SOFTWARE QUALITY ATTRIBUTES AS DESIGN DRIVERS

Dr. Tony D. Barber / Prof. Sahana Kini

Fall, 2022

TODAY'S CLASS OBJECTIVES AND OUTCOME

- Define what software quality attributes and scenarios are
- Explain the importance and relation between quality and design
- Represent software quality scenarios
- Identify software characteristics trade-offs
- Prioritize quality scenarios
- Define characteristics of security and usability
- Explain how to build-in security during software lifecycle
- Describe the next assignment for the project

OUTLINE

We are here

- Lecture
  - Software quality attributes and non-functional requirements
  - Quality scenarios
  - Utility tree
    - Class exercise – start project LMS utility tree
  - Security in the software life cycle
    - Class exercise – start project LMS abuse cases
- Assignments
  - Requirements Analysis Quiz
  - Software development project assignment

RESOURCES

- Textbook – Chapter 3, 9
- *Software Architecture in Practice,* by Len Bass and Paul Clements
- Quality Attribute Scenarios in Practice

- *Using Abuse Case Models for Security Requirements Analysis*, by John McDermott and Chris Fox

https://www.acsac.org/1999/papers/wed-b-1030-john.pdf

OUTLINE

- Lecture
  - We are here
  - Software quality attributes and non-functional requirements
  - Quality scenarios
  - Utility tree
    - Class exercise – start project LMS utility tree
  - Security in the software life cycle
    - Class exercise – start project LMS abuse cases
- Assignments
  - Requirements Analysis Quiz
  - Software development project assignment

REQUIREMENTS TYPES

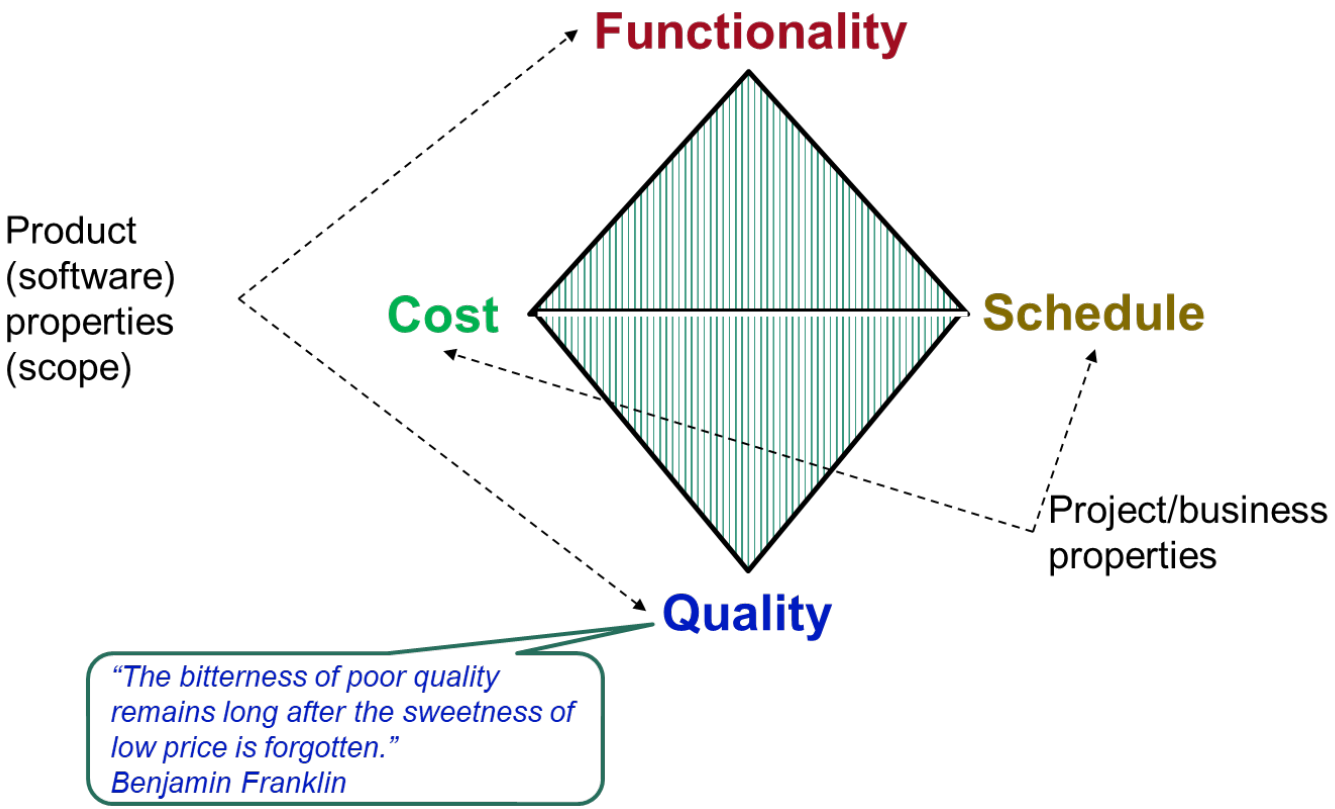
- **Functional requirement**:
  - describes required behavior in terms of required activities.
- **Quality requirement (nonfunctional requirement)**:
  - describes some quality characteristics that system must posses.
- **Design constraint**:
  - a design decision such as a choice of platform or interface components.
- **Process constraint**:
  - a restriction on the techniques or resources that can be used to build the system.
- **Development and maintenance constraint:**
  - describes required quality control procedures, priorities of implementation.

SOFTWARE STAKEHOLDERS

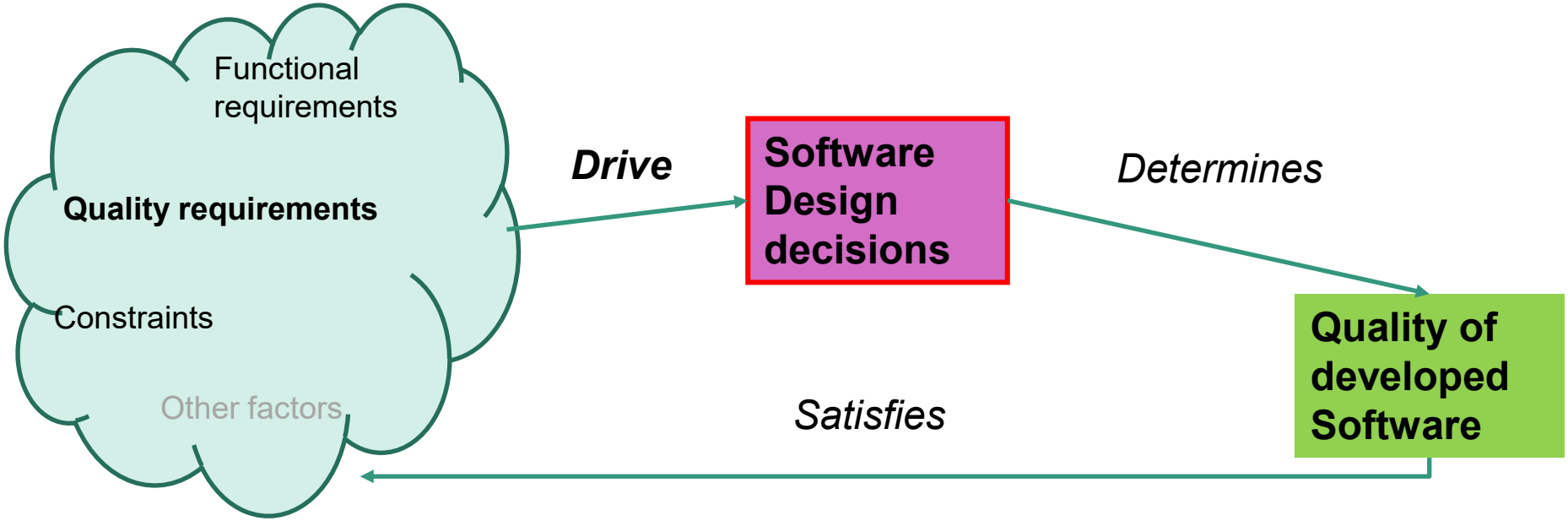
- Stakeholder -
  - Any person or organization, who
    - Can be positively or negatively impacted by the software, or
    - Can cause an impact on the software development, operation, evolution, acquisition
- Who are the stakeholders?
  - Customer
  - User
  - Developer (business analyst, designer, programmer, tester, quality assurance, marketing, legal)
  - Vendor
  - Integrator
- Stakeholders can have similar or conflicting needs, interests, and viewpoints
  - Remember “point of view” in critical thinking?
- **Elicit, analyze, balance, and prioritize *functional* and *non-functional (quality)* needs of all key stakeholders**

STAKEHOLDERS NEEDS

Balance and Trade-off



WHY CONSIDER SOFTWARE QUALITY IN SOFTWARE DESIGN?

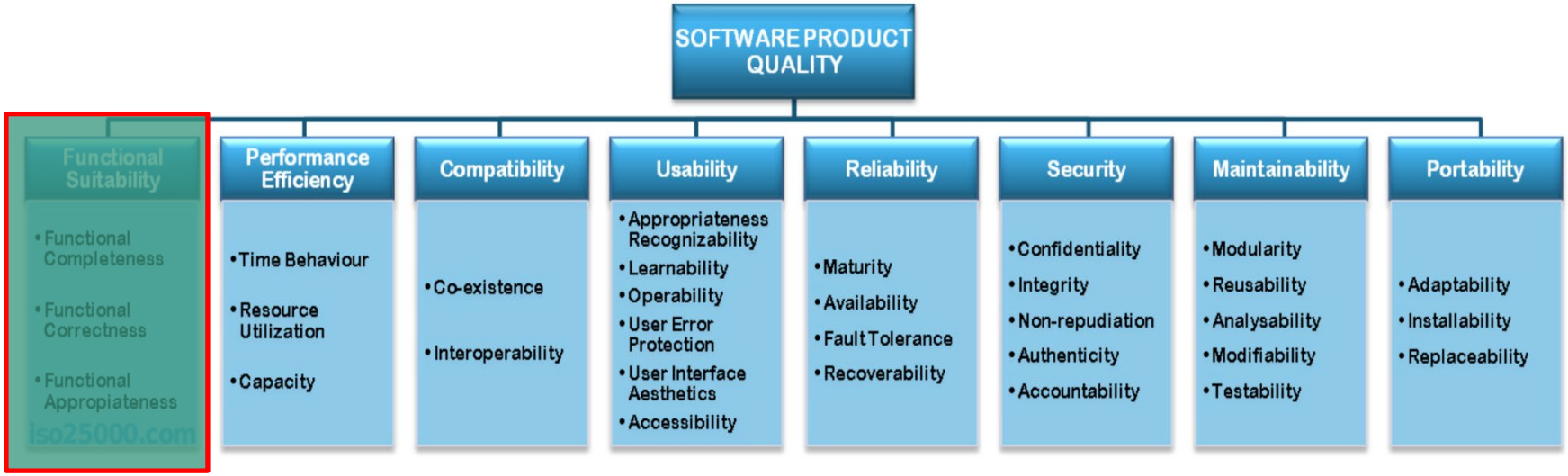


HOW ARE REQUIREMENTS ADDRESSED IN ARCHITECTURE DESIGN?

- *Functional requirements*
  - By assigning appropriate responsibilities to software pieces/parts/components throughout the design
- *Quality requirements*
  - By the various structures designed into the architecture, and
  - The behaviors and interactions of the elements that populate those structures
- *Design, technology, process, constraints*
  - Restrict or impose design decisions

SOFTWARE QUALITY

- Characteristics (or ***attributes***) that the system must posses



ISO/IEC 25010:2011 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models
From: http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

There might be some variations between different definitions of the software quality attributes and the way of organizing them (quality "frameworks", "models", …)

ANOTHER QUALITY ATTRIBUTES FRAMEWORK

- System/product/software **quality attributes**
  - Availability
  - Performance
  - Security
  - Usability
  - Reliability
  - Safety
  - …
  - Modifiability
  - Testability
  - Interoperability
  - Scalability
  - Deployability
  - …

Operational

Development / Evolution

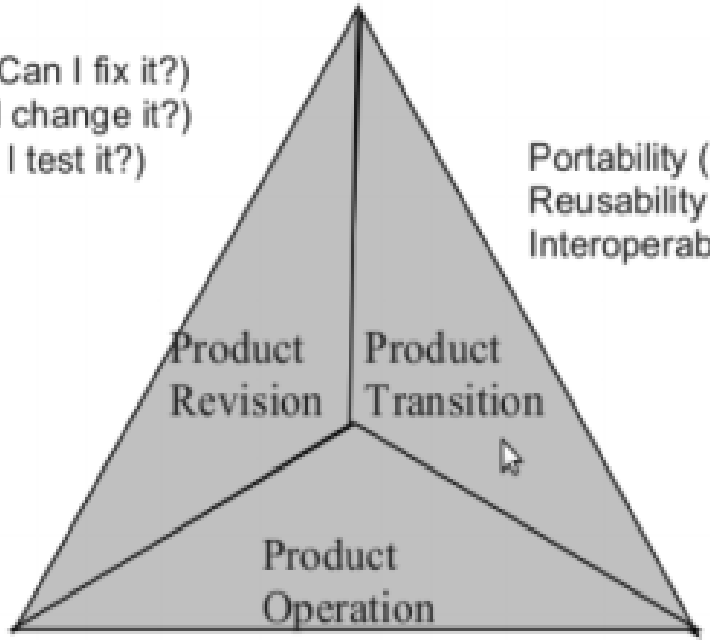
- Business characteristics
  - Time to market
  - Cost
  - Projected lifetime
  - Targeted market
  - Roll-out schedule
  - Integration with legacy systems

Constraints

[From L. Bass and P. Clements, *Software Architecture in Practice*]

ANOTHER QUALITY ATTRIBUTES FRAMEWORK

Maintainability (Can I fix it?)
Flexibility (Can I change it?)
Testability (Can I test it?)

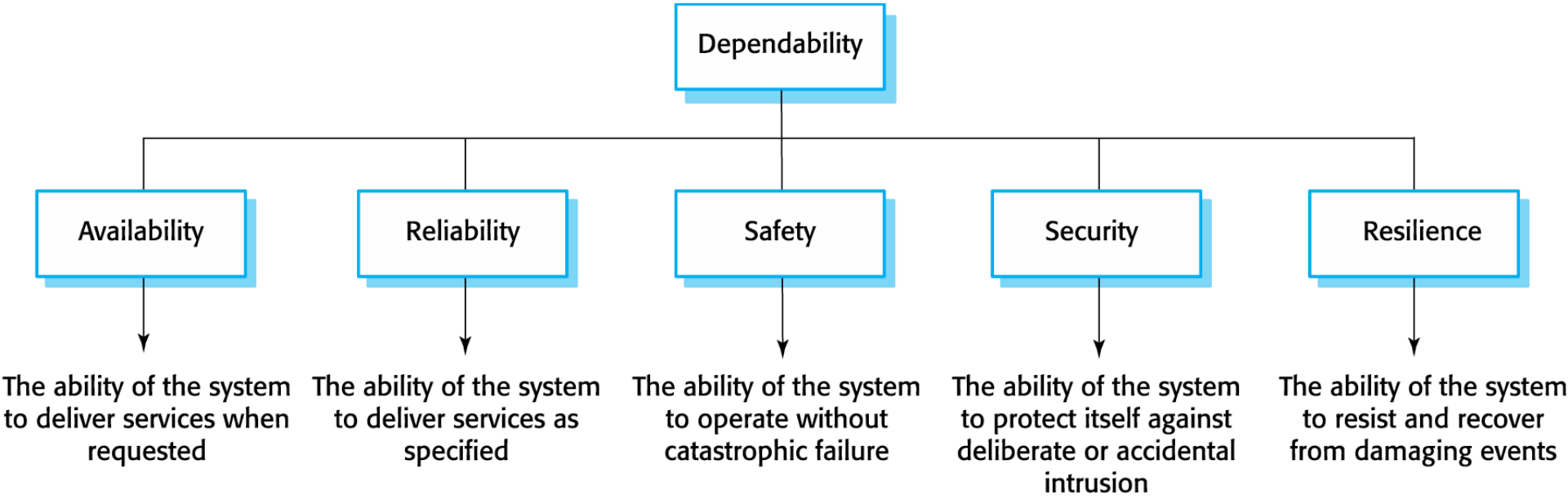


Portability (Will I be able to use it on another machine?)
Reusability (Will I be able to reuse some of the software?)
Interoperability (Will I be able to interface it with another system?)

Correctness (Does it do what I want?)
Reliability (Does it do it accurately all of the time?)
Efficiency (Will it run on my hardware as well as it can?)
Integrity (Is it secure?)
Usability (Is it designed for the user?)

McCall's software quality factors

DEPENDABILITY ATTRIBUTE(S)/PROPERTIES



[From: Software Engineering 10th Edition by Ian Sommerville – Ch10..13]

QUALITY ATTRIBUTES DEFINITION

- *Performance*—Ability to accomplish product functions within time or resource limits
- *Usability* - the degree to which software can be used by specified consumers to achieve quantified objectives with effectiveness, efficiency, and satisfaction in a quantified context of use
- *Availability*—Readiness for use
- *Reliability*—Ability to behave in accord with requirements under normal operating conditions
- *Security*—Ability to resist being harmed or causing harm by hostile acts or influences
- *Maintainability*—Ease with which a product can be corrected, improved, or ported
- *Reusability*—Degree to which a product's parts can be reused in another product
- *Safety* - the control of recognized hazards in order to achieve an acceptable level of risk

QUALITY ATTRIBUTES DEFINITION (CONTINUED)

- *Scalability*
  - Ability to keep the application online and functioning, for:
    - A large and growing number of customers
    - A large and growing quantity of data used by customers
    - A growing complexity in what customers want to accomplish with the application.
  - Ability to add more developers working on the application (as the company's needs expand)
    - Must do so without sacrificing development speed, efficiency, or application quality

QUALITY ATTRIBUTES RELATIONS

- Some quality attributes support each other, while others are in conflict/compete

	Availability	Efficiency	Flexibility	Integrity	Interoperability	Maintainability	Portability	Reliability	Reusability	Robustness	Testability	Usability
Availability								+		+		
Efficiency			-		-	-	-	-		-	-	-
Flexibility		-		-		+	+	+		+		
Integrity		-			-				-		-	-
Interoperability		-	+	-			+					
Maintainability	+	-	+					+			+	
Portability		-	+		+	-			+		+	-
Reliability	+	-	+			+				+	+	+
Reusability		-	+	-				-			+	
Robustness	+	-						+				+
Testability	+	-	+			+		+				+
Usability		-								+	-	

Need to trade-off and prioritize

+ means “support”
- means “conflict or compete”
Blank means “no relation”

EXAMPLE: SECURITY AND USABILITY

- Traditionally, security and usability were considered in conflict

- However, a solution that promotes both security and usability must be developed

SOFTWARE NON-FUNCTIONAL REQUIREMENTS

- Capture the *quality* aspect of software
- Specify *how well* the product does what it does
- Can be grouped under categories – corresponding to *quality attributes*
- Also known as “ilities”
- Can positively or negatively impact each other
  - (+) Security and Safety e.g., Stuxnet [The Real Story of Stuxnet]
  - (-) Security and Usability e.g., password rules

SOFTWARE NON-FUNCTIONAL REQUIREMENTS

- ***Quality scenarios*** are **a** format (not the only one) for specifying quality requirements
  - Associated with quality attributes
  - ***Quality attribute scenarios*** are used to specify **quality attribute requirements**
- **Use cases (and use case scenarios)** are essential in determining **functional requirements;** similarly**, quality scenarios** determine **quality requirements**

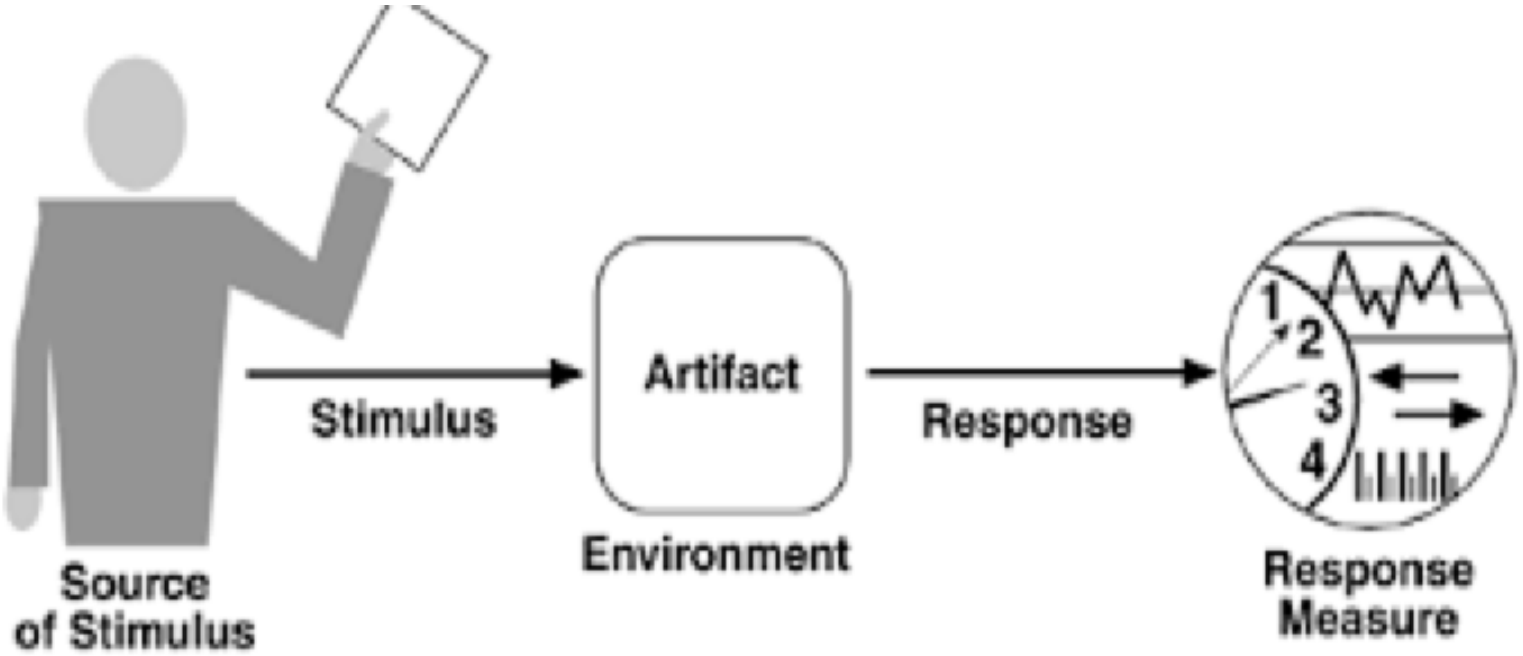
OUTLINE

- Lecture
  - Software quality attributes and non-functional requirements
  - Quality scenarios
  - Utility tree
    - Class exercise – start project LMS utility tree
  - Security in the software life cycle
    - Class exercise – start project LMS abuse cases
- Assignments
  - Requirements Analysis Quiz
  - Software development project assignment

We are here

SOFTWARE QUALITY SCENARIO

Software Engineering Institute (SEI) systematic specification of quality scenarios



[From: *Software Architecture in Practice*, by Len Bass, Paul Clements, Rick Kazman]

SOFTWARE QUALITY SCENARIO ELEMENTS AND TEMPLATE

- **Source** of stimulus
  - *This is some entity (a human, a computer system, or any other actuator) that generated the stimulus.*
- **Stimulus**
  - *The stimulus is a condition that needs to be considered when it arrives at a system.*
- **Environment**
  - *The stimulus occurs within certain conditions. The system may be in an overload condition or may be running when the stimulus occurs, or some other condition may be true.*
- **Artifact**
  - *Some artifact is stimulated. This may be the whole system or some pieces of it.*
- **Response**
  - *The response is the activity undertaken after the arrival of the stimulus.*
- **Response measure**
  - *When the response occurs, it should be measurable in some fashion so that the requirement can be tested.*

Using the template helps specifying all important elements of a scenario, in a systematic way

SECURITY GENERAL SCENARIO

Portion of Scenario	Possible Values
Source	Individual or system that is
	correctly identified, identified incorrectly, of unknown identity
	who is
	internal/external, authorized/not authorized
	with access to
	limited resources, vast resources
Stimulus	Tries to
	display data, change/delete data, access system services, reduce availability to system services
Artifact	System services; data within system
Environment	Either
	online or offline, connected or disconnected, firewalled or open
Response	Authenticates user; hides identity of the user; blocks access to data and/or services; allows access to data and/or services; grants or withdraws permission to access data and/or services; records access/modifications or attempts to access/modify data/services by identity; stores data in an unreadable format; recognizes an unexplainable high demand for services, and informs a user or another system, and restricts availability of services
Response Measure	Time/effort/resources required to circumvent security measures with probability of success; probability of detecting attack; probability of identifying individual responsible for attack or access/modification of data and/or services; percentage of services still available under denial-of-services attack; restore data/services; extent to which data/services damaged and/or legitimate access denied

[From: *Software Architecture in Practice*, by Len Bass, Paul Clements, Rick Kazman]

SECURITY CONCRETE SCENARIO EXAMPLE

Scenario name: Unauthorized data modification attempt

Source: An authorized user

Stimulus: Tries to modify information outside his/her privileges

Artifact: Data repository

Environment: Normal operation

Response: Denies access and records transaction attempt in audit trail

Response measure: Data integrity is unaffected 98% of the time

More structured

Less structured

Equivalent non-functional requirement: Under normal operation, when an authorized user attempts to modify data outside his/her privileges, in 98% of the cases, the software shall deny access and record transaction attempt in audit trail.

USABILITY GENERAL SCENARIO

Portion of Scenario	Possible Values
Source	End user
Stimulus	Wants to learn system features; use system efficiently; minimize impact of errors; adapt system; feel comfortable
Artifact	System
Environment	At runtime or configure time
Response	System provides one or more of the following responses: to support "learn system features" help system is sensitive to context; interface is familiar to user; interface is usable in an unfamiliar context to support "use system efficiently": aggregation of data and/or commands; re-use of already entered data and/or commands; support for efficient navigation within a screen; distinct views with consistent operations; comprehensive searching; multiple simultaneous activities to "minimize impact of errors": undo, cancel, recover from system failure, recognize and correct user error, retrieve forgotten password, verify system resources to "adapt system": customizability; internationalization to "feel comfortable": display system state; work at the user's pace
Response Measure	Task time, number of errors, number of problems solved, user satisfaction, gain of user knowledge, ratio of successful operations to total operations, amount of time/data lost

[From: *Software Architecture in Practice*, by Len Bass, Paul Clements, Rick Kazman]

USABILITY SCENARIO EXAMPLE 1

Scenario name: Effective search

Source: A user

Stimulus: Searches for specific information on the site

Artifact: Software web site

Environment: At runtime

Response: Displays retrieved information

Response measure: With 80% accuracy

Equivalent non-functional requirement: When a user searches for specific information on the web site, the system shall display that information with 80% accuracy.

USABILITY SCENARIO EXAMPLE 2

Scenario name: Error prevention

Source: A user

Stimulus: Initiates an action that alters stored data

Artifact: Software application (executable) UI

Environment: At runtime

Response: Requires actions confirmation, with a clear message

Response measure: 95% of the times when a data altering action is initiated by the user

Equivalent non-functional requirement: In 95% of the times when a data altering action is initiated by the user, the system shall display a clear message to require action confirmation.

USABILITY SCENARIO EXAMPLE 3

Scenario name: Learnability

Source: A user that is familiar with online shopping

Stimulus: Uses the X online shopping system for the first time

Artifact: Online shopping software application X's UI

Environment: At runtime

Response: User learns how to use the system X

Response measure: The time for a user familiar with online shopping to learn how to use system X is less than 15 minutes

Equivalent non-functional requirement: The time for any user familiar with online shopping to learn how to use system X shall be less than 15 minutes.

AVAILABILITY GENERAL SCENARIO

Portion of Scenario	Possible Values
Source	Internal to the system; external to the system
Stimulus	Fault: omission, crash, timing, response
Artifact	System's processors, communication channels, persistent storage, processes
Environment	Normal operation; degraded mode (i.e., fewer features, a fall back solution)
Response	System should detect event and do one or more of the following: record it notify appropriate parties, including the user and other systems disable sources of events that cause fault or failure according to defined rules be unavailable for a prespecified interval, where interval depends on criticality of system continue to operate in normal or degraded mode
Response Measure	Time interval when the system must be available Availability time Time interval in which system can be in degraded mode Repair time

[From: *Software Architecture in Practice,* by Len Bass, Paul Clements, Rick Kazman]

AVAILABILITY SCENARIO EXAMPLE

Scenario name: Receive unanticipated message

Source: An external system

Stimulus: Unanticipated message

Artifact: Process

Environment: Normal operation

Response: Inform operator and continue to operate

Response measure: No Downtime

Equivalent non-functional requirement: Under normal operation, when receiving an unanticipated message from an external system, the software shall inform the operator and shall continue to operate with no downtime.

PERFORMANCE GENERAL SCENARIO

Portion of Scenario	Possible Values
Source	One of a number of independent sources, possibly from within system
Stimulus	Periodic events arrive; sporadic events arrive; stochastic events arrive
Artifact	System
Environment	Normal mode; overload mode
Response	Processes stimuli; changes level of service
Response Measure	Latency, deadline, throughput, jitter, miss rate, data loss

[From: *Software Architecture in Practice,* by Len Bass, Paul Clements, Rick Kazman]

PERFORMANCE SCENARIO EXAMPLE

Scenario name: Timely computation

Source: Data feed

Stimulus: Change of input data values

Artifact: Simulation algorithm

Environment: Normal operation

Response: Computes and displays updated graph

Response measure: Within 2 seconds

Equivalent non-functional requirement: When values of input data change, the simulation algorithm shall compute and display updated graph within 2 seconds.

MODIFIABILITY GENERAL SCENARIO

Portion of Scenario	Possible Values
Source	End user, developer, system administrator
Stimulus	Wishes to add/delete/modify/vary functionality, quality attribute, capacity
Artifact	System user interface, platform, environment; system that interoperates with target system
Environment	At runtime, compile time, build time, design time
Response	Locates places in architecture to be modified; makes modification without affecting other functionality; tests modification; deploys modification
Response Measure	Cost in terms of number of elements affected, effort, money; extent to which this affects other functions or quality attributes

[From: *Software Architecture in Practice,* by Len Bass, Paul Clements, Rick Kazman]

MODIFIABILITY SCENARIO EXAMPLE

Scenario name: Deploy patches

Source: Sys Admin

Stimulus: Wants to update the code

Artifact: Application (executable)

Environment: During maintenance

Response: Successfully deploys a patch

Response measure: Within 10 minutes

Equivalent non-functional requirement: During maintenance, a sys admin shall update the code within 10 minutes.

TESTABILITY GENERAL SCENARIO

Portion of Scenario	Possible Values
Source	Unit developer
	Increment integrator
	System verifier
	Client acceptance tester
	System user
Stimulus	Analysis, architecture, design, class, subsystem integration completed; system delivered
Artifact	Piece of design, piece of code, complete application
Environment	At design time, at development time, at compile time, at deployment time
Response	Provides access to state values; provides computed values; prepares test environment
Response Measure	Percent executable statements executed
	Probability of failure if fault exists
	Time to perform tests
	Length of longest dependency chain in a test
	Length of time to prepare test environment

[From: *Software Architecture in Practice,* by Len Bass, Paul Clements, Rick Kazman]

TESTABILITY SCENARIO EXAMPLE

Scenario name: Code coverage

Source: Tester

Stimulus: Runs test cases

Artifact: Simulation algorithm

Environment: During unit testing

Response: Achieves condition coverage

Response measure: Within 90%

Equivalent non-functional requirement: When running unit test cases on the simulation algorithm, the tester achieves 90% condition coverage.

EACH SYSTEM HAS SPECIFIC SCENARIOS

- Note: For the class project - express quality attributes scenarios in terms of **specific** and **concrete** anticipated/potential stimuli and responses, e.g.
  - **Modifiability**
    - Stimulus: Add a different type of user
    - Response: Developer makes changes to the software and the UI
    - Response measure: Changes take less than 20person hours of effort
  - **Portability**
    - Stimulus: Port application to a different operating system or browser
    - Response: Software executes according to its requirements on the new OS (browser)
    - Response measure: All software functionality is executed correctly; the UI looks the same on all browsers
  - **Scalability**
    - Stimulus: Add 1000 new users
    - Response: All system features are successfully executed
    - Response measure: Each user will see a delay no longer than 2 seconds

OUTLINE

- Lecture
  - Software quality attributes and non-functional requirements
  - Quality scenarios
  - Utility tree
    - Class exercise – start project LMS utility tree
  - Security in the software life cycle

We are here

    - Class exercise – start project LMS abuse cases
- Assignments
  - Requirements Analysis Quiz
  - Software development project assignment

THE UTILITY OF A SCENARIO

- ***Utility*** – a measure of the value (importance) of the scenario to stakeholders (user, customer)
- Similar to utility (importance) of features or functional requirements
- Scenario *utility* to its user
  - High/Medium/Low or
  - Scale 1-3

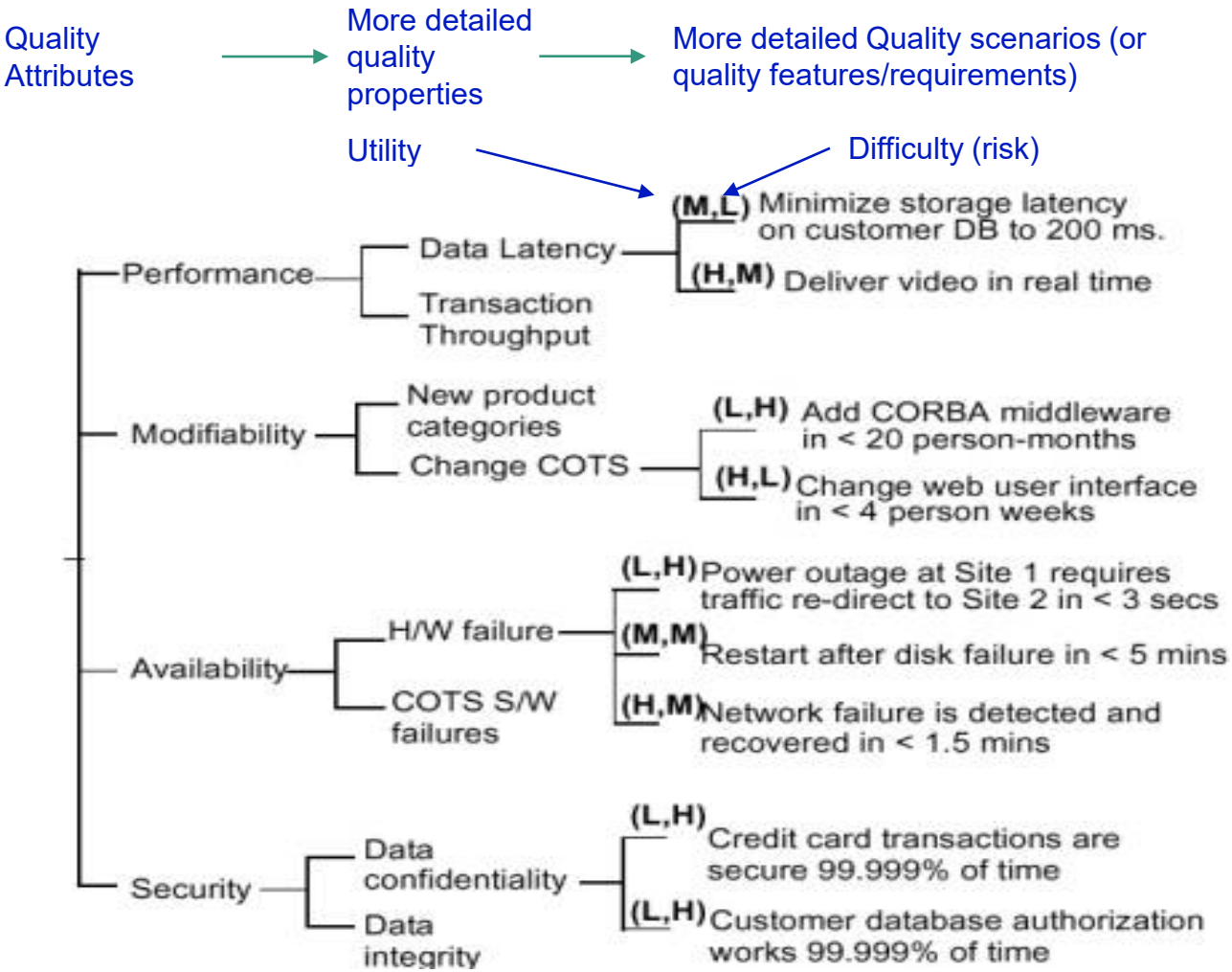
DIFFICULTY (RISK) OF A SCENARIO

- How difficult or risky it is to implement a scenario
  - Effort, cost, schedule
- This is estimated by software engineers
  - Part of requirements engineering and revisited later, as needed
- Similar to difficulty/risk of features or functional requirements
- Scenario implementation *difficulty*
  - High/Medium/Low or
  - Scale 1-3

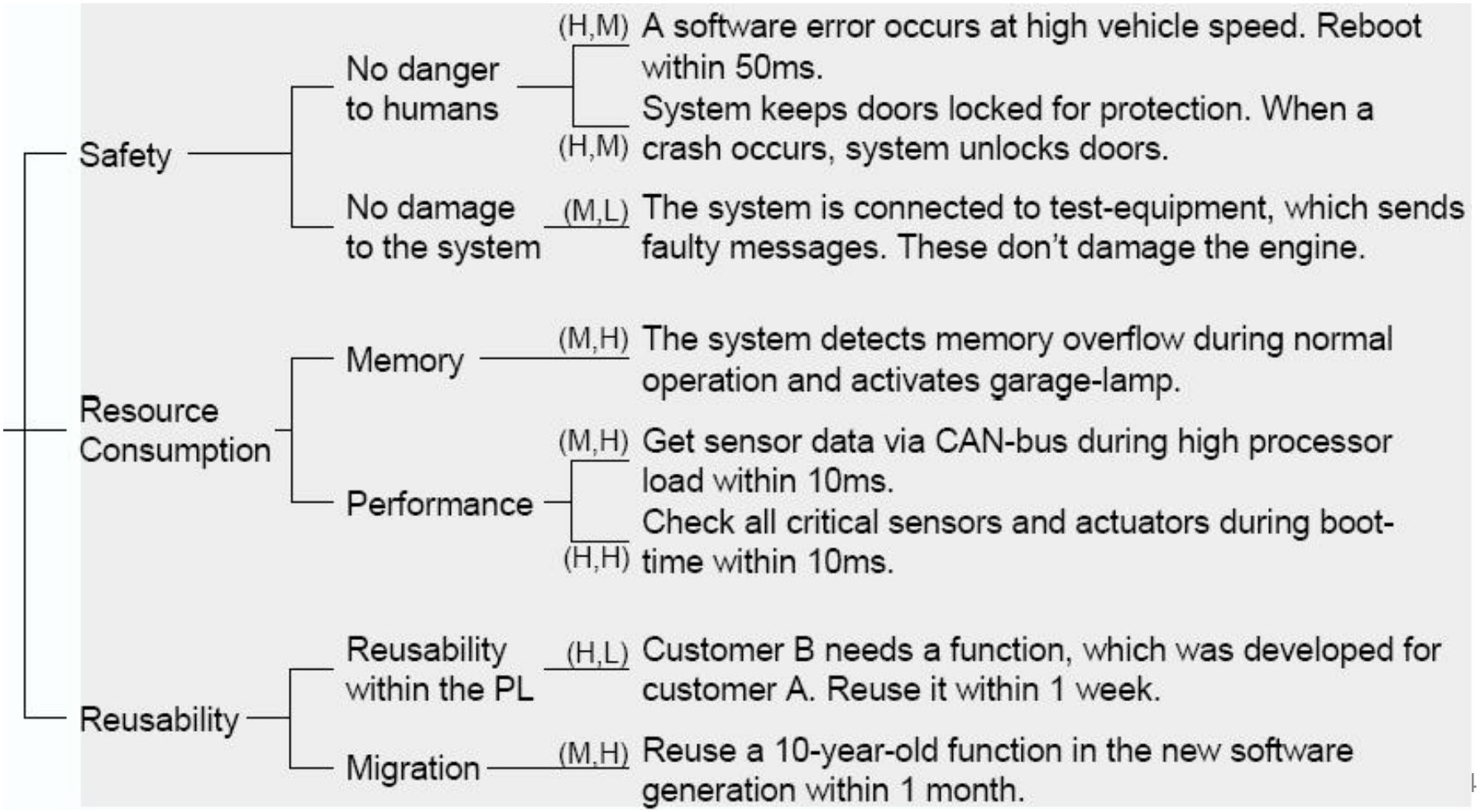
SCENARIO PRIORITIZATION

- Develop a *utility tree*
  - A way to hierarchically organize quality scenarios/quality requirements/features by quality attributes
- Prioritize scenarios
  - By utility and difficulty
  - Some scenarios might be eliminated, if deemed too low utility
    - Negotiate/refine the software scope definition
  - Determine the order of implementation

SOFTWARE QUALITY UTILITY TREE EXAMPLE - ON DEMAND VIDEO SYSTEM



SOFTWARE QUALITY UTILITY TREE EXAMPLE – CAR SOFTWARE



OUTLINE

- Lecture
  - Software quality attributes and non-functional requirements
  - Quality scenarios
  - Utility tree
    - Class exercise – start project LMS utility tree
  - Security in the software life cycle

We are here

Class exercise – start project LMS abuse cases

- Assignments
  - Requirements Analysis Quiz
  - Software development project assignment

CLASS EXERCISE

- For your class LMS development project, identify two quality attributes and one scenario for each
- Rate scenarios for their utility (to the system users) and difficulty or risk to implement
- Start developing the utility tree

CYBERSECURITY

- Cyber security (information technology security) focuses on protecting computers, networks, programs and data from unintended or unauthorized access, change or destruction.

SECURITY CONCERNS

- **C**onfidentiality
  - Data or services are protected from unauthorized access
    - e.g., a hacker shall not access your income tax returns on a government computer
- **I**ntegrity
  - Data or services are protected from unauthorized modification
    - e.g., a grade shall not be changed since the instructor assigned it
- **A**vailability
  - The system will be available for legitimate use
    - e.g., a denial-of-service attack shall not prevent an authorized customer from ordering a book

SECURITY CONCERNS (CONTINUED)

- Nonrepudiation
  - A transaction (access to or modification of data or services) cannot be denied by any of the parties to it.
    - e.g., you cannot deny that you ordered that item over the Internet if, in fact, you did
- Authentication
  - The parties to a transaction are who they claim to be.
    - e.g., when a customer sends a credit card number to an Internet merchant, the merchant is who the customer thinks they are
- Auditing
  - The system tracks activities within it at levels sufficient to reconstruct them.
    - e.g., if you transfer money out of one account to another account, the system will maintain a record of that transfer
- Privacy
  - Protect the privacy of user's data - usually associated with (but not limited to) personally identifiable information (PII) - See Terms and Conditions May Apply

SOME DEFINITIONS/TERMINOLOGY

Term	Definition
Asset	Something of value which has to be protected. The asset may be the software system itself or data used by that system.
Attack	An exploitation of a system's vulnerability. Generally, this is from outside the system and is a deliberate attempt to cause some damage.
Control	A protective measure that reduces a system's vulnerability. Encryption is an example of a control that reduces a vulnerability of a weak access control system
Exposure	Possible loss or harm to a computing system. This can be loss or damage to data, or can be a loss of time and effort if recovery is necessary after a security breach.
Threat	Circumstances that have potential to cause loss or harm. You can think of these as a system vulnerability that is subjected to an attack.
Vulnerability	A weakness in a computer-based system that may be exploited to cause loss or harm.

THREAT CATEGORIES – STRIDE MODEL

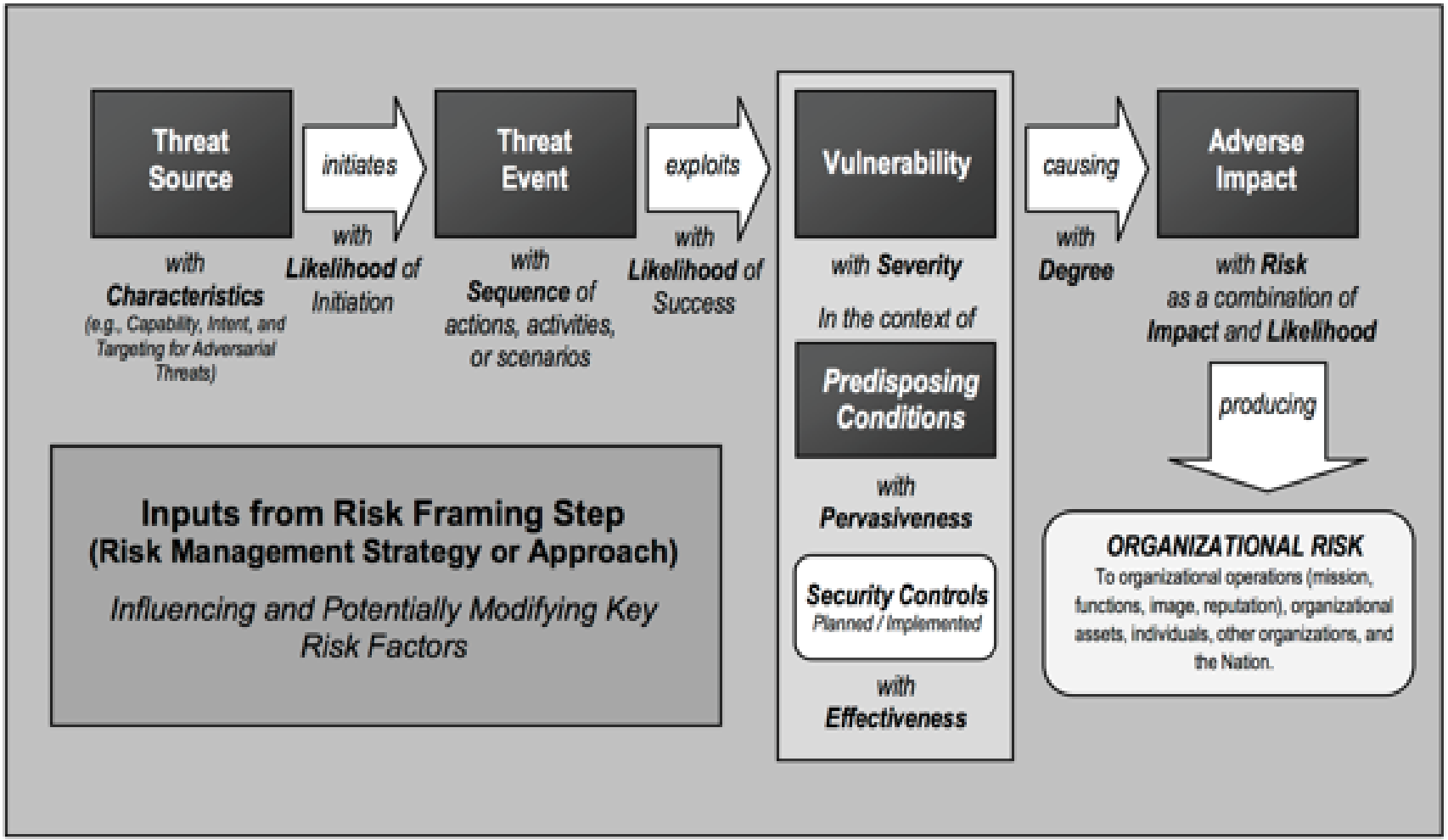
- **Spoofing identity.** An example of identity spoofing is illegally accessing and then using another user's authentication information, such as username and password.
- **Tampering with data.** Data tampering involves the malicious modification of data. Examples include unauthorized changes made to persistent data, such as that held in a database, and the alteration of data as it flows between two computers over an open network, such as the Internet.
- **Repudiation.** Repudiation threats are associated with users who deny performing an action without other parties having any way to prove otherwise—for example, a user performs an illegal operation in a system that lacks the ability to trace the prohibited operations.
  - **Nonrepudiation** refers to the ability of a system to counter repudiation threats. For example, a user who purchases an item might have to sign for the item upon receipt. The vendor can then use the signed receipt as evidence that the user did receive the package.
- **Information disclosure.** Information disclosure threats involve the exposure of information to individuals who are not supposed to have access to it—for example, the ability of users to read a file that they were not granted access to, or the ability of an intruder to read data in transit between two computers.
- **Denial of service.** Denial of service (DoS) attacks deny service to valid users—for example, by making a Web server temporarily unavailable or unusable. You must protect against certain types of DoS threats simply to improve system availability and reliability.
- **Elevation of privilege.** In this type of threat, an unprivileged user gains privileged access and thereby has sufficient access to compromise or destroy the entire system. Elevation of privilege threats include those situations in which an attacker has effectively penetrated all system defenses and become part of the trusted system itself, a dangerous situation indeed

EXAMPLE THREATS, EFFECTS AND COUNTERMEASURES

Threat	Property Affected	Countermeasure
Spoofing	Authentication	• Passwords, multi-factor authN • Digital signatures
Tampering	Integrity	• Permissions/ACLs • Digital signatures
Repudiation	Non-Repudiation	• Secure logging and auditing • Digital Signatures
Information Disclosure	Confidentiality	• Encryption • Permissions/ACLS
Denial of Service	Availability	• Permissions/ACLs • Filtering • Quotas
Elevation of privilege	Authorization	• Permissions/ACLs • Input validation

The STRIDE Threat Model (2005). Retrieved from Microsoft Developer Network: https://msdn.microsoft.com/library/ms954176.aspx

SECURITY *RISK*



From: Blank, R. M., & Gallagher, P. D. (2012, September). *NIST Guide for Conducting Risk Assessments.* Retrieved from http://csrc.nist.gov/publications/nistpubs/800-30-rev1/sp800_30_r1.pdf

CYBERSECURITY ASPECTS

- Physical (infrastructure) security
- IT system
  - Infrastructure
    - Network
    - Machine
  - Software: Operating system and application software
- Personnel
  - Policies and procedures (e.g., Acceptable Use Policy AUP)
  - Insider threat – see https://www.fbi.gov/file-repository/insider_threat_brochure.pdf/view

- Infrastructure security is a *systems management problem* where the infrastructure is configured to resist attacks.
- **Application security** is a ***software engineering problem*** where **the system is designed to resist attacks**.

SOFTWARE-RELATED SECURITY RISKS

- Software can be used as a means to attack – e.g., malware
- Software can be an unintended enabler for attacks if it contains vulnerabilities
  - Known vulnerabilities -> fix by patching
  - Unknown vulnerabilities (zero-day vulnerabilities)

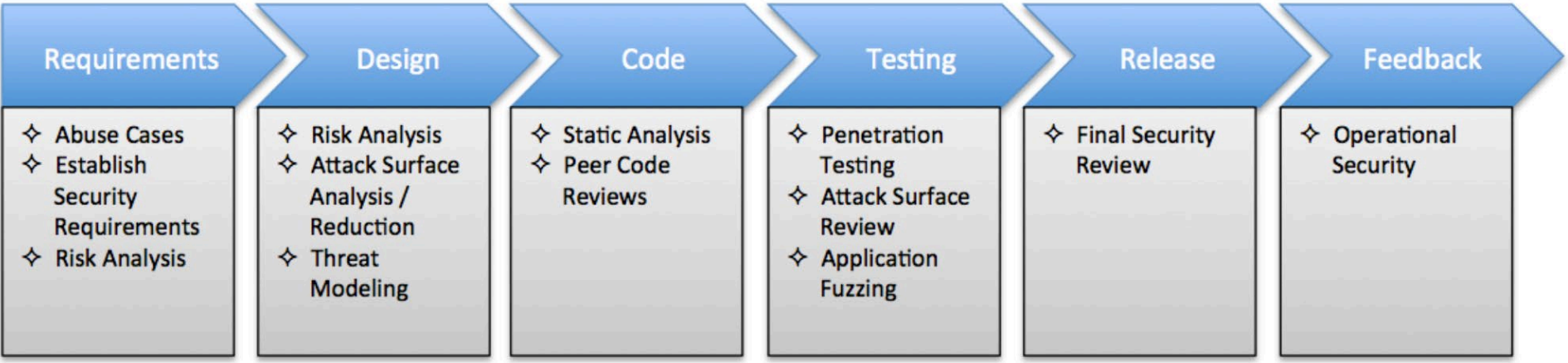
- Bug bounty programs
- Underground malware and SW vulnerability economy
  - Greenberg, A. (2012, March). *Shopping For Zero-Days: A Price List For Hackers' Secret Software Exploits.*

WHEN SHOULD SECURITY BE ADDRESSED IN SW DEVELOPMENT?

- Explain to your pair when and how do you think security should be addressed in the software lifecycle
- Provide arguments for your opinion

BUILDING-IN SOFTWARE SECURITY

- Security must be addressed in all SW engineering phases



Integrating Security into the Software Development Life Cycle
© Capstone Security, Inc.

From: http://capstonesecurity.com/services/application-development/

PENETRATION TESTING ☺

https://www.youtube.com/watch?v=VErpgNK77AE

RESOURCES

- IEEE Cybersecurity
- US CERT – Build Security in
- Microsoft Security Development Lifecycle
- Book: *The Security Development Lifecycle*, by Michael Howard and Steve Lipner

RISK ANALYSIS EXAMPLE

Threat: Conduct brute force login attempts/password guessing attacks.

Threat source: external actor or insider

ASSET	
Type	**Value**
Personal data	High (8)

THREAT						
Source	**Threat type**	**Effect**	**Capabilit**	**Intent**	**Targeting**	**Likelihoo**
External or internal, human, adversarial	Conduct brute force login attempts/password	Access and potential deletion of to confidenti[illegible]es	High (8)	High (8)	High (8)	High (8)

VULNERABILITY	
Name	**Severity**
Weak password	Very high (10)

IMPACT	
Type	**Level**
Harm to assets (confidentiality, integrity)	High (8)
Harm to individuals (privacy)	High (8)

RISK			
Threat	**Vulnerability**	**Likelihood**	**Impact**
Conduct brute force login attempts/password guessing attacks	Weak password	High	High

RISK ANALYSIS EXAMPLE (CONTINUED)

Threat Event (Attack)	Threat source (Attacker)	Threat characteristics			Relevance	Likelihood of attack	Vulnerabilities	Severity and pervasiveness	Likelihood of attack success	Overall likelihood	Level of impact	Risk	Countermeasures
		Capability	Intent	Targeting									
Conduct brute force login attempts/password guessing attacks	External or internal, human, adversarial	High	High	High	High	High	Weak password	Very high	High	High	High	High	-Immediate password change -Change, communicate, and enforce password strenght policy

SECURITY IN REQUIREMENTS - ABUSE CASES

- *Abuse case* - a specification of a type of complete interaction between a system and one or more actors, where the results of the interaction are harmful to the system, one of the actors, or one of the stakeholders in the system

- We can use a UML use case diagram to represent abuse cases

From: *Using Abuse Case Models for Security Requirements Analysis*, by John McDermott and Chris Fox
https://www.acsac.org/1999/papers/wed-b-1030-john.pdf

COMPARISON USE CASES – ABUSE CASES

Use Case

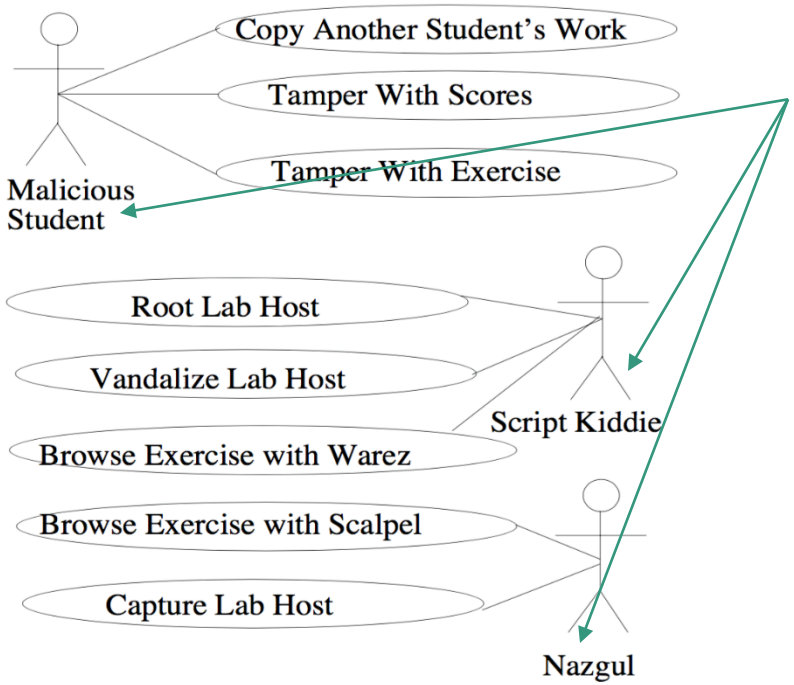
- A complete transaction between one or more actors and a system.
- UML-based use case diagrams.
- Typically described using natural language. Could also use a graphical representation, e.g., UML activity diagram

Abuse Case

- A complete transaction between one or more actors and a system, **that results in harm.**
- UML-based use case diagrams.
- Typically described using natural language. Could also use a graphical representation, e.g., UML activity diagram (A tree/DAG diagram may also be used)
- Potentially, one family member for each kind of privilege abuse and for each component that might be exploited.
- Includes a description of the range of security privileges that may be abused.
- **Includes a description of the harm that results from an abuse case.**

ABUSE CASE DIAGRAM EXAMPLE

Malicious goals of the actors



Actors that cause harm

See definitions on the next slide

Figure 3. Abuse Case Diagram for an Internet-Based Information Security Laboratory

From: *Using Abuse Case Models for Security Requirements Analysis*, by John McDermott and Chris Fox
https://www.acsac.org/1999/papers/wed-b-1030-john.pdf

CLARIFICATION OF TERMS FOR THE EXAMPLE DIAGRAM

“Script Kiddie”

- *Definition* – A person who uses existing computer scripts or code to hack into a computer, lacking the expertise to write their own.
- *Resources* - The Script Kiddie operates alone, although he or she may exchange some information with fellow Script Kiddies. The Script Kiddie has hardware, software, and Internet access that might be available to an individual through purchase with personal funds or by theft from an employer. Our model assumes that the Script Kiddie is willing to spend about 24 hours trying to defeat the security of a particular system.
- *Skills* - Script Kiddies have limited technical skills. The majority of their activities are carried out using tools and techniques devised by other people.
- *Objectives* - Script Kiddies may have a variety of criminal objectives including vandalism and theft. They also are interested in demonstrating their technical prowess.

CLARIFICATION OF TERMS FOR THE EXAMPLE DIAGRAM

“Nazgul”

- *Definition* – a person set out for destruction or total annihilation.
- *Resources* - Nazguls operate on behalf of groups that have budgets set aside to accomplish some form of harm. They may have technical assistance from an organization that is tasked with supporting them. They have hardware, software, tools, and Internet access provided by a business, a government, or a quasi-government. They have significant access to documentation of the systems they intend to abuse and may be able to test or simulate an intended exploit on a copy of the target system. We assume that Nazguls may spend up to 90 days in preparation and execution of an attempt to breach the security of the system.
- *Skills* - Nazguls have superior technical skills. They can design operating systems, network protocols, computer hardware, and cryptographic algorithms. They apply software engineering technology, mathematics, computer engineering, and similar disciplines to their exploits.
- *Objectives* - Nazguls are primarily interested in accomplishing the objective of the organization that supports them. They also seek to increase their own skills and knowledge, but not to demonstrate them to anyone. Organizations that support Nazguls do so to carry out espionage, warfare, terrorism or similar harmful activities.

CLARIFICATION OF TERMS FOR THE EXAMPLE DIAGRAM

”Warez" = packages and tools that allow a user to mount attacks on a system from a GUI, without knowledge of the principles involved in carrying out the attack.

"Scalpel" = a well-engineered attack designed specifically to penetrate the system.

ABUSE CASE TEXTUAL DESCRIPTION

Use the same *template* provided for textual description of a use case

Name: the name of the abuse case

Actors: The primary and secondary attacker(s) that participate in the abuse case

Trigger: what external event happens that triggers the start of the abuse case

Preconditions: any prerequisites before the abuse case can be started

Postconditions:

- **Success postconditions**: the system protects its assets, and the attack fails
- **Failure postconditions:** the system fails to protect against the attack and harm happens to the system assets (specify what *assets* and what *security properties* are affected)

ABUSE CASE TEXTUAL DESCRIPTION (CONTINUED)

Basic flow

- *Attacker* performs typical *attack*
- System responds such that that, in the end, the attack is **not** successful (does not impact system's security properties), e.g.: system *authenticates*, *authorizes* (or not), *saves/recovers data*, *logs incident*…

Alternative flow

- Attacker performs attack and tries something unusual, or different choices
- System *responds* such that the attack is **not** successful (does not impact system's *assets* and *security properties*)

Exception flow – attacker performs attack, system's response fails to prevent or mitigate the attack; in the end, the attack is successful, and harm is done

ABUSE CASE TEXTUAL DESCRIPTION - EXAMPLE

Name: Browse Server Exercise With Warez
Actors: Script Kiddie
Trigger: Script Kiddie has access to the Lab and intends to access exercises on a lab server
Preconditions: Script Kiddie has a one-time control of a single student session on a server
Postconditions:

Success postconditions: Script Kiddie fails to gain access to the exercise documentation on the server
Failure postconditions: Script Kiddie gets access and browses the exercise documentation on the server

ABUSE CASE TEXTUAL DESCRIPTION – EXAMPLE (CONTINUED)

Basic flow

1. Script Kiddie requests to initiate a session on a lab host, using the TCP/IP protocol suite and an attack tool called Warez 1
2. The lab host establishes the session with the Warez 1 tool
3. Script Kiddie requests the lab host to browse files containing exercise documentation stored on a lab server
4. The lab host rejects the request to browse files containing exercise documentation stored on a lab server
5. Script Kiddie gives up and ends session

ABUSE CASE TEXTUAL DESCRIPTION – EXAMPLE
(CONTINUED)

Alternative flow

4.a Script Kiddie uses an additional tool Warez2 to request an increase in privilege

4.b The lab host does not grant an increase in privilege.

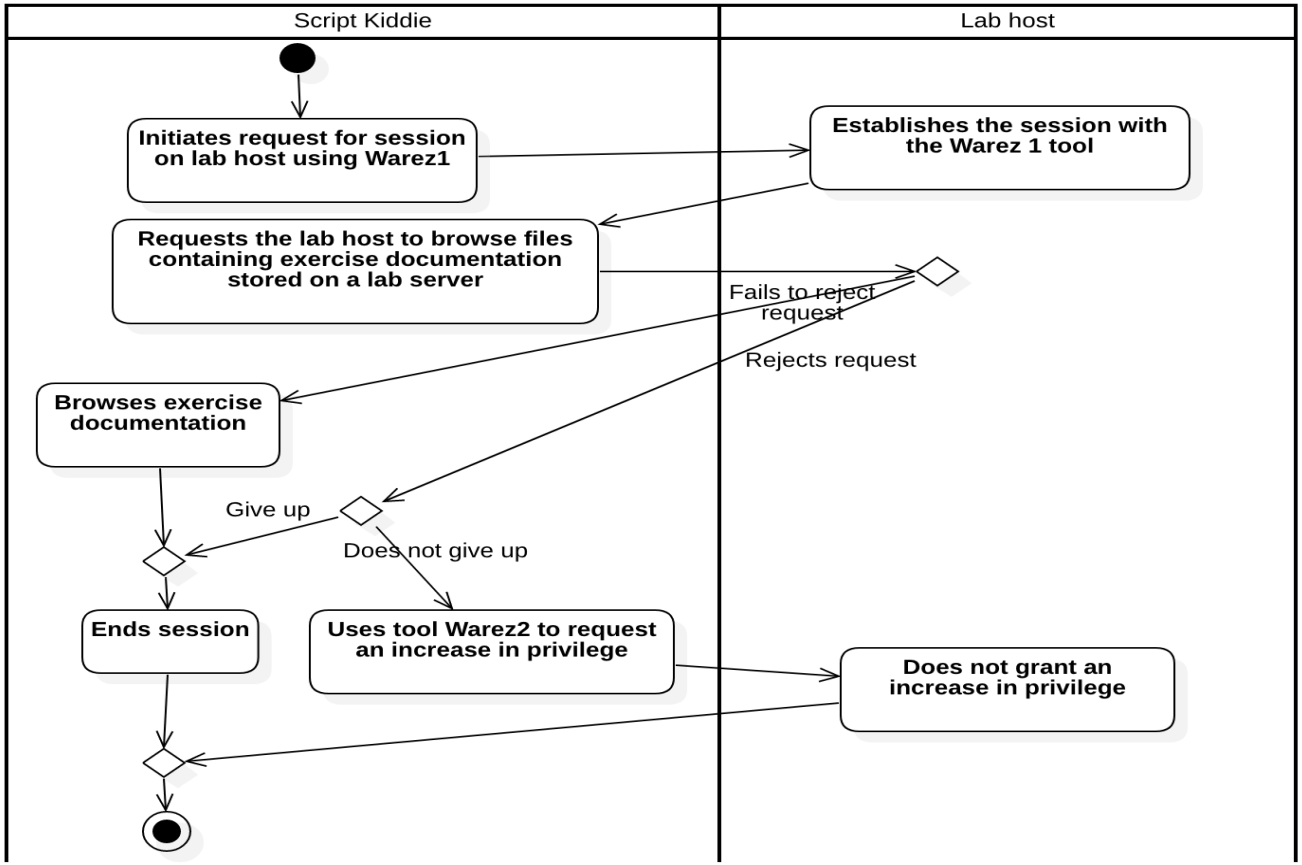
Exception flow

4.a The lab host fails to reject the request to browse exercise documentation stored on lab server

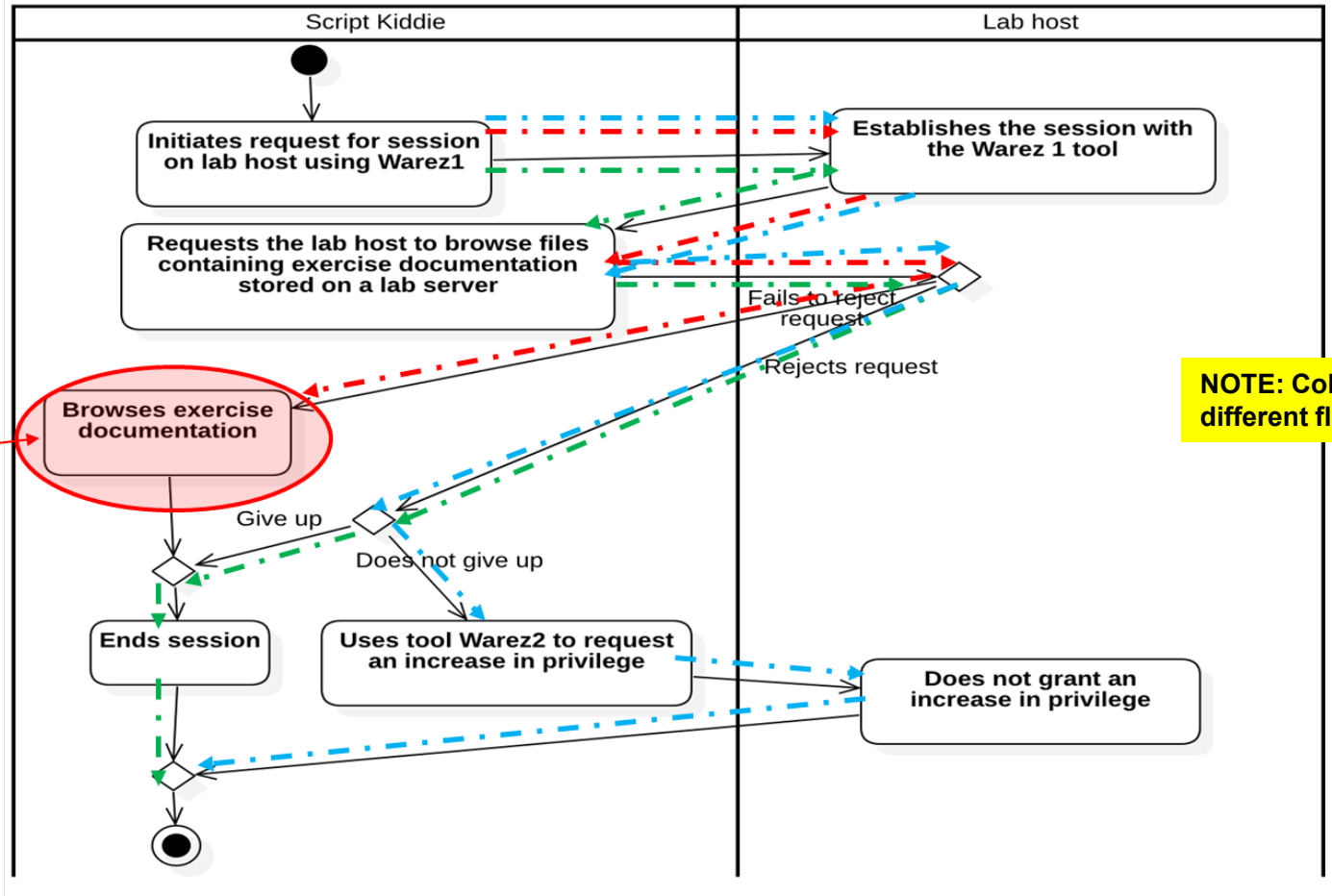
4.b Script Kiddie gets access to files and browses exercise documentation stored on lab server

ABUSE CASE GRAPHICAL DESCRIPTION - EXAMPLE USING UML ACTIVITY DIAGRAM

Abuse case: Browse Server Exercise With Warez



ABUSE CASE DESCRIPTION EXAMPLE – GRAPHICAL REPRESENTATION, USING UML ACTIVITY DIAGRAM



Successful attack!

NOTE: Colored lines mark different flows (scenarios)

ABUSE CASES AND SECURITY SCENARIOS

- On the previous slide, differently colored dashed lines mark different security scenarios
  - Green marks the basic flow (scenario)
  - Blue marks an alternate flow (scenario)
  - Red marks an exception flow (scenario)
- Together, these scenarios form the abuse case “Browse Server Exercise With Warez”
  - Similar to how a use case consists of a set of related, cohesive use case (functional) scenarios

HOW DO WE INTEGRATE ALL THIS ANALYSIS/TECHNIQUES?

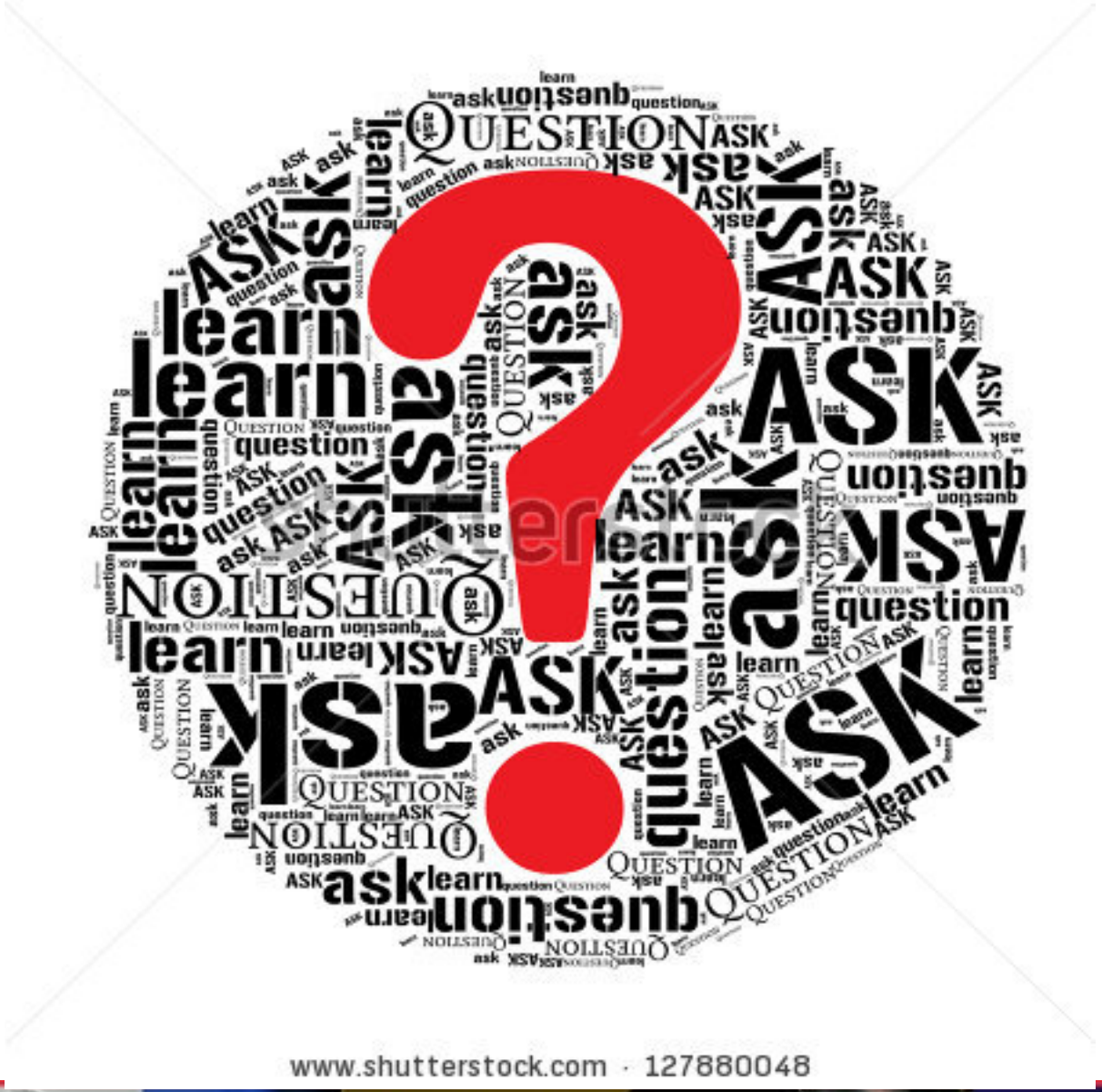
- **Steps**:
  - Identify ***system assets*** and their security properties ("CIA")
    - Assign *criticality* to each asset property, based on its value to the stakeholders
  - Identify potential ***threat sources***
    - Prioritize them by how dangerous they can be, based on their *capability* and *intent/motivation*
  - Identify potential threat events (***attacks***)
    - Assign *risk value* to each threat, based on *likelihood of occurrence* and *harm/ impact*
  - Identify potential ***software vulnerabilities***
    - Assign *severity* to vulnerabilities
    - Associate vulnerabilities to attacks that would try to exploit them
- **Outcome**: Actors (attackers) and use cases (attack/security scenarios)
  - **Security scenarios are input to the *Utility tree***
    - The *Utility* value for a security scenario reflects its criticality, as a function of its *likelihood* of success and *criticality* of the impact (affected asset value and its damage)

CLASS EXERCISE

- Start risk analysis and develop an abuse case model for your project LMS
  - Represent the abuse case as a UML use case diagram

SUMMARY

- Software quality attributes and non-functional requirements
  - Their importance relative to software design
- Quality scenarios
- Utility tree - prioritization
- Security in the software life cycle
  - Abuse cases


www.shutterstock.com · 127880048

OUTLINE

- Lecture
  - Software quality attributes and non-functional requirements
  - Quality scenarios
  - Utility tree
    - Class exercise – start project LMS utility tree
  - Security in the software life cycle
    - Class exercise – start project LMS abuse cases
- Assignments
  - Requirements Analysis Quiz
  - Software development project assignment

We are here

REQUIREMENTS ANALYSIS QUIZ

- Individual, online

OUTLINE

- Lecture
  - Software quality attributes and non-functional requirements
  - Quality scenarios
  - Utility tree
    - Class exercise – start project LMS utility tree
  - Security in the software life cycle
    - Class exercise – start project LMS abuse cases
- Assignments
  - Requirements Analysis Quiz
  - Software development project assignment

We are here

THINGS TO AVOID FOR HAVING A SUCCESSFUL TEAM

1 - Ego
2 - Negative competition
3 - Poor communication
4 - Micromanagement
5 - Criticism without praise
6 - Unreasonable expectations
7 - Half-hearted work
8 - Stubbornness
9 - Leading with emotions

https://www.linkedin.com/pulse/do-you-have-successful-team-here-9-things-should-avoid-bernard-marr/

SOFTWARE DEVELOPMENT PROJECT ASSIGNMENT

- *Requirements analysis*: use cases, abuse cases, features (with prioritization), quality scenarios (with prioritization), utility tree, bi-directional traceability matrices
- *Individual Student Contribution* for this activity
  - Provide a **detailed** description of each team member's contribution
- Update the *Project Management* workbook: actual effort, revisit risks, lessons learned during this activity
- Deliverables and due dates are in ELMS
- Online submission, **team assignment -> only one member of each team needs to submit**

SOFTWARE DEVELOPMENT PROJECT REVIEWS

- **Team to team (T2T) review** of the *Requirements Analysis* submission, for the respectively assigned team

Example

Author team	Reviewer team
Group 1	Group 2
Group 2	Group 3
Group 3	Group 4
Group 4	Group 5
Group 5	Group 1

SOFTWARE DEVELOPMENT PROJECT REVIEWS (CONTINUED)

- Reviews will be assigned in ELMS to each student
- Each student must review the assigned artifact
- The phase leader collects review comments from all team members and provides them on behalf of the team
- Review comments will be provided in ELMS, using the ELMS review mechanism
- If providing review comments in ELMS does not work, the *Review team's* comments will be emailed to **all** the members of the *Author team*
  - **CC the instructor and TAs/graders** (if any)

- **Review counts towards the grade**

REVIEW GUIDELINES

- The Review team:
  - Use the assignment description and rubric as review criteria
  - Provide specific, detailed, and concise feedback, point the issues and highlight the positives
  - Formulate your comments as statements, not as questions
  - You may contact the author team if you have any questions
- The Author team:
  - Read the review comments and address them to the reviewer's satisfaction
- Recommended reading:
  - Guidelines for Students - Peer Review
  - *The Soft Side of Peer Reviews*, by Karl Wiegers