

Chapter 11

■ Requirements Modeling: Behavior, Patterns, and WebApps

Slide Set to accompany
Software Engineering: A Practitioner's Approach
by Roger S. Pressman

Slides copyright © 1996, 2001, 2005, 2009 by Roger S. Pressman

For non-profit educational use only

May be reproduced ONLY for student use at the university level when used in conjunction with *Software Engineering: A Practitioner's Approach*. Any other reproduction or use is prohibited without the express written permission of the author.

All copyright information MUST appear if these slides are posted on a website for student use.

Table of Contents

11.1 Creating a Behavioral Model

11.2 Identifying Events with the Use Case

11.3 State Representations

11.4 Patterns for Requirements Modeling

11.1 Requirements Modeling Strategies

■ Structured analysis

■ considers

- **Data objects** are modeled in a way that defines their attributes and relationships.
- **Processes** that manipulate data objects are modeled to show how they transform data as data objects flow through the system.



■ Object-oriented analysis

■ focuses on

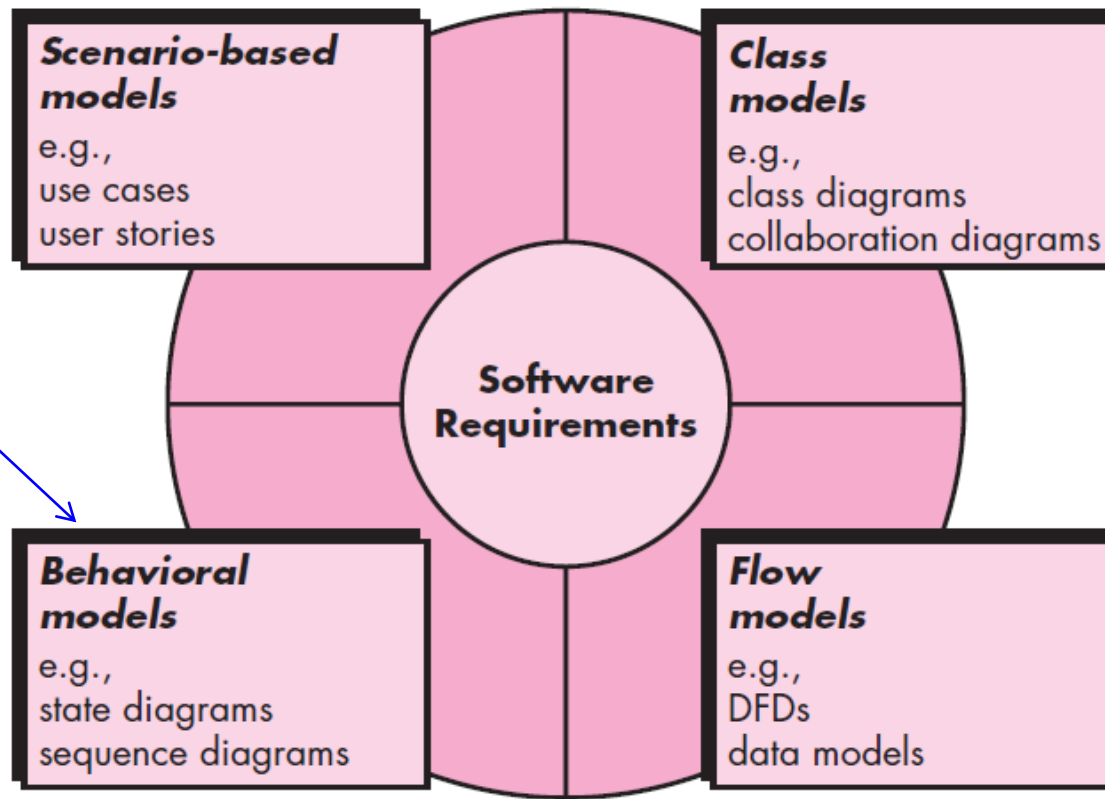
- the definition of **classes**
- the manner in which they **collaborate** with one another to effect customer requirements.

