System Quality Attributes, Non-Functional Requirements, Cross-Functional Constraints

100+ Concepts For Successful Software

By Joel Parker Henderson & ChatGPT

2023-03-24

Contents

System quality attributes (SQAs)

System quality attributes (SQAs) refer to a set of characteristics that define the overall quality of computer software systems. These attributes determine how well the system performs in terms of functionality, reliability, usability, efficiency, security, and maintainability.

- Functionality: The ability of the software to perform its intended tasks without errors or bugs. It includes features such as accuracy, completeness, and compliance with user requirements.
- Reliability: The software's ability to operate consistently and predictably under various conditions. It includes attributes like fault tolerance, availability, and recoverability.
- Usability: The ease of use and learnability of the software. It includes attributes like user interface design, user experience, and accessibility.
- Efficiency: The software's ability to perform its tasks in a timely and resource-efficient manner. It includes attributes like speed, processing time, and memory usage.
- Security: The software's ability to protect user data and prevent unauthorized access. It includes attributes like confidentiality, integrity, and availability.
- Maintainability: The software's ability to be modified or updated easily to meet changing requirements. It includes attributes like scalability, modularity, and extensibility.

Overall, the system quality attributes play a critical role in ensuring that computer software systems are effective, reliable, and secure, and meet the needs of their users.

Cross-functional constraints (CFCs)

Cross-functional constraints (CFCs) refer to limitations or restrictions imposed on software development projects that arise due to the interaction between different functional areas of a business or organization. These areas may include technical areas such as design, development, testing, and operations, as well as non-technical areas such as marketing, finance, and project management.

For example, a cross-functional constraint could be a delay in the project caused by a bottleneck in the testing phase because the design team did not provide the necessary documentation to the testing team to carry out their testing activities. Another constraint could be a budgetary constraint that limits the amount of resources available to the development team, which in turn affects the quality of the final product.

While cross-functional constraints, system quality attributes, and non-functional requirements are all related, they focus on different aspects of a system's design and implementation:

- Cross-functional constraints highlight the need for coordination and collaboration between different functions within an organization to ensure that decisions are made in a holistic and integrated manner.
- System quality attributes and non-functional requirements, on the other hand, help to ensure that the system performs efficiently and effectively based on the organization's needs and requirements.

In general, cross-functional constraints can arise from a variety of factors, including communication gaps, conflicting priorities, competing goals, and organizational silos. To manage these constraints effectively, teams must have a clear understanding of the interdependencies among areas, and must collaborate to identify and address potential issues. Effective communication, stakeholder management, and strategic planning are critical to managing cross-functional constraints and delivering successful projects.

Non-functional requirements (NFRs)

Non-functional requirements (NFRs) refer to the characteristics or qualities of software that do not relate to its specific functionality. These requirements represent the quality attributes of a system and define its overall aspects such as performance, scalability, reliability, usability, accessibility, maintainability, and more.

Examples:

- Performance: The system should carry out tasks within an expected response time, resource utilization, and throughput rate.
- Scalability: The system should handle increasing amounts of users, data, and tasks, while maintaining its performance.
- Reliability: The system should be working well enough, even when showing indications of deteriorating performance.
- Usability: This system should provide user interfaces and user experiences that help end-users succeed.
- Accessibility: The system should work well for people with disabilities, and also satisfy legal and ethical obligations.
- Maintainability: The system should be easily accessible for maintenance, and also maintenance costs should be low.

Non-functional requirements are critical to ensure the success of the system. By including non-functional requirements in the software development process, businesses can ensure that their software meets the expectations of stakeholders and end-users.

Cross-cutting concerns (CCCs)

Cross-cutting concerns refer to aspects of software design and implementation that happen across multiple modules or components of the software system. These aspects typically relate to functionality that is not directly related to the primary purpose or business logic of the software system.

Examples of cross-cutting concerns in software design include:

- Security: authentication, authorization, and access control
- Logging and auditing: tracking user actions and system events
- Caching: optimizing performance by storing frequently accessed data in memory
- Error handling: managing and reporting errors and exceptions
- Transaction management: ensuring data consistency

Cross-cutting concerns are often implemented using aspect-oriented programming (AOP) techniques, which allow developers to modularize and encapsulate these concerns separately from the primary functionality of the software system. This can reduce code duplication, and make the system easier to maintain and evolve over time.

- AOP achieves this by introducing the concept of aspects. Each aspect encapsulates a specific behavior or logic and can be applied to multiple code modules or classes.
- AOP aspects are defined separately from the core program logic and can be woven into the program at various stages, such as compile-time or runtime.
- AOP differs from typical procedural programming and object-oriented programming, which typically has concerns interspersed throughout the code, leading to code duplication, increased complexity, and reduced reusability.

Accessibility

The quality/ability/extent of being accessible.

Accessibility refers to the ability of a system to be easily accessed and used by people with disabilities or special needs. Accessibility is an important consideration for any system or application that aims to accommodate a diverse user base and promote inclusivity. It is a mandatory requirement for public sector websites and applications.

As a system quality attribute, accessibility refers to the degree to which a system is designed and developed in a way that enables users with disabilities, or special needs, to access and use it. This includes a system's ability to provide alternative input and output mechanisms, such as voice commands or text-to-speech, as well as its compatibility with assistive technologies such as screen readers and Braille displays.

As a non-functional requirement, accessibility is an essential consideration when designing and developing systems and applications that are accessible to people with disabilities. It requires a considered and strategic approach to support the different forms of accessibility.

As a cross-functional constraint, accessibility affects all aspects of the system, from design to implementation and testing. It requires collaboration between designers, developers, and quality assurance teams to ensure that the system meets accessibility standards and guidelines, and complies with company policies and governmental laws.

Define accessible: Accessible in the context of computers and software refers to ensuring that individuals with disabilities can utilize technology in the same manner as their non-disabled counterparts. This includes making digital information and technology easily accessible to people with visual, auditory, physical, or cognitive disabilities, such as by incorporating assistive technology and features into hardware and software design.

Accountability

The quality/ability/extent of being accountable.

Accountability refers to the ability of a system to have clear identification and tracking of responsibilities, actions, and outcomes related to a system or process.

As a system quality attribute, accountability assesses the reliability and trustworthiness of a system, as it ensures that individuals or entities involved in a given process are responsible and answerable for their actions or decisions.

As a non-functional requirement, accountability sets expectations for the performance and security of a system, specifying that it must provide clear records and audit trails, access controls, and other measures that mitigate the risk of fraud, corruption, or errors.

As a cross-functional constraint, accountability affects several aspects of a system's design and operation, such as governance, risk management, compliance, and legal requirements. It often involves collaboration and alignment between different stakeholders, including developers, administrators, auditors, and users, and may require specific policies, procedures, or tools to ensure its proper implementation.

Define accountable: Accountable, in the context of computers and software, refers to the state or capability of a system or application to record and report on the actions of its users or components. It means that every action taken within the system or application can be traced back to the person or entity responsible for it. This can include logging user activity, tracking data changes, and maintaining an audit trail of all system events. This feature is typically implemented to ensure transparency, meet regulatory requirements, and enable effective troubleshooting and incident response.

Accuracy

The quality/ability/extent of being accurate. Compare precision.

Accuracy is a system quality attribute, which refers to the degree to which the system produces correct and reliable results. It measures how closely the output of the system matches the expected output. Accurate systems provide reliable and trustworthyresults and may also enhance user confidence and satisfaction.

Accuracy is a non-functional requirement, and affects many aspects of the system and depends on the interplay of various components and processes. It is often linked with other system quality attributes such as completeness, consistency, and correctness, and is related to factors such as data quality, input validation, and error handling. Accurate results are essential in systems where decisions are based on the output, such as financial, medical, or safety-critical systems.

Accuracy is a cross-functional constraint, ensuring accuracy in a system requires careful design, testing, and validation, with collaboration among stakeholder teams such as business analysis and subject-matter experts (SMEs). This includes verifying data inputs, detecting and handling errors, validating algorithms and calculations, and establishing quality control mechanisms. Accuracy may have legal, contractual, and financial implications, thus involving an organization's lawyers, compliance officers, and risk management planners.

Define accurate: Accurate refers to the ability of computers and software systems to produce results that are free from errors or discrepancies and are very close to the true or expected value. In other words, accuracy is the degree of conformity of the output of a system to the correct or expected value. It is a critical measure of the reliability and effectiveness of computer-based systems, as accurate information is essential for decision-making, analysis, and other important tasks. The accuracy of a system depends on various factors, including the precision of the algorithms used, the quality of the data input, and the limitations of the hardware being used.

ACIDity

The quality/ability/extent of being ACID.

ACID is an acronym for Atomicity, Consistency, Isolation, and Durability. These properties guarantees reliable processing of data transactions.

Atomicity: the property of a transaction that ensures that either all changes made during the transaction are committed or none at all. If a transaction fails for any reason, all the changes made within it are rolled back, returning the system to its original state.

Consistency: the fact that the system's data state will remain valid before and after a transaction. This means that the rules and constraints defined in the database must be followed consistently.

Isolation: the property that ensures that a transaction runs independently of other transactions. This is important to prevent interference from other transactions and errors that could result from concurrent access to data.

Durability: the property that guarantees that once a transaction has been committed, it is permanently saved even in the case of system failures or crashes.

ACIDity is a system quality attribute, which is a property or characteristic of a system that describes its overall quality, especially with regard to databases and transactions.

ACIDity a non-functional requirement, as it describes how the system should behave overall. For some high scalability systems, ACID is difficult to achieve practically or cost-effectively, and may be superseded by eventual consistency.

ACIDity is typically not an important cross-functional constraint, because in practice many organizations are in favor of ACID and using it regularly.

Define ACID: ACID refers to a set of properties that guarantee that

database transactions are processed reliably. These properties ensure that data remains accurate, reliable, and consistent, even when multiple users are simultaneously accessing and modifying data.

Adaptability

The quality/ability/extent of being adaptable. Compare adaptivity.

Adaptability is a system quality attribute that refers to the ability of a system to have its users adjust or modify its behavior or structure in response to changing environmental conditions. This can include changes to the system's input or output, its user interfaces, its resources or processing capabilities, or its communication protocols.

As a non-functional requirement, adaptability is a qualitative attribute that describes a system's user's ability to change the system effectively. It is often an essential consideration for systems with users who need to function under different operating conditions, where specific environmental factors (such as load fluctuation, network connectivity, and data volume) might impact system performance.

Adaptability is also a cross-functional constraint that can affect various parts of the system's design, including user interfaces, and data management. For example, adaptable user interfaces enable users to modify the system to accommodate changing needs. In the same way, adaptable data management should be capable of having users change data structures, data business logic, and so forth.

Define adaptable: Adaptable means capable of adjusting or modifying to fit different situations or conditions as needed, especially in response to changing circumstances without significant difficulty. In the context of computers and software, adaptable refers to the ability of software or hardware to easily and efficiently adjust to new or changing requirements, configuration, or interfaces without requiring extensive reconfiguration or coding changes. Adaptable systems can easily integrate new features, upgrades, or changes to the environment without compromising performance, user experience, or stability.

Adaptivity

The quality/ability/extent of being adaptive.

Adaptivity is a system quality attribute that refers to the ability of a system to respond and adapt to changes in its environment or requirements. Compare adaptability and antifragility.

Adaptivity is a non-functional requirement because it doesn't relate to the specific functionality of a system, but rather the system's ability to adapt to changes in its environment, such as an increasing need for system resources, or service degradations in system dependencies.

Examples of postive-oriented adaptivity are autoscaling and elastic clouds, such as when a system can automatically allocate more processing power, memory, storage, bandwidth, and the like.

Examples of negative-oriented adaptivity are throttling and backpressure, such as when a system can slow its inbound queues, as well as report the issue to upstream providers, in order to prevent overloading.

Adaptivity is a cross-functional constraint, in practice, because different organization teams have different perspectives on the value of adaptivity, and it can be challenging to understand. As a typical example, a sales team may want a web sales process that is always available, even during peak usage periods that may be ten times higher than normal; this can be solved by the sales team and development team coordination on the adaptivity qualities.

In summary, adaptivity is a critical system quality attribute that enables a system to remain effective and robust in the face of changes and uncertainties. It is essential for systems that operate in complex or dynamic environments and must balance competing requirements and constraints to achieve their objectives.

Define adaptive: Adaptive in the context of computers and software refers to the ability of the system to adjust and respond to changing

circumstances. Adaptive software is designed to gather information about its usage, and then use that information to modify its own behavior to better meet the system's goals.

Administrability

The quality/ability/extent of being administerable.

Administrability is a system quality attribute that refers to the ease with which a system can be managed and maintained by system administrators or operators. An administrable system is one that allows administrators to perform tasks such as configuration, deployment, monitoring, and troubleshooting with ease. An administrable system is important because it reduces the time and resources required for administration, increases system reliability and availability, and improves user satisfaction.

Administrability is also considered a non-functional requirement as it focuses on how the system should perform rather than what it should do for end-users. To enhance administrability, a system should be designed with features such as user-friendly interfaces, clear documentation, automated processes, efficient logging and reporting, and effective security controls.

Administrability is a cross-functional constraint because it affects the entire system and requires collaboration between different roles such as developers, testers, and stakeholder administrators. Additionally, system administrators should be involved in the development process from the start to identify potential issues and ensure that the system is designed with manageability in mind.

Define administerable: Administerable refers to hardware or software that can be managed and controlled by an administrator or an authorized user. This control can include tasks such as installation, maintenance, updates, configurations, and security settings. An administerable system is designed to allow an administrator to maintain and manage the system remotely, making it easier to keep the system running smoothly and securely.

Affordability

The quality/ability/extent of being afforable.

Affordability is a system quality attribute, which refers to the ability of a system or product to be economically feasible for its intended use or market. In other words, affordability is the measure of how easily accessible and reasonable the cost of a product or service is to its potential customers.

Affordability can be considered as a non-functional requirement, as it directly impacts the usability of the system. If a system is very expensive to build, maintain or operate, it may not be feasible for a large number of users or organizations, and hence become less desirable.

Affordability can also be viewed as a cross-functional constraint, as it affects multiple aspects of the development process, such as design, testing, manufacturing, distribution, marketing, and sales. Affordability is also critically important for product teams that are doing market discovery, customer discovery, competitive research, and business analysis such as SWOT (Strengths, Weaknesses, Opportunities, Threats).

It is essential that the development team considers affordability as a key factor during the entire development lifecycle to ensure that the final product or service is economically viable, useful, and accessible to end-users.

Define affordable: Affordable refers to something that is priced reasonably and within financial reach of a particular customer. In regards to computers and software, affordable could mean products that are priced lower than the average market price or those that offer a good value for money to customers on a budget. Again, what is considered "affordable" depends on each individual's financial situation, needs, and preferences.

Agility

The quality/ability/extent of being agile.

Agility is a system quality attribute that refers to the ability of a system to adapt to changing requirements and environments quickly and efficiently.

Agility is a non-functional requirement because it does not describe the specific functionalities of a system, but rather how well it can operate under certain circumstances. To achieve agility, a system should be designed with modularity, flexibility, scalability, and robustness in mind. Additionally, an agile system requires a strong testing and validation process to ensure that changes can be made while maintaining system integrity and functional requirements. An agile system can easily change and evolve over time to meet new demands, make modifications or enhancements, and respond to unforeseen events.

Agility can also be considered a cross-functional constraint because it affects multiple stakeholders of a system, including its developers, project managers, product leaders, and accountable executives. Notably, Agile is a software development methodology, and an approach to project management, that emphasizes cross-functional collaboration, flexibility, continuous improvement, and customer satisfaction.

Define agile: Agile an approach to project management. It relies on iterative and incremental development, quick responses to changes and feedback, and effective communication among team members. Agile principles prioritize delivering value to the customer and achieving business goals through adaptive planning, regular inspection, and adaptation of the project scope, design, and requirements. Agile methods are widely used in software development, and they are also applicable to other industries and domains seeking to increase efficiency, innovation, and quality in their products and services.

Antifragility

The quality/ability/extent of being antifragile.

Antifragility is a system quality attribute that refers to the ability of a system to thrive and improve under stress, shock, or change, instead of just surviving it. It is a property of systems that have the ability to constantly adapt and grow stronger in the face of volatility, uncertainty, complexity, and ambiguity of their environment. Compare self-sustainability.

As a non-functional requirement, antifragility is typically related to the resilience, reliability, and robustness of a system. It defines the system's ability to withstand unexpected or unforeseen changes in its environment, mitigate risks and failures, and recover quickly from any disruption.

Antifragility is also a cross-functional constraint because it impacts various aspects of a system, such as its design, architecture, operations, maintenance, and security. It requires the collaboration of different teams and disciplines, such as development, testing, operations, security, and business, to ensure that the system is built, deployed, and managed in a way that enhances its antifragile properties.

