Transactions on Information and Communications Technologies vol 4, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

The software life-cycle model: An alternative perspective

V.J.D. Sivess[a], M.C. Curtis[b]
[a]*Department of Electronic & Computer Science, University of Southampton, Hampshire, UK*
[b]*ICL Associated Services Division, Eskdale Road, Wokingham, Berkshire, UK*

ABSTRACT

The quality of a software product depends on the proper management of the technical process of software development. Often conflicts arise because the technical structure of a project does not fit well with the temporal structure that management desires. The Life-Cycle Model is fundamental to both aspects of a project. This paper presents the outline of a new Life-Cycle Model which emphasises the static nature of a project, where all phases can coexist, while allowing for temporal progression. Central to this philosophy is the concept of a project database with a highly integrated toolset that can check for consistency and completeness over the whole project and generate metrics that can measure progress.

1 INTRODUCTION

The successful completion of a software development project involves the timely delivery of a high-quality product, that meets the client's needs, in terms of correctness, availability and pleasantness to use, within the specified budget. These attributes form part of the quality of the product, in terms of the ISO 8402 definition of quality as

> the totality of features and characteristics of a product, process or service that bear on its ability to satisfy stated or implied needs [8].

To achieve this, Ould [9] shows the need for *technical planning*. This involves assessment of risk, leading to a choice of process model, and characterisation of the problem, leading to the choice of appropriate methods and tools.

In contrast to this desireable outcome, the shortcomings of much software are well documented. Costs and timescales are not kept to, and quality is below standard. Cave and Maymon [3] say that the difficulties in assessing the risks involved in developing a piece of software are a result of the complexity and intangible nature of the final product. It is not always easy to distingish between requirements that will take a day to implement and those that will take a week, with the result that many projects are skimped at crucial stages and thus run late or are brittle and hard to maintain. It is clear that many software developers are vague about which methods and tools are appropriate for any one stage of a project.

The life-cycle model (LCM), or process model, plays a central role in the management of medium and large-scale projects. Ould says that it "determines the overall shape of the project" [9]. It offers a framework for organising and assessing the progress of the work by providing checkpoints in terms of deliverables, and thus allows the project to be controlled and timetabled. This paper aims to explore ways in which such a model may be made more effective. It shows that the overlaying of multiple semantics is a possible source of confusion and suggests an alternative model based on constraints.

A major problem is the complexity of a large project. The sheer number of interrelationships between objects can make it almost impossible to check for consistency and monitor progress. This paper shows that with all information concerning a project being held in a fine-grained database, managed by an integrated set of tools it is possible to maintain consistency and determine progress.

This work is at an early stage and this paper presents a broad set of ideas pertaining to the software development process. Early results are encouraging and it is hoped that a full development environment based on these principles will become available within the next five years.

2 LIFE-CYCLE MODELS

Figure 1 shows an early process model of software development. This is known as the *stagewise model* of software development [1]. It emphasises the chronological unfolding of events in sequence as development proceeds. Coding is the last event in creating a deliverable piece of software which then needs to be tested and thereafter maintained. Although an advance at the time, using this model is misleading. In particular, verification needs to be carried out at each stage of the process, and not

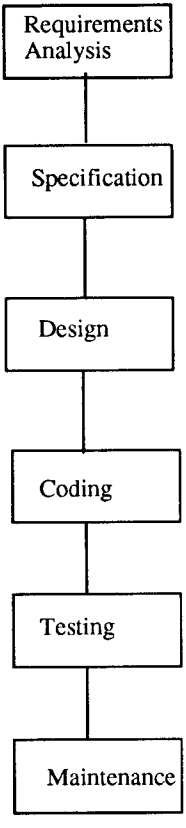

Figure 1: The Stagewise Process Model

just at one separate stage called "testing", and this provides feedback between the stages with a consequent reiteration of some of the activities. In reality, the last two stages were often unaccounted for in the planning of a project which would then run over schedule. The model fails to capture something that is well known by those involved in designing and writing software: the sooner an error is discovered, the easier and cheaper it is to put right. A design error that is not discovered until the coding is tested may have devastating effects on the resulting quality of the finished product. The stagewise model completely hides this.

