A formal development framework and its use to manage software production
Cooke, J.; Stone, R.
Tools and Techniques for Maintaining Traceability During Design, IEE Colloquium on
Volume , Issue , 2 Dec 1991 Page(s):10/1 -

A Formal Development Framework and its use to Manage Software Production

John Cooke and Roger Stone

Summary

Within an ESPRIT project called FORMAST the authors devised the concept of a ƒorm to bring together all aspects of the development of a 'module' (i.e. specification, design and verification) within a distributed asynchronous system. The use of ƒorms can be extended to other compositional system development scenarios. Using ƒorms in a top-down fashion means that the logical interaction between the modules already designed and the formal specifications of modules required to complete the implementation can readily be ascertained. Moreover, this can be done at any stage of development. Thus one can maintain an overall view of the entire system design and use this to monitor the progress of program construction. This then provides a notion of traceability from a management standpoint.

The development of software by the use of formal methods guarantees traceability of (functional) requirements from the original specification through refinement to the final acceptable executable code. This is achieved either by the use of transformations, or by 'invention' and the discharge of appropriate proof obligations. Although formal methods do not dictate a particular strategy, it is natural to proceed in a hierarchical fashion and typically there is no executable code produced until a significant fraction of the project period has elapsed. This is of concern to management because traditional programming productivity has often been measured by the number of lines of code produced each day and, once under way, the rate of production is expected to be more or less linear.

Within an ESPRIT project called FORMAST we devised the concept of a ƒorm to bring together all aspects of the development of a module (i.e. specification, design and verification) within a distributed asynchronous system. The use of ƒorms can be extended to other compositional system development scenarios.

Using ƒorms in a top-down fashion means that the logical interaction between the modules already designed and the formal specifications of modules required to complete the implementation can readily be ascertained. Moreover, this can be done at any stage of development. Thus we can maintain an overall view of the entire system design and use this to monitor the progress of program construction. This then provides a notion of traceability from a management standpoint.

The hierarchical development of software can be viewed as a tree, growing from the root, the original - high-level and non-algorithmic - specification by means of refinement steps to final code at the leaves of the tree. Essentially a ƒorm documents the development that takes place at the internal nodes of the tree. A ƒorm begins with a statement of the requirements of the associated sub system/program. Until a suitable design idea has been supplied, all we can do is use this specification as a logical 'stub' which can be used in prototyping and as a measure of 'work still to be done'. A completed ƒorm includes a design for the sub-system, its formal justification and, where appropriate, the specifications of any smaller components needed for its completion.

To support such development there is a suite of CASE tools called ƒorm-tool. While these have been designed specifically for use by software engineering students, all the necessary management information is inherent within the system and the construction of a suitable interface would provide a simple and accurate monitoring tool for tracing (tracking) progress of software/system development.

Department of Computer Studies, Loughborough University of Technology, Loughborough, Leics LE11 3TU