Most computing systems are extremely complex, as is evidenced by the intricacy of their hardware and software implementations. This complexity is to some extent a consequence of the many and various requirements which are routinely imposed on these systems: requirements for general purpose facilities, and at the same time for highly specialized facilities; requirements to provide service to numerous users with diverse demands, and to provide this service simultaneously (or seemingly so); requirements for extremely sophisticated facilities, and for simple and convenient access to those facilities; requirements for prompt and timely service, and for efficient, economical and reliable operation. It can be persuasively argued that modern computing systems constitute the most complex artifacts ever constructed. Given the complexity of these systems, and accepting that their design and construction are susceptible to the inherent fallibility of those who design and construct, it would be surprising indeed if a modern computing system provided its intended service with perfect dependability.

To obtain substantial gains in reliability it is essential that the complexity in systems is brought under control. The designers of hardware for computing systems have made a virtue out of necessity by accepting the constraints imposed by physical factors (e.g. limits on the number of components which can be interconnected). By adhering to strictly enforced design disciplines applied to components having relatively simple interfaces an enviable degree of control has been maintained over the complexity
of hardware systems. However, these disciplines may well prove inadequate as advances in VLSI expose hardware designers to the problems of complexity which have always confronted software designers.

Software has one great advantage over hardware, and that is the apparent ease with which it can be constructed and subsequently modified. Software systems are built, extended and adjusted to meet changing requirements simply by typing appropriate characters (the difficulties all lie in selecting those characters). The relative difficulty and cost of making belated design changes in hardware systems has meant that the complexity of most computing systems is largely to be found in their software. For this reason much ongoing research is concerned with mastering software complexity. Techniques such as structured programming, top-down development, information hiding and separation of concerns have a common thread in that they all embrace the principle of divide and (hope to) conquer. The notion of abstract data types and object-oriented systems and the programming languages which provide encapsulation mechanisms by which a programmer can define his own data types, are founded on the same principle. All of these techniques try to ensure that a system is not constructed as a monolithic entity.

It is now widely accepted that systems should be designed and implemented so that they are well-structured, being built up as a coherent assembly of component sub-systems, which are themselves built up from smaller components, and so on. Ideally, each stage of combining a set of component systems to form some larger system should be kept sufficiently simple for it to be easily comprehended, in order to minimize the risk of mistakes in design. Although such an approach cannot be expected to eliminate all faults from an operational system, the identification and imposition of structure when the system is being designed and constructed should greatly reduce the number of residual faults.

It is particularly necessary to maintain system structure when fault tolerance techniques are considered. As will be seen, if the structure which the system designer envisaged is indeed present in the actual implementation of a system, and if the constraints implied by that structure are enforced during the operation of the system, then assumptions may be made which greatly simplify the provision of fault tolerance. Furthermore, a well-structured approach to the design and inclusion of fault tolerance techniques is a prerequisite for their success; an unstructured approach could easily reduce system reliability by introducing more faults than those to which tolerance was provided.