IMPLEMENT QUALITY ATTRIBUTES IN SOFTWARE ENGINEERING

Soniya S Dadhania
Lecturer in Computer Engineering
R C Technical Institute, Ahmedabad

ABSTRACT: Quality attributes are the overall factors that affect run-time behaviour, system design, and user experience. They represent areas of concern that have the potential for application wide impact across layers and tiers. Some of these attributes are related to the overall system design, while others are specific to run time, design time, or user centric issues. The extent to which the application possesses a desired combination of quality attributes such as usability, performance, reliability, and security indicates the success of the design and the overall quality of the software application. When designing applications to meet any of the quality attributes requirements, it is necessary to consider the potential impact on other requirements. You must analyse the trade-offs between multiple quality attributes. The importance or priority of each quality attribute differs from system to system; for example, interoperability will often be less important in a single use packaged retail application than in a line of business (LOB) system.

Keyword: Conceptual integrity, Maintainability, Reusability, Availability, Interoperability

1. INTRODUCTION

The difference between an amateur product and a carrier grade product is not much in functionality; it is in Quality. For any serious business to depend on a piece of software to continue to function and evolve as needed a long list of quality attributes or 'abilities' are required. The list seems to be long, but each ability is vital. If you get stuck with something that doesn't have any one of the required abilities, that inability manifests itself in different problematic ways. It is imperative for operators to look at the products they are purchasing from all the quality attributes that may affect them in future. While most products claim to have similar functionality, it is important for the prudent buyer to ask tough questions about each individual required quality attribute and look under the hood on to the building blocks to see if the product they are buying really has the required quality attributes to deliver on all areas. Following is a list of commonly expected quality attributes with an introduction on how AdvOSS uses its technology and architecture to achieve them. These quality attributes are categorized with respect to the roles that typically have an interest in learning about these aspects of carrier grade software.

2. LIST OF ATTRIBUTES

High Availability

High Availability is the measure of the quality of software to keep functioning in spite of problems. Since the 'problems' can be of many types, different technologies work in tandem to achieve high availability for the overall system.

Redundancy

AdvOSS uses different technologies in combination to achieve redundancy in the system and make sure that the redundancy is used towards availability when needed.

Disaster Recovery

Disaster Recovery is the ability of the software to continue to function when a Disaster occurs.

Security

Security is the ability of the software to remain protected from unauthorized access. This includes both change access and view access.

Flexibility

Flexibility is the ability of software to adapt when external changes occur.

Traceability

Traceability is the ability of the Software to offer insight into the inner processing when required. A higher level of traceability is required at time of debugging a problem or at times of new interoperability testing. AdvOSS Switching and AAA products offer layers of traceability that can be turned on by a Maintenance engineer. Going from normal error only traces, to warning traces, to activity traces to full verbose tracing, AdvOSS products make it easy for the engineer to be able to see what is going under the hood.

Maintainability

Maintainability is the ability of a software to adapt to changes, improve over time, correct any bugs and be proactively fixed through preventive maintenance.

Testability

Testability is the ability of software to be tested thoroughly before putting into production. Although AdvOSS does internal testing before releasing any new versions, they can never be sure to work at an Operator's production systems without testing. AdvOSS offers a SandBox environment for each of its products. The Debugging Sandbox can be configured against given rules to siphon selected traffic to staging servers for real production simulated testing. This gets very useful in testing the new releases before production.

3. RELATIONSHIPS BETWEEN ATTRIBUTES

Each of the attributes examined has evolved within its own community. This has resulted in Inconsistencies between the various points of view.

3.1 Dependability Vis-a-vis Safety

The dependability tradition tries to capture all system properties (e.g., security, safety) in terms of dependability concerns—i.e., defining failure as not meeting requirements.‖ It can be argued that this is too narrow because requirements could be wrong or incomplete and might well be the source of undesired consequences. A system could allow breaches in security or Safety and still be called dependable. The safety engineering approach explicitly considers the system context. This is important because Software considered on its own might not reveal the potential for mishaps or accidents.

Types of factor

- the planes must be too close
- the pilots are unaware of that fact or
- the pilots are aware but
- fail to take effective evading action
- are unable to take effective evading action

3.2 Precedence of Approaches

Safe software is always secure and reliable — Neumann [Neumann 86] presents a hierarchy of reliability, safety, and security. Security depends on reliability (an attribute of dependability) and safety depends on security, hence, also reliability.

- A secure system might need to be reliable because a failure might compromise the system’s security (e.g., assumptions about atomicity of actions might be violated when a component fails).
- The safety critical components of a system need to be secure to prevent accidental or intentional alteration of code or data that were analysed and shown to be safe.
- Finally, safety depends on reliability when the system requires the software to be operational to prevent mishaps.

3.3 Applicability of Approaches

The methods and mind set associated with each of the attributes examined in this report have evolved from separate schools of thought. Yet there appear to be common underpinnings that can serve as a basis for a more unified approach for designing critical systems.

