SAPM Meeting 4
Contexts for Software Architecture
Shaping Design

Stuart Anderson

Contexts for Software Architecture

- Software architects and architecture have arisen as systems have grown in: scale, economic importance and criticality.
- Architecture plays different roles in different contexts. The main contexts are:
  - Technical: where architecture supports technical activity like measurement, V&V, compliance,…
  - Project lifecycle: where architecture interacts with and supports development process
  - Business: where architecture supports organisations e.g customer organisations and development organisations.
  - Professional: where the role of architect defines requirements and constraints on architects.

Technical Context

- Architectures provide a means of controlling quality attributes of the system
  - In the context of design activities we try to choose architectures that enable the attributes we care about.
  - In the case of analysing existing systems as part of reengineering or evolving the system we may find that the architecture can inhibit particular attributes.
- Architectures don't often have much to say about functionality because architectures provide containers for functionality

Controlling Quality Attributes

- Usually we care about several quality attributes at once. For example:
  - The availability of a system where we worry about ensuring there is a system to take over if a system fails.
  - The safety of a system where we have to worry about ensuring that the system only behaves as intended and has no additonal behaviour.
  - The testability of a system where we are concerned about ensuring:
    - elements are clearly able to be isolated
    - we know what behaviour to expect of components of the system
    - we know how components relate to modules so we can track down faulty code
    - We know how components are intended to integrate to give the overall behaviour
- Other quality attributes are things like performance, usability, interoperability,..

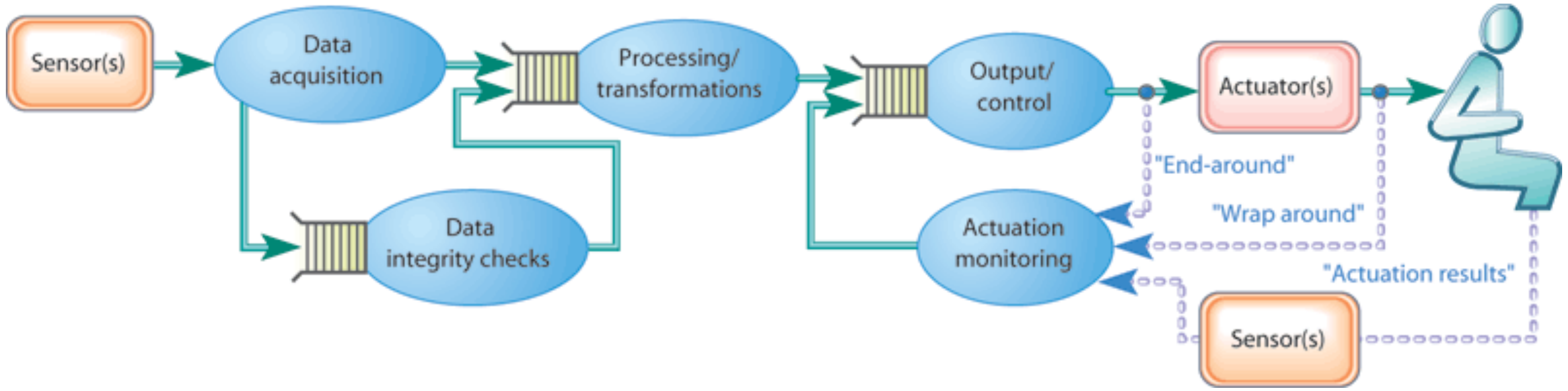
Safety Critical Example

- In the following diagrams:
  - orange blobs are sensors that sense the environment
  - Pink blobs are actuators that change the world around the system by exerting forces on it.
  - Sensors and actuators usually have software embedded in them.
  - blue blobs are components
  - the person icon is the system under control
- Safety systems are constructed as channels that read information from sensors and tell actuators what to do.

Testability Concerns

- In building a safety system we might worry about two aspects of actuators:
  - Actuators are mechanical things so they always fail in the end because they wear out.
  - How to test that that the actuators behave as expected by the safety system.
- To handle this we might us an actuator monitoring architecture.

Actuator Monitoring



Actuator Monitoring

- Here we include an actuator monitoring component that compares three things:
  - What the actuators are told do by the system
  - What the actuators do in response
  - How the system responds (this may require additional sensors to detect the response)
- The actuator monitoring will cause the system to fail if it detects an anomaly
- Notice we have said nothing about functionality here

