Abstracting Errors from Faults

1. Understand *why* each fault you identified represents a defect in the document.

   You should first think about the types of information that have been stated incorrectly in these faults. The crucial thing will be to identify the aspect of the system with which a fault is concerned. This may be a particular functionality, a user of the system, a performance requirement, etc. Once you can identify which parts of the system have faults associated with them, you can start to reason about what the larger mistakes might have been that have caused these faults.

   You may find the fault classification from Assignment 2A to be of some help in this regard. If information is found in the wrong section of the document, it might mean nothing more than that whoever wrote the requirements organized the document badly. On the other hand, if there is a contradiction within the document, then the requirements may reflect an underlying confusion about some of the functionality that is to be provided by the system. The distinction is not always so clear: if a necessary test is misstated in the requirements, does this represent an isolated typo or a fundamental misunderstanding about error conditions?

   For example, consider these faults, which were found in the requirements for the computer system of a video store. They all come from the part of the requirements that details how tape rentals are to be entered into the system.

   F3: The requirements say “The system keeps a rental transaction record for each customer giving out information and currently rented tapes for each customer.” However, an explanation of exactly what information is given out for each customer has been omitted.

   F9: The requirements say that when a tape is rented, the “rental transaction file is updated.” However, what it means to update the rental transaction file is not specified. The information to be stored here is not discussed.

   F10: The requirements say that when the rental is successfully completed, a form will be printed for the customer to sign. They also specify that the forms “will be kept on file in the store for one month after the tapes are returned.” This is classified as a fault because the filing and storage of the forms is outside the scope of the system, and should not be specified in the requirements since no corresponding system functionality will be implemented.

2. Identify errors by classifying the faults under progressively more abstract themes.

   Identifying fundamental errors means finding patterns in a collection of isolated faults. As such, it is a creative process for which we can give only a few guidelines.

   Take a look at the types of misunderstood information you identified in the previous step. Do particular functionalities, system users, or performance requirements appear in multiple faults? This may indicate that these are particular aspects of the system that are not well understood.

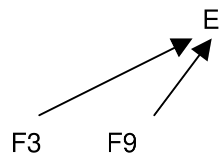
   It may also be helpful to remember that not every fault is indicative of a larger error; some will undoubtedly result just from typos or misunderstandings of very small pieces of the functionality. Not every fault has to be a part of a pattern.

   Two of the example faults discussed above can help demonstrate this. F3 and F9 both involve missing information about how the information in the database is to be updated. This may be evidence of an underlying error that goes beyond these specific faults:

E1: Misunderstanding of the process for logging rentals.

Faults F3 and F9 may be assumed to appear in the document because this underlying error has been made. F10 may simply represent nothing more important than an isolated fault, but if we could find other faults involving the rental forms, then F10 might be evidence that the role of the rental forms in the system was not completely understood. In that case, we would have to ask whether there isn't an even larger error, namely, that what historical information is to be recorded, and how it is to be kept, are not adequately understood.

We can illustrate F3 and F9 abstracting to a common error in the following way:



Naturally, this may not be the only mapping of faults to errors. Multiple ways of categorizing the faults may certainly exist.

To get the right level of abstraction requires asking yourself how useful the error classes you have identified are. If every fault represents a single associated error, then your error classes are very detailed, but probably not general enough to be helpful in finding any additional faults. On the other hand, if every fault stems from one error (i.e. "the system isn't completely specified") then the error classes are very general but not detailed enough to be of much use. It is necessary to find a middle ground between generality and detail such that a relatively small set of errors are responsible for most of the faults in the requirements. In this way the errors can optimally be used to find additional faults.

3. For each error, return to the document and make sure you have located all of the faults for which it is responsible.

   As a final step, you should return to the document to see if the errors you identified lead you to other faults that you might have missed in the first analysis. Now that you know the fundamental misunderstandings, you can see if those errors showed up in other places as well.

   In our example, once we realize that there may be a fundamental error involving the understanding of how transactions are to be logged, we can return to the document and look more closely for other faults that may have resulted. In this way, we could find:

   F12: The requirements state that when a customer pays a late fine, the rental transaction record is updated. Again, we realize that the exact fields and values that are to be updated are never specified.

   We can think of the new error as leading to other faults in the same category: