

# **D11 UNIT 5-A**

## **QUALITY**

Criteria, metrics, assurance and techniques



# Software quality

## Motivation

2

Software quality is motivated by at least two main perspectives:

- Software failure has caused more than inconvenience and software errors can cause human fatalities.
- Good quality software costs less to maintain, is easier to understand and can be changed more cost-effectively in response to pressing business needs.

# Software quality

## Standards

3

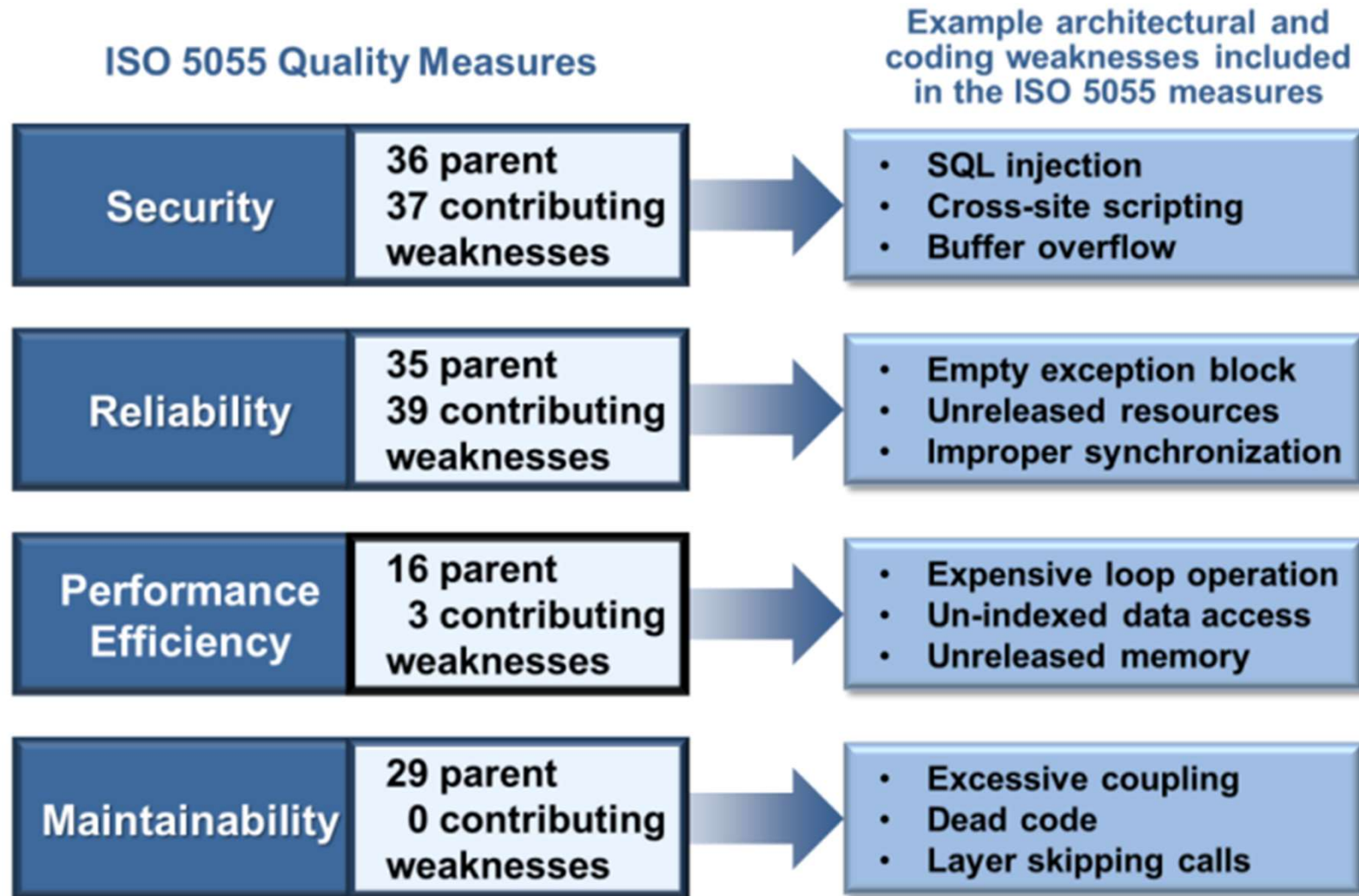
ISO/IEC 5055:2021 (which will be referred to as ISO 5055 hereafter) is an ISO standard for measuring the internal structure of a software product on four business-critical factors:

