



# Software Security

SC359

Lec1


Introduction Software Quality

By:

Email: [dr.sharaf@from-masr.com](mailto:dr.sharaf@from-masr.com)

Dr. Hussien M. Sharaf

# Course review

- ▶ Poor software design and engineering are the root causes of most security vulnerabilities in deployed systems today.
  - ▶ This course takes a close look at software as a mechanism for an attack, a tool for protecting resources, and as a resource to be defended.
- 

# Lecture outline

- A. Software quality.**
- B. Measuring software quality factors.**
- C. Measure for Integrity of software systems.**
- D. Verification, validation, and testing**

# A. Software quality

## What is quality?

- ▶ **Software quality** is the **characteristics** of a software that make it fit for its purpose.
- ▶ **A software of appropriate quality**, is in conformance or related to the requirements and expectations of the customer.

# Software quality factors

- ▶ We can characterize a **good software system** as **useful, usable, reliable, flexible, affordable and available**.
- ▶ **Software quality factors (SQFs)** are the attributes of a software product.
- ▶ **SQFs are affected by three general types of requirements:**
  - **Product operation requirements**, how the product will be used.
  - **Product revision requirements**, how the product will be changed.
  - **Product transition requirements**, how the product will be modified for different operating environments.

# Software quality factors

SQFs that are affected by *product operation requirements* are:

- ▶ **Correctness**, determines how well the software does what the customer wants;
- ▶ **Reliability**, determines how well the software does what it is supposed to do;
- ▶ **Efficiency**, determines how well the system runs on the customer's hardware;
- ▶ **Integrity**, determines how well the data is secured;
- ▶ **Usability**, determines how easy the system is to use.

# Software quality factors

SQFs that are affected by *product revision requirements* are:

- ▶ **Maintainability**, determines how easily bugs can be found and fixed;
- ▶ **Flexibility**, determines how easily the system can be changed while in service;
- ▶ **Testability**, determines how easily the system can be tested to show that the customer's requirements have been met.



# Software quality factors

**SQFs that are affected by *product transition requirements* are:**

- ▶ **Portability**, determines how easily the system can be used on another machine, platform or another OS.
- ▶ **Reusability**, determines how easy it is to reuse some of the software to make future developments more cost-effective;
- ▶ **Interoperability**, determines how easy it is to interface the system with another system.



# Software quality factors

## SAQ 1

- ▶ Identify some pairs of SQFs which are not independent. Explain how increasing integrity within a system could affect efficiency.

## ANSWER (cont.).....

- ▶ Some pairs of SQFs which are not independent :

- ▶ **Integrity and efficiency:**

**Increasing integrity within a system could affect efficiency:** Increasing integrity means making the system more secure. This might involve the use of passwords to access certain data and an authentication server to check a user's identity, or it might mean that network traffic needs to be encrypted and decrypted. Each of these factors adds overhead to processing, so efficiency is likely to be reduced.

- ▶ **Usability and portability:**

Many of the features of the Apple Macintosh that contribute to its reputation for usability are built into its operating system. Applications that take advantage of these features are less portable to other systems, such as Windows or Linux.

# Primary software quality factors

- ▶ The primary SQFs are **correctness, integrity, maintainability and usability**.
- ▶ How to measure the primary SQFs?
  1. Measure for **correctness** is **defects per thousand lines of code (defects per KLOC)**,
    - A **defect** is a verified lack of conformance to requirements.
  2. Measure for **maintainability** is **mean time to change (MTTC)**,
    - **MTTC** is the average of the times it takes to analyze a bug report, design an appropriate modification, implement the change, test it and distribute the change to all users.
    - The lower the MTTC the more maintainable the software product is.

# Primary software quality factors

3. Measure for **Integrity** is considering the propability of 'attacks' on a product. **Some examples of attacks are:**

- **Denial of service**, a legitimate user is prevented from accessing a service to which they are entitled;
- **Man in the middle and replay**, false communications between computers are generated;
- **Accidental attacks**, failures of the product or its users are responsible for the attack.
- **Input attacks**, where data is chosen to cause an application to function incorrectly.

# Primary software quality factors

## For a particular type of attack:

- ▶ **threat<sub>attack</sub>** is defined as the likelihood that an attack of this type will occur within a given time, and
- ▶ **security<sub>attack</sub>** is defined as the likelihood that an attack of this type will be repelled.
- ▶ threat<sub>attack</sub> and security<sub>attack</sub> can be given values between 0 and 1, which can be calculated from historical data, such as that contained in a log file.

