Parallel System Development:  
A Reference Model for CASE Tools

Derek J. Hatley
Grand Rapids, Michigan 49518-8727

ABSTRACT

The study results summarized here conclude that no simple sequential model can describe the development process; that a parallel model, adaptable to the needs of the particular project, is more realistic. This parallel model encompasses all of the disciplines involved in development: systems, hardware, software, reliability, testing, manufacturing, and so on. A project database is needed to accommodate all the many products of development and their complex relationships across all these disciplines. Such a model and database provide a reference for evaluating existing CASE tools and developing new ones. Existing CASE tools fall far short of meeting the needs of this model.

INTRODUCTION

Spurred by the need to improve our development efficiency, to meet the needs of the Ada language and DOD-STD-2167A[8], and to address the challenges of the Software Engineering Institute's Capability Assessment Program[7], we at Smiths Industries (SI) are conducting an ongoing study and enhancement program on our development process. The study is being conducted by a team of people with extensive practical experience in the development of complex avionics and other systems, and has consisted of examining each of the traditional phases of the process as they have been conducted at SI, attempting to tie these phases together into a unified model, and conducting a simulated project, in which we not only worked the project, but studied in detail the processes we went through in working it. We are now starting to apply our findings to real projects, and will be tracking the results closely. This paper summarizes some of our results so far, and shows how they provide a reference model for CASE tools.

BEYOND SOFTWARE

Most of the thrust in improving the development of today's systems has been directed at their software. Certainly, the challenges of software development are daunting, but software is nevertheless just one component of the total system. Required system capabilities are fulfilled by hardware, software, and manual procedures, all of which must work together harmoniously. In addition, there are other types of requirements, such as customer imposed design constraints, hardware and software reuse, testability, reliability, safety, maintainability, manufacturability, and many others. All of these play a role in the total system design. Meeting them all requires multiple tradeoffs between different design choices, and frequent reviews of earlier tradeoff decisions as the development proceeds.

Many of these tradeoffs must be made before the software development can be tackled, and they may have profound influences on the difficulty of the software task, in extreme cases even making the task impossible. Examples of such tradeoffs are: number of physical boxes; number of processors in a box; types of processors; and data channels between boxes and between processors. With the increasing availability and power of application-specific integrated circuits (ASICs), and special-purpose processors, such as Fast Fourier Transform (FFT) processors, the allocation of capabilities between hardware, software, and manual processes becomes one of the major tradeoffs. Which tradeoffs are most important is very much a function of the particular system: for example, the priorities for a new type of system will be very different from those for an enhancement to an existing system. Clearly, such tradeoffs must be addressed very early in the development; they require the involvement of all the major disciplines, and may require extensive design analysis of all the principal candidate system configurations.

Thus we see that system development is multi-disciplinary, and that close interactions between disciplines are required throughout. This suggests that, as has been pointed out in recent literature[2,3], the difficulties of system development are much more psychological than technical. It is evident that no simple sequential model of the process will suffice. At Smiths Industries we have developed a multi-layered, parallel model, in which all disciplines are involved to some degree from day one, and continue their involvement throughout the system lifecycle. Our contention is that this is what really happens
(or should happen) anyway, and we need a model that acknowledges that fact.

It is also evident that CASE tools must support such a model if they are to serve a useful purpose in the development process. A major problem with CASE tools to date has been the lack of understanding of their role in the development process. Typically, much more has been expected (and claimed) of them than present tools can possibly provide, resulting in inappropriate, ineffective, and costly use. The establishment of a comprehensive model of the total development process allows us to (a) identify the proper scope of existing CASE tools within that process, and use them appropriately, and (b) provide a framework for the development of more comprehensive CASE tools. We believe that ultimately, to achieve their promise, CASE tools must, like the development process, be multi-disciplinary. That is, they must be useful to all the disciplines involved in development, not just the software disciplines. To achieve this, they will have to interface directly with a comprehensive project database, and with existing CAE tools used in hardware development and manufacturing.