- Security
- Reliability
- Performance efficiency
- Maintainability

# Software quality

## Standards

4



# Software quality Standards

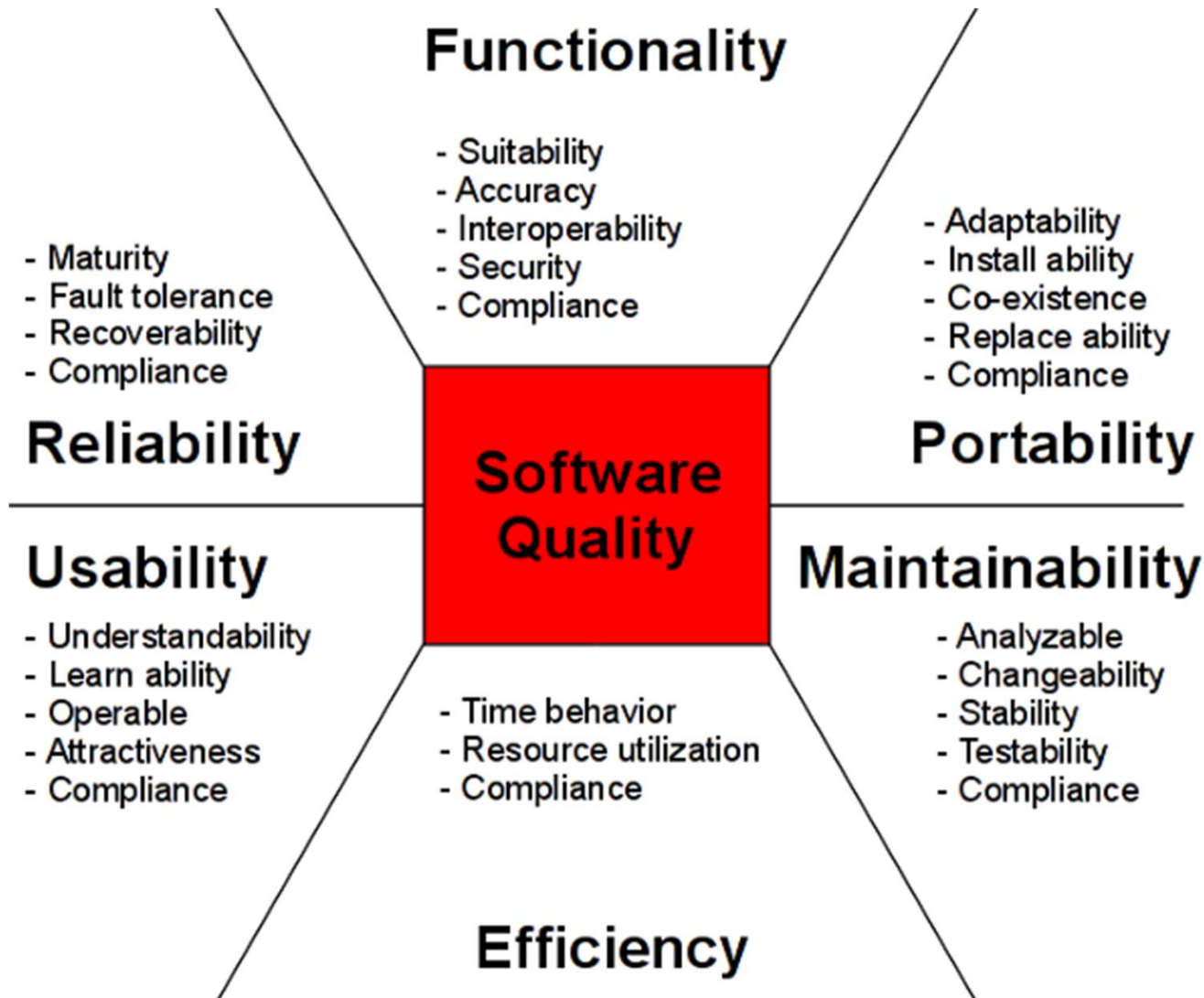
5



# Software quality

## Standards

6



# Software quality

## Desired characteristics

7

