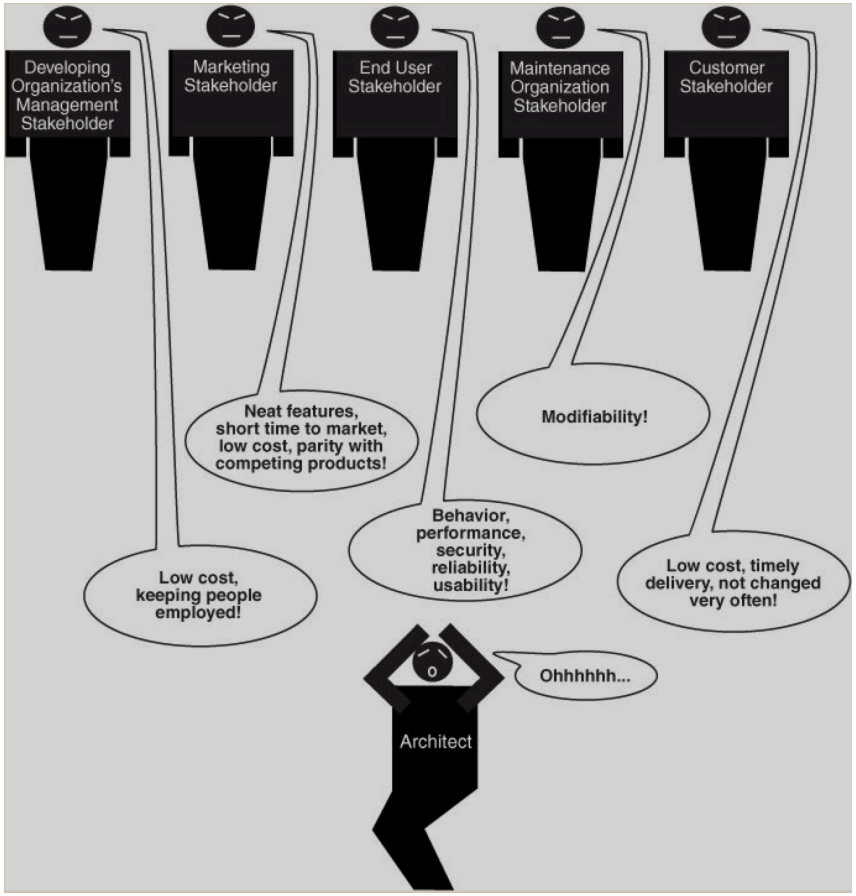
SAPM Meeting 4
Quality Attributes

Architect Influences

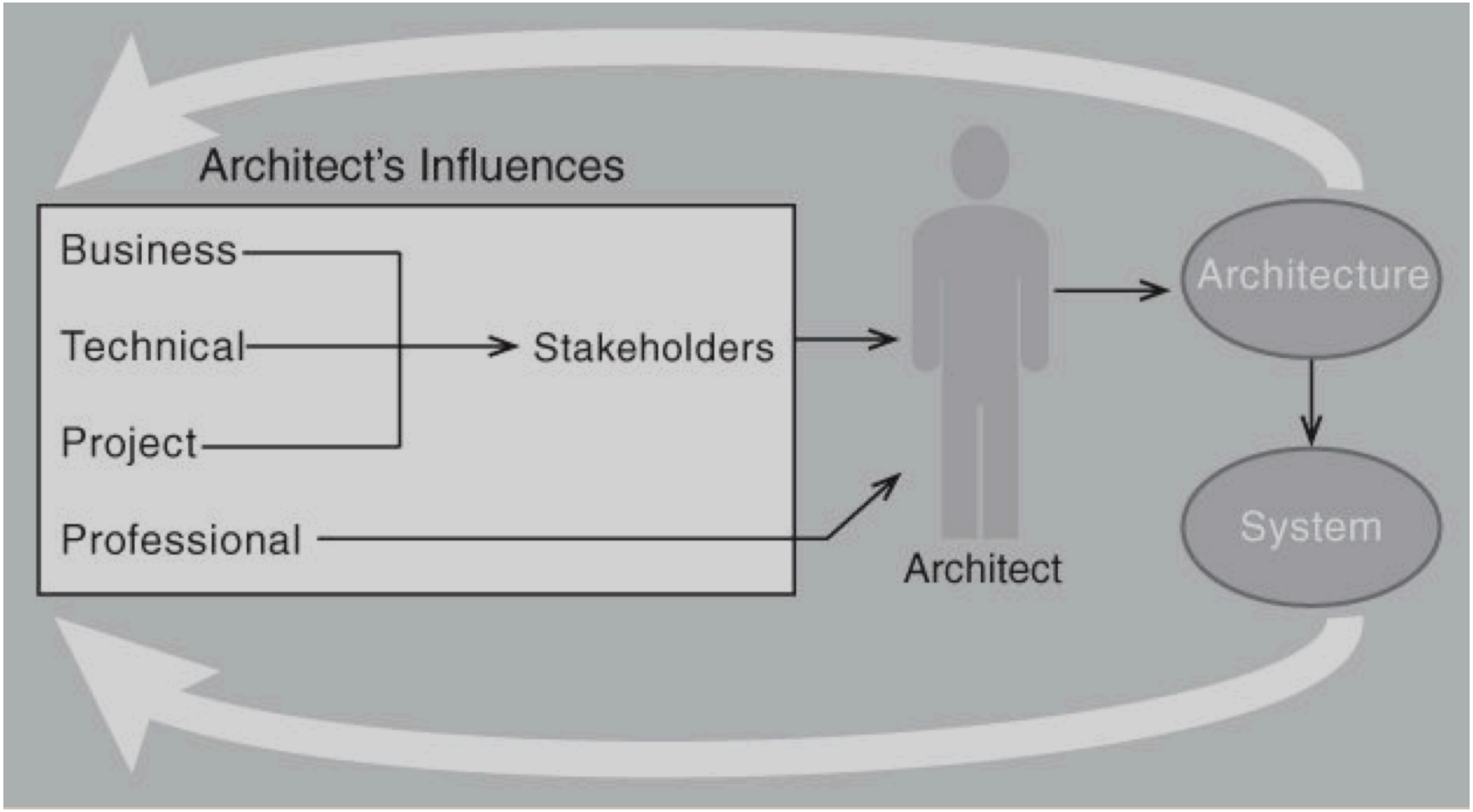
- We saw in the previous lecture that architects are influenced by business, technical, project and professional contexts.
- In the case of business, technical and project contexts these influences are carried by stakeholders.
- Stakeholders represent different perspectives on the system and attempt to influence the architect.
- Later we will see how the architect needs to tradeoff the different influences of stakeholders.

Stakeholders

- Stakeholders typically have conflicting perspectives on systems.
- For example, marketing might like to have as big a market as possible but the maintenance organisation might want the system to be a simple as possible.
- Architecture is the right level of abstraction to resolve these conflicts.



