

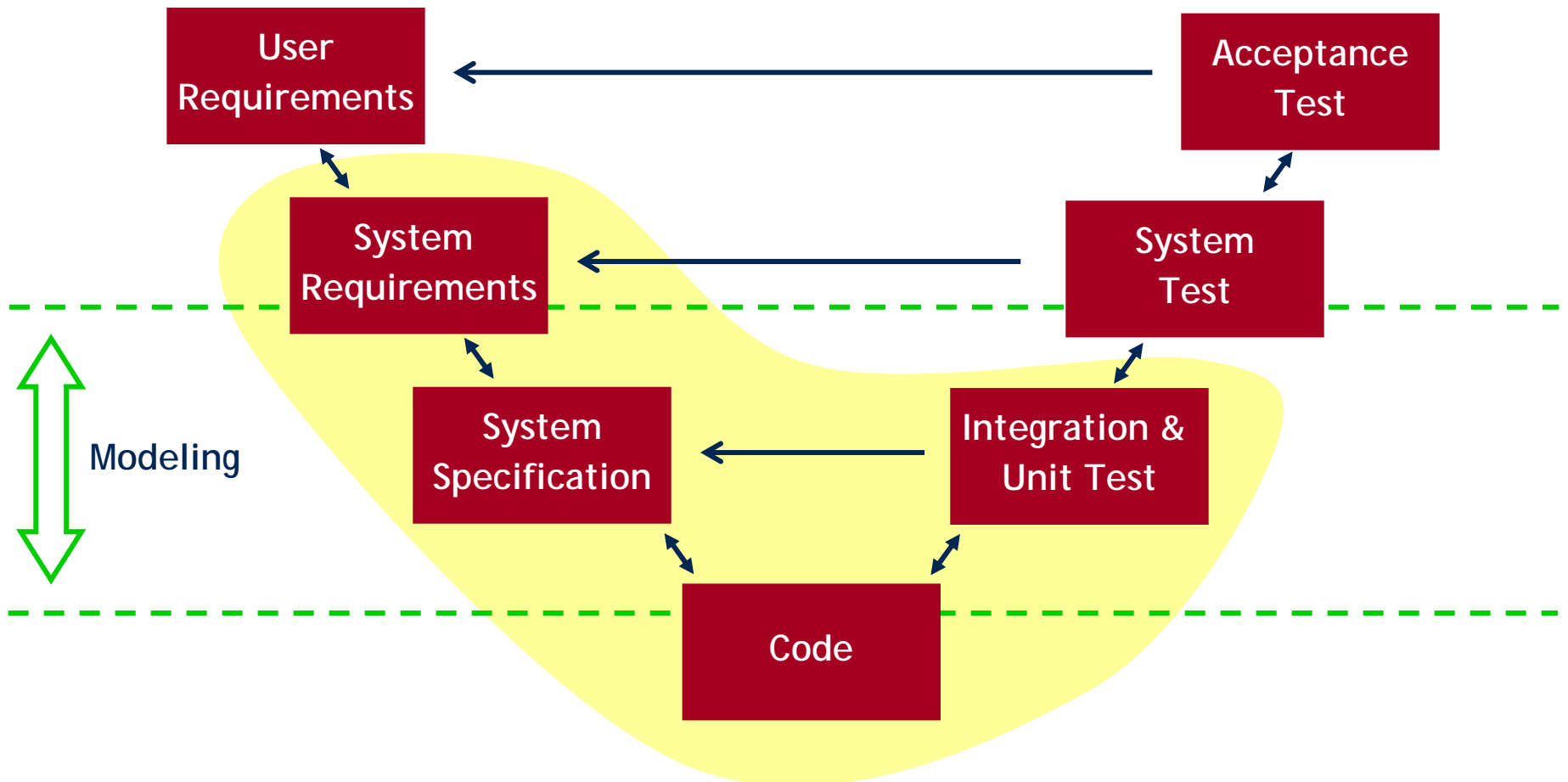


# Linking Code to Requirements through Modeling

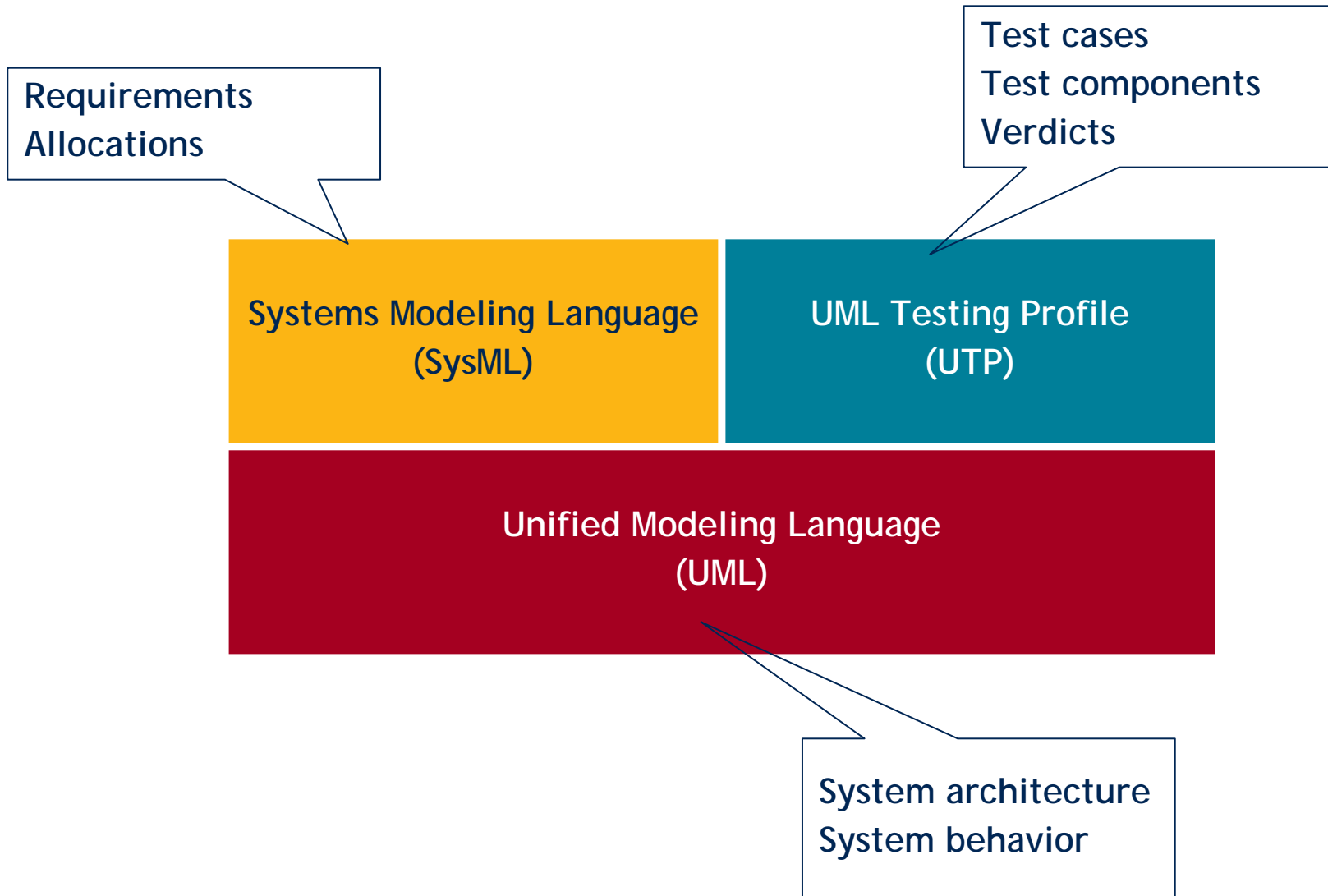
Morgan Björkander



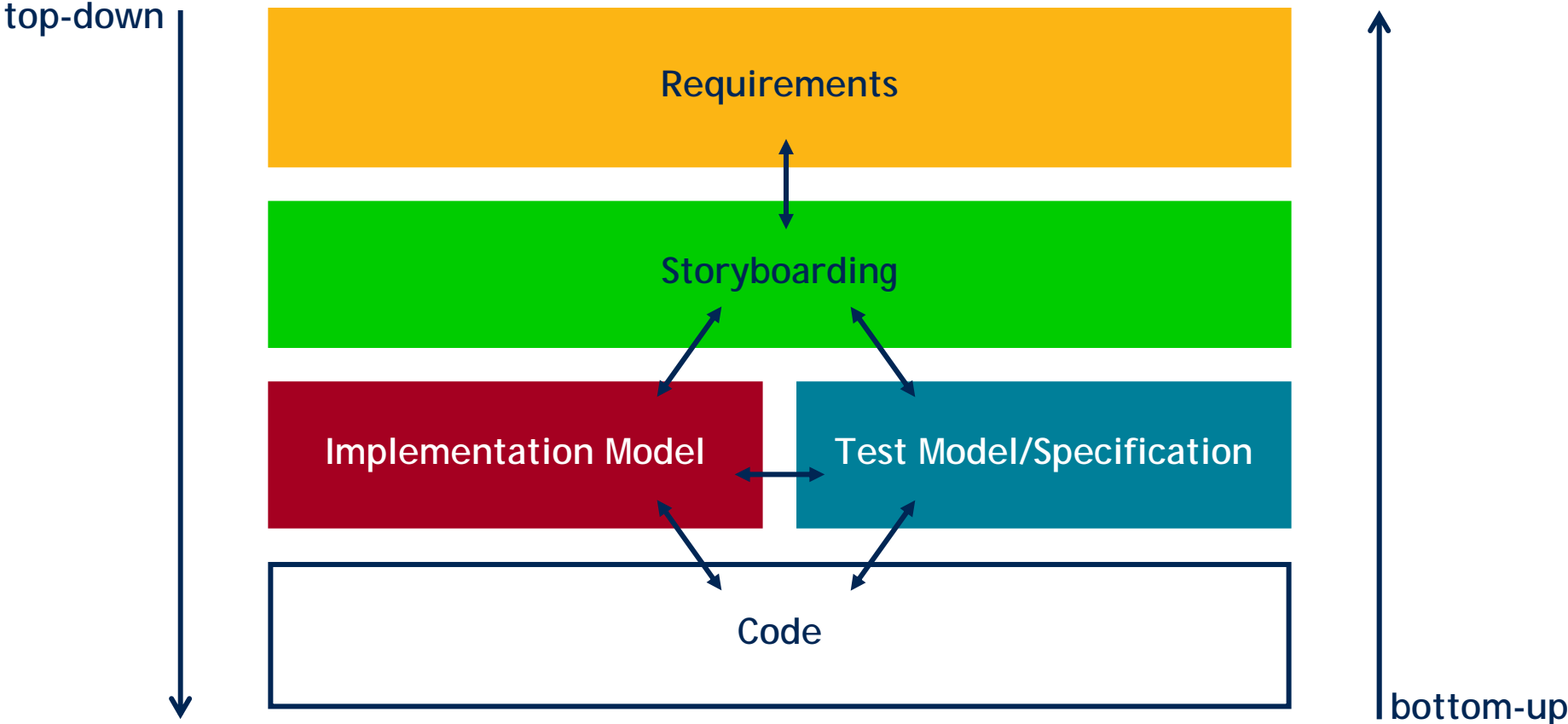
# The V-Model



# Some Relevant OMG Standards

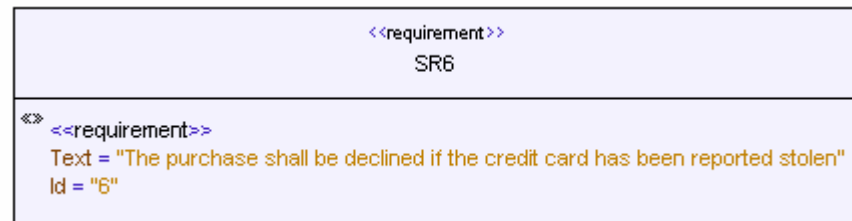


# A Layered Approach



# SysML

- Specifically geared towards systems engineers
  - but applicable also to software engineers
  - a scaled down variation of UML with specific extensions
- Most important feature is the **requirement diagram**