Overall, antifragility is a critical quality attribute that organizations should strive to achieve in modern-day systems that operate in highly dynamic and unpredictable environments. By embracing antifragility, organizations can increase their resiliency and agility, gain a competitive advantage, and improve their overall business performance.

Define antifragile: Antifragile is a term coined by Nassim Nicholas Taleb in his book "Antifragile: Things That Gain from Disorder." It describes a system or entity that becomes stronger and more resilient when exposed to stressors or adversity, as opposed to simply withstanding them or breaking under the pressure.

Atomicity

The quality/ability/extent of being atomic transactionally.

Atomicity is a system quality attribute that refers to the property of a system where a sequence of operations is treated as a single, indivisible transaction. In other words, the atomicity of a system is the guarantee that a series of related operations will either succeed or fail as a single unit, without any partial completion.

Atomicity is also considered a non-functional requirement because it does not deal with the specific functionalities of the system, but rather with its overall behavior and performance. It is a critical non-functional requirement for systems that handle complex transactions, such as finance and banking.

Atomicity can be a cross-functional constraint, because most organizational teams want atomicity, and rely on it. However, some high-scalability systems may need to change from atomicity to other approaches such as eventual consistency, or out-of-order event stream processing, or probability-weighted transactions.

To achieve atomicity, a system should have mechanisms that ensure that transactions are executed reliably, securely, and consistently. These mechanisms should support rollback and recovery, isolation, and consistency of the data, and ensure that the system is fault-tolerant and available.

Define atomic: An atomic transaction is a data transaction in which a series of data operations are treated as a single, indivisible operation. In other words, either all of the operations are executed and the transaction is committed, or none of the operations are executed and the transaction is rolled back. This guarantees that data are always in a consistent state, even if an error occurs during the transaction. Atomicity is one of the four key properties of an ACID data transaction.

Auditability

The quality/ability/extent of being auditable.

Auditability is a system quality attribute that refers to the ability of a system to provide reliable and accurate data for auditing purposes. Auditability is important in many industries, such as finance, healthcare, and government, where strict regulations often require auditing of system activities and transactions. Auditability can also help organizations to identify issues and improve processes.

Auditability is a non-functional requirement that is typically also a cross-cutting concern, because a system must be designed with clear and transparent processes that can be easily tracked and recorded, across all functions and modules. The system should store and maintain appropriate logs, records, and metadata that can be used to reconstruct system activities over time; this includes data such as user activity, access logs, transaction records, and system changes. Audit logs can be stored and analyzed to detect security breaches, fraud, data breaches or any other potential risks to the system or organization.

Auditability is a cross-functional constraint, requiring collaboration between various departments such as development, operations, cybersecurity, physical security, legal, compliance, and risk managment. The system must be designed with security in mind to ensure that data is not compromised or lost, and compliance regulations are met. It must also be easy to audit, with clear documentation and reporting that can be easily reviewed by auditors or other stakeholders.

Define auditable: Auditable refers to the capability of tracking events or changes in a system, so the tracking can be reviewed later for compliance, security, or regulatory purposes.

Automatability

The quality/ability/extent of being automatable.

Automatability is a system quality attribute that refers to the ease with which a system can be automated or integrate with automation tools.

Automatability is a non-functional requirement that involves creating software that can easily automate its processes, i.e., make it easy to integrate with scripts or other automated tools. Automatability can save time, extend capabilties in new ways, and potentially reduce costs.

Automatability is a cross-functional constraint that affects developers, testers, and system administrators who can automate their workflows with the help of automation tools. In addition, automatability is gaining in importance thanks to low-code/no-code tools, as well as natural language queries.

Define automatable: Automatable refers to the capability of a system or process to be executed automatically by a computer or software without human intervention. An automatable process should have well-defined inputs, outputs, rules, and steps that can be accurately captured and executed by software tools or scripts. By automating repetitive or time-consuming tasks, organizations can achieve greater efficiency, consistency, and scalability in their operations. Common examples of automatable processes include data entry, file processing, testing, deployment, and monitoring.

Autonomicity

The quality/ability/extent of being autonomous.

Autonomicity is a system quality attribute that refers to a system's ability to self-manage, self-heal, and self-configure without the need for human intervention. Autonomicity includes other system quality attributes, such as adaptivity and antifragility.

Autonomicity is a non-functional requirement that emphasizes the importance of a system's ability to operate seamlessly and adapt to its environment to achieve its goals. Autonomicity can be especially valuable to improve other non-functional requirements, such as availability, dependability, reliability, and scalability.

Autonomicity is a cross-functional constraint because it affects multiple aspects of a system, including its performance, reliability, and security. It requires collaboration among different stakeholders, such as developers, system administrators, and end-users, to ensure that the system is fully autonomous in its operation.

Define autonomicity: Autonomous refers to systems, computers, and software that have the ability to operate and function independently without requiring human intervention or control. Autonomous systems can be programmed to perform specific tasks and make decisions based on pre-determined criteria, and can continue to function even in the absence of a human operator. They utilize artificial intelligence and machine learning algorithms to analyze and respond to data, allowing them to adapt and improve their performance over time. Examples of autonomous systems include self-driving cars, drones, and automated customer service chatbots.

Availability

The quality/ability/extent of being available.

Availability is a system quality attribute that refers to the ability of a system or application to be accessible and operational for use by authorized users. It indicates the percentage of time that a system or application remains available and responsive to user requests.

Availability is a non-functional requirement, which means that it is a specification of how the system should behave rather than what it should do. It is an important factor for systems that require high up-times, such as critical systems in healthcare, finance, or transportation.

Availability is also a cross-functional constraint, meaning that it affects multiple aspects of the system. It can impact the performance, scalability, and reliability of the system, as well as the user experience and overall satisfaction.

To maintain high availability, systems need to be designed with redundancy, fault-tolerance, and disaster recovery mechanisms. They also need to be monitored and maintained regularly to ensure that any issues are identified and resolved quickly.

Define available: Available refers to computers and software that are currently accessible and ready for use. These resources can be either physical or virtual and are typically ready to be utilized at any time without requiring extensive setup or configuration. Availability may be affected by factors such as network connectivity, hardware reliability, and software compatibility. In general, available computers and software are considered to be reliable resources that can be used to perform a variety of tasks, ranging from basic productivity work to complex analytical tasks.

Compatibility

The quality/ability/extent of being compatible.

Compatibility is a system quality attribute, non-functional requirement, and cross-functional constraint that ensures the system can function correctly and cohesively with other hardware, software, networks, and operating systems. Compatibility is essential for systems to communicate and exchange data with each other seamlessly.

Compatibility can be tested and measured by ensuring that the system performs as expected in different environments, configurations, and platforms. Compatibility testing involves testing the system's ability to coexist with other systems and to function flawlessly in various scenarios.

Compatibility also helps to ensure that the system can evolve and adapt to new technologies, trends, and standards. It is a crucial factor that impacts the system's reliability, performance, and usability.

In summary, compatibility is an essential system quality attribute, non-functional requirement, and cross-functional constraint that enables the system to work cohesively with other systems and technologies. It ensures that the system is interoperable, adaptable, and functional.

Define compatible: Compatible refers to the ability of computers and software to work together without any technical issues or conflicts. It means that they are designed to interact seamlessly and can exchange information without any errors or problems. This is important for system performance and efficiency, as well as for the user experience. Incompatible systems may cause crashes, errors, or data loss, making it difficult or impossible to accomplish the intended tasks.

Composability

The quality/ability/extent of being composable.

Composability is a system quality attribute that determines the degree to which different components or services within an architecture can be combined or composed to create new functionalities or systems without requiring significant changes to the individual components. In other words, composability is the ability of a system or a component to be combined or integrated with other systems or components in a flexible and seamless manner, allowing for easy customization, scalability, and interoperability.

Composability is essential for complex software systems with multiple components or services, as it enables teams to build and maintain systems that are flexible, modular, and easily adaptable to changing requirements. It is also critical for organizations that rely on third-party software components or services, as it ensures that these components can be easily integrated into the organization's existing technology infrastructure.

Overall, composability enables organizations to build and maintain software systems that are scalable, adaptable, and interoperable. It requires careful consideration of the design and architecture of the system, as well as the selection of the appropriate components and services that can be easily integrated and combined to create new functionalities or systems.

Define composable: Composable in the context of computers and software refers to the ability of different software components to be deconstructed into smaller, standalone components that can be combined and orchestrated in new ways to create custom applications and services. This approach enables users to rapidly build and deploy applications by combining pre-existing components to meet their specific business needs. Composable architecture is characterized by its ability to create applications that are highly flexible, scalable and can easily accommodate new changes or updates.

Configurability

The quality/ability/extent of being configurable.

Configurability is a system quality attribute that refers to the ability of a system to be easily customized or adapted. Configurable systems are designed to be flexible, adaptable, and user-friendly. Additionally, configurable systems often have a range of pre-built options that can be easily adjusted to match different use cases. Compare customizability.

Configurability is a non-functional requirement that specifies the degree to which a system can be modified to suit different user needs, preferences, and changing environments. Configurability may involve multiple aspects of the system, including its architecture, design, implementation, and maintenance. It requires a system to have a modular and flexible design that allows for the addition or removal of components without affecting other parts of the system.

Configurability is a cross-functional constraint, especially when stakeholder teams need to adapt to changing business requirements, market forces, or user preferences. High configurability enables stakeholder teams and end-users to tailor the system to their specific needs, which and ensures that the system remains relevant and valuable over time.

Define configurable: Configurable refers to a feature or capability of computer hardware or software that can be easily customized or adjusted according to the user's preferences or requirements. This means that users can modify certain settings or options within the system to meet specific needs, such as changing the language, font size, screen resolution, or input/output devices.

Consistency

The quality/ability/extent of being consistent.

Consistency is a system quality attribute that denotes a requirement for a system to provide reliable and uniform behavior across different contexts and time periods. It refers to the degree of similarity between system components, interfaces, interactions, and functionalities, which can make the system more usable, predictable, and maintainable.

As a non-functional requirement, consistency is an essential aspect of the user experience, as it ensures that the system meets the users' expectations and needs without causing confusion or errors. A consistent system is more intuitive and familiar to users, reducing the learning curve and the chances of making mistakes. It also enhances transparency and accountability, enabling users to track and understand the system's behavior and outcomes.

Consistency is also a cross-functional constraint that affects various aspects of the system development and operation, such as design, coding, testing, documentation, and maintenance. It requires cooperation and collaboration among different stakeholders, such as designers, developers, testers, users, and managers, to ensure that the system's consistency is maintained throughout its life cycle. Examples of techniques and tools that support consistency include coding standards, style guides, user feedback, version control, test automation, and change management processes.

Define consistent: In the context of computers and software, consistent refers to the state of being uniform or constant in behavior, appearance, or performance. A consistent software application behaves in the same way each time it is used, providing predictable and reliable results. Similarly, consistent user interfaces across different applications or devices make it easy for users to navigate and understand the system. Consistency in coding practices makes it easier for developers to maintain and modify code over time. Overall, consistency in technology is essential for ensuring efficient and effective outcomes.

Continuity

The quality/ability/extent of being continuous.

Continuity refers to the system quality attribute that describes the ability of the system to keep functioning under normal and abnormal conditions, without any noticeable impact on the performance and availability of the system.

Continuity is a non-functional requirement that specifies the expected level of reliability, availability, and maintainability of the system. The continuity of a system is a cross-cutting concern that affects various aspects of the system, including design, development, testing, deployment, and operation.

The continuity of a system encompasses the ability to recover from failures, handle unexpected events, and maintain its functionality and performance in the face of disruptions. It requires that the system design and architecture incorporate redundancy, fault tolerance, and resilience mechanisms that can handle various types of failures and errors. The system should also be capable of monitoring and detecting faults, and initiating appropriate corrective actions.

Continuity is critical in many domains, including financial services, transportation, healthcare, and emergency services, where service disruptions can have severe consequences. It is also essential in industrial and manufacturing settings, where downtime and equipment failures can lead to significant losses in productivity and revenue.

Overall, ensuring the continuity of a system requires a comprehensive approach that involves understanding the system's context, identifying potential risks and failures, and designing appropriate mitigation strategies. It is a vital aspect of system reliability and availability, and an essential requirement for meeting the expectations of users and stakeholders.

Define continuous: Continuous in the context of computers and software refers to a process or methodology in software development

where the code is continuously built, tested, and deployed to production. The developers work on small incremental changes to the codebase, and with each change, the code is automatically tested and deployed. This process helps teams to detect and fix issues and bugs quickly, thereby delivering new features and updates to the end-users rapidly. Additionally, it also helps in increasing the overall reliability and quality of the software product.

Controllability

The quality/ability/extent of being controllable.

Controllability is a system quality attribute that refers to the ability of a system or its components to be easily managed, monitored, and controlled, typically by its users, administrators, and other stakeholders.

Controllability is a non-functional requirement, and it impacts various aspects of a system, including usability, reliability, and security. Controllability may also be a cross-cutting concern, for example if controls span multiple areas of the system.

Controllability is also a cross-functional constraint: TODO

In practical terms, controllability means that a system should provide interfaces, tools, and mechanisms that enable users to manage and control its behavior, performance, and other aspects. For example, an operating system should provide a user-friendly interface to allow users to manage system resources, such as memory, disk space, and CPU usage. Likewise, a database management system should provide tools to monitor and maintain data integrity, performance, and security.

A system that is difficult to control and manage can lead to operational issues, such as downtime, data loss, and security breaches. Therefore, system designers and business stakeholders should work toward systems that are easy to control and manage.

Define controllable: In the context of computers and software, controllable refers to the ability of a user or program to manipulate and manage the behavior and operations of the system. This can include changing settings, configuring options, selecting inputs and outputs, and executing commands to achieve desired outcomes. It may also refer to the degree to which a system can be easily monitored and modified by an administrator or other authorized user. Overall, controllability is a key aspect of ensuring that a computer or software system is effective, efficient, and optimally performing its intended functions.

Correctness

The quality/ability/extent of being correct.

Correctness is a system quality attribute that refers to the accuracy and precision of the system's output or behavior with respect to its intended purpose or specification. Correctness includes related system quality attributes of accuracy and precision.

Correctness is a non-functional requirement that is crucial for ensuring the reliability, usability, and security of the system. The correctness of a system depends on its ability to handle different input scenarios, detect and handle errors and exceptional conditions, fulfill the user's expectations, and maintain consistency and coherence of its data and behavior.

Correctness is a cross-functional constraint: TODO

Achieving correctness requires a systematic and rigorous approach to the system's design, development, testing, and maintenance. It involves identifying and validating the system's requirements, ensuring the completeness and consistency of its design, implementing and testing the system's functionality and behavior, and monitoring and maintaining the system's quality over time.

Define correct: In the context of computers and software, "correct" refers to the accuracy and functionality of a program or system in performing its intended tasks without errors or issues. A correct program will operate as intended, producing accurate results in accordance with its specifications and without causing any unexpected issues or errors.

Credibility

The quality/ability/extent of being credible.

Credibility is a system quality attribute, a non-functional requirement, and a cross-functional constraint in software development. It refers to the degree to which users, stakeholders, and customers can trust the system or its information. It is a measure of the system's perceived reliability, accuracy, and consistency.

As a system quality attribute, credibility is essential for user satisfaction and overall system success. It is critical in domains such as healthcare, finance, and security, where incorrect information or unreliable systems can have severe consequences.

As a non-functional requirement, credibility must be addressed in the system design and implementation. The system must be designed to ensure that data is accurate and reliable, and that the user interface is user-friendly and transparent. System testing and validation are critical for verifying the credibility of the system.

As a cross-functional constraint, credibility impacts different areas of the system, including data management, user experience, and system performance. It requires collaboration and communication across different teams, such as the development team, data management team, and quality assurance team, to ensure that credibility is maintained throughout the development lifecycle.

In conclusion, credibility is a crucial attribute for any software system. It needs to be addressed as a non-functional requirement, a system quality attribute, and a cross-functional constraint to ensure that it is maintained throughout the development lifecycle.

Define credible: Credible, in the context of computers and software, refers to the trustworthiness and reliability of a system, application or information source. Credibility is achieved when a system or software accurately performs the intended function, operates correctly, and without errors or vulnerabilities. Additionally, a credible system or

software should be backed up by a reputable company or organization that takes responsibility for its development and maintenance, and has a track record of delivering quality products. In general, users have confidence in credible computer software and systems and depend on them to perform their intended tasks.

Customizability

The quality/ability/extent of being customizable.

Customizability is a system quality attribute that refers to the ability of a system to be easily modified or adapted to different user needs and preferences.

Customizability is a non-functional requirement because it doesn't describe what the system should do, but rather how it should behave. It is also a cross-functional constraint because it affects various aspects of a system, such as user interface design, software architecture, and data management.

