A rule of thumb for system analysis is that “The Model should focus on requirements that are visible within the problem or business domain”. It would seem obvious that if there are system requirements, that they ought to be included in the system model. However, the very reason that abstracted and graphical system models exist is to facilitate communication between stakeholders (Sommerville, 2011). When developing a system, communication with customers about the system requirements can be difficult and it is in this context that requirements may become “hidden”.

**Lack of Understandability**

One way in which requirements can become hidden is due to a lack of understandability (Sommerville, 2011). That is, the requirements might be specified in the language specific to the domain and therefore not fully understood by stakeholders outside of that domain. To clarify further, the jargon used by the customer in their particular domain may not be well understood by the software engineers who are building the system. Any circumstances where there is a lack of understanding of the requirements can lead to a failed system project.

An example of this comes from the train protection system in this week’s readings (Sommerville, 2011). One of the requirements given for the train protection system is that the computation for the deceleration of a train be given by the formula Dtrain = Dcontrol + Dgradient. Although this formula and it’s variables’ meanings might be obvious to experts in this domain, the software developers working on this project may not have a full grasp of the meanings of these variable and how they interact with other requirements (Sommerville, 2011). In this sense, the requirements and their interactions might not be fully visible and difficult to add to a system model.

**Implicitness and Ambiguity**

Another way in which requirements can be hidden is due to implicitness or ambiguities (Sommerville, 2011). That is, the customer may not express a requirement precisely, or worse at all, because they do not think that it needs to be explicitly stated.

An example of this comes from the patient records system (Sommerville, 2011). The user of the software system might want to do a search for a patient name that produces all of the patient’s appointments across several clinics. They may assume that if they include a “search” requirement in the system, that the search will produce the information they expect based on their domain experience. However, the software developer, who is not immersed in the domain may interpret the customer’s ambiguous requirement as searching for the patient name in only a specific clinic (Sommerville, 2011). Misunderstandings due to ambiguous requirements stemming from implicit knowledge of the domain can lead to a product not being what the customer intends.

**Conclusion**

The goal for requirements engineering is to determine the complete set of requirements for a system (Sommerville, 2011). A first step toward achieving a complete and consistent requirements document is to acquire information from the customer and other stakeholders. Unfortunately, communication with stakeholders is not always effective. There may be a lack of understandability or there may be ambiguities caused by implicit knowledge of the domain. Any of these situations can lead to requirements that are so-called “hidden”. Using a system model to facilitate communication among stakeholders as well as following processes to make the full set of requirements discoverable can prevent failed system projects.

**References**

Sommerville, I. (2011) "Chapter 4 - Requirements Engineering: Lecture 1-3.” Software Engineering, Ed. 9. Retrieved from <http://my.uopeople.edu/mod/resource/view.php?id=57336>

Sommerville, I. (2011) "Chapter 5 - System Modeling: Lecture 1-2.” Software Engineering, Ed. 9. Retrieved from <http://my.uopeople.edu/mod/resource/view.php?id=57337>