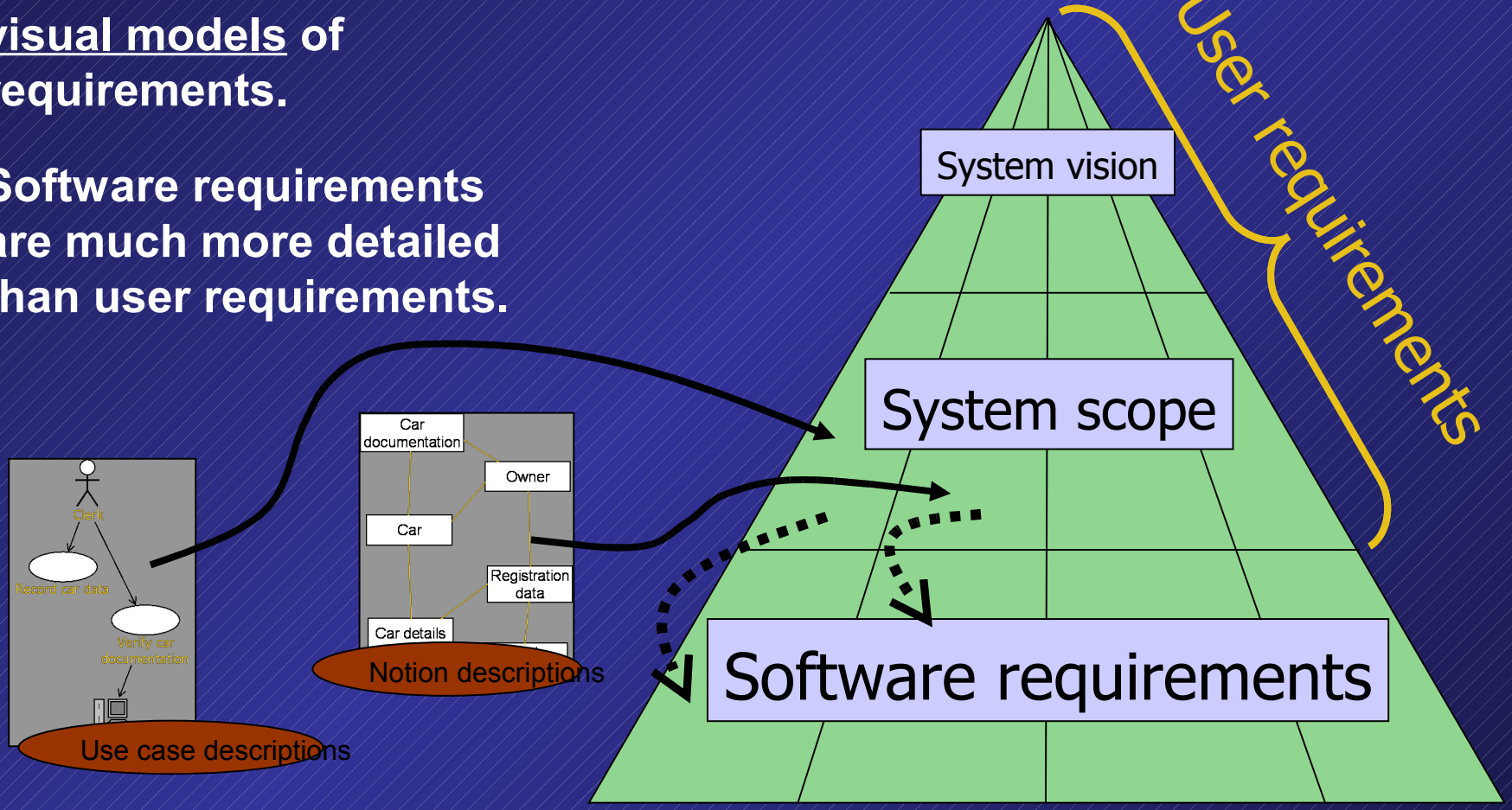
Lecture 4: Software requirements – structure and associated process

- SRS and an iterative process
- Criteria for use case prioritization
- SRS and acceptance testing
- SRS and user documentation
- Software requirements documentation

Reminder - where we are?

We still want to create visual models of requirements.

Software requirements are much more detailed than user requirements.