Customizability is important in software systems because it allows users to tailor the application to their specific requirements, making it more efficient and effective. For example, a customizable email client may allow users to organize their inbox in different ways, change the color scheme, or add new features like spell-checking or message threading.

To achieve customizability, software systems must be designed with flexibility and modularity in mind. This may require the use of modular architectures, flexible data structures, and open APIs that allow for easy customization of the system. Similarly, user interfaces must be designed with customization in mind, providing users with intuitive tools to modify and personalize the system to their liking.

Define customizable: Customizable in the context of computers and software refers to the ability to modify or tailor the functionality, appearance, and features of a particular hardware or application according to specific user preferences or requirements. It allows users to customize or personalize their devices or applications according to their unique needs, interests, and work processes to optimize their user experience and improve efficiency. Some examples of customizable features include changing the desktop background, adding or removing toolbars, customizing keyboard shortcuts, and configuring software settings.

Debugability

The quality/ability/extent of being debuggable.

Debugability is a system quality attribute, which is also considered a non-functional requirement and cross-functional constraint. It refers to the ease of finding and fixing bugs or errors in a software system. Debugability enables developers to identify and correct errors quickly and efficiently, which can save time and reduce costs in the long run.

A system with good debugability is easier to maintain, and issues can be resolved swiftly, reducing downtime, and increasing overall system performance. Debugability is achieved through the use of proper logging and tracing, debugging tools, and automated testing frameworks. It also requires good documentation, well-designed code, and proper error handling.

Debugability is essential for all software systems and is particularly critical for mission-critical applications. It is a fundamental system quality attribute that helps ensure that software is reliable, maintainable, and resilient, providing users with a seamless, error-free experience.

Define debuggable: Debuggable refers to a software application or program that is designed to be easily and effectively diagnosed and corrected for any bugs, errors, or problems that may arise during its development or use. It is a quality that allows developers, testers, and system administrators to locate and fix any issues that may occur in the application, ensuring its optimal performance and functionality. Debuggable applications are essential for minimizing downtime, improving user experience, and maintaining the overall health and stability of software systems.

Degradability

The quality/ability/extent of being degradable.

Degradability is a system quality attribute that refers to the ability of a system to operate at an acceptable level of performance even when certain components or resources are unavailable.

Degradability is a non-functional requirement that is essential for systems that are critical to business operations or public safety.

Degradability is a cross-functional constraint that affects multiple aspects of a system's design, implementation, and maintenance. It requires careful consideration of factors such as fault tolerance, redundancy, load balancing, and failover mechanisms.

A system that is not sufficiently degradable may experience catastrophic failure in the event of an unexpected outage or overload. This can lead to significant financial losses, loss of life, or damage to the reputation of the organization that operates the system.

In contrast, a highly degradable system can continue to operate even during adverse conditions, allowing critical functions to continue without interruption. This can help to increase the reliability, availability, and maintainability of the system, which are all important considerations for ensuring business continuity and public safety.

Define degradable: Degradable refers to computer hardware or software that progressively loses functionality or performance over time. It may occur due to aging, obsolescence, wear and tear, or becoming outdated as newer and faster technology becomes available. Degradable components may cause computer systems to operate slower, glitch, or eventually fail altogether. Upgrading or replacing degradable parts is necessary to maintain system performance and prevent further degradation.

Demonstrability

The quality/ability/extent of being demonstrable.

Demonstrability is a system quality attribute that represents the ability of a system to demonstrate its functionality and capabilities to its users. It is a non-functional requirement that specifies how well the system can provide evidence or proof of its correct functioning, as well as its ability to trace and reproduce errors and failures.

Demonstrability is also considered as a cross-functional constraint as it influences different aspects of the system development process. It requires collaboration between developers, testers, and subject matter experts to create a system that is easy to demonstrate and validate to different stakeholders.

A system with high demonstrability enables users to interact and verify the system's behaviors, inputs, and outputs. It involves providing necessary evidence, documentation, and tools to enable users to understand the system's purpose, operation, and performance.

Overall, demonstrating the system's functionality and capabilities is essential for ensuring user confidence and trust in the system, and therefore demonstrating this ability is crucial to the success of a system.

Define demonstrable: Demonstrable means capable of being shown or proven with evidence. In the context of computers and software, it refers to a feature or function that can be clearly demonstrated or proven to work as intended. This can involve providing clear documentation, testing, or examples to demonstrate that a particular program, software module, or component meets the requirements or specifications it was designed to satisfy. Demonstrability is an important factor in software development, as it enables developers to verify that their software is functioning correctly and meeting user needs.

Dependability

The quality/ability/extent of being dependable.

Dependability is a system quality attribute that refers to the ability of a system to consistently and reliably perform its intended function under normal and abnormal conditions. It encompasses a range of related qualities such as reliability, availability, maintainability, safety, and security.

As a non-functional requirement, Dependability sets the expectation for the system's performance in terms of reliability, availability, maintainability, safety, and security. It is a crucial factor in ensuring system usability, trustworthiness, and customer satisfaction.

Dependability is also a cross-functional constraint that affects various aspects of the system development lifecycle. It requires comprehensive design, testing, and validation of the system to ensure its reliability, availability, and safety. It may also impact the choice of technologies, standards, and techniques used in system development, testing, and maintenance.

Overall, Dependability is a critical system quality attribute that ensures the system's consistent and reliable operation, enhances its trustworthiness and usability, and promotes customer satisfaction.

Define dependable: Dependable refers to the reliability and consistency of a software program or computer system to consistently produce accurate results with a low probability of failure, error or unexpected behavior, even under adverse conditions, such as heavy loads or unexpected inputs. A dependable software or system is one that can be trusted to work as expected and meet the intended goals with a high level of confidence. Dependable systems and software are critical in industries such as healthcare, aviation, and finance, where unexpected errors or glitches can lead to catastrophic consequences or loss of life.

Deployability

The quality/ability/extent of being deployable.

Deployability refers to the ease and speed with which a software system can be deployed or installed. It is a system quality attribute that measures the system's ability to be deployed in different environments and configurations, without requiring extensive development efforts or causing disruptions to other systems.

Deployability is considered a non-functional requirement, as it focuses on the performance and behavior of the system, rather than the specific features or functionalities. It is often linked to other non-functional requirements, such as reliability, scalability, and maintainability, as a highly deployable system is usually more reliable, efficient, and adaptable to changing demands.

Deployability is also a cross-functional constraint, as it involves collaboration and coordination among various teams and stakeholders involved in the deployment process. These may include developers, operators, security experts, and end-users, who need to work together to ensure a smooth and seamless deployment experience.

In summary, deployability is a critical system quality attribute that ensures that software systems can be easily and quickly deployed in various environments, without causing disruptions or requiring extensive development efforts. It is a non-functional requirement that depends on other factors such as reliability, scalability, and maintainability, and is a cross-functional constraint that requires collaboration and coordination among various teams and stakeholders.

Define deployable: Deployable refers to computers and software that can be easily and quickly deployed or implemented in a specific environment or system. This means that the device or application can be configured and installed without requiring extensive setup or configuration, allowing for rapid deployment and scaling of a solution. These technologies are typically designed to be highly portable and

modular, making them ideal for use in a variety of settings and contexts.

Determinability

The quality/ability/extent of being determinable.

Determinability refers to the ability of a system or software application to identify, diagnose, and rectify any errors or problems that may arise during its operation. It is a system quality attribute that ensures the reliability and stability of the system. This non-functional requirement is a cross-functional constraint that impacts various aspects of software development, including design, implementation, testing, and maintenance.

Determinability is an essential system quality attribute because it enhances the system's ability to self-diagnose and self-correct any problems that may occur. This attribute contributes to the dependability, fault tolerance, and availability of the system. It also helps to reduce downtime, improve user experience, and increase system efficiency.

To achieve determinability, software developers must design and implement robust error handling mechanisms that can detect and diagnose failures, log error messages, and provide feedback to users. Moreover, they must ensure that the system is capable of recovering from failures and that the recovery process is transparent and reliable.

In summary, determinability is a system quality attribute that ensures the reliability, stability, and robustness of a system. It is a non-functional requirement that impacts various aspects of software development, including design, implementation, testing, and maintenance. Achieving determinability requires robust error handling mechanisms, fault tolerance, and reliable recovery processes.

Define determinable: Determinable in computers and software refers to the quality of being able to be determined, resolved or ascertained. It suggests that a particular software feature, attribute or function can be clearly defined, analyzed, and measured, either by using specific tools, techniques or approaches. In other words, a determinable software component is one that can be quantified, tested, and evaluated based on

certain criteria, such as performance, reliability, security, efficiency or scalability, among others. This attribute is essential in software development, as it allows developers to design, implement, and maintain software solutions that meet specific requirements and deliver measurable outcomes.

Discoverability

The quality/ability/extent of being discoverable.

Discoverability is a system quality attribute that refers to the ease with which users can find and access functionality within a system, particularly for new users who may be unfamiliar with the interface or functionality. It is a non-functional requirement that is critical to the usability of a system and can significantly impact user satisfaction and overall system performance.

Discoverability is also a cross-functional constraint that affects all aspects of system design, including the user interface, navigation, and information architecture. It requires careful consideration of user needs and expectations, as well as an understanding of how users interact with the system.

To enhance discoverability, designers and developers can employ a range of strategies, such as clear and intuitive labeling, consistent navigation and layout, and context-sensitive help and feedback. By prioritizing discoverability as a system quality attribute and non-functional requirement, organizations can improve the overall usability and effectiveness of their systems, while also enhancing user satisfaction and engagement.

Define discoverable: Discoverable refers to the ability of computers and software to be found and recognized by other devices or software on a network. For instance, when a device is discoverable, it means other devices on the same network can detect its presence, exchange data, and communicate with it. This feature is essential for wireless connections like Bluetooth and Wi-Fi, where devices need to identify and connect with each other. In software, discoverability refers to the ease with which users can locate and access certain functions or features within an application. A well-designed application will have easy-to-find commands or menus that make it straightforward for users to navigate and perform tasks.

Distributability

The quality/ability/extent of being distributable.

Distributability refers to the capability of a system or application to be distributed over multiple nodes or servers, enabling it to provide higher availability, scalability, and fault-tolerance.

Distributability is a system quality attribute that emphasizes the importance of sharing workloads among different nodes or servers, thus avoiding single points of failure and reducing the risk of system downtime.

As a non-functional requirement, distributability demands that the system architecture is designed in such a way that it can distribute and balance workloads efficiently without sacrificing performance or security.

Distributability can also be viewed as a cross-functional constraint because it affects multiple areas of the system design and implementation, such as network topology, data storage, communication protocols, and security mechanisms. Therefore, the ability to distribute a system and its components is an important consideration for architects, developers, and operations teams alike.

Define distributable: Distributable refers to software or computer programs that can be legally distributed or shared with others. These programs are typically designed to be installed on multiple devices and used by multiple users. Distributable software often includes licensing terms that specify the conditions under which it can be distributed, such as the number of users or devices allowed to use the program, and any restrictions on its use or distribution.

Durability

The quality/ability/extent of being durable.

Durability is a system quality attribute that refers to the ability of a system to withstand the rigors of use over an extended period of time. It is a non-functional requirement that must be satisfied along with other quality attributes such as performance, reliability, and maintainability. Durability is also a cross-functional constraint that applies to different parts of the system such as hardware, software, and infrastructure.

In general, a durable system is one that can continue to function as expected without experiencing any significant degradation in performance or reliability over time. This requires the use of high-quality materials, components, and design techniques that are resistant to wear, corrosion, and other forms of damage. Additionally, the system must be able to handle the stresses and strains of everyday use without breaking down or developing faults.

Durability is particularly important for systems that are used in high-stress environments such as industrial equipment, transportation systems, and military applications. In these contexts, a failure to meet durability requirements can result in serious consequences such as equipment malfunction, production delays, and even loss of life.

Overall, durability is a critical system quality attribute that must be carefully considered and designed for in any system to ensure that it can perform reliably and safely over an extended period of time.

Define durable: Durable in the context of computers and software refers to the ability of a device or program to withstand wear and tear, perform consistently over a long period of time, and withstand changes or updates in the technological environment without becoming obsolete. Durable computers and software are designed with reliable components and feature regular maintenance to ensure longevity and avoid data loss or corruption.

Effectiveness

The quality/ability/extent of being effective.

Effectiveness is a system quality attribute or non-functional requirement that is described as the capability of a system to perform tasks efficiently and accurately, achieving the intended goals or objectives.

Effectiveness is a crucial system quality attribute, as it measures the success of a system in delivering its intended functionality while satisfying user needs and business requirements. The effectiveness of a system is often measured through performance metrics, such as transaction throughput, response time, error rate, and completion rate.

In addition to being a system quality attribute, effectiveness is often considered as a cross-functional constraint, as it is closely related to other quality attributes such as usability, reliability, and security. A system that is not effective in performing its tasks may also suffer from poor usability, unreliable performance, and security vulnerabilities.

Overall, effectiveness is a key system quality attribute that can significantly impact the success of a system in meeting its intended goals and satisfying user needs.

Define effective: Effective means that a computer or software system performs the tasks it was designed for efficiently and successfully achieves the intended outcome. It is a reflection of how well a tool or technology can achieve its goals while minimizing errors, bugs, and inefficiencies. An effective computer or software system provides users with a positive user experience, is user-friendly and easy to navigate, and is generally reliable in its performance.

Efficiency

The quality/ability/extent of being efficient.

Efficiency refers to the ability of a system to accomplish tasks with minimal waste of resources, such as time, memory, and energy.

Efficiency is a system quality attribute that describes the extent to which a system can provide optimal performance at a reasonable cost.

Efficiency is a non-functional requirement that helps to ensure that a system meets its functional requirements while optimizing its performance.

Efficiency is also a cross-functional constraint that affects different parts of a system, including hardware, software, and human factors. For example, an efficient website must have a fast and reliable server, a well-designed user interface, and optimized code that minimizes the amount of data transmitted over the network.

Ultimately, an efficient system can help to reduce costs, improve productivity, and enhance user satisfaction. Therefore, it is essential to consider efficiency as a key factor in the design, development, and operation of any system.

Define efficient: Efficient in the context of computers and software refers to the ability of a system to achieve its intended purpose or complete tasks quickly and accurately without wasting unnecessary resources, such as processing power, memory, or energy. An efficient system is one that can target its resources effectively and complete tasks with maximum speed and minimum waste. An efficient software program is one that is streamlined, performs its functions accurately, and does not consume excessive memory or CPU resources.

Encryptability

The quality/ability/extent of being encryptable.

Encryptability refers to the capability of a system to secure sensitive data by utilizing encryption methods. In technical terms, encryption is the procedure of transforming plain text data into an unreadable form that can only be decoded with the help of a unique cryptographic key.

Encryptability is a system quality attribute since it is concerned with the system's capability to preserve data confidentiality and integrity. It ensures that data remains secure during transmission and storage, thereby providing protection against unauthorized access, disclosure, or tampering.

Encryptability is also a non-functional requirement, meaning that it specifies how a system should perform rather than defining a specific feature. It is a critical requirement in many systems, especially those associated with sensitive information such as payment processing, medical records, or government data.

In addition, encryptability can be considered as a cross-functional constraint, as it may affect various aspects of a system, including performance, usability, and maintainability. Encryption processes may impact system performance by adding overhead in terms of processing power and resources. However, this overhead may be outweighed by the benefits of data security.

In summary, encryptability is a critical system quality attribute, non-functional requirement, and cross-functional constraint that ensures the confidentiality and integrity of sensitive information.

Define encryptable: Encryptable refers to the ability to transform plain, readable information into a coded format that cannot be understood by any unauthorized party. This feature is crucial in computers and software to ensure secure communication, data protection, and privacy. Encryptable data can only be accessed with a decryption key or password, making it challenging for hackers or cybercriminals to read

or manipulate the information.

Evolvability

The quality/ability/extent of being evolvable.

Evolvability is a system quality attribute that refers to the ability of a software system to adapt and evolve to meet changing business or technological needs. It is a non-functional requirement because it does not directly specify the system features or functionality, but rather its ability to adapt and change over time. It is also a cross-functional constraint because it affects multiple aspects of the system, such as its architecture, design, and development practices.

Evolvability is achieved through the use of modular, scalable, and maintainable software architecture and design principles. This allows the system to be easily modified, extended, or enhanced as new functionality or requirements are introduced. In addition, evolvability requires robust testing and validation procedures to ensure that changes to the system do not introduce new errors or vulnerabilities.

An evolvable system is able to remain relevant and useful over time, even as the business environment or technology landscape evolves. This makes it a valuable asset for organizations that need to stay competitive and responsive to changing market conditions.

