INTEGRATED PROJECT SUPPORT ENVIRONMENTS:
GENERAL PRINCIPLES
and
ISSUES IN THE DEVELOPMENT OF HIGH INTEGRITY SYSTEMS

John A McDermid

Professor of Software Engineering
Department of Computer Science, University of York
and
Director, York Software Engineering Limited

Part I: Introduction

There have been many advances in software development technology and in software engineering methods and tools since the introduction of computers in the late 1940's and early 1950's. Perhaps the most significant advance in software quality and individual programmer productivity has arisen from the development, and evolution, of the high level programming language. A significant effect on software development productivity, if not always quality, has also arisen from the dramatic increase in the performance/price ratio of computer hardware, particularly from the advent of the workstation.

These developments, particularly in hardware, have also (at least partially) contributed to increased expectations about what can be achieved with computers. These expectations have led to the undertaking of many large, and all too often unsuccessful, software development projects. Brooks in his article "No Silver Bullet" [Brooks1987] articulates some of the problems underlying large software developments and casts doubts on the possibility of general solutions to these problems. Brooks, and many other authors, make it clear that the problem of managing the interactions and communication within large development teams is one of the key difficulties facing the software industry.

Intellectual solutions are required to the problems of organising and managing large team projects. However the issues of scale make it clear that such solutions will only be practical if they are given adequate machine support. Unfortunately, until recently, there was little in the way of support for team working. Integrated Project Support Environments (IPSEs) are intended to address these problems - the objective being to provide a "complete" development facility for a project team. The primary means of doing this is by providing an infrastructure and a set of tools to:

• facilitate communication within the team;
• support all (or most) of the technical and managerial activities in the software development and maintenance processes; and
• control access to data shared by members of the team in order to prevent inconsistent modification to the software under development.

In practice IPSEs often fall far short of these objectives, e.g. by supporting only a limited
number of activities in the development process. We will expand on these basic require­ments for IPSEs, and illustrate the capability of current environments, later in this chapter.

A further trend in the defence sector, industry and commerce is to use computers and software in increasingly large numbers of increasingly critical applications. Example classes of critical applications include:

- safety critical - where deaths or injury may be caused by computer or software malfunction, eg fly by wire aircraft, and active suspension for cars;
- military security critical - where loss or disclosure of sensitive information may be highly damaging to the nation, eg a command and control system providing information on troop disposition to military commanders;
- enterprise critical - where malfunction of a computer leading to incorrect operation or loss of information could bankrupt of otherwise critically damage some business or enterprise, eg programmed trading systems on the stock market.

We will use the term high integrity as a generic name for the classes of applications illustrated above to indicate that integrity (trustworthiness; freedom from impairment or corruption) is required for the software in those applications.

As the above classes of application require high integrity software there are demands on the integrity of the development processes, tools and the IPSE (if any) used to produce the application programs. Consequently there are additional integrity require­ments on IPSEs intended for use in support of the development and maintenance of high integrity applications, by comparison with IPSEs intended for less stringent applications. We will address the basic requirements of IPSEs for developing high integrity systems later in this chapter, and we will also present a fuller treatment of the concept of integrity.

We are now in a position to outline the aim and contents of the chapter. The aim is to explicate the principles and concepts of IPSEs; to indicate how effective current IPSEs are when judged against these principles; and to consider what additional requirements are placed on IPSEs by the need to support the development of high integrity systems. Due to limitations of space we can only briefly address these issues and we have to gloss over a number of issues altogether, eg implementation strategies for IPSEs.

Part II thus sets out the basic concepts and principles of IPSEs, focusing on the term integration. It analyses five different forms, or types, of integration that (we believe) are required in order that IPSEs can satisfy their basic objectives.

Part III considers the evolution of computer based support tools from simple, single user, tools to the current concepts of IPSE architecture. It also reviews the capabili­ties of three, very different, current examples of IPSEs.

Part IV discusses the high integrity issues. The treatment is in terms of likely requirements for high integrity development - we must stress the term "likely" as this is still an active area of research. The focus is on integrity in the IPSE itself.

Part V considers trends and presents some general conclusions.