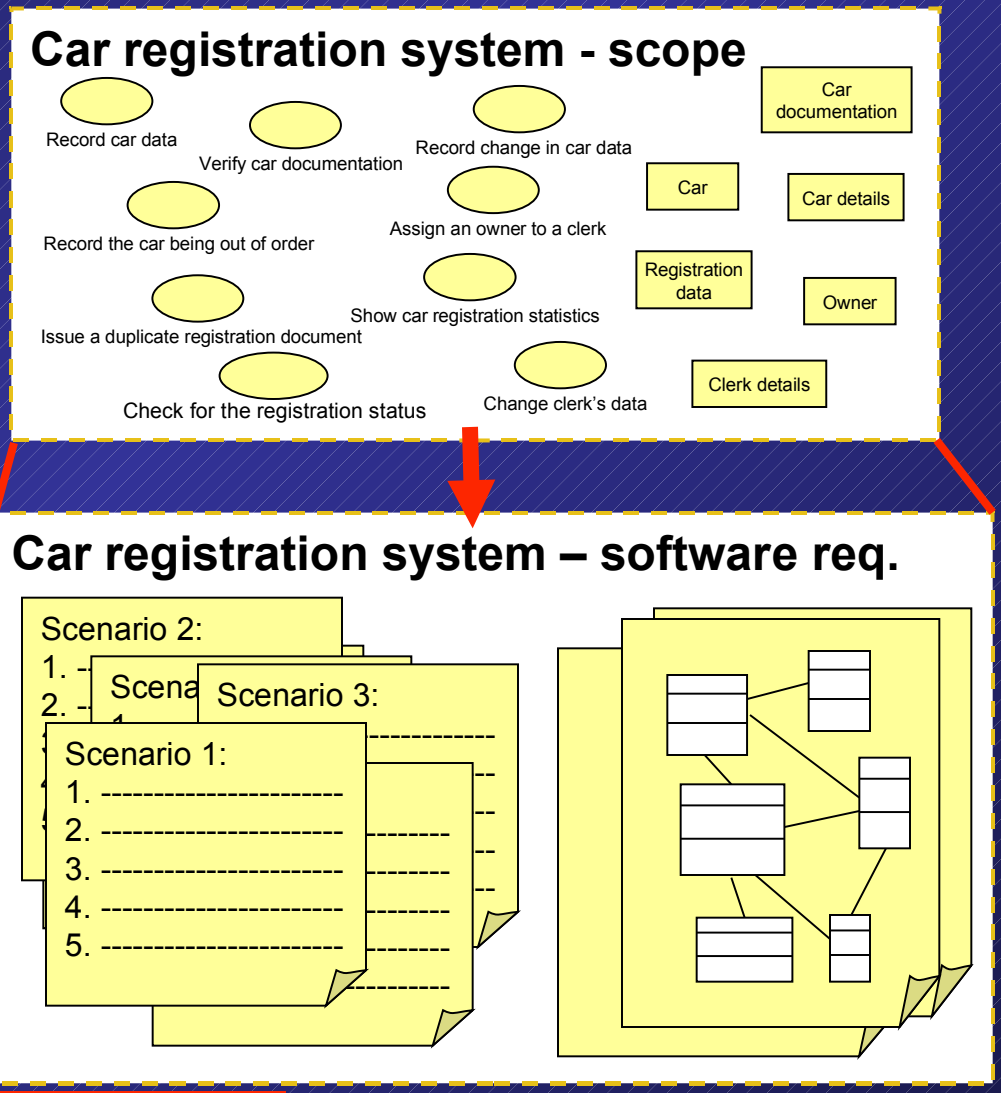


From system scope to software requirements

System's scope requirements determine mainly the size of the system.

Before developing the system we need to discuss detailed functionality and data characteristics of the system.

Software requirements describe all the details of how the system will function in dialogue with the actors and what data it will exchange.



Software requirements – what is the purpose?

Software requirements allow to answer three main groups of questions.

How the system shall function in detail?

- detailed scenarios for the system's dialogue with the users and other systems
- detailed "storyboards" containing description of the user interface dynamics

What data will the system handle?

- detailed description of data elements exchanged with the users and other systems
- detailed descriptions of menus, forms, messages through which data is exchanged with the users

What other detailed characteristics shall the system have?

- detailed description of non-functional requirements and constraints

Reminder: What is a use case?

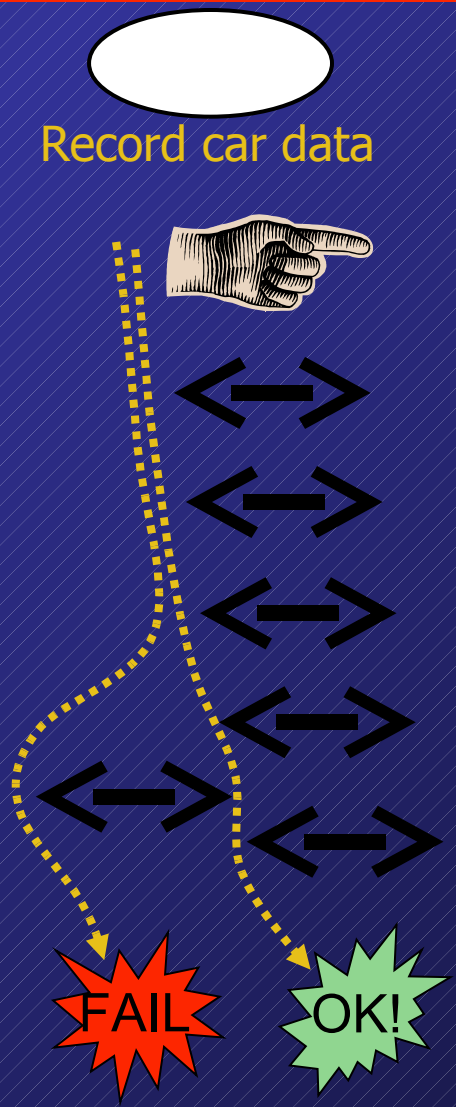
A use case is a description of behaviour of a system communicating with one or more actors.

In order for a behaviour description to constitute a use case three conditions have to be met:

- The description should start with an actor's interaction with the system
- The description should present messages exchanged between the system and an actor
- The description should clearly state the final goal reached at the end of message passing

After reaching the goal, it should be possible to start another use case (or repeat the current one).

Important: use case usually includes alternative paths of message passing that (try to) lead to the same goal but sometimes fail.



Use cases to scenarios

What is a scenario?

- A sequence of interactions between an actor and the system
- Starts with the actor's interaction
- Ends when the use case's goal is reached, but… it might end with a failure!

A use case is a group of similar scenarios.

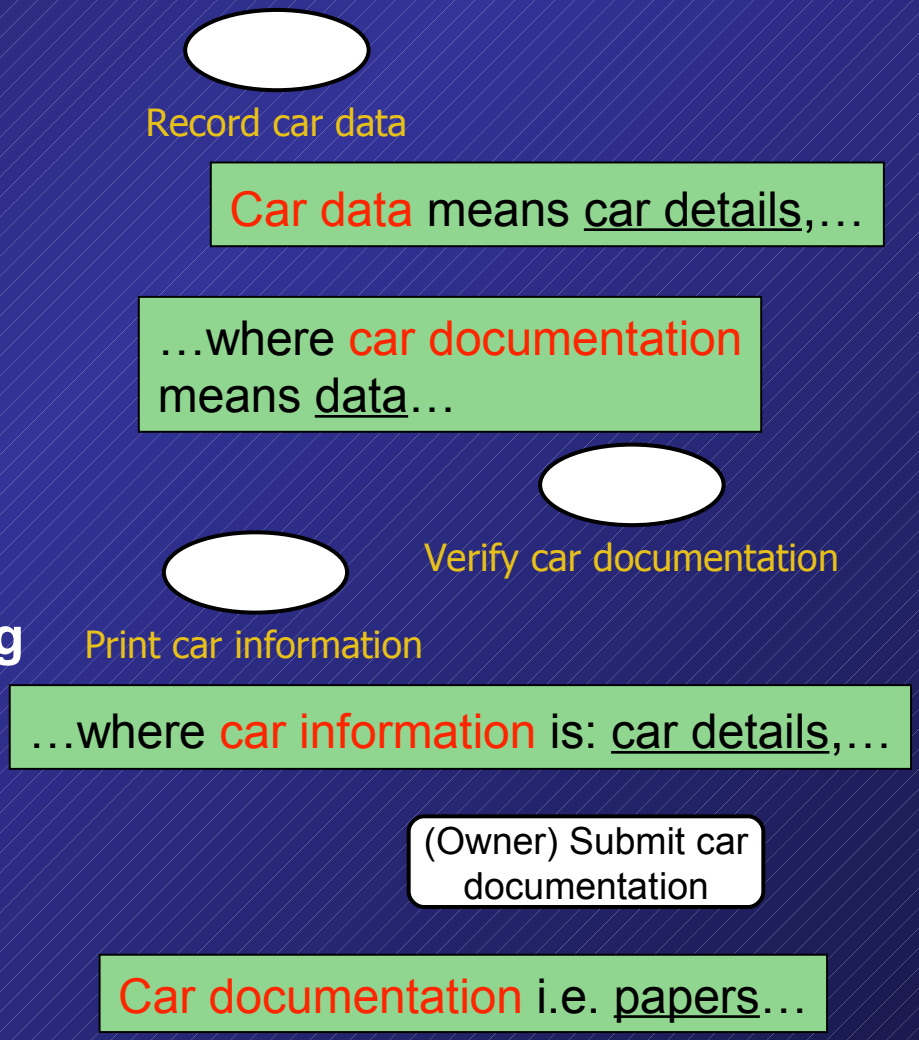
When writing software requirements we have to determine most of the possible scenarios (perhaps leaving out some obvious ones described in general at the beginning in the specification (e.g. cancel type scenarios).

