Artificial Intelligence and Systems Engineering

Ian Sommerville,
Computing Department, Lancaster University, LANCASTER LA1 4YR, UK.

Abstract. This paper discusses the problems of applying artificial intelligence technology in the domain of systems engineering. The different process models used for systems engineering and AI are discussed and it is suggested that these differences are largely responsible for some of the mutual hostility which exists between the systems engineering and AI communities. Contributions of AI in this area are discussed , particularly the applications of AI in the programming process. The technical difficulties of applying AI to support software engineering are covered and, the final section of the paper, looks forward to suggest systems engineering activities where future application of AI technology may be fruitful. Without exception, these activities are dominated by human rather than technical considerations.

1. Introduction

Ten years ago, AI came to life after a number of years in the doldrums. The Japanese Fifth Generation Initiative sparked off research programmes in the US and in most European countries. Direct responses included the Alvey initiative in the UK and the European ESPRIT initiative, both of which made relatively large sums of research support available with the ultimate aim of turning AI research into commercial products.

The proposals for systems engineering research were based on the notion of integrated project support environments which provided automated support for systems development activities. It was envisaged that research prototypes of so-called third-generation environments, based on AI technology, should be available about now. In fact, there are few second-generation environments available and there is no realistic prospect of developing an AI-based environment in the foreseeable future.

Of course, AI was oversold and has not met its expectations. Of course, these expectations were unrealistic. Of course, the initial flush of funding was not followed up by the steady influx of support which is required to take good ideas through working prototypes to commercial products. Nevertheless, in many areas AI has made an impact and, while the unrealistic enthusiasm has faded, in many application domains there is a feeling that 'knowledge-based systems' do have a role to play.

Why then, in a recent seminar attended by many leading software engineering researchers to discuss the future of software engineering [1] was AI never mentioned? Why is there a perceptible hostility between many systems engineering practitioners and AI researchers? Why do some engineering disciplines embrace AI enthusiastically but software and systems engineers ignore the possibilities of this technology? What of the future - can AI make a contribution to systems engineering?

This paper is neither a general discussion of how AI might be applied to software engineering nor a general survey of work in this area. There have been various general papers [2, 3, 4] and collections of papers summarising the state-of-the-art [5, 6]. Rather, I address

the above questions from a very subjective viewpoint. My perspective is not that of an AI specialist but that of a software engineer who has been involved in the systems engineering process. At the same time, I have some background knowledge of AI and, I hope, I lack the jaundiced view of some of my systems engineering colleagues. Existing approaches to systems engineering support are running out of steam as it becomes clearer that the real problems of systems engineering are human problems which cannot be tackled by formalisation. Perhaps AI technology with its foundations in the study of intelligence can now be applied effectively in systems development?

The paper is divided into three principal sections. The first section discusses the systems engineering process and contrasts this with the development process for AI systems. The mutual suspicion which is sometimes evident between AI researchers and systems engineering practitioners is, in my view, caused by a misunderstanding of the central role these process models play in the respective types of system development.

In the second section, I look at what work has been done in applying AI to systems (mostly software systems) engineering. The application of AI in software engineering has not been particularly fruitful at least as far as practical applications are concerned. Recently, other engineering disciplines have turned to AI as a potential tool for tackling their problems. Again, practical applications are rare but there is noticeably more enthusiasm for AI amongst 'traditional' engineers than software engineers. I believe that there are fundamental reasons why AI is likely to be more effective in 'traditional' engineering and I discuss some of these reasons here.

In the final section, I present three important problems areas of systems engineering which I believe are amenable to support using AI-based tools. Without exception, these problems are human problems (or, at least are dominated by human behaviour) and I see no realistic way to tackle these problems without AI technology. Most of the discussion is concerned with why these problems can (and should) be addressed using AI technology but some AI research which has considered these problems is briefly discussed.

2. Systems Engineering

Systems engineering is concerned with the construction of systems which deliver some useful function. These range from relatively simple systems (a word processor, a television, a lawnmower) to immensely complex systems such as an air traffic control system, a broadcasting network, or a warship. In general, all but the simplest systems now include:

1. Mechanical components which transfer (and perhaps transform) energy from one part of the system to another or to and from the system's environment.
2. Electrical components which transfer (and perhaps transform) signals from one part of the system to another or to and from the system's environment.
3. Software components which control the mechanical and electrical components and which specify the information transfer and transformations which take place in the system.

Many systems also involve complex civil engineering to provide a suitable environment for these components and their interface to their environment or may themselves be structures such as bridges. As I have no experience of this class of system, they will not be discussed in this paper.

Systems are normally procured and built using a process whose general form is shown in Figure 1.