Architecture Influence Cycle



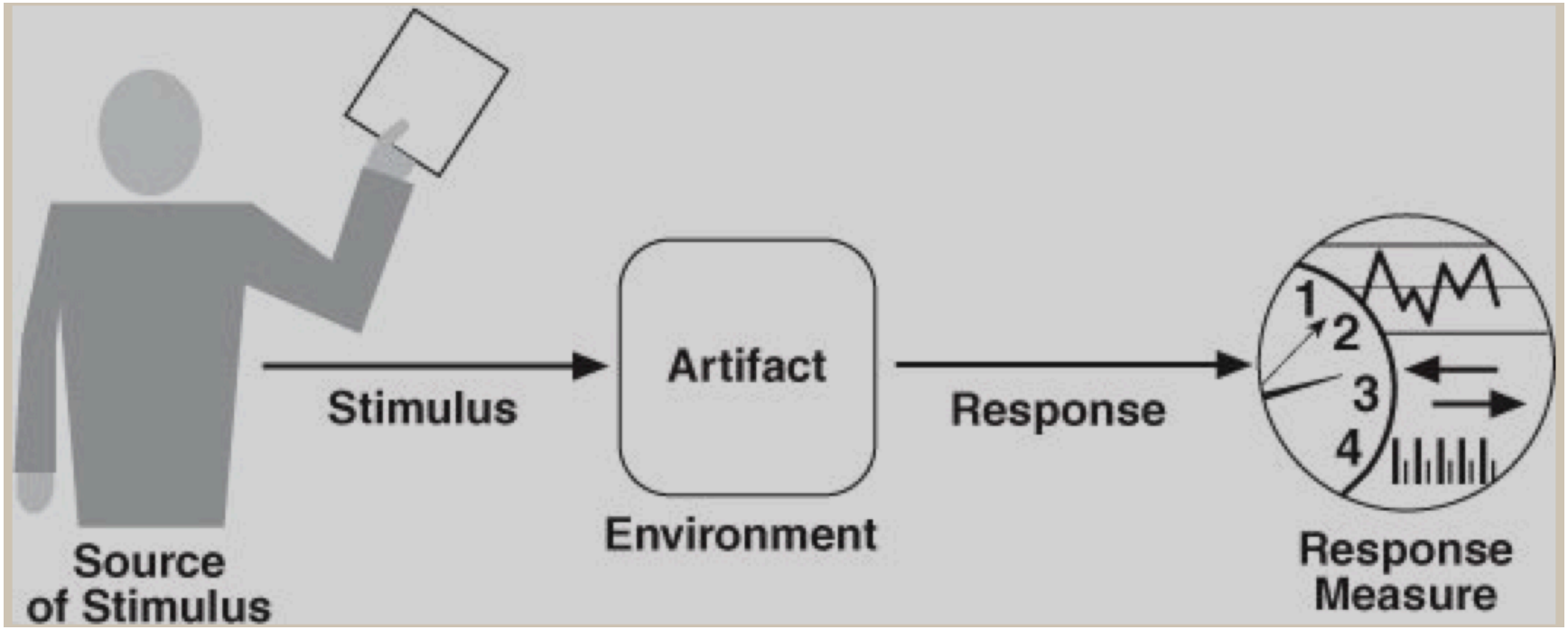
Requirements

- Functional requirements: these specify what the system does, architecture is important to this by structuring the containers that hold functionality.
- Quality Attributes: these specify, usually quantitative, requirements on particular bits of functionality or on the whole systems (e.g. that the system should be available 99% of the time).
- Constraints: these are decisions that have already been taken e.g. we will use the Java programming language (because we have a Java development team available) or the system will only support certain web browsers

Problems With Quality Attributes

- Often QA requirements are not “testable”, for example what does it man to say something is modifiable or usable or dependable or resilient?
- It can be difficult to map from a concern about the system to a QA. For example, a high failure rate in some transaction could be a performance issue or it could be an availability issue.
- Communities around a particular quality attribute have developed their own terminology (e.g. security has attacks, performance has events, etc).
- The solution is to use quality attribute scenarios to provide sufficient specificity to avoid some of these issues