- ▶ The integrity against attacks of this type, **integrity<sub>attack</sub>**, is defined to be:

$$\text{integrity}_{\text{attack}} = 1 - \text{threat}_{\text{attack}} (1 - \text{security}_{\text{attack}})$$

- ▶ The **integrity of the software product** is defined to be the sum, over all attack types, of integrity<sub>attack</sub>:

$$\text{integrity} = \sum_{\text{attack}} \text{integrity}_{\text{attack}}$$

- ▶ If there are n different types of attack, then integrity will be a value between 0 and n; **the higher the value, the better the integrity.**
- ▶ **Note** that for software connected to the Internet, the probability of many attacks is 1.

# Primary software quality factors



Universe = Attacks and no attacks




Attack that occur against the system

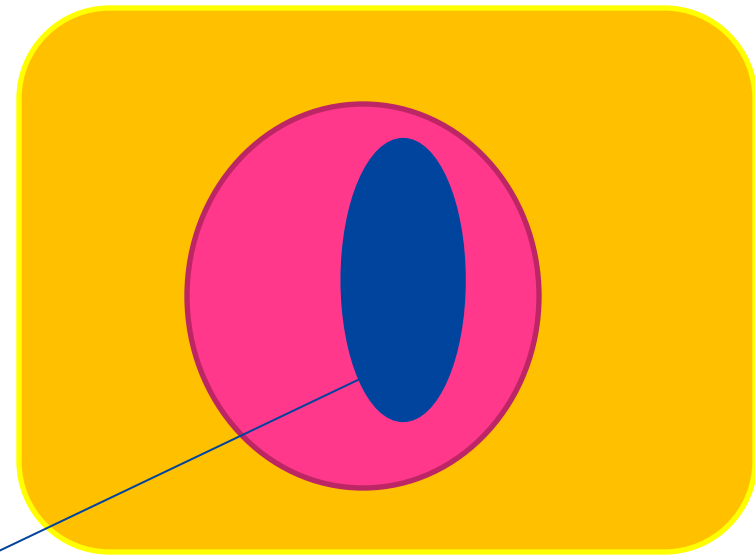


Set of attacks that are repelled

Required:

Get the sum of 

excluding  and including 



**security**<sub>attack</sub>

# Primary software quality factors

## Example 1:

- ▶ A web server has been running for a month. From the log files for that month we see that, of 2000 accesses, 100 attacks were made. Of these, 50 were denial-of-service attacks, of which 10 were successful, 25 were password guessing (of which none were successful) and 25 were accidental attacks (caused by errors on the part of the user), of which 25 were successful.
- ▶ To calculate  $\text{threat}_{\text{attack}}$  and  $\text{security}_{\text{attack}}$  for each type of attack, we can tabulate the data as in Table 1.

Attack type	Threat	Security	Integrity
Denial of service	$50/2000 = 0.025$	$40/50 = 0.8$	$1 - 0.025 * (1 - 0.8) = 0.995$
Password guessing	$25/2000 = 0.0125$	$25/25 = 1$	$1 - 0.0125 * (1 - 1) = 1$
Accidental	$25/2000 = 0.0125$	$0/25 = 0$	$1 - 0.0125 * (1 - 0) = 0.9875$

Table 1 Calculating the integrity of a web server

- ▶ The integrity of the server is thus  $0.995 + 1 + 0.9875 = 2.9825$ .

# Primary software quality factors

## SAQ 2

- ▶ Log entries in a web server indicate that, over the previous month, 113 attacks were made on the web server, but none were successful. Is the web server necessarily secure?

## ANSWER.....

- ▶ The web server repelled all attacks, so security is 1. From the formula, integrity is 1 (perfect). We get this from working out the formula:

$$\text{integrity}_{\text{attack}} = 1 - \text{threat}_{\text{attack}} (1 - \text{security}_{\text{attack}})$$

This would seem to indicate that the web server is entirely secure.

However, all we really know is that the web server was secure against these particular attacks. If the culprits were to use some other kind of attack, or change the parameters on the attack they used, they might well succeed. It is also possible that successful attacks have been made using some other method, but there were no log entries of sufficiently suspicious activity to attract our attention, perhaps because the attacker deleted them.



# Primary software quality factors

## Exercise 2

- ▶ What does 'quality' mean in contexts other than software?

## Solution .....

- ▶ You may have chosen to discuss a number of things here: a quality car service, a comparison of picture quality between DVD and video tape, the quality of services provided by local government, even the way in which light is depicted (the quality of the light) in Monet's paintings of London. Whatever you have chosen, **consider whether there are measurable attributes relating to quality, or whether quality refers to something less tangible.**

# B. Verification, validation, and testing

## Verification and validation

- ▶ **Verification and validation** are ways of assessing whether a software product does what it is supposed to do.
- ▶ **Validation**, process of checking that the developer is building the 'right' product.
- ▶ **Verification**, process of ensuring that the product being built 'right'.
- ▶ **How to carry out validation:**
  1. Make sure that all system descriptions are **consistent with the customer's requirements**.
- ▶ **How to carry out verification:**
  1. Make sure that all system descriptions are **self-consistent**;
  2. Make sure that all system descriptions are **consistent and complete** with respect to those from which they were derived.