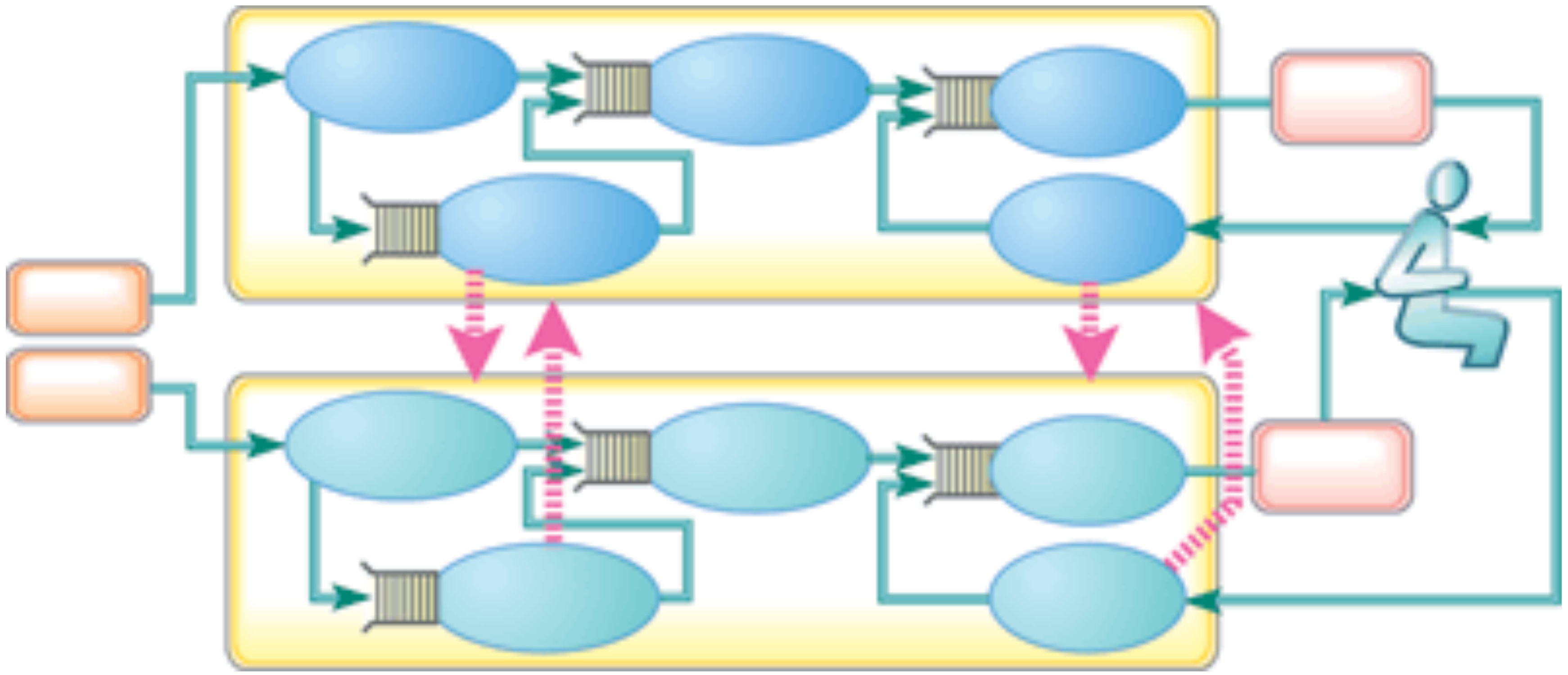
Actuator Monitoring

- At the architectural level:
  - We don't consider the functionality of the monitor or what sensors we might need.
  - We might set a target in terms of how often we are prepared to tolerate an undiagnosed sensor failure (because this is a quality attribute).
  - These sorts of decisions interact with costs (because adding extra sensors has costs)
  - Sensors also fail so we may need to take that into account

Availability

- After looking at the actuator monitoring issue we might worry that the system will fail too often and will take too long to repair.
- The standard approach to this is to use a dual channel architecture where two channels operate concurrently one is in control the other a backup and we switch to the backup if the first system fails.
- This is the architectural approach to managing availability issues. Here there are two copies of our earlier architecture.