- Used to show **requirements** and their relationships
  - to other requirements
  - to model elements
  - to test cases
- An **allocation** is a cross-cutting construct
  - indicates that one thing maps to another in some way, undecided how (e.g. logical class to physical node, use case to class, etc.)
- Numerous other concepts that are out of scope for this presentation



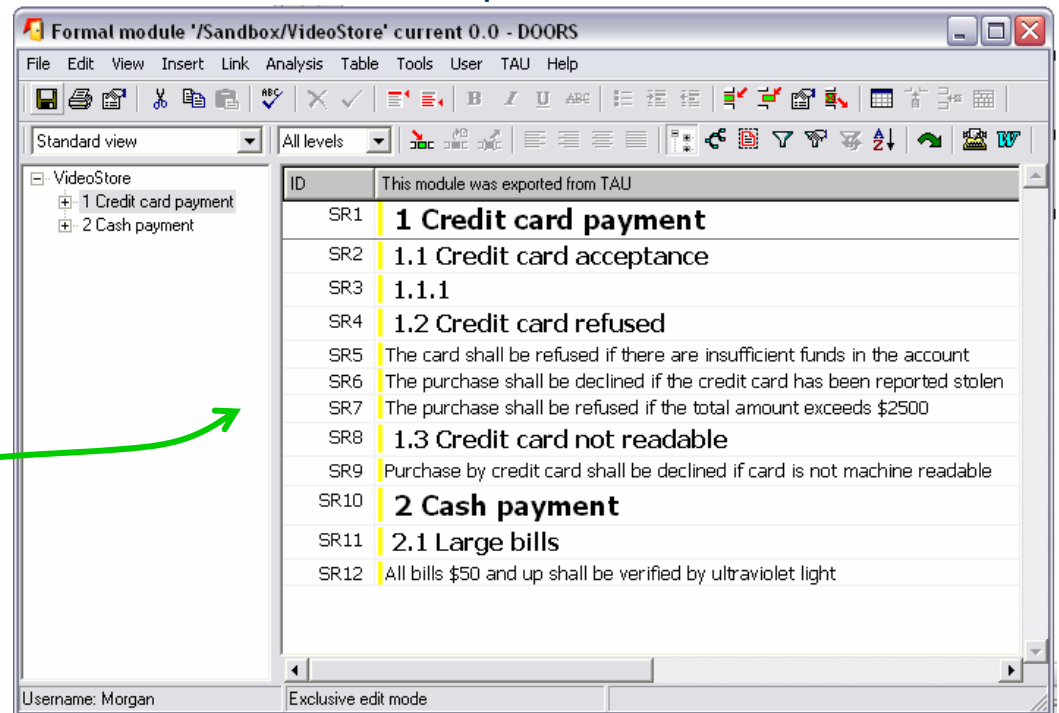
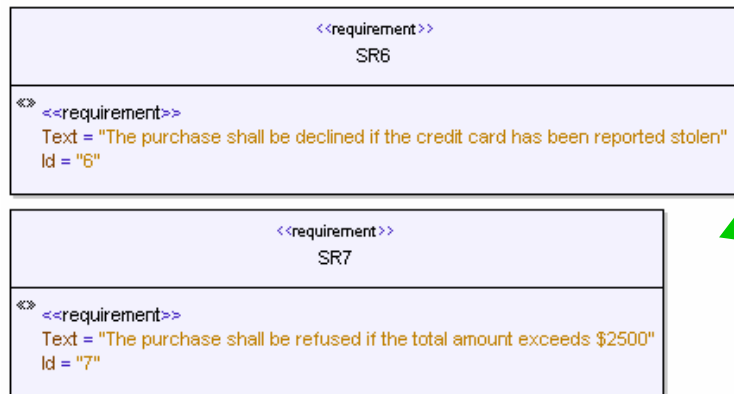
# Why Model Requirements?