- Safety and dependability are concerned with detecting error states (errors in dependability and hazards in safety) and preventing error states from causing undesirable behaviour (failures in dependability and mishaps in safety).
- Security and performance are concerned with resource management (Protection of resources in security and timely use of resources in performance.)

The previous section offered examples of the applicability of methods usually associated with one attribute to other attributes. The applicability of methods developed for one attribute to another attribute suggests that differences between attributes might be as much a matter of sociology as technology. Nevertheless, there are circumstances for which an attribute-specific mind set might be appropriate.

Examples include the following:

- The dependability approach is more attractive in circumstances for which there is no safe alternative to normal service—a service must be provided (e.g., air traffic control).
- The safety approach is more attractive where there are specific undesired events — an accident must be prevented (e.g., nuclear power plant).
- The security approach is more attractive when dealing with faults of commission rather than omission — service must not

be denied, information must not be disclosed.

4. QUALITY ATTRIBUTES AND SOFTWARE ARCHITECTURE

A (software) system architecture must describe the system’s components, their connections and their interactions, and the nature of the interactions between the system and its environment. Evaluating a system design before it is built is good engineering practice. A technique that allows the assessment of a candidate architecture before the system is built has great value. The architecture should include the factors of interest for each attribute. Factors shared by more than one attribute highlight properties of the architecture that influence multiple attribute concerns and provide the basis for trade-offs between the attributes. A mature software engineering practice would allow a designer to predict these concerns through changes to the factors found in the architecture, before the system is built. All the attributes examined in this report seem to share classes of factors. There are events (generated internally or coming from the environment) to which the system responds by changing its state. These state changes have future effects on the behavior of the system (causing internal events or responses to the environment). The environment‖ of a system is an enclosing system,‖ and this definition applies recursively, up and down the hierarchy. In addition to evaluating individual patterns, it is necessary to evaluate compositions of patterns that might be used in architecture. Identifying patterns that do not compose‖ well (i.e., the result is difficult to analyze or the quality factors of the result are in conflict with each other) should steer a designer away from difficult architectures, towards architectures made of well-behaved compositions of patterns. In the end, it is likely that we will need both quantitative and qualitative techniques for evaluating patterns and architectures. Promising quantitative techniques include the various modeling and analysis techniques, including formal methods mentioned in this report. An example of a qualitative technique is being demonstrated in a related effort at the SEI. The Software Architecture Analysis Method (SAAM) illustrates software architecture evaluations using scenarios (postulated set of uses or transformations of the system). Scenarios are rough, qualitative evaluations of architecture; scenarios are necessary but not sufficient to predict and control quality attributes and have to be supplemented with other evaluation techniques.

5. CONCLUSION

The qualities presented those most often the goals of software Architects. Since their definitions overlap, we chose to characterize them with general Scenarios. We saw that qualities can be divided into those that apply to the system, those that apply to the business environment, and those that apply to the architecture itself. In the next chapter, we will explore concrete architectural approaches for following the path from qualities to architecture.

REFERENCE

[1] Al-Kilidar, H., Cox, K., &Kitchenham, B. (2005).The use and usefulness of the ISO/IEC 9126 Quality Standard. In International symposium on empirical software engineering (pp. 126–132). Noosa Heads,.Australia: IEEE Computer Society.

[2] Ameller, D., Galster, M., Avgeriou, P., &Franch, X. (2013). The role of quality attributes in service- based systems design. In 7th European conference on software architecture (ECSA) (pp. 200–207). Montpellier, France: Springer.

[3] Bachmann, F., & Bass, L. (2001). introduction to the attribute driven design method. In 23rd international conference on software engineering (pp. 745–746). IEEE Computer Society. Bachmann, F., Bass, L., Klein, M., & Shelton, C. (2005).

[4] Designing software architectures to achieve quality attribute requirements. IEE Proceedings Software, 152, 153–165.

[5] Balasubramaniam, S., Lewis, G. A., Morris, E., Simanta, S., & Smith, D. B. (2009). Challenges for assuring quality of service in a service-oriented environment.

[6] Barbacci, M. R., Ellison, R. J., Lattanze, A. J., Stafford, J. A., Weinstock, C. B., & Wood, W. G. (2003). Quality attribute workshops (QAWs), third edition. Technical report, SEI CMU.

[7] Basili, V., Caldiera, G., & Rombach, D. (1994). The goal question metric approach. In J. J. Marciniak (Ed.),Encyclopedia of software engineering (Vol. 1, pp. 528–532). New York, NY: Wiley.

[8] Software quality attributes and trade-offs, Patrik Berander, Lars-Ola Damm,

Blekinge Institute of Technology June 2005

[9] A tertiary study on links between source code metrics and external quality attributes, Umar Iftikhar,Nauman Bin Ali, Volume 165,January 2024

[10] Quality Attributes Optimization of Software Architecture: Research Challenges and Directions, Daniele Di Pompeo University of L'Aquila, L'Aquila, Italy, 10.1109/ICSA-C57050.2023.00061

[11] Enhancing Software Quality in Architecture Design: A Survey- Based Approach, Shravan Pargaonkar, 10.29322/IJSRP.13.08.2023.p14014, August 2023