# Chapter 4

# System Dependability

**Ian Sommerville, University of St Andrews**

## Background

Complex software-intensive computer systems now run all aspects of our society and critical infrastructure from businesses to the power grid. Many of these systems have to be continuously available and must operate with few or no failures. Unavailability or system failure can mean that the organisation running the system loses business, incurs additional costs or, in the worst case, people are harmed by the system failure.

The term dependability was proposed to cover the related systems attributes of availability, reliability, safety and security.

1. **Availability**. Informally, the availability of a system is the probability that it will be up and running and able to deliver useful services to users at any given time.

2. **Reliability**. Informally, the reliability of a system is the probability, over a given period of time, that the system will correctly deliver services as expected by the user.

3. **Safety**. Informally, the safety of a system is a judgement of how likely it is that the system will cause damage to people or its environment.

4. **Security**. Informally, the security of a system is a judgment of how likely it is that the system can resist accidental or deliberate intrusions.

These are not independent system characteristics. Systems that are insecure can be attacked and their availability can be compromised; systems that are unavailable may not provide essential safety checks; systems that are unreliable may have to be taken down for repairs and so become unavailable.

Since the 1980s, there has been a large body of work looking at technical aspects of dependability. This has been based on two notions:

1. That system failures result from faults that have been introduced during the development process.

2. That the number of system failures can be reduced by avoiding the introduction of faults in the first place, by detecting faults before the system is put into use and, in some cases, by using run-time mechanisms to tolerate faults if they occur.

The technical developments since the 1980s have meant that the dependability of software has, across the board, increased very significantly. For some classes of critical system, such as control systems, the application of dependable software engineering techniques has meant that we can now build software that functions very reliability with a very high level of availability.

However, for enterprise systems, software systems that are typically used to support many different functions in an enterprise, with different classes of user, our track record on dependability improvement is much poorer. These systems still commonly fail to deliver the expected services to their users.

The fundamental reason for this is that all our technical methods of dependability achievement, based on fault avoidance, detection and tolerance, all rely on their being agreed system specification. This specification has to accurately reflect what the software is really required to do.

For control systems and for systems, such as air traffic management systems which are highly proceduralised, it is possible to write a definitive system specification. Most of the system's services are based on procedures or on reacting according to measurements of the system's environment. However, for enterprise systems, with diverse user groups, it is practically impossible to construct a definitive specification that is relatively stable and meets the needs of all users.

The reason for this is that the needs of users are not necessarily consistent and may conflict. So one group of users (group A) may require another group (group B) to enter information but group B may not wish to enter such information as they do not need to do so for their job. Whatever is specified, either Group A or Group B will be unhappy.

Enterprise software systems are part of broader socio-technical systems and the human, social and organisational issues in these socio-technical system pro-

foundly influences both the use of a software system and the views of users on that system. Many so-called 'failures' of enterprise systems are not technical failures in the sense that there are faults in the system, but are judgements made by one or more user groups about the effectiveness of the system in supporting their work.

For this reason, we are convinced that there is little point in extending technical approaches to dependability achievement for such systems. The problem does not lie in the way that the software is build but on its fit with the organisation and the work done in that organisation. To improve dependability, we need to take a socio-technical approach where we try to develop a better understanding of the settings in which a software system is used, the services needed by its users, how the system supports work and the organisational goals in introducing and operating the system.

The approaches that we can use to carry out a socio-technical analysis are covered elsewhere in this handbook and here we will simply focus on two important issues in socio-technical dependability:

1. The nature of failure, where we argue that failure is a judgement rather than an absolute.
2. The importance of designing technical systems to allow the broader socio-technical system to recover from what is judged to be a technical system failure.

## The nature of failure

The drive in technical approaches to dependability has been to avoid system failure by ensuring that faults are not introduced into a system or are tolerated during operation. There is an assumption that failures can be recognised as the system's behaviour deviates from its specification. From this perspective, an observer can examine a system's behaviour and decide whether or not a failure has occurred.

When we look at systems from a socio-technical perspective, however, failures are not so simple. Whether or not a system has failed cannot be decided objectively but depends on the judgement of the system user who has been exposed to the system's behaviour.

The reasons for this are:

1. Specifications are a gross simplification of reality for complex systems – its is practically impossible to specify everything that a complex system should and should not do.

2. Users don't read and don't care about specifications. They therefore don't know what the system is expected to do.

3. Because there are multiple stakeholders in a system, behavior that one stakeholder wants may be unacceptable to some other stakeholder. One sees desirable behaviour, the other sees a failure.

There are several factors that influence a user's judgement of whether or not a system has failed. These include the observer's expectations, the observer's knowledge and experience, the observer's role, the observer's context or situation and the observer's authority.

The socio-technical nature of failure means that it is impossible to build a system that will not fail. Changing contexts of use of a system mean that system behaviour that is acceptable at some point in time, may be deemed a failure because of changes in the way that the system is used. Furthermore, because different people have different expectations of the system, it is practically impossible to satisfy all of these – some will, inevitably, be judged unacceptable.

## Designing for recovery

It is indubitably the case that the trigger for many system 'failures' is some human action, which is, in some way, erroneous. The human action triggers a sequence of events in the system that ultimately leads to failure. For this reason, some advocates of technical approaches to dependability suggest that replacing humans with automated systems will lead to dependability improvement.

However, we should also remember that, as well as contributing to errors, system failures are often avoided through human actions or checks. People have a unique characteristic to reason about situations which they have not seen before and to take actions in response to these.

Furthermore, after some system has failed, recovery actions are necessary and these are designed and implemented by people. Hence, maintaining human operators and managers in systems is essential in our view for long-term dependability.

As we have discussed,s system failures are unavoidable so it will become increasingly important to design systems to support recovery. Recovery often involves people taking actions which are, in some way, abnormal. For example, if a particular file is corrupt and causing system failure, deleting the file may be the best thing to do. This may lead to a conflict between recovery and security. Users have to take actions to recover but the security features in a system stop

them doing so. Finding the right balance between recovery support and security is a difficult challenge.

Some general design guidelines that we have developed to support recovery include:

1. **Avoid automation hiding**. This means that information should not be hidden in a system and only revealed to users when the system believes it necessary. It also means documenting configurations and making public where information is stored in the system. Users should be able to access information about the system and its state so that they can be informed about recovery decisions.

2. **Provide an emergency mode of operation**. In an emergency mode of operation, normal checks that stop users doing things should be switched off and normally forbidden actions allowed. However, actions should all be logged and it should be made clear to users that they may have to justify steps that they have taken.

3. **Provide local knowledge**. Local knowledge is often incredibly valuable when recovering from problems so systems should include lists of responsibilities, should maintain information about who did what, the provenance of data, etc.

4. **Encourage redundancy and diversity**. For efficiency reasons, maintaining copies of information in a system is often discouraged. However, redundant information is often immensely valuable when recovering from failure. It also makes sense to maintain some of these copies in different forms – paper copies, in particular, can be useful as these are accessible even without power.