- Requirements always drive any development project
  - determines what should and (indirectly) should not be done
- Requirements are typically expressed textually
  - in Word or Excel
  - in RM tools (such as DOORS)
- The major reason to model requirements is to enable **traceability**
  - end-to-end
- **Coverage** analysis
  - which requirements have been implemented?
  - which parts of the code are not based on requirements?
- **Derivation** analysis
  - how many files in the code are affected if a requirement changes?
- **Impact** analysis
  - what other parts of the system would be affected if a component is rearchitected?



# Requirements Visualization

- **Visualization** of requirements
  - show relationships graphically using a standard notation
  - simple to establish traceability links in the context of the model
- Allows easy **integration** between requirements tools and modeling tools
  - a complementary way to define requirements



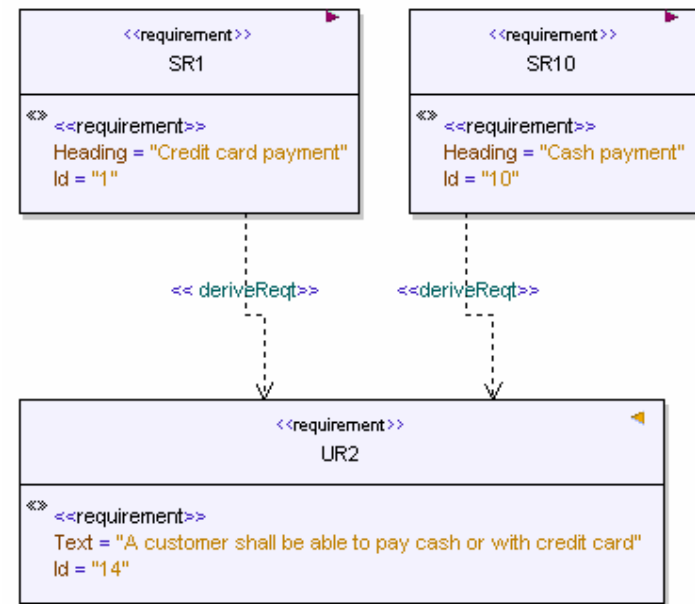
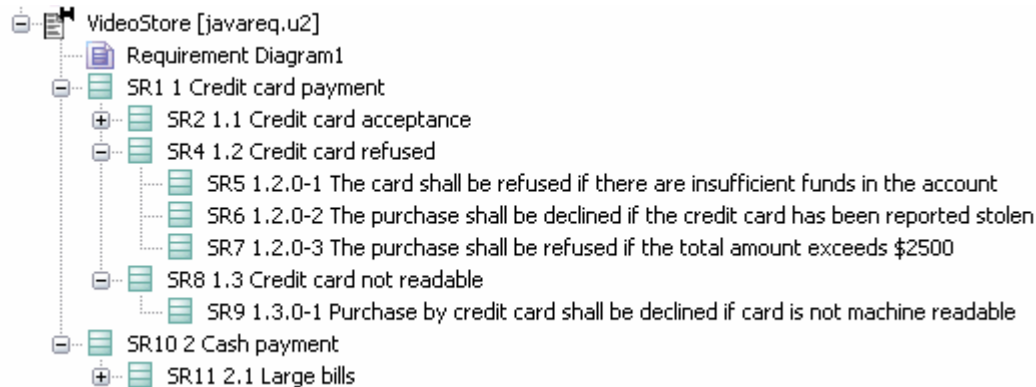
# Relating Requirements

- Contain

- a requirement may contain (sub-)requirements
- pretty much the same way a book may contain chapters, headings, and text

- Derive

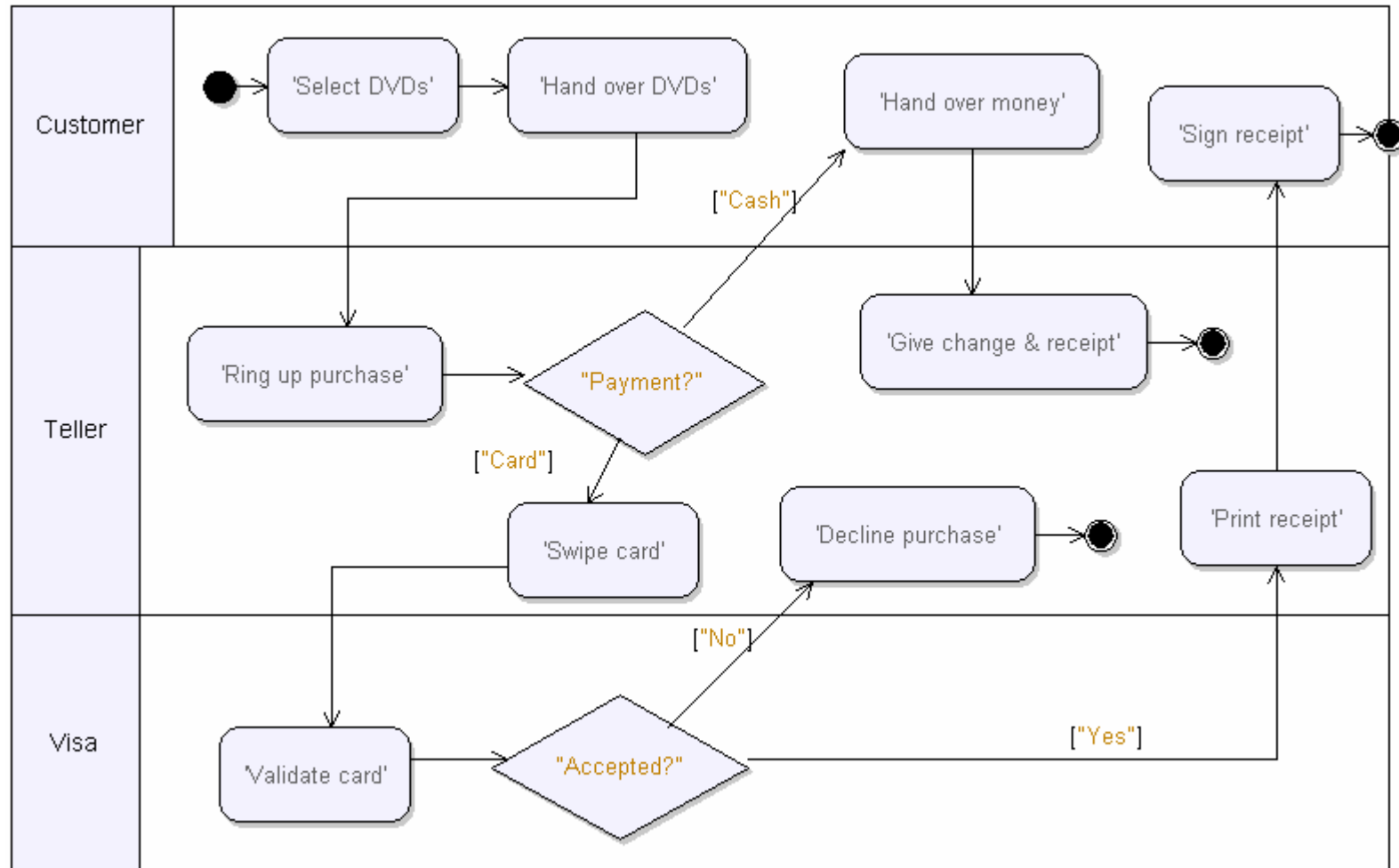
- a requirement can be derived from other requirements
- typically from system requirements to user requirements, etc.



