Favorite sentences from Jalote

- "The detailed requirements section describes the details of the requirements that a developer needs to know for designing and developing the system." (pg 47)
- "Use cases specify the functionality of a system by specifying the behavior of the system, captured as interactions of the users with the system." (pg 49)
- "The third common requirements error is *incorrect fact*. Errors of this type occur when some fact recorded in the SRS is not correct." (pg 64)

Jalote on Good SRSs

- “The origin of most software systems is in the needs of some clients.” (pg 38)
- “The problem is that the client usually does not understand software . . . and the developer often does not understand the client’s problem. . . ” (pg 38)
- “An error in the SRS will manifest itself as an error in the final system implementing the SRS.” (pg 39)

The **application domain** represents all aspects of the user's problem. This includes that physical environment, the users and other people, their work processes, and so on. It is critical for analysts and developers to understand the application domain for a system to accomplish its intended task effectively. . . .

The **solution domain** is the modeling space of all possible systems. Modeling the solution domain represents the system design and object design activities of the development process. . . .

Object-oriented analysis is concerned with modeling the application domain. **Object-oriented design** is concerned with modeling the solution domain. (Bruegge and Dutoit, pg 41)

The final output is the SRS document. As analysis precedes specification, the first question that arises is: If formal modeling is done during analysis, why are the outputs of the modeling not treated as an SRS? The main reason is that modeling generally focus on the problem structure, not its external behavior. Consequently, things like user interfaces are rarely modeled, whereas they frequently form a major component of the SRS. (Jalote, pg 41)

A new user who sits down to use a program does not come with a completely blank slate. They have some expectations of how they think the program is going to work. This is called the *user model*: it is their mental understanding of what the program will do for them. . . .
The program, too, has a model, only this one is encoded in bits and will be faithfully executed by the CPU. This is called the *program model*, and it is *The Law*. Nothing short of electrical storms and cosmic rays can convince a CPU to disobey the program model. (Spolsky, pg 8)

The **analysis model** is composed of three individual models: the **functional model**, represented by use case and scenarios, the **analysis object mode**, represented by class and object diagrams, and the **dynamic model**, represented by statechart and sequence diagrams. . . .

The analysis model represents represents the system under development from the user's point of view. The analysis object model is a part of the analysis model and focuses on the individual concepts that are manipulated by the system. . . . The dynamic model focus on the behavior of the system. (Bruegge and Dutoit, pg 175–176)

- Functional requirements
- Nonfunctional requirements
  - Usability
  - Reliability (dependability)
    - Robustness
    - Safety
  - Performance
    - Response time
    - Throughput
    - Availability
    - Accuracy
  - Supportability
    - Adaptability
    - Maintainability
    - Portability (Bruegge and Dutoit, pg 125–126)

Modify a Computer Record (Fox, pg 170)
Actors: Administrator
Stakeholders and needs:

- Administrator: To modify the database.
- Computer users: To have accurate data in the database.
- Accountants: To have accurate and complete data in the database.

Preconditions: The administrator is logged in and has a computer identifier.
Postconditions: The database is modified only if all correctness and completeness checks on the modified record succeed and the Administrator confirms the changes. Computer record edits are always saved unless the Administrator cancels the transaction.
Trigger: Administrator initiates a computer record modification transaction.
Basic flow:

1. Administrator initiates the transaction and enters the computer identifier.
2. CAS displays all data for the indicated computer.
3. Administrator edits the data for the computer.
4. CAS verifies the changes and asks for confirmation that they should be accepted.
5. Administrator confirms the changes.
6. CAS modifies its database and informs the Administrator that the transaction is complete.

(Continued from Fox, pg 170. . .)

Extensions:

- a Administrator cancels the operation: The use case ends.
- 1a The computer identifier is invalid:
  - 1a1. CAS alerts the Administrator of the problem.
  - 1a2. Administrator may Make a Query and correct the problem, and activity resumes.
- 3a Administrator directs CAS to execute a held transaction (see 6a):
  - 3a1. CAS modifies its database to complete the held transaction and informs the Administrator that the transaction is complete, and the use case ends.
- 4a CAS detects invalid or incomplete data.
  - 4a1. CAS alerts the Administrator to the problem.
  - 4a2. Administrator corrects the problem and activity resumes.
- 5a Administrator does not confirm the changes: The use case ends.
- 6a CAS is unable to modify its database:
  - 6a1. CAS records the transaction for later completion, informs the Administrator of the problem, and asks whether the transaction should be held.
  - 6a2. Administrator confirms that the transaction should be held.
  - 6a3. CAS verifies that it is holding the transaction and the use case ends.

Simple Use Case Writing Guide (Bruegge and Dutoit, pg 137)

- Use cases should be named with verb phrases, what the user is trying to accomplish.
- Actors should be named with noun phrases.
- The boundary of the system should be clear.
- Use case steps should be phrased in the active voice.
- The causal relationship between successive steps should be clear.
- A use case should describe a complete user transaction.
- Exceptions should be described separately.
- A use cases should not describe the user interface of the system.
- A use case should not exceed two or three pages

Consider a calendar application that allows users to categorize events by whether they are recurring or one-time and also by nature (school, work, personal, church, with various organization). The calendar can be viewed day-at-a-time, week-at-a-time, month-at-a-time, or as a continuous scroll. The view can show events only of a selected nature, or they can show events of many natures, color-coded. The user can also publish public views of the calendar containing events appropriate for public knowledge.
Get into groups of 2 or 3 and write use cases for adding, modifying, and deleting events and for viewing the calendar or publishing views.

Kent Beck: I coined the term User Story, as far as I know, so I'll tell you what I had in mind.
My purpose is to maintain a balance of political power between business and development. Use cases as I have seen them used are complex and formal enough that business doesn't want to touch them. This leads to development asking all the questions and writing down all the answers and taking responsibility for the resulting corpus. Business is reduced to sitting on the other side of the table and pointing.
I want a very different dynamic. I want business to feel ownership of and take responsibility for the care and maintenance of "the requirements". . . .
The idea of specifying the behavior of the system from an outside perspective, and using those specifications throughout the life of the system is the same. The execution is quite different.
(http://c2.com/cgi/wiki?UserStoryAndUseCaseComparison)

Jim Little: A user story is very simple and is written by the customer. It's incomplete, possibly inaccurate, and doesn't handle exceptional cases because not a lot of effort is expended making sure it's correct. ...
A use case is more complex and is written by the developer in cooperation with the customer. It attempts to be complete, accurate, and handle all possible cases. A lot of effort it expended to make sure it's correct. ...
My biased opinion is that user stories work better in any case that the customer is readily available. In my opinion, the use-case approach is wasteful since it tries to anticipate the questions that need to be answered, and that's naturally going to result a few questions being missed and a few more being researched unnecessarily. On the other hand, if the customer isn't immediately available, then the use case approach is probably better since it avoids the overhead of customer latency most of the time.
(http://c2.com/cgi/wiki?UserStoryAndUseCaseComparison)

Rachel Davies: It appears that Use Cases and XP Stories have a common purpose, to describe functional requirements.. . .
However, the purpose of the XP Story is not to document requirements but to enable incremental software development to proceed in an environment where requirements change is expected
. . .
In XP, Stories are by definition time-bounded (in estimated development time) to enable their complete implementation in a single iteration. In contrast, the scope of a Use Case depends on applying an abstract definition, concerning system interaction with external actors, to the development domain. (Davies 2001)

Frequently the client and the users do not understand or know all their needs, because the potential of the new system is often not fully appreciated. . . [analysts] also act as *consultants* who play an active role of helping the clients and users identify their needs. (Jalote, pg 58)