2.1 The Waterfall Model

The Waterfall model, introduced by Royce in 1970 [11] improved on these shortcomings by adding backward arrows between nodes to represent a verification and acceptance process between one phase and the next. The nodes represent *documents* or *deliverables* and the arcs are *phases* in which

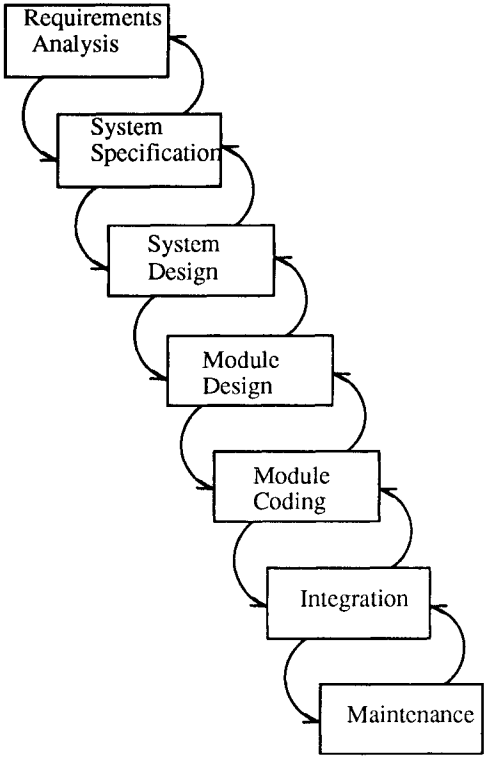

Figure 2: The Waterfall Model

the deliverables are created. A change at one level affects the level above and this may have a ripple effect. The model makes explicit the need to allow time for validation and indicates that there may be a need to redo some of the activities of a previous stage.

2.2 The EWICS Model

The EWICS model, presented by Mitchell [7], further explicates the verification procedures, showing, for example, an *interpretation and agreement* relationship between the requirements specification and the description of the problem domain in the real world, and arcs representing consistency checking. Maintenance is not a separate phase, but another dimension to the model into which modifications feed via a *modification specification.*

The requirements specification covers not only the functional requirements, but also requirements related to performance and resources, and so on. In a refinement to the model, instead of carrying the non-functional details of the requirements specification down to the implementation

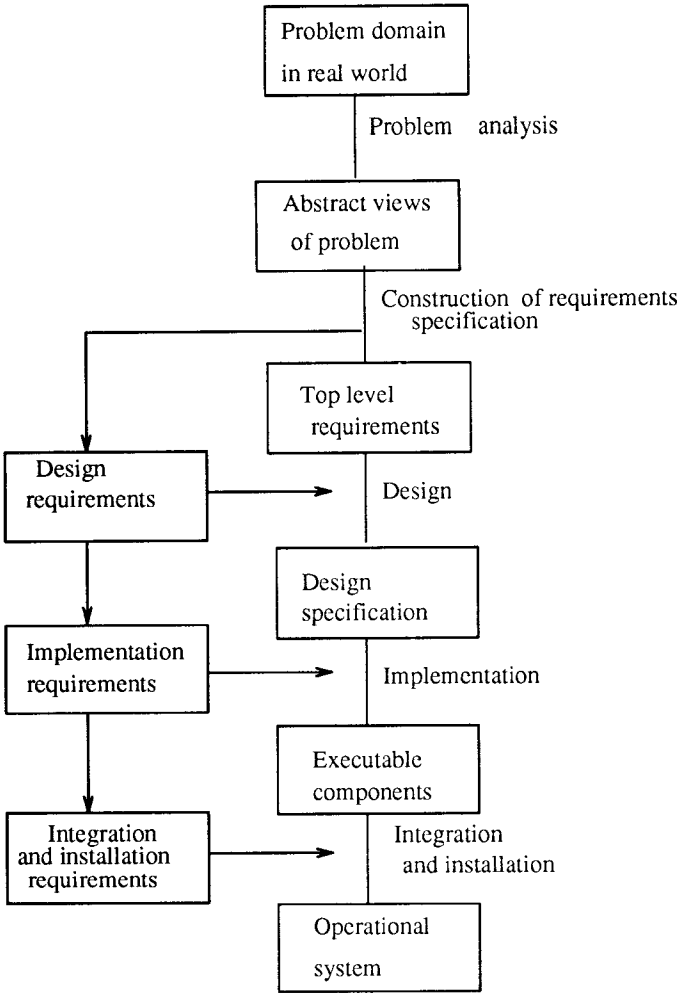

Figure 3: The Refined EWICS Model

phases, the requirements specification is itself divided into a number of levels as shown in figure 3, which, for simplicity, excludes the arcs representing verification.