# Storyboarding

- Storyboarding describes **high-level** system functionality
  - “pseudocode”
  - simulatable
- Provides a logical model that is common to implementation and test models
- Typically expressed in the form of **activity diagrams**
  - to show workflow or performed actions
  - may be partitioned to show control flow between different classes
- Usually tied to use cases
  - activity partitions correspond to actors

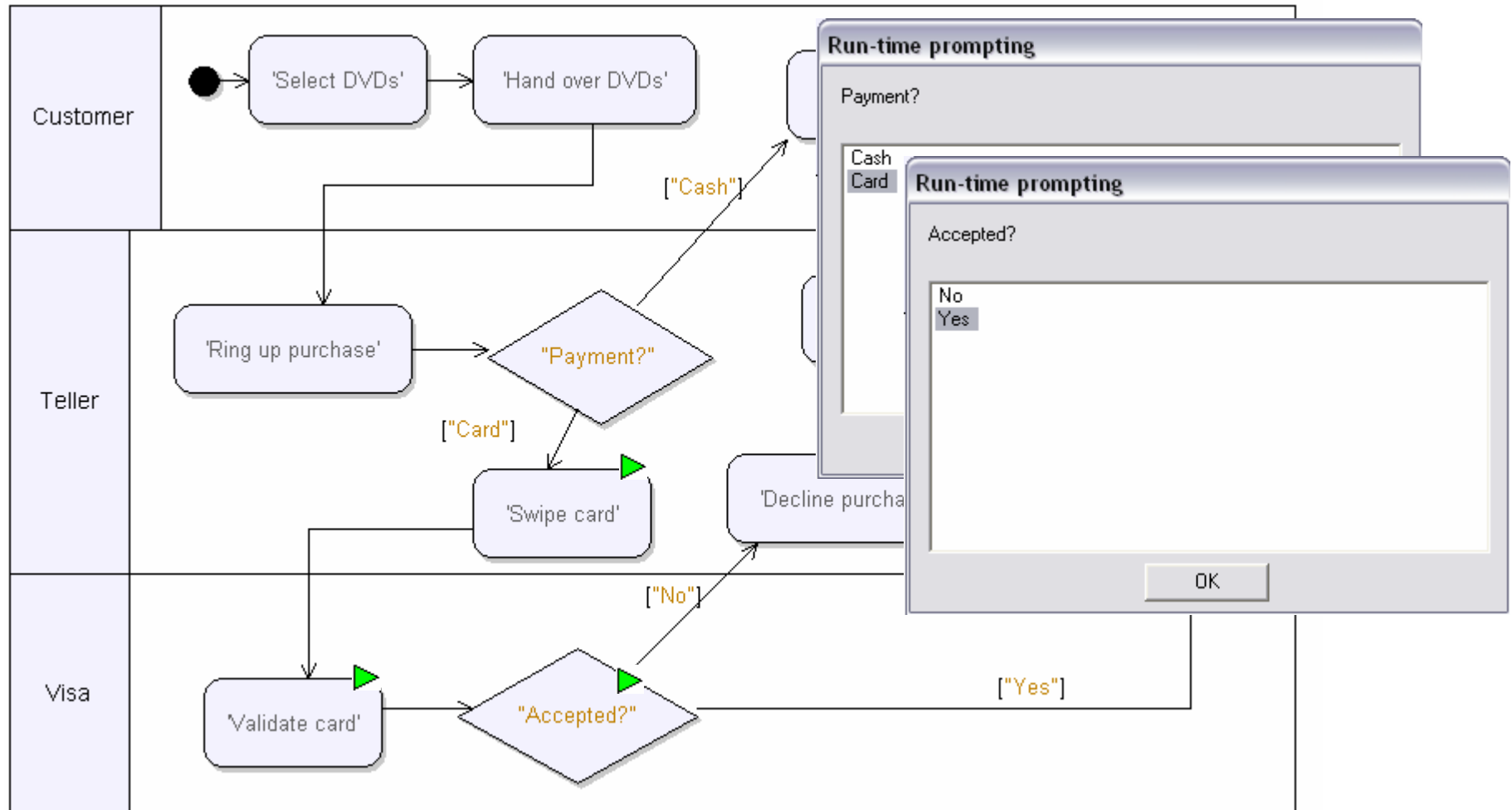
# A Sample Storyboard



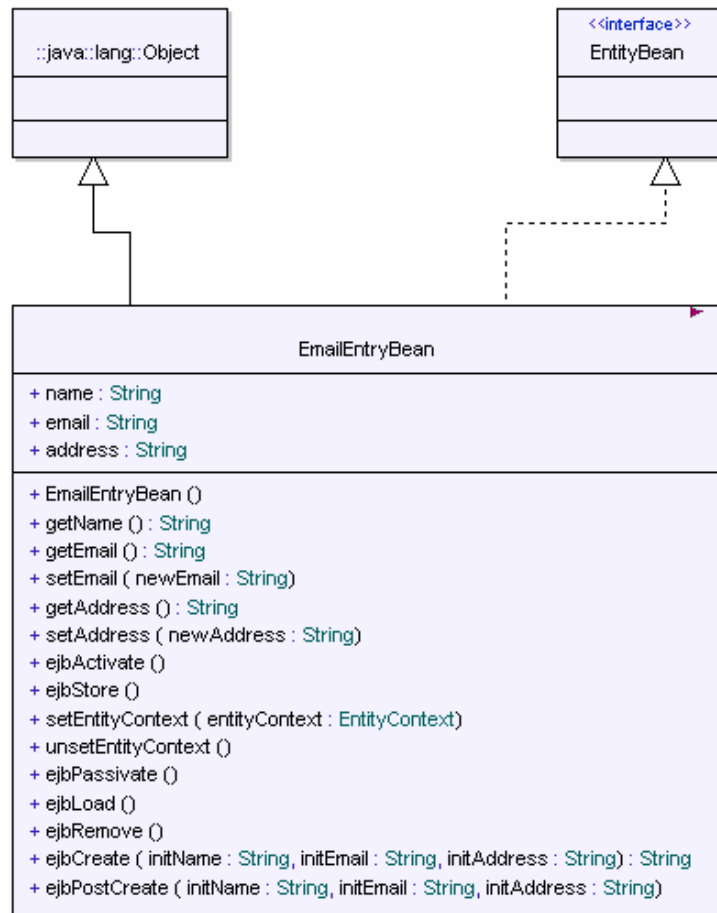
# Benefits of Storyboards

- A storyboard serves multiple purposes
  - helps understand how requirements impact implementation and test
  - describes how a system or part of a system is *intended* to work “before the fact”
  - documents system functionality
- Communicate and verify functionality with relevant stakeholders
  - often not technical or programming savvy
- Conversely, storyboards can be used to document legacy code
