SHEIL83
Sheil, B.;
Power Tools for Programmers;
Datamantion (February 1983).

This paper presents motivations for exploratory programming environments and characterizes the tools and techniques needed in such systems.

Conventional programming tools and methods, which protects against unintentional changes to design, restrains programmers by imposing structure and control over the implementation process. However, for applications which are prone to specification instability, intentional changes are difficult to bring about and generaly end in implementation disasters. Such applications clearly support the notion that system design is intertwined with system implementation. As such, they require exploratory programming systems that allow the exploration of design problems by experimenting with designs.

To do this, these systems require power tools with considerable capacity to maximize the programmer's effectiveness. Sheil identifies several sources of such power. Namely, interactive graphics, integrated facilities, and application abstractions (operations and objects) embedded directly into the environment.

To minimize constraints on programmers, the programming languages exclude redundancy structuring mechanisms from the language. Also, they should allow the programmer to defer commitments for as long as possible by making extensive use of late binding. Examples of this include providing storage management facilities in the environment itself and making dynamic typing of variables and dynamic binding of procedures available.

If the dynamic binding of procedures is carried one step further, programs may be represented as data structures (e.g. syntax trees) which in turn can be manipulated by other programs. This technique leads to two significant developments. First, one can develop an application by designing a special language (and thus provide an interpreter) which provides a concise representation in which the application is relatively easy to state. Second, it becomes easy to write program manipulation subsystems. Consequently, exploratory programming languages themselves can grow and programming support tools can be developed.

Sheil lists a variety of other programming tools. One such type of programming tool is knowledge based, (i.e. a tool with knowledge about the user's program and the context in which it operates). Another type include tools for program contraction (i.e. take the functionally adequate program and make it more useable). This places two burdens on the exploratory programming system which must be addressed. First, the architecture must allow for an efficient implementation. Second, the environment must provide performance engineering tools just as it provides design tools.

In conclusion, Sheil states that the increasing importance of poorly understood applications coupled with the increasing cost effectiveness of such systems will make exploratory development a key technique for the 1980's. Also, the author states that the tools of such systems could be adapted to support programming in conventional languages since they do not depend on the more exploratory attributes of either the language or environment.

This paper is an adequate introduction to exploratory programming environments. It provides insights to the nature of such systems and the basic tools found in such systems. One can see the strong influence of these systems to many areas such as OIS, CAD/CAM, data base in which integrated application environments are now more the norm. Most notably, it has encouraged the work on user interfaces. User interface development systems (i.e. specialized environments for prototyping user interfaces) are clearly possible and not just fiction.