Elements of the Analysis Model

Ch. 9

Ch. 10

Ch. 11



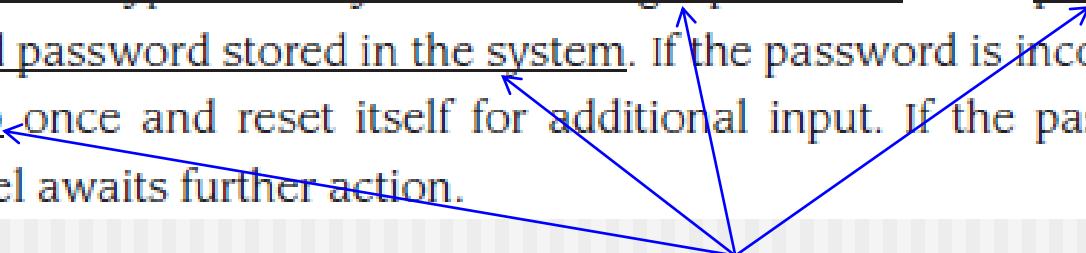
11.1 Creating a Behavioral Model

- The behavioral model indicates **how software will respond to external events or stimuli**.
- To create the model, perform :
 1. Evaluate all use-cases to fully **understand** the sequence of interaction within the system.
 2. Identify **events** that drive the interaction sequence and understand how these events relate to specific objects.
 3. Create a **sequence** for each use-case.
 4. Build a **state diagram** for the system.
 5. **Review** the behavioral model to verify accuracy and consistency.

11.2 Identifying Events with the Use Case

- An *event* changes the state of a system and/or triggers a sequence of actions.

The homeowner uses the keypad to key in a four-digit password. The password is compared with the valid password stored in the system. If the password is incorrect, the control panel will beep once and reset itself for additional input. If the password is correct, the control panel awaits further action.



events

- An actor should be identified for each event.
- The information exchanged should be noted.
- Any conditions or constraints should be listed.