This version gives more information to managers and developers by making clear the fact that users often have very specific low-level requirements, such as the type of screen required. These requirements must be passed to the appropriate phase of development. If these requirements are not met, this again affects the quality of the product in that the user will be less satisfied.

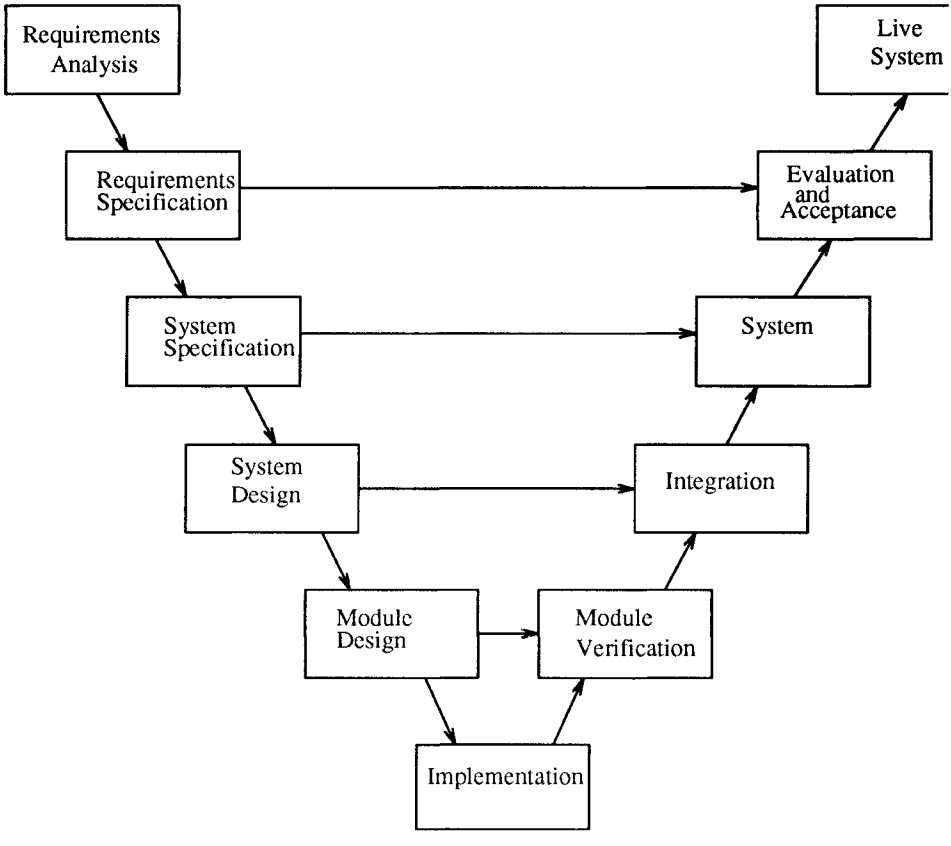

Figure 4: The V-Diagram

2.3 The V-Diagram

The V-diagram emphasises the build and test activities of software development by showing them explicitly along the right hand side of a V, as shown in figure 4.

Again, each document is verified against the level above and checked for internal consistency. Development proceeds down the left-hand arm of the V and up the right-hand one. Associated with each document on the left-hand side is a verified document on the right-hand side. The information for test cases and test data is derived from the document at the same level and the one below.

2.4 Boehm's Spiral Model

Boehm directly tackles the control of risk in his model [2]. As the arc of the spiral travels in a clockwise direction it passes through the four quadrants

representing statement of requirements, assessment and resolution of risk, development and planning. As the model spirals outward, the cumulative cost is represented by distance along the radial dimension.

3 SOME POSSIBLE SEMANTICS

3.1 The Transformation Model

Early software process models were concerned with chronological stages of development. Another way of viewing the LCM is to see the arcs as representing *transformations* between the documents or descriptions at the different levels. Where formal methods are being used, each document may be seen as a syntactic object, with an interpretation in the real world, which by successive transformations becomes the required executable program. These models overlay the original time based semantics with a transformation semantics which views the model as a static description of a system in terms of *levels*. By definition, the boundaries correspond to the documents which are being transformed.