Availability



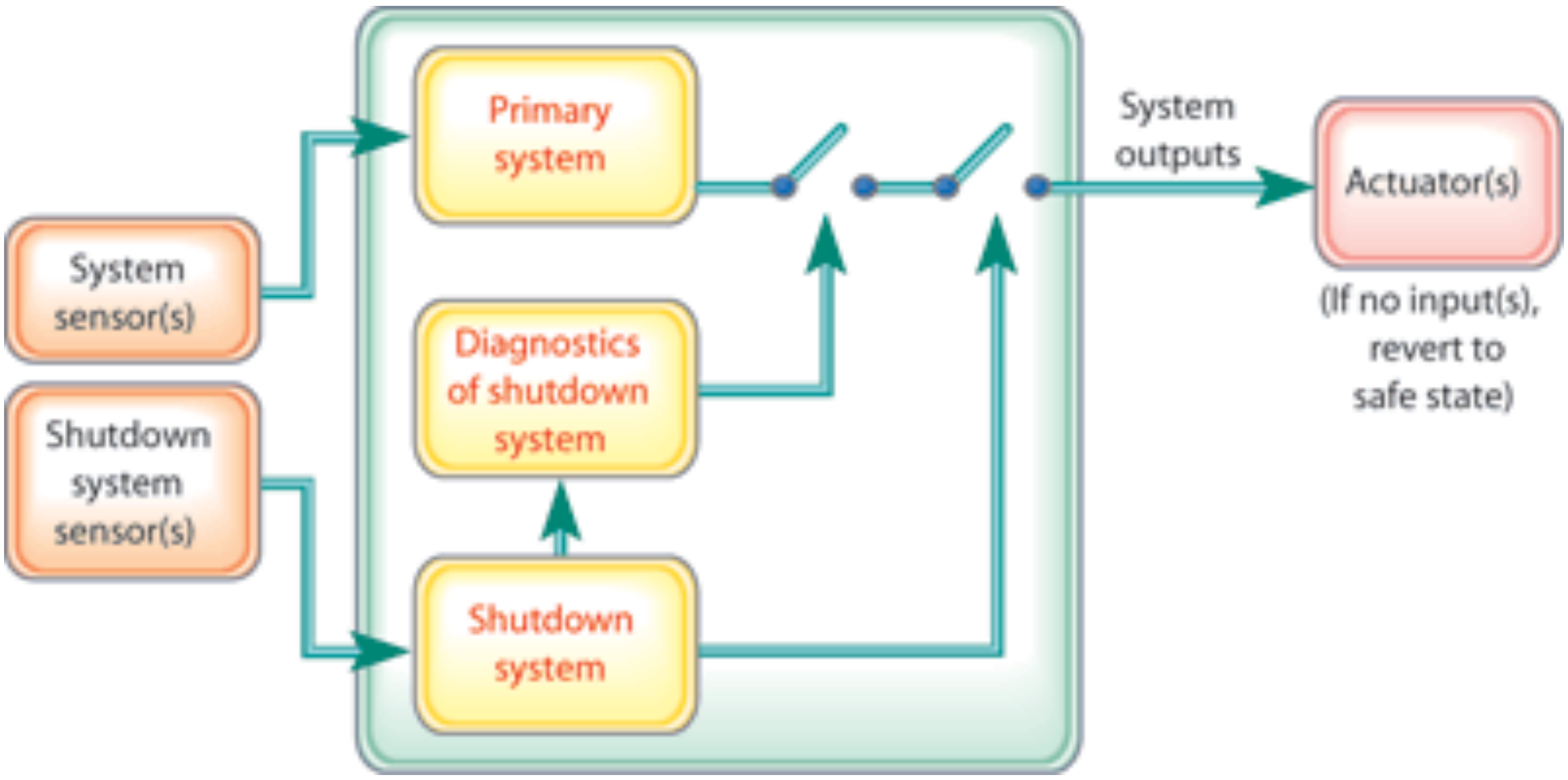
Availability

- We might want to assess how much benefit we would gain from this approach.
- At a detailed design level we might want to look at features in the functionality that cause coordinated failure in both systems
- At the architectural level we might only want to know how correlated failures are between two copies of the system. This would let us assess the impact of the architectural change.

Safety

- Looking at our dual channel, actuator monitored system it might still be the case that the consequences of failure may be dangerous.
- If the system has a “safe state” that we can set the actuators to achieve we might want to use a shutdown system to achieve this.
- See the shutdown architecture to achieve this

Safety


System sensor(s)
Shutdown system sensor(s)
Primary system
Diagnostics of shutdown system
Shutdown system
System outputs
Actuator(s)
(If no input(s), revert to safe state)

http://www.embedded.com/design/prototyping-and-development/4006464/Architecture-of-safety-critical-systems

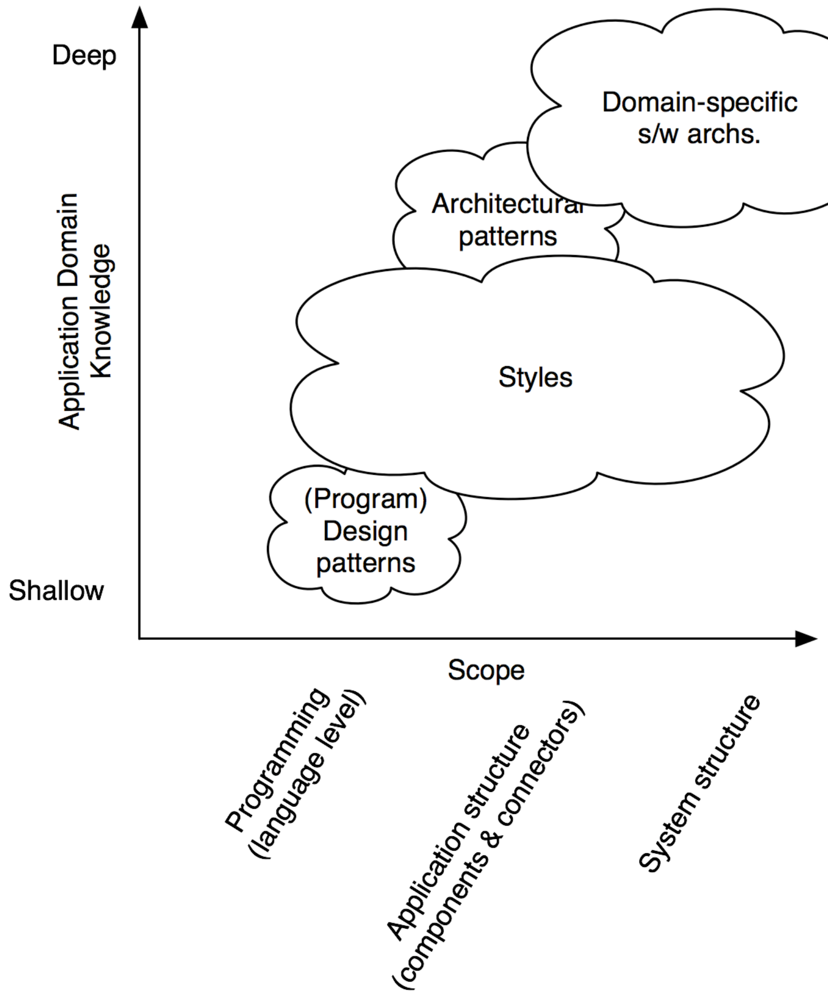
Safety

- Now our dual channel system is the “primary system” this might be quite complex because it attempts to achieve good control over the system.
- If the primary system fails the system under control may become dangerous.
- The shutdown system is very simple and just detects when the system is outside normal operation then it shuts the system down
- Shutdown is achieved by disconnecting from the actuators and they go into safe mode that brings the system under control to its safe state.
- For additional safety we also monitor the shutdown component to detect failure.

Design in the Technical Context

- Design is a mixture of creativity and the use of knowledge that is institutionalised in the context.
- This takes the form of reusable structures (at varying levels of completeness) that can be reused.
- These reusable structures also influence other aspects of context, helping to shape processes, organisations and professions.
- We can plot different sorts of architectural structure depending on the degree to which it is specific to a domain and the extent to which it influences the system.