Reminder: What is a vocabulary notion? What is it used for?

Vocabulary notions denote data that need to be handled by the software system.

Consistent vocabulary enforces coherency of language used in the model

- **no contradictory or overlapping terms**
- **precise and agreed-upon term definitions**



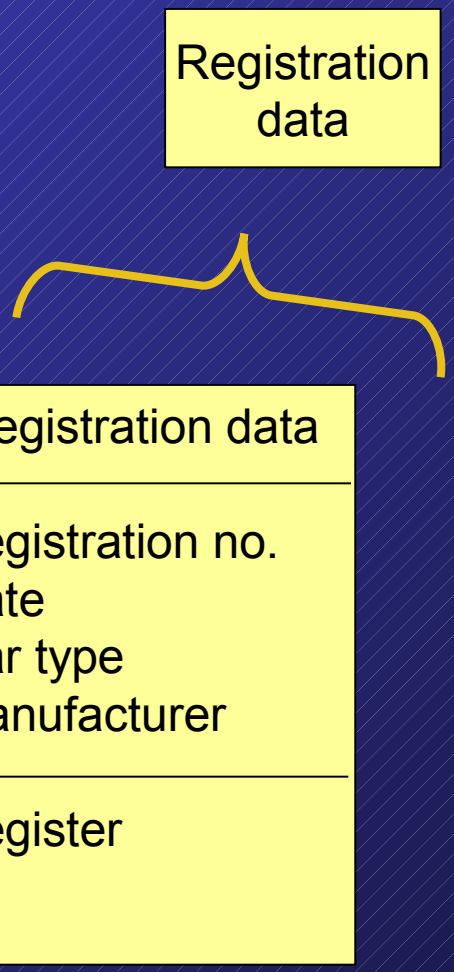
Vocabulary notions to detailed classes

When writing software requirements we transform vocabulary notions into classes.

Classes denote all the data and operations on that data that pertain to the modelled problem domain.

We use class names inside scenario sentences (as nouns).

Classes and relationships provide much more detailed information than just notion definitions.

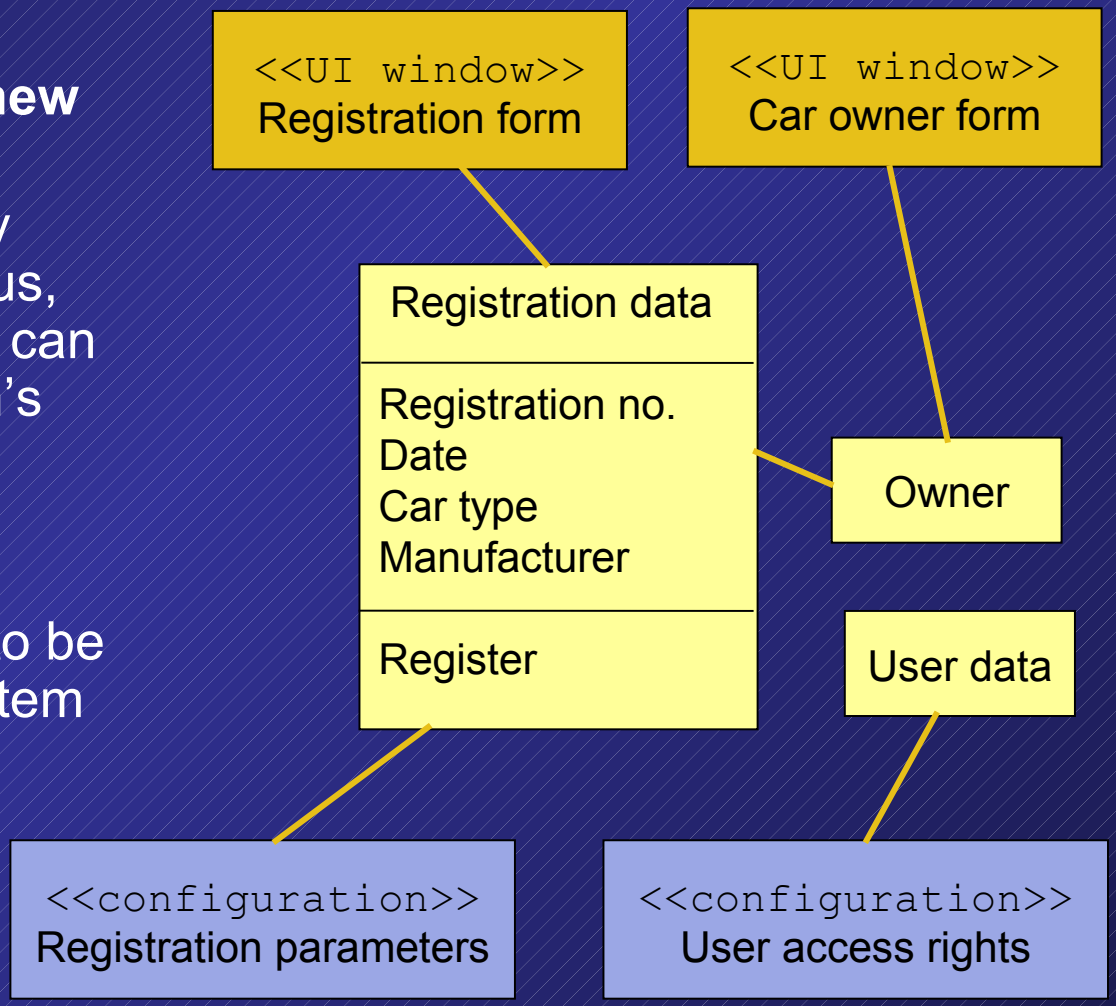


New notions – new classes

Software requirements specification introduces new notions:

- Window classes – they denote windows (menus, forms, messages) that can be found in the system's user interface
- Software configuration classes – they denote parameters that have to be set in order for the system to function properly

These new types of classes can be denoted with stereotypes.