  - convey how a system *actually* works “after the fact” (at a higher level than code)

# Animating a Storyboard



# Model-Code Synchronization



```
System Requirements | EmailEntryBean.java
package emailRegistry.ejb;

import javax.ejb.*;
import java.rmi.*;

public class EmailEntryBean
extends Object
implements EntityBean {

    public String name;
    public String email;
    public String address;

    // public static int instanceCount = 0;

    public EmailEntryBean() {
        // int instanceNr = instanceCount++;
    }

    public String getName() {
        return name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String newEmail) {
        email = newEmail;
    }

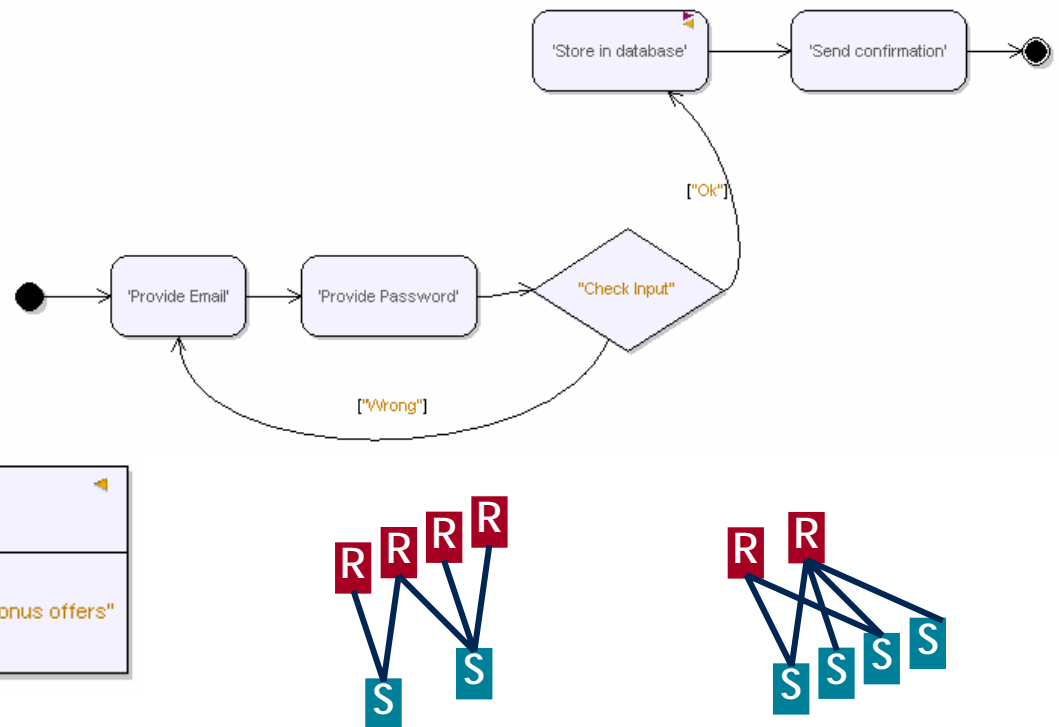
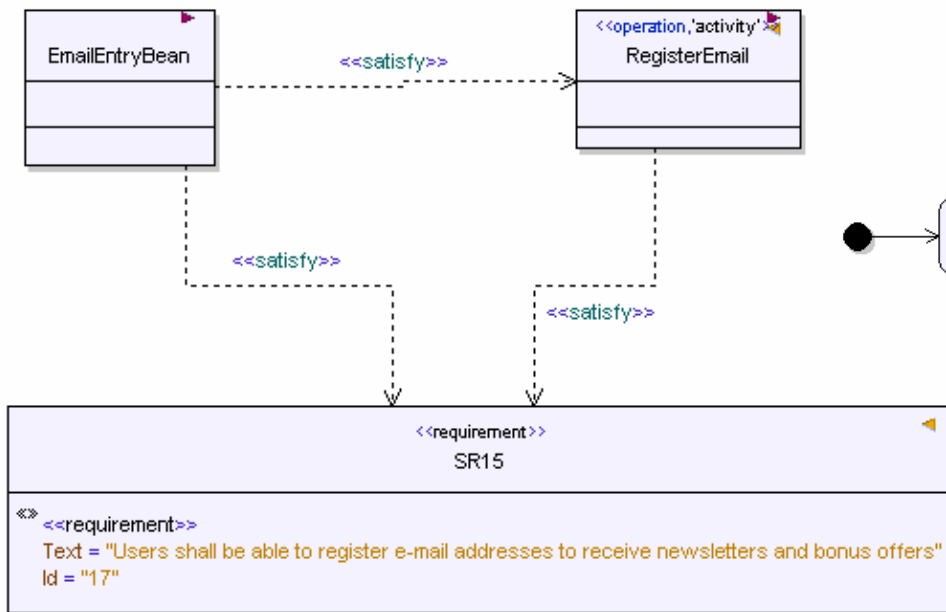
    public String getAddress() {
        return company;
    }

    public void setAddress(String newAddress) {
        address= newAddress;
    }
}
```

# Models and Requirements

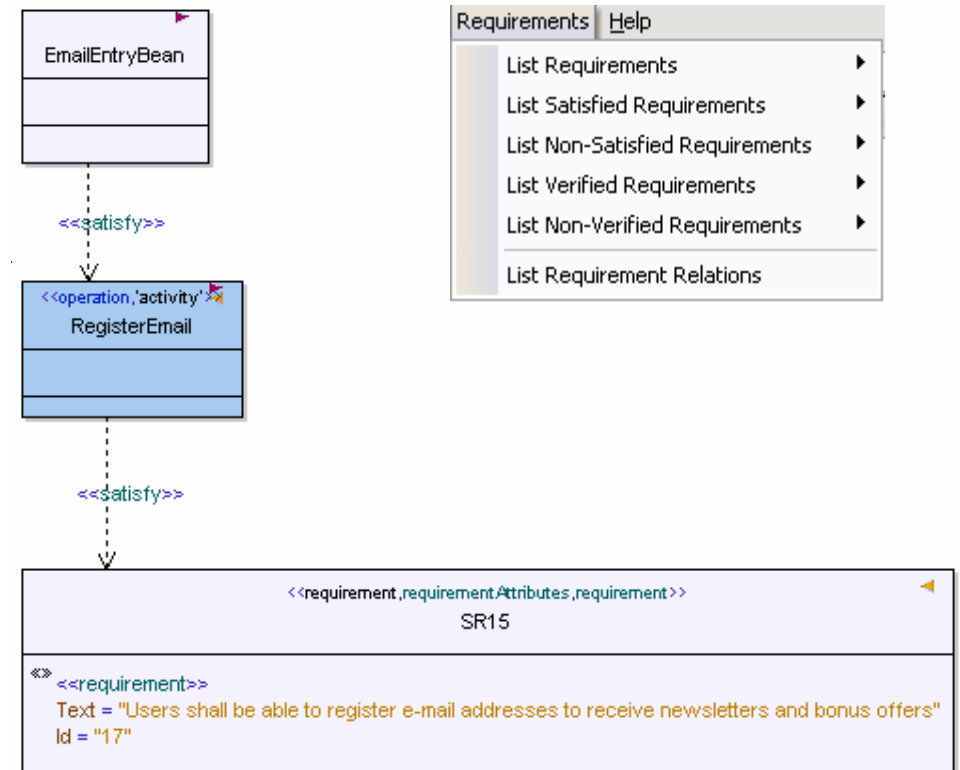
- Satisfy
  - a model element can satisfy a requirement