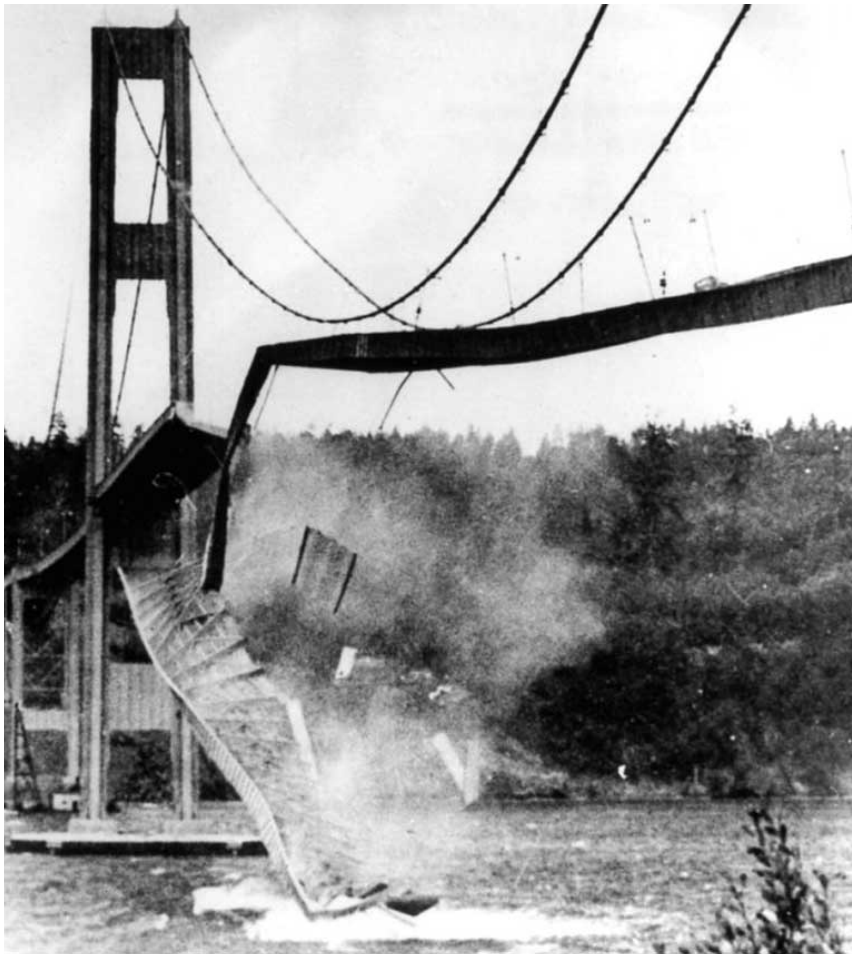
Patterns, Styles, Domain Specific Architecture


Deep
Shallow
Application Domain Knowledge
Domain-specific s/w archs.
Architectural patterns
Styles
(Program) Design patterns
Scope
Programming (language level)
Application structure (components & connectors)
System structure

Domain-Specific Software Architectures

- A DSSA is collection of (pre-decided) design decisions that
  - Capture important aspects of particular tasks (domain),
  - Are common across a range of systems in the domain
  - Typically they will have some predefined structures depending on the attributes we want to control and a disciplined way of extending such structures.
- These are not general purpose because they incorporate many specific characteristics of the domain.
- The main benefit is the extent to which such domain-specific architectures capture design knowledge acquired by the community (but we do forget things – see over…).

Tacoma Narrows Bridge



Just four months after Galloping Gertie failed, a professor of civil engineering at Columbia University, J. K. Finch, published an article in Engineering News-Record that summarized over a century of suspension bridge failures. In the article, titled "Wind Failures of Suspension Bridges or Evolution and Decay of the Stiffening Truss," Finch reminded engineers of some important history, as he reviewed the record of spans that had suffered from aerodynamic instability. Finch declared, "These long-forgotten difficulties with early suspension bridges, clearly show that while to modern engineers, the gyrations of the Tacoma bridge constituted something entirely new and strange, they were not new--they had simply been forgotten."

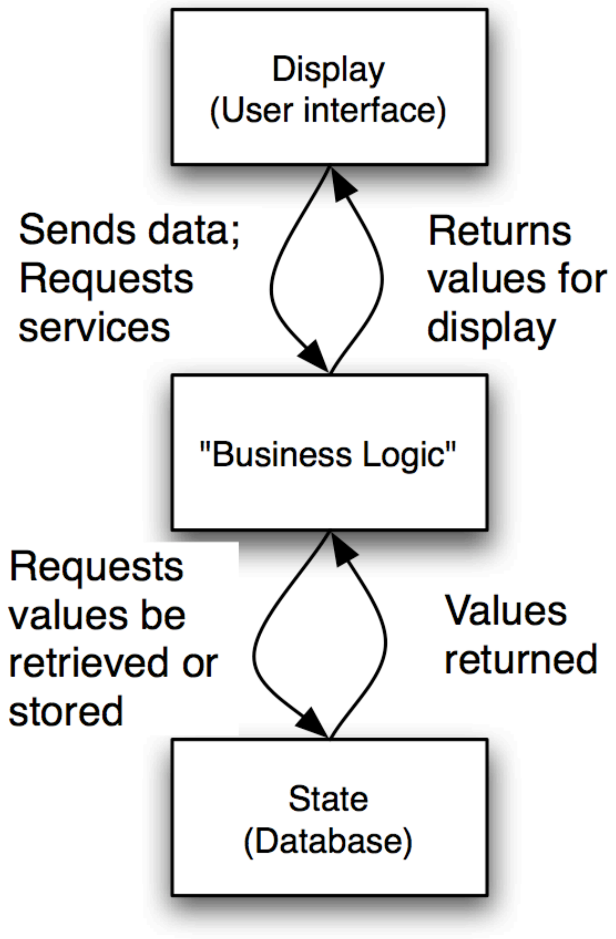
An entire generation of suspension bridge designer-engineers forgot the lessons of the 19th century. The last major suspension bridge failure had happened five decades earlier, when the Niagara-Clifton Bridge fell in 1889.

