

Software Requirements Engineering

Three main tasks in RE:

- ① *Elicit* — find out what the customers really want.
 - Identify stakeholders, their goals and viewpoints.
- ② *Document* — write it down (Requirements Specification).
 - Understandable
 - Precise
 - Complete
 - Consistent
 - Unambiguous
 - Modifiable
 - Correct
- ③ *Analyze* — check for above qualities.

Requirements Elicitation

Key problem areas:

Scope — what is the boundary of the system?

Understanding

- What is needed
- What is possible/practical
- Difficulties in communication of concepts/ideas
- Domain knowledge
- Conflicting needs or priorities

Volatility — Requirements change over time.

Collaborative Requirements Gathering

- Meetings with designers, customers and other stakeholders.
- Rules for preparation & participation.
- Agenda.
- Facilitator.
- “Definitions mechanism” (work sheet, flip charts, wall stickers etc.)
- Goals:
 - identify the problem,
 - propose elements of the solution,
 - negotiate different approaches,
 - specify preliminary set of solution requirements.

Example sequence

- ① Stakeholders write and distribute “product request”
- ② Pre-meeting, each participant identifies
 - objects in system
 - objects in environment
 - services (processes or functions)
 - constraints
 - performance criteria
- ③ Meeting
 - present lists
 - combine, move etc. to agree on consensus lists
 - refine definitions for each word or phrase (objects etc.)
 - maintain list of issues raised that aren't resolved.

Requirements Specification

Essential parts of an SRS:¹

① Introduction

- ① Purpose — of this document, including intended audience
- ② Scope
 - ① what will be produced
 - ② what will it do
 - ③ application, benefits, objectives and goals
- ③ Definitions, acronyms and abbreviations
- ④ References
- ⑤ Overview — what is to follow and how is it organized

¹From IEEE Std 830-1998 Software Requirements Specifications

SRS Essential parts (cont'd)

- ② Overall description — general factors that affect the product (not detailed requirements).
 - ① Product perspective — how does it relate to other products?
 - Interfaces (system, user, hardware, other software, communications)
 - Memory constraints
 - Operation constraints (could be part of UI)
 - Site adaption requirements — how will product be adapted for specific sites?
 - ② Product functions — description of major functions. Organized in logical way (for customer).
 - ③ User characteristics — what do we know about the intended users?
 - ④ Constraints — any other items that will limit design decisions.
 - ⑤ Assumptions and dependencies
 - ⑥ Apportioning of requirements — what may be delayed until later?

SRS Essential parts (cont'd)

③ Specific Requirements

- All requirements
- Detailed enough to enable designers to design a satisfactory system and for testers to test it.
- All must be externally perceivable (e.g., by users, operators or other systems).
- Must include description of every input/stimulus and output/response and their relations.
- Each requirement should be uniquely identifiable.
- Each requirement should be verifiable.
- Organized for readability.

Use Cases

- A technique for capturing functional requirements of system.
- Describe typical interactions between users (*actors*²) and the system.
- UML does not specify a standard presentation format or notation for Use Cases.
- Note that use cases are quite distinct from use case diagrams, which illustrate relationships between use cases and actors.

²An *actor* is actually a role that a user may play w.r.t. the system.

Scenario Example

Consider first a *scenario*—a particular sequence of events in the interaction between the user and the system.

Consider Automated Banking Machine example:

Goal: *Withdraw money.*

The customer inserts the bank card in the ATM card slot and is prompted to enter her personal identification number. The system checks that the PIN is correct and then prompts the customer to select a transaction type, account and amount. The user selects a withdrawal. The system verifies that the withdrawal is permitted (e.g., funds are available), dispenses the money and returns the card.

Scenario (cont'd)

- There are many possible variations on this sequence (e.g., PIN is not correct) so there will be many scenarios with the same goal.
- A use case describes a set of scenarios with a common goal.