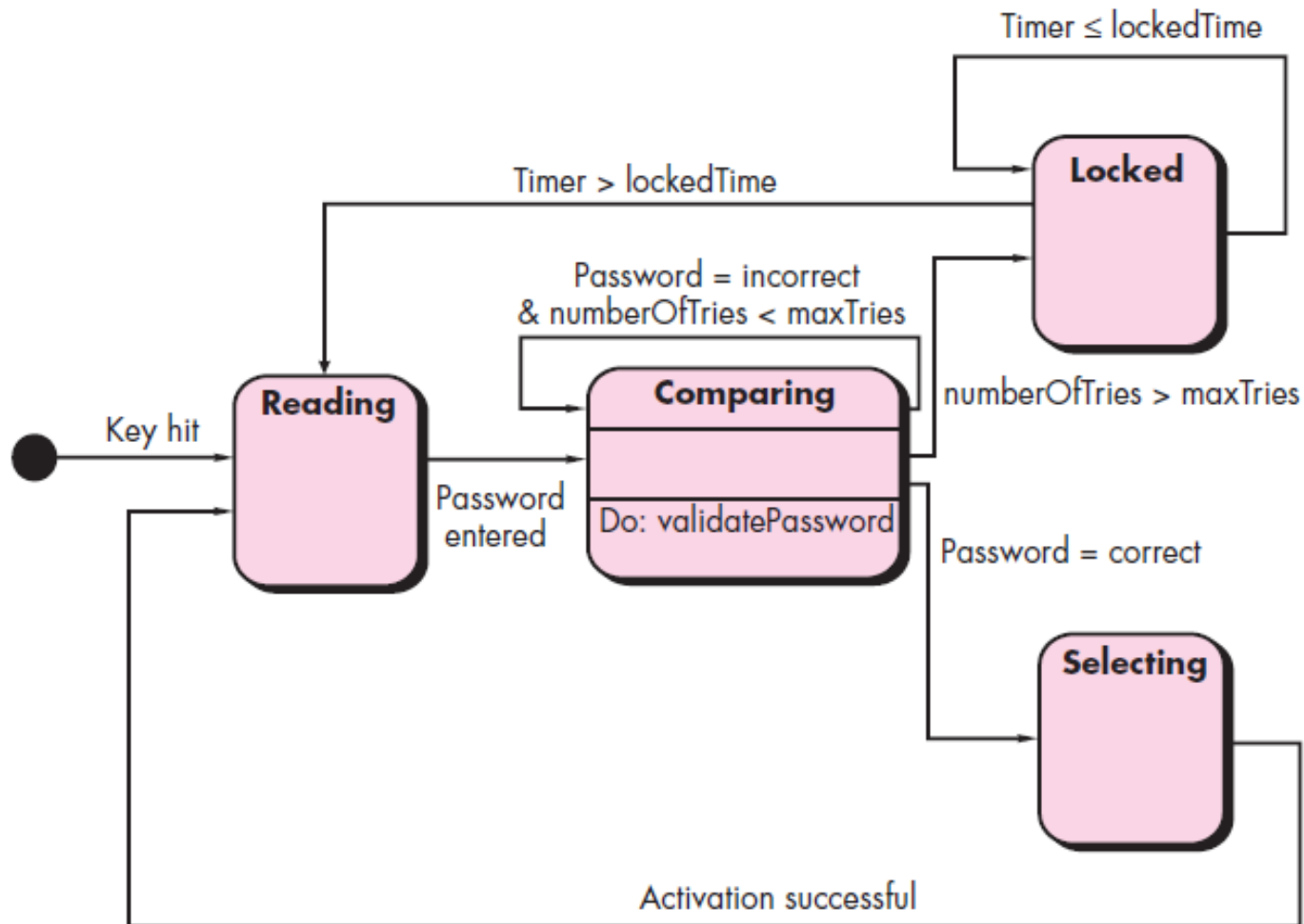
11.3 State Representations

- In the context of behavioral modeling, two different characterizations of states must be considered:
 - the state of each class as the system performs its function
 - the state of the system as observed from the outside as the system performs its function
- The state of a class takes on both passive and active characteristics [CHA93].
 - *passive state* : the current status of all of an object's attributes.
 - *active state* : the current status of the object as it undergoes a continuing transformation or processing.

Behavioral Modeling

- make a list of the different states of a system (How does the system behave?)
- indicate how the system makes a transition from one state to another (How does the system change state?)
 - indicate event
 - indicate action
- draw a **state diagram or a sequence diagram**