Architectural Patterns

- An architectural pattern is a set of architectural design decisions that are applicable to a recurring design problem, and parameterized to account for different software development contexts in which that problem appears.
- Architectural patterns are similar to DSSAs but capture less of the behaviour and attributes of the system but are more general because they are intended to abstract a common pattern over several domains.

State-Logic-Display: Three-Tiered Pattern

- Application Examples
  - Business applications
  - Multi-player games
  - Web-based applications

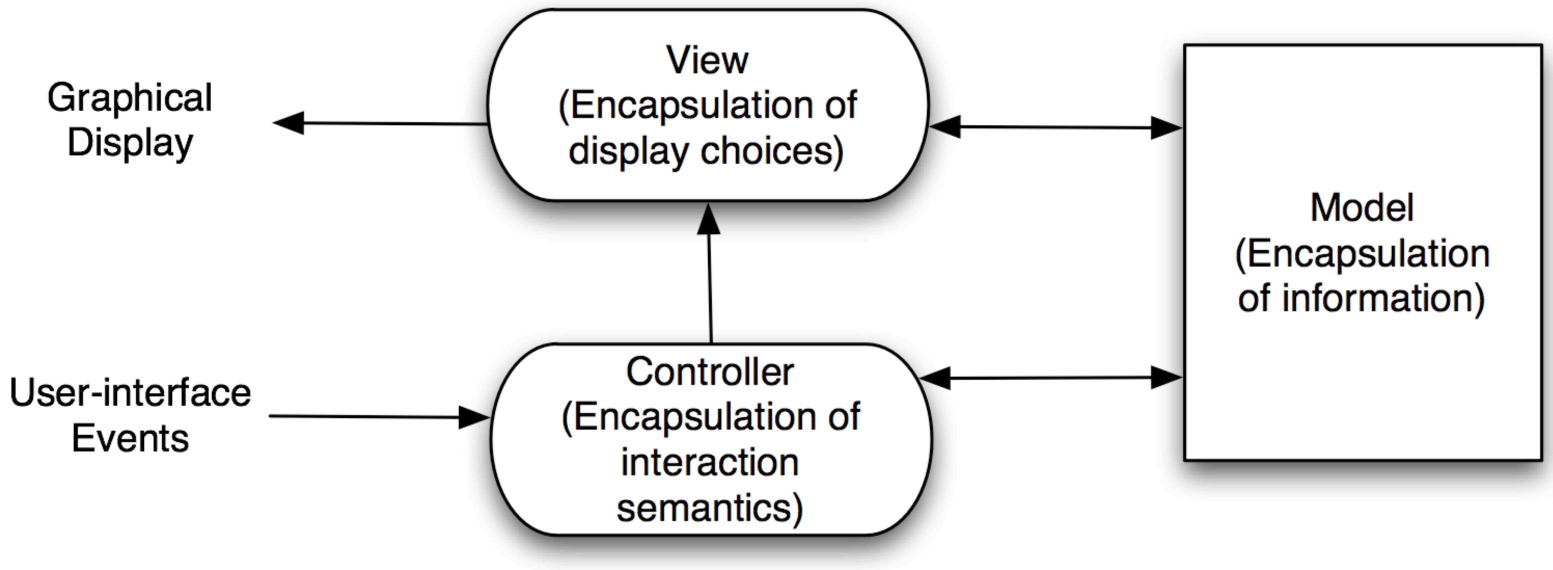


Display
(User interface)
Sends data; Requests services
Returns values for display
"Business Logic"
Requests values be retrieved or stored
Values returned
State
(Database)