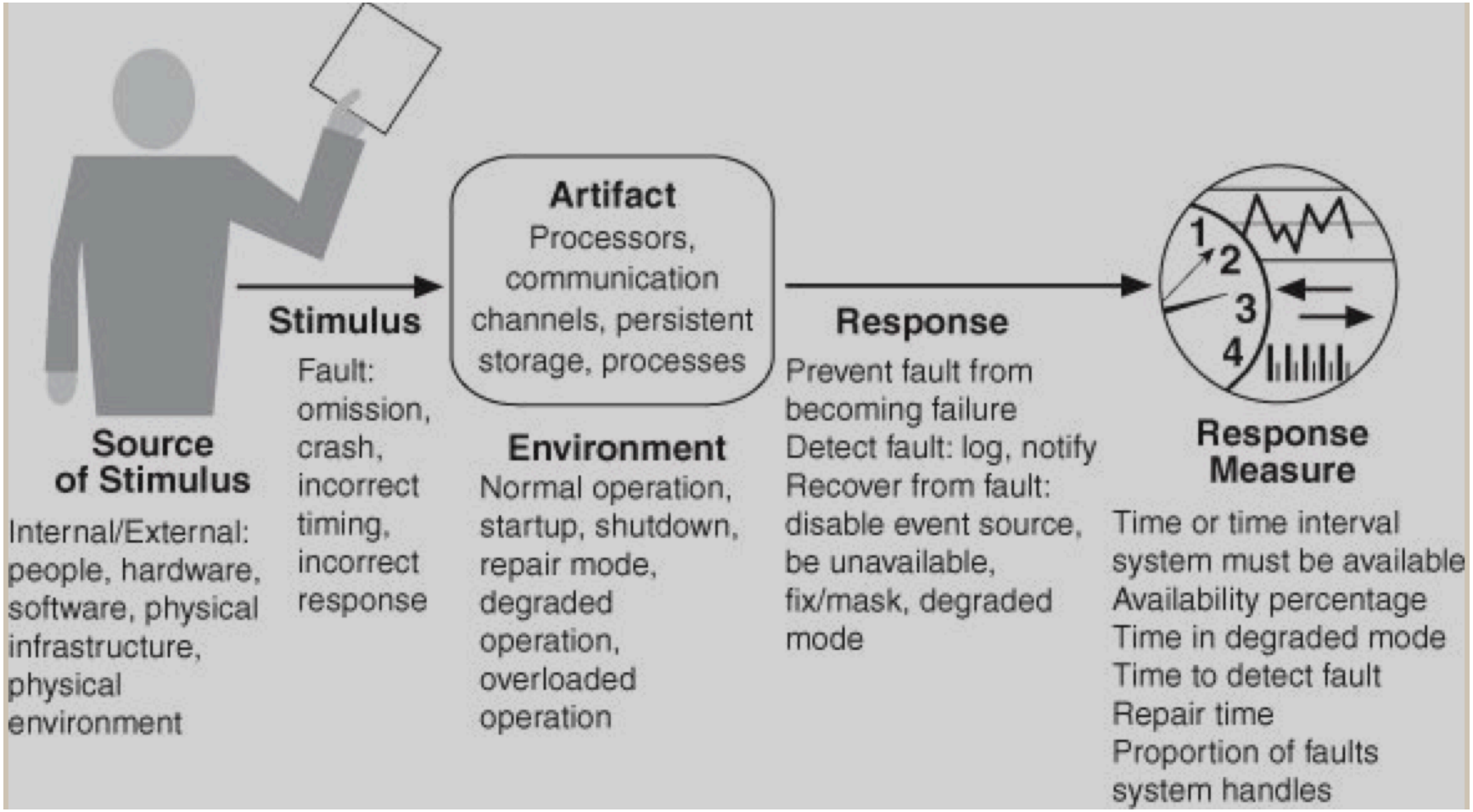
Quality Attribute Scenarios



QA Scenarios

- QA Scenarios have the following components:
  - Source of stimulus: might be a person or another system e.g. a system administrator.
  - Stimulus: this is something that the system needs to respond to e.g. using the wrong configuration specification for the system.
  - Environment: This is the broader context that captures wider aspects of the system e.g. that the system is starting up
  - Artifact: this is the part of the system that is stimulated e.g. the configuration checker in the system
  - Response: The response is the activity resulting from the stimulus e.g. the configuration issue is identified and then repaired
  - Response measure: this is how to measure the response so the scenario is testable e.g. time to detect the wrong configuration and the time to repair.

General Scenario for Availability



General and Specific QA Scenarios

- Each QA has a general scenario associated with it that tries to capture the possible components involved in that particular QA
- This acts as a template or guide for the architect specifying a specific QA Scenario.
- Specific QA scenarios take account of specific stimuli and measures on response, they capture the specification of the QA for a particular system.