Storyboards – simulating the system

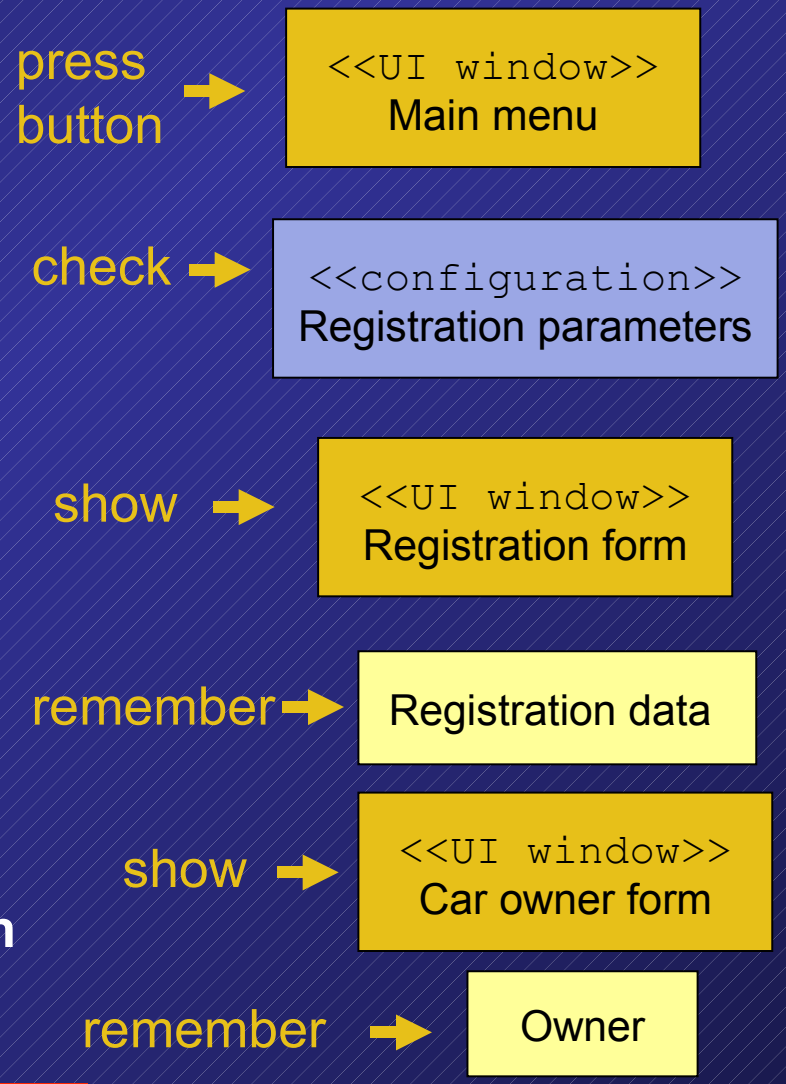
Storyboards contain information about the scenario, plus:

- Information about domain classes
- Information about window classes
- Information about configuration classes

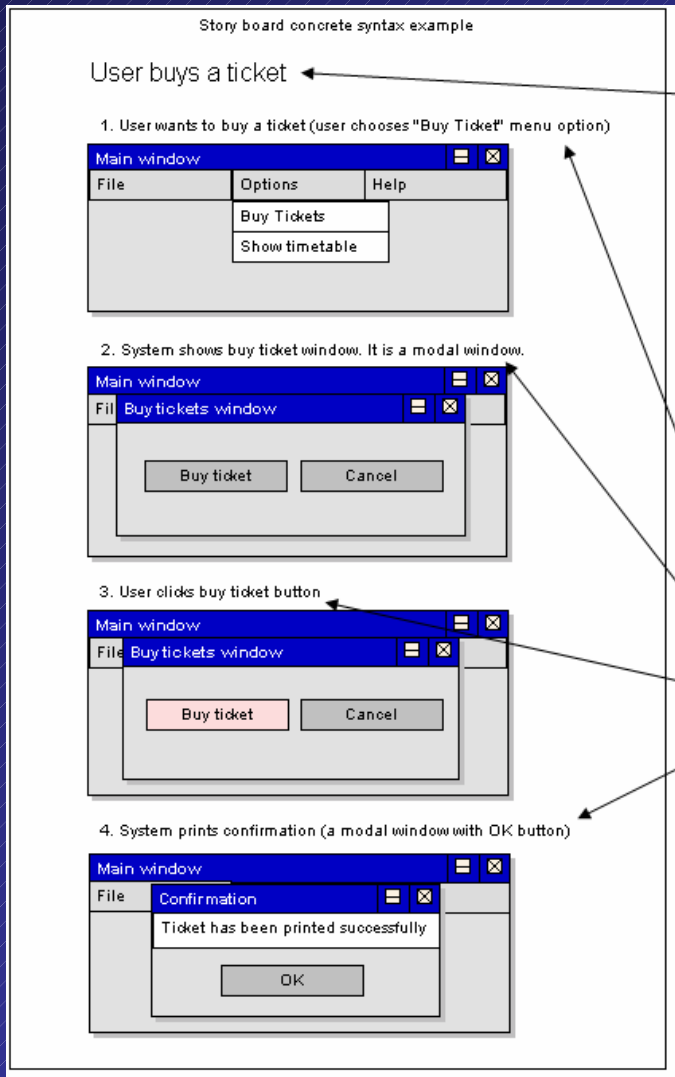
Storyboards are the artefacts where dynamics of the system meets its static description.

Storyboards are important for understanding the system's behaviour.

Storyboards facilitate discussion with the prospective system's users.



Storyboards – another visual language



UseCase name

User buys a ticket
1. User wants to buy a ticket.
2. System shows buy ticket window.
3. User clicks buy ticket button.
4. System prints tickets.
5. System prints confirmation.