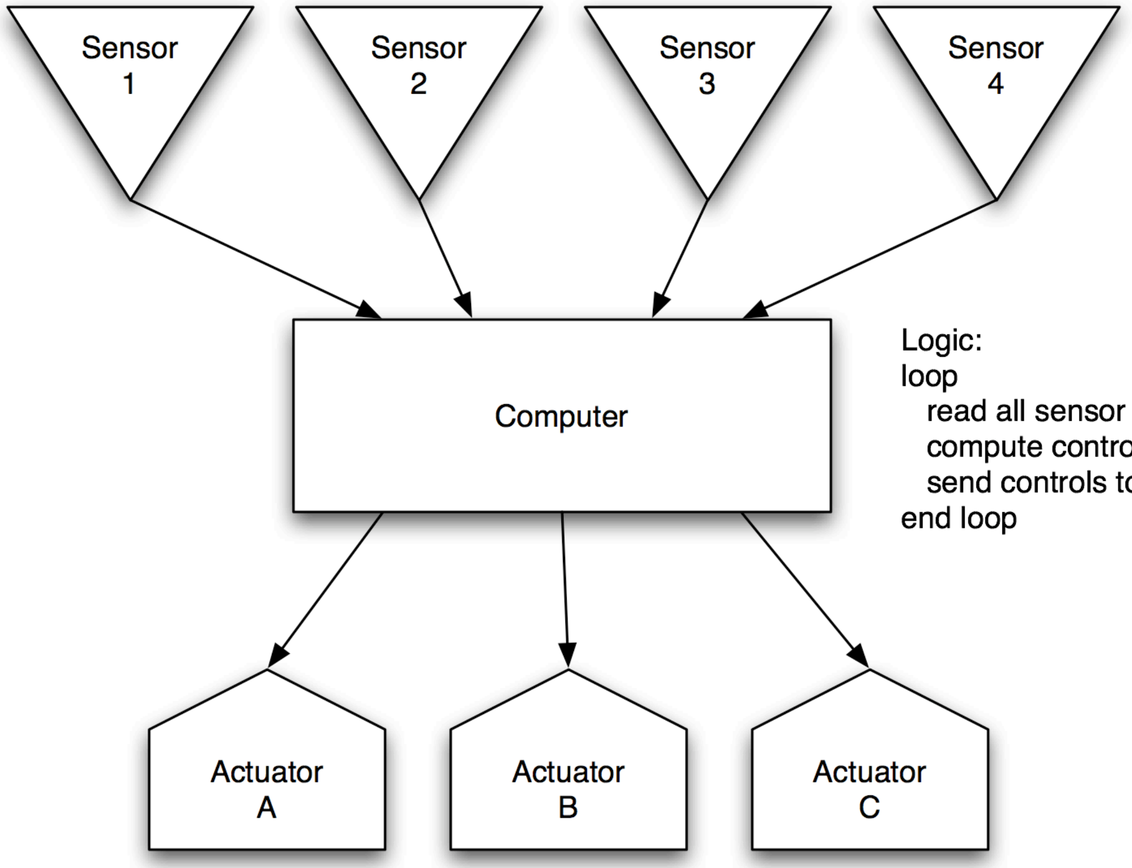
Model-View-Controller (MVC)

- Objective: Separation between information, presentation and user interaction.
- When a model object value changes, a notification is sent to the view and to the controller.
  - Thus, the view can update itself and the controller can modify the view if its logic so requires.
- When handling input from the user the windowing system sends the user event to the controller.
  - If a change is required, the controller updates the model object.

Model-View-Controller


Graphical Display
View (Encapsulation of display choices)
Model (Encapsulation of information)
User-interface Events
Controller (Encapsulation of interaction semantics)

Sense-Compute-Control



```
Logic:
loop
  read all sensor values
  compute control outputs
  send controls to all actuators
end loop
```

Objective: Structuring embedded control applications

Software Architecture: Foundations, Theory, and Practice; Richard N. Taylor, Nenad Medvidovic, and Eric M. Dashofy; © 2008 John Wiley & Sons, Inc. Reprinted with permission.

Contexts Shape Design

- The technical context identifies features we want to control and packages a range of other properties.
- Standard architecture (patterns, domain specific architectures etc.) package these. The other context we consider also help to shape the choice of architecture.
- Design uses pre-decided structures and then alters and extends structure as necessary.

