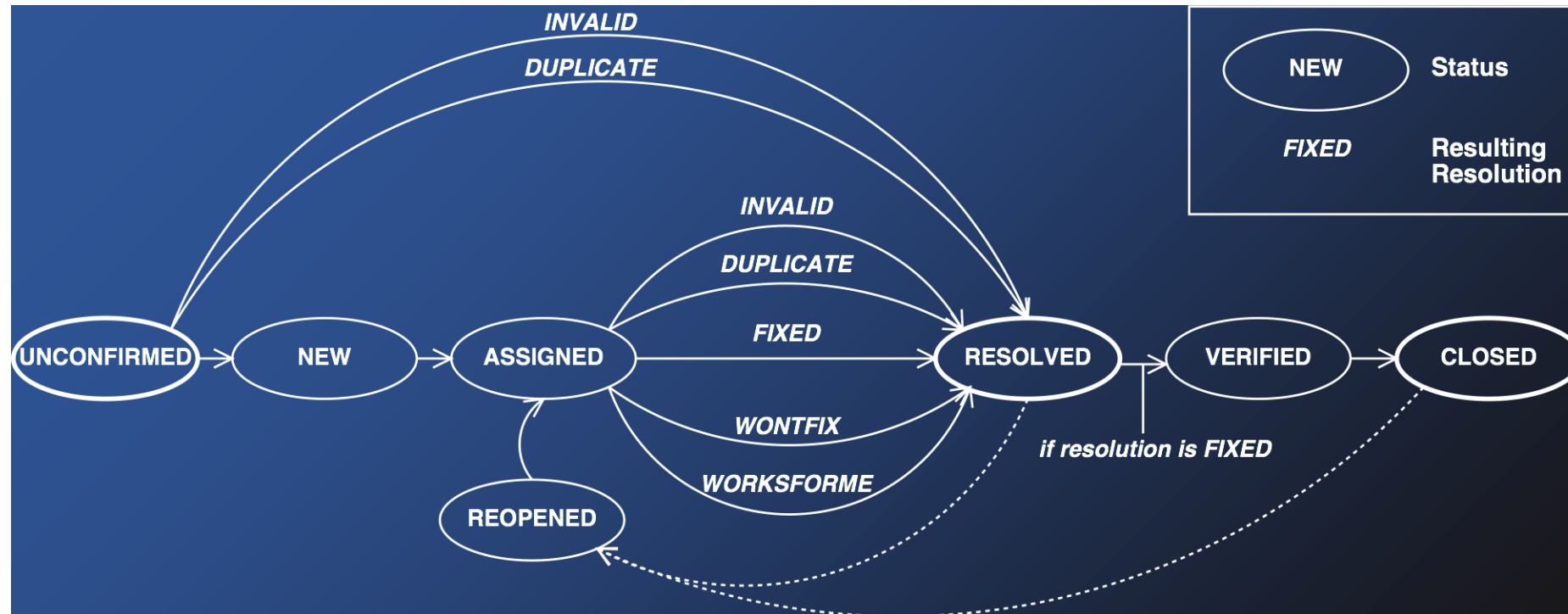
- Describes the “technical” impact or seriousness of a bug on the functionality or usability of the software.
- Describes the degree of impact the bug has on the system or users.
- The severity level is typically set by the person reporting the bug or by a triage team.

Blocker	blocks development and/or testing work
Critical	crashes, loss of data, severe memory leak
Major	major loss of function
Normal	regular issue, some loss of functionality under specific circumstances
Minor	minor loss of function, or other problem where easy workaround is present
Trivial	cosmetic problem like misspelled words or misaligned text
Enhancement	request for enhancement

Priority

- Priority in Bugzilla represents the order in which bugs should be addressed or fixed based on factors like urgency, business requirements, and resource availability.
 - From P1 (highest priority) to P5 (lowest priority).
- Utilized by the management/programmers/engineers to prioritize their work to be done.
- According to the business value of the function
 - P1.** Must fix to proceed with the rest of the project effort, including testing.
 - P2.** Must fix for release; no customer will buy our product with this bug.
 - P3.** Fix desirable prior to release, as customers will object to the problem.
 - P4.** Time-to-market is definitely more important than fixing this bug, so fix it only if the release date not delayed.
 - P5.** Fix whenever convenient

The Defect Lifecycle (Bugzilla)



Status

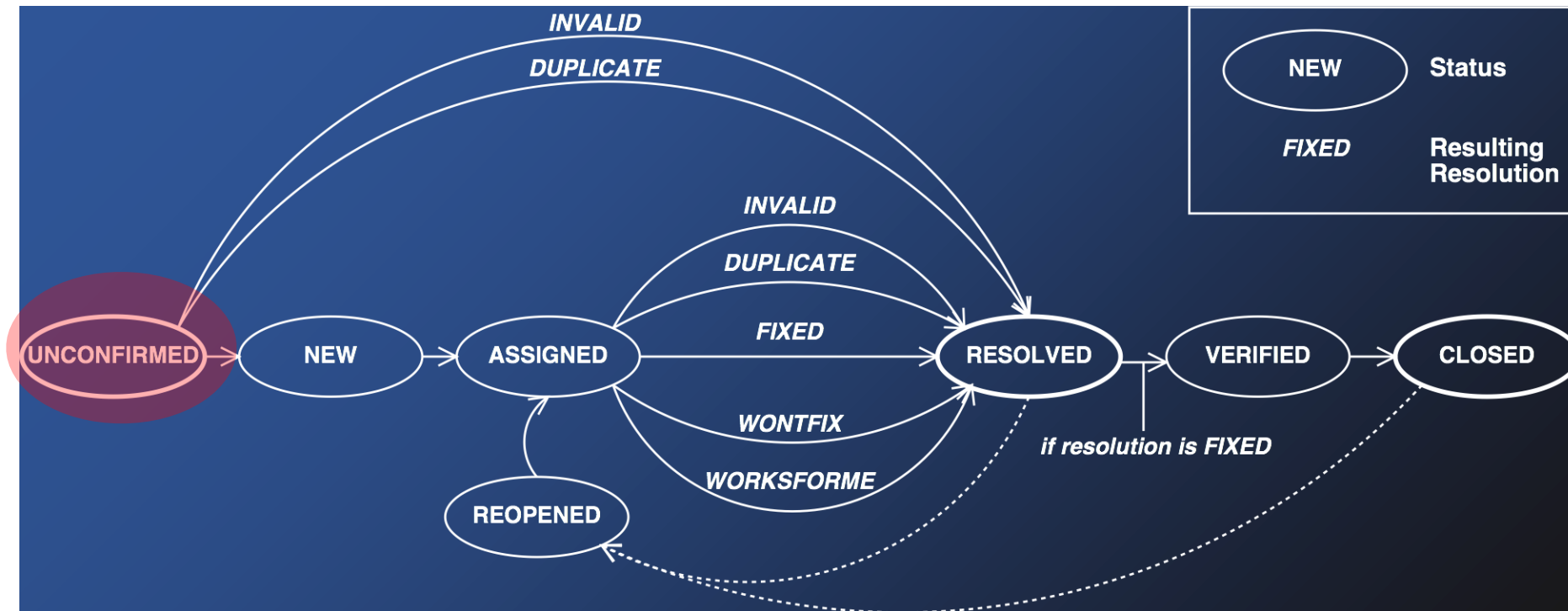
- Status describes the state of the bug at a particular point in time.

UNCONFIRMED	This bug has recently been added to the database. Nobody has validated that this bug is true.
NEW	This bug has recently been added to the assignee's list of bugs and must be processed
REOPENED	This bug was once resolved, but the resolution was deemed incorrect (e.g. a bug is REOPENED when more information shows up and the bug is now reproducible.)
ASSIGNED	The person has accepted to handle the bug
RESOLVED	A resolution has been taken, and it is awaiting verification by QA.
VERIFIED	QA has looked at the bug and the resolution and agrees that the appropriate resolution has been taken.

