Requirement Engineering

By

Dr Damien Hanyurwimfura

Behavioral Modeling

The behavioral model

- The *behavioral model* indicates how software will respond to external events or stimuli.
- To create the model, the following steps are performed:

  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.

Identifying Events with use cases

- A use-case is examined for *points of information exchange*.

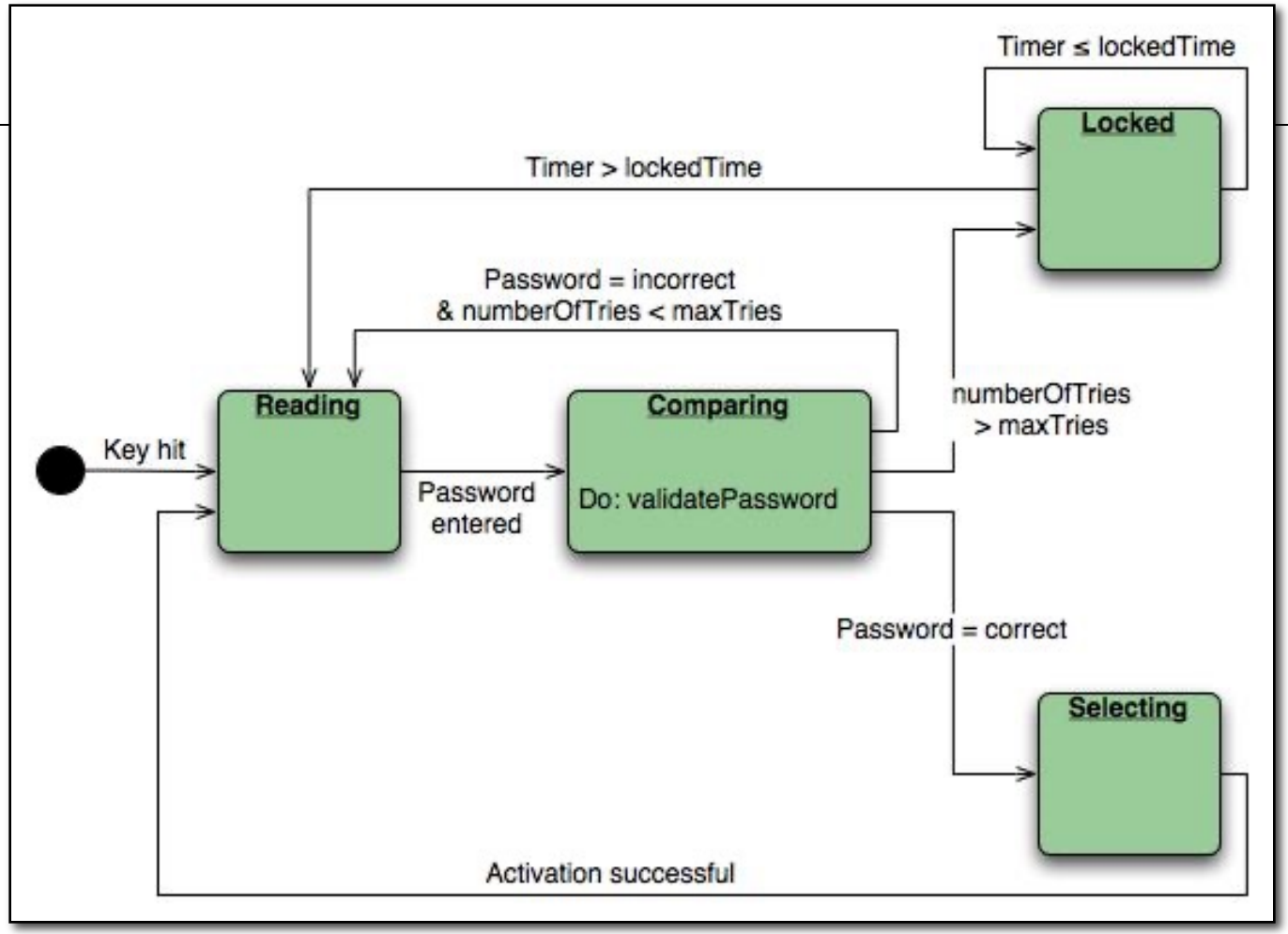
The homeowner uses the keypad to key in a four-digit password. The password is compared with the valid password stored in the system. If the password is incorrect, the control panel will beep once and reset itself for additional input. If the password is correct, the control panel awaits further action.

- The underlined portions of the use case scenario indicate events.
- An actor should be identified for each event; the information that is exchanged should be noted, and any conditions or constraints should be listed.

State Representations

- **State diagrams for analysis classes.** One component of a behavioral model is a UML state diagram that represents active states for each class and the events that cause changes between these active state.