See: Transaction

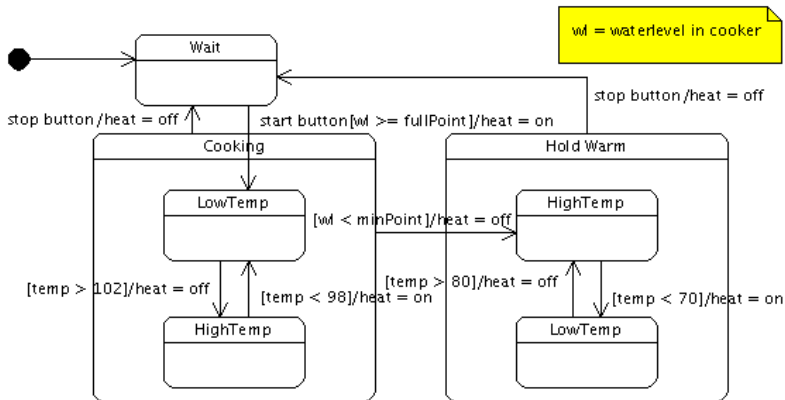
Use Case tips

- Always keep in terms of external view (problem domain).
- Name should be short active verb phrase.
- Steps clearly describe action by actor or system.
- Other information:
 - entry condition(s)** Condition(s) that must be true before a use case can begin. Also known as *pre-condition*.
 - exit condition(s)** Condition(s) that the system will ensure is true at the end of the use case. Also known as *guarantee* or *post-condition*
 - trigger** Event that gets the use case started.
- See also useful links on the web page.

State Diagrams

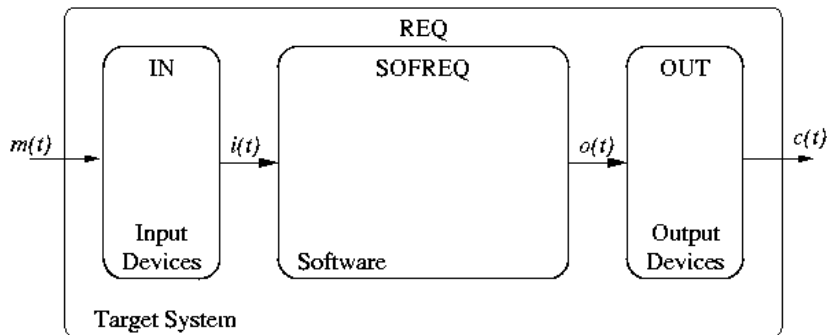
- Shows behaviour in terms of events and states
- Events occur at instants, time elapses in states
- States can be nested either orthogonal (parallel) or not.

State Diagram Example



Created with Poseidon for UML Community Edition. Not for Commercial Use.

Four Variable Model



SCR Concepts

Controlled variables quantities external to the system whose values are to be changed by the system.

Monitored variables quantities external to the system whose values affect the current or future behaviour of the system.

Output variables quantities internal to the system whose values represent controlled variables.

Input variables quantities internal to the system whose values represent monitored variables.

SCR Concepts (cont'd)

Condition predicate on past or current values of variables.

Event instant when one or more relevant conditions change value.

Event class a subset of the events relevant to a particular system.

History initial conditions plus sequence of events.

Mode a set of system histories that are equivalent in some respect.

Mode class a set of modes that partition the possible histories (i.e., every history is in exactly one mode in a particular mode class).

Controlled value relations define the acceptable values of each controlled quantity for any possible history.

- Will often use modes in one or more mode classes.
- Tabular expressions to aide readability.

Event Class Notation

Notation		Interpretation
scalar	tabular	
$@T(p_i)$	p_i $@T$	p_i becomes true
$@F(p_i)$	$@F$	p_i becomes false
WHILE(p_i)	T	p_i remains true
WHILE($\neg p_i$)	F	p_i remains false
WHEN(p_i)	t	p_i true before
WHEN($\neg p_i$)	f	p_i false before
	t'	p_i true after
	f'	p_i false after
CONT(p_i)	—	p_i doesn't change
	*	<u>true</u> (don't care)
	\emptyset	<u>false</u> (can't happen)

Example Mode Class Definition

Mode Class : Cl_{probe}

Modes : Md_{test} , Md_{pulse}

Initial Mode : Md_{test}

Transition Relation :

Mode	Event	New mode
Md_{test}	@T (${}^mPulse = {}^sDown$)	Md_{pulse}
Md_{pulse}	@T ($Since(@T ({}^{Md}pulse)) > 100 \text{ ms}$)	Md_{test}