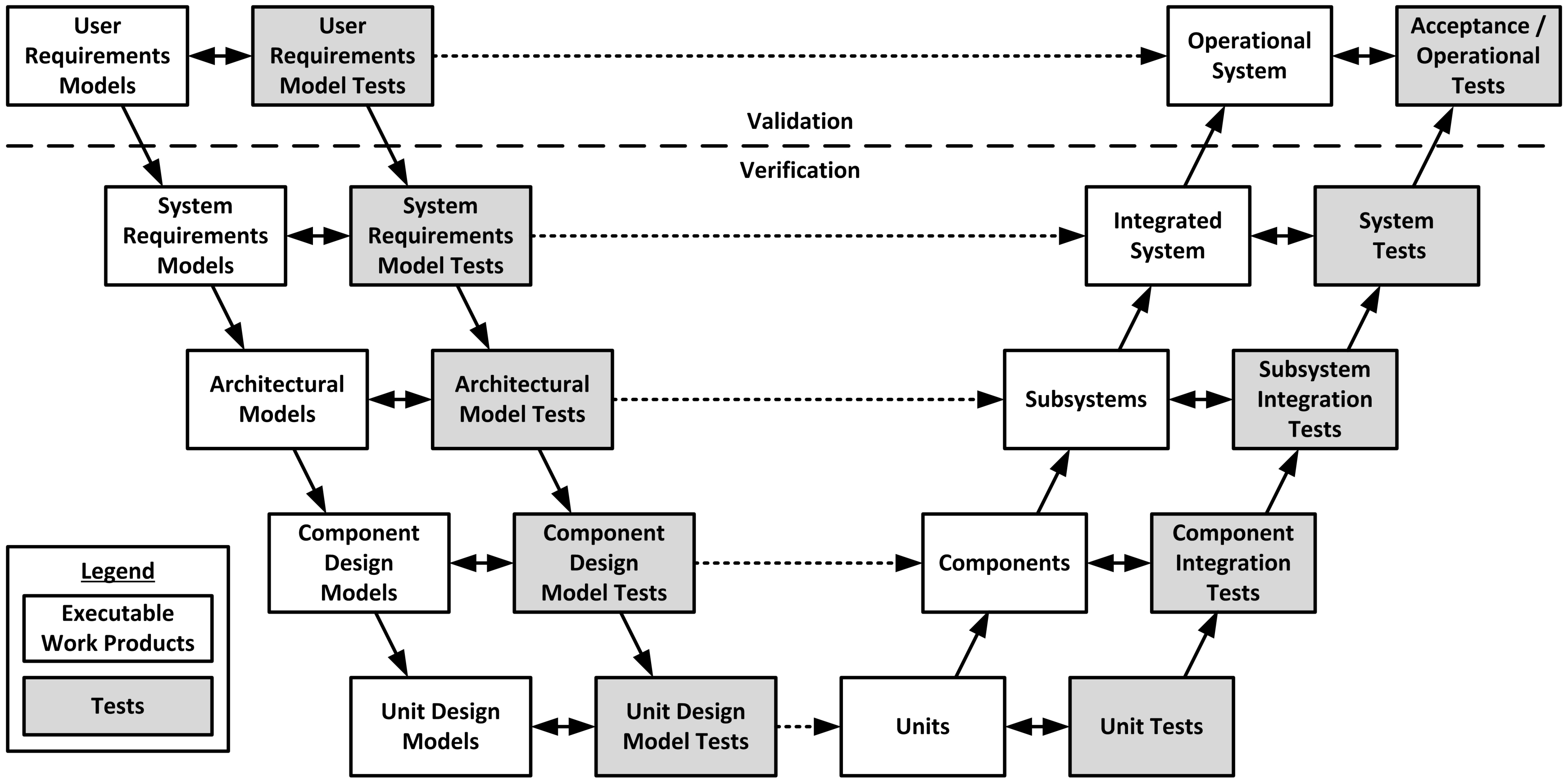
Project Lifecycle Context

- We will consider this in more detail later in the course. We know that whatever lifecycle we use we will need to do the following (and these are all done best by talking about architecture:
  - Make a business case for the system
  - Understanding requirements that concern quality attributes
  - Deciding on architecture
  - Documenting architecture
  - Analysing and evaluating architecture
  - Implementing and testing the system based on architecture
  - Ensuring the implementation conforms to the architecture
- These take place in all lifecyles but in different places

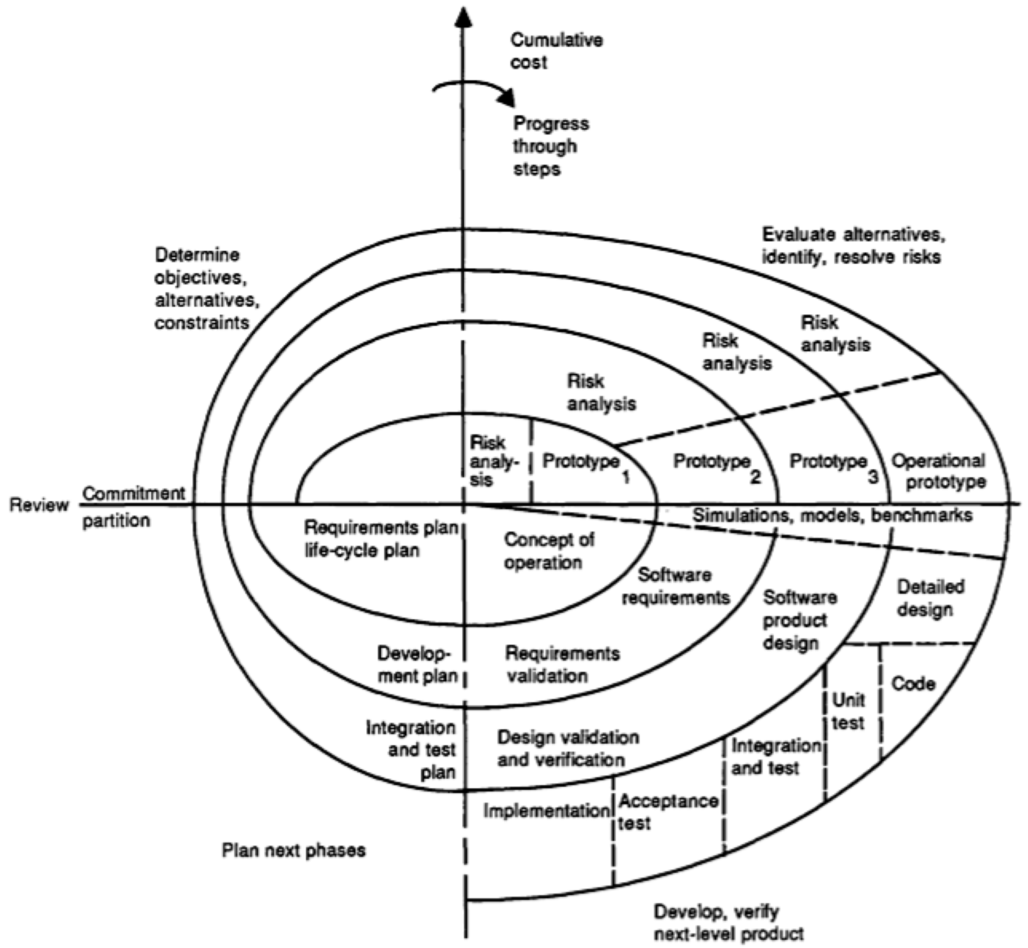
Lifecycle Models

- We will consider three models (or classes of models in future lectures):
  - The V-model which is a development of waterfall and explicitly includes architectural design as a stage.
  - Iterative models as exemplified by Boehm's spiral model that focusses on project risk management. Here architecture that gives us vision of how development tasks are interrrelated will play a key role as well as ensuring quality requirements are met.
  - Agile – here some decisions are more stable than others and the more stable ones concern architectures. We will return to this later in the lecture series.