State Diagram



State diagram for the ControlPanel class

State diagrams for analysis classes

- Each arrow shown in above Figure represents a transition from one active state of an object to another.
- The labels shown for each arrow represent the event that triggers the transition.
- Although the active state model provides useful insight into the "life history" of an object, it is possible to specify additional information to provide more depth in understanding the behavior of an object

State diagrams for analysis classes

- In addition to specifying the event that causes the transition to occur, you can specify a guard and an action
- A *guard* is a Boolean condition that must be satisfied in order for the transition to occur.
- For example, the guard for the transition from the "reading" state to the "comparing" state in Figure above can be determined by examining the use case:
- if (password input = 4 digits) then compare to stored password

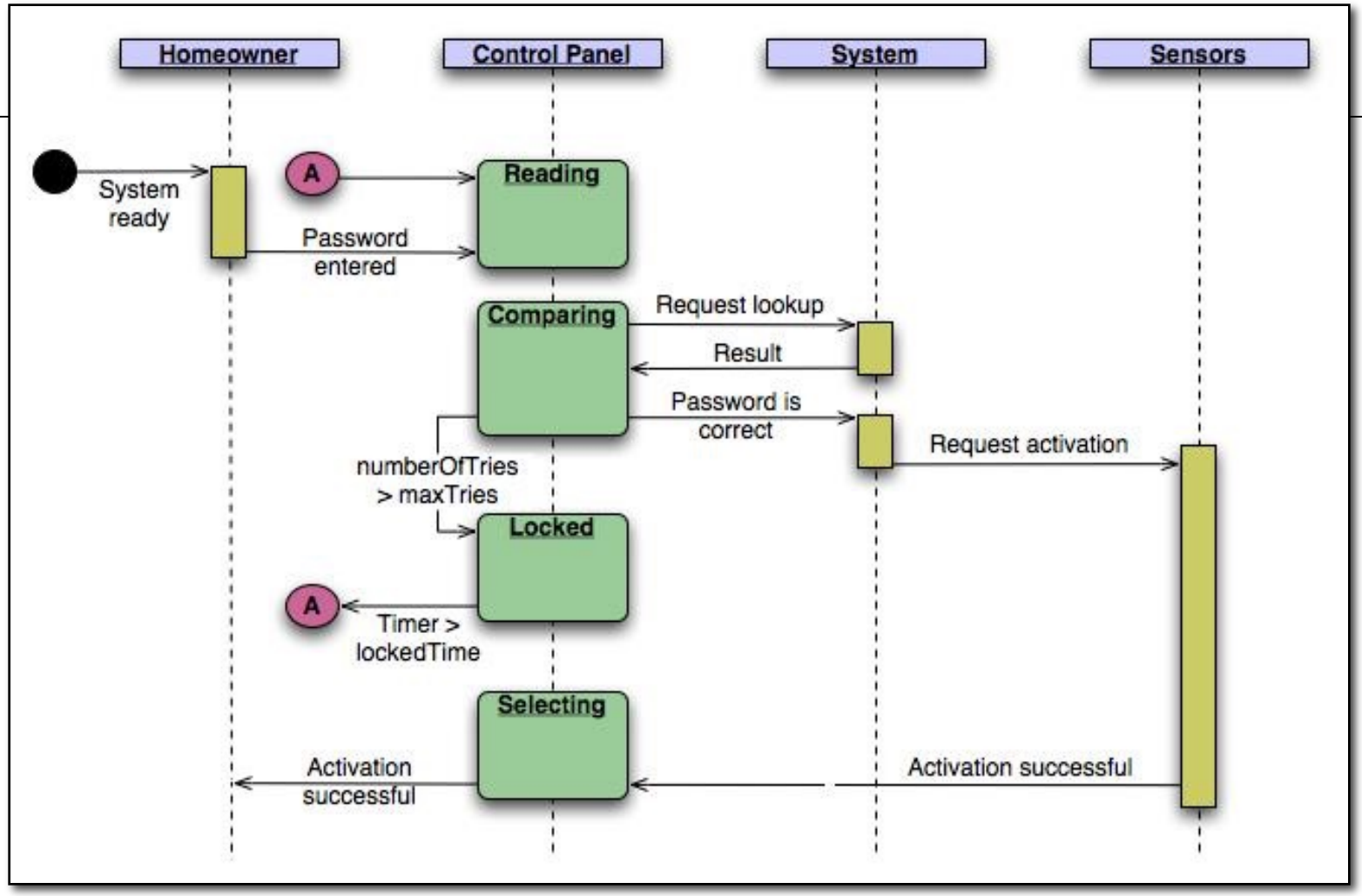
State diagrams for analysis classes

- An *action* occurs concurrently with the state transition or as a consequence of it and generally involves one or more operations (responsibilities) of the object.
- For example, the action connected to the *password entered* event is an operation named *validatePassword()* that accesses a **password** object and performs a digit-by-digit comparison to validate the entered password.