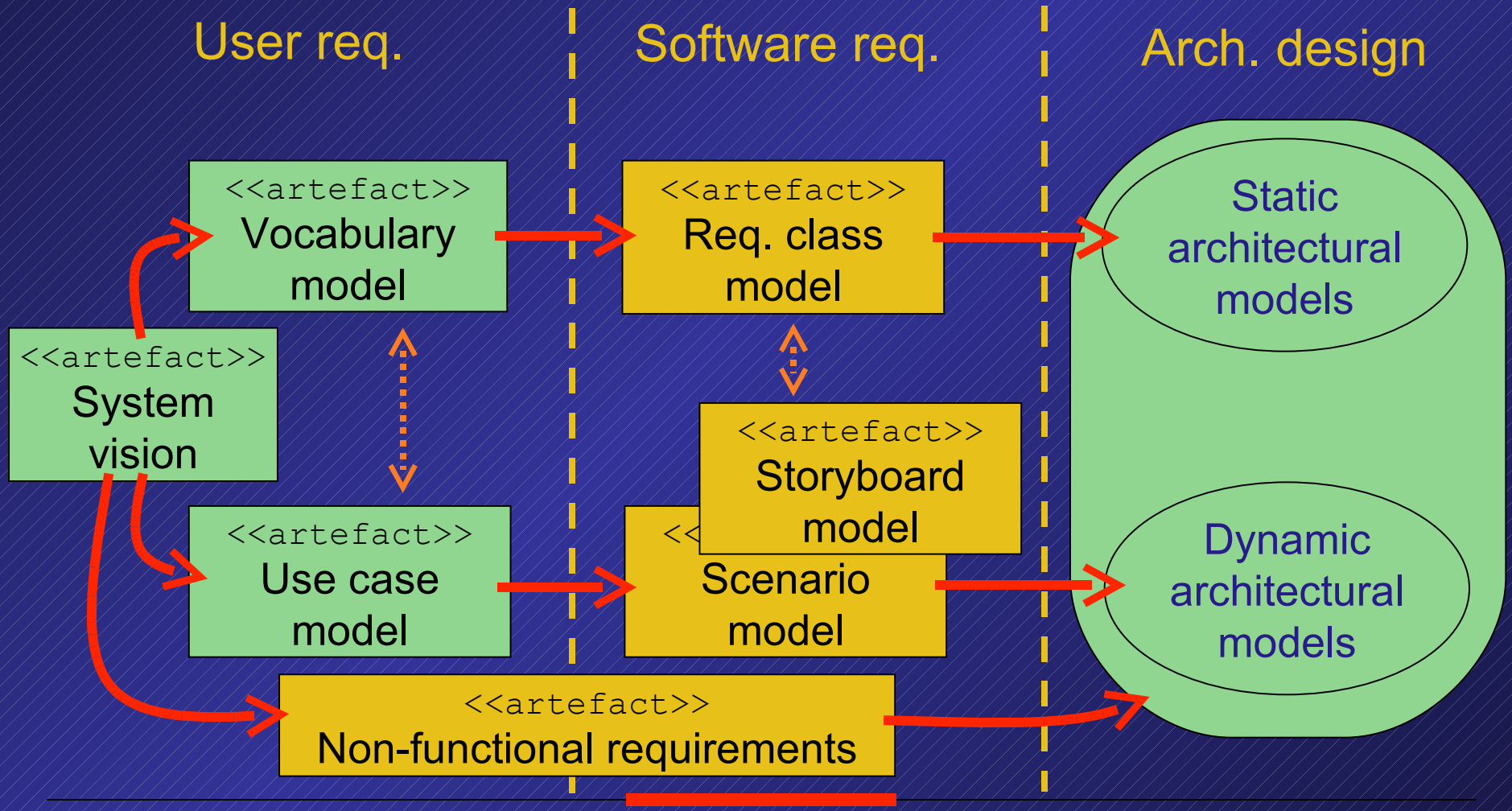
scenario sentences

UIScenes' descriptions with screenshots below them

source: RSL specification

Software requirements' artefacts as part of the development process

Software requirements "translate" the system scope into architecture.



Software requirements in the development lifecycle

Software requirements are the foundational part of the development cycle (iterations).

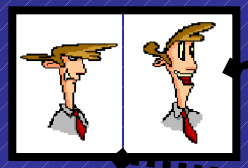
All other development efforts are performed on the basis of use case scenarios and requirements class model.

The final system is tested by comparing it (basically) to software requirements.

User's business needs

User Requirements

Software Requirements



?