- Most often, storyboards serve as requirements for the implementation model
  - satisfy or trace



# Traceability Analysis

- A very simple but effective form of analysis is to generate diagrams showing dependencies
  - visualization of traceability links
- More specialized tools can provide extensive analysis information based on the links
  - all information is available in the model and can be processed in many different ways



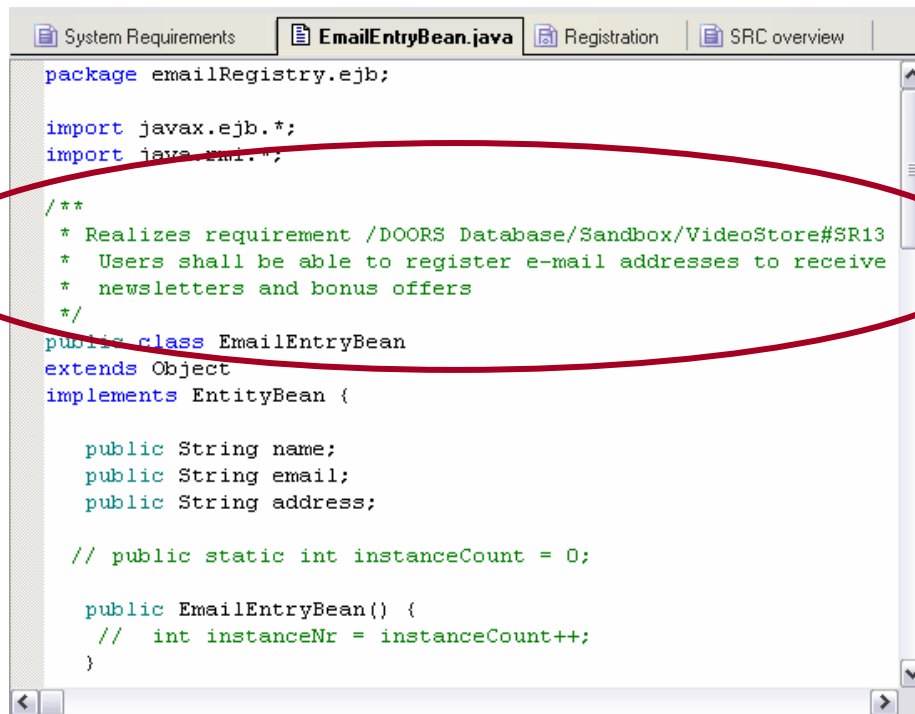
# Code and Requirements

- Indirectly

- linked only through model-code synchronization

- Satisfy

- a piece of code can satisfy a requirement
- represented in code/tools as comments, annotations, or direct links



```
System Requirements | EmailEntryBean.java | Registration | SRC overview
package emailRegistry.ejb;

import javax.ejb.*;
import java.util.*;

/**
 * Realizes requirement /DOORS Database/Sandbox/VideoStore#SR13
 * Users shall be able to register e-mail addresses to receive
 * newsletters and bonus offers
 */
public class EmailEntryBean
extends Object
implements EntityBean {

    public String name;
    public String email;
    public String address;

    // public static int instanceCount = 0;

    public EmailEntryBean() {
        // int instanceNr = instanceCount++;
    }
}
```

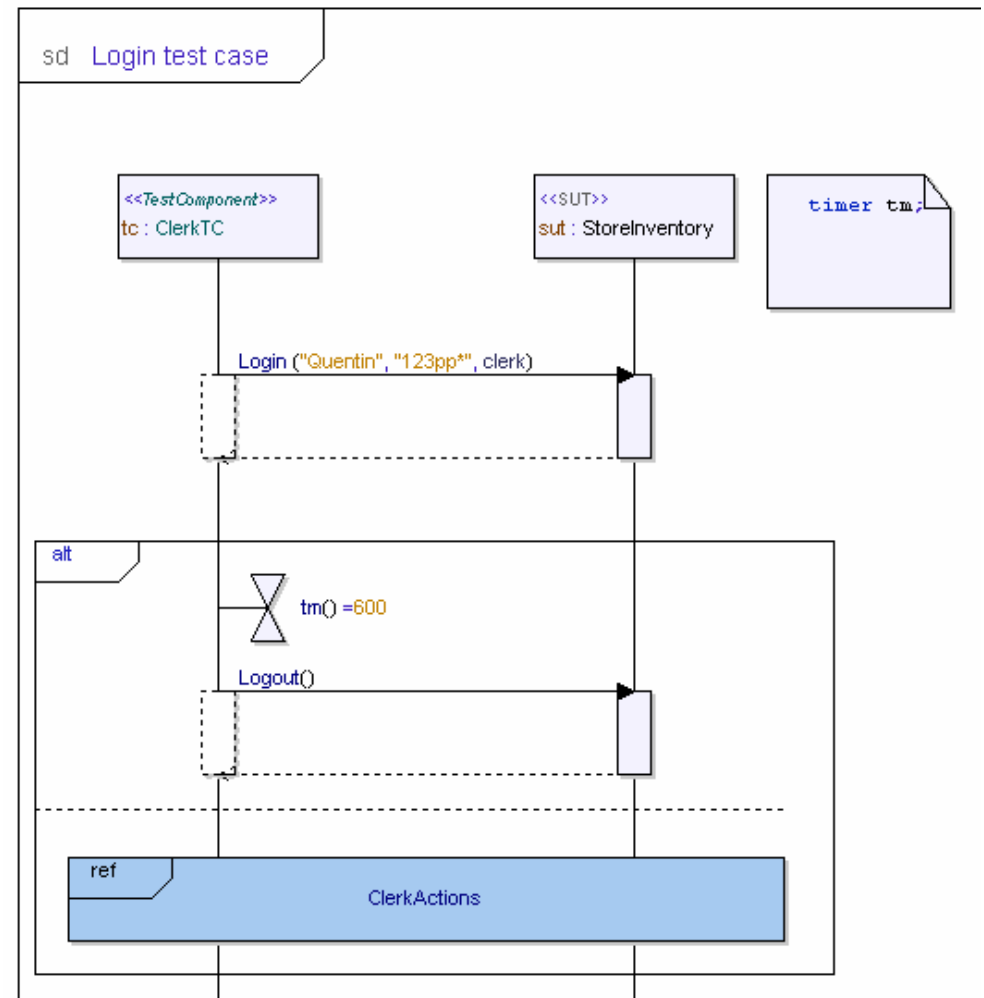
# UML Testing Profile

- Specifies a framework for creating test suites
  - at the same abstraction level as the system(s) being modeled
- Describes **test cases**
  - and the test components that execute them
  - defines a set of verdicts to determine the outcome of a test case
- Provides a test context
  - relates system under test with test components and relevant test cases
- The test model or test specifications complements the implementation model
  - describes the relevant tests
  - derived from the requirements and corresponding storyboard (or less ideally from the code)