Open bugs

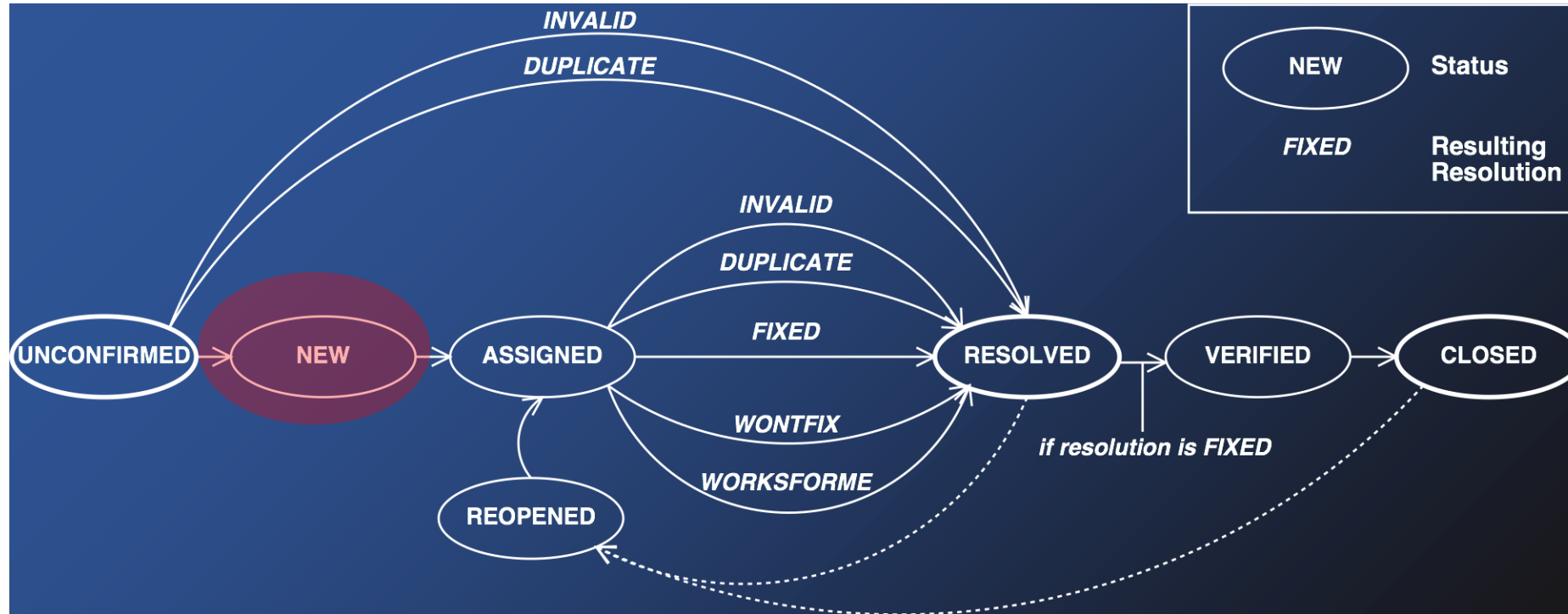
Closed bugs

Unconfirmed



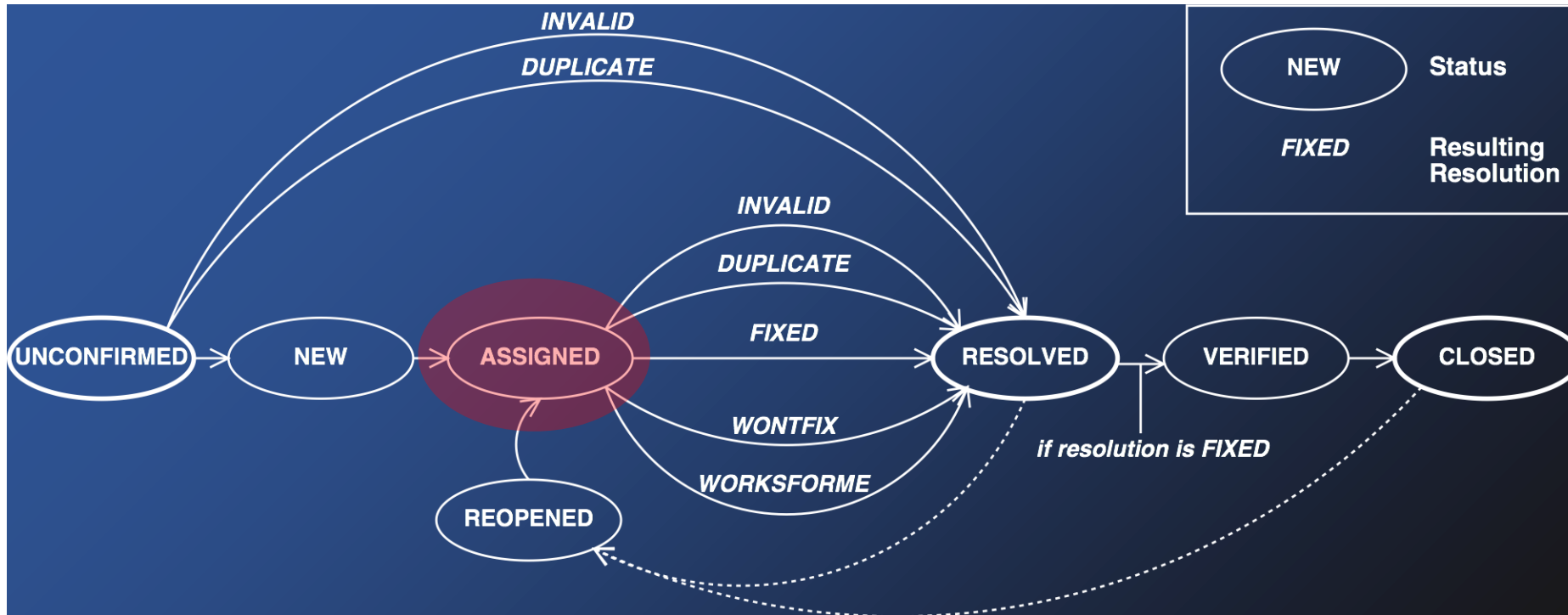
- The bug report has just been entered into the database
 - Testers, users, customers, etc

New



- The report is valid (contains the relevant facts) and not a duplicate. If not, it becomes resolved.

Assigned



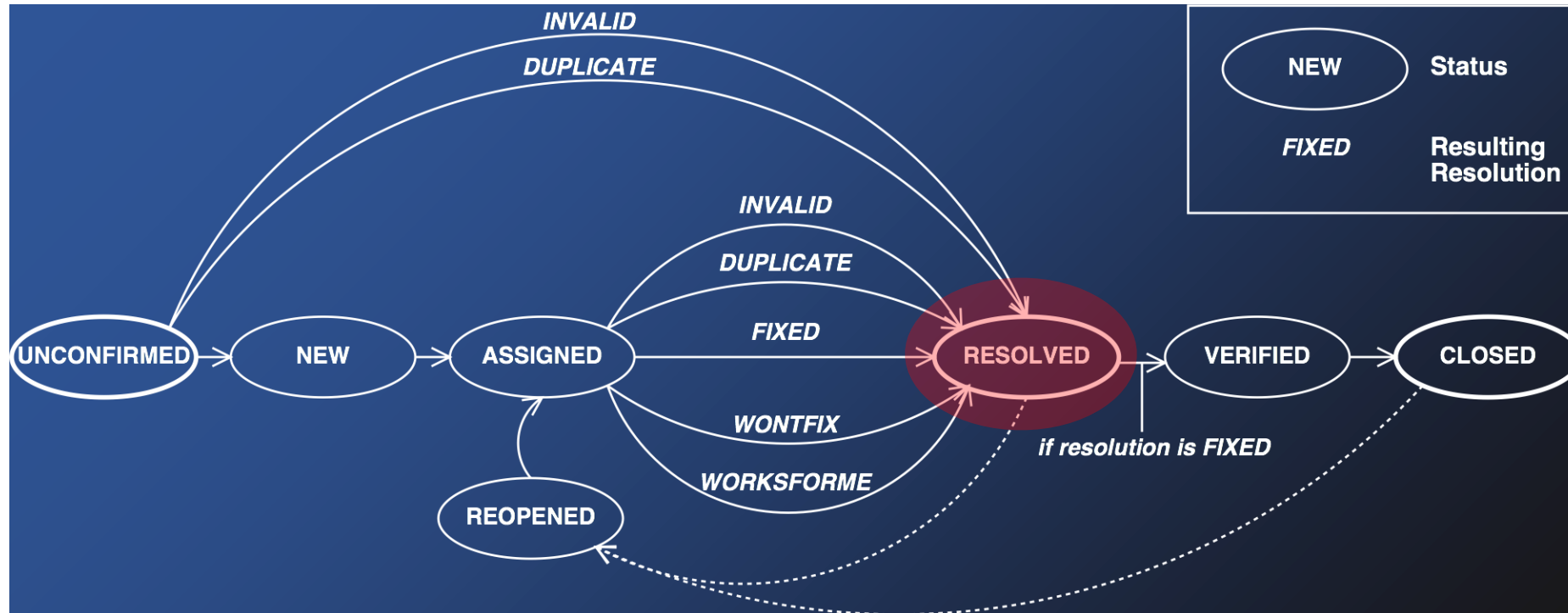
- The bug is assigned to a developer

Resolution for closed bugs

- The resolution of a bug in Bugzilla describes the final outcome or disposition of the bug report. It indicates how the bug was resolved or why it was closed.