THE DEVELOPMENT PROCESS

The traditional representation of the development process has been the waterfall model, which shows sequential progress through the development activities. Even though the model shows feedback between adjacent phases, the concept is essentially sequential, and project plans have tended to follow this principle, with one activity expected to be fully completed and closed at the time, or soon after, the next activity is started.

Other models have explicitly shown the feedback between phases. One such model is Boehm's spiral development model[1]. Another is the commercial avionics suite of the aircraft, followed by the flight management system, then a particular box in that system, then one of the several processors in that box, and finally, the hardware and software associated with that processor.

An important feature of Figure 1 is that it nowhere represents a time sequence. In fact, it is intended to represent a generally parallel process. The message here is that development consists of an ongoing sequence of tradeoffs between desired system capabilities, design constraints, and actual capabilities of the available technology. Each tradeoff can encompass any number of layers of the development process.

Notice too that the outside influences on the project do not all apply to the top layer: they are distributed over all the layers. For example, a customer will have many specific requirements of the flight management system independent of the aircraft it is to be used in, and further, may insist on a particular software programming language, which applies directly to the implementation layer. This "lateral influence" has a profound impact on the traceability paths that must be supported: traceability does not just flow up and down the development layers.

This observation on the pervasive nature of requirements led us to revisit the old impasse on the definitions of, and distinction between, requirements and design. The word "requirements" is frequently interpreted to mean "functional requirements", which is itself a term open to misinterpretation. DOD-STD-2167A has helped this situation by using instead the term "capabilities", and we have adopted this convention. Many of the requirements mentioned in the preceding paragraph can be classified as design constraints. (The frequent complaint that the customer is messing with the design is a futile one—he who
Figure 1: The Layered Development Process
pays the piper calls the tune, and customers have every right to impose design constraints if they choose, as they always have, and always will. "Requirements", then, is a very broad term which we suggest subdividing as follows.

**Capabilities.** Capabilities describe what each system output is to provide (not necessarily as a function of system inputs), but not how it is to be accomplished.

Operational characteristics comprise that subset of capabilities (system outputs) that are directly observable by, and interact with, the system operator.

**Performance requirements.** The accuracy, timing (response or rate), range, and resolution of each system output are included in performance requirements. Performance requirements are always associated with one or more capabilities.

**Design Constraints.** The bounds within which the design must fit are identified as design constraints. Design constraints can be categorized broadly as technical and non-technical constraints.

Technical restrictions are placed on the system itself or its documentation.

Operational constraints apply to interactions between the system and any of its human operators.

System constraints will affect both hardware and software and/or tradeoffs between them.

Hardware constraints directly constrain the hardware design.

Software constraints directly constrain the software design.

Algorithms may be specified from outside the project to meet any of the capabilities.

Documentation requirements, including military and commercial standards, may be placed on the format or content of project technical documentation.

Non-technical restrictions are placed on the development process or the development environment.

**Descriptive Narrative.** Narrative which does not constitute requirements, but which gives background information is generally invaluable in understanding the requirements, and should be included or referenced in all derived specifications.

So, what is the distinction between requirements and design? The answer is that it depends on where you are standing in the layers of Figure 1. To someone who is concerned only with one particular layer, all the inputs in the above categories which that person receives from the surrounding layers are requirements, and what he or she does to satisfy them is the design. This design can then in turn create requirements for other layers.

In Figure 2, the elements of Figure 1 are expanded into more detail. Here, some of the specific types of outside influences are shown. The required capabilities analysis consists of identifying the required information interfaces and system capabilities, and enhancing them to account for additional interfaces and interface integrity requirements. The system physical analysis consists of choosing a system partitioning with physical interfaces and interconnects, and allocating the information interfaces and system capabilities to the physical units so created. The physical units are finally partitioned into hardware, software, and manual processes, to which the original system capabilities are allocated and developed into the actual designs. As each physical partitioning is established, new interfaces are created, giving rise to new required interfacing capabilities at the levels below. As with the overview diagram, there is no time sequence implied in this detailed diagram, and outside influences take effect at any level. Figures 1 and 2 are considerably more complex than earlier development process models. Our position is that this simply reflects the nature of the beast, and that the earlier models were too simplistic to be useful.

**Parallel Development**

Figure 3 illustrates a time sequence that goes along with the process of Figures 1 and 2. The three diagrams together represent a model that is more realistic than the earlier models. For the purpose of illustration, Figure 3 just shows the activities that are usually shown in the waterfall model, but it can be extended to cover all development activities. The columns represent the major development activities, and can be thought of as vessels all of which must be filled with work to complete the project. The curved lines represent the actual amount of work completed in each activity at given percentages of the total development calendar time, from the pre-proposal phase to project completion. Thus, all activities are progressing concurrently throughout the project. Certainly there is a path of peak activity which corresponds roughly with the sequence of the waterfall model, but this does not preclude all the other activities. Projects planned on
Figure 2: Layered Development Details

LEGEND

- Solid arrows: Relationship of influence from lower level to higher level.
- Dashed arrows: Relationship of influence from higher level to lower level.
- Stellar arrows: Inherent or late-arriving influence.
- Solid arrows with an arrowhead: Feedback relationship.
- Dashed arrows with an arrowhead: Unilateral feedback relationship.

Figure 2: Layered Development Details
a model such as this will properly anticipate the ongoing, parallel nature of development activities.

It is emphasized that the development process as described here encompasses the total development lifecycle including the pre-contract phase, which must be considered an integral part of the process. In order to make proposals—often with firm fixed-price quotations—many of the most important tradeoffs must be addressed in this phase. It is important that these tradeoffs be recorded, together with alternatives considered, and rationales for the choices made, so that later the development team (often not the proposal team) can re-evaluate them.

What does parallel development mean in practice? No one sequence of development will prove successful for all projects. One project might make its central goal the maximized reuse of existing units; another might have some severe customer-imposed physical constraints as its main challenge; and yet another might follow the "conventional" route of starting with a set of abstract required capabilities with few design constraints. These scenarios all require multiple tradeoffs to produce a successful design, but both the starting point and the particular tradeoffs needed are different in each case. Insisting that development must always start at the same point is as absurd as insisting that a jigsaw puzzle must always be started in the top, left-hand corner. Our development model must be flexible enough to accommodate any possible sequence, and the project database approach, which we propose here, being entirely non-sequential, provides this flexibility.

Multi-Disciplinary Development

We have already pointed out that development is a process of repeated tradeoffs over any number of layers, resulting in a process more parallel than sequential. This is not a new principle: it simply reflects reality better than traditional models. A corollary of this parallel tradeoff process is the multi-disciplinary nature of the process. Any time a capability cannot be implemented using the configuration currently chosen, or conversely, can be implemented better than anticipated, then new tradeoffs are needed, possibly involving software, hardware, test and test equipment, analysis, manufacturing, and any of the "ilities". For the same reasons, the contents of a document typically cover multiple disciplines, and its preparation should not be assigned to just one discipline.

With the technology we now have at our disposal, we should not assume at the outset that all of the capabilities (previously known as functional requirements) will be implemented in software. For example, we now
have Application Specific Integrated Circuits (ASICs), Programmable Logic Arrays (PLAs), and such special purpose devices as Fast Fourier Transform (FFT) processors. Any of these might well provide better implementations for a given capability than software in a general purpose processor. It is very likely that the technology will continue to grow in these directions, making it increasingly important that choices of technology must involve, at the very least, systems, hardware, and software people.

**THE PROJECT DATABASE**

The creation of information during a systems development project has a protracted analogy with the definition, population, and management of a large database, as illustrated in Table 1. In both of these, information is derived from many sources, its integrity maintained to support consistency and unforeseen uses, and its acquisition carefully managed. Notice that the data involved is not just text: it includes figures, code, and everything else that represents the system development. It is useful then, to view the system development process as the gradual population of a large database. With this view, the database becomes the infrastructure of the project. Entities in the database are developed in response to steps populated during the system development process. Very often, dependencies exist among these entities, hence the population of the database is correctly viewed as concurrent (parallel), where the existence of one piece of information stimulates the creation of “contiguous” pieces.