Sequence diagrams

- The second type of behavioral representation, called a *sequence diagram* in UML, indicates how events cause transitions from object to object.
- Once events have been identified by examining a use case, the modeler creates a sequence diagram—a representation of how events cause flow from one object to another as a function of time.
- Each of the arrows represents an event (derived from a use case) and indicates how the event channels behavior between *SafeHome* objects.
- Time is measured vertically and the narrow vertical rectangles represent time spent in processing an activity.
- States may be shown along a vertical time line.

Sequence Diagram



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

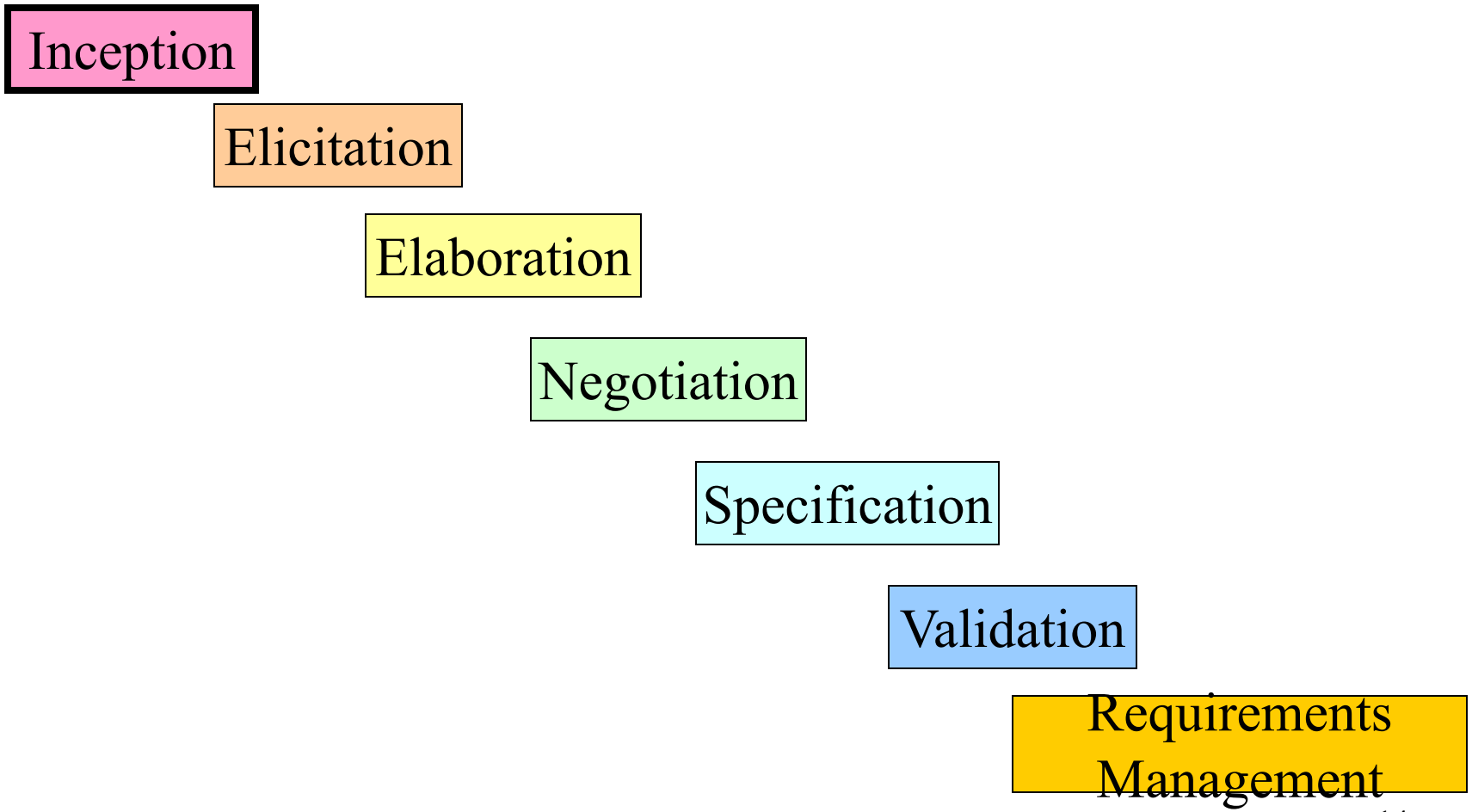
Sequence Diagram

- The first event, *system ready*, is derived from the external environment and channels behavior to the **Homeowner** object.
- The homeowner enters a password.
- A *request lookup* event is passed to **System,** which looks up the password in a simple database and returns a *result* (*found* or *not found*) to **ControlPanel** (now in the *comparing* state).
- A valid password results in a *password=correct* event to **System,** which activates **Sensors** with a *request activation* event.
- Ultimately, control is passed back to the homeowner with the *activation successful* event