Architectural Tactics

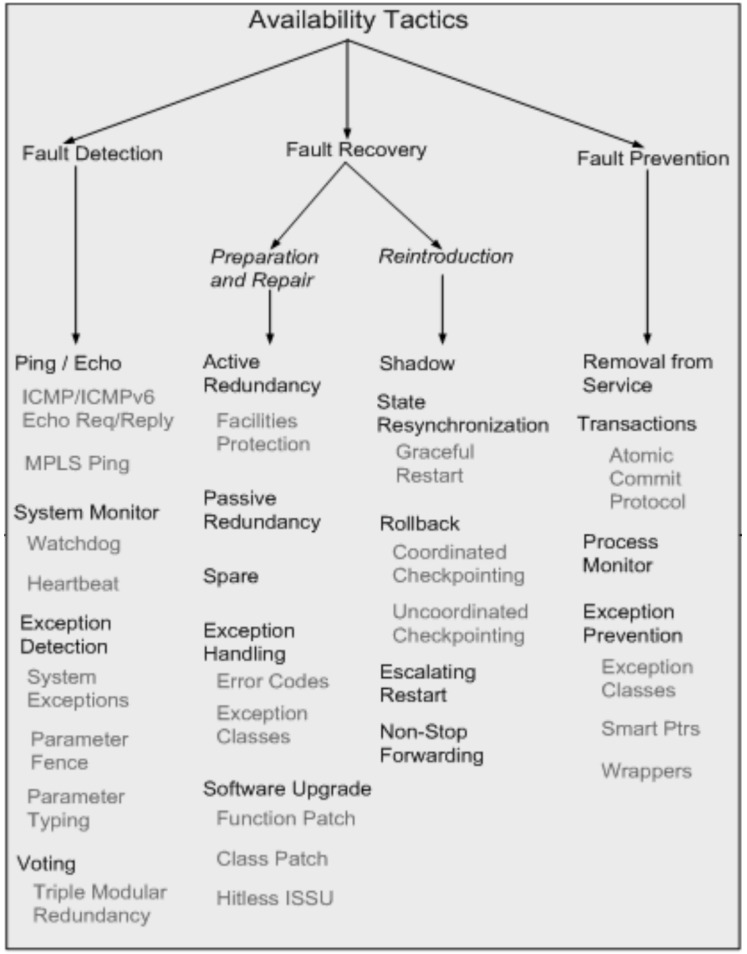
- An *architectural tactic* is a design decision that influences the achievement of a quality attribute response.
- Tactics are focussed on single QAs and don't try to take tradeoffs into account.
- Tactics:
  - Gather the options available to the architect
  - Are more primitive than patterns.
  - Systematically enumerate possible design decisions

Example Tactics

- Tactics are often generic and need to be specialised to a specific context. Example tactics:
  - *Schedule resources:* this is common in refining performance or responsiveness but the particular approach to scheduling will depend on the context.
  - *Use an intermediary:* this is common in modifiablility (e.g. an intermediary might be a layer to control access to some functionality) but the choice of intermediary will depend on the context
  - *Manage sampling rate:* this is already quite specific and will usially apply to real-time systems but it may not be applicable to all real-time systems.

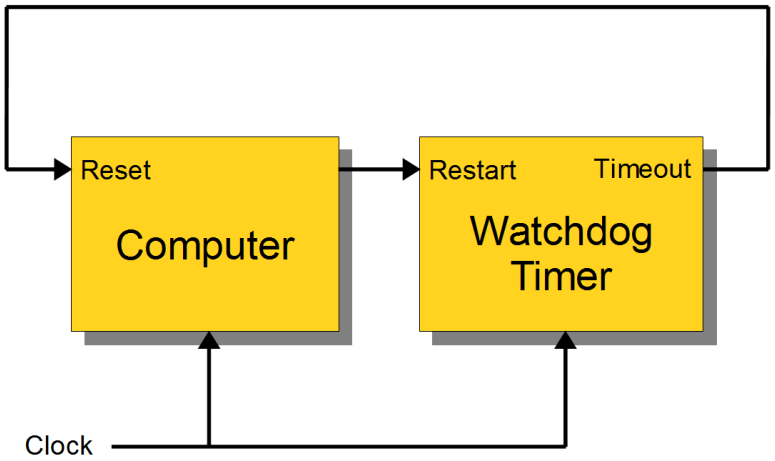
Availability Tactics

- For an example QA and its tactics read: http://www.sei.cmu.edu/reports/09tr006.pdf
- The tactics identify different sorts of responsibilities and how to achieve them