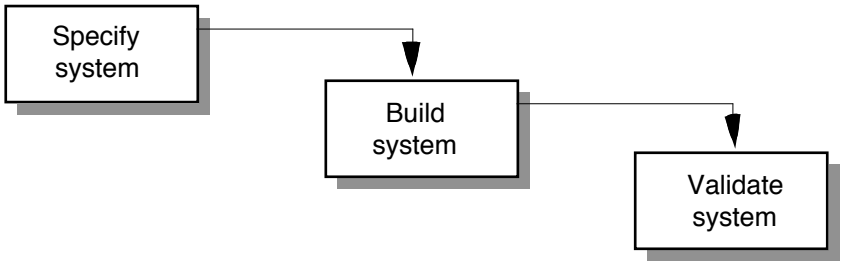

Figure 1 The systems engineering process

This process has both significant strengths and weaknesses. The strengths of this process are:

1. It provides an effective basis for system procurement. Organisations who require a system must develop the system specification but can then contract out the development and the validation of the system to other organisations. Thus, organisations concerned with banking (say) need not maintain in-house teams of mechanical and electrical engineers to design and build their cheque sorting system. Without this model, specialist engineering companies could not exist.
2. It provides a basis for the division of labour. The expertise required to develop all but the most trivial systems is such that they cannot be developed by a single individual but must involve teams of engineers drawn from different backgrounds. Given that the interfaces between the system parts can be specified, the mechanical, electrical and software components can be built and (to some extent) validated concurrently.
3. It provides a basis for the training of engineers. Typically, engineers learn about the process by assisting with the building of systems and, only after the problems of system building have been understood, do they move onto other activities. Thus, designers and those involved in system specification should have an understanding of how the limitations of the building process constrain the system design.

This process is almost universally applied in systems engineering and works reasonably well when the systems are mechanical or electrical systems built out of tangible physical components. In these systems, there is a clear relationship between the system specification and the system design. Indeed, the specification is usually simply expressed as a design schematic and the design process (which is encompassed in the 'Build system' box above), involves developing the details of the specification. The validation process for these systems, which involves checking that the built system meets its specification, is mostly concerned with checking for defects introduced during the building process. Because the gap between design and specification is narrow, it is assumed that the design is 'correct'.

This process becomes problematical when systems include components such integrated circuits or software which are intangible. The major weakness of the process is that it forces premature commitment to a specification before the needs of the system procurers are fully understood. Specifications are usually expressed informally with attendant inconsistencies and incompleteness. There is a wide gap between the specification and the design with no clear distinction (in software) between the design and ‘manufacturing’ processes. Validation is concerned with discovering design rather than manufacturing problems and hence impacts the specification.

The AI community recognised these difficulties at an early stage and has developed an exploratory process model for software which is quite different from that in Figure 1. This is shown in Figure 2.

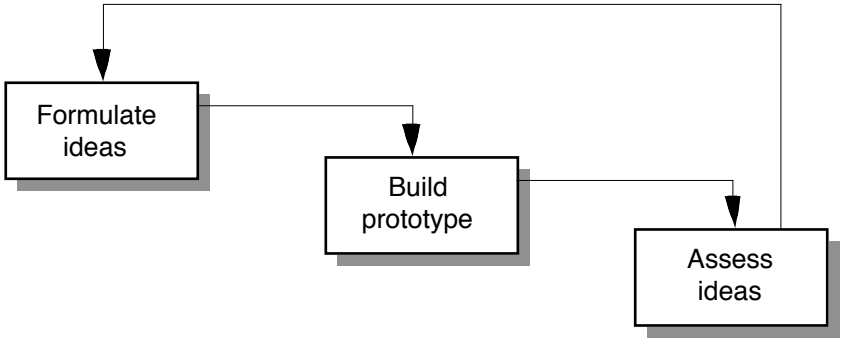

Figure 2 Exploratory development

This model rightly recognises that it is often extremely difficult to develop a software specification partly because of the intangible nature of software (specifiers can't visualise what it will do) and partly because the function of software is a control function. The number of control possibilities is much larger than the number of practical ways of building the mechanical or electrical parts of a system and specifiers have no mechanisms to assess which of these is most effective or even what their control requirements are.

Although the exploratory programming model is, perhaps, a more realistic technical approach to software development, it does not allow two critical questions to be answered:

1. What system is to be built?
2. Has the system been built as specified?

These questions are absolutely central to the engineering process as they allow engineering to be an economic and not just a technical activity. I believe that much of the mutual suspicion which exists between the systems engineering and the AI communities is a direct result of the mismatch between these process models. Although it can be argued that the specify-build-validate approach to systems development must evolve towards the exploratory development model as systems become more and more software-intensive, the current approach is so deeply embedded in our organisations and in our economy that it is unrealistic to think that it will change significantly in the foreseeable future.