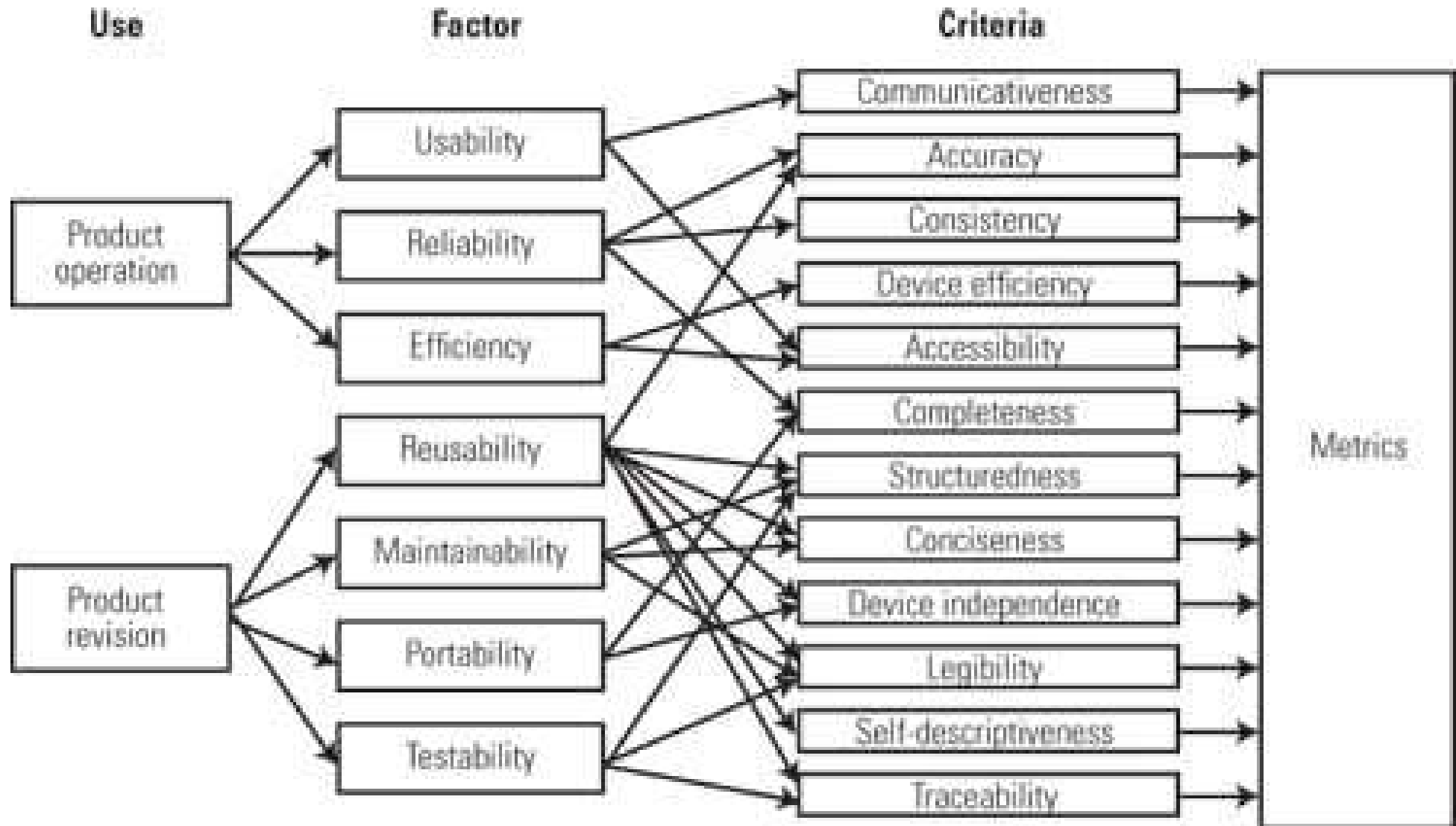
The Consortium for IT Software Quality (CISQ) has defined five major desirable structural characteristics needed for a piece of software to provide business value:

- Reliability
- Efficiency
- Security
- Maintainability
- (adequate) Size.