Example Availability Architectures

- Watchdog times ensure the availability of systems by initiating a restart in the event of apparent inactivity.
- Watchdog timers are used extensively by the Jet Propulsion Laboratory in space mission development. Read the following slideset to get an idea of the issues: https://indico.esa.int/indico/event/62/contribution/16/material/1/0.pdf



Heartbeat

- Used in high availability clusters.
- Used to provide availability of a shared resource.
- One system “owns” the resource and regularly sends out a heartbeat message indicating the system is alive and in charge of the resource.
- If the heartbeat is missed a backup system takes over and takes over responsibility for the shared resource.
- This is important where the resource can be updated and we want to ensure only one system is the gatekeeper for updating

Categories of Architectural Design Decisions

- There are seven broad categories of design decision:
  1. Allocation of responsibilities
  2. Coordination model
  3. Data Model
  4. Management of resources
  5. Mapping among architectural elements
  6. Binding time decisions
  7. Choice of technology

Allocation of Responsibilities

- Identify the important responsibilities
- Determining how to allocate these to runtime and static elements.
- These are often specific to a QA and so the specifics depend on the QA, for example in the case of availability *fault detection* is an important responsibility that will be further decomposed and distributed in the architecture.

Coordination Model

- Components in the architecture interact with one another via a collection of mechanisms. This is called the coordination model.
- We need to decide on:
  - What elements in the system need to coordinate with one another.
  - What properties the coordination needs to have (e.g. timing properties, security of coordination, …)
  - Choosing the mechanisms (ideally a small number) and these have properties like statefulness, synchrony, delivery guarantees, performance properties.

Data Model

- Choosing abstractions, operations, and properties. How data is created and destroyed, access methods, ...
- Maintaining metadata that controls the interpretation of the data.
- Organising the data, what kind of system will be used to store it, how will it be backed up, how do we recover from data loss

Management of Resources

- Resources can be hard (CPU, memory, battery, …) or soft buffers, processes, …)
- Resource Management includes:
  - Identifying resources to be managed
  - What system element should manage a resource
  - Work out sharing strategies and how to arbitrate in contention situations
  - Consider the consequences of running out of a resource (e.g. Memory).

Mapping among Architectural Elements

- There are two important types of mapping:
  - Mapping between different types of elements in the architecture e.g. from static development structures (modules) to execution elements e.g. threads or processes.
  - Mappings between software elements and environment elements e.g. from processes to specific processors and other hardware.
- The main appings are code to runtime structure; runtime elements to environment; data model elements to data stores

Binding time decisions

- This looks at when you might want to make a decision between a predetermined range of options.
- This can range from design time when a designer might allocate a responsibility through to runtime when an end user might allocate a responsibility.
- Often these are about allowing variability in some of the decisions we have already seen. E.g. we might want some variability in the resources to be managed determined at run time or we might make the coordination model negotiable at runtime if we want to interoperate with a range of systems.

Choice of Technology

- This is critical to being able to realize all the other decisions in a concrete system. We need to consider:
  - What technologies are available
  - What tools are available to support technologies
  - How much training will it take to be able to use a technology?
  - What are the full range of consequences of the choice of a technology (e.g. it may restrict markets because it is incompatible with some other technologies).
  - If the technology is new, how does it fit into the existing preferred technologies for the organisation.

Summary

- We looked at the architecture influence cycle to see the role of stakeholders.
- We then looked at QA Scenarios as a means to make QA requirements testable.
- We looked at architectural tactics as a way of documenting routes to achieve a QA requirement.
- We looked at availability as an example.
- Finally we considered the main areas of architectural design decisions.