3.2 Phases, Stages and Activities

The EWICS model is defined in terms of *documents* and *activities*, and Mitchell says

> ... words such as "design" and "verification" used to label arcs on the life-cycle diagram are intended to denote activities or processes which can be performed a number of times during the life of a system rather than phases in the life of a system or steps in the development of a system. [7]

McDermid [6] distinguishes between *stages*, found in what he terms *technically* oriented models and *phases*, found in *management* oriented ones. Stages are "technically related collections of activities", each at a different level of abstraction, and are thus clearly compatible with a transformation semantics. Phases are "temporally related groups of activities", which "terminate with a review of some products". There is not necessarily a one-to-one mapping between the two concepts. The EWICS model is a technical one, according to this definition, and it is possible to view it as a transformation model. The Waterfall model and V-Diagram are classed by McDermid as management models.

4 LEVELS AND ABSTRACTION

In hierarchy theory and systems theory, simple systems form simple hierarchies where levels are based on grain size and elements of one

level compose simply to form elements in the next level up [5] [10]. More complex systems form only partially decomposable hierarchies where the relationship between two adjacent levels is stronger and one level can provide a context for emergent properties to appear from the level below it. Such systems include biochemical systems. For instance, Grobstein [10] cites the emergence of enzymes from amino acid chains. Such systems are often associated with different levels of description.

Life-cycle models exhibit both kinds of level: levels based on granularity and context-shifting levels. For instance, Ould says that in the V-diagram "there is an implied decomposition from system, through subsystems, to modules" [9] and indeed it is sometimes drawn with the system design node fanning out to several nodes representing the design of the modules at the level below. On the other hand, quality factors such as maintainability and reliability emerge from the way in which the code is structured and written, and the higher level descriptions of functionality exhibit a selective loss of detail that is characteristic of more complex hierarchies. McClamrock [5] claims that there is a confusion between the use of both types of level in Marr's three proposed levels of cognition and that it is helpful to be aware that both types are present. We make the same claim for their use in the software process model.

To confuse matters further, documents at a more abstract level in a transformation model contain information that is only relevant to a much later phase of the time-based models. Mitchell speaks of low-level requirements being passed down through the different levels and McDermid speaks of quality requirements "migrating" down the various levels, a process referred to by Ould as "quality factoring".

The requirements specification can be seen as giving a *context* to the functional parts of the system. McClamrock says "The functional properties of parts of complex systems are *context-dependent* properties - ones which depend on occurring in the right context, and not just on the local and intrinsic properties of the particular event or object itself." [5] We will interpret the information in the requirements specification in this light and thus propose a new process model which we have termed the *Ferris-Wheel model.*

5 THE FERRIS-WHEEL MODEL

The functional information in the requirements specification is contextual information for the system specification downwards. The non-functional information is contextual information for the final product and is passed up through the levels in the form of constraints. This gives the circular structure shown in figure 5. The model as it stands is a simple transformation model. The system specification is transformed into a design specification. The transformation is constrained by the information com-

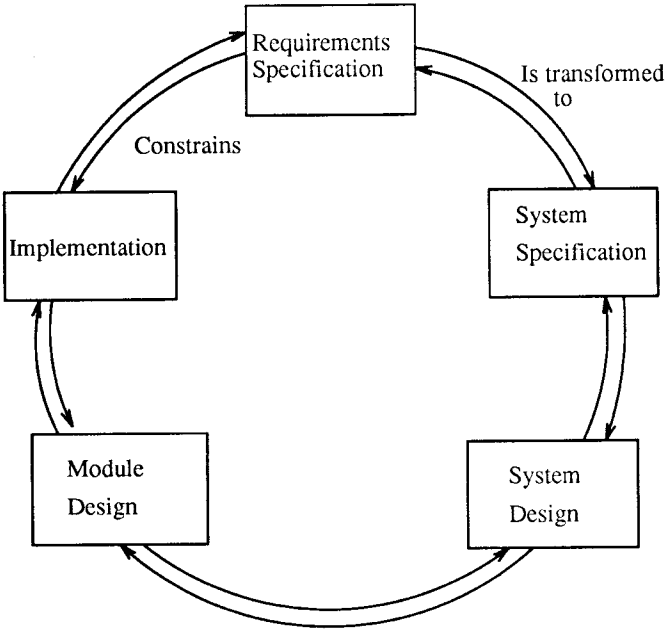

Figure 5: Circular Structure of Constraint Based Model

ing from the other direction. For simplicity, arcs showing a **verification** relation are not shown.

5.1 Propagating Constraints

All non-functional constraints are passed straight to the implementation level. Any constraints that are not applicable at this level are passed to the design level. Any that are applicable are dealt with and any consequent constraints are passed on along with appropriate unchanged constraints. In this way, the constraints are passed around the model in an anti-clockwise direction. Thus, activities are being carried out by members of the implementation team while the top-level specifications are being written. In this way, most of the risks associated with later parts of the project can be assessed and taken account of at an early stage.

5.2 The Hub of the Wheel

We then add to the basic circular structure a hub, comprising a project database, with arcs to each of the nodes. This will be explained below.

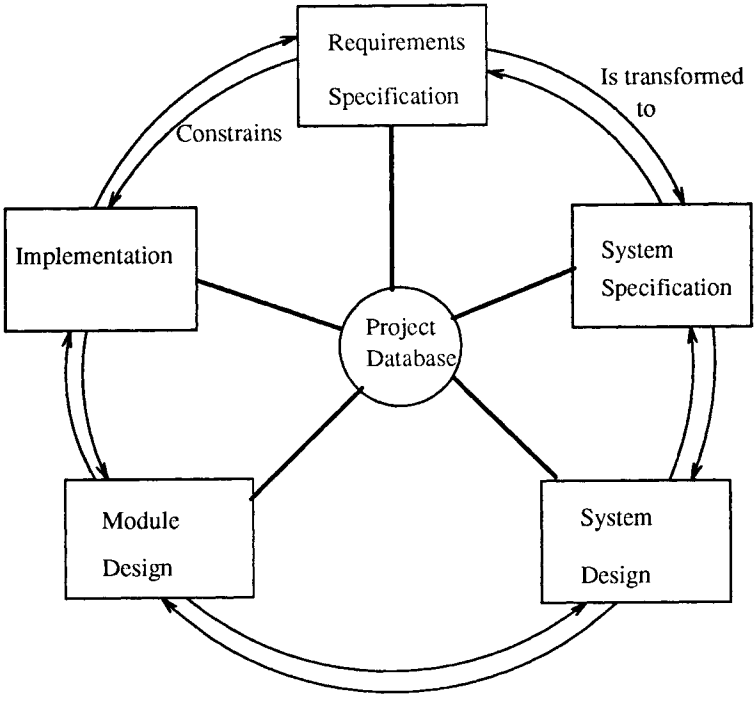

Figure 6: The Ferris-Wheel Model

6 A Brief Example

An electronic diary system is to be produced for an organisation of five thousand people. Each member must be able to **add**, **delete**, and **view** entries. People are organised into non-disjoint groups and certain people can book appointments for other individuals and groups. The final system is to be implemented using the Ada language on a network of PCs.

The system specification defines the system boundaries. We may choose to model an individual's diary as a set of mappings from date to list of appointments, while the organisational diary may be a set of non-disjoint sets of individual diaries. In order to define the operations on an entry, a careful modelling of time will be needed. We may also decide that we need to model concurrency explicitly.

Meanwhile, the non-functional constraints are passed in the other direction; the system must be programmed in Ada, it must run on a network of PCs and must cater for at least five thousand entries. Management must make sure that enough Ada expertise is available in the programming team. If not, training courses, or the cost of buying in expertise, needs to be costed. They also need to check that suitable development hardware and software is available and what version of the compiler is to be used for the live system. When these constraints have been fulfilled, then we can see if there are any consequent constraints that need to be passed on.

We will propagate the requirement for Ada expertise to the level of module design. The Ada compiler imposes further constraints: data structures must not exceed 64K and there is a limit of three thousand records for indexed-sequential files. These, along with the volume requirement of at least five thousand entries, are passed as far as system design where we decide on the main components of the system. An indexed-sequential file organisation is a fairly natural one for this application and since there is a limit of three thousand records, a file handling package based on B-trees must be written or purchased. By the time we actually code the system, we should not run into problems with the compiler, because any limitations have been taken care of as far back as the system design stage.

7 TOOLS

A life cycle model should not be prescriptive about methods or methodologies that may be used to accomplish any of the phases of the development process. We assume that each phase will be accomplished by the use of one or more tools. Normally each tool will have its own internal descriptions of the objects that it processes and any relationships that exist between objects residing in different tools can be difficult to indicate

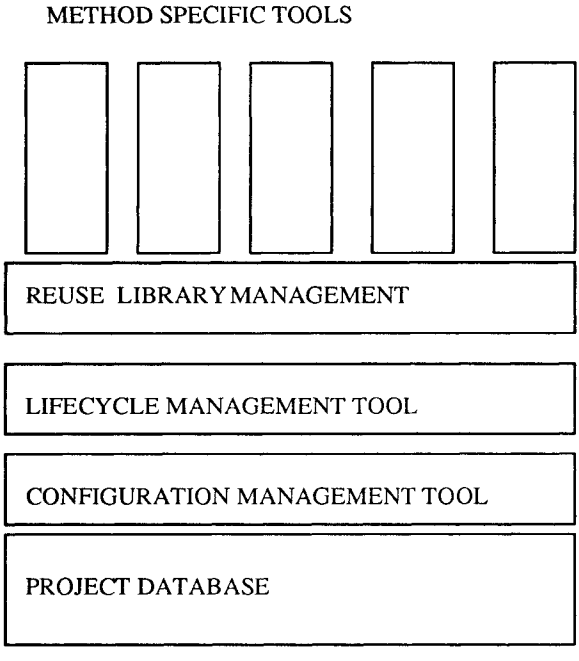

Figure 7: Tool Architecture

and maintain. It is however essential that relationships indicating the constraints that exist between such objects be maintained so that it is possible to determine the impact of changes and so that progress can be measured. This assumes a degree of integration between possibly heterogeneous tools that is difficult to achieve except by the use of PTI (Public Tool Interface) such as that offered by PCTE (Portable Common Tool Environment). A PCTE tool manages all its objects and the relationships between them as items within a common database, accessing them by means of the OMS (Object Management System). The tool supplies a PII (Public Integration Interface) which allows other tools access to its objects in a managed and secure way. In particular it may allow additional attributes to be applied to its objects and relationships to be defined with external objects.

For example a Requirements Analysis Tool may produce a set of requirements, each of which is an object. A Design Tool may produce a set of modules. A requirement may constrain one or more modules, or a module may represent a transformation of a requirement into a form that imposes further constraints on other elements of the system. Each of these relationships can be represented by links within the database.

The method specific tools access the database through other tools which have a universal role. Three specific examples of these are:

- A Reuse Library Manager, which classifies each object, adding classification attributes and links into a reuse library.
- A Lifecycle Management Tool which attaches project description attributes and relationships. This would incorporate a project planning tool which can navigate the database and produce metrics of progress, estimates of the impact of changes and generate reports from the information contained in the attributes.
- A Configuration Management Tool which manages the versions of individual objects and informs each tool when one of its objects has been made obsolete.

8 THE PROJECT DATABASE

It is fundamental to the nature of software projects that it must be possible to describe the project both as a whole and various aspects of it. Every document within the project, including the source code, can be thought of as a part of the overall description of a project. It is important that this description be thought of as a single entity. If it is not then the all too common results are:

- Some parts of the project, or some views of parts of the project, are not described at all.
- Some parts are described in the same way in a number of different places.
- Contradictory descriptions may occur.
- A wide variety of different description languages are used.
- It is not always easy to see relationships between different descriptions.

The totality of information generated and maintained by the various tools forms a project database. The existence of relationships between the items produced by the tools enables this database to be regarded as a complete and consistent description of the project.

This description, or parts of it, must be accessible in a textual form. Each individual item within the database should be accompanied by a textual description, preferably in a standard format. Report generators can directly access these descriptions to produce documents in a standard format. The converse may also be true in that documents that describe parts of the system in a form of structured English can be parsed and used to construct parts of the database. Details of the format need not be fixed but may be held in some appropriate form as an item within

the database. Many tools incorporate their own methods of recording textual descriptions of their elements. The Lifecycle Management Tool can maintain links both to these text items and to any other tools necessary to translate internal formats into an acceptable form.

For example one tool may generate TEX[1] format while another may generate troff format. It is relatively straightforward to manipulate both formats so that the appearance of a printed document is consistent. It is therefore possible to maintain a single document that actually consists of a number of elements in different formats yet which appears in print as a single document in a consistent style. Of course the use of standard formats such as SGML (Standard Generalised Markup Language) is to be encouraged.

One important aspect of software project management is that of encouraging and enabling reuse, both internally within a project and externally between projects. The existence of a complete project database allows individual items and groups of items at any level to be reused within a project, simply by the construction of a new set of relationships, and externally by allowing the extraction of composite objects complete with the constraints that they apply to the rest of the system.

The project database is represented by the hub of the ferris wheel, the spokes of the wheel show the connections between the hub and the tools that implement the various phases. They also show that the strength of the entire structure is determined by the strength of these connections.

9 Incremental Development

The turning of the wheel indicates the temporal progress of the project. The model shows that all phases can be active at all times and that intercommunication between phases can occur to some extent independently of the temporal stage. The wheel does not have to turn just once, indeed were it to do so it would simply represent a slightly different view of a stagewise model. The wheel may turn a number of times. The first turn of the wheel is used to build the basic structure of the database and it is to be hoped that this structure would remain essentially fixed thereafter. Subsequent turns build on to this structure producing prototypes that allow parts of the database to be made complete and their consistency checked. Once a product has been issued maintenance and development are represented by further turns of the wheel, which emphasises the point that these are not separate activities but all part of one lifecycle. Gilb [4] shows that an evolutionary approach to system development gives greater control over risk elements. This model encourages the evolutionary approach and provides a mechanism by which its advantages can be

[1] TEXis a trademark of the American Mathematical Society

realised.

10 CONCLUSIONS

The life-cycle model proposed here presents an alternative view of the system development process that treats all aspects of a project as part of a single entity. Functional and non-functional requirements are treated as contextual constraints which are passed through the various phases of a project in both directions to reach the point at which they may be applied. This gives the model a circular structure. Temporal progression is modelled by the turning of the wheel. The constraints are treated as relationships within a central project database.

Managers would like to partition a project into *risk increments* and commit resources on an incremental basis [3]. Incremental and evolutionary approaches to project development are becoming more popular. Our model is compatible with and indeed encourages these development methods.

The motivation for this work has been the desire to increase the quality and reduce the risk of software projects. It is our contention that the best way to do this is to achieve a high degree of integration between all the elements that make up the project and to be able to view and manage the project as a single entity. The lifecycle model described here allows this to be done and provides a suitable framework for reasoning about the nature and structure of a project.

There is much work still to be done. A language for expressing constraints is under development. It takes the form of a structured English framework in which textual or mathematical descriptions of objects can be embedded. This language will be supported by tools which treat the constraints as relationships between objects in a PCTE database and can use the descriptions to check for consistency and completeness of the structure and provide metrics.

A generic project database has been specified and will soon be used to support small scale projects. This will provide the opportunity to determine the most appropriate metrics that provide sufficient information for management to construct a temporal plan and analyse risk.

Work is progressing on the integration of tools and the definition of standards for Public Integration Interfaces on PCTE as part of a EUREKA project.

References

[1] H. D. Benington. Production of large computer programs. In *Proceedings of Symposium on Advanced Programming Methods for Digital Computers*, 1956.

[2] B. W. Boehm. A spiral model of software development and enhancement. *IEEE Computer*, pages 61–72, 1988.

[3] William C. Cave and Gilbert W. Maymon. *Software Lifecycle Management: The Incremental Method*. Macmillan, 1984.

[4] Tom Gilb. *Principles of Software Engineering Management*. Addison-Wesley, 1988.

[5] Ron McClamrock. Marr's three levels: A re-evaluation. *Minds and Machines*, 1, 1991.

[6] John McDermid, editor. *Software Engineer's Reference Book*. Butterworth-Heinemann, 1991.

[7] Richard Mitchell. European workshop on industrial computer systems - life-cycle model. Technical report, The Hatfield Polytechnic, 1983.

[8] International Standards Organisation. *ISO 8402: Quality Assurance - Vocabulary*.

[9] Martyn A. Ould. *Strategies for Software Engineering*. Wiley, 1990.

[10] Howard Pattee, editor. *Hierarchy Theory - The Challenge of Complex Systems*. George Braziller, New York, 1974.

[11] W. W. Royce. Managing the development of large software systems: Concepts and techniques. In *Proceedings of WESCON*, 1970.