Principles of Architecture Design and Evaluation

Len Bass

Sponsored by the U.S. Department of Defense © 2004 by Carnegie Mellon University
The key question

How do we systematically move from a set of requirements to a software architecture that satisfies those requirements?
Why build a computer system?

Designer → Software Architecture → System
To achieve some business goals

Business Goals

- Financial/mission
  - Improve operations
  - Offer new services
- Marketing
  - Extend market share
  - Improve customer satisfaction
- Other
  - Get tenure/degree
  - Make political point
Business Goals Beget Requirements

But a funny thing happens on the way to a system...
Some sources of requirements

- Stakeholder Needs
- Developing Organization
- Legal/Contractual Issues
- Commercial/Competitive Pressures
- Technical Environment
- Political Issues
- Life cycle Issues

Architect's Experience

Quality Attribute Requirements
Business Requirements
Functional Requirements

Architecture
System
Key ideas of this section

Many different sources of requirements

Written requirements are almost always incomplete

Architect's experience is vital element in interpreting remaining sources of requirements
Outline

Motivation

Principles

Application of principles to evaluation and design
How does the architect make sense of the requirements?

Architect's Experience

Quality Attribute Requirements

Business Requirements

Functional Requirements

© 2004 by Carnegie Mellon University
Types of requirements

Requirements
- Business – constraints on design
- Functional requirements – what adds value to the user (e.g., what the system does)
- Quality Attribute – how well the system does by various measures (e.g., how timely, secure, modifiable it is)

Software Architectures
- Functions are:
  - Features + necessary non user visible computations
  - Represented within architecture by responsibilities
Constraints – pre-specified design decisions

Requirements

Software Architectures

Constraints

Constraints reduce the space of architectures in which to search for a solution

© 2004 by Carnegie Mellon University
Must live within constraints

Very little software design is "greenfield"

Frameworks, large-grained components are frequently required.

Disciplined design must accommodate constraints.

Designer does not make design decisions to achieve constraints – constraints are given.

Designer makes design decisions to achieve other requirements within given constraints.
Do functional or quality requirements drive remaining architectural design?

QA Quality requirements determine most architectural design decisions – not functional requirements

If the only concern is functionality then a monolithic system would suffice.

However is it quite common to see:
• Redundancy structures for reliability
• Concurrency structures for performance
• Layers for modifiability
Principle 1

Quality attribute requirements are those that drive the design of the software architecture

Leads to several questions:
1. How are quality attribute requirements specified
2. How are quality attribute requirements achieved
3. How can understanding of the impact of quality attributes on design be used to improve the design and evaluation processes?
Characterizing quality attributes

Requirements such as "the system shall be modifiable", are meaningless. What does it mean to say "the system shall be modifiable"?

Taxonomies of quality attributes do not help. Is a denial of service attack a security problem, a performance problem, an availability problem, or a usability problem?
A common form for the specification of quality requirements

We use **quality attribute general scenarios**, which are system independent, to guide the specification of quality attribute requirements.

We characterize quality attribute requirements for a specific system by a collection of **concrete quality attribute scenarios**. These are instances of general scenarios.

We use **general scenario generation tables** to construct well-formed general scenarios for each attribute.
General scenarios

General scenarios have six parts. The "values" for each part define a vocabulary for articulating quality attribute requirements. The parts are:

• Stimulus
• Source of stimulus
• Environment in which the stimulus arrives
• Artifact influenced by the stimulus
• Response of the system to the stimulus
• Response measures
### Availability scenario generation table