The mismatch between these models is also revealed in the way in which each community thinks about programming. For software engineers, programming is a relatively simple activity which consumes, normally, less that 20% of the system development effort. The challenges to the software engineer are in specification and design and in system validation. By contrast, AI systems development is programming so we have therefore seen the development of very powerful languages (such as Lisp and Prolog) which simplify the programming process. This bias towards programming has influenced the type of software engineering problems which have been tackled by AI researchers.

3. The contributions of AI to systems engineering

It can be argued that AI has done an immense amount for system engineering. Although the origin of object-oriented programming is a matter of some debate, clearly the contribution made by frame-based systems was very important. Object-oriented development will become, over the next 10 years, the principal development approach for large systems. Another important contribution was the influence of AI programming environments such as InterLisp on environments for software engineering and CASE tools.

The problems concerned with the engineering of software dominated systems have been recognised for 25 years and AI research has been going on for rather longer than this. However, relatively few AI researchers have concerned themselves directly with systems engineering problems. Furthermore, AI research which has been concerned with the domain of systems engineering has sometimes been less concerned with addressing practical problems than with modelling programming as a human activity [7, 8]. This is, of course, a perfectly reasonable research objective but it has not yet led to any exploitable results.

Until relatively recently, most AI work in this domain focused on software engineering and was dominated by research on automatic programming [9, 10]. Automatic programming is concerned with utilising knowledge-intensive methods to derive an efficient program from a formal specification. This serves a dual function:

1. It partially addresses the problems of specification. If a formal system specification is available, this provides a less ambiguous basis for developing the system than a specification expressed using diagrams and natural language. The state of the art in this area is such that formal specifications can only be produced for some classes of system and can only express the system's functional characteristics.
2. It reduces (or should reduce) the cost of system building and validation. Programming is automatic and correctness-preserving transforms are used to convert the specification to the program. If these transforms can be validated, there is no need to demonstrate that the program meets its specification.

Work on automatic programming led to the development of a commercial product called REFINE [11] which synthesised Lisp and C programs from formal specifications. However, this product has had little or no impact on systems development and has not been a commercial success. By all accounts, REFINE worked reasonably well so why has this system (and automatic programming in general) had little or no impact on software development? Leaving aside defects in the REFINE system, why are software engineers almost completely uninterested in automatic programming. I believe that the reasons for this are:

1. Automatic programming is concerned with *software* rather than *systems* engineering. A program could be generated but there was no straightforward way to interface this program with any hardware whose characteristics were unknown to the automatic programming system. Hardware interfacing required the generated code to be changed but the workings of this code were opaque to the engineers responsible for making the change.
2. The specification languages and techniques which can be processed efficiently by automatic programming systems are limited. This means that there is a high cost incurred in translating user requirements into the specification language. Of course, this is a problem with all formal specification but is worse when the specification has to be transformed to a reasonably efficient program. Other problems with specification languages include a lack of modularity so that it is difficult to generate the system incrementally.
3. Transforming a specification to a program is a slow process irrespective of how fast the underlying hardware. This means that there is a high overhead associated with fault repair. This is unacceptable to most engineers who like to fix faults discovered during the development process as quickly as possible.

The long-running Programmer's Apprentice project at MIT has the objective of supporting all stages of the software process. Based on a model of programming through

clichés which Soloway and others suggest closely follows the approach used by human programmers [12, 13], the Programmer's Apprentice is a knowledge-driven system (more accurately, a number of separate systems) to support software development. A programming cliché is a representation of a 'common' programming construct such as loop iteration. A notation called the Plan Calculus is used to represent programming clichés. Waters argues [7] that programming is a process of combining these clichés into programs.

The first stage of the Programmer's Apprentice project was concerned with providing support for programming. A system called KBEmacs (Knowledge based EMACS) was developed to the stage where reasonably complex Ada programs can be synthesised from a small number of user commands.

This approach is distinct from automatic programming as it does not rely on formal specifications to generate the program and because it allows end-user modification of the generated system. Rather, an informal specification is used to select and adapt clichés from the knowledge base and these are combined to create the program.

Although KBEmacs can generate non-trivial programs (up to several hundred lines of code), the usefulness of the system for practical software engineering is limited. The cost of discovering clichés increases as they become more abstract yet the return on these more abstract clichés is reduced as more abstract constructs are less frequently used. KBEmacs may save a little programming effort but these savings may be lost if the generated program has to be manually optimised.