Define evolvable: Evolvable refers to the ability of computers and software to adapt and evolve over time through complex algorithms and learning processes that are designed to enhance their functionality and performance. Evolvable software allows for continuous updates and improvements, and can effectively respond to changing user requirements or shifts in market demands. This can lead to greater efficiency, flexibility, and scalability in computing systems, allowing them to meet the demands of a rapidly evolving technological landscape.

Extensibility

The quality/ability/extent of being extensible.

Extensibility is a system quality attribute that represents the system's capacity to extend its functionality or accommodate new features and requirements efficiently. It is a non-functional requirement that describes the ease with which the system can incorporate new functionality without disrupting the existing system's structure or requiring significant changes.

Extensibility is a cross-functional constraint as it impacts various stakeholders involved in the system's development, such as developers, testers, end-users, business analysts, and system administrators. It influences the system's overall architecture, design, and implementation, and affects the system's maintainability, scalability, and adaptability.

In summary, extensibility is an essential system quality attribute that enables a system to incorporate new features and requirements without disrupting existing functionality. It is a non-functional requirement that has cross-functional implications and plays a critical role in the success of a syste.

Define extensible: In the context of computers and software, extensible means that a system or program is designed in a way that allows it to be easily enhanced or modified with new features or functionalities without requiring significant changes to the underlying code or architecture. An extensible system typically provides mechanisms such as APIs (application programming interfaces), plugins, or scripts that developers can use to extend its capabilities and integrate it with other systems, tools, or platforms. This flexibility and adaptability make extensible systems and software ideal for various applications, from complex enterprise systems to consumer applications and mobile apps.

Failure-transparency

The quality/ability/extent of being failure-transparent.

Failure-transparency is a system quality attribute that ensures that a system's failures are communicated to its users, administrators, and other stakeholders in a clear and timely manner. This attribute is essential for building trust in the system and fostering a culture of responsibility and accountability.

As a non-functional requirement, failure-transparency specifies how a system should handle errors, exceptions, and other types of failures. It requires that the system provide detailed and accurate error messages, logs, and other diagnostic information to help users and administrators understand what went wrong and how to fix it.

Failure-transparency is also a cross-functional constraint, meaning that it affects multiple aspects of a system and requires collaboration across different teams and departments. For example, developers must design the system to generate informative error messages, while operations teams must ensure that logs are accessible and easy to analyze.

In summary, failure-transparency is an essential system quality attribute that helps ensure that failures are communicated effectively and efficiently to all stakeholders. As a non-functional requirement and cross-functional constraint, it requires collaboration and coordination across different teams and functions to achieve.

Define failure-transparent: Failure-transparent is a term used in computer science and software engineering to describe a system or software that is resilient and able to handle failures without disrupting or completely stopping the system's operation. A failure-transparent system can detect and handle errors, failures, and unexpected events without compromising the overall functioning of the system. Such a system is designed to provide high availability and fault tolerance, ensuring the continuity of the system's operation even in the face of hardware, software, or network failures. In simple terms, a

failure-transparent system is one that fails gracefully, allowing the system to continue functioning despite any issues or faults that may arise.

Fault-tolerance

The quality/ability/extent of being fault-tolerant.

Fault-tolerance is a system quality attribute that denotes a system's ability to continue functioning in the event of a failure or fault in one or more components.

Fault-tolerance is a non-functional requirement that ensures that a system remains available, reliable, and operational even in the face of hardware, software, or communication failures. Fault-tolerance is a cross-functional constraint that impacts several aspects of system design, including architecture, software design, hardware selection, testing, and fault management.

In essence, fault-tolerance is about ensuring that a system can continue to provide its services despite any failures or interruptions, ensuring that the system remains available and able to serve its users at all times. This requires careful planning and design, including the use of redundant components, error detection and correction mechanisms, and backup systems, among other techniques. The goal is to minimize the impact of potential failures, maintain system integrity, and provide a seamless and reliable user experience.

Define fault-tolerant: Fault-tolerant refers to the ability of computer systems or software to continue functioning, even in the event of a failure or error. In other words, a fault-tolerant system is designed to be resilient and reliable, able to detect and recover from faults or errors automatically, without disrupting overall system performance or compromising data integrity. This can be achieved through various techniques, such as redundancy, backup systems, error detection and correction, and failover mechanisms. The goal of fault-tolerant computing is to maximize system uptime and minimize the risk of system failure, which is critical for mission-critical applications and systems that require high availability and reliability.

Fidelity

The quality/ability/extent of having fidelity.

Fidelity is a system quality attribute that refers to the degree of accuracy and exactness with which a system or its components perform their intended functions or tasks. It is concerned with the system's ability to correctly represent, store, retrieve, and process data or information, as well as its capacity to provide consistent and reliable results. High-fidelity systems are generally more reliable, accurate, and trustworthy, whereas low-fidelity systems are prone to errors, inconsistencies, and inaccuracies.

Fidelity is also a non-functional requirement that specifies the system's performance and behavior characteristics that cannot be directly related to its primary functions or features. It includes aspects such as reliability, availability, maintainability, and usability, which are important for ensuring that the system can be effectively used and operated by its users.

Finally, fidelity can be regarded as a cross-functional constraint, as it affects all aspects of the system design and development process, including software engineering, hardware engineering, user experience design, testing, and maintenance. A system that lacks fidelity may lead to significant performance issues, user frustration, and maintenance challenges, making it crucial for designers and developers to incorporate fidelity considerations into all aspects of the system design and development process.

Define fidelity: Fidelity refers to the degree of accuracy, reliability, and faithfulness in reproducing or representing data, graphics, or sound in digital form. It is used to describe the quality of the output from software or hardware devices such as monitors, speakers, printers, and graphics cards. High fidelity means that the output accurately represents the original input or source material, with minimal loss of detail or distortion. In the context of digital audio, fidelity is also referred to as the quality of the sound and the absence of any noise or

distortion. High-fidelity sound is often associated with high-quality audio equipment and the use of lossless or uncompressed digital audio formats.

Flexibility

The quality/ability/extent of being flexible.

Flexibility is a system quality attribute that allows a system to adapt and respond to changing requirements, conditions or user needs over time. It ensures that the system can be easily modified, extended or replaced without disrupting the functioning of the system or causing any significant downtime.

Flexibility is also considered as a non-functional requirement as it does not relate to the specific functionality of the system but rather to the way the system is designed to operate. It is often included as part of the system's overall quality requirements and is typically evaluated in terms of the extent to which the system can be easily modified or integrated with other systems or components.

In addition, flexibility is also a cross-functional constraint as it impacts multiple aspects of the system such as its architecture, design, development, testing, maintenance and usability. It requires coordination and collaboration among multiple stakeholders including developers, users, designers, testers, and project managers. Ultimately, flexibility is critical to ensuring that a system remains useful over time and meets evolving business requirements and user needs.

Define flexible: In the context of computers and software, "flexible" refers to the ability of a system or program to adapt and adjust to different requirements or changes easily, without significant modifications or disruptions. Flexible systems or software are designed to be easily customizable, configurable or personalized to specific needs, while maintaining their core functionality and performance. This can apply to various aspects such as user interface, data management, processing rules, and integrations with other systems or devices. Flexibility is a desirable characteristic in many applications, especially those that require frequent updates, customization, or scalability.

Heterogeneity

The quality/ability/extent of being heterogeneous; contrast homogeneity.

Heterogeneity is a system quality attribute, a non-functional requirement, and a cross-functional constraint that refers to the ability of a system to interact with different types of components, systems, or platforms.

Heterogeneity is a system quality attribute, and refers to the ability of a system to support the integration and interoperability of diverse components, systems, or platforms. This means that the system should be able to communicate with other systems or components, regardless of their make or model, and exchange data and information seamlessly.

Heterogeneity is a non-functional requirement, and specifies the need for a system to operate in a diverse and complex environment. It requires the system to be flexible, adaptable, and scalable, enabling it to work effectively in different contexts, scales, and configurations.

Heterogeneity is a cross-functional constraint, and defines the need for different functional groups or teams to collaborate and work together to achieve system interoperability and integration. It requires communication, coordination, and alignment across different domains and disciplines, such as software engineering, network engineering, security, and performance engineering. Overall, heterogeneity is a central attribute of modern systems that operate in complex, distributed, and dynamic environments, such as the Internet of Things, cloud computing, and software ecosystems.

Define heterogeneous: Heterogeneous refers to a mixture or combination of different types, models, or brands of computers and software. It implies that the computing environment is not uniform, and different systems with diverse platforms, operating systems, architectures, or configurations are interconnected and used together for a common purpose, such as data processing, communication, or

collaboration. Heterogeneous computing requires specialized tools and techniques to manage the complexity and ensure compatibility and interoperability among the different components.

Homogeneity

The quality/ability/extent of being homogenous; contrast heterogeneity.

Homogeneity is a system quality attribute that refers to the consistency and uniformity of the system components and behavior.

Homogeneity is a non-functional requirement that ensures that the system functions consistently and reliably across different platforms, environments, and use cases.

Homogeneity is also a cross-functional constraint as it affects various aspects of the system, including performance, usability, scalability, and security.

A homogeneous system can provide the following characteristics: consistency, uniformity, stability. A homogeneous system may simplify system maintenance, and reduces the risk of errors and failures, because of the sameness of compononets, or subsytems, or technologies.

Define homogenous: Homogeneous in the context of computers and software refers to a system or environment where all components or devices use the same type or version of software and have similar technical specifications. In other words, each component in a homogenous system is identical or very similar to the others, allowing for easy communication and compatibility between them. This ensures a streamlined and efficient performance of the system as a whole.

Horizontal scalability

The quality/ability/extent of being scalable across more load providers, such as by adding more servers and clusters; contrast vertical scalability.

Horizontal scalability refers to the ability of a system to scale out by adding more instances or nodes to distribute the workload, rather than scaling up by increasing hardware capacity on a single node. It is a system quality attribute that ensures that the system can handle a growing number of users, requests or data size without any significant reduction in performance.

Horizontal scalability is a non-functional requirement that focuses on performance, availability and reliability. A system that can horizontally scale can easily handle high volumes of traffic, data or requests without any downtime or performance degradation. It is a critical requirement for modern web and mobile applications that demand high availability and real-time responsiveness.

Horizontal scalability is also a cross-functional constraint, as it impacts many aspects of a system, including architecture, design, development, deployment, testing, and monitoring. It requires a distributed architecture that can scale horizontally with more instances, load balancing to distribute traffic among instances, data partitioning to manage large datasets across instances, and fault tolerance to deal with instances failures.

In summary, horizontal scalability is a vital quality attribute that ensures a system can handle increasing volumes of traffic and data without losing performance. It is a non-functional requirement that impacts many aspects of a system and is a critical cross-functional constraint for modern web and mobile applications.

Define horizontal scaling: Horizontal scaling is the process of adding more computing resources such as servers, storage devices, or network bandwidth to a system to increase its overall processing capacity. It is a

scalability strategy that allows an application or system to handle more workloads and traffic by distributing the workload across multiple servers or nodes, rather than relying on a single larger server. Horizontal scaling is an important feature of modern cloud-based infrastructure, as it allows businesses to easily scale their resources up and down to meet changing demands without disrupting the end-users' experience.

Inspectability

The quality/ability/extent of being inspectible.

Inspectability is a system quality attribute that refers to the ability of a system to be analyzed and evaluated for its faults, defects, and errors on multiple levels by different stakeholders including developers, testers, and end-users. It involves the ability to review and modify the software code and architecture design, as well as the ability to trace and debug errors and performance issues in the system.

Inspectability is considered a non-functional requirement, as it doesn't relate to the system functionality itself, but to how the system is built and designed to be examined and analyzed. It is also a cross-functional constraint since it affects multiple aspects of the system development process and the stakeholders involved.

The inspectability attribute is essential for ensuring that the system remains maintainable, scalable, and adaptable over time. By enabling inspectors to identify and fix issues early on, inspectability can help to reduce the total cost of ownership of the system and improve its overall quality and reliability.

Define inspectible: Inspectible refers to the ability to examine or scrutinize the codes, settings, or properties of the computer hardware or software in question. It means that the system is open and transparent for analysis, debugging, or troubleshooting. The ability to inspect a hardware or software program helps ensure that it's functioning in accordance with specifications or standards and that it does not contain any security loopholes or vulnerabilities that may compromise its performance or user data.

Installability

The quality/ability/extent of being installable.

Installability is a system quality attribute that refers to the ease and simplicity with which a software application can be installed, configured, and deployed on a specific platform or device. It is a non-functional requirement that ensures that an application can be installed and configured quickly and easily, without requiring extensive technical knowledge or expertise.

Installability is a cross-functional constraint that involves multiple stakeholders, including developers, IT operations teams, and end-users. It encompasses factors such as software packaging, system dependencies, installation scripts, user interfaces, and documentation.

In order to achieve high installability, software developers need to prioritize aspects like software modularity, compatibility, and simplicity, which help to streamline the installation and configuration process. Additionally, they need to provide clear and concise installation instructions and user documentation, which help users to get up and running with the software quickly and effectively.

In summary, installability is a critical quality attribute that has a significant impact on the user experience and overall usability of a software application. It is an essential consideration for developers, IT operations teams, and end-users alike, and should be taken into account throughout the entire software development lifecycle.

Define installable: Installable refers to software or applications that can be installed on a computer or other electronic device. This typically involves downloading or copying software files onto the device, and then running an installation program or wizard to set up the software and configure it for use. Installable software can include programs for productivity, games, utilities, or many other types of computer applications. Some installable software may require specific operating systems or hardware specifications in order to function properly.

Instrumentability

The quality/ability/extent of being instrumentable.

Instrumentability is a system quality attribute that refers to the ease with which a system can be monitored and diagnosed for errors or malfunctions. It is a non-functional requirement that specifies how well a system can be managed, maintained, and troubleshooted by administrators and end-users. It is also a cross-functional constraint that affects several aspects of a system's design and implementation, including its architecture, interfaces, protocols, and data formats.

Instrumentability is important for systems that are critical to business operations, security, compliance, or performance. The ability to instrument a system means that it can be monitored in real-time, logged for historical analysis, and audited for compliance with regulations, standards, and policies. Good instrumentability means that administrators and users can quickly identify problematic areas, diagnose root causes, and perform remedial actions without disrupting the system or causing further issues.

From a design perspective, instrumentability requires that a system is designed with sufficient visibility, controllability, and testability. This may involve integrating instrumentation hooks, diagnostic probes, and logging mechanisms into the system's architecture, as well as providing configurable settings, administrative interfaces, and debugging tools. Instrumentation also requires that the system's performance and behavior are well-defined and predictable, so that meaningful metrics can be collected and analyzed over time.

In summary, instrumentability is a system quality attribute that ensures that a system can be easily monitored, diagnosed, and maintained. It is essential for systems that need to meet certain standards, regulatory requirements or business objectives. It is a non-functional requirement that must be considered during system design and implementation, and it imposes cross-functional constraints on the system's architecture, interfaces, protocols, and data formats.

Define instrumentable: An instrumentable system exposes a set of internal metrics and statistics that can be collected, analyzed and visualized by various tools. This allows developers and operators to identify issues or bottlenecks, make informed decisions, and optimize the system's behavior. Instrumentation is a crucial aspect of modern monitoring and observability solutions.

Integrity

The quality/ability/extent of having integrity.

Integrity is a system quality attribute, a non-functional requirement, and a cross-functional constraint that refers to the accuracy, consistency, and reliability of data or information in a system. It ensures that the data in the system is complete, accurate, and secure throughout its lifecycle. This is achieved through measures such as data encryption, access controls, and data validation.

As a system quality attribute, integrity is an essential characteristic that determines the effectiveness and reliability of a system. It should be considered as a key requirement alongside other functional and non-functional aspects of the system. A well-designed and implemented system must have integrity to prevent errors, inaccuracies, and inconsistencies in the data.

As a non-functional requirement, integrity specifies a system's performance, security, and usability aspects that cannot be directly measured or observed. It is typically derived from business needs and regulations that require the system to maintain the accuracy, trustworthiness, and confidentiality of data. Examples of non-functional requirements that align with integrity include data validation processes, data encryption mechanisms, system log and audit trails, and access controls and permissions.

As a cross-functional constraint, integrity is a requirement that cuts across multiple functions and stakeholders in a system. It requires collaboration between developers, testers, security professionals, and other stakeholders to ensure that the system meets the required level of integrity. The implementation of integrity controls often requires close coordination and communication between different teams to minimize the risk of data breaches, system failures, or data loss.

Define integrity: Integrity in computers and software refers to the accuracy, consistency, and reliability of data and system operations. It

involves protecting data and systems from unauthorized modification, deletion, or corruption. Maintaining the integrity of computers and software ensures that they function as intended and produce dependable results. It is crucial in areas such as data security, financial transactions, and legal documentation, where inaccurate or unreliable information can have serious consequences.