TP

several times

ADP

DDP

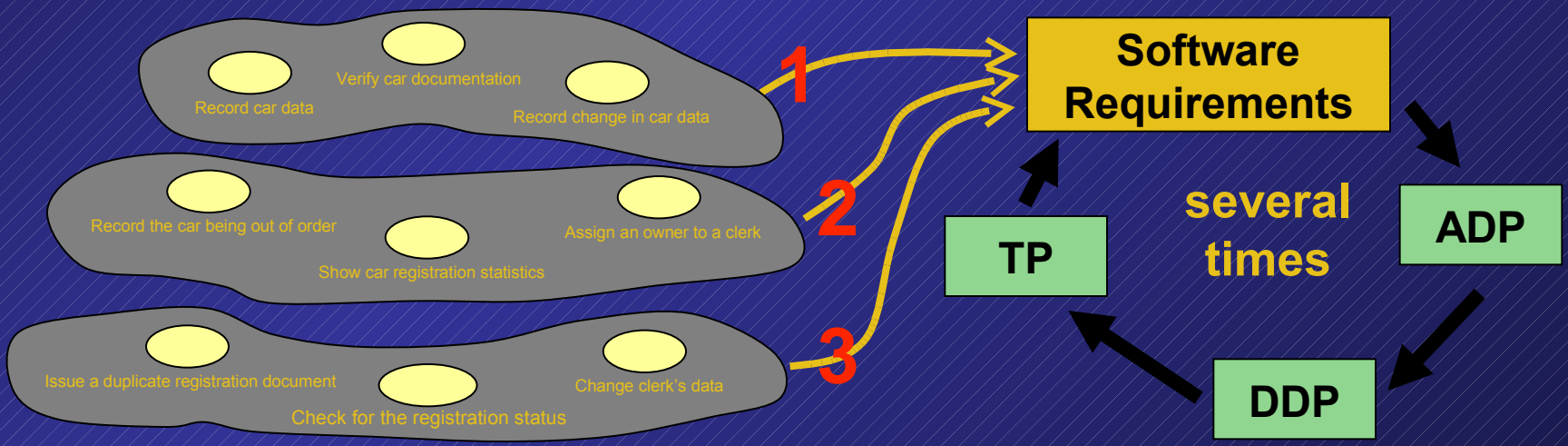
Delivered system

Prioritizing use cases – putting order in software requirements

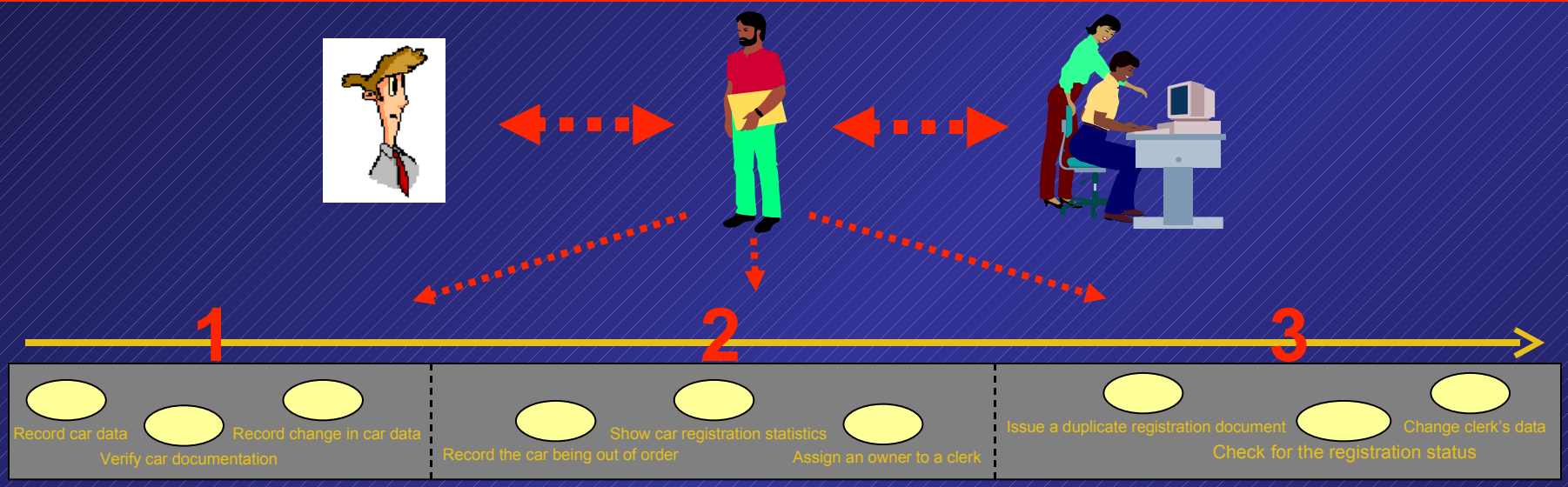
Before writing software requirements we need to give them priorities. We mainly prioritize functional requirements – use cases.

Priorities should be given to determine the order of development. Remember: the system should be developed "use case after use case". The number of priorities should reflect the number of iterations.

It is important to implement high priority requirements first! This reduces the risk of the project being unsuccessful.



User and technical priorities of use cases

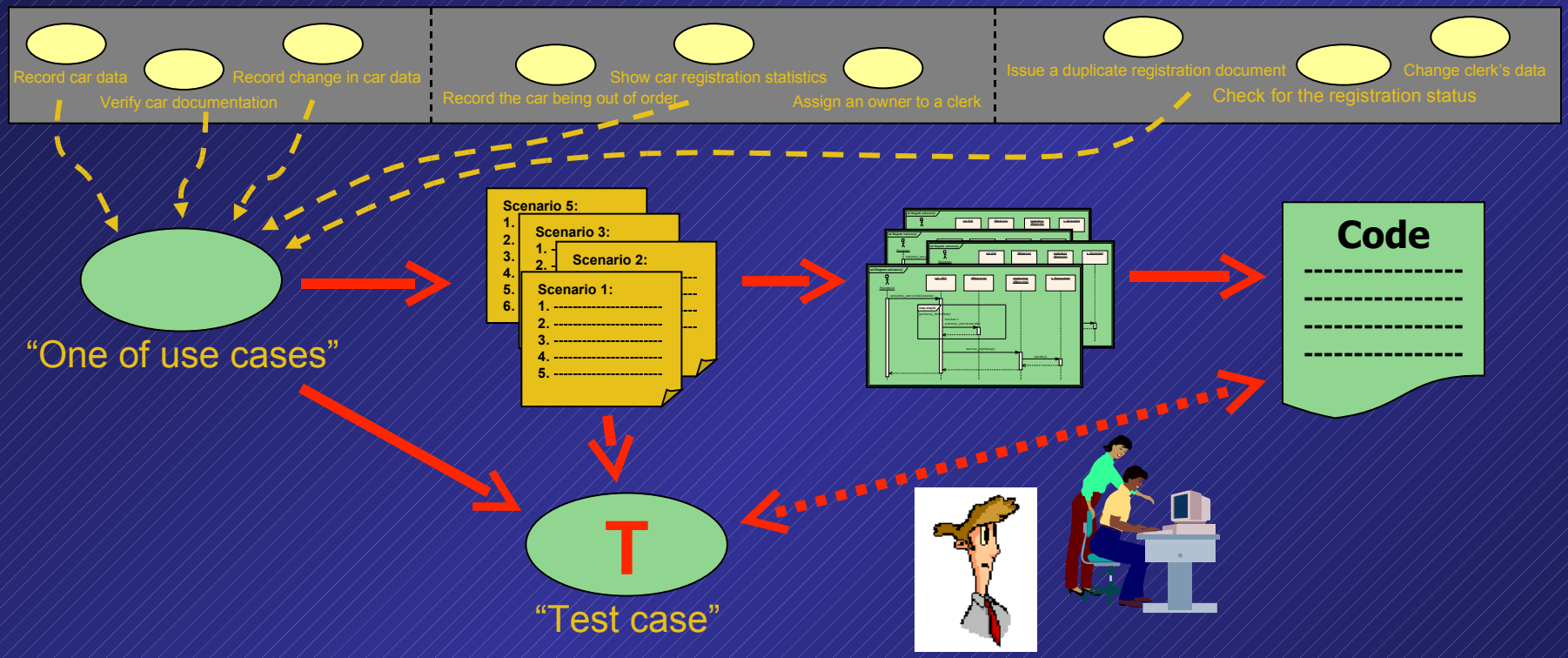


Requirements writers should talk both to the clients (users) and developers (architects) about use case priorities.

Every use case should be classified from the point of view of importance to the users and technical difficulty.

The resulting assignment of use cases to iterations should constitute a balance between these two points of view!

Realization of use cases



For use cases we write: scenarios and storyboards, then we develop interaction diagrams and write code.

On the basis of use cases we write test cases to verify formally that the system complies with the software requirements.