# The development process

## ► In the development process:

- **Testing** is usually happened at late stage, so if any requirement were missing, or misunderstood, it will be costly to recover it.
  - An alternative model is The use of **prototypes**, facilitate understanding requirements and figure out any missing or wrong requirement early in the development process.
- **Verification techniques should be used during each activity in the development process** to ensure that the deliverable from each activity is consistent and complete with respect to the input to that activity.

# Testing

- ▶ **Testing** is the process of exercising software to check that it does what it is supposed to and doesn't do what it is not supposed to.
- ▶ **Testing** is one technique used for validation and verification.
- ▶ **The role of testing** is to find as many bugs as possible.
- ▶ **By finding and fixing bugs, software quality can be improved.**

# The testing process

## Four types of testing during a software development project:

1. **Usability testing**, the functionality and user interface are tested.
2. **Developmental testing**, checks that developmental activities have been carried out correctly.
  - a) **Unit testing**, units of functionality (for example, the classes in an object-oriented system) are tested in isolation;
  - b) **Integration testing**, previously tested units are tested together;
  - c) **System testing**, the completed system is tested against the customer's requirements.
3. **Requirements-based testing**, checks that a system meets the customer's requirements.
4. **Regression testing**, occurs during developmental testing and system maintenance, and checks that fixing one bug has not introduced others.

# Unit testing

- ▶ Unit testing is performed on classes.
- ▶ The unit is tested using the **test data**, and **test results** are collected. By comparison with the **criteria**, the results can be analyzed to reveal the presence of **bugs**.
- ▶ For a particular class, unit testing should consist of five parts:
  1. **interface testing**, where test data is chosen to check that the flow of information into and out of an object of the class is correct;
  2. **boundary-condition testing**, where test data are chosen to check that methods perform correctly at the extremities of input ranges;

# Integration testing

- ▶ **Integration testing** involves checking that unit-tested classes interface correctly together.
- ▶ **One major problem with integration testing** is that **polymorphism and inheritance** can greatly complicate testing.



# System testing

- ▶ **System testing** consists of checking that a completed software system performs in accordance with its requirements specification.

## System testing consist of the following generic tests:

- ▶ **start-up and initialization testing:** tests the system's ability to be started in a working hardware/software configuration;
- ▶ **performance testing:** tests that the system meets all specified operating requirements for speed, number of concurrent users permitted, etc.;
- ▶ **stress testing:** tests that the system can operate reliably at the limits of each of its resources (for example, test a web server with thousands of users accessing it all at the same time to see if it can cope with the load);
- ▶ **security testing:** tests that the system does not offer opportunities to breach security.

# System testing

- ▶ A **system test matrix** is an important document in **system testing**, as it relates classes/packages that will be executed when the various system tests are carried out.

# Acceptance testing

- ▶ Is the formal testing of an entire system, conducted by a customer, to **determine whether it satisfies its acceptance criteria** and thus whether the customer should accept the system.

## SAQ 5

- ▶ What do you think is the relationship between system and acceptance testing?

## ANSWER.....

- ▶ In general, the same tests will be carried out during acceptance testing and system testing.
- ▶ **System testing** is an in-house activity and a customer need never know how system testing went; any bugs can be dealt with before the customer sees them.
- ▶ **Acceptance testing**, on the other hand, is conducted with much more at stake; the customer can accept or reject a system based on its performance at acceptance testing.

# Regression testing

- ▶ Regression testing is one of the four major types of software testing.
- ▶ It checks that fixing one bug has not introduced others.
- ▶ Occurs during developmental testing and system maintenance.

## SAQ 6

- ▶ Why should regression testing be necessary, since, after all, the customer has accepted the product after acceptance testing?

## ANSWER.....

- ▶ Acceptance testing is the process of showing that the software meets the customer's requirements, not that there aren't bugs in the code. In fact, given that a system is put into use, bugs that require fixing are almost certain to be found after acceptance testing. In addition, the system will be maintained, adding and changing functionality as needed. Regression testing is therefore necessary.

# Regression testing

## Exercise 2

- ▶ Which of unit, integration, system and acceptance testing are parts of validation and which are parts of verification?

## Solution .....

- ▶ **Unit and integration testing** concentrate on whether parts of the system perform according to their specifications, answering the **verification** question (have we built the system correctly?).
- ▶ **System and acceptance testing** concentrate on showing that the customer's requirements have indeed been met, answering the **validation** question (have we built the right system?).
- ▶ **Note**, however, that there are no hard and fast distinctions. Unit testing can be used to demonstrate that a component satisfies a customer's requirements – thus it can be viewed as validation, and system testing can be used to demonstrate that a system operates according to specification – in which case it can be viewed as verification.