# Software quality

## Uses, factors and criteria

8





# Software quality factors

## Usability

9

- Usability can be described as the capacity of a system to provide a condition for its users to perform the tasks safely, effectively, and efficiently while enjoying the experience.



# Software quality factors

## Reliability

10

- Software reliability is the probability of failure-free operation of a computer program for a specified period in a specified environment.



# Software quality factors

## Efficiency

11

- Efficiency is the software ability to provide appropriate performance, relative to the amount of resources used, under stated conditions at a given point in time.

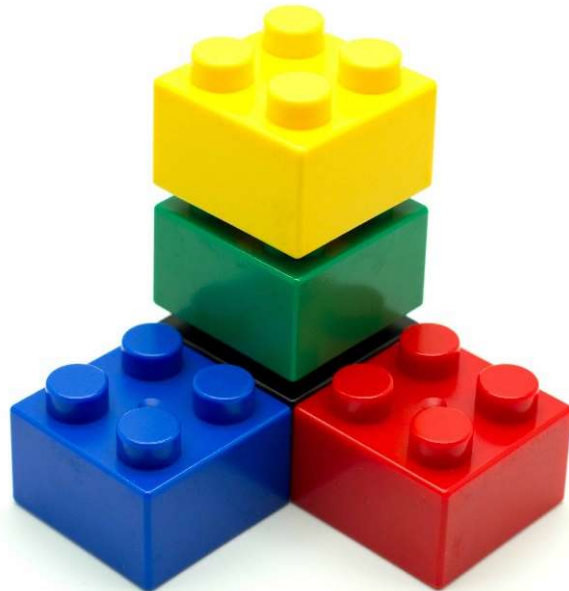


# Software quality factors

## Reusability

12

- Reusability is the capacity to repurpose pre-existing code when developing new software applications.



# Software quality factors

## Maintainability

13

- Software maintainability is defined as the degree to which an application is understood, repaired, or enhanced.



# Software quality factors

## Portability

14

- Software portability is a characteristic attributed to a computer program if it can run with minimal rework on operating systems (OSes) other than the one for which it was created.



# Software quality factors

## Testability

15

- Software testability is the degree to which a software artifact (i.e. a software system, software module, requirements- or design document) supports testing in a given test context.



# Dynamic analysis

## Golden rules

16



### Strive for consistency

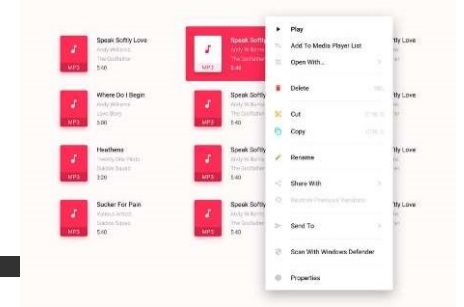
- Familiar icons, colors, menu hierarchy, call-to-actions, sequences of actions, capitalization, fonts...
- Users are able to apply knowledge from one click to another.
- Identical terminology
- Exceptions, such as required confirmation of the delete command or no echoing of passwords, should be comprehensible and limited in number



# Dynamic analysis

## Golden rules

17



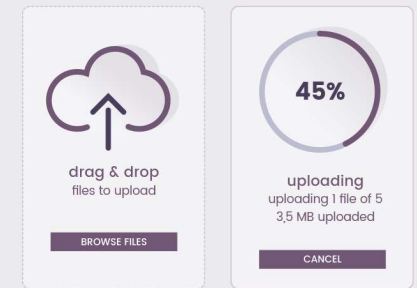
### Seek universal usability

- Novice to expert differences, age ranges, disabilities, international variations, and technological diversity each enrich the spectrum of requirements that guides design.
- Adding features for novices, such as explanations, and features for experts, such as shortcuts and faster pacing.
- Enable frequent users to use shortcuts. With increased use comes the demand for quicker methods of completing tasks.