Interchangeability

The quality/ability/extent of being interchangeable.

Interchangeability is a system quality attribute that refers to the ability of a component or system to be easily changed or replaced without affecting the functioning of the whole system. It allows for flexibility and ease of maintenance, as components can be swapped out if they become unusable or are no longer fit for purpose.

Interchangeability is also considered a non-functional requirement, as it is not directly related to the system's primary function but rather its performance and maintainability. It is often included in system requirements documents as a necessary feature for a high-quality system.

Additionally, interchangeability may be considered a cross-functional constraint, as it can impact multiple aspects of system development and operation. For example, if a component is not interchangeable, it may require more labor and resources to maintain or replace, which could affect project timelines, budget, and overall usability.

Interoperability

The quality/ability/extent of being interoperable.

Interoperability refers to the ability of different systems or components to work together effectively, without imposing any restrictions or limitations on each other.

Interoperability is a system quality attribute because it relates to the overall ability of a system to function properly and deliver the desired outcomes.

Interoperability is a non-functional requirement because it specifies how different system components should interact with each other in terms of performance, reliability, and security.

Interoperability is a cross-functional constraint because it affects multiple aspects of system design, development, and operation, including hardware and software compatibility, network connectivity, data exchange protocols, and user interface standards. Achieving high levels of interoperability requires careful planning, testing, and coordination, as well as the use of open standards and established best practices.

Define interoperable: Interoperable refers to the ability of computers and software systems to communicate and work seamlessly with each other, regardless of the operating system or programming language used. Interoperability enables different systems to share data and access each other's resources, enabling efficient data transfer and collaboration between different applications or platforms. This is crucial for ensuring that different technologies can work together seamlessly and can be used interchangeably to achieve a common goal or objective.

Isolateability

The quality/ability/extent of being isolatable.

Isolateability is a system quality attribute, which refers to the degree to which a system's components can be tested or modified in isolation, without impacting the rest of the system. In other words, it is the ability of a system component to function independently of other components.

Isolateability is also a non-functional requirement, which specifies the conditions that a system must meet regarding its performance, security, reliability, and other qualities that cannot be directly observed by users.

In addition, isolateability can also be considered as a cross-functional constraint, as it affects multiple aspects of the system design, including testing, maintenance, and scalability. It is essential to design systems with high isolateability to reduce the risk of system failures and improve maintainability.

Define isolatable: Isolatable refers to the ability to separate or isolate a particular component, system, or application from the rest of the computer or software environment in a way that it can be tested, modified, or debugged independently. This separation allows developers to identify and fix issues without affecting other parts of the system, making it easier to maintain and update the software. Isolatable components are often designed to function seamlessly with other parts of the system, but can also be effectively isolated when necessary.

Learnability

The quality/ability/extent of being learnable.

Learnability is a system quality attribute that measures the ease with which users can learn to use a system. It is a non-functional requirement that is related to the usability of a system. A system that is designed to be easy to learn should provide users with a clear and concise interface that is intuitive and easy to navigate.

Learnability is also a cross-functional constraint because it affects the way users interact with a system across different domains. For example, if a user is familiar with one software application, they will expect similar interaction patterns and design principles in another software application. If a system is difficult to learn, it can negatively impact user adoption and satisfaction, as well as productivity and efficiency.

In summary, learnability is an important system quality attribute that ensures that users can quickly and easily use a system without extensive training or support. It is a non-functional requirement that should be considered during the design and development of a system, and it is a cross-functional constraint that extends to all areas of the system.

Define learnable: From a computer and software standpoint, learnable refers to the ability of the system to adapt and improve its performance based on real-time feedback and data. A learnable computer system or software is one that can constantly refine itself by learning from user behavior, past actions, and outcomes to optimize its functionality and accuracy. This can be accomplished through machine learning algorithms or other artificial intelligence technologies that enable computers and software to "teach themselves" based on the data available. By being learnable, computers and software can become more efficient, effective, and personalized for individual users, leading to better user experiences and outcomes.

Maintainability

The quality/ability/extent of being maintainable.

Maintainability is a system quality attribute that refers to the ease with which a software application can be modified, updated, and extended by developers, without requiring excessive effort, time, and costs. It is a non-functional requirement that defines the ability of a system to be easily maintained, repaired, or upgraded without compromising its performance, functionality, or reliability.

Maintainability is a cross-functional constraint that affects all aspects of the software development process, including design, development, testing, and deployment. It impacts the cost and time required for software maintenance, as well as the productivity, expertise, and skills of the development team.

Some of the key factors that influence maintainability include the use of modular and well-structured code, adherence to coding standards and best practices, documentation, version control, testing, and debugging. A highly maintainable system provides benefits such as reduced costs, improved reliability and availability, increased user satisfaction, and enhanced agility and flexibility.

Define maintainable: Maintainable refers to the ability of a computer or software system to be easily maintained, repaired, or modified over time. A maintainable system is designed to minimize downtime, reduce the risk of errors or bugs, and allow for upgrades or changes to be implemented with minimal disruption. It typically involves using best practices for programming, testing, and documentation, as well as ensuring that the system is scalable and adaptable to future needs. Ultimately, a maintainable system will save time and resources, improve reliability and efficiency, and help to ensure that the system continues to meet the needs of its users over the long term.

Manageability

The quality/ability/extent of being manageable.

Manageability is a system quality attribute that refers to the ease and efficiency of managing and administering a system. It is a non-functional requirement that defines the ability of a system to be monitored, configured, and maintained by system administrators or operators.

The manageability attribute is a cross-functional constraint that affects various aspects of a system, including its security, performance, reliability, and scalability. For example, a highly manageable system would have easy-to-use management interfaces, automatic monitoring and alert systems, and self-healing capabilities, which would help to ensure that the system is always available and responsive to user needs.

In summary, manageability is an important attribute that is critical for ensuring the smooth functioning and reliable operation of a system. It is a non-functional requirement that should be considered during the design and development of a system and is a cross-functional constraint that affects various aspects of the system.

Define manageable: The term manageable typically refers to the ability of computers or software to be easily controlled, monitored, or modified by a user or administrator. This includes the ability to perform tasks such as software updates, configuration changes, and remote administration without requiring extensive technical knowledge or expertise. In general, manageable products are designed to simplify the management and maintenance of complex systems, making them easier to use and maintain over time.

Mobility

The quality/ability/extent of being mobile.

Mobility is a system quality attribute, non-functional requirement, and cross-functional constraint that refers to the ability of a system to operate or be used in different locations and by different users with different devices. It is a key feature of many modern software systems, including mobile apps, web applications, and cloud-based services.

As a system quality attribute, mobility refers to the ability of a system to provide a seamless user experience across multiple devices and platforms, including smartphones, tablets, laptops, and desktop computers. This requires the system to be designed and developed with a focus on responsiveness, adaptability, and compatibility with different screen sizes and resolutions.

As a non-functional requirement, mobility specifies the performance and availability of a system when accessed from different locations and devices. This includes factors such as network connectivity, data transfer speeds, and device compatibility. To meet the mobility requirement, a system must be designed and developed to provide reliable and fast access to data and functionality regardless of the location and device of the user.

Finally, mobility is also a cross-functional constraint that affects all aspects of the development and deployment of a system. This includes factors such as security, usability, and maintainability, which must be designed and implemented with mobility in mind. For example, security measures must be implemented that allow users to access the system securely from different devices, while usability enhancements must be designed to ensure that users can easily navigate and interact with the system regardless of the device they are using.

Define mobile: Mobile refers to computers, devices, or software that can be easily transported and used while on the move or away from a stationary desk or location. Mobile computers can include laptops,

tablets, smartphones, and other portable devices that enable users to work or access information no matter their location. Mobile software refers to applications or programs that can be used on these portable devices, providing users with the same functionality and features as traditional desktop software, but designed specifically for use on a mobile device.

Modifiability

The quality/ability/extent of being modifiable.

Modifiability is a system quality attribute that refers to the ease with which a software system can be modified or adapted to meet changing user needs or requirements.

As a non-functional requirement, modifiability specifies the system's ability to be modified with minimal impact on its other functionalities, performance, and reliability. It is important for systems to be modifiable as business needs, market conditions, or technology evolves over time.

Modifiability is a cross-functional constraint because it impacts the entire software development process. Modifiability requires software engineers to focus on creating design and architecture that are flexible, scalable, and maintainable. It is important to maintain code consistency, to ensure that new features or changes can be added without breaking the existing code. Additionally, modifiability requires attention to documentation, version control, and testing to ensure that changes can be tracked, tested, and merged effectively.

In summary, modifiability is a system quality attribute, a non-functional requirement, and a cross-functional constraint that requires software engineers to create software architecture and design that can evolve over time to meet changing requirements.

Define modifiable: Modifiable in the context of computers and software refers to the ability to make changes or modifications to the code or settings of an application or system. This can be done by a user, developer or administrator to customize or optimize the performance, improve functionality, or add new features to the software. Modifiability is a desirable quality of software, as it allows for flexibility and adaptability to changing user needs and business requirements.

Modularity

The quality/ability/extent of being modular.

Modularity is a system quality attribute that refers to the degree to which a system is composed of separate, independent components or modules that can be easily interconnected, modified, and replaced without affecting the rest of the system. Modularity allows for flexibility and ease of maintenance, as well as scalability and interoperability with other systems.

Modularity is also a non-functional requirement, as it is not directly related to the primary functionality of the system but rather how that functionality is achieved. It is usually specified as a requirement in system design and architecture documents, and can be measured through metrics such as coupling and cohesion.

Finally, modularity can also be considered a cross-functional constraint, as it impacts multiple aspects of the system, including development, testing, maintenance, and integration. It requires coordination and collaboration between different teams and stakeholders involved in building and using the system, and can affect the overall cost, schedule, and quality of the system.

Define modular: Modular refers to a design or structure that is composed of separate parts or modules that can be easily interconnected or separated. In the context of computers and software, a modular system refers to a system where the overall functionality is achieved by combining smaller, independent components called modules. These modules can be combined or replaced with other compatible modules to easily modify or upgrade the system without affecting other parts of the system. This approach to design allows for greater flexibility, scalability, and customization of the system.

Monitorability

The quality/ability/extent of being monitorable.

Monitorability is a system quality attribute that ensures that the system can be monitored and measured to ensure that it is functioning as expected. It is a non-functional requirement that specifies the system's ability to provide feedback, diagnostics, and performance metrics that can be used to identify issues, troubleshoot problems, and optimize its performance.

In addition to being a system quality attribute and non-functional requirement, monitorability is also a cross-functional constraint. It requires collaboration between different teams and departments, including development, operations, and support, to ensure that the necessary monitoring and measurement tools are available and integrated into the system.

The monitorability of a system is critical for maintaining its reliability, availability, and performance. It enables stakeholders to detect and diagnose issues quickly, reduce downtime, and ultimately provide better service to end-users. Therefore, it is essential to include monitorability as a key consideration in the design and development of a system.

Define monitorable: Monitorable refers to the ability of a computer system or software to be easily observed, tracked, and analyzed for changes or issues. This can include monitoring system performance, network traffic, user activity, and other relevant metrics to ensure the system is operating efficiently and effectively. Essentially, a monitorable system is one that allows for easy monitoring and analysis of its various components and activities.

Observability

The quality/ability/extent of being observable.

Observability is a system quality attribute that refers to the ability of the system to be easily monitored and measured, allowing for timely detection and resolution of issues. It is a non-functional requirement that ensures the system's readiness for debugging, maintenance, and troubleshooting.

Observability also serves as a cross-functional constraint, as it impacts teams responsible for different aspects of the system, such as development, operations, and customer support. It requires collaboration among these teams to ensure that the necessary logs, metrics, and other monitoring tools are in place and accessible to all stakeholders.

In summary, observability is crucial for maintaining a high level of system availability, reliability, and performance. It helps in identifying and resolving issues quickly and efficiently, thereby minimizing downtime and ensuring a positive user experience.

Define observable: Observable in the context of computers and software refers to a property or state of a piece of code or an application that can be easily monitored or measured through some sort of instrumentation or tooling. This can include things like performance metrics, error logs, and other output that can be easily tracked and analyzed by developers and system administrators. The ability to observe the behavior of a system or application is critical for identifying and resolving any issues or inefficiencies that arise, and for optimizing overall system performance.

Operability

The quality/ability/extent of being operable.

Operability is a system quality attribute, which refers to the ease with which a system can be operated, controlled, and monitored. It is a non-functional requirement, which focuses on how well the system works rather than what it does.

Operability relates to the cross-functional constraints that affect the system design and implementation. It is affected by various factors such as human factors, system reliability, system availability, system maintainability, and system safety.

Operability is critical for software systems that require frequent monitoring, configuration, and maintenance. An operable system can minimize the time it takes to perform operations, troubleshoot problems, and recover from failures. It can enhance the usability of the system, reduce errors, and improve user satisfaction.

In summary, operability is an essential quality attribute that ensures that the system is easy to operate, monitor, maintain, and support. It is a non-functional requirement that should be considered in the overall system design and implementation.

Define operable: Operable in the context of computers and software means the ability of the system or program to function properly and as intended. It refers to the capability of the hardware or software to perform its intended tasks without experiencing any errors or issues. An operable system or program is one that works efficiently and offers the expected functionality to the user.

Orthogonality

The quality/ability/extent of being orthogonal.

Orthogonality is a system quality attribute that refers to the degree to which different components or subsystems of a system can be developed, tested, or modified independently without affecting each other. It is a key aspect of system design that aims to minimize the interdependence of system components, allowing for more flexible and efficient development and maintenance.

From a software perspective, orthogonality is typically associated with the concept of modularity, where different modules or components of a system are designed to be loosely coupled and independently testable. This enables developers to work on different parts of the system in parallel, without needing to coordinate or integrate their changes as tightly.

As a non-functional requirement, orthogonality is often essential for achieving other key quality attributes, such as scalability, maintainability, and extensibility. By reducing the coupling between different parts of the system, it becomes easier to scale the system horizontally, add new features or functionality, and maintain the codebase over time.

As a cross-functional constraint, orthogonality can impact many different aspects of system design, including architecture, interfaces, and testing strategies. It requires careful consideration of system dependencies and interactions, as well as communication and coordination among different stakeholders with different areas of expertise.

Define orthogonal: In computing and software, orthogonal refers to the idea that different features or aspects of a system are independent and do not affect each other. This means that a change in one feature will not affect the behavior or function of another feature. In other words, orthogonal features should not have any dependencies or correlations

with each other. This approach is often used in the design of programming languages, where orthogonal features make the language more modular, easy to use, and maintainable.

Portability

The quality/ability/extent of being portable.

Portability is a system quality attribute, cross-functional constraint, and non-functional requirement, which refers to the ability of software or hardware to be easily transferred or adapted from one environment to another. It can also refer to the ease of moving code from one platform or operating system to another. In other words, it's the degree to which a system can be moved or made to work on different platforms or operating systems with minimal adjustments or modifications, without compromising its functionality or performance.

Portability is a crucial attribute in modern software development, especially in the context of ubiquitous computing and cloud-based services. It enables the deployment of software systems across multiple platforms and environments, making it accessible to a wider audience. Portability also improves the maintainability and scalability of systems as it allows developers to design and implement systems that can easily adapt to changes in the technological landscape.

From a cross-functional perspective, portability is important as it allows different stakeholders to access and use the software or hardware regardless of the platform they are using. It takes into account the different user needs, preferences, and capabilities, and ensures that the system can be easily transferred across different environments without loss of quality or functionality. This makes it possible for organizations to reach a wider range of users and reduce their reliance on specific technologies or platforms.

In summary, portability is a critical system quality attribute, cross-functional constraint, and non-functional requirement that ensures the ease of transferring software or hardware across different environments, platforms, and operating systems. It plays a significant role in enabling the deployment of software systems across multiple environments, improving their scalability and maintainability, and making them accessible to a wider audience.

Define portable: Portable refers to a device or software program that can be easily carried or transported. In the context of computers, a portable computer is a laptop, notebook, or tablet that can be easily moved from one location to another. Portable software refers to programs that can be installed on a computer or a portable device such as a USB drive and can be run without having to install them on the host computer.

Precision

The quality/ability/extent of being precise. Compare accuracy.

Precision is a system quality attribute that refers to the level of accuracy and exactness in data, calculations, and outcomes of a system. It is a non-functional requirement that specifies the level of accuracy needed to meet the user's requirements and expectations. Precision is also a cross-functional constraint, as it affects the performance of the system across various business functions, such as finance, engineering, and healthcare.

In a financial system, precision is critical to ensure accurate calculations of amounts, interest rates, and fees. In an engineering system, precision is necessary to ensure accurate measurements, designs, and simulations. In a healthcare system, precision is critical to ensure accurate diagnoses, treatments, and medication dosages.