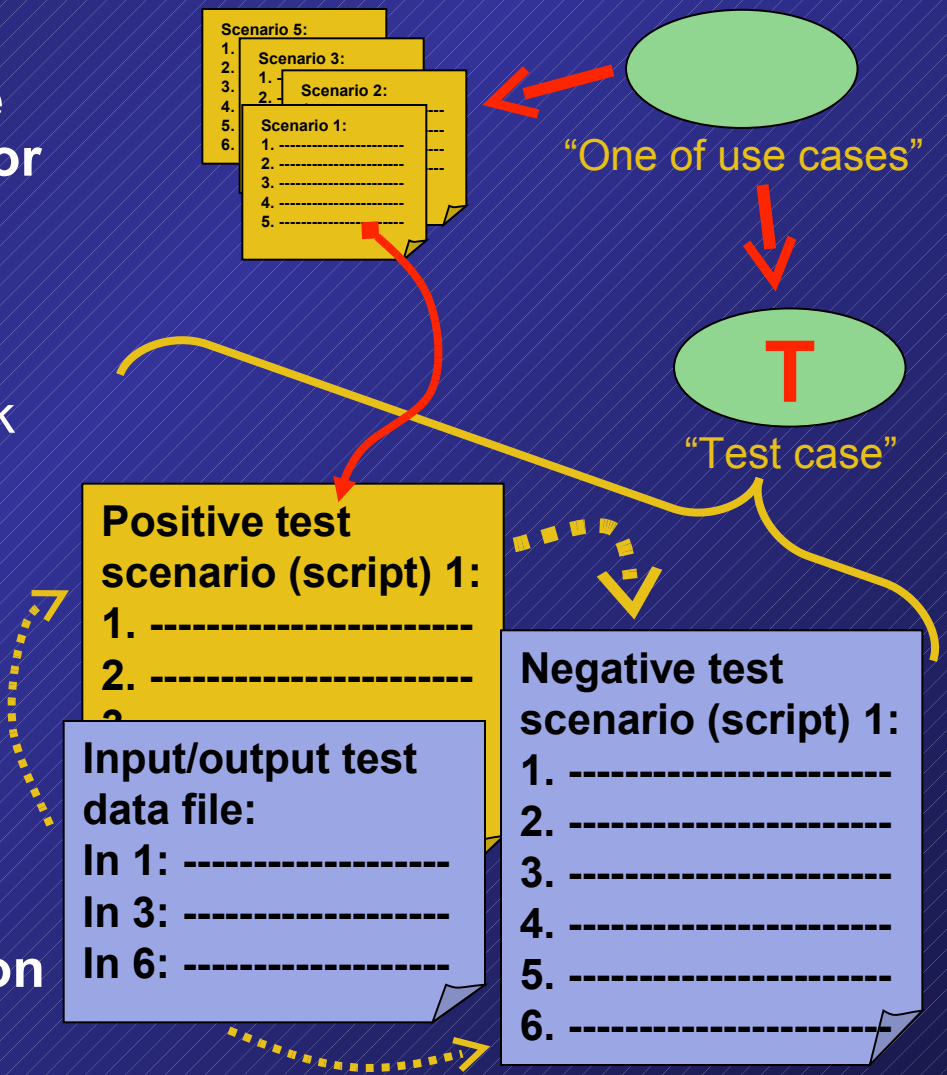
Use cases and scenarios to test cases

Test cases should be based directly on use cases. Every use case scenario should have one or more test scenarios:

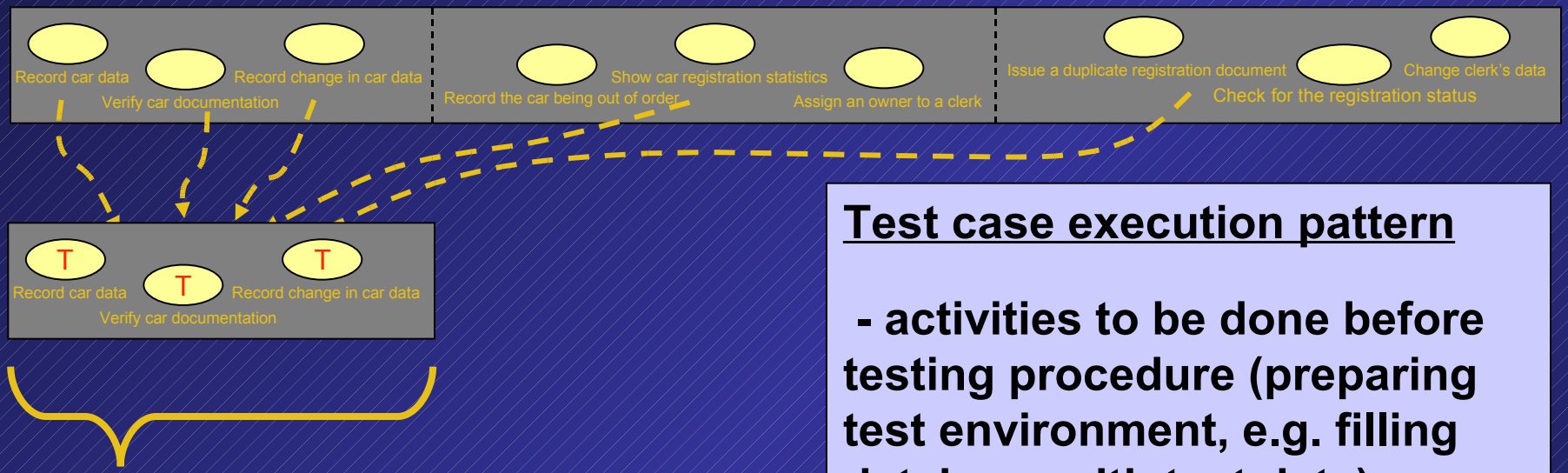
- Positive test scenarios check for desired system behaviour
- Negative test scenarios check for undesired system behaviour

Test scenarios contain the sequence of events (like in normal scenarios) plus description of test data.

Test scenarios and data files might be used for test automation (writing automatic test scripts).



Organizing tests



A set of test cases – a "test suite"

Every test case is executed in an isolation (as a unit – hence the „unit testing"). Other parts of the system or external systems are represented by mock-up objects, method stubs etc.