{  
pass  
fail  
inconclusive  
error  
}

# Test Cases

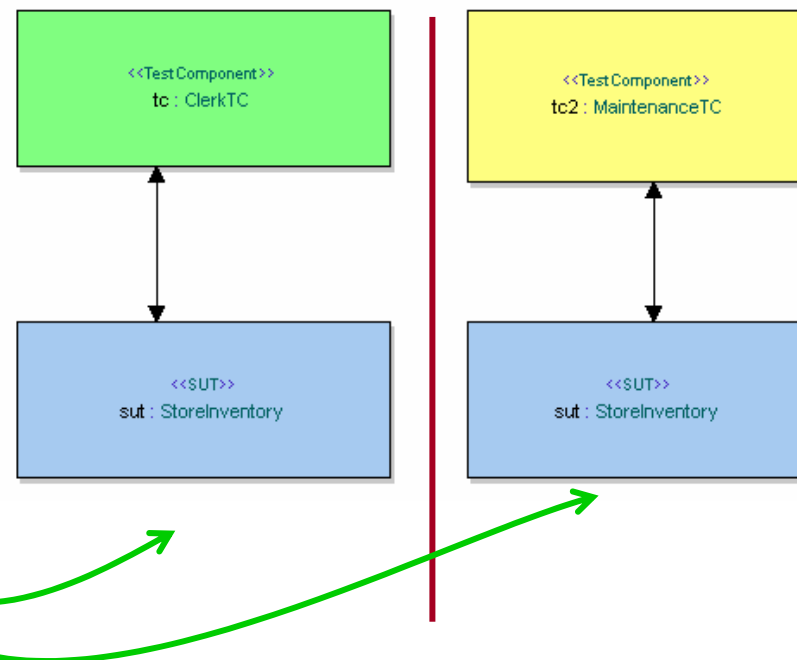
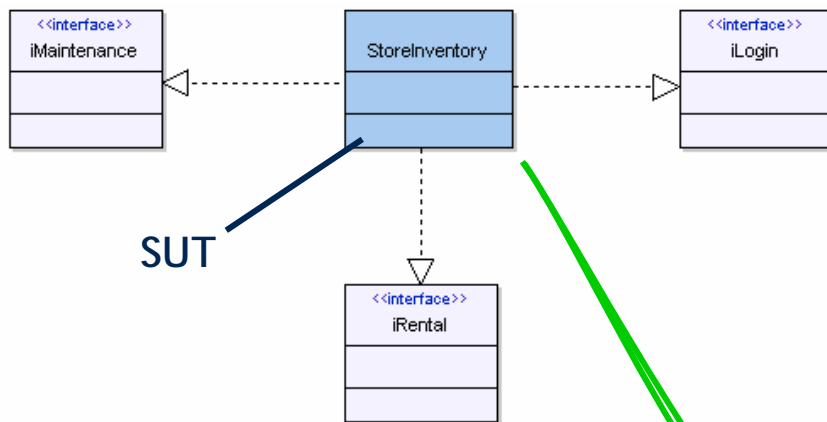
- Test cases can be described using any kind of UML behavior
  - interactions
  - activities
  - state machines
- Can capture synchronous as well as asynchronous behavior
  - verdicts can be explicitly set or derived from the actual message flow as tests are executed



# System Under Test and Test Components

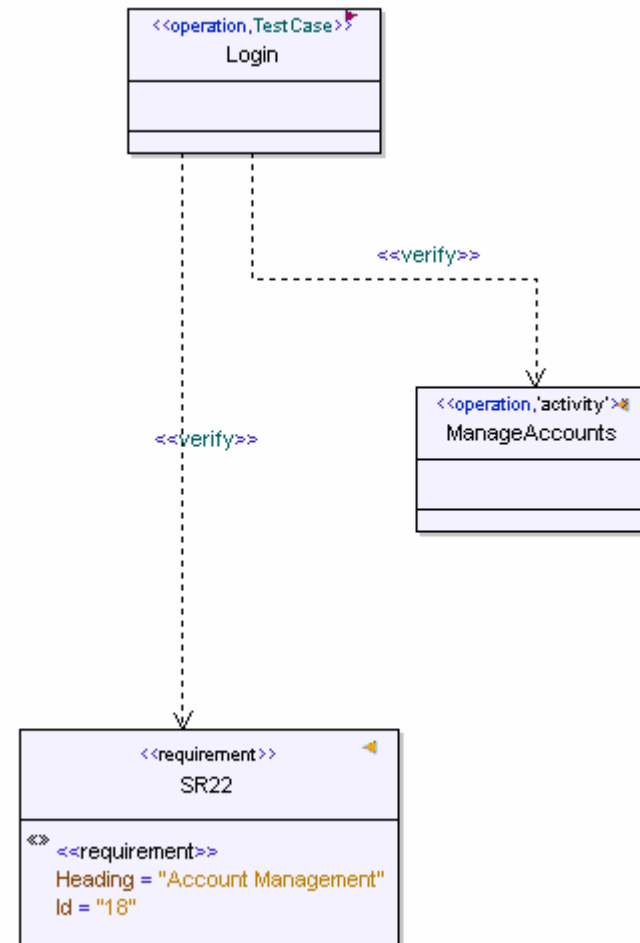
- System Under Test (**SUT**)
  - any system, sub-system, component, or class being tested

- Test component
  - describes and executes test cases against SUT
- Each test context is standalone
  - composite structure diagrams



