The audience will…

1. Become familiar with patterns that can be used to define non-functional requirements with clarity, precision, and required level of detail.
2. Learn how to use non-functional requirement patterns to better define quality scenarios that are the basis for the definition of the software architecture.
Introduction: Requirements Engineering

- Requirements Engineering (RE) is a set of activities devoted to understanding the needs of relevant stakeholders; comprehending the context in which the future software will operate.
- Requirements analysts typically start with poorly defined and often conflicting ideas on what the system is supposed to do.
- When stakeholders define a software-intensive system, it is more natural that they focus on specifying the functional requirements.
- Users have also expectations on how well the product will work such as how easy it is to use, how quickly it runs, etc.
- These characteristics are called software quality attributes.
- Quality attributes are difficult to define.
- Excellent software products exhibit an exquisite balance among competing quality attributes.
- Quality attributes drive architectural decisions for the software system.
The Volere Shell © Elements

- Requirement number (unique identifier).
- Requirement type (functional, usability, etc.).
- Even/use case number (to cross-reference).
- Description (natural language statement).
- Fit Criterion/criteria (quantified goal that requirement has to meet).
- Rationale (reason behind requirement).
- Source (origin of requirement).
- Customer satisfaction (value on how satisfied a customer would be if this requirement were to be included in the software).
- Customer dissatisfaction (value on how dissatisfied the customer would be if requirement were not included in the software).
- Dependencies (associated requirement).
- Conflicts (conflict requirement ID).
- Supporting materials.
- History (relevant historical information on requirement).

Software Architecture
The Process
Performance Requirements
Patterns

Software Performance Quality
Dependencies

- Performance
  - Architectural Dependencies
    - Communication volume among components
    - Component functionality
  - Non-Architectural Dependencies
    - Shared resources allocation
    - Choice of algorithms
    - Algorithm implementation
Performance Requirements
Patterns

<table>
<thead>
<tr>
<th>Response Time Requirements Pattern</th>
<th>Throughput Requirements Pattern</th>
<th>Dynamic Capacity Requirements Pattern</th>
<th>Static Capacity Requirements Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common Performance Requirements Principles</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Common Performance Principles

- Performance requirements are easy to write and hard to implement
- Software and hardware work together
- Identify where in the system the performance target belongs
- Avoid arbitrary performance requirements
- Define criticality of meeting performance requirement
- Measure the performance of the current system
- Define the timing of performance target.
- Define one performance target per requirement.
- Think options on how to meet a performance requirement
- Build a sizing model to conduct sensitivity analyses of performance

(Withall, 2007)
Response Time Requirement Pattern

- This requirement pattern is typically used when the user is interested in the time that takes an operation to be performed.
- This requirement is stated as response time.
- Response time is the length of time between a request being submitted at a particular location of a system and a response being perceived at the same location.
- It is typically applied to user response time, which is the length of time between a user submitting the request and a response being displayed on the screen.
Throughput Requirement Pattern

- This requirement determines how fast the system can deliver its output. It is essential to have a sound basis for determining a target throughput. Consider the following steps to define a target throughput:
  - Decide what to measure in the system based on highest importance. Distinguish between incoming and outgoing throughput.
  - Identify other relative volumes of secondary activities if needed.
  - Specify required hardware set-up.
  - Determine average throughput over a relatively long period of time.
  - Determine peak throughput the system must cope with the peak load.
Dynamic Capacity Requirement Pattern

- This requirement pattern specifies the quantity of a particular type of entity for which the system must be able to perform processing at the same time.
- This pattern helps specifying the quantity of a particular type of entity for which the system must perform processing at the same time.
- It is intended primarily for the number of simultaneous users a system must be capable of handling. It also suggests what to do when too many users come along all at once.
- Specifying dynamic capacity is difficult unless there is an existing system for which figures can be obtained.
Static Capacity Requirement Pattern

- This requirement specifies the quantity of a particular type of entity that the system must be able to store permanently (typically in a database).
- It should not be used to specify how long the data must be stored or how much disk space the system requires.
- The importance of static capacity is indirect in that all aspects of the system must be designed and built so as to be practical and work well when the target number of entities is present.
- Most systems have one type of entity that determines the quantity of most or all other high volume entities; customer is often the best type of entity to use.

Static Capacity Requirement Pattern

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Requirement Type</th>
<th>Prove/disprove</th>
</tr>
</thead>
<tbody>
<tr>
<td>231</td>
<td>Performance</td>
<td></td>
</tr>
</tbody>
</table>

Description
- Operation type: The system shall be able to process a minimum of

Fill Criteria
- Entity Count: 1,000,000
- Entity Type: Customers
- Entity Inclusion Criteria: This figure covers only those customers who have accessed the Web site in the past three months or placed an order within the past twelve months.
- Achievement Timeline Statement: It is not expected that this level of business will be reached earlier than two years after initial implementation.

Rationale
- Previous version of the software has shown this to be an acceptable number of customers.

Source

Motivation
- Customer Satisfaction
- Customer Disatisfaction

Dependence

Supporting Material

History
Architectural Scenario

Elements

- A quality attribute scenario is directly derived from a non-functional requirement and consists of the following parts:
  - **source of stimulus**, which is the entity that generated the stimulus;
  - **stimulus**, which is the condition that needs to be considered when it arrives at a system;
  - **environment**, which determines under which conditions the stimulus occurs;
  - **artifact**, is the element that receives the stimulus;
  - **response**, which is the activity undertaken after the arrival of the stimulus;
  - **response measure**, which represents the way a response is measured when it occurs.

Architectural Scenario

Throughput Requirement

Source: Customer

Stimulus: Enters orders

Artifact: Processor Module

Response: Orders completed

Response Measure: 10 orders per second

Environment: Under normal conditions
Conclusions

- Defining functional requirements in software-intensive systems is always complex as required level of specificity is often “elusive”
- If defining functional requirements is difficult, defining non-functional requirements seem to be even more difficult because:
  - Clearly defining quality attributes of a system requires a lot of precision
  - Often testing non-functional requirements is not in the minds of people that define the non-functional requirements
- Architectural scenarios can be easily derived as all information needed is defined in the requirements patterns shown in this presentation
- Non-functional requirements patterns help to define test cases for the architectural scenarios

Summary

Contact Information

Author: Aldo Dagnino
ABB Corporate Research
Industrial Software Systems Program
940 Main Campus Drive
Raleigh, NC 27606 USA

Email: aldo.dagnino@us.abb.com
Phone: (919) 807-5731
References