Misuse of use cases

- Tim Korson: Object Magazine, May 98, page 18
- Important for white-box testing: Requirements are often expressed in the form of use cases which are source of test requirements

Bad example

- Large project
- Framework team asked application developers: “send us your use cases”
- 12,386 use cases were received
  - were too detailed and full of errors
  - at wrong level of abstraction

Requirements should be organized hierarchically

- Levels of abstraction are also needed for the requirements: use cases need to be organized into layers. Reasons:
  - manage complexity of requirements
    - top-level requirements should be expressed in no more than 12 use cases
    - a layer of use cases should have at most 5-10 times the number of use cases of previous layer

Requirements should be organized hierarchically

- Why layers of uses cases?
  - Complete set of requirements for “testing” at appropriate level of abstraction. Test domain, application, architectural and detailed design models.

Business requirements/ Development deliverables

Requirements Artifacts	Development Artifacts
Business Requirements –first few levels of use cases	Domain Models
Interface Specifications	Application Models
	Architectures
More details	Detailed designs
Complete detailed specifications	Source code

Use case layers

- Each level of use cases should be complete
- No new categories of requirements in lower levels
- Develop use cases iteratively and incrementally

Hierarchical classification and functional decomposition

- Are different!
- Use case 1.1 is not the first step of use case 1.
- Use case 1.1 is a specific, more detailed, use case within the category of use cases defined by use case 1

Business requirements/ Interface specifications

- Keep them separate.
- First express business requirements in interface-neutral terms
- First level of requirements should not jump directly to interface specifications, otherwise
  - harder to identify other interfaces
  - designers not prompted to add extensibility

Example

- “deposit coin”
- Is interface binding for “accept payment”
- Could consider other interface bindings, such as electronic cash

Design and use cases

- Do not derive designs directly from use cases
- Use cases stop at the system interface boundary. Use cases describe sequences that actors follow in using the system
- Use cases never specify what steps the system takes internally to respond to a stimulus

Architecture

- A software system is a specific instantiation of an architecture customized to satisfy a specific set of requirements. Architecture was chosen because of
  - standard domain relationships
  - other system requirements: time, space, reliability, extendability, etc.

  and not so much because of functional reqs

Architecture instantiation

- Instantiation of an architecture to implement a set of functional requirements is documented by object interaction diagrams, and not by use cases
- www.software-architects.com