Precision can be measured using various metrics, such as error rate, variance, and standard deviation. To achieve high precision, a system should have reliable data sources, accurate algorithms and calculations, and robust testing processes.

Predictability

The quality/ability/extent of being predictable.

Predictability is a system quality attribute, which refers to the ability of a system to produce consistent and expected results under varying conditions. It is a non-functional requirement that specifies the system's behavior in response to different inputs and environmental factors.

Predictability is also a cross-functional constraint as it has an impact on various aspects of the system, such as usability, reliability, and performance. A predictable system can enhance the user experience, increase confidence in the system's reliability, reduce the risk of errors, and facilitate maintenance and troubleshooting.

For example, in a critical healthcare system, predictability is essential to ensure accurate and timely delivery of patient information and decision-making support. In a financial system, predictability is crucial to maintain transaction integrity and prevent fraud.

Overall, predictability is an important system quality attribute that should be considered during the design and development of any system, especially in mission-critical and high-security environments.

Define ?: In the context of computers and software, "define" typically refers to the act of explaining, clarifying or describing the meaning and purpose of a term, feature, function or other aspect of a digital system or program. It may involve providing a precise definition or definition of key technical concepts or terms, outlining the steps involved in a particular command or process, or elucidating the intended use or benefits of a software tool or feature for users. This can often involve referencing documentation or manuals, consulting online resources or engaging in communication with other professionals in the field.

Process capability

The quality/ability/extent of being process capable.

Process capability refers to the ability of a system or process to consistently produce products or services that meet predetermined specifications and quality standards. It is a system quality attribute, meaning that it reflects the overall effectiveness and efficiency of the system as a whole.

Process capability is also considered a non-functional requirement, as it is not directly related to the functional requirements of the system (i.e. the specific features and capabilities that it must perform), but is instead focused on the quality and performance of those features.

Additionally, process capability can be considered a cross-functional constraint, as it generally involves the coordination and integration of multiple functions within an organization, from design and manufacturing to quality control and customer service. It requires a collaborative approach to ensure that all aspects of the system are working together to achieve consistent quality and performance.

Define process: Process capable refers to the ability of a computer or software to handle and execute a specific task or set of tasks efficiently and accurately. A process capable system can perform its function without errors or delays, even under high workload or demanding conditions. The capability of a system is determined by factors such as its processing power, memory, and software design. A process capable system is essential for carrying out complex operations such as data analytics, modeling and simulation, and advanced graphics processing.

Producibility

The quality/ability/extent of being producable.

Producibility is a system quality attribute which refers to the ease and efficiency with which a software system can be developed, tested, deployed and maintained. It is primarily concerned with ensuring that the software system can be feasibly and effectively produced.

As a non-functional requirement, producibility is often included as part of the requirements specification for software development. It ensures that the software system can be developed and maintained within available resources, time, and budget. Additionally, it is important to understand how producibility affects the quality of the system as a whole.

Producibility is also a cross-functional constraint because it affects all aspects of the software development lifecycle from the better design concept phase to the final delivery phase. Professionals from different departments, such as developers, testers, deployment engineers, and maintenance personnel must all be able to work together to ensure that the software system is not only robust and efficient but also commercially viable.

Therefore, producibility is a critical aspect of any software development project which ensures that software resources are optimized and are cost-effective.

Define producable: Producible refers to the ability of a computer or software program to be manufactured or developed within relevant technological constraints, in a cost-effective manner, and within a defined timeframe. It involves ensuring that the technology and resources required to create the product are available, and that the design and manufacturing processes are optimized to achieve efficient production. In the context of software development, producibility also includes considerations such as coding standards, testing procedures, and documentation practices to ensure that the final product is

functional, reliable, and maintainable.

Provability

The quality/ability/extent of being provable.

Provability is a system quality attribute that refers to the ability of a system to provide evidence or proof that certain properties or behaviors hold or do not hold within the system. This attribute is closely related to other quality attributes such as correctness, reliability, and security.

As a non-functional requirement, provability specifies that the system must be able to demonstrate its correctness or accuracy in terms of its functionality, data processing, or algorithmic behavior. This requirement often involves the use of formal verification techniques, testing methodologies, or statistical analysis to provide evidence of the system's correctness.

As a cross-functional constraint, provability affects multiple aspects of the system development process, from design to implementation to testing and maintenance. It requires collaboration between different stakeholders, such as developers, testers, and auditors, to establish and verify the system's properties and assumptions.

Overall, provability is an important quality attribute that ensures the reliability and trustworthiness of a system. It helps to ensure that the system behaves as expected under different conditions and provides a basis for making decisions about system design, implementation, and testing.

Define provable: Provable in the context of computers and software refers to being able to demonstrate or verify that a program or system will perform as intended under specified conditions, using mathematical principles and formal methods of analysis. This involves using rigorous mathematical techniques to establish the correctness and reliability of software systems, ensuring that they meet specific functional requirements and behave predictably under all circumstances. Provable software is critical for safety-critical systems, such as aerospace, automotive, medical, and military applications, as

well as for ensuring data privacy and security in various domains.

Recoverability

The quality/ability/extent of being recoverable.

Recoverability is a system quality attribute that refers to a system's ability to recover from failures and errors without losing the data or causing disruptions in other parts of the system. It is a non-functional requirement that ensures the system can recover from errors, failures, or crashes promptly, safely, and efficiently.

Recoverability is a crucial and challenging characteristic for any system because failures and errors are not uncommon. The system needs to guard against these failures and errors and recover as soon as possible to ensure users can continue working with minimal downtime. Recoverability sometimes includes features like automated backups, redundancy, and more.

Recoverability is also a cross-functional constraint that influences other system qualities like reliability, maintainability, and availability. It is a critical attribute that must be addressed throughout a system's lifecycle, including design, development, testing, deployment, and maintenance.

Therefore, recoverability is a term that traverses through different domains of the software and system architecture and is essential in keeping up with the user requirements, business goals, and technical operations of the system.

Define recoverable: Recoverable refers to the ability of computers and software to retrieve lost data or functionality. It means that in case of system failure or malfunction, the data or the system can be restored to its previously functioning state. Data recovery tools are used to recover lost files or information, and system restore points are used to recover a functioning operating system after a crash or critical error. Backup systems and redundancy can also help ensure recoverability in case of data loss or system failure.

Refactorability

The quality/ability/extent of being refactorable.

Refactorability is a system quality attribute that refers to the ease with which a software system can be modified or improved without affecting its functionality. It is a non-functional requirement that is often a cross-functional constraint, affecting the development, maintenance, and evolution of the software system.

Refactorability depends on the design, architecture, and coding practices of the software system. A well-designed, modular, and structured system is easier to refactor than a monolithic or poorly structured system. Likewise, a system with clean, readable, and maintainable code is easier to refactor than a system with messy or spaghetti code.

Refactoring is an essential activity in software development that helps to improve code quality, maintainability, and scalability of a system. Refactorability, therefore, plays a crucial role in the overall success of a software system by enabling developers to make changes to the system easily and quickly without introducing bugs or breaking the system.

Define refactorable: Refactorable refers to the ability of a computer program or software to be easily modified or optimized without changing its behavior or functionality. Refactoring involves restructuring, simplifying, or improving the code without altering the end result. A refactorable codebase allows developers to make changes efficiently and maintain it well over time.

Relevancy

The quality/ability/extent of being relevant.

Relevancy, in the context of software development, refers to the degree to which a system meets the user's needs, expectations, and goals. Relevancy is essential in developing high-quality software that fulfills the intended purpose and satisfies the stakeholders. Depending on the perspective, relevancy can be seen as a system quality attribute, a non-functional requirement, or a cross-functional constraint.

As a system quality attribute, relevancy measures the degree to which a system is fit for its intended use. It considers whether the system's features and functionalities are relevant to its users, whether its outputs provide value to the business, and whether it supports the organization's goals and strategies. A system that lacks relevancy may be difficult to use, lack user adoption, or fail to meet the desired business outcomes.

As a non-functional requirement, relevancy defines the quality attributes that quantify the system's ability to meet its user needs, such as usability, efficiency, and effectiveness. User feedback and observation may help in determining the necessary non-functional requirements that support relevancy. For example, the system may need to be responsive and fast, error-free, secure and scalable, and allow for seamless integration with other software or hardware tools.

As a cross-functional constraint, relevancy influences how different parts of the system interoperate and affects the overall software architecture. It considers the necessary trade-offs when different stakeholders have different goals or infrastructural constraints such as resource limitations. The design and development team may need to ensure that the system is adaptable to different user scenarios, locations, and usage demands.

In summary, relevancy is a critical concept in software development that ensures that the system meets the user's needs, expectations, and goals. It is a system quality attribute, a non-functional requirement, and a

cross-functional constraint that affects the software development process from design to implementation and maintenance.

Define relevant: Relevant, in the context of computers and software, refers to something that is significant or applicable to a particular situation or purpose. It can refer to data, information, features, or functions that are useful or necessary for achieving a specific task or goal. Relevant information or features are considered valuable because they directly contribute to the effectiveness or efficiency of the task or goal at hand. In order to determine if something is relevant, it is important to consider the context, goals, and requirements of the situation or task.

Reliability

The quality/ability/extent of being reliable.

Reliability in software engineering refers to the ability of a system or component to function consistently and accurately. It is a system quality attribute, non-functional requirement, and cross-functional constraint because it affects various aspects of the system, including performance, availability, safety, maintainability, usability, and customer satisfaction.

Reliability is achieved through various techniques, such as fault tolerance, error detection and correction, redundancy, backup and recovery, monitoring and testing, and adherence to standards and best practices. These measures ensure that the system can withstand expected and unexpected loads, handle input and output correctly, recover from failures, and maintain consistent behavior over time.

Reliability is expressed through metrics, such as mean time between failures (MTBF), mean time to repair (MTTR), availability, uptime, failure rate, and error rate. These indicators help developers and stakeholders to assess the system's performance, identify defects and improvements, and make informed decisions about deployment, maintenance, and optimization.

In conclusion, reliability is a fundamental aspect of system quality, and it affects many aspects of software engineering. It requires a comprehensive approach that integrates technical, managerial, and societal considerations and demands continuous attention and improvement throughout the system's life cycle.

Define reliable: Reliable software and computers are those that consistently perform their intended functions without errors, crashes, or unexpected behavior. They are built with high-quality components and are designed to withstand various stresses and failures while maintaining their functionality. A reliable software or computer system should also be able to recover quickly from any failures or malfunctions to minimize downtime and prevent data loss. Overall, reliability is an

important characteristic of software and computers that ensures their effective and efficient functioning over a long period of time.

Repeatability

The quality/ability/extent of being repeatable.

Repeatability is a system quality attribute, which refers to the ability of a system or process to produce consistent results every time it is executed or carried out. Essentially, repeatability ensures that the system delivers accurate and predictable results, with a relatively low margin of error, regardless of the inputs or conditions present.

As a system quality attribute, repeatability is an important consideration for any system or process that requires a high level of precision, accuracy, and consistency, such as in manufacturing, scientific research, or financial forecasting. In these domains, even slight variations in results can have significant consequences, making repeatability a critical requirement.

Repeatability can be a cross-functional constraint, as it can have implications for multiple areas of a system or process, including design, development, testing, and maintenance. To ensure high repeatability, specific design and development practices may need to be implemented, and rigorous testing and quality control measures may need to be put in place.

Repeatability can be a non-functional requirement, as an important consideration for system performance, reliability, and usability. Ensuring that a system is highly repeatable can help to reduce the risk of errors, increase efficiency, and improve user satisfaction.

Define repeatable: Repeatable in the context of computers and software refers to the ability of a process or operation to be executed multiple times with consistent results. In other words, a repeatable process or operation should produce the same output or behavior every time it is executed under the same conditions. This is important in software development and testing to ensure consistent and reliable results. It also allows for automation of processes, as they can be programmed to run repeatedly without requiring manual intervention.

Reproducibility

The quality/ability/extent of being reproducable.

Reproducibility is a system quality attribute that refers to the ability of a system to produce consistent and accurate results when the same input is provided multiple times. It is a non-functional requirement that is essential for scientific research, particularly in fields that rely on experimental data analysis.

From a cross-functional perspective, reproducibility is considered a constraint that affects various stakeholders involved in the development and implementation of a system. It imposes requirements on the design, implementation, testing, and maintenance of the system, and requires collaboration between different teams to ensure that the system meets the necessary standards for reproducibility.

Reproducibility is important because it enhances the reliability and credibility of scientific research and enables scientists to verify and validate their findings. It also promotes transparency and accountability in research, as it allows others to replicate and evaluate the methodology and results of a study.

Define reproducable: In the context of computers and software, "reproducible" refers to the ability to recreate a specific outcome or result consistently and reliably, typically in the same computing environment, with the same input data, and using the same software settings and configurations. Reproducibility is important for ensuring the accuracy and validity of scientific experiments and computational analyses, as well as for troubleshooting and debugging technical issues. To achieve reproducibility, developers or researchers may document their methods and code, use version control systems, and apply standardization and automation practices.

Resiliency

The quality/ability/extent of being resilient.

Resilience is a system quality attribute that refers to the ability of a system to recover quickly and effectively from disruptions or failures.

It is also considered as a non-functional requirement, as it is not related to the functional behavior of the system but rather its ability to accommodate change and maintain its performance under different conditions.

Resilience is often viewed as a cross-functional constraint because it impacts multiple aspects of a system, including performance, reliability, and security. It requires coordination across different functions and stakeholders to ensure that the system is able to withstand and recover from unexpected events.

In summary, resilience is a critical attribute that ensures the sustainability of a system over time and across different situations. It is essential for any system performing critical functions or dealing with sensitive information.

Define resilient: Resilient refers to the ability of computers and software to recover quickly and effectively from difficulties or malfunctions. A resilient system is able to withstand unexpected events or disruption, maintain its functionality, and continue to deliver high-performance without significant interruption. This may involve features such as redundancy, failover mechanisms, automated recovery processes, and backups to ensure that data and operations can be quickly restored in the event of a failure.

Responsiveness

The quality/ability/extent of being responsive.

Responsiveness is a system quality attribute that refers to how quickly and reliably a system can respond to user requests, events, or changes in the system's environment. It is a non-functional requirement that defines the timeliness and efficiency of a system's responses to various inputs and stimuli.

Responsiveness is critical for many systems, especially those that require real-time processing or that serve high-traffic environments, such as e-commerce sites or online gaming platforms. It can also be important in safety-critical applications, such as aviation control systems or medical equipment, where the speed of a response can impact the safety of people or equipment.

As a cross-functional constraint, responsiveness can impact various aspects of a system, including its architecture, design, and implementation. It may require specific technologies or approaches, such as fast processing algorithms, load-balancing techniques, or distributed system architectures, to ensure timely and reliable responses.

In summary, responsiveness is a critical system quality attribute that defines the speed and efficiency of a system's responses to various inputs and stimuli. It is a non-functional requirement that can impact various aspects of a system's design and implementation and is a cross-functional constraint that affects different parts of the system.

Define responsive: Responsive refers to the ability of a computer or software system to adapt to different screen sizes and operating environments, providing an optimal viewing and user experience across multiple devices such as computers, tablets, and smartphones. A responsive system automatically adjusts its layout and functionality to ensure that the content is easily accessible and visually appealing on screens of all sizes.

Reusability

The quality/ability/extent of being reusable.

Reusability is a system quality attribute as it refers to the extent to which a system or its components can be utilized in other systems or applications without requiring significant modification or reconfiguration. It is also a non-functional requirement as it focuses on performance, maintainability, and scalability aspects. Furthermore, it can be considered a cross-functional constraint as it affects multiple aspects of a system, including its design, implementation, testing, and deployment. In short, reusability is an important attribute that helps reduce development costs, speeds up time-to-market, and enhances the overall quality of software systems.

Define reusable: Reusable in the context of computers and software refers to code, scripts, modules, or applications that can be used again in different contexts or projects without modification, saving time and effort. These reusable components are generally designed to be generic and adaptable to different scenarios, making them a cost-effective and efficient way to build software solutions. In general, reusable code is characterized by its modularity, maintainability, flexibility, and portability.

Robustness

The quality/ability/extent of being robust.

Robustness is a system quality attribute that describes the ability of a system to function reliably and efficiently under varying conditions, especially in the face of unforeseen or exceptional events. It is a non-functional requirement that outlines the system's ability to resist and recover from failure, as well as its ability to handle errors and exceptions.

A robust system should be able to operate under unexpected events such as system overload, network failure, hardware failure, input errors and other unforeseen conditions. It should be able to detect, isolate and isolate errors, and recover from them with minimum impact on the system's performance.