<table>
<thead>
<tr>
<th>Source of stimulus:</th>
<th>Stimulus:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Internal to the system</td>
<td>✓ Unanticipated event</td>
</tr>
<tr>
<td>✓ External to the system Environment</td>
<td></td>
</tr>
<tr>
<td>✓ Normal operation</td>
<td>• Update to a data store</td>
</tr>
<tr>
<td>• Degraded mode</td>
<td></td>
</tr>
<tr>
<td>Response:</td>
<td>Artifact:</td>
</tr>
<tr>
<td>✓ record it</td>
<td>✓ Process</td>
</tr>
<tr>
<td>✓ notify parties</td>
<td>• Persistent storage</td>
</tr>
<tr>
<td>✓ operate in normal or degraded mode</td>
<td>Response measures:</td>
</tr>
<tr>
<td></td>
<td>✓ Availability percentage</td>
</tr>
<tr>
<td></td>
<td>• Time range in which the system can be in degraded mode</td>
</tr>
</tbody>
</table>

**Example Scenario:**

"An unanticipated message is received by a system process during normal operation. The process has to record it, inform the appropriate parties and continue to operate in normal mode without any downtime."

© 2004 by Carnegie Mellon University
Constructing quality requirements from general scenarios

Generate a possible general scenario by choosing one or more entries from each list and combining them.

Not all:

- general scenarios are relevant to specific system
- generated scenarios make sense

Make each scenario system specific (concrete scenario).

May be multiple concrete scenarios for each general scenario.

E.g., modify function.

Eliminate duplicates.
Which attributes?

We have focused on six quality attributes:

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

Others are equally important:

- Buildability
- Interoperability

...
What about function and quality?

Software for garage door opener

Some scenarios:
"Halt garage door when an obstacle is detected"
"respond to user's requests to raise/lower the door within .5 second"
"replace sensor/actuator within 40 staff hours"

Functional or quality requirements?
Functional AND Quality

Every requirement has both functional AND quality portions.

E.g. Halt garage door when an obstacle is detected.

Function: detect obstacle, halt garage door

Quality: within time limit (implicit in this example).

Scenario template provides means for eliciting quality requirements associated with functions.

Quality portion leads to design template in which to situate functionality.
Key Ideas of this section

Can express business goals as quality scenarios

Can characterize quality scenarios in structured fashion

For six attributes, we have tables that enable the generation of quality attribute scenarios

Quality attribute scenarios provide quality attribute requirements to particular functions
Principle 2

Quality attribute requirements can be specified through concrete scenarios with six parts.

Still questions left:
1. How are quality attribute requirements achieved
2. How can understanding of the impact of quality attributes on design be used to improve the design and evaluation processes?
What does it mean to achieve a quality attribute requirement?

A quality attribute requirement defines a region within the set of quality attribute measures. An architecture can be interpreted in terms of a quality attribute model which, in turn, can be evaluated to determine which quality attribute value the software architecture will achieve for a particular stimulus.

If evaluated value is inside the region defined by the requirement, the requirement is satisfied.
Achieving quality attribute requirements

Design is the process of making decisions about various design options.

We can enumerate decisions known to be useful in achieving different quality attributes.

For example: what are some decisions used to improve performance? To improve modifiability? To improve availability?
What are these architectural decisions?

For the six quality attributes — availability, modifiability, performance, security, testability, usability — we have enumerated a collection of “tactics”

Formal definition: An architectural tactic is a means of satisfying a quality attribute response measure by manipulating some aspect of a quality attribute model through architectural design decisions.
An architectural tactic moves from one architecture to another where a parameter of the quality attribute model moves in a known direction.
Tactics identify key quality attribute concepts and bridge quality attribute model and architectural design

- Modifiability model has concepts such as "dependency"
- A tactic for controlling dependency is "use an intermediary"

Quality attribute models (analytic, empirical or qualitative) drive the identification of tactics

- Derived from well-known analytic models
- Also derived from attribute experts
Availability tactics

Availability

- Fault Detection
  - Ping/Echo
  - Heartbeat
  - Exception
- Recovery - Preparation and Repair
  - Voting
  - Active Redundancy
  - Passive Redundancy
  - Spare
- Recovery - Re-Introduction
  - Shadow
  - State Re-Synchronization
  - Rollback
- Prevention
  - Removal from Service
  - Transactions
  - Process Monitor
Principle 3

Quality attribute requirements can be achieved through application of architectural tactics

Still one question left:

1. How can understanding of the impact of quality attributes on design be used to improve the design and evaluation processes?
Architectural evaluation

Architectural evaluation – examine existing architecture to determine how well it satisfies its quality attribute requirements.