# Dynamic analysis

## Golden rules

18



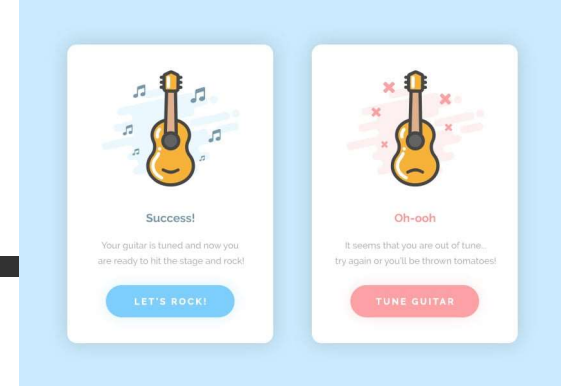
### Offer informative feedback

- For every user action, there should be an appropriate, human-readable interface feedback.
- For frequent and minor actions, the response can be modest.
- For infrequent and major actions, the response should be more substantial.
- Visual presentation of the objects of interest provides a convenient environment for showing changes explicitly.

# Dynamic analysis

## Golden rules

19



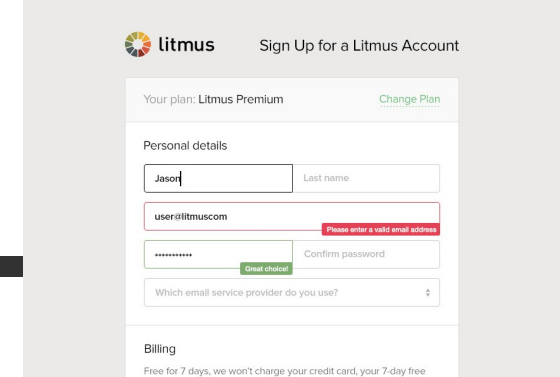
### Design dialogue to yield closure

- Sequences of actions should be organized into groups with a beginning, middle, and end.
- Informative feedback at the completion of a group of actions gives users the satisfaction of accomplishment.
- Don't keep your users guessing. Tell them what their action has led them to.

# Dynamic analysis

## Golden rules

20



The screenshot shows the Litmus sign-up interface. At the top, it says 'litmus' and 'Sign Up for a Litmus Account'. Below this, it indicates 'Your plan: Litmus Premium' with a 'Change Plan' link. The 'Personal details' section contains several input fields: 'First name' (with 'Jasor' entered), 'Last name', 'Email' (with 'user@litmus.com' entered), and 'Password' (with '\*\*\*\*\*' entered). A red error message 'Please enter a valid email address' is displayed below the email field. A green 'Great choice!' message is shown below the password field. There is also a 'Confirm password' field and a dropdown for 'Which email service provider do you use?'. At the bottom, the 'Billing' section states 'Free for 7 days, we won't charge your credit card, your 7-day free'.

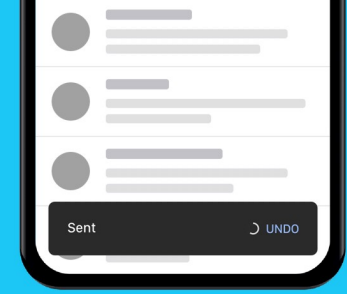
### Offer simple error handling (prevent errors)

- As much as possible, design the interface so that users cannot make serious errors.
- If users make an error, the interface should offer simple, constructive, and specific instructions for recovery.
- Erroneous actions should leave the interface state unchanged, or the interface should give instructions about restoring the state.

# Dynamic analysis

## Golden rules

21



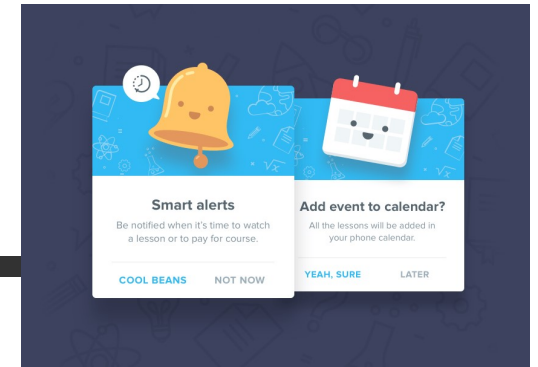
### Permit easy reversal of actions

- As much as possible, actions should be reversible.
- This feature relieves anxiety, since users know that errors can be undone, and encourages exploration of unfamiliar options.
- The units of reversibility may be a single action, a data-entry task, or a complete group of actions, such as entry of a name-address block.
- Designers should aim to offer users obvious ways to reverse their actions. These reversals should be permitted at various points whether it occurs after a single action, a data entry or a whole sequence of actions.

