SCS 2203 Software Engineering

Assignment 1

Achieving Performance and Usability

Name: Nethsara Sandeepa Elvitigala

Index No: 20000482

1. Achieving performance and usability Quality Attributes

As opposed to functional requirements which are the basic capabilities, services and behavior of a system, Quality attributes are measurable or testable properties of a system that can indicate how well the system satisfies needs of its various stakeholders. This can mean quite a lot of perspectives such as how easy it is for users to use the system, how it performs under heavy load etc. We can think of quality attributes as ways to measure how well a functional requirement is implemented in the system. [1]

While functional requirements are attainable regardless of the architecture most of the time, achieving quality attributes is hard and careful consideration should be done from the start. Even though functional requirements take the priority in most projects, it's probably not the best approach as when most systems fail, it's not due to a lack of functionality but lack of these quality attributes. Thus, it's important to choose right architectures for the scenarios at hand to make sure that system is successfully used by users. The structures and behaviors that incorporate these structures in an architecture are the main driving force behind achieving these quality attributes. [1]

Following are the main quality attributes we need to take care of when designing a system.

- Availability
- Interoperability
- Modifiability
- Performance
- Security
- Testability
- Usability

Aside from these, concerns about following attributes frequently arises as well, variability, deployability, scalability etc.

Let's look at Performance and Usability to figure out how to achieve those.

2. Performance

Performance of a system is one of the most important quality attributes of a software system. It's all about the system's ability to respond to events in acceptable time frames. There are multiple types of events that can occur in a system, like requests from users or other systems, UI interactions, system interrupts etc. And the system must be well equipped to respond to them in time. Based on the type of system, performance can be measured in different ways. For an example, in a web-based system, we could measure performance by measuring how many user requests can be processed by the system per minute. [1]

While performance requirements are not always clearly expressed, it's a very noticeable attribute. As such, performance has been the driving factor in choosing a system architecture for a long time. The general scenario for performance is like this. Main part of this is the "stimulus" or the arrival of an event. These stimuli arrive from an external or internal "source". "Artifact" is the parts of them system affected by this arrival of the event. "Environment" refers to the operational mode the system is currently running, for an example peak load and normal operation. System processes these events, creating "response"s. This can change the environment of the system. Finally, we have "Response Measure", which are the various time measures relating to the response such as latency, jitter, throughput and miss rate. [1]

To achieve performance goals, we have performance tactics. The goal of them is to generate responses to stimuli within an acceptable time constraint. Let's look at these tactic categories.

2.1 Control resource demand

Controlling resource demand improves performance by reducing the load on system. This can be achieved by limiting the rate the system responds to events or limiting the rate of events. There are some techniques to achieve this in a software system. One way is to reduce the respond rate for incoming events. However, this must be implemented carefully with a technique like queueing events to ensure that all events are responded albeit at a reduced rate. But in case we purposely design the system to drop some events, we must decide on parts of the system that should be informed. [1]

Another way of increasing performance is to prioritize events. In a particular system, there might be events that are not as important as other ways. By implementing a way to properly categorize them according to priority and responding them in a respective order, we can ensure that top priority events are always processed first. [1]

There's also another way of controlling resource demand that is a tradeoff between modifiability and performance. Most system include some intermediate steps put there to increase the modifiability, but this in turn reduces performance because of overhead. Removing these unnecessary extra steps can improve performance by reducing metrics like network latency. As this is a conflicting situation, a decision must be made carefully as to whether prefer performance or modifiability is the most important in the scenario. [1]

In some systems, performance can be improved by setting time limits on how long an event should be processed. This also comes with a caveat as reduced time limit means less accurate calculations in these kinds of systems. So, it must be considered carefully. But there is a way to improve performance in some systems that doesn't have any obvious caveats. That is to develop more efficient algorithms for processing. Since algorithms determine the time spent on a calculation, a better algorithm can lead to drastically improved performance. [1]

Also, in systems which work with data streams, performance can be improved by reducing the fidelity in turn improving the consistency. So as most of above explained techniques, a careful consideration is required. Next main tactic to improve performance is Managing resources

2.2 Managing resources

Even in scenarios where controlling demand is not possible, resources can be added or improved to increase performance. Following are some techniques of improving resources

- Increase resources
  By simply adding new processors, additional memory, faster networks or even faster storage devices, we can reduce the latency of most systems, while cost plays a huge factor in deciding on this route, this maybe the cheapest way to see immediate improvements in performance. [1]
- Introduce concurrency

  By processing requests in parallel, if possible, we can reduce the time where a system is blocked due to its processing another event. [1]

- Maintaining multiple copies of data
  This refers to caching data. In most implementations, data is cached based on recent requests. But in some advanced implementations, we can even cache beforehand by predicting user behavior. However, there is responsibility to make sure that irrelevant (old) copies of data are discarded when they are modified. [1]

- Maintaining multiple copies of computations
  In a client-server architecture pattern, we can process events on multiple servers to eliminate the waiting that happens when all events are processed in the same server. However, a load balancer must be configured to correctly distribute incoming event among these servers [1]
- Bound queue sizes

By controlling the maximum sizes of queues, we limit the processing of events. However, since this means some events are not processed when the queue is overflowed, we must have a policy in the system to handle that. [1]

3. Usability

The ease with which a user may complete a desired task and the type of user support the system offers are both factors in usability. An emphasis on usability has proven to be among the cheapest and simplest improvements over time. [1]

The end user, who may be acting in a specialized capacity such as a system or network administrator, is always the source of the stimulus for usability in a general scenario. In stimulation, the end user wants to utilize a system effectively, learn how to use it, reduce the impact of errors, customize it, or adapt it. [1]

In an environment, user actions that are relevant to usability always take place during runtime or during system configuration. The system or particular component of the system that the user is dealing with is the artifact. When it comes to responsiveness, the system should either give the user the functionality they require or foresee their demands. Task time, the number of errors, the number of tasks completed, user happiness, knowledge gain, the proportion of successful operations to all operations, or the amount of time or data lost when an error occurs are all used to gauge the response. Usability relates to how easy it is for the user to accomplish a desired task, as well as the kind of support the system provides to the user. Researchers in human computer interaction have used the terms user initiative, system initiative and mixed initiative to describe which of the human-computer pair takes the initiative in performing certain actions and how the interaction proceeds. [1]

Usability is improved once a system has started working by informing the user of what the system is doing and allowing them to respond appropriately. The system's duties for responding to the user command are listed and distributed by the architect as they create a response for user initiative. Cancel, undo. Aggregate and pause/resume are some examples of user initiative.

When the system initiates, it must rely on a model of the user, the activity the user is performing, or the current state of the system. Each model needs a different kind of input to carry out its initiative. The strategies that identify the models the system employs to forecast either its own behavior or the user's intention are known as support system initiative strategies. This information will be easier to alter or modify if it is encapsulated. Customization and change can be done offline during development or dynamically based on previous user behavior.

Reference List

[1] Len Bass, Paul Clements, Rick Kazman, *Software Architecture in Practice*, Third Edition.