State Diagram for the Control Panel Class

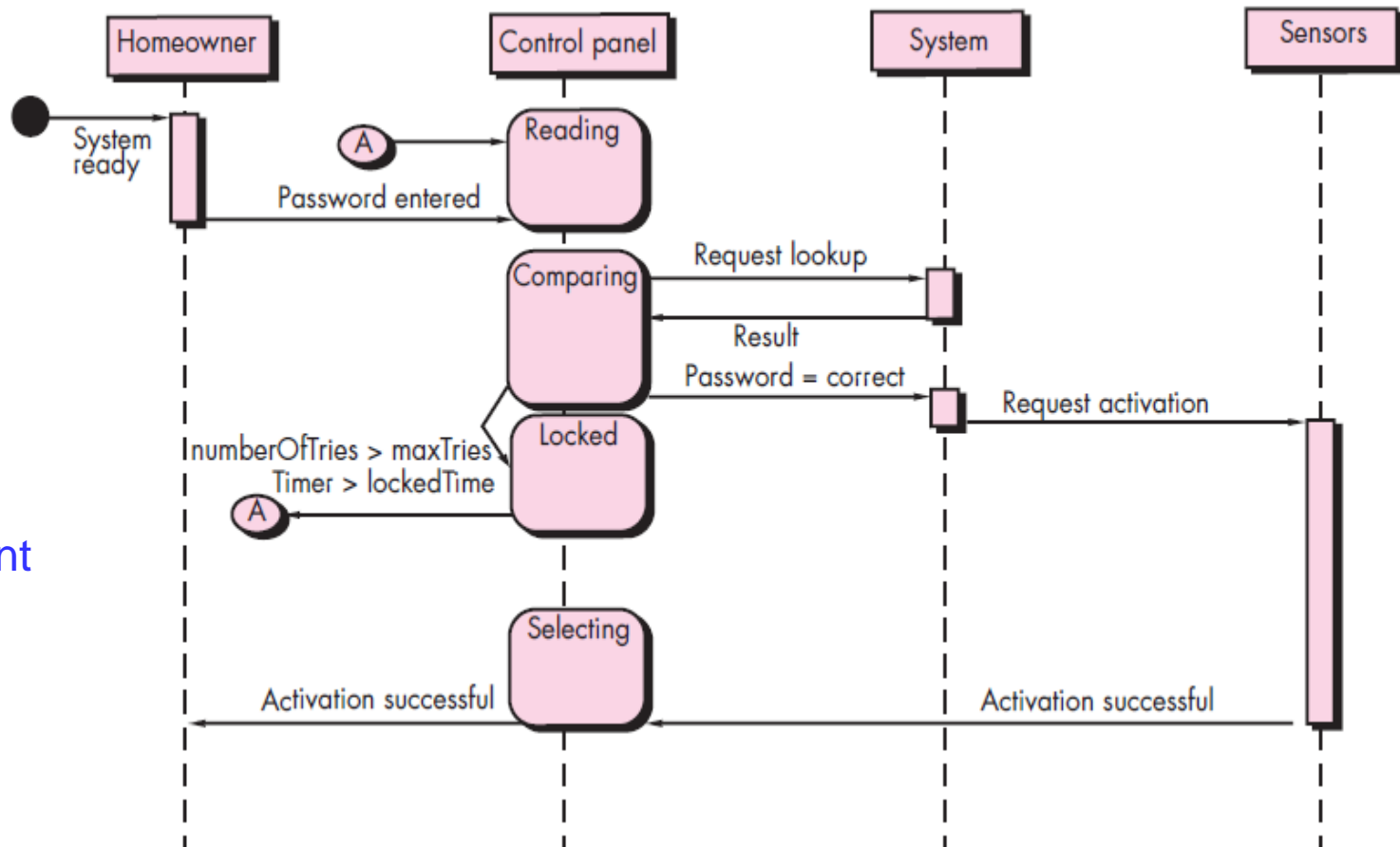


Active
state
model
uses only
these
elements.

- **state**—a set of observable circumstances that characterizes the behavior of a system at a given time
- **state transition**—the movement from one state to another
- **event**—an occurrence that causes the system to exhibit some predictable form of behavior
- **action**—process that occurs as a consequence of making a transition
- **guard** - a Boolean condition that must be satisfied in order for the transition to occur

Sequence Diagram

FIGURE 7.7 Sequence diagram (partial) for the *SafeHome* security function



Arrows
represent
events.

11.4 Patterns for Requirements Modeling

- Analysis patterns:
 - **mechanism for capturing domain knowledge** in a way that allows it to be reapplied when a new problem is encountered
 - domain knowledge can be **applied** to a new problem within the same application domain
 - the domain knowledge captured by a pattern can be applied **by analogy** to a completely different application domain.
- The original author of an analysis pattern does not “create” the pattern, but rather, **discovers** it as requirements engineering work is being conducted.
- Once the pattern has been discovered, it is **documented and applied**.

11.4.1 Discovering Analysis Patterns (To Apply)

- The most basic element in the description of a requirements model is the **use case**.
- *Semantic analysis pattern (SAP)*
 - “describes a small set of coherent use cases that together describe a basic generic application.” [Fer00]
- Can be considered a **technique** to find analysis patterns to apply

-
- Use case to control and monitor a real-view camera and proximity sensor for an automobile:

Use case: *Monitor reverse motion*

Description: When the vehicle is placed in *reverse* gear, the control software enables a video feed from a rear-placed video camera to the dashboard display. The control software superimposes a variety of distance and orientation lines on the dashboard display so that the vehicle operator can maintain orientation as the vehicle moves in reverse. The control software also monitors a proximity sensor to determine whether an object is inside 10 feet of the rear of the vehicle. It will automatically brake the vehicle if the proximity sensor indicates an object within 3 feet of the rear of the vehicle.