# Dynamic analysis

## Golden rules

22



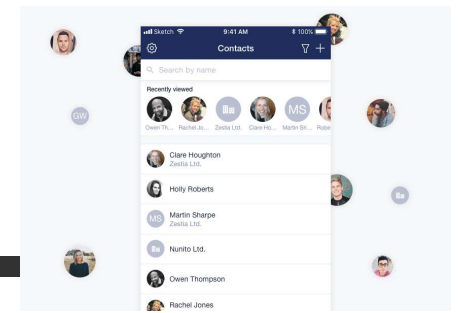
### Keep users in control (support internal locus of control)

- Experienced users strongly desire the sense that they are in charge of the interface and that the interface responds to their actions.
- They don't want surprises or changes in familiar behavior, and they are annoyed by tedious data-entry sequences, difficulty in obtaining necessary information, and inability to produce their desired result. Avoid surprises, interruptions, or anything that hasn't be prompted by the users.
- Allow your users to be the initiators of actions.
- In personality psychology, locus of control is the degree to which people believe that they have control over the outcome of events

# Dynamic analysis

## Golden rules

23



### Reduce short-term memory load

- Humans' capacity for information processing in short-term memory is limited (people can remember “seven plus or minus two chunks” of information).
- This requires that designers avoid interfaces in which users must remember information from one display and then use that information on another display.
- Choose recognition over recall.

# Quality assessment

## Static analysis

24

- Static analysis involves examining the code without actually executing it. This analysis is performed on the source code or, in some cases, on the compiled version of the code, to identify potential issues.
- Its primary goal is to detect coding errors, security vulnerabilities, style issues, and other defects early in the development cycle.
- Techniques include code reviews, automated code analysis tools, and model checking. Tools used for static analysis can scan the entirety of the codebase for issues such as syntax errors, type mismatches, memory leaks, and more.



# Quality assessment

## Static analysis

25

- Advantages: Can be performed early in the software development life cycle; doesn't require the code to be running; can potentially examine all paths through the code.
- Limitations: May produce false positives or negatives; cannot detect runtime errors or performance issues; doesn't account for the interaction with specific hardware or the external environment.

# Quality assessment

## Dynamic analysis

26

- Dynamic analysis involves analyzing and evaluating the software by executing it in real-time. It aims to uncover issues that become evident only during the execution of the software.
- Used to detect runtime errors, performance issues, and other defects that cannot be identified through static analysis. It is crucial for understanding the software's behavior in a live environment.
- Techniques include unit testing, integration testing, system testing, performance testing, and using automated tools that monitor the behavior of an application during execution.

# Quality assessment

## Dynamic analysis

27

- Advantages: Can identify runtime issues, including memory leaks, performance bottlenecks, and concurrency problems; provides insights into the software's behavior in real-world scenarios.
- Limitations: Is generally more resource-intensive; cannot cover all execution paths; testing is limited by the scenarios and test cases used.

# Quality assessment

## Static vs Dynamic

28

- Focus: Static analysis focuses on code quality, security vulnerabilities, and compliance with coding standards. Dynamic analysis focuses on runtime behavior, including performance, memory usage, and handling of real-world data and conditions.
- Coverage: Static analysis can potentially cover all code paths and detect issues early in the development process. Dynamic analysis is limited by the scenarios and inputs provided during testing but offers insights into the software's actual execution and environment interactions.

# Software quality

## Functional vs structural

29

- Software functional quality reflects how well it complies with or conforms to a given design, based on functional requirements or specifications.
- Software structural quality refers to how it meets non-functional requirements that support the delivery of the functional requirements.

