SAPM Design Example
Availability

30 January 2017
Availability

• Some systems we need to be there whenever they need to be used.
• These are usually called high availability systems.
• There can be different reasons for high availability:
  – 999 (or 911 or ...) telephone system
  – Interplanetary spacecraft systems
  – Electricity supply grid
  – Large (and expensive) computer power supply
Availability

• From hardware there are two key measures:
  – mtbf – mean time between failures
  – mttr – mean time to repair

• The availability of a system is usually defined to be the probability it will be there when you ask it to work: \( \frac{mtbf}{mtbf+mttr} \)

• So there are two ways to make this number bigger (i.e. closer to 1): make the mean time between failures longer, make the system faster to repair
Faults, Errors, Failures

• A *fault* is something in the system (e.g. a broken wire, failed component, wrong bit of code, ...) that can cause the system to move into an *error* state when the fault is activated, an error may then eventually cause an externally observable deviation from the intended operation and this is called a *failure*.

• Most high availability systems try to tolerate or mask faults by detecting erroneous conditions before they move into failure conditions.
Faults, Errors, Failures, Example

• A *fault* in a sorting routine means that under some circumstances it fails to sort an array.
• Under there conditions, the system might be assuming an array is sorted but it isn’t. In this state there is an *error* in the system because things are not as they should be.
• If the system uses binary search to look for things in the array, sometimes an item will be in the array but will not be found – this might cause a visible *failure* of the system.
Generic Scenario

- **Source**: Internal or external sources important to differentiate because different measures are possible.
- **Stimulus**: Fault causes errors: omission (no result), crash (repeated omissions), timing (late, early), response (incorrect value).
- **Artifact**: Specifies what has to be available: process, channel, store, ...
- **Environment**: what the mode of operation is: normal, degraded, startup, shutdown, ...
- **Response**: how to respond to the stimulus
- **Response measure**: *this will be some measure related to the availability or the “liveness” of the artifact*
A Concrete Scenario

• In mission critical systems there is typically a schedule that activates a sequence of tasks in turn. These take longer or shorter times to complete and the whole set is carried out cyclically.

• What happens if there is a bug in a task and it never completes?

• So the concrete scenario might be as follows.
A Concrete Scenario

• Source: internal task process
• Stimulus: either omission or timing depending on how you look at it but the task does not respond as expected.
• Environment: normal operation
• Response: abandon task
• Response measure: system always responds within 200ms
Simple Approach

- Cycles through each of the tasks.
- Passes control to the task
- Waits for control to pass back.
- What happens if a task fails?
- This architecture fails the scenario
Availability Tactics

• Our current architecture fails the scenario, why?
• Because we can’t detect the error arising from the fault in a task.
• We look at the tactics to see how we might fix this.
What tactic might we take

• Use a watchdog:  
  https://en.wikipedia.org/wiki/Watchdog_timer

• This would enable a correct response in the absence of a response from a task.

• Does it solve all the problems?

• What else might be necessary?

• Look at the tactic list
Architectural Design Decisions

• We can specialise our 7 categories to consider availability:
  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

• Determine what needs to be high availability (maybe not all functions).
• Responsibility for detecting error (and possible cause).
• Responsibility to log errors
• Responsible respond to a detected error
• Manage sources of events
• Decide on mode of operation
• Decide on how to repair faults
• ...
• In our example we introduce the watchdog and give it responsibility for responding to error.
Coordination Model

• Are the error detection capabilities of the coordination model adequate to detect errors?
• Is the coordination model sufficient to ensure communication and coordination between error detection, log and response.?
• Will coordination work in the presence of error, degraded modes?
• If repair involve replacement of elements will the coordination model allow this?
• In our example the wakeup between watchdog and controller might be an addition to the coordination mechanism.
Data Model

• How do error conditions affect the data model?
• Does this mean we have to deal with some forms of corrupt data or incomplete operations?
• Perhaps the data model needs to be extended to include new operations to recover from failed earlier operations.
• For example, extending the model with checkpoint and rollback operations may be enough in some situations.
Management of Resources

• See what resources are essential to maintain operation in the presence of errors.
• Identify what resources are necessary for meaningful degraded modes.
• Work out if different scheduling changes the demand on critical resources.
• In our example if task 1 is in error because of a bad processor and task 4 is OK but not necessary for some degraded mode it may be best to switch task 1 and 4 and never schedule task 4 again to provide a degraded mode of operation.
Mapping between elements

• Determining what resources might be in error or might be affected by errors.
• Checking that remapping of elements is possible dynamically.
• How fast can elements be restarted or reinitialised, can a process be moved to a new processor, ...
• In our example it may be necessary to identify the watchdog as a new element and that a failing task may need to be mapped to a different processor.
Binding time decisions

• Look at binding time and see where this will allow flexibility.

• For example, if we can tolerate a 0.5s delay on a response but are currently using 0.1s as the time to signal an error then we might want to rebind and operate in a degraded mode.

• In our example, if the the taks code is burned into PROMS on the processors there is no chance to rebind task/processor.
Choice of technology

• Explore technologies that package useful functionality for availability.
• Use an established element if it is available.
• Use already established data on the availability characteristics of components.
Summary

• Availability is a good example QA.
• We looked at the definition.
• We saw how a scenario might capture a testable Availability requirement.
• We saw how an architecture might fail such a scenario.
• We say how tactics suggest ways to improve architectures.
• We also saw how the seven architecture design decisions map onto Availability as an example.