-
- This use case implies a variety of functionality that would be refined and elaborated (into a coherent set of use cases).
 - Regardless of how much elaboration is accomplished, the use case(s) suggest(s) **a simple, yet widely applicable Analysis Pattern** —
i.e. *the software-based monitoring and control of sensors and actuators in a physical system.*
 - “sensors” provide information about proximity and video information
 - “actuator” is the breaking system of the vehicle (invoked if an object is very close to the vehicle).
 - But in a more general case, a widely applicable pattern is discovered
--> **Actuator-Sensor**

11.4.2 A Requirement Pattern Example: Actuator-Sensor

Pattern Name: *Actuator-Sensor*

Compare with
Sensor-Control-Actuator Arch Pattern

Intent: Specify various kinds of sensors and actuators in an embedded system.

Motivation: Embedded systems usually have various kinds of sensors and actuators. These sensors and actuators are all either directly or indirectly connected to a control unit. Although many of the sensors and actuators look quite different, their behavior is similar enough to structure them into a pattern. The pattern shows how to specify the sensors and actuators for a system, including attributes and operations. The *Actuator-Sensor* pattern uses a *pull* mechanism (explicit request for information) for **PassiveSensors** and a *push* mechanism (broadcast of information) for the **ActiveSensors**.

Constraints:

Each passive sensor must have some method to read sensor input and attributes that represent the sensor value.

Each active sensor must have capabilities to broadcast update messages when its value changes.

Each active sensor should send a *life tick*, a status message issued within a specified time frame, to detect malfunctions.

Each actuator must have some method to invoke the appropriate response determined by the **ComputingComponent**.

Each sensor and actuator should have a function implemented to check its own operation state.

Each sensor and actuator should be able to test the validity of the values received or sent and set its operation state if the values are outside of the specifications.

Actuator-Sensor Pattern

Applicability:

Useful in any system in which multiple sensors and actuators are present.

Structure:

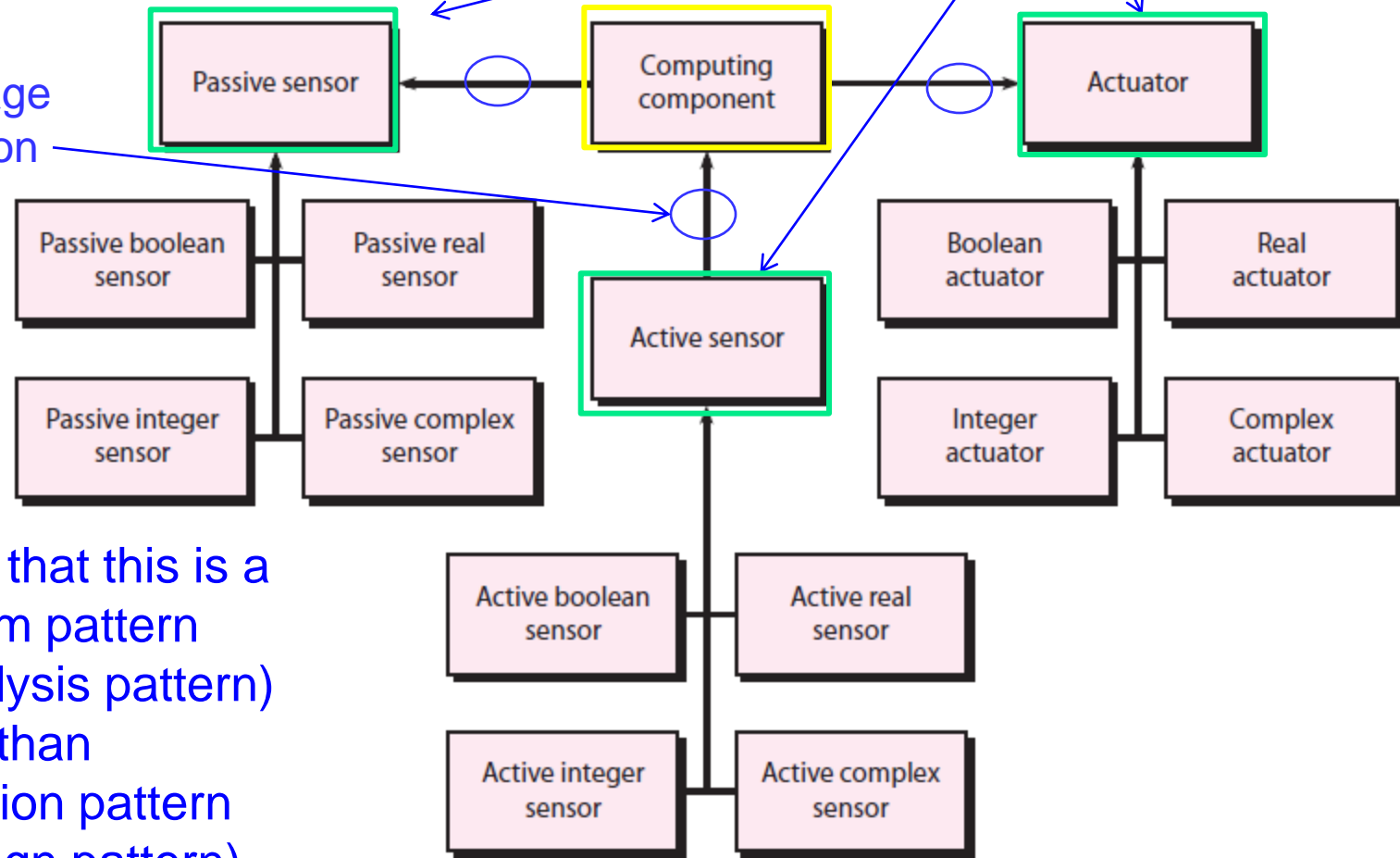
For the UML class diagram for the *Actuator-Sensor* Pattern see the next slide.

Actuator-Sensor Pattern

abstract classes

UML class diagram for the Actuator-Sensor pattern. Source: Adapted from [Kon02] with permission.

