

## Common mistakes in GA3

Here are some of the common issues in groups' SRSs that Jörn Guy noted down and we thought were important to point out, to improve your understanding of how to produce a consistent and useful UML model and SRS.

### Introductory Material

The introduction needed to mention the requirements to connect a travel planning service, and a mapping service. The introductory material of an SRS must identify who has written the document, who has revised it, and list secondary material necessary to read it if this goes beyond the UML standard.

Common Faults:

- Missing one or more of the services
- Describing the document as a *design*, rather than a *requirements* specification
- Diagrams without legends
- Technical details that were unmotivated [by client needs or a creative requirement]

### Analysis Models

At the core of the specification are formalised *models*. In the case of this SRS, they are defined in UML. Their purpose is to be understandable by both clients and implementers. They cover the functional requirements and display them consistently through a number of diagrams and textual representations. It is important to understand that all the diagrams are *connected*.

These diagrams are *not* intended as implementation blueprints, but as guides to understanding and as a means to distinguish between a correct implementation and an incorrect one. All correct implementation must behave according to the requirements described.

### *Class Model*

At the heart of the requirements-specification is a class model that interconnects *all* the functional concepts that the specification deals with. If a term isn't present in the class model, it shouldn't appear in any of the other views. (One of the most common forms of inconsistency is where there the other views use terms that do not appear in the class model.) So, the class model is particularly important.

The class diagram can either show only the persistent classes in the class model, or it can show the non-persistent classes as well. (The non-persistent classes can be omitted if the Use Case diagrams provide the signatures for them.)

### Diagram Syntax – Grammar Check

Common syntactic issues found:

- Missing (undefined) cardinalities
- Cardinalities inconsistent with relationships, e.g. 1..\* for owner of an aggregation
- Cardinalities defined on generalisation relationship

- Wrong/Missing method signatures (e.g. the number of parameters was different in behaviour diagrams (sequence, state, activity) and the class diagram)
- Wrong compartment used

## **Diagram Semantics – Meaning of the Content**

Common semantic issues found:

- Classes that represent technical/implementation artefacts (for example System, WebInterface, Database)
- Missing Concepts (for example interest group list administration, Mail system, Transport System, Mapping System). You can cross-check this by seeing whether the items appearing in your other diagrams all appear in your class diagram.
- Classes that Emulate UML concepts (XxxxList)
- Cardinalities that are too loose
- AssociationEnds with generic names
- Aggregation cycles
- Generalisation cycles

## **Diagram Syntax – Readability and Intent**

Are the diagrams consistent with the intent of the presentation? Common presentational issues:

- Empty Compartments
- Missing role names / Missing association names
- Generic role names
- Visibility modifiers
- Too many classes in diagram (if your diagram has more than 9 classes you should probably break the diagram up)
- Use of Interface notation for technical purposes

## ***Use Case Model***

The Use Case Model defines the interfaces of the system through interactions with the system. The diagram is so simple, that most syntactical errors have semantic consequences. However, additional issues occur in the text description.

## **Diagram**

The following mistakes are common in the UC diagram

- Functional decomposition: The use case does not model a business case, but an implementation or system execution step.
- System Actors missing (email system, mapping system, transport system)
- Actors arranged in composition instead of generalisation: Two or more actors are connected to a use case. In the semantics of UML, this implies that they **all** need to be connected to the system for the use case to be performed. Often, the authors meant to use a *hierarchy* of actors instead.
- Generic Actor: SRSs that have only one Actor (called “User”)

## **Description**

The following issues are common in the UC description

- Generic Actor: The only actor that exists is ‘User’

- Generic Information: The use case text does not reference the classes of the class diagram, or their methods or properties.
- Missing requirement: the use case fails to mention obligations on other system actors (e.g. system using email to confirm the identity of an academic user on sign-up, or ownership not checked on delete of a talk)
- Technical artefacts: The Use case text refers to specific UI implementation artefacts like 'link', 'form', 'page', click... or other technical implementation artifacts like 'database', 'buffer'...
- Alternative flows: The alternative flows do not merge back into the main flow as required.
- Missing precondition: The use case can start even though a necessary prerequisite is not fulfilled

### ***Sequence Diagram***

The sequence diagram is used to display a specific scenario for the client. It is usually used where an interesting interaction spans several domain objects. A sequence diagram that is attached to a use case provides one example of how that use case plays out and what must happen with the domain objects in any implementation. As such, the sequence diagram must be specific, and is usually wide as it includes the particular domain objects involved in the use case. The opposite is a *generic* sequence case diagram, that only includes an arbitrary user and system. Common issues:

- Generic sequence diagram – contains 'user', 'interface', 'system'
- Technical sequence diagram – contains 'form', 'page', 'buffer', 'database', 'file'

### ***Activity Diagram***

The activity diagram is used to describe an algorithm or workflow that spans a set of classes. An activity diagram is useful when describing what effect a use case has inside the system. This implies that the activities must be mapped to methods in the design model. Otherwise, it is not possible to interpret the diagram consistently.

There are two different semantics for activity diagrams, state diagram semantic and petri net semantic. They are compared in "A Formal Semantics for UML Activity Diagrams" ( Eshuis, Rik and Wieringa, Roel (2001) ) <http://doc.utwente.nl/37504/1/0000004e.pdf>

Some common issues are:

- Inconsistency with Use Case description
- Inconsistency with Class model
- Decision branches with unlabeled conditions
- Parallelism branches without required joins

### ***Statechart Diagram***

The state chart diagram can be applied to describe two elements of the requirements specification: The state-space of the use case, and the state-space of domain objects. In both cases, the diagram has to match the signature of the underlying classes. Often the distinction between the *state* diagram and the *activity* diagram was not clear. Common issues:

- No associated class or use case (interface)
- States are associated with the system, rather than with particular domain objects.

- No guards on branches
- State diagram for use case without an end

## General Description

Beyond the Diagrams, the following items should have been present and often weren't:

- A description of the contents of the class model in table or text form.
- A list of references, including proper formatting and time and date information on web addresses if used
- A section on non-functional requirements
- References to standards used in the description

## Presentation

The SRS is essentially meant to be unambiguous, so the following are important:

- Correct spelling, (you should use your word processor's Spell-Check functionality)
- Correct grammar, (you should use your word processor's Grammar-Check functionality)
- Clear and readable English (some word processors can check this for you using the Flesch–Kincaid and Gunning-Fog index)
- Minimal use of the passive voice
- Page and section numbering ("Page 5 of 43" is a helpful format)
- List of tables and figures
- Hyperlinking (optional, but helpful if the document will mostly be read in electronic form)

## Excellence and Innovation

There are any number of ways a group can show "excellence and innovation" in their work.

There were many inventive additional functional requirements that groups came up with, though some SRSs didn't justify what the benefit of those additional requirements would be.

Other forms of excellence and innovation came in the area of methods. Below are a few examples for these:

- Defining the terminology they used in their requirements (RFC 2119)  
<http://www.ietf.org/rfc/rfc2119.txt>
- Including a sign-off list for clients in the front of the document
- Explicit requirements lists using a custom notation
- Appending various prototyping contents as a non-normative Appendix.
- Version numbers in sections
- Description of templates (for Use Cases) and formalisms used/amended