Robustness is a cross-functional constraint as it affects multiple aspects of the system, including the architecture, design, implementation, testing, and maintenance. Robustness requires careful consideration of system interactions, dependencies, scalability and fault tolerance, and it is achieved through a combination of redundancy, monitoring, resilience, and fault-tolerance techniques.

Define robust: Robust in the context of computers and software refers to the ability of a system or application to function with a high level of reliability, stability, and resilience under various stress conditions, such as high traffic volume or unexpected errors. A robust system should be able to handle errors and recover from them without crashing, and it should be able to adapt to changing demands or environments without losing performance or functionality. A robust software also should be well-tested, well-documented, and easy to maintain, with clear and well-defined interfaces and error-handling mechanisms.

Safety

The quality/ability/extent of being safe.

Safety is a system quality attribute, which is an important non-functional requirement and a cross-functional constraint that focuses on ensuring that a system is free from harm or danger to users, operators, the environment, and other stakeholders.

A high level of safety is a critical aspect of a system's functionality, particularly in applications where human life, health, or the environment are at risk. It is essential to consider safety requirements while designing, developing, and testing a system. The goal is to prevent or mitigate any potential hazards, such as physical injury, property damage, or environmental harm.

A system's safety requirements usually include regulatory compliance guidelines, industry standards, best practices, and risk assessments. To meet these requirements, developers must incorporate safety-conscious design practices and implement appropriate safety controls in their systems. These practices may include incorporating redundant systems, implementing fail-safe mechanisms, and providing adequate training and support for users and operators.

Overall, safety is an essential system quality attribute that must be considered throughout the entire system development lifecycle. Failure to meet safety requirements could result in severe consequences, including legal liabilities, damage to the system's reputation, or loss of life or property. Therefore, it is vital to ensure that a system is safe and secure, in addition to its functional capabilities.

Define safe: In the context of computers and software, "safe" generally means that the technology does not do any harm to the user's system, data or privacy. A safe software or application is one that functions properly and reliably, without crashing, corrupting data or opening security vulnerabilities. Safe software is also free from malware or viruses, and does not engage in any malicious activities such as phishing,

spying, or stealing user data. Safe computers refer to systems that are secure from external and internal threats, such as cyber-attacks, viruses, and unauthorized access.

Scalability

The quality/ability/extent of being scalable.

Scalability is a system quality attribute that refers to the ability of a system to handle growing amounts of workload or handle more users, resources, or data without sacrificing performance or crashing. Scalability is a non-functional requirement in software engineering that determines how well a system responds to increasing amounts of data, traffic, or requests.

Scalability is also a cross-functional constraint that impacts various stakeholders including developers, operations, and business teams. From the development standpoint, scalable systems require robust and flexible architectures, optimized coding practices, and efficient utilization of hardware resources. From the operations side, scalable systems demand reliable infrastructures, high availability, and efficient monitoring and maintenance processes. From the business perspective, scalable systems drive revenue, customer satisfaction, and competitive advantage by enabling easy and seamless scaling without disrupting the user experience.

In summary, scalability is a crucial attribute that determines the long-term viability and success of a system, application, or service. It enables businesses to grow and adapt to changing market conditions and user demands while ensuring consistent performance and reliability.

Scalability can be measured over multiple dimensions, such as:

- Administrative scalability: The ability for an increasing number of organizations or users to access a system.
- Functional scalability: The ability to enhance the system by adding new functionality without disrupting existing activities.
- Geographic scalability: The ability to maintain effectiveness during expansion from a local area to a larger region.
- Load scalability: The ability for a distributed system to expand and

contract to accommodate heavier or lighter loads, including, the ease with which a system or component can be modified, added, or removed, to accommodate changing loads.

- Generation scalability: The ability of a system to scale by adopting new generations of components.
- Heterogeneous scalability is the ability to adopt components from different vendors.

Define scalable: Scalable refers to the ability of computer hardware, software or applications to adapt to increased demand or usage, without sacrificing performance or functionality. It means that a system can handle an increase in workload or users in a cost-effective manner without compromising quality or speed. The scalability of a system is usually achieved through the use of efficient hardware and software architectures, as well as the ability to add more resources or capacity as needed.

Schedulability

The quality/ability/extent of being scheduleable.

Schedulability is a system quality attribute, a non-functional requirement, and a cross-functional constraint. It refers to the ability of a system to complete tasks within their specified time constraints. A system that is schedulable can effectively manage its resources, prioritize tasks, and execute them in a timely and efficient manner.

Schedulability is a critical system quality attribute because it impacts user satisfaction and overall system performance. If a system cannot meet its scheduling requirements, it may fail to deliver required functionality, cause delays or missed deadlines, or even crash. Therefore, it is essential for systems to be designed and tested for their schedulability.

Schedulability is a non-functional requirement because it does not pertain to the functionality of the system itself, but rather to its performance characteristics. It is a constraint that must be satisfied in order to ensure the overall usability and effectiveness of the system.

Finally, schedulability is a cross-functional constraint because it affects multiple aspects of the system. It requires collaboration between different stakeholders and teams, including developers, testers, project managers, and users. Schedulability must be considered throughout the entire system development lifecycle, from requirements gathering to deployment and maintenance.

Scriptability

The quality/ability/extent of being scriptable.

Scriptability is a system quality attribute that refers to the ability of a system to be controlled or automated using scripts. Scripting languages enable users twowrite scripts that interact with the scriptable system or application to automate tasks, manipulate data, or perform other operations. Scriptability is a desirable feature in software as it allows for customization and integration with other systems. By providing a scripting interface, software can be easily automated and extended to meet specific user needs.

Scriptability a non-functional requirement that specifies the system's ability to execute scripts or command sequences that can automate repetitive or complex tasks.

Scriptability can also be a cross-functional constraint in that it requires collaboration between different functional areas, such as development and operations, to ensure that the system can be easily automated and maintained. It is also important for ensuring the system's reliability, as automation can reduce the risk of errors and improve consistency.

Seamlessness

The quality/ability/extent of being seamless.

Seamlessness is a system quality attribute that refers to the smooth and coherent functioning of a system, without any visible or noticeable interruptions or transitions. It is a non-functional requirement that ensures the system operates effectively and efficiently, with ease of use and minimal disruption to the user.

Seamlessness is a cross-functional constraint that involves multiple aspects of the system, such as performance, reliability, usability, and security. It encompasses features such as fast response times, high availability, ease of navigation, consistent interfaces, and secure data transfer.

Overall, seamlessness aims to provide a user-friendly and engaging experience to the end-users, with minimal effort and maximum satisfaction. It is critical for systems that deal with mission-critical functions, such as financial transactions, healthcare, or emergency services, as any interruption or delay can have severe consequences.

Define seamless: Seamless in the context of computers and software refers to a smooth and uninterrupted flow or integration of processes, applications, or systems. A seamless system or software doesn't require manual intervention or adjustment to function properly, and it operates seamlessly with other systems, applications, or devices. It eliminates interruptions or delays in task execution and provides a seamless user experience, allowing users to accomplish their goals with minimal effort or disruption.

Securability

The quality/ability/extent of being securable.

Securability is a system quality attribute that refers to the ability of a system to protect against unauthorized access, modification, disclosure, or destruction of information or system resources. It is a critical non-functional requirement that ensures the security of a system and is achieved through the implementation of various security controls, such as authentication, authorization, encryption, and audit trails.

Securability is also a cross-functional constraint as it affects multiple aspects of a system, including design, development, testing, deployment, and maintenance. It requires collaboration between different stakeholders, such as developers, system administrators, security analysts, and auditors, to ensure that the system is secure throughout its entire lifecycle.

Overall, securability is a critical system quality attribute that ensures the protection of a system's assets against security threats and risks. It is essential for systems that handle sensitive and confidential information or provide critical services, such as financial transactions, healthcare, and national security.

Define securable: In the context of computers and software, securable refers to the ability of a system or application to be secured from unauthorized access, tampering or data breaches. This includes implementing security measures such as firewalls, encryption, authentication and authorization mechanisms to protect both the system and the data it contains. A securable system or application is one that is designed and maintained with security in mind, and is resistant to attacks and vulnerabilities.

Self-sustainability

The quality/ability/extent of being self-sustainable. Compare antifragility.

Self-sustainability refers to the ability of a system to maintain its intended functionality without external intervention or support.

Self-sustainability is a system quality attribute that focuses on the system's ability to sustain itself over time, even as its environment and circumstances change. Self-sustainability is a non-functional requirement, which means that it does not relate to the system's specific functionality but rather to how the system operates.

As a cross-functional constraint, self-sustainability impacts multiple parts of a system and requires coordination and cooperation among various stakeholders. For example, a self-sustaining system may require the use of renewable energy sources or recycling of materials to reduce waste and minimize external support. It may involve the use of automation and smart technology to optimize energy efficiency and reduce maintenance needs.

Overall, self-sustainability is a crucial aspect of system design, as it helps to create more efficient and environmentally-friendly systems that can operate indefinitely without requiring significant external support. By prioritizing self-sustainability as a system quality attribute, organizations can reduce their dependence on external resources and create more resilient and reliable systems.

Define self-sustainable: Self-sustainable computers and software are those that can function and operate independently without relying on external resources for power or maintenance. They are designed to be capable of supporting and maintaining their own operational needs, including power management, data storage, and security.
Self-sustainable systems are often used in environments with limited access to resources, such as remote areas or emergency situations, where traditional infrastructure is not available. They are also used in

critical systems where downtime is not an option, such as in space missions or military operations.

Separability

The quality/ability/extent of being separable.

Separability is a system quality attribute that refers to the ability of a system to be divided into modules or components that can be developed, tested, maintained, and replaced independently.

It is also a non-functional requirement that emphasizes the need for a system to have a cohesive architecture that allows for ease of maintenance and modification. Separability enables developers to work on different parts of the system separately, without affecting other components, allowing for better management of code and reducing the complexity of the system.

In addition, separability represents a cross-functional constraint, as it requires collaboration between developers, testers, and other stakeholders. It is an important factor in ensuring that a system is sustainable, adaptable, and scalable. By enabling modular design, separability can also help to reduce costs, reduce risk, and improve quality by allowing for more effective testing and validation of each component.

Define separable: In the context of computers and software, separable refers to the ability to physically or logically disconnect one component or module from another without affecting the functionality of the remaining parts. This means that different parts of a system can be changed or upgraded without disrupting the operation of other components. Separability is an important design principle that allows for flexibility, scalability, and maintainability of computer systems and software applications.

Serviceability

The quality/ability/extent of being servicable.

Serviceability is a system quality attribute that refers to the ease and speed with which a system can be repaired, maintained, and updated. It is a critical non-functional requirement that affects a system's availability, reliability, and performance.

Serviceability is often viewed as a cross-functional constraint because it affects multiple stakeholders, including end-users, system administrators, maintenance personnel, and developers. Serviceability requirements must be considered and addressed throughout the system development life cycle, from design and implementation to testing, deployment, and maintenance.

Some examples of serviceability requirements include:

- Ease of installation and configuration
- Availability of documentation and support resources
- Ability to diagnose and troubleshoot issues quickly and accurately
- Flexibility to upgrade and scale the system without disrupting services
- Compatibility with existing infrastructure and tools
- Robustness to handle changes in the operating environment

In summary, serviceability is a critical system quality attribute that affects the usability, maintainability, and scalability of a system. Addressing serviceability requirements is essential for delivering a reliable and effective system that meets the needs of all stakeholders.

Define servicable: The term "serviceable" in regards to computers and software means that the system or program is able to perform essential functions effectively and reliably. It should be functioning well enough to serve its intended purpose and meet the user's needs without constant glitches or errors. A serviceable computer or software should also be able to receive regular maintenance and updates to ensure its continued functionality. Additionally, it should be compatible with other systems

and software with which it needs to interact.

Simplicity

The quality/ability/extent of being simple.

Simplicity is a system quality attribute, cross-functional constraint, and non-functional requirement that refers to the ease of understanding and using a system. It is the degree to which a system is easy to use, learn, and maintain. The simplicity of a system can be measured by the amount of effort required to accomplish tasks or the number of steps required to complete a task.

As a system quality attribute, simplicity is a desirable characteristic of a system because it allows users to quickly and efficiently accomplish their goals without the need for extensive training or specialized knowledge. A system that is too complex or difficult to use can result in frustrated users, decreased efficiency, and increased errors.

As a cross-functional constraint, simplicity imposes limitations on the design and development of a system. It requires the involvement and input of multiple stakeholders, including users, developers, and designers to ensure that the system is both functional and easy to use.

As a non-functional requirement, simplicity is a design parameter that must be considered when developing a system. It must be defined in clear and measurable terms, and the system must be tested to ensure that it meets the defined requirements. Simplicity is often measured in terms of user satisfaction, efficiency, and effectiveness.

Define simple: Simple in the context of computers and software refers to something that is easy to use, understand, and learn. It is typically designed to be user-friendly and intuitive, with a minimal learning curve. A simple software program or application is often characterized by a clean and uncluttered user interface, streamlined functionality, and straightforward navigation. Similarly, simple hardware components are usually uncomplicated in design and assembly, with minimal components and easy assembly instructions. Overall, simplicity in computing and software refers to a product that is accessible to users of

all levels of technical expertise.

Stability

The quality/ability/extent of being stable.

Stability is a system quality attribute that refers to the ability of a system to maintain a steady and consistent performance level, even under high loads, stresses or unexpected events. In other words, stability ensures that a system has the capacity to resist failures, crashes, and other disruptions that may occur due to a variety of factors, such as insufficient resources, network glitches, or security breaches.

As a non-functional requirement, stability is a critical aspect of a system's overall performance, as it contributes to user satisfaction, operational efficiency, and the overall success of the system. Meeting stability requirements involves ensuring that the system operates reliably and consistently, and that it can recover quickly from failures or disruptions.

As a cross-functional constraint, stability requires the collaboration and cooperation of different functional teams, such as software development, infrastructure, security, and testing. Achieving stability involves taking a holistic approach to system design and development, which considers the various factors that can impact system performance, including resource availability, network topology, software architecture, and user behavior. Overall, stability requires a balance of technical and non-technical factors to ensure that the system performs reliably and efficiently, and meets the needs of its users.

Define stable: In the context of computers and software, stability refers to the ability of a system to function reliably and consistently without crashing or malfunctioning under normal operating conditions. A stable computer or software is one that can effectively manage the resources it uses, handle user input, and respond to external inputs or events without causing unexpected errors or data loss. A stable system is crucial for ensuring optimal performance, user satisfaction, and data security.

Standards compliance

The quality/ability/extent of being standards compliant.

Standards compliance refers to the extent to which a system adheres to established guidelines, regulations, or frameworks that dictate how software should be designed, developed, and deployed. This is important because it ensures that the system meets certain levels of quality, interoperability, security, and reliability, among other characteristics that are critical to successful system performance.

As a system quality attribute, standards compliance is concerned with how well a system meets specific requirements that are imposed by regulatory bodies, industry consortia, or other authoritative organizations. For example, a system that complies with the ISO/IEC 27001 standard for information security management demonstrates its ability to safeguard sensitive data and maintain the confidentiality, integrity, and availability of information across the enterprise.

As a non-functional requirement, standards compliance represents a set of criteria that the system must meet in order to be considered acceptable for deployment. Non-functional requirements are not features or functionalities per se, but rather operational characteristics, such as performance, scalability, maintainability, and usability, that must be satisfied in order to ensure successful system operation.

As a cross-functional constraint, standards compliance reflects the need for multiple system components, modules, or subsystems to work together seamlessly and in conformity with a set of predefined rules or procedures. This may require the use of specific data formats, communication protocols, or other mechanisms that facilitate interoperability and communication between different system elements. For example, a system that complies with the HL7 standard for healthcare data exchange can seamlessly share patient information across different provider organizations, enabling better coordinated and more efficient care.

Define standards: Standards compliance in the context of computers and software refers to the extent to which a product or system adheres to recognized norms or best practices. Compliance with standards can ensure that software and hardware components are interoperable, reliable, and secure. Standards compliance may include adherence to technical specifications, industry-specific regulations or requirements, and compatibility with other systems, platforms, or devices. In addition, compliance with standards may be required to protect against legal liabilities, ensure data privacy, and achieve industry certifications or accreditations.

Supportability

The quality/ability/extent of being supportable.

Supportability is a systems quality attribute, which is a type of non-functional requirement and cross-functional constraint. It refers to the ability of a system to be supported and fully maintained throughout its lifecycle, including installation, maintenance, troubleshooting, and upgrades. Supportability includes features such as providing adequate documentation, easy maintenance procedures, and training support staff to administer and troubleshoot the system. A system that is highly supportable requires less effort and time from support staff, reducing the overall cost of maintaining the system. High supportability is especially important for mission-critical systems where downtime can have severe consequences.