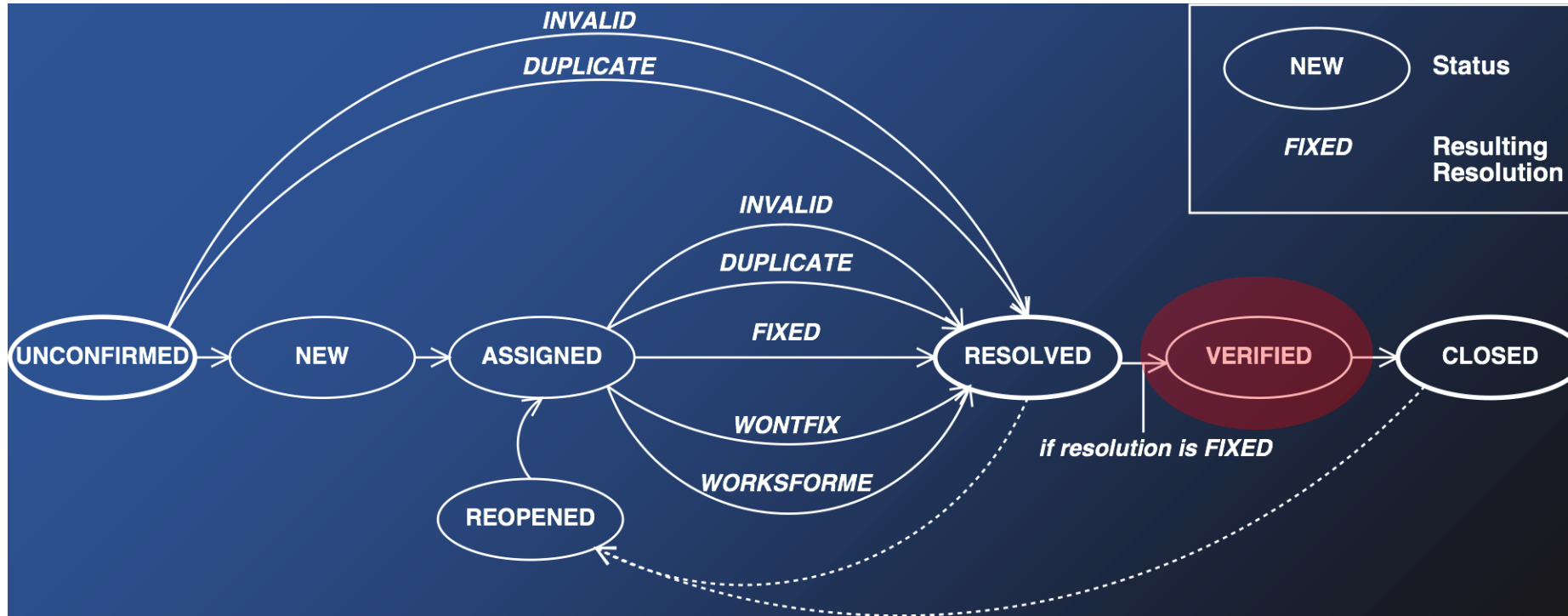
FIXED	A fix for this bug is checked into the tree and tested.
INVALID	The problem described is not a bug.
WONTFIX	The problem described is a bug which will never be fixed.
DUPLICATE	The problem is a duplicate of an existing bug.
WORKSFORME	All attempts at reproducing this bug were futile, and reading the code produces no clues as to why the described behavior would occur
INCOMPLETE	The problem is vaguely described with no steps to reproduce, or is a support request.

Resolved



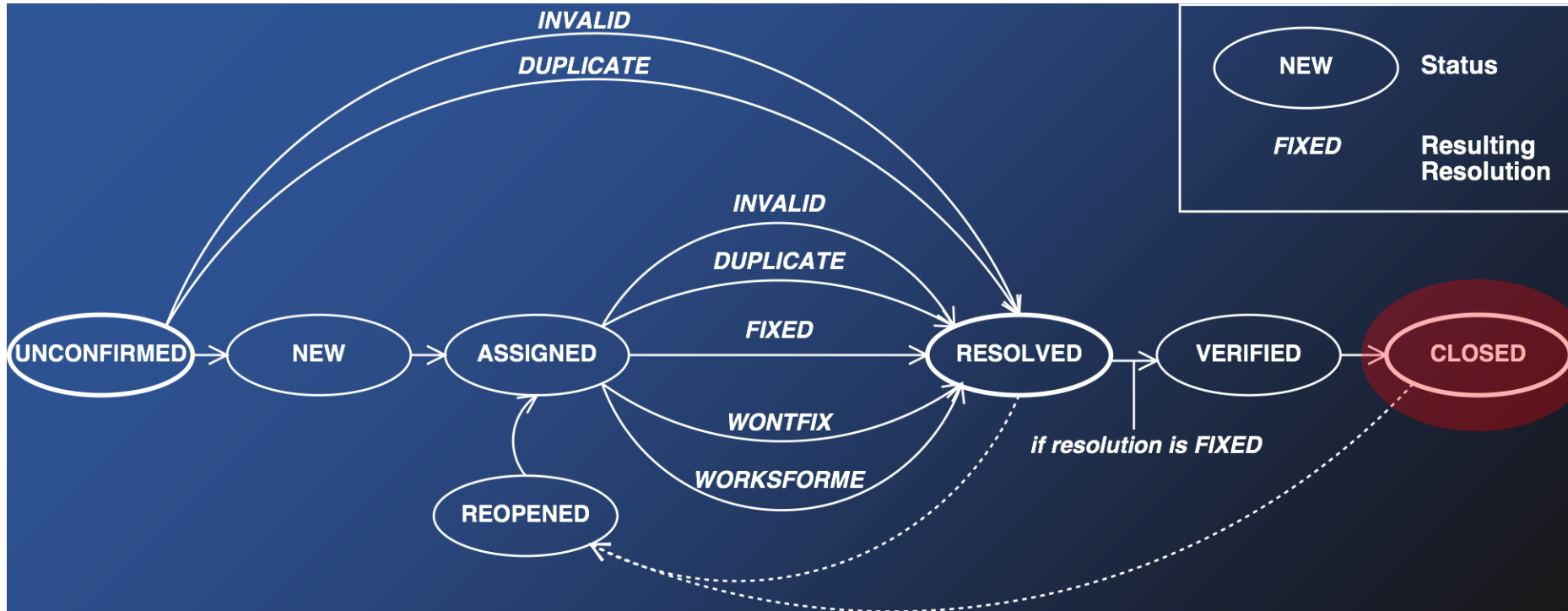
- The defect report has been processed and the resolution defines the outcome

Verified



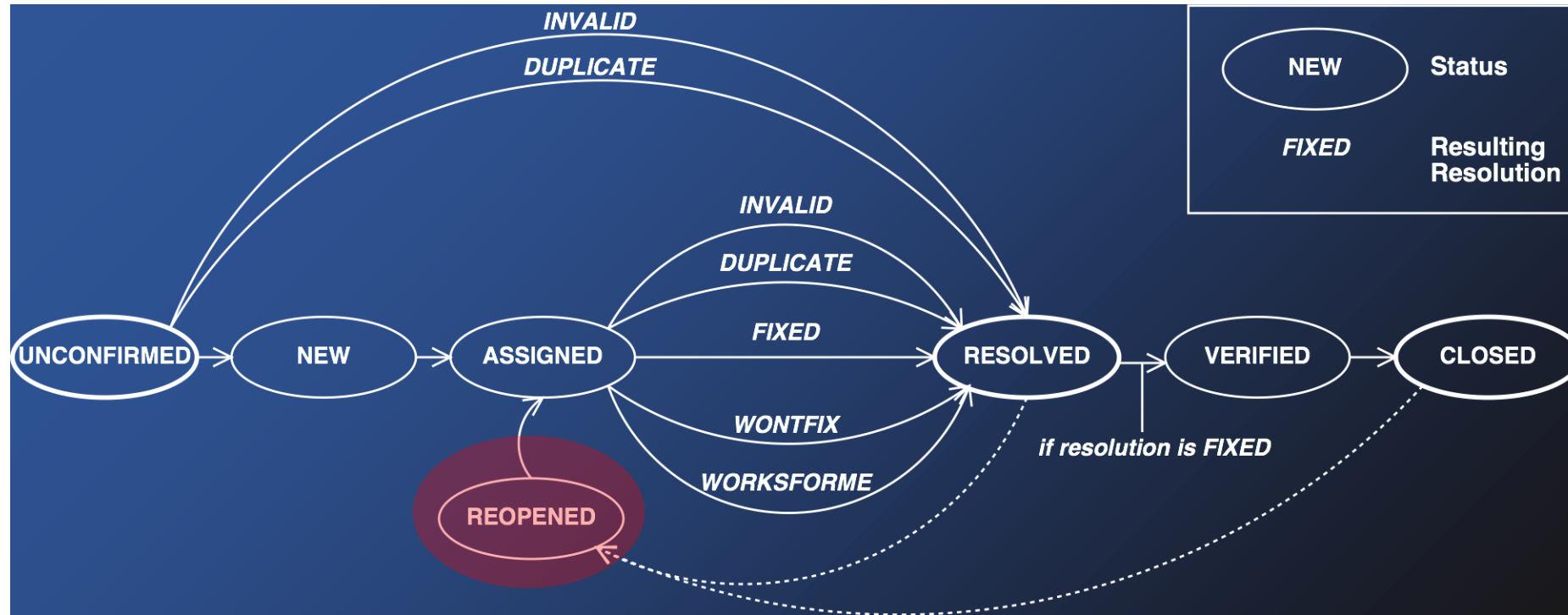
- The defect is fixed and the fix has been successful (E.g. verified by the tester)

Closed



- A new version with the fix has been released and shipped.

Reopened



- This bug was once resolved, but the resolution was deemed incorrect.
 - E.g., a WORKSFORME bug is REOPENED when more information shows up and the bug is now reproducible.

Testing vs. debugging

Specification

A savings account in a bank earns a different rate of interest depending on the **balance** in the account:

If a **balance** in the range \$0 up to \$100 has a 3% interest rate and **balances** of \$1000 and over have a 7% interest rate.