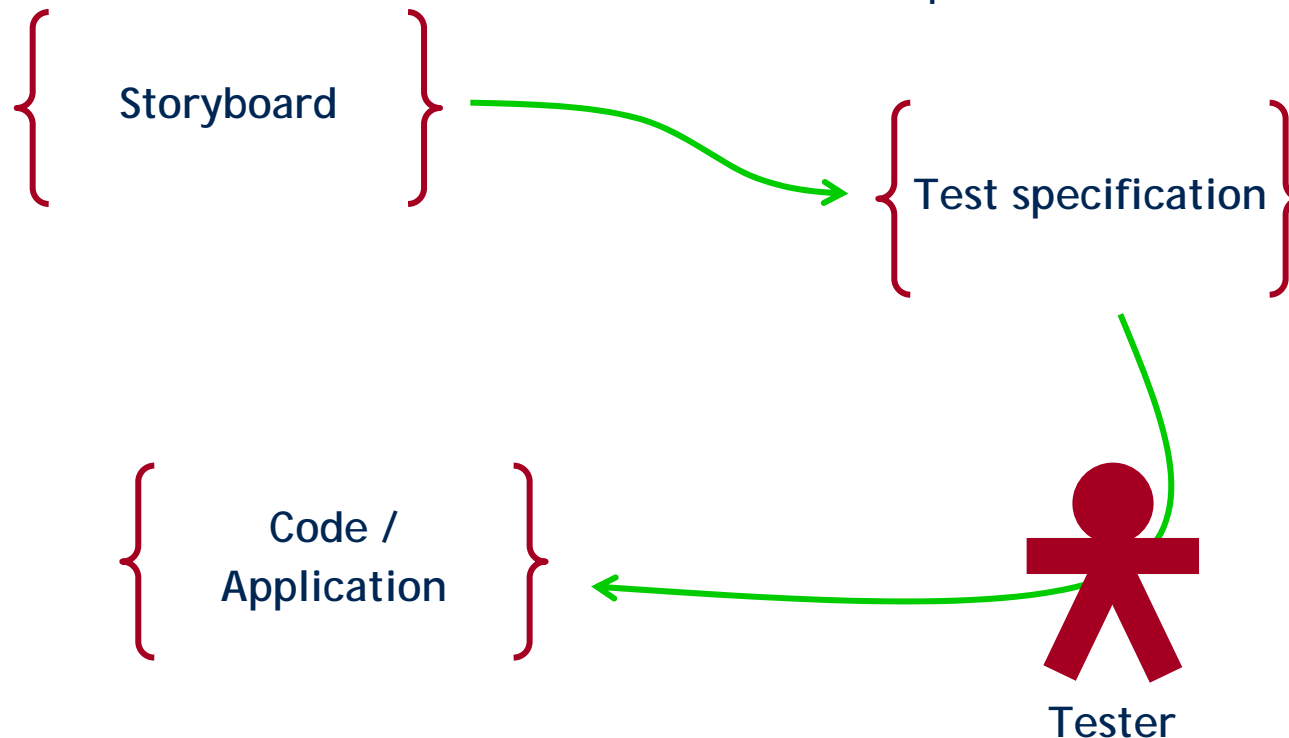
# Requirements and Test Cases

- Verify
  - a test case usually verifies requirement(s)
  - the test case is executed against the (implementation) model or the code



# Storyboards and Manual Test Scripts

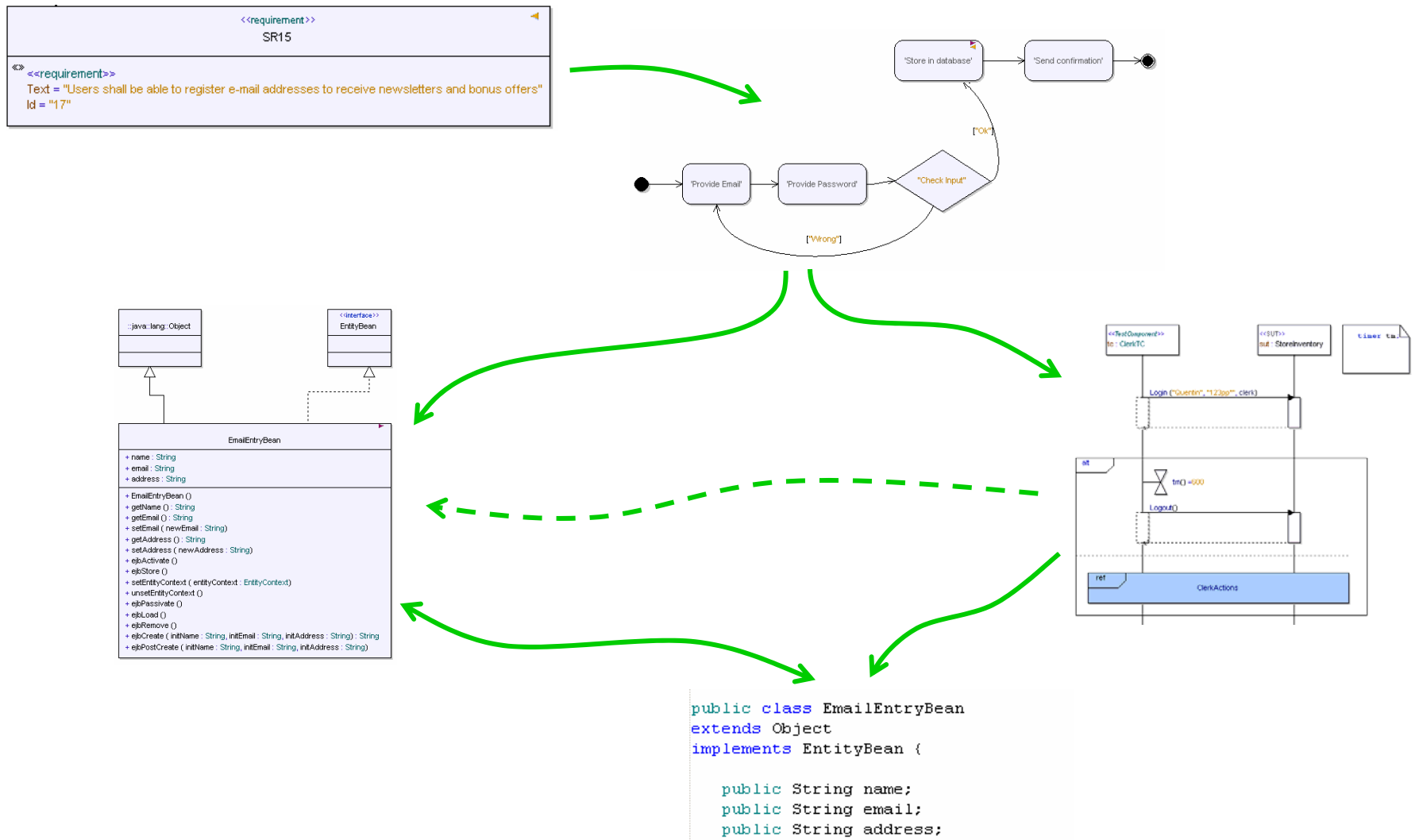
- Much testing rely on testers following test scripts in a manual fashion
- Storyboards can be used as input for producing the documents that make up the test specifications



# Maintaining Legacy Systems

- Import Java code (or C#, or C++) into the model
  - automatically create diagrams to quickly visualize dependencies between code structures
- For understanding
  - produce high-level storyboards to describe functionality
  - provide links from code/model
- For detailed investigation
  - produce sequence diagrams that capture current implementation in terms of message passing
- Determine new requirements
  - don't necessarily have to be modeled, but should be recorded
- Fit requirements into existing storyboards or create new ones
  - determine to what extent new functionality is affected
- Define new test cases or change existing ones
- Adapt implementation model to deal with new or changed functionality
  - manually or automatically synchronize the code

# Summary—Tying It All Together



# Conclusions

- Requirements provide the primary means to hold together any development project
  - through system design, test design, and implementation
- Depending on the desired granularity, it is possible to show traceability to requirements all the way from code
  - enables various kinds of analysis that makes it easier to plan and control projects and maintain applications
- Larger organizations or longer or bigger projects must take this more seriously
  - there is always a tradeoff between level of effort and return of investment
  - when the system grows bigger, an architecture model is a must
  - smaller projects and systems can be pretty sweeping and “light” on the requirements traceability and storyboarding

# Thank you

morgan.bjorkander @ telelogic.com