message direction

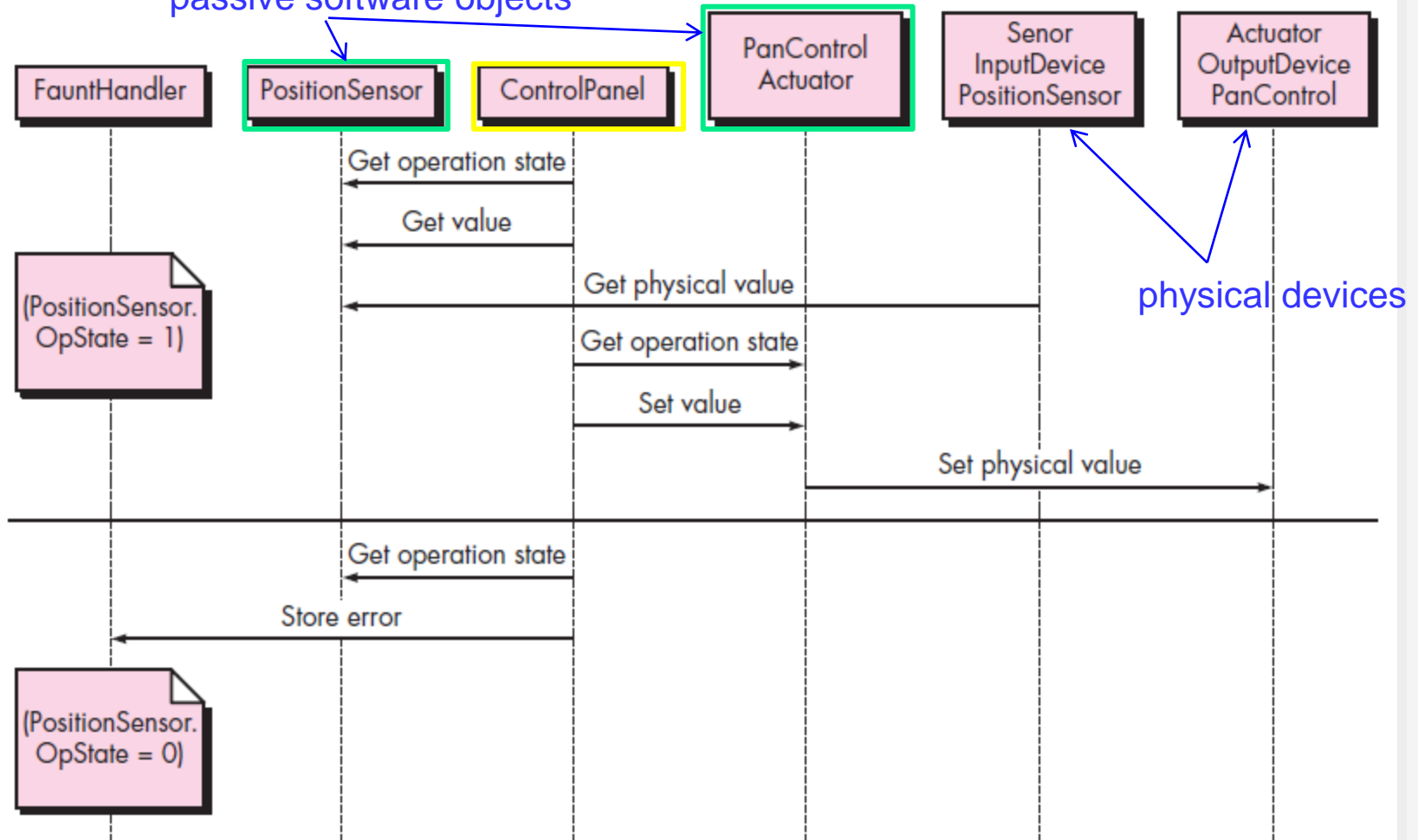


Notice that this is a problem pattern (= analysis pattern) rather than a solution pattern (= design pattern)

Actuator-Sensor Pattern – Applied

UML sequence diagram for the Actuator-Sensor pattern. Source: Reprinted from [Kon02] with permission.

passive software objects



Writing the Software Specification



Software quality attributes

Safety	Understandability	Portability
Security	Testability	Usability
Reliability	Adaptability	Reusability
Resilience	Modularity	Efficiency
Robustness	Complexity	Learnability

- And many more !
- Should include at least two among your requirements
- Should show your plan and what you did to achieve those quality attributes and also the results
 - Your project plan should include “quality assurance plan”



| 2/11/2016 4:15 PM EST

RLINGTON, VA. — The Pentagon must mitigate key software issues in [Lockheed Martin's](#) (LMT) F-35 by late spring, or else the Air Force might have to delay declaring its variant combat-ready.