Implementation

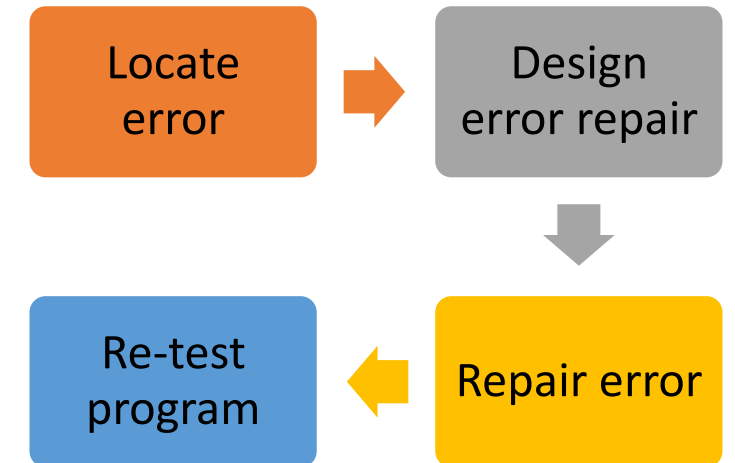
```
.....  
if (balance <=100)  
    interest = 3%  
else if (balance>1000)  
    interest = 7%  
.....
```

Three types of Defects

- **Error:** An incorrect/missing human action
- **Fault:** Incorrect specification, design, software implementation (a bug in the code), or artifacts related to software (e.g. user manual)
- **Failure:** An observable incorrect program behaviour

What's the fault?

- Testing
 - Detect the presence of faults
 - Detect the presence of faults by observing failure
- Debugging
 - The process of locating the exact cause of a defect (fault), and removing the fault



Confirmation vs. Regression Testing

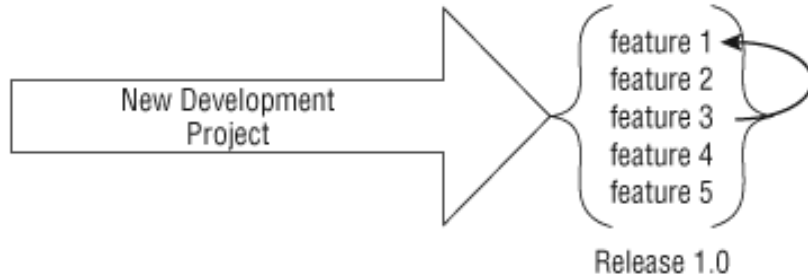
- **Confirmation testing**

- Testing that run test cases that failed the last time, in order to verify the success of the correctness action.

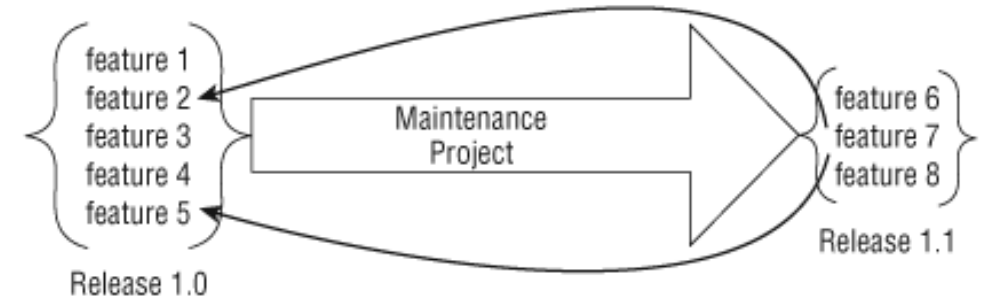
- **Regression Testing**

- **Regression:** The misbehavior of a previously correct function, attribute or feature
- Testing of a previously tested program following modification (e.g. after bug fixing or adding new features)
- Existing test cases are re-run to ensure that defects have not been introduced or uncovered in unchanged area of the software as a result of change.

Example



- Receive a feature-complete test release and start testing.
- Test features 1 and 2 without any problem.
- Find a bug in feature in 3, which is fixed by the programmer.
- Regression testing for feature 1 and 2

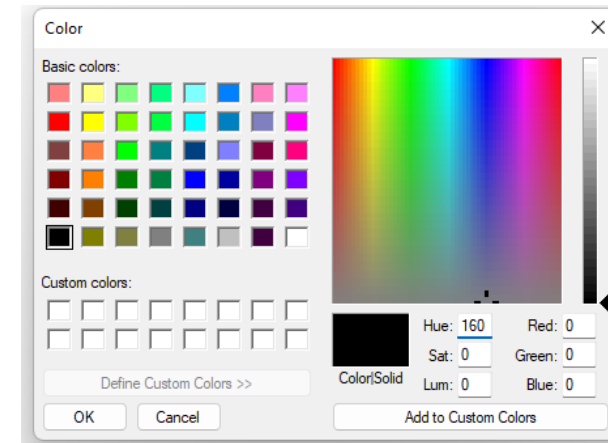
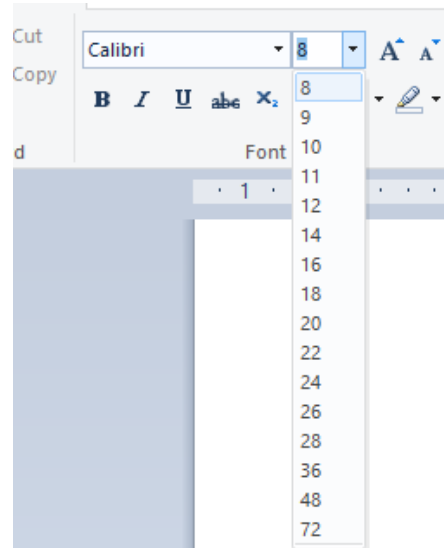
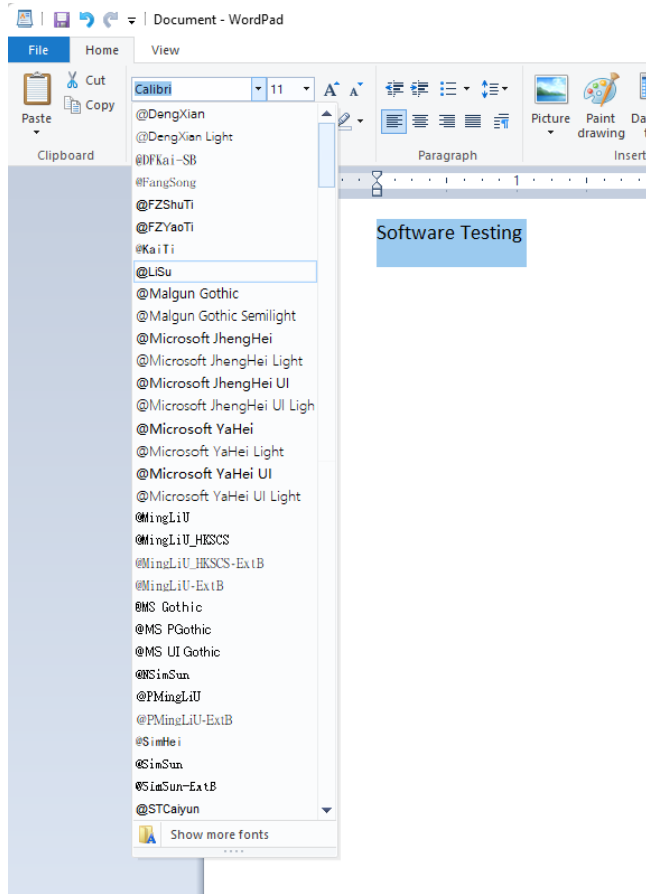


- Major update, release 1.1
- Test the new features
- Re-test the features from the first release