# Software quality

## Functional vs structural

30

Functional quality is typically assessed dynamically but it is also possible to use static tests.

Many aspects of structural quality can be evaluated only statically through the analysis of the software's inner structure, its source code.

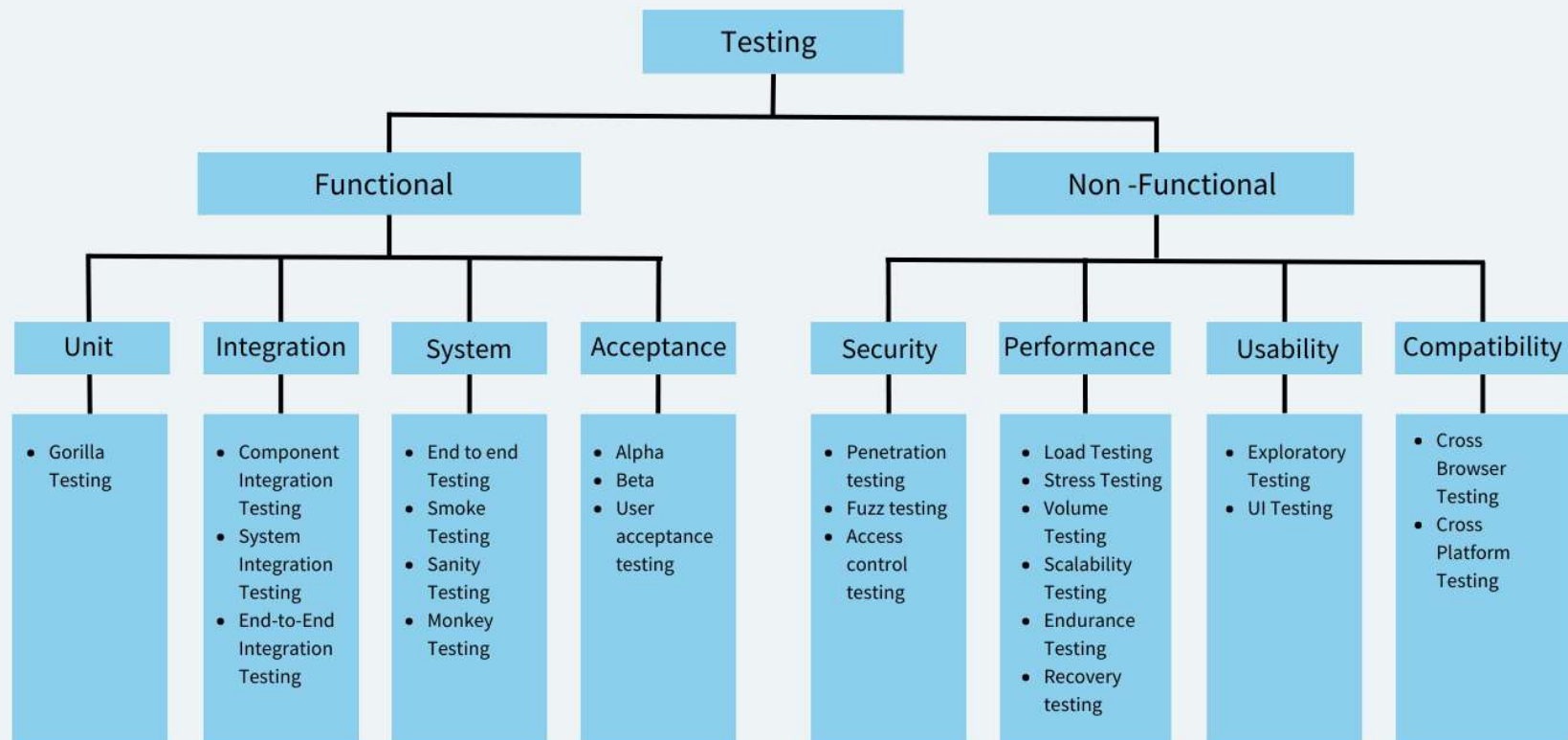
Some structural qualities, such as usability, can be assessed only dynamically.