Assignment 2

- Refer to the SRS example and template, draft Software Requirement Specification (SRS) document for your own software of the project topic selected in previous assignment

Requirements engineering tasks



Requirements validation

- Concerned with demonstrating that the requirements define the system that the customer really wants.
- Requirements ***validation covers a part of analysis*** in that it is concerned with ***finding problems*** with requirements.
- ***Requirements error costs are high*** so validation is very important
  - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.
  - In fact, a change to the requirements usually means that the ***system design*** and the ***implementation*** must also be changed and the ***testing*** has to be performed again.

Validating Requirements

- Is each requirement consistent with the objective of the system?
- Is the requirement really necessary?
- Is each requirement bounded and unambiguous?
- Does each requirement have attribution?
- Do any requirements conflict with other requirements?
- Is each requirement achievable in the system′s technical environment?
- Is each requirement testable, once implemented?
- Does the model reflect the system′s information, function and behavior?
- Has the model been appropriately "partitioned"?
- Have appropriate requirements patterns been used?

- These and other questions should be asked and answered to ensure that the requirements model is an accurate reflection of stakeholder needs and that it provides a solid foundation for design.

Requirements validation techniques

The following techniques can be used *individually* or in *conjunction*.

- ***Requirements reviews***
  - Systematic manual analysis of the requirements performed by a team of reviewers .
- ***Prototyping***
  - Using an executable model of the system to check requirements.
- ***Test-case generation***
  - Developing tests for requirements to check testability.
  - If the test is difficult to design, usually the related requirements are difficult to implement.

Requirements reviews

- A requirements review is a **manual process** that involves both ***client and software*** *and* should be involved in reviews.
- In other words these people **should discuss.**
- Regular reviews should be held **while the requirements definition is being formulated**.
- Reviews may be formal (with completed documents) or informal.
- Good communications between developers, customers and users can resolve problems at an early stage.

Formal and informal reviews

- ***Informal reviews*** simply involve contractors ***discussing*** requirements with as many system stakeholders as possible;
- ***Formal reviews*** the development team should **"take" the client through**
  - the system requirements and
  - explaining the implications of each requirements.

Checks required during the requirements validation process

- Validity checks. Does the system provide the functions which best support the customer's needs?
- Consistency checks. Are there any requirements conflicts?
- Completeness checks. Are all the requirements needed to define all functions required by the customer sufficiently specified?
- Realism checks. Can the requirements be implemented given available budget, technology and schedule?
- Verifiability. Can the requirements be checked?

Checks that should be performed by reviews cont..

- Verifiability. Is the requirement realistically testable?
- Comprehensibility. Is the requirement properly understood?
- Traceability. Is the origin of the requirement clearly stated? It might be necessary to go back to the source of a requirement to assess the impact of the change.
- Adaptability. Can the requirement be changed without a large impact on other requirements?

Requirements validation outputs

- The requirements validation process **should output a report** which points out potentially conflicts, contradictions, errors and omissions.
- Then it is up to the user, the contractor staff, the developers **to negotiate a solution** to the reported problems.

Requirements management

- The requirements for large systems are frequently changing.
- In fact, during the software process, the **stakeholders' understanding of the problem** is constantly changing.
- Requirements management is the process of managing ***changing requirements*** during the ***requirements engineering process*** and ***system development.***
- Requirements are inevitably incomplete and inconsistent
  - New requirements emerge during the process as business needs change and a better understanding of the system is developed;
  - Different viewpoints have different requirements and these are often contradictory.

Requirements management

- It is hard for the users and customers to anticipate what effects the new system will have on the organization.
- Often, only when the system has been deployed, new requirements inevitably emerge.
- This is mainly due to the fact that, **when the end-users have experience of the new system**, they discover new needs and priority.

Requirements change

1. The priority of requirements from different viewpoints changes during the development process.
2. **Conflicts** have to inevitably converge in a compromise.
3. System customers may specify requirements from a business perspective that conflict with end-user requirements.
4. The business and technical environment of the system **changes** during its development.
5. New hardware, new interface, business priority, new regulations, etc.

Requirement changes and the requirements management

The requirements management is the process of identifying, understanding and controlling changes to system requirements.

- It might be useful **to keep track of individual requirements and maintain links** between dependent requirements so that you can asset the impact of requirements changes.
- The process of requirements management ***should start as soon as*** a draft a version of the requirement document is available.

Requirements evolution