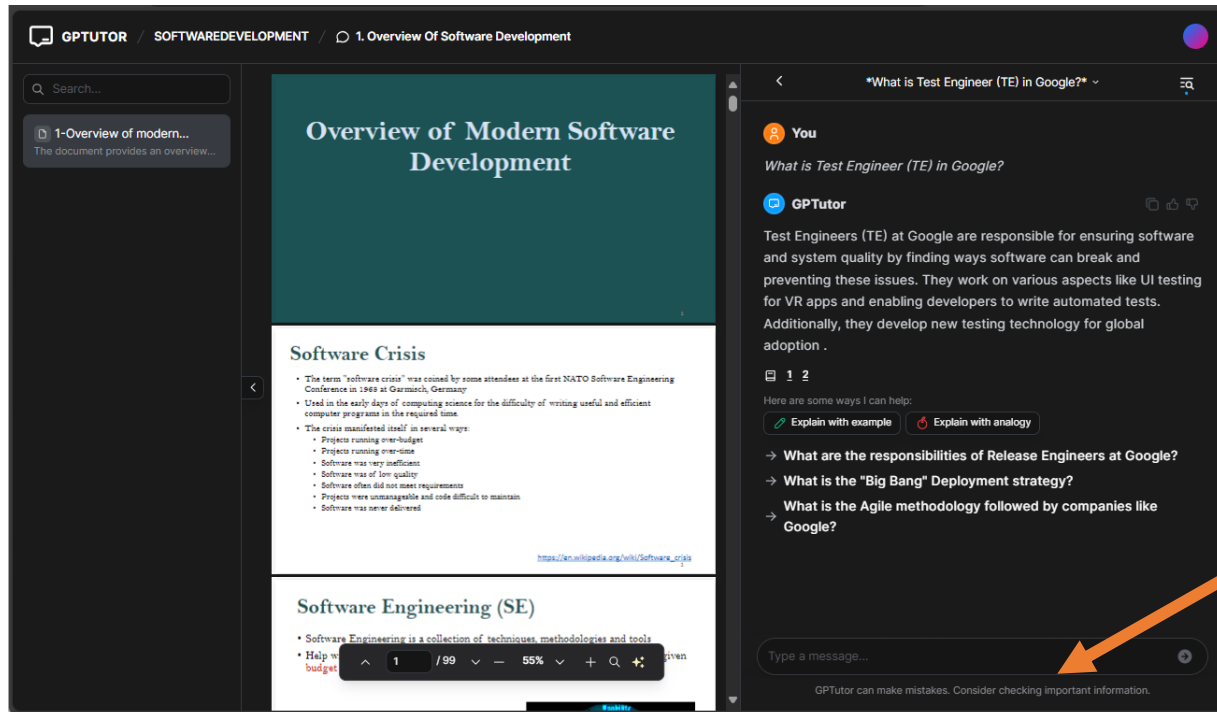
Strategies for regression testing

- Conservative approach: re-run all test cases
 - Test automation may speed up regression testing.
- Select subset of features for re-testing
 - E.g. Run only tests that reach or related to the modified statements/modules

What are the test cases in WordPad?



What are the test cases in GPTutor?



- What can the user input in this textbox?
- What is the expected output for the different types of input?
- What are the challenges in testing an application based on LLM?

Exercise

Think of the different types of test cases we may need to handle.

For each type of test case, define the expected response of the chatbot. Does it meet your expectation?

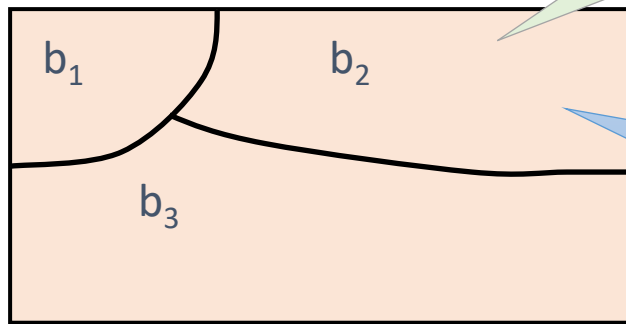
Complete testing is impossible

- Testing everything (all combinations of input and preconditions) is not feasible except in trivial cases.
- The number of possible inputs is very large.
- The number of possible outputs is very large.
- The number of paths through the software is very large.
- The software specification is subjective. You might say that a bug is in the eye of the beholder.

Instead of running all the test cases, we select only the most representative test cases to test the program!

Equivalence Class Testing

- Goal: to reduce the number of test cases
- Sub-divide the set of all possible inputs into equivalence classes (EC) that receive equivalent treatment
- Select one test case from each partition



If one test case in an EC detects a defects, all other test cases in the same EC are likely to detect the same defect


If one test case in an EC does not detect a defect, no other test cases in the same EC is likely to detect the defect.

Example

Input variable
Deposit Amount

Valid ECs:
EC1: \$4,999.99 or below
EC2: \$5,000.00 to \$499,999.99
EC3: \$500,000.00 to \$999,999.99
EC4: \$1,000,000.00 or above

Invalid ECs:
IEC1: \$0 or below

 大新銀行 DAH SING BANK			
Hong Kong Dollar Deposit Rates	Foreign Currency Deposit Rates	FX Rates	Other Rates
Hong Kong Dollar Deposit Rates			
HKD Savings Rate			
Deposit Balance (HKD)		Interest Rate (p.a.)	
\$1,000,000.00 or above		0.8750%	
\$500,000.00 to \$999,999.99		0.8750%	
\$5,000.00 to \$499,999.99		0.8750%	
\$4,999.99 or below		0.0000%	

<https://www.dahsing.com/pws/rate-enquiry/hkd?lang=en-US>

Test Case	Valid EC	Deposit Amount	Expected output (Interest rate)
TC1	\$0.01 to \$4,999.99	\$1000	0%
TC2	\$5,000.00 to \$499,999.99	\$10,000	0.8750%
TC3	\$500,000.00 to \$999,999.99	\$600,000	0.8750%
TC4	\$1,000,000.00 or above	\$2,000,000	0.8750%

Test Case	Invalid EC	Deposit Amount	Expected output
TC1	0 or below	0	ERROR

Boundary Value Testing (BVT)

- Errors tend to occur near the extreme values of an input variable
- Examples of test cases
 - Just below minimum (min-)
 - Minimum (min)
 - Nominal value
 - Maximum (max)
 - Just above maximum (max+)

EC2: \$5,000.00 to \$499,999.99

Test Cases

- \$4999.00
- \$5,000.00
- \$10,000
- \$499,999.99
- \$500,000.00

Types of Fault related to BVT

- Closure fault
 - The relational operator \leq is implemented as $<$
 - The relational operator $<$ is implemented as \leq .
- Shifted-Boundary fault
 - When the constant term of the inequality defining the boundary takes up a value different from the intended value
 - E.g. $x > 5$ becomes $x > 4$

Test Case	EC	Deposit Amount	Expected output (Interest rate)
TC1	\$0.01 to \$4,999.99	\$1000	0%
TC2	\$5,000.00 to \$499,999.99	\$10,000	0.8750%
TC3	\$500,000.00 to \$999,999.99	\$600,000	0.8750%
TC4	\$1,000,000.00 or above	\$2,000,000	0.8750%

```
def calculate_interest(balance):  
    if balance >= 1000000:  
        interest_rate = 0.00875  
    elif balance >= 500000:  
        interest_rate = 0.00875  
    elif balance >= 5000:  
        interest_rate = 0.00875  
    else:  
        interest_rate = 0.0  
  
    interest = balance * interest_rate  
    return interest  
  
interest_earned = calculate_interest(balance)  
print("Interest earned: $", interest_earned)
```

What is a successful software testing?

- If no defect is found in testing, can we conclude that the software is defect-free?
- If the testing does not reveal any bug, may be it's just the case that the test case is not good enough to reveal the bug!
- Software testing can only show the presence of defects
- A “successful” test is one that can detect as many defects as possible
 - Prerequisite for locating and eliminating the defects, which may lead to an improvement in software quality
- More testing => more confidence that software is of good quality

*Software testing should **NOT** be used
to show that the software work.*

Primary Goals of Software Testing

- **Verification** is a process of evaluating the software products of a software development lifecycle to check if we are in the right track of creating the final product.
 - E.g. Reviews, Walkthroughs, Inspections, Unit Testing
 - Typically done by the developer to ensure that the software meets specified requirements and is correctly implemented.
 - Making sure we build the product right
- **Validation** is the process of evaluating the software product to check whether the software meets the user/business needs
 - E.g. Acceptance testing, Testing in production
 - Making sure we build the right product



Test Coverage

White box vs. Black box testing

Black-box testing

- Aka specification testing
- Test case generation are based solely on the knowledge of the system requirements.
- Ensures that specified features of the software are addressed by some aspect of the program



White-box testing

- Aka Code-based testing
- Test data selection is guided by information derived from the internal structure and internal functional properties of the program

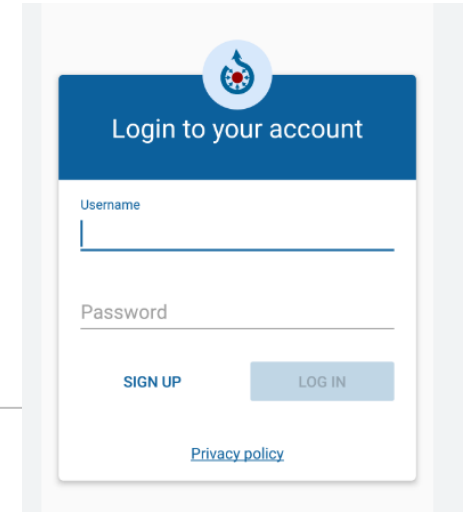
```
if (testscore >= 90) {  
    grade = 'A';  
} else if (testscore >= 80) {  
    grade = 'B';  
} else if (testscore >= 70) {  
    grade = 'C';  
} else if (testscore >= 60) {  
    grade = 'D';  
} else {  
    grade = 'F';  
}
```

Test Coverage

- Helps answer the following questions
 - Is your software tested sufficiently?
 - In what software component you haven't tested sufficiently?
- Black box coverage criterion
 - Requirements coverage: $\frac{\text{\# of tested requirement}}{\text{total requirements}}$
- White box coverage criterion
 - Statement coverage
 - Decision coverage
 - Condition coverage
 - Multiple condition coverage
 - Modified condition/Decision coverage (MC/DC)

Example: Requirement Coverage

Each requirement should be covered by at least ONE test case.