More recently, work on the Programmer's Apprentice project has switched to producing a system called the Requirements Apprentice which supports the specification stage of the software process. I believe this is a more important aspect of the Programmer's Apprentice project and discuss it briefly in the following section.

I believe that the focus by AI researchers on the programming process has, in fact, actively retarded the application of AI to systems engineering. Practical software engineers dismiss the research as it is not concerned with the real problems in this area and hence reject AI technology as a possible solution.

By contrast to the attitudes of software engineers, other engineering disciplines have been enthusiastic about the application of AI to their work. As early as 1986, a special issue of IEEE Computer [14] was devoted to expert systems in engineering. Engineering design problems have been addressed by the application of AI techniques. For example, the PIAF system is concerned with layout design for integrated circuits [15], the detailed design of air cylinders is supported by the AIR-CYL system [16], the Pride system [17] is concerned with the design of paper feeders for photocopiers and the SIMAD system, described elsewhere in these proceedings [18] is a system which suggests improvements to the design of axisymetric assemblies.

3.1. Problems of applying AI to support software development

Why are there differences between 'traditional' engineering and software systems engineering? Why are software engineers lukewarm about AI whereas other types of engineer see it as a key technology for the future development of their discipline? What is it about software that makes it different? I believe that there is an important human reason for this and several technical reasons.

I believe that a major handicap to the application of AI in software engineering is that AI researchers are themselves software engineers! Hence, of course, they know (or think they know) the problems of software engineering. Unfortunately, as discussed in the first part of this paper, the type of software that AI researchers build and the development process which they follow is atypical. Their idea of priority problems may not be the same as systems

engineering practitioners. By contrast, when AI researchers tackle problems in other engineering disciplines, they know that they are not domain experts and are perhaps more willing to listen to domain experts and to learn about the real problems which they face.

Although this human problem is a real and an important one, there are also important technical reasons why the application of AI in mechanical or electrical systems engineering is likely to be more successful than in software systems engineering.

1. *Well-founded theory* Software systems design is not founded on any verified theory whereas electrical and mechanical systems are clearly bound by physical laws. While there is little direct reasoning in engineering systems based on these physical laws, the laws allow some aspects of a design to be verified. These laws are analogue rather than digital so that computations over a design can give some approximation of 'goodness' and designs can be compared (albeit approximately) according to these computations. Design proposals produced by a system may be validated and invalid suggestions factored out at an early stage. By contrast, we have no useful metrics which can detect poor software designs so heuristics to prune the design search space are very difficult to establish.

2. *Focused, well-understood domains* All of the successful examples of AI applications in engineering are concerned with very specialised domains. For example, the AIR-CYL system [16] is only concerned with the design of air cylinders and our work at Lancaster has been concerned with the design of axisymetric assemblies such as those in pumps [18, 19]. The emphasis on specialised domains means that detailed knowledge can be brought to bear on the design process.

   Domains in engineering design are easy to identify as they can be associated with tangible products. In software systems, on the other hand, it is much harder to associate systems with a specific domain as the approaches used to design have a great deal in common. For example, on the surface, it would appear that a library system and a radar system have little in common but, as Reubenstein and Waters point out [20], both are examples of 'tracking' systems as both keep track of the state of some target object.

   Unfortunately, this general knowledge of software design techniques is not enough for good systems design; specialised knowledge is also necessary. Understanding and capturing this intangible specialised knowledge is itself difficult. Furthermore, both general design knowledge and the relationships between this general knowledge and the specific domain information are very difficult to encapsulate as system-processable knowledge.

3. *Clearer entity and relationship semantics* Because engineers deal with tangible physical objects, they can generally agree on what these objects are. Ignoring tolerances, there is little dispute over the form of a shaft, the function of a nut or a switch or the relationship of abutment, say. By contrast, entities and relationships in software engineering are abstractions and hence it is more difficult to establish agreed semantics for them.

While future prospects for using AI in the design of the electrical or mechanical parts of systems are reasonably good, I believe that these factors mean that the successful application of AI to software design problems (where programming is really detailed design) cannot be realised in the short to medium term. A necessary pre-condition for the application of AI in any design problem is that we have to have a clear understanding of what is to be designed and the characteristics of a 'good' design. Software engineers must understand their discipline much better before AI techniques can be successfully used.

5. What can AI do for systems engineering?

In an introduction to a special session on AI and Software Engineering at the 9th International Software Engineering conference, Barry Boehm, who is a leading and highly regarded US software engineer, suggested that there are two possible ways of bringing AI and software engineering together.

1. The AI/SE approach. Reformulate software engineering processes in AI terms and attempