Safety/Security and Extensibility/Scalability in Software System Design and Architecture

Chen Junda

161250010

Software System Design and Architecture

December 22, 2018

Definitions and Comparisons

In this part, the definitions of, relationships and differences between selected pairs of quality attributes (security/safety and extensibility/scalability) are analyzed and presented with examples.

Safety/Security

A "safe" system is such that the harm from accidental mishaps to the system itself can be minimized or avoided. A "secure" system is such that some important properties of a system (like integrity, access, accountability, availability and confidentiality) can still be maintained under intentional attacks.

In another word, "safety" means the ability to *reduce* the *risk of* and *harm* from **unintentional** mishaps to the system's stakeholders and valuable assets, whereas the "security" indicates lowering of the *risk* of and *harm* from **intentional** attacks.

It can be observed that both attributes require a system to be able to **preserve** important **properties** of a system and **minimize** the **harm**, but from difference types of accidents.

"Safety" focuses on **unintentional** incidents, i.e. the incidents that not meant to cause damage to the system itself. For example, a "safe" social media system should still be functional if one of its servers is disconnected from Internet because of a maloperation during a construction (like cutting a critical wire by mistake); an artificial satellite should keep intact for its major components operational when facing a strong cosmic ray, but some degree of slowdown is tolerable; in a safety system, important data (like financial transactions) won't be unrecoverably lost if a disk containing these data malfunctions unexpectedly.

"Security", on the other hand, talks about **intentional** attacks, i.e. the attacks that targets to the system from the very beginning. For example, in a secure system, no plaintext passwords shall be compromised when hackers are attacking database; a medical system should hold long enough under terroristic cyber attacks to be able to get professional assists from law enforcement department, since a breakdown of such system might cause disasters; if a hacker had gained access to the system illegally, the system should be able to detect their existence, remove their privilege, and then report the bug that was abused as soon as possible.

Extensibility/Scalability

System always grows as time goes, but by two directions: **vertically** or **horizontally**. A system might need to implement more functions than originally anticipated or change the implementation that has already been made (**vertically**); a system might also be required to process more requests without excessive changes to the system itself in the future (**horizontally**).

Both extensibility and scalability focus on the growth of a system, but from difference perspective. Extensibility is the ability to **extend vertically**: that is to add “extensions” (like new features, modification to existing modules etc.) without too much changes and impacts to be expected. Scalability, on the other hand, is the ability and potential to **scale horizontally**: i.e. the ability to effectively handle growing or reducing amount of work using existing system or with minimal changes to the system itself as the number of works increases or decreases.

For example, an extensible frontend project usually indicates that adding a new page (to meet newly-derived business requirements) can be easily implemented without a deep dive into existing code. It also might mean that changing a style to an existing common component can be done within one place which takes effect for all of its occurrences. An extensible system with complex dataflow should be able to integrate a data processing module without an overhaul to the whole dataflow.

As for scalability, existing cloud service providers (Microsoft Azure, AWS etc.) all provide a scalable infrastructure that can gradually accommodate more demands as more companies are migrating their services to cloud based platform for better performance, maintainability and cost. A new concept of computing, serverless or functional computing, are gaining ground in cloud service territory because of its “infinite scalability”, which means a service can adapt to handle any amounts of requests the service is actually facing, freeing developers from caring infrastructure and codebase themselves as the number of requests increases.

General and Concrete Scenarios

In this part, scenario-based analysis method is applied on the aforementioned pairs of quality attributes to create their general and two concrete scenarios respectively.

Safety

Portion of Scenario	Possible Values
Source	Internal or external to the system
Stimulus	Events that is unintentional to damage but will affect the system
Artifact	System or one or more modules of the system
Environment	Portion of system might be influenced by this event
Response	Detect the event ● Detect the event’s occurrence ● Analyze the affected modules ● Notify related entities

	Avoid or minimize the damage ● Disable or isolate affected modules ● Deploy backup assets to restore functionality ● Switch into a degraded mode ● Be offline during repair ● Restore to normal after the damage is fixed Avoid future occurrence ● Find the vulnerabilities ● Report the vulnerabilities ● Fix the vulnerabilities
Response Measure	Time to detect the events Time to notify the related entities Time to mask affected modules Time to restore functionality Time to repair critical modules Estimated damage for accidents Time to find the vulnerabilities Time to fix the vulnerabilities

Samples:

Portion	Description
Source	A construction team unrelated to the system
Stimulus	Cut a network wire that connects system to the Internet by mistake
Artifact	Network module
Environment	The affected network module is using the wire when the event occurs
Response	The module detects the event and switches to another router bypassing the broken wire
Response Measure	The **detection** and **reaction** take only 30s for the system to go back to normal, during which the **throughput** dropped 5%.

Portion	Description
Source	Nature
Stimulus	Unexpected earthquake
Artifact	A disaster control system for a nuclear power plant
Environment	The earthquake damages containers and causes leaking of radioactive materials.
Response	Detect the leaking, initiate early emergency process (like shutting related reactors), notify authorities
Response Measure	The **detection**, **initiation** and **notification** take only 3 min. Leaked materials are within control and **won't cause severe biohazard.**

Security

Portion of Scenario	Possible Values
Source	Internal or external to the system
Stimulus	Intentional attacks to the system
Artifact	System or one or more components
Environment	The system doesn't foresee the attack
Response	Detect the attack ● Detect the attack's occurrence ● Analyze the damage ● Report the attack Maintain the properties ● Disable compromised modules ● Deploy backup assets ● Lock critical modules and data ● Remove attackers from the system ● Switch to degraded mode ● Restore to normal after the attack is resolved Avoid future occurrence ● Find the vulnerabilities ● Report the vulnerabilities ● Fix the vulnerabilities
Response Measure	Time to detect the attack Time to switch to degraded mode Time to secure critical modules and data Time to remove or block the attackers Time to deploy backup assets Estimated damage of the attack Time to find the vulnerabilities Time to fix the vulnerabilities

Samples

Portion	Description
Source	A malicious hacker team
Stimulus	A well-coordinated and massive DDoS attack
Artifact	Some part of the system that can barely hold the attack
Environment	The hacker initiated a massive DDoS attack to the system
Response	Detect the attack; Detect and block attack source; Disable compromised module; Deploy backup server resources; Notify security team.
Response Measure	**Time** to **detect** and **notify** the attack is 30s. **80% attack sources have been blocked after 30 minutes.** The system **goes back** to normal in 1 hours.

Portion	Description
Source	A lone-wolf hacker
Stimulus	An unauthorized entry to critical database
Artifact	A database that contains critical and confidential data
Environment	The database is operational
Response	Detect the entry; Block the entry; Report the event to authority; Report the vulnerabilities the intruder uses.
Response Measure	The **detection**, **blocking** and **reporting** take 15 seconds and **no data is leaked**. The **vulnerabilities** are fixed in 1 day.

Extensibility

Portion of Scenario	Possible Values
Source	End user, developers, requestor
Stimulus	A directive to add/delete/modify functionality on existing system
Artifact	Code, data, interfaces, components, resources, configurations…
Environment	Business analysis time, runtime, compile time, build time, initiation time, design time, test time
Response	● Understand extension ● Design extension ● Make extension ● Test extension ● Deploy extension
Response Measure	● Time and material cost of communicating, understanding, designing, making, testing and deploying of the system extension ● Time and material cost of reeducating users or other stakeholders after system extensions ● Other affected modules not originally anticipated during actual processing ● Potential time and material cost of newly-introduced defects

Samples

Portion	Description
Source	Requestor
Stimulus	Wish to add a new functionality onto an existing website
Artifact	Code
Environment	design time, business analysis time
Response	Understand, design, make, test and deploy the new requirement
Response Measure	All changes deployed in **3 days** but brought **70 more bugs** which took **3 days** more to resolve. **New tutorials** are written to teach

	the end users about the new functionality

Portion	Description
Source	Developer
Stimulus	Wish to change a provider for a service that depends on third-party services
Artifact	Interfaces
Environment	design time, runtime, test time
Response	Design, make, test and deploy the new requirement
Response Measure	All changes made in **1 days**. Only **1 module** are affected, and **no more defects** are introduced.

Scalability

Portion of Scenario	Possible Values
Source	End user, developers, requestor
Stimulus	The need to use existing system to handle different number of requests from originally designed with minimal change
Artifact	System or one or more components in the system
Environment	System's operation mode
Response	● Evaluate possibilities and potential change ● Make the change, if necessary ● Process requests
Response Measure	Latencies on different level of load Max and min number of requests The improvement brought by the scale The cost to expand or shrink scale The cost to change existing system to adapt for the change The cost to resolve defects and interference when executing a scaling

Samples

Portion	Description
Source	Requestor
Stimulus	Wish to handle 3 times more requests than originally planned during a unit time
Artifact	The whole system
Environment	System hasn't been online yet.
Response	The system is evaluated as capable to accommodate the increase of requests, so no changes should be made.
Response Measure	The max number of requests is **5 times more than plan** and the

	latency increases **only 10% after the 3 times increase**. **No more cost** needed.

Portion	Description
Source	Developer, end user
Stimulus	The need to improve calculation precision for a complex algorithm within original time
Artifact	The calculating module
Environment	System has been operational.
Response	800 more CPUs are added into the mainframe as the evaluation indicates, no more changes required.
Response Measure	The process **doesn't interfere normal operation.** No more **cost** or change except CPUs' are needed. The precision improvement **increases the sale of the system by 30%**.

Strategies and Tactics

In this part, strategies and tactics to improve each QAs are presented as well as their benefits and penalties to other attributes and QAs.

Safety

Strategy	Tactic	Description	Benefits	Penalties
Avoid	Redundancy	Introduce redundant assets into the system	Increase robustness and security, avoid single-point of failure	Increase cost, complicate system arch design
	Avoid risk design	Avoid making arch designs that has high possibility to cause problem in the future.	Reduce the possibility for problems to occur	Limit the decisions that can be beneficial in other perspectives
Detect	Designated monitoring system	A complete and separate system to constantly monitor the critical perspectives of the system	Get accurate, complete data and error report in time without interfering original system	Increase cost, a new point of failure to be kept watch on

	Heartbeat	System sends a signal every time interval to report its status	Easier to implement and integrate; detect event in time	May affect system performance
Handle	Degrade	Limit the system's functionality to limit the potential damage	Maintain basic functionality while handling the problem	Affect user experiences during degradation
	Disable affected modules	Disable the affected modules completely, fix it and then goes back to normal	Completely avoid further damage and be able to be fixed quickly	Might cause a complete breakdown of a function

Security

Strategy	Tactic	Description	Benefits	Penalties
Avoid	Test	Find and fix as many vulnerabilities as possible before putting the system into use	Avoid further and usually more damage at a security breach in runtime	Increase development time and material cost
	Simplify design	Simplify the architecture design to avoid vulnerabilities that comes with unnecessary components	Avoid vulnerabilities and save resources	Might be negative for other QA like modifiability and extensibility
	Add security strategies	Add more strict security methods (like 2-step auth) to protect the system from being hacked	Increase the cost of hacking to reduce the hacker's benefit and interest	Increase the complexity. Negative impacts on usability and efficiency
Detect	Log	Log all entries to protected area	Easy to integrate and implement	Entries are too many to check
	Report	Report suspicious and abnormal operation	Reduce amount of work to check all the logs	Some operation might be mistakenly ignored
Handle	Isolate or	Isolate or shutdown the	Completely	Might cause a

	shutdown compromised modules	compromised modules to limit the damage	avoid further damage	complete breakdown of a function
	Delete critical data	Delete critical and data, if backed up, to avoid data leaking	Avoid data leaking	Not applicable if no backup is available.

Extensibility

Strategy	Tactic	Description	Benefits	Penalties
Improve inner architecture	Split by function	Split a large system by function so that each function can be run individually	Adding or modifying function won't affect existing ones	Need careful design; might not be the most efficient and performant
	Constant refactoring	Constantly refactor the arch as development goes on, not relying on an unrealistic "perfect" arch	A good balance between cost and quality within a development cycle	High skill requirement for developers and teams
Improve outer interface design	Expose only necessary interfaces	Only exposes necessary APIs	Increase implementation flexibility; improve security	Reduce usage; hard to determine the "necessity" of interfaces
	Do one thing, do it well	An interface should focus on one small piece of work and do it well.	Improve usability, implementation flexibility and interoperability; helps scalability	Need careful design

Scalability

Strategy	Tactic	Description	Benefits	Penalties
Split	Split by responsibility	Split a system by different responsibilities into difference layers (data accessing,	Optimize each layer with their own characteristics; easy to scale	More complicated architecture design; more time and

		calculating, viewing etc.)	each layer accordingly	material cost
	Partition database	Partition database so that pressure to database can be “divided and conquered".	More throughput and scalability from the database	Not always applicable; inappropriate partition may lower performance
Make use of cache	Deliver static contents separately	Split static contents and deliver them with less costly sources	Reduce server pressure and make the most use of precious calculating resources	data synchronization might be a problem
	Use in-memory database as cache	Use in-memory database (like redis) to avoid frequent access to actual database	Reduce access to database, improve performance and responsiveness	A new layer to worry about; more complicated architecture design

References

5.10 Measuring the System Scalability. (n.d.). Retrieved from Lebanese Republic Office of the Minister of State for Administrative Reform: http://www.omsar.gov.lb/ICTSG/105OS/5.10_Measuring_the_System_Scalability.htm

Bloch, J. (2006, October 22-26). How to Design a Good API and Why it Matters. *Proceeding OOPSLA '06*, (pp. 506-507). Portland, Oregon, USA. doi:10.1145/1176617.1176622

Firesmith, D. G. (2010). *Engineering Safety- and Security-Related Requirements for Software-Intensive Systems.* Carnegie Mellon University, Software Engineering Institude, Pittsburgh, PA 15213.

Kellyh, T. (2008). *Safety Tactics for Software Architecture Design.* The University of York, High Integrity Systems Engineering Group, Department of Computer Science .

Seovic, A. (2010). Achieving Performance, Scalability and Availability Objectives. In M. F. Aleksandar Seovic, *Oracle Coherence 3.5.*

Serhiy. (2017, April 14). *How to Increase The Scalability of a Web Application.* Retrieved from Romexsoft: https://www.romexsoft.com/blog/improve-scalability/

Shoup, R. (2008, May 27). *Scalability Best Practices: Lessons from eBay.* Retrieved from InfoQ: https://www.infoq.com/articles/ebay-scalability-best-practices