Login to your account

Username

Password

SIGN UP LOG IN

[Privacy policy](#)

1. Requirement: Minimum and Maximum Length

1. The username field should accept a minimum of 6 characters and a maximum of 20 characters.
2. The password field should accept a minimum of 8 characters and a maximum of 30 characters.

2. Requirement: Character Types Accepted

1. The username field should accept alphanumeric characters (A-Z, a-z, 0-9) and special characters such as ".", "_", and "-".
2. The password field should accept a combination of alphanumeric characters (A-Z, a-z, 0-9) and special characters such as ".", "_", "@", and "#".

3. Requirement: Character Types Not Accepted

1. The username field should not accept spaces or any other special characters apart from ".", "_", and "-".
2. The password field should not accept spaces or any special characters other than ".", "_", "@", and "#".

4. Requirement: Password Strength

1. The password field should enforce a minimum level of complexity by requiring at least one uppercase letter, one lowercase letter, one numeric digit, and one special character.

State transition testing

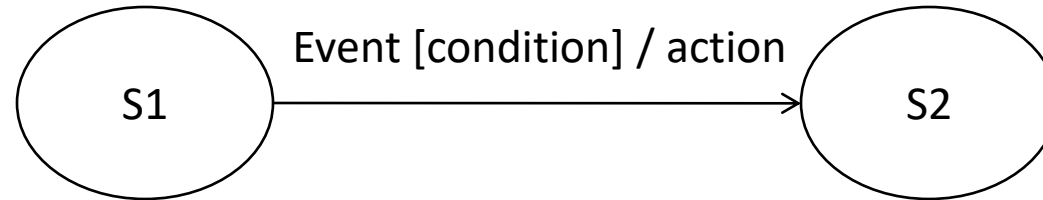
- In many systems, behaviour/output depends on the input as well as the current state
- The current state depends on
 - The initial state
 - The sequence of inputs the system has received in the past
- Example: ON/OFF button, ATM



State transition diagram

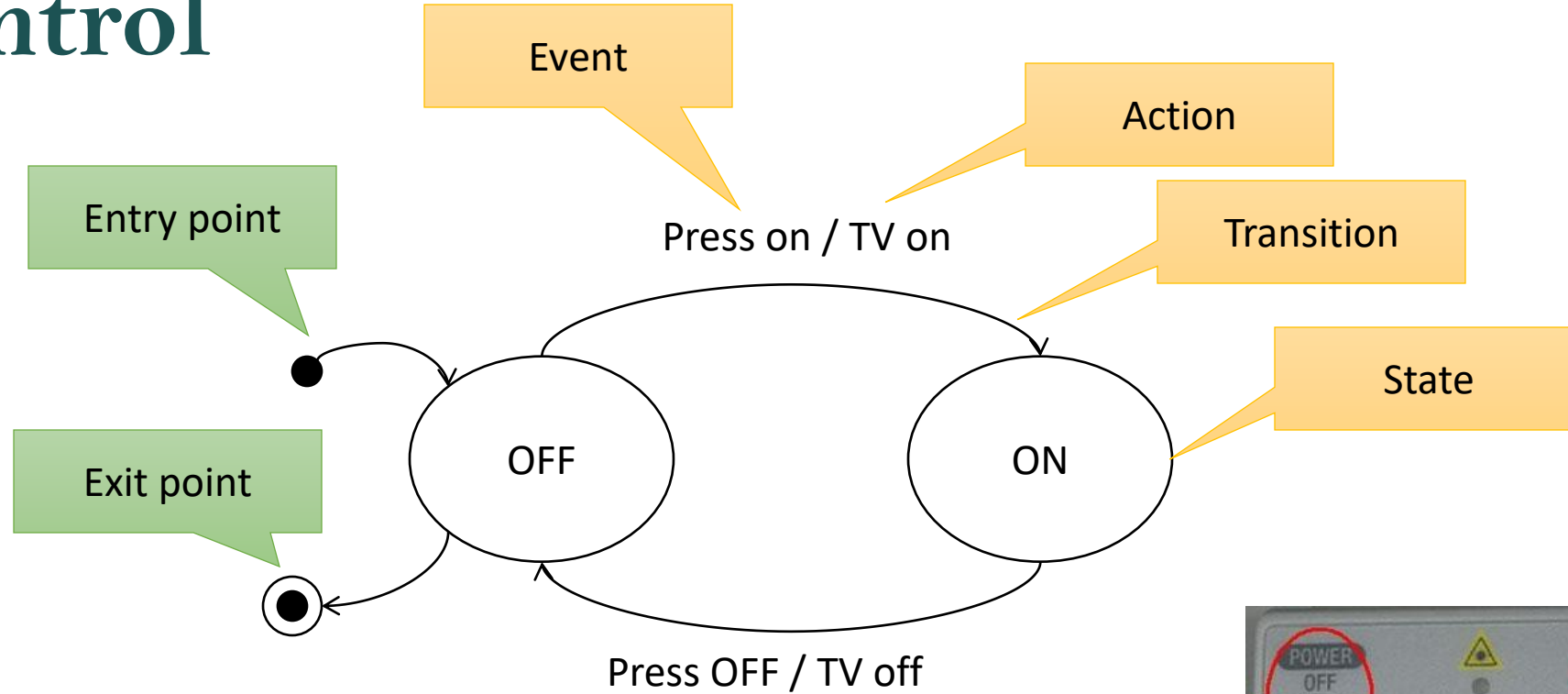
- In state transition testing, test cases are generated from the state-transition diagrams (the specification).
- The diagram documents the states that a system can exist in and the events that come into and are processed by a system as well as the system's responses
- Components
 - State
 - Condition in which a system is waiting for one or more events.
 - Represented by a circle
 - Event
 - May be external (e.g. input by the user) or event generated within the system (e.g. timeout event).
 - With an event, the system can change state or remain in the same state and/or execute an action
 - Action
 - An operation initiated because of a state change
 - E.g. output a certain message, start a timer, generate a ticket

Notation



- Transition (represented by an arrow)
 - represents a change from one state to another
 - each transition can be associated with an event (which triggers the transition) and action (which specify the behaviour/output)
- Entry point (represented as a black dot)
- Exit point (represented as a bulls-eye symbol)

Example: ON/OFF Button for remote control



Test case:
1) (Press ON, Press OFF)



Coverage Criteria for state transition testing

$$\text{state coverage} = \frac{\text{no. of states exercised}}{\text{total no. of states}}$$

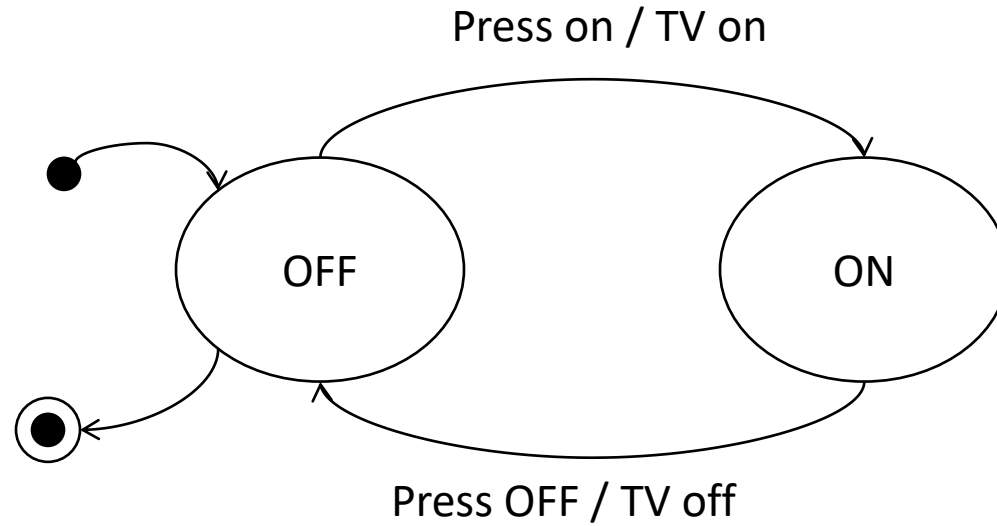
$$\text{transition coverage} = \frac{\text{no. of transitions exercised}}{\text{total no. of transitions}}$$

$$\text{2 - transition coverage} = \frac{\text{no. of sequences of 2 transitions exercised}}{\text{total no. of sequences of 2 - transitions}}$$

Note: We can generalize the 2-transition coverage to n transitions coverage.

$$\text{state - input coverage} = \frac{\text{no. of state - input pairs exercised}}{\text{no. of states} \times \text{no. of inputs}}$$

Improved test set

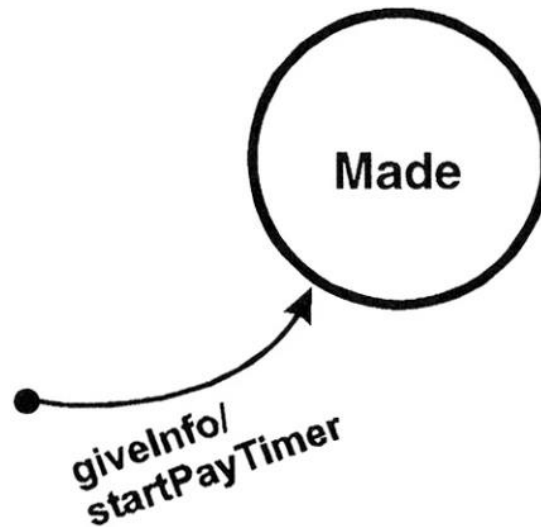


Test case:

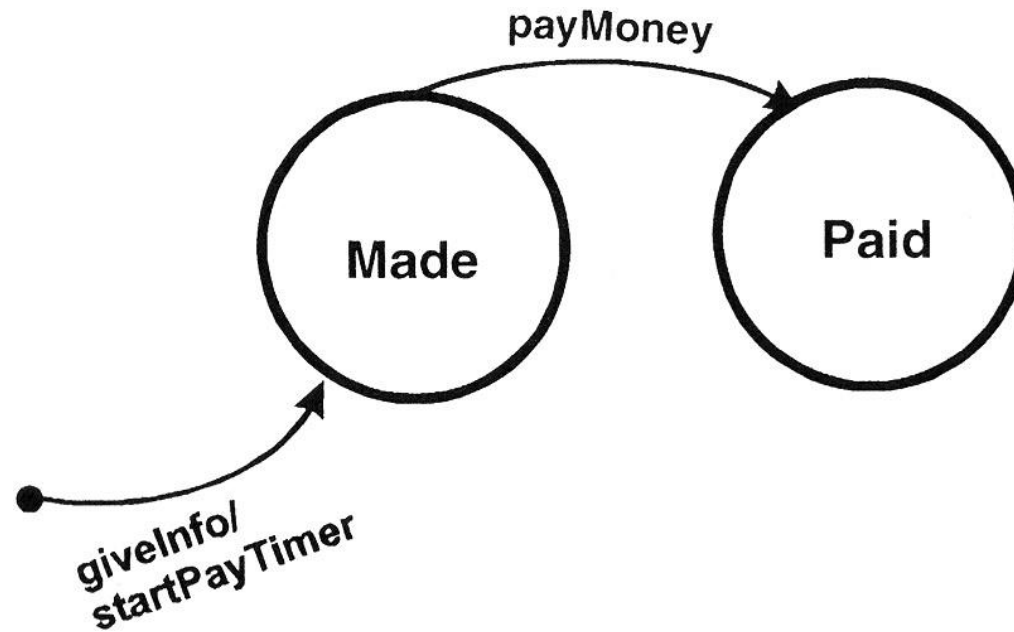
- 1) (Press ON, Press OFF, Press ON, Press OFF)
- 2) (Press OFF, Press OFF, Press ON, Press ON, Press OFF)

Example: Airline reservation application

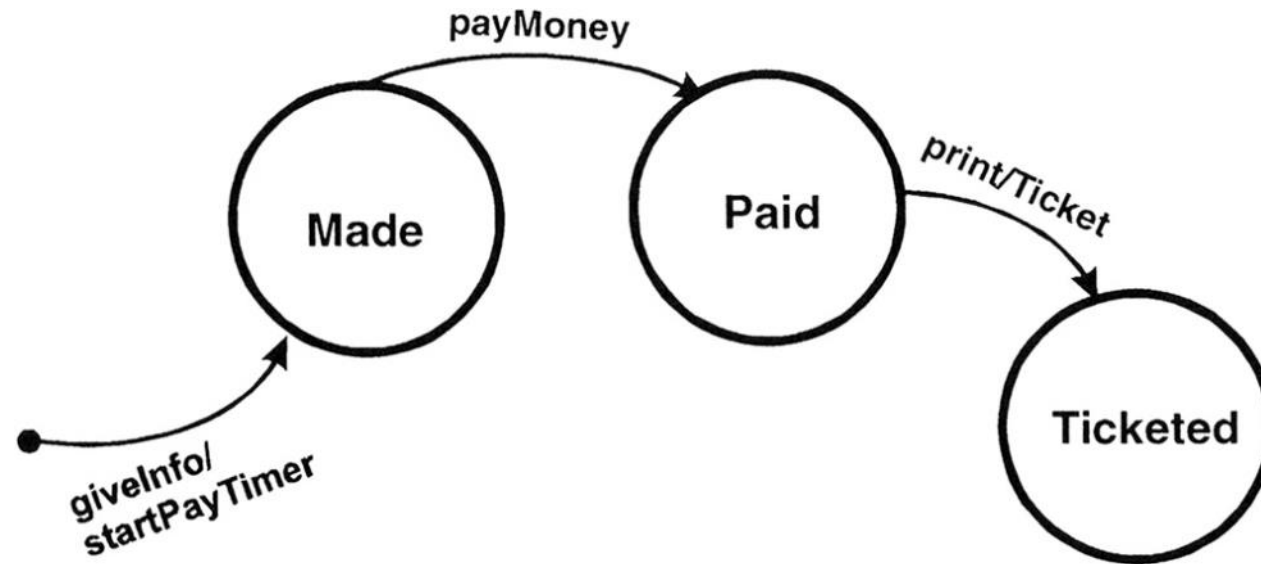
- ▶ The customer first provide some information including departure and destination cities, dates, and times, which is used by the reservation agent to make a reservation.
- ▶ The reservation is now at the “made” state.



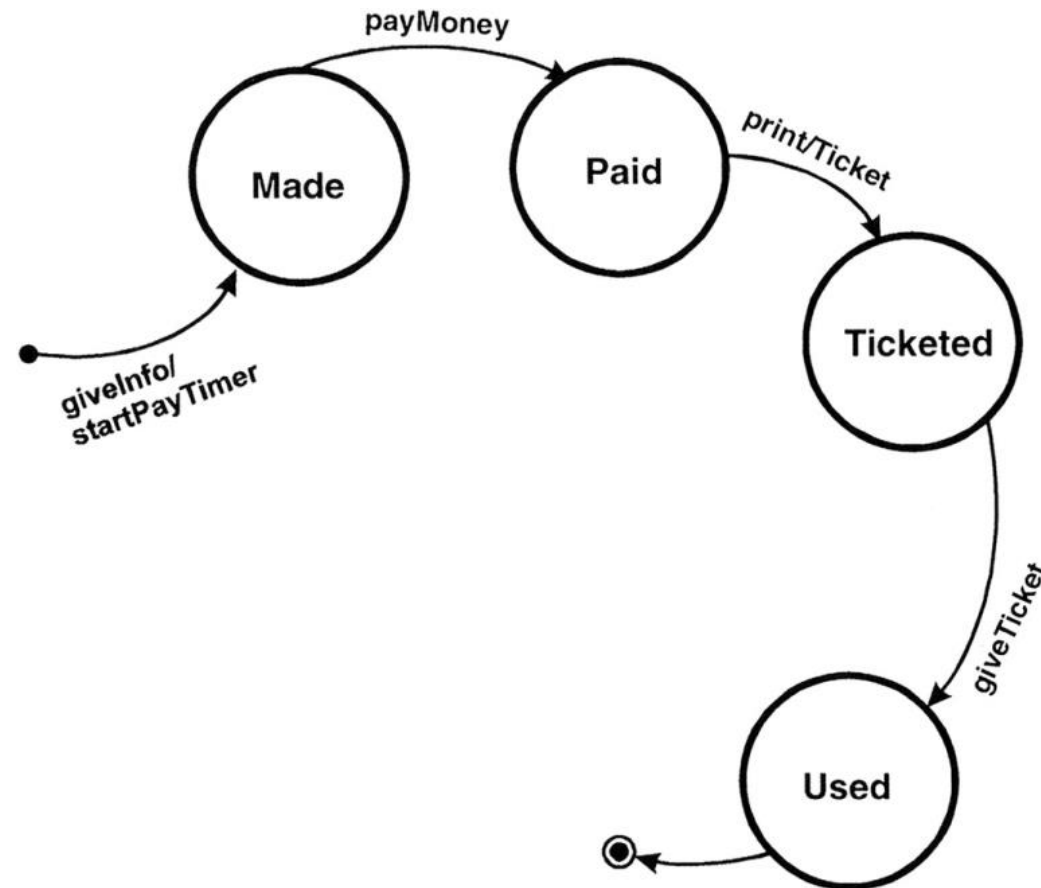
- If the customer pays before the startPayTimer expires, the reservation transits to the “Paid” state.



- From the Paid state the Reservation transitions to the Ticketed state when the print command (an event) is issued.
- In addition to entering the Ticketed state, a Ticket is output by the system.



- From the Ticketed state we giveTicket to the gate agent to board the plane.