Other aspects, such as reliability, might involve not only the software but also the underlying hardware, therefore, it can be assessed both statically and dynamically.

# Dynamic analysis

## Testing

31



# Dynamic analysis

## Testing

32

Functional testing ensures that each discrete function of a software application operates according to requirement.

Non-functional testing focuses on the software's attributes, such as security, performance, and scalability, rather than specific behaviors.

Unit testing is the process of verifying the smallest testable parts of an application, individually and independently.

Integration testing involves combining individual units of code and testing them as a group to uncover interface defects between the units.

System testing is a holistic testing process that evaluates the complete system's compliance with specified requirements.



# Dynamic analysis

## Testing

33

Incremental testing refers to testing an application step by step with incremental builds, where modules are added one by one to check for proper functionality.

Non-incremental testing is a method where the entire software system is tested as a whole after all the components are developed.

Performance testing assesses how well the software performs under specific conditions, focusing on responsiveness and stability under a particular workload.

Usability testing evaluates how user-friendly, efficient, and convenient the software is for end-users.

Compatibility testing checks if the software can run in different browsers, databases, hardware, operating systems, mobile devices, and networks.

# Dynamic analysis

## Testing

34

Load testing examines the software's behavior under normal load conditions to ensure it can handle anticipated usage.

Stress testing is used to evaluate how the software behaves under extreme conditions, often beyond its operational capacity.

Scalability testing determines if the software application can handle projected increases in user traffic or data volume.

Stability testing testing checks if the application remains stable over prolonged periods under normal load conditions.

# Static analysis

## Vulnerabilities

35

- A software vulnerability is a defect in software that could allow an attacker to gain control of a system. These defects can be because of the way the software is designed, or because of a flaw in the way that it's coded.

# Static analysis

## Technical debt

36

- Extra developmental work that arises when code that is easy to implement in the short run is used instead of applying the best overall solution.
- It occurs as the result of a "build now, fix later" mentality.
- Leaders delay features and functionality, cut corners, or settle for suboptimal performance to push the project forward.
- In software engineering, tech debt is sometimes called code debt.
- Technical debt is like a financial debt that incurs interest, making it more costly to fix the issue later than it would have been to do it right the first time.

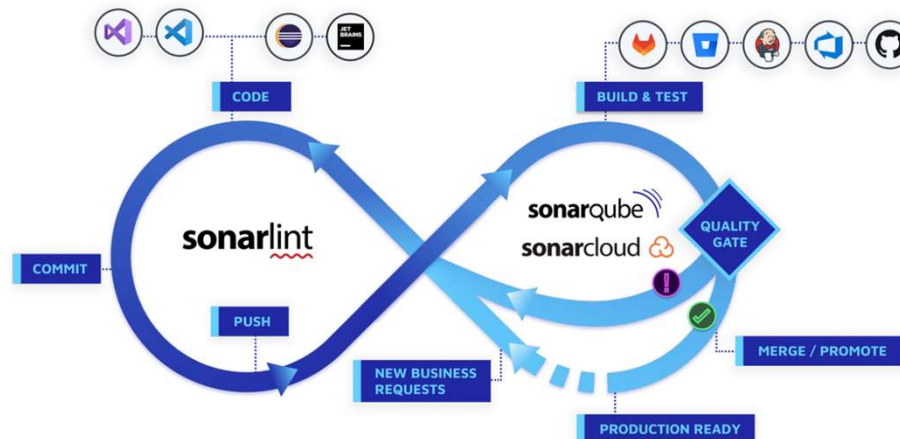
- Quantitative metric used to calculate the number of paths through the code. Whenever the control flow of a function splits, the complexity counter gets incremented by one. Each function has a minimum complexity of 1.

# Dynamic analysis

## SonarQube

38

- SonarQube executes rules on source code to generate issues. There are four types of rules:
  - Code smell (maintainability domain)
  - Bug (reliability domain)
  - Vulnerability (security domain)
  - Security hotspot (security domain-risky zone or piece of code)



# Static analysis

## Static code analysis tools

39

**sonarqube** 

 **COVERITY**<sup>®</sup>

**rax** 

**01**  
**VERACODE**

**CodeScene**<sup>™</sup>

Powered by Empear