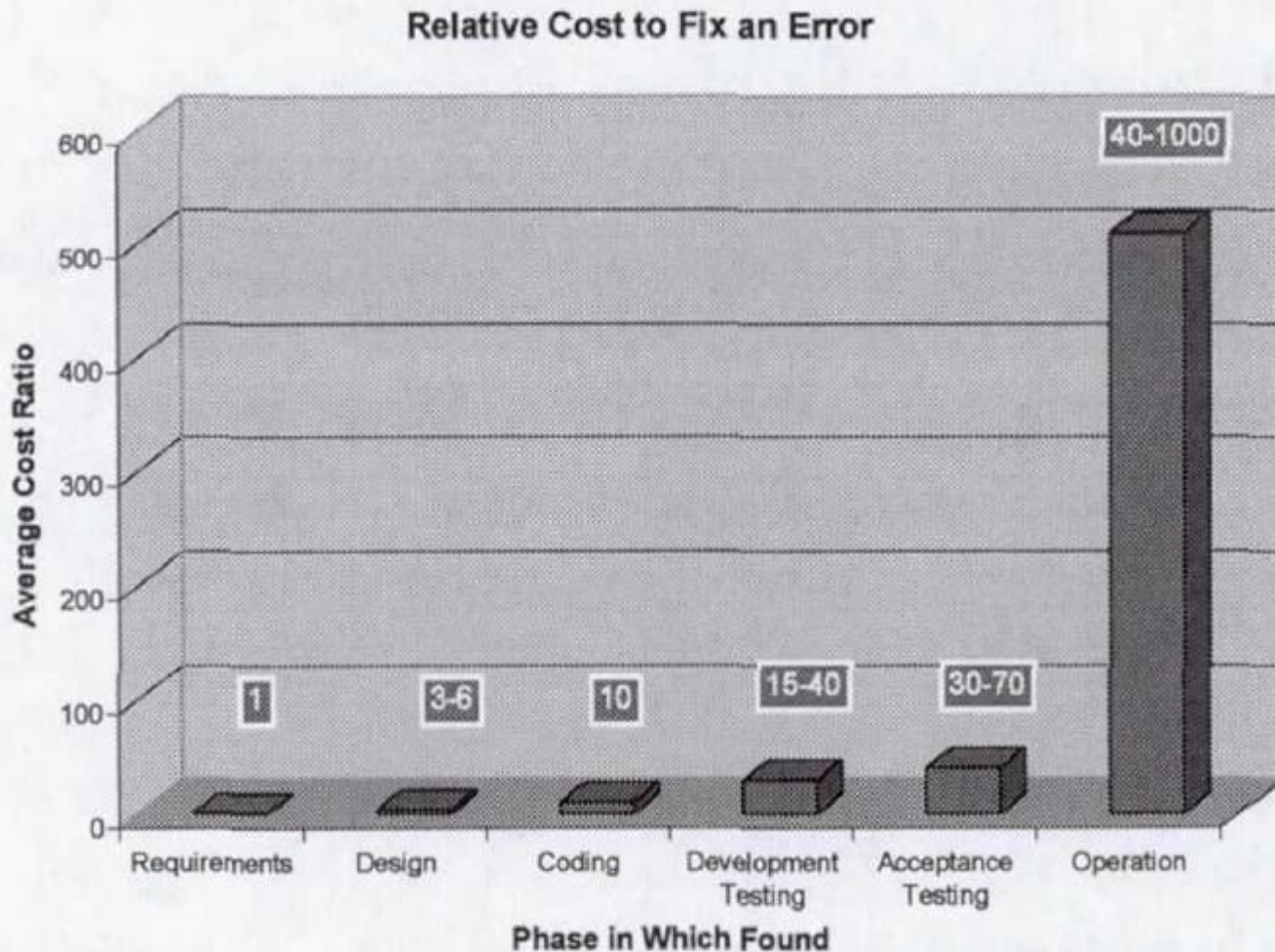
The plane needs to be rebooted once every four flight hours, Lt. Gen. Chris Bogdan, the F-35 program's executive officer, said at a briefing late Wednesday. "Not a good metric."

F-35 Could Be Delayed Another Four Years, But Pentagon And Lockheed Martin Deny Delays

BY CHRISTOPHER HARRESS  ON 12/31/14 AT 8:48 PM

America's most expensive military project looks like it may be delayed until 2019 after it was discovered that a software glitch in the ultra-sophisticated F-35 stealth jet could prevent the aircraft from firing its powerful cannon, according to the Daily Beast. The jet, which costs about \$161.1 million each and will set back the U.S. taxpayer nearly \$400 billion in total, has been scheduled to enter service next year.

Why requirements engineering is important



Source of Acquisition
NASA Johnson Space
Center