Ideally limited amount of time is spent on the evaluation.

Leads to three problems:
1. Which quality attribute requirements are the focus of the evaluation?
2. Which portion of the architecture is the focus of the evaluation?
3. What do you look for inside the architecture?
Which quality attribute requirements?

Want to focus on those requirements that are most important for business goals and have largest impact on architecture.

Suggests that input is needed from:
business people – e.g. marketing
technical people – e.g. architect
How ATAM focuses on Requirements

The SEI method – ATAM™ - has steps that select scenarios to focus on. These steps involve multiple stakeholder representatives including architect and marketing.
Which portion of the architecture is the focus of the evaluation?

Architecture is large and it is time consuming to examine all of it.

Want to emphasize that portion of the architecture that realizes the scenarios that are the focus of the evaluation.
How ATAM focuses on the architecture

ATAM uses the architect to identify the portion of the architecture affected by a scenario.

The architect walks through how the scenario is achieved.
What do you look for inside the architecture?

Once focused on a portion of the architecture the evaluator must determine whether any architectural decisions affect business goals.

Evaluator examines the architecture for any business goals, not just ones affected by focusing scenario.
ATAM uses tactics (and lack of tactics)

Use of tactics (and lack of use of tactics) are indicators of whether the architecture has a problem achieving a particular business goal.
Key ideas of this section

Can evaluate software architecture early to determine whether there are risks in satisfying important quality attribute requirements.

ATAM is a method for evaluation that:

• Uses stakeholders to determine important quality attribute requirements
• Uses architect to focus on important portions of the architecture.
• Uses use of tactics (or lack) to determine potential problems.
Design decisions

Recall that design is the process of making decisions

Some decisions are more important and are made earlier

We have a collection of decisions we call "early design decisions"

- May be constrained by environment and problem
- May be free for architect to decide
Sample early design decisions

Statefull vs. stateless

Synchronous vs. asynchronous

Point-to-point vs. client/server

Federated (loosely coupled) vs. tightly coupled

...
Sample – statefull vs. stateless - 1

Statefull portions of the system have following characteristics:

Communication requirements reduced because state does not need to be communicated with each message

Security easier to maintain since state is maintained within scope of components being considered

...
Sample – statefull vs. stateless - 2

Stateless portions of the system have following characteristics:
- Availability increased since do not lose state when component fails
- Load balancing easier since can always replicate stateless component to handle multiple requests.

...
Early decisions

Characterized in terms of quality attributes and how they are supported by tactics

How are early decisions evaluated?

- Through consideration of business goals.
Architectural drivers

An architectural driver is a quality attribute requirement (concrete scenario) that has major impact on the architecture.

Usually small number of architectural drivers (~5-8)

Located through examination of high business priority quality attribute requirements
Attribute driven design (ADD)

Decomposition method beginning with constraints

Decomposition is designed to satisfy the architectural drivers of the system

Decomposition intended to place early design decisions in a pattern
Key ideas of this section

Since quality attribute requirements are most important design drivers, must identify most important quality attribute requirements

Can use early design decisions, quality attribute requirements and tactics to define pattern that satisfies most important requirements
Summary

Quality attribute requirements determine architectural design

Quality attributes requirements can be expressed in a common form

Architectural tactics are an enumeration of techniques that architects use to achieve particular quality attributes

Knowledge of quality attributes can be embodied in design and evaluation methods.
Lists of general scenarios and tactics, descriptions of ATAM and ADD are available in second edition of *Software Architecture in Practice*

http://sei.cmu.edu/architecture

Software architecture curriculum

Len Bass: lj@sei.cmu.edu