Some alternative paths

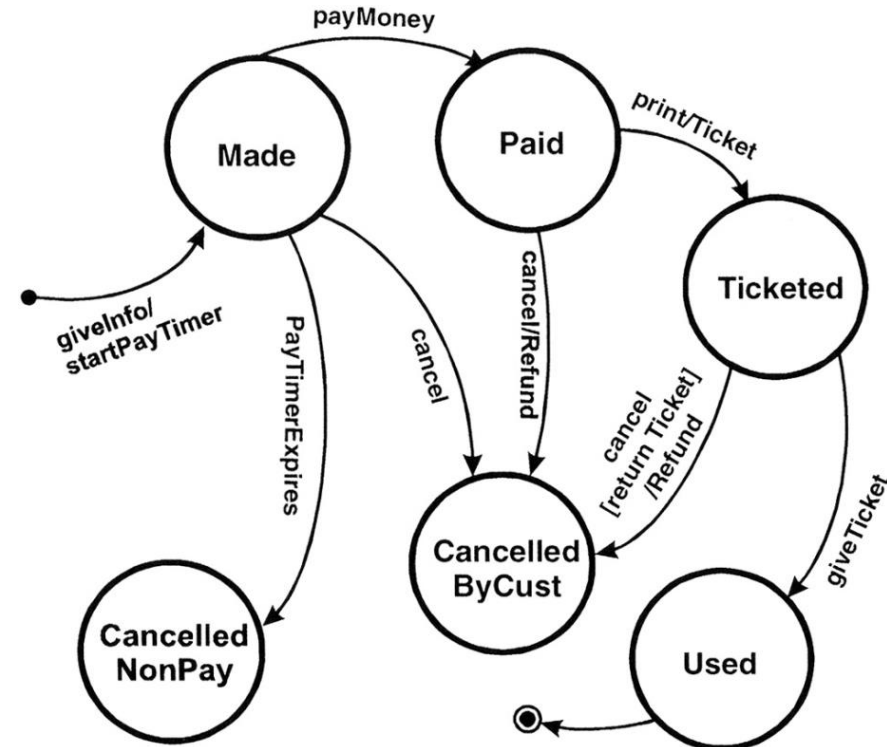
- If the reservation is not paid for in the time allotted (the PayTimer expires), it is cancelled for non-payment.
- From the Made state the customer (through the reservation agent) asks to cancel the reservation. A new state, Cancelled By Customer, is required.
- In addition, a reservation can be cancelled from the Paid state. In this case a Refund should be generated and leave the system. The resulting state again is Cancelled By Customer.
- From the Ticketed state the customer can cancel the Reservation. The airline will generate a refund but only when it receives the printed ticket from the customer.

Exercise

Perform state transition testing for the airline reservation application.

What is the minimum number of test cases required to

- i) Covers all states?
- ii) Covers all transitions?



White box vs. Black box testing

Black-box testing

- Aka specification testing
- Test case generation are based solely on the knowledge of the system requirements.
- Ensures that specified features of the software are addressed by some aspect of the program



White-box testing

- Aka Code-based testing
- Test data selection is guided by information derived from the internal structure and internal functional properties of the program

```
if (testscore >= 90) {  
    grade = 'A';  
} else if (testscore >= 80) {  
    grade = 'B';  
} else if (testscore >= 70) {  
    grade = 'C';  
} else if (testscore >= 60) {  
    grade = 'D';  
} else {  
    grade = 'F';  
}
```

Statement coverage

- Test cases should cover all the executable statements
- Excluding comments, empty lines, etc.

```
examScore = int(input("Enter exam score: "))
isPass = False

if examScore >= 70:
    isPass = True
print(isPass)
```

Statement Coverage =

Test Case	examScore	attendance_rate	Expected Result
1	80	0.8	True

Decision (Branch) Coverage

- Derive test cases to cover all the decisions (branches) in the program.
- A decision/branch is covered if it evaluates to both true and false outcome by at least one test cases.

```
examScore = int(input("Enter exam score: "))  
if examScore >= 70:  
    isPass = True  
else  
    isPass = False  
print(isPass)
```

Two decisions outcomes to be covered:
examScore: {T,F}

Decision Coverage =

Test Case	examScore	attendance_rate	Expected Result
1			
2			

Condition Coverage

- A decision may be composed of one or more simple conditions connected by logical operators (e.g. AND, OR, XOR).
 - E.g., the decision `(x>0 && !y>0)` consists of two conditions, `x>0` and `!(y>0)`
- The test should cover all the conditions in the program.
 - A condition is covered if it evaluates to both true and false outcome by at least one test cases.

```
examScore = int(input("Enter exam score: "))
attendance_rate = float(input("Enter attendance rate: "))

if examScore >= 70 and attendance_rate>0.5:
    isPass = True
else:
    isPass = False
print(isPass)
```

Four condition outcomes to be covered:
`examScore>=70`: {T,F}, `attendance_rate>0.5`: {T, F}

Condition Coverage =

Test Case	examScore	attendance_rate	Expected Result
1			
2			

Condition/Decision Coverage

- Condition coverage does not imply decision coverage (and vice versa)
- For condition/decision coverage, the test set should cover all the decisions and conditions in the program.
 - Each condition and decision should be evaluated to both true and false outcomes

```
examScore = int(input("Enter exam score: "))
attendance_rate = float(input("Enter attendance rate: "))

if examScore >= 70 and attendance_rate>0.5:
    isPass = True
else:
    pass = False
print(isPass)
```

Two decisions outcomes to be covered:

examScore >= 70 and attendance_rate>0.5: {T,F}

Four outcomes to be covered:

examScore>=70: {T,F}, attendance_rate>0.5: {T, F}

Test Case	examScore	attendance_rate	Expected Result
1			
2			

Condition/Decision Coverage =

Multiple condition coverage

- The coverage domain consists of all the combinations of conditions in each decision.
- A decision with n conditions requires 2^n test cases!

```
examScore = int(input("Enter exam score: "))
attendance_rate = float(input("Enter attendance rate: "))

if examScore >= 70 and attendance_rate>0.5:
    isPass = True
else:
    isPass = False
print(isPass)
```

4 combinations of conditions
TT, TF, FT, FF

Multiple condition coverage =

Test Case	examScore	attendance_rate	Expected Result
1			
2			
3			
4			

Modified Condition/Decision (MC/DC) Coverage

Each simple condition within a compound condition C in P has been shown to independently effect the outcome of C.

```
examScore = int(input("Enter exam score: "))
attendance_rate = float(input("Enter attendance rate: "))

if examScore >= 70 and attendance_rate>0.5:
    isPass = True
else:
    isPass = False
print(isPass)
```

The decision: **examScore >= 70** and attendance_rate>0.5

To test the independent effect of the first condition, we set the second condition to be True.

Why? If we set the second condition to be false, the decision is false no matter what the first condition is True or false.

Test Case	examScore	attendance_rate	Expected Result
1	80	0.7	True
2	60	0.7	False

The decision: examScore >= 70 and **attendance_rate>0.5**

Similarly, to test the independent effect of the second condition, we set the first condition to be True.

Why? If we set the first condition to be false, the decision is false no matter what the second condition is True or false.

Test Case	examScore	attendance_rate	Expected Result
3	80	0.7	True
4	80	0.4	False

Path coverage

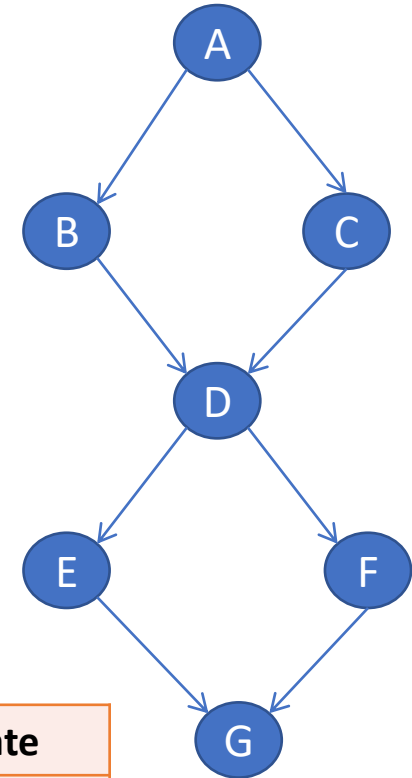
- The coverage domain consists of all paths in the control flow graph (CFG)
- May generate many test cases if the CFG is complex
 - e.g. multiple levels of nested if-else statements, loops

```
examScore = int(input("Enter exam score: "))
attendance_rate = float(input("Enter attendance rate: "))

# First independent if-else statement
if examScore >= 70:
    pass_exam = True
else:
    pass_exam = False

# Second independent if-else statement
if attendance_rate > 0.5:
    pass_attendance = True
else:
    pass_attendance = False
```

Test Case	examScore	attendance_rate
1		
2		



Loop Coverage

- Zero-pass : The loop is not executed at all.
- Single-pass : The loop is executed exactly once.
- Two-pass : The loop is executed exactly twice.
- Multi-pass : The loop is executed more than twice.

What are the test cases?

```
# input number of students
n = int(input("Enter number of students: "))

for i in range(n):
    # Input examScore
    examScore = int(input(f"Enter exam score for student {i+1}: "))
    attendance_rate = float(input(f"Enter attendance rate for student {i+1}: "))

    # Print whether the student pass/fail
    if examScore >= 70 and attendance_rate > 0.5:
        isPass = True
    else:
        isPass = False
    print(f"Student {i+1} passed? {isPass}")
```

Comparison of the coverage criteria based on control flow

- Condition, condition/decision, or decision coverage may not be able to reveal some common faults.
- Multiple condition coverage may reveal more faults, but it may generate too many test cases.
- MC/DC coverage is a weaker criterion than the multiple condition coverage criterion.
 - The number of test cases is much less than multiple condition coverage (in particular when there are many conditions in a decision), but it can detect most types of faults.