Define supportability: Supportability in computers and software refers to the ability of a product to be supported effectively and efficiently throughout its lifecycle by the manufacturer or vendor. This includes providing technical assistance, updates, patches, and other means to regulate the functioning and repair of the product. The aim is to ensure that the product consistently meets user requirements, performs optimally, and complies with industry standards and regulations. Moreover, supportability also involves the availability and quality of documentation, training, and other resources necessary for end-users to maintain or troubleshoot the product independently. It is an essential aspect of software development and hardware engineering, as it assures consumers of the product's usability, reliability, and long-term viability.

Survivability

The quality/ability/extent of being survivable.

Survivability is a system quality attribute that refers to the ability of a system to maintain its critical functionality in the face of various disturbances or threats. It is a non-functional requirement that defines the capacity of a system to operate continuously, effectively and efficiently in the presence of disruptive changes or disruptions.

The goal of survivability is to ensure that the system remains available, secure, and functional even in the event of partial or complete failures or attacks. It is a critical attribute for systems that are involved in mission-critical or operational-critical applications, where the failure of the system can cause significant damage, loss of life or financial penalties.

Survivability is often considered a cross-functional constraint as it involves the consideration of multiple system aspects such as security, availability, reliability, and performance. It requires the involvement of different domains such as security, operations, networking, and software development to jointly evaluate the system requirement, design and provide an adequate level of survivability.

Define survivable: Survivable means having the ability to remain operational even under adverse conditions, such as hardware failures, cyber attacks, or natural disasters. In the context of computers and software, survivability refers to their ability to maintain critical functions or services despite being compromised or under attack. Survivability requires redundancy, fault-tolerance, and proactive security measures to mitigate risks and ensure continuity of operations.

Sustainability

The quality/ability/extent of being sustainable.

Sustainability is a system quality attribute that refers to a system's ability to maintain its performance and functionality over time, while also minimizing its impact on the environment and society.

Sustainability is a cross-functional constraint, as it involves various aspects of a system, including its technology, design, materials, and business practices, as well as its impact on the environment and society.

Sustainability is a non-functional requirement, as it is a characteristic that is not directly related to the system's primary function or features, but rather its overall quality and impact.

Sustainability is an essential consideration for any system or technology that seeks to minimize its impact on the environment, conserve resources, and promote social responsibility.

Define sustainable: Sustainable computers and software refer to products that have been designed and developed with a focus on minimizing their environmental impact throughout their entire life cycle. This includes the production, usage, and disposal stages. Sustainable computers and software are designed to last longer, use less energy, and be easily recyclable or biodegradable. They also prioritize ethical considerations such as fair labor practices and responsible sourcing of materials. Sustainable computers and software aim to address the negative impact of technology on the environment and human health.

Tailorability

The quality/ability/extent of being tailorable.

Tailorability is a system quality attribute that represents the degree to which a system can be configured or customized to meet the specific needs of its users or stakeholders. It is a non-functional requirement that allows users to mold the system to fit their unique requirements or preferences.

Tailorability is a cross-functional constraint because it affects various aspects of the system, such as its usability, adaptability, and maintainability. For example, a highly tailorability system may require additional user training and documentation to guide users on how to tailor the system effectively.

Tailorability is essential for systems that serve different users or user groups with varying needs and preferences. It enables users to personalize their experience, improve their efficiency and productivity, and reduce the risk of errors or dissatisfaction. Systems with low tailorability may create frustration or inefficiency due to the inability to meet specific user requirements.

Define tailorable: Tailorable in the context of computers and software means the ability to customize or modify the functionality, user interface, or features of a software application according to specific needs or preferences. It enables users or developers to adapt the software to suit their unique requirements, be it personal or business, without the need for extensive programming skills. Tailorable software can be easily configured or modified through user-friendly options, plug-ins, or scripting languages to provide a more streamlined, intuitive, and efficient user experience.

Testability

The quality/ability/extent of being testable.

Testability is a key system quality attribute that refers to the ease with which a software program or system can be tested to ensure that it functions correctly and meets the requirements of the end user. It is a non-functional requirement that is critical to the development of high-quality software, as it enables developers to verify and validate the functionality, performance, and reliability of the software before it is released to end users.

Testability is a cross-functional constraint, as it requires collaboration between different teams within the software development process, including developers, testers, and quality assurance professionals. It also requires the use of specialized testing tools and techniques to ensure that the software is thoroughly and effectively tested.

A system with good testability has a clearly defined and well-documented architecture, clean and modular code, and comprehensive test plans and procedures. It also incorporates features such as debuggers, logging, and monitoring tools to help identify and resolve issues quickly and efficiently.

Overall, testability is an essential attribute of software quality that helps ensure that software is reliable, robust, and meets the needs of end users.

Define testable: In the context of computers and software, testable refers to a characteristic of a program that allows it to be easily and effectively evaluated, measured, and verified through testing methods. A program or software is considered testable if it can be systematically tested and its functionality, accuracy, and performance can be measured and validated through any testing method, including automated testing. Testability is important in software development as it helps to identify defects and issues, improve software quality and reliability, and ensure that the software meets the requirements and specifications of the users.

Timeliness

The quality/ability/extent of being timely.

Timeliness is a system quality attribute that refers to the ability of a system to deliver results or respond to requests within the specified time frame. It is a non-functional requirement that is essential for many systems, particularly those that involve real-time processing or mission-critical applications.

Timeliness is also a cross-functional constraint, as it can impact multiple aspects of a system's functionality, including performance, usability, and reliability. For example, if a trading system is not able to respond to market events in a timely manner, it may result in lost opportunities and financial losses for the users. Similarly, if an e-commerce website takes too long to process customer requests or transactions, it can lead to dissatisfaction, abandoned shopping carts, and loss of business.

To ensure timeliness, systems are often designed with response time objectives and performance metrics that measure how well the system meets those objectives. Various techniques such as load balancing, caching, and optimization are used to improve system performance and reduce response times. Testing and monitoring are also crucial in ensuring that the system meets the timeliness requirements throughout its life cycle.

Define timely: In computers and software, timely refers to the ability of a system or program to execute tasks within a defined period or interval. It means that an action is performed when it is needed or required, and not too early or too late. Timeliness is crucial in real-time systems, where actions must be executed at precise moments, and delays can cause errors or even failures. It is also essential in general software development, where meeting deadlines and optimizing performance are critical factors.

Traceability

The quality/ability/extent of being traceable.

Traceability refers to the ability of a system to trace and track the changes made during its development lifecycle. It is an important system quality attribute as it ensures that the system is developed and maintained according to the specified requirements and design. Traceability also helps in identifying the impact of changes made to the system and reduces the risk of errors or inconsistencies.

Traceability is also considered as a non-functional requirement as it does not directly affect the functionality of the system but is necessary for ensuring the quality of the system. It is used to measure the effectiveness of the system and ensure that it meets the desired level of reliability, maintainability, and efficiency.

Additionally, traceability can also be considered as a cross-functional constraint as it involves the collaboration between different teams and stakeholders involved in the development, deployment, and maintenance of the system. It is important to ensure that all departments are aligned in terms of the system goals, requirements, and design to ensure efficient and effective development and maintenance of the system.

Translatability

The quality/ability/extent of being translatable.

Translatability is a system quality attribute that refers to the ability of a system to be easily translated or localized for different languages, cultures, and regions. To achieve translatability, a system should use standard encoding and character sets, support internationalization and localization features, and provide clear documentation and resources for translators and localizers.

A translatable system can help reach a wider audience and be more accessible to users from different backgrounds. It is especially important for systems that are intended for use in multiple regions or that need to support a diverse user base. By providing support for multiple languages and cultures, a system can reduce the risk of misunderstandings and errors caused by language and cultural differences, and ultimately improve the user experience.

Translatability is a non-functional requirement that specifies the system's ability to support multiple languages, character sets, and cultural conventions.

Translatability can also be a cross-functional constraint, as it requires collaboration between different functional areas, such as development, design, and localization, to ensure that the system can be easily translated and localized. It is also important for ensuring the system's usability, as it can reduce the risk of misunderstandings and errors caused by language and cultural differences.

Transparency

The quality/ability/extent of being transparent.

Transparency is a system quality attribute that refers to the ability of a system to provide visibility and clarity into its operations, internal workings, and decision-making processes. It is often used to promote trust and accountability in systems, particularly those that are used by multiple stakeholders or that involve significant amounts of data or decision-making.

As a non-functional requirement, transparency specifies that a system must be designed and implemented in such a way that it allows users to easily access, understand, and verify its output and decision-making processes. This can include providing detailed documentation, clear user interfaces, and audit trails that allow users to track how data is being processed and how decisions are being made.

As a cross-functional constraint, transparency can impact a range of different aspects of a system's design and implementation, from the choice of algorithms and data structures used to process information, to the level of external oversight and governance that must be put in place to ensure that the system is being used ethically and effectively.

Overall, transparency is an important quality attribute that can help to promote trust and accountability in systems, particularly those that are used in sensitive or critical contexts. Investing in transparency can help to ensure that systems are designed with the needs of users and stakeholders in mind, while also promoting ethical and responsible use of data and technology.

Define transparent: In the context of computers and software, transparent means functioning in such a way that the user is not aware of its operation or inner workings. It suggests that the system operates smoothly and efficiently without causing any disruptions or requiring any special attention from the user. A transparent system is typically intuitive and user-friendly, making it easy for the user to interact with

and access the information they need without any unnecessary complexity or confusion.

Tryability

The quality/ability/extent of being tryable.

Tryability is a system quality attribute that refers to the ease and convenience of testing and evaluating a system or software product. It is related to non-functional requirements but also has cross-functional constraints as it involves different aspects of the product development lifecycle such as design, development, testing, and maintenance.

Tryability is essential because it allows developers, testers, and other stakeholders to evaluate the product's quality and identify any issues or defects in a timely manner. It also helps in reducing the cost and effort required to fix these issues at a later stage in the development process.

Some of the key factors that determine the tryability of a system include the availability of testing tools and resources, the ease of replicating issues, the clarity and completeness of documentation, and the ease of setting up test environments.

Overall, tryability is an important aspect of system quality as it enables better collaboration between different teams, reduces development time and cost, and ensures the delivery of a high-quality, reliable product.

Define tryable: As an AI language model, I cannot find the word "tryable" in any technical or computing context. Please provide more context or a specific definition if you have one.

Ubiquity

The quality/ability/extent of being ubiquitous.

Ubiquity is a system quality attribute that refers to the ability of a system to be available and accessible across different devices, platforms, and locations. It is important for modern systems that need to be accessible anytime and anywhere.

As a non-functional requirement, ubiquity sets the standard for the reliability and availability of a system. It ensures that a system is always reachable and responsive to users' needs. This can be achieved through redundancy, load balancing, and other measures that maintain system availability.

Ubiquity can also be a cross-functional constraint, meaning that it affects multiple aspects of a system's design and development. For example, a system that needs to be available on both desktop and mobile platforms may require different user interfaces and architectures to achieve ubiquity. Thus, cross-disciplinary collaboration and communication are essential in designing and implementing a system with strong ubiquity.

Define ubiquitous: Ubiquitous, in relation to computers and software, means existing or being present everywhere at the same time. It refers to the pervasive nature of technology, where computer devices and software are used across various platforms, operating systems, and networks. Ubiquitous computing involves the integration of technology into everyday life, making it an inseparable part of our daily routines. This includes technologies such as the Internet of Things (IoT), cloud computing, artificial intelligence, and mobile computing, which have become ubiquitous in modern computing.

Understandability

The quality/ability/extent of being understandable.

Understandability is a system quality attribute and a non-functional requirement that defines the ease with which a user can comprehend and use a system. It is the measure of how easy it is for users to understand the system's functions and features.

Understandability is also a cross-functional constraint because it involves different stakeholders, including developers, testers, and users. It requires a collaborative effort across different departments to ensure the system is designed and developed with the user's ease of understanding in mind.

The understandability of a system is crucial to usability and user satisfaction. An understandable system reduces user frustration and errors, and makes it easier for users to achieve their goals and complete tasks efficiently. Therefore, understandability should be a primary consideration during the design, development, and testing of the system.

Define understandable: In the context of computers and software, "understandable" refers to the ability of users to easily comprehend and use the features and functions of a program or system. This includes interfaces that are intuitive and user-friendly, documentation that is clear and concise, error messages that provide meaningful explanations or suggestions for troubleshooting, and overall design that is logical and easy to navigate. The goal of creating understandable software is to minimize user frustration and improve productivity by reducing the learning curve and enabling users to make efficient use of the system.

Upgradability

The quality/ability/extent of being upgradable.

Upgradability is a system quality attribute, which is a measure of how well a software system can be modified or enhanced to meet changing needs. It is a non-functional requirement that describes the ease with which a system can be upgraded or updated in terms of hardware, software, or functionality.

Upgradability is a cross-functional constraint because it affects multiple aspects of a system, including its design, architecture, and implementation. For example, a system's upgradability may be influenced by its modularity, reusability, compatibility, and scalability.

An upgradable system is desirable because it enables organizations to keep up with changing business requirements, technology advancements, and security threats. In addition, a system that is easy to upgrade can reduce the cost and effort associated with system maintenance, testing, and deployment.

Overall, upgradability is a critical system quality attribute that requires careful consideration during the design and implementation of software systems.

Define upgradable: Upgradable refers to the ability of a computer or software to be upgraded or improved with newer or better components or versions. This may involve replacing or adding hardware components such as RAM, hard drives, or graphics cards, or upgrading to a newer version of software. An upgradable system or software is designed with the flexibility to be expanded or improved over time, allowing for increased performance, functionality, or compatibility with new technologies.

Usability

The quality/ability/extent of being usable.

Usability is a system quality attribute, which means that it is a characteristic or property of a system that determines its ability to meet user needs and expectations. Usability refers to how easy and intuitive it is for users to interact with a system and accomplish their tasks, without experiencing frustration or confusion.

Usability is also considered a non-functional requirement, which are requirements that specify how the system should behave, rather than what functions it should perform. Non-functional requirements are important because they influence the overall performance, reliability, and maintainability of the system.

Moreover, usability is a cross-functional constraint, which means that it affects multiple aspects of the system, including the user interface design, system architecture, development methodology, and testing procedures. Usability is a constraint because it sets limits on how the system can be designed and developed to ensure that it meets the needs and expectations of its users.

Define usable: Usable refers to the ease with which a computer or software can be utilized by a user to achieve a desired task. It includes the efficiency, effectiveness, learnability, and overall user satisfaction with the technology. A usable technology is intuitive, well-organized, and requires minimal effort to operate.

Vertical scalability

The quality/ability/extent of being scalable within one provider, such as by adding processors, memory, or storage.

Vertical scalability refers to the ability of a system to handle an increasing workload by adding more resources to a single node or machine, such as adding more CPUs or RAM to a server. It is a system quality attribute, as it defines the ability of the system to function properly under increased workload.

Vertical scalability is also a non-functional requirement, as it is not directly related to the system's functionality, but rather to its performance characteristics. It is a crucial requirement for systems that handle large amounts of data or perform complex computations, as they require more resources to function effectively.

Vertical scalability is also a cross-functional constraint, as it affects multiple areas of the system. It affects the system's performance, capacity, and availability, and requires coordination between the development, operations, and infrastructure teams to implement properly. An efficient and scalable architecture can help ensure that a system can accommodate future growth and provide a reliable user experience.

Define vertical scaling: Vertical scaling refers to the process of increasing the capacity of a single server by adding more resources, such as RAM, CPU, or storage. This differs from horizontal scaling, which involves adding more servers to handle increased demand. Vertical scaling is often used to improve the performance of applications that require high levels of processing power or memory, and is commonly used in database systems or middleware applications. It is also known as scaling up or upgrading.

Warrantability

The quality/ability/extent of being warrantable.

Warrantability is a system quality attribute that describes the ability of a software system to be maintained and supported by the software vendor or by the customer's own technical team. It is a non-functional requirement that specifies that the software must be designed in such a way that it can be easily updated, maintained, and repaired as needed throughout its lifetime.

Warrantability is a cross-functional constraint that covers both technical and business aspects. From a technical point of view, warrantability requires that the system architecture and design be modular, extensible, and well-documented. It also requires that the system be built using industry standards and best practices to ensure that it can be easily understood and maintained by other developers.

From a business perspective, warrantability imposes a set of obligations on the software vendor to provide customer support, bug fixes, and updates to ensure the software remains functional and secure over time. It also imposes an obligation on the customer to keep their own technical team current and knowledgeable about the software and to provide the vendor with the necessary feedback to improve the software as necessary.

In sum, warrantability is a critically important quality attribute that ensures the continued usability and value of a software system over time. Software vendors and customers must work together to ensure that the software is well-designed, well-documented, and well-maintained to meet the needs of all stakeholders.