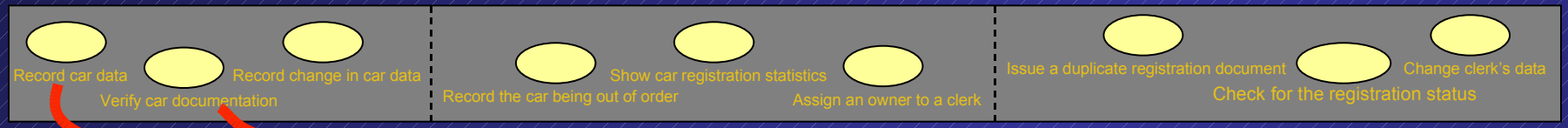
Test case execution pattern

- activities to be done before testing procedure (preparing test environment, e.g. filling database with test data)

- test procedure (reproducing the steps of the scenario path)

- clean-up (reverting the environment to the original state)

Use case models to user manuals



User manual can have the following structure:

- Use cases denote sections on "how to do this or that with the system"
- Scenarios describe the "how to" of the system
- UI window classes describe the "look" of the system

Well written scenarios and storyboards save a lot of work when writing user manuals!

2.2 How to verify car documentation

Verify car documentation

2.1 How to record car data

Record car data

Scenario 1:

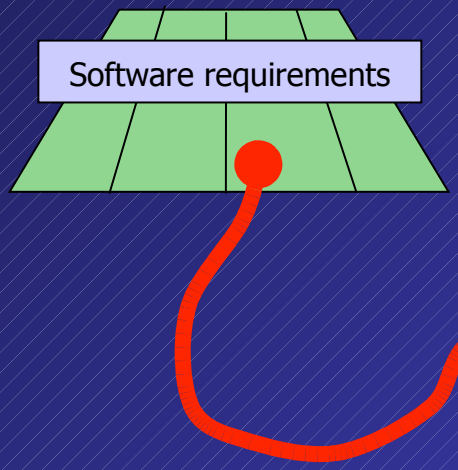
1. ----------
2. ----------
3. ----------
4. ----------
5. ----------
6. ----------
7. ----------

Scenario 2:

4. ----------
5. ----------
6. ----------
7. ----------
8. ----------

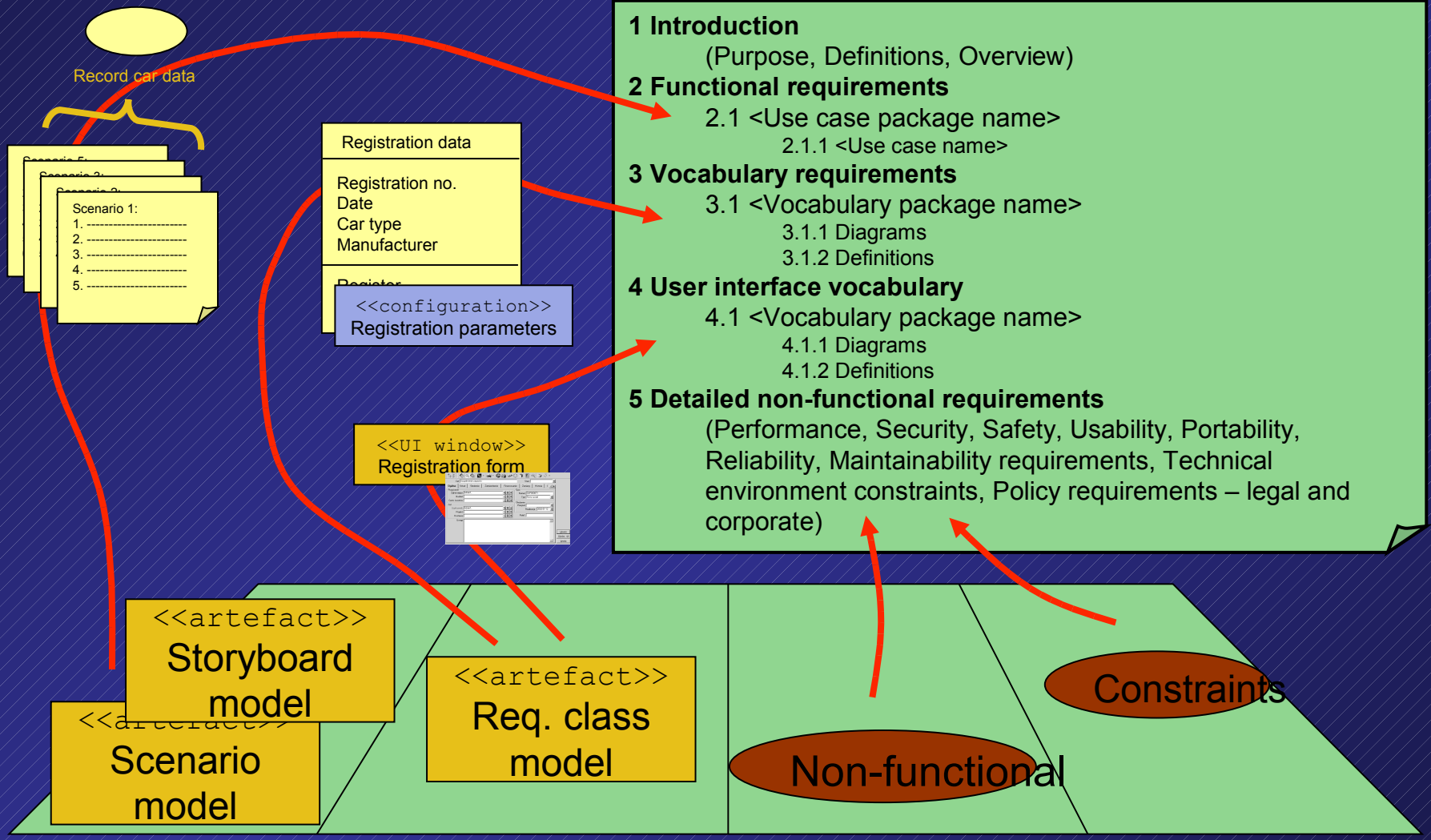
Summary: documenting it all…

All the software requirements form a model that can be expressed in a single document: Software Requirements Specification.



1 Introduction
(Purpose, Definitions, Overview)
2 Functional requirements
2.1 <Use case package name>
2.1.1 <Use case name>
3 Vocabulary requirements
3.1 <Vocabulary package name>
3.1.1 Diagrams
3.1.2 Definitions
4 User interface vocabulary
4.1 <Vocabulary package name>
4.1.1 Diagrams
4.1.2 Definitions
5 Detailed non-functional requirements

Summary: how will WE do it? (1)



Summary: how will WE do it? (2)

We will perform three iterations!

First: prioritize use cases!

Then: write scenarios and storyboards (domain classes, UI windows) in three increments.