The key element in the database view of systems development is that development documents are generated by (macro) queries of the database. This view is particularly useful in terms of consistent (and non-redundant) information, which are normally the domain of SQA and Configuration Management. The database view can also serve as the basis for project management metrics: for example, milestone completeness is the extent to which various relations are populated. (Quality may be taken as the extent to which integrity is maintained.)

As might be expected from the analogy between the development process and the database, the process model described by Figures 1 and 2 can be used rather directly as the source for an entity-relationship diagram defining the requirements of the project database. This requirements definition is now being developed at Smiths Industries.

**METHODS AND CASE TOOLS**

There are now abundant development methods and automated tools available, most of whose proponents claim their particular approaches will solve all our problems. We have found these claims to be false. For example, a number of CASE tools claim to automate the real-time requirements specification method[6], developed at Smiths Industries, but all have so far failed to do so adequately. The two we have used in practice have been useful, but, because of their deficiencies, not overwhelmingly so.

In the course of developing and using our own methods, examining other methods, and evaluating many CASE tools, we have acquired a great deal of experience in the method and tool evaluation process. Other companies have made major commitments to CASE tools without such comprehensive evaluations, and have discovered too late that they have made a bad investment. The Research and Technology Institute of West Michigan (RTI), in Grand Rapids, Michigan, is establishing a CASE tool laboratory which will make many tools available for comparative evaluation. This will be a valuable resource for anyone wishing to make an objective choice.

One of the important lessons we have learned from our evaluation experiences is that, before choosing methods
and tools, the total development process to which they will be applied must be well established[5]. Once this has been done, the exact role of the methods and tools within that process can be defined.

From the development process definition described here, it becomes even more clear that no existing method alone, nor any CASE tool we have seen so far, covers more than a small part of the process. The combination of the requirements and architecture models[6] comes closer, but so far, no tool has attempted to automate the latter. In addition, support for testing, data modeling, and the "ilities" mentioned earlier must be provided, and the tools must integrate with a project database, and with existing CAE tools that are used in electrical and mechanical design. Much work needs to be done to define how this will be achieved, and in the mean time, development methods and CASE tools should be judged according to the limited segment of the total process they support. Some of these judgements follow.

The real-time analysis methods described by Ward and Mellor[11], Harel[4], and by Pirbhai and myself[6], and tools that support them are suitable for the capabilities specification of one level at a time of Figure 1. They are not well suited to representing design constraints and designs: the architecture model of [6] is specifically intended for this task, at least at the higher levels, but so far it is not supported by any tool. For detailed design at the hardware/software levels, specific methods such as structured design, its real-time extensions, and object-oriented design are appropriate, but these have not yet been well integrated with the analysis techniques either in the methods or tools.

A number of tools are providing an "executable specification" capability, which can be useful as a dynamic consistency check of the model, but is not applicable to the higher levels of Figure 1. At these levels, the requirements are necessarily not specific enough to be executable. Furthermore, executing a model provides little or no information about the actual performance of the system, which is determined by the target processor, physical interfaces, and software design. Thus, executable models have their place, but it is a very small place, and we should not expect too much of it.

CONCLUSIONS

Our principal conclusion with regard to CASE is that the CASE industry as a whole has been guilty of a cart-before-horse mentality, by seeking to provide a solution before the problem has been defined. A description of the total development process, such as the one described here, provides a reference model for both evaluating and developing CASE tools, whereby both the developer and the user will have a clear picture of a tool's role in the process, and false claims and expectations will be avoided.

We are continuing to develop the model at Smiths Industries: in particular, we are developing the specification for the project database, in preparation for designing this database for use throughout our development projects, in conjunction with whatever CASE tools we decide to use.

ACKNOWLEDGEMENTS

I am deeply indebted to Smiths Industries for permission to publish this paper: to my colleagues—Andy Arkusinski, Rich Oom, and Dave Zeitler—who participated in the development process study from which the paper was derived; and to Paul Jorgensen of the Research and Technology Institute of West Michigan who also participated, and who gave us valuable insights into the project database approach.

REFERENCES