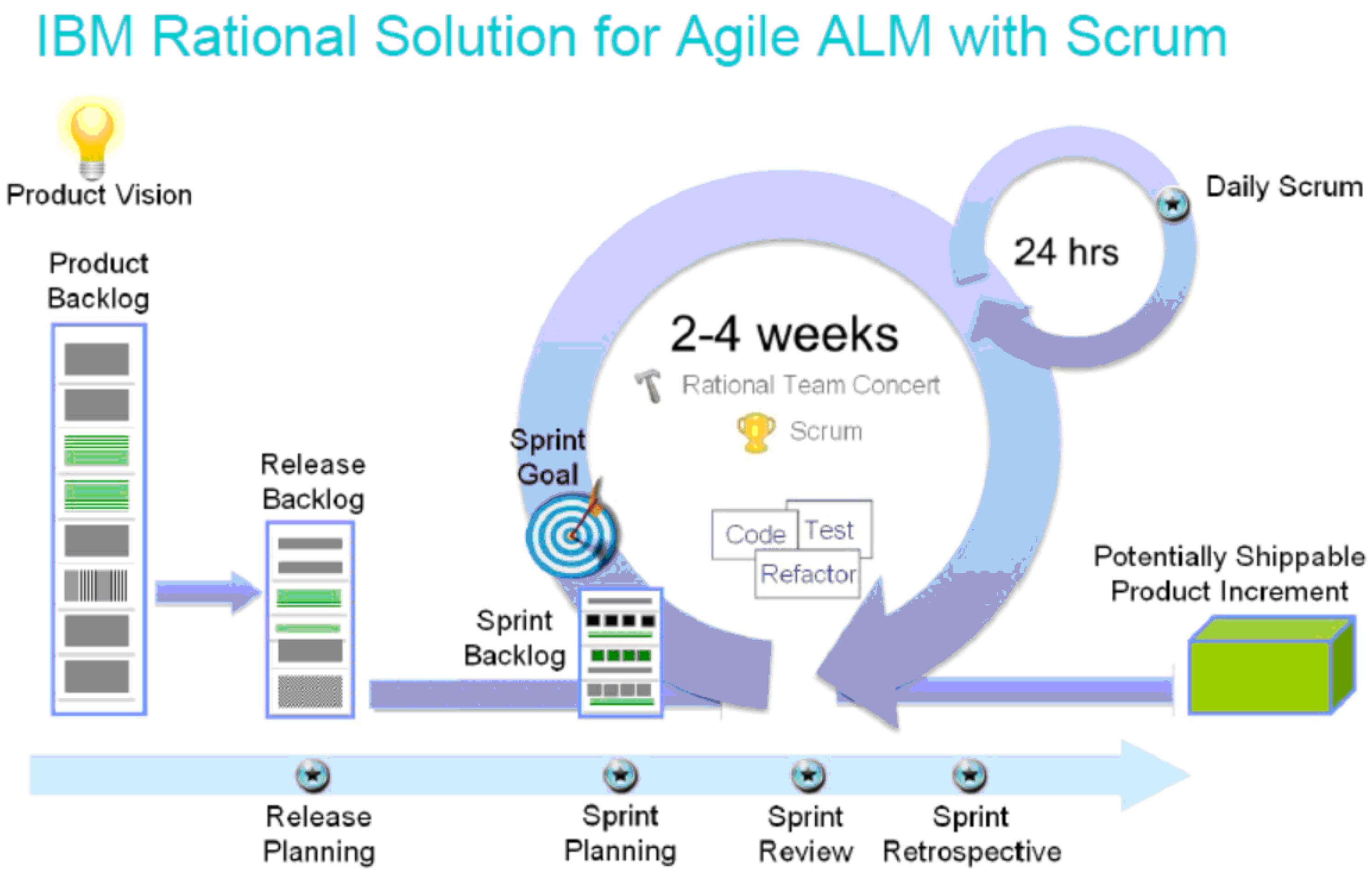
V-model


User Requirements Models
User Requirements Model Tests
Operational System
Acceptance / Operational Tests
Validation
Verification
System Requirements Models
System Requirements Model Tests
Integrated System
System Tests
Architectural Models
Architectural Model Tests
Subsystems
Subsystem Integration Tests
Component Design Models
Component Design Model Tests
Components
Component Integration Tests
Unit Design Models
Unit Design Model Tests
Units
Unit Tests
Legend
Executable Work Products
Tests

Spiral Model



Agile


IBM Rational Solution for Agile ALM with Scrum
Product Vision
Product Backlog
Release Backlog
Sprint Goal
Sprint Backlog
2-4 weeks
Rational Team Concert
Scrum
Code
Test
Refactor
24 hrs
Daily Scrum
Potentially Shippable Product Increment
Release Planning
Sprint Planning
Sprint Review
Sprint Retrospective

Business context

- Here we will consider two aspects in later lectures:
  - How the organisational structure of stakeholders can drive architectural decisions and shapes decision taking around architecture.
  - How architectural expertise drives the structure of development organisations in terms of their functional units and interrelationships.
- This is considered in depth in the SEI report: “*Relating Business Goals to Architecturally Significant Requirements for Software Systems*” linked to from the SAPM wiki.

Professional Context

- The architectural perspective gives you as a professional:
  - A way of describing your expertise.
  - Your skills as an architect will be recognised within organisations you work with.
  - You can use architecture as a way of describing your past experience.
  - You can specialise in particulr classes of architecture e.g. financial architecture.

Summary

- We identified the four main contexts for architecture: technical, lifecycle, business, and professional.
- Architecture plays a different role in each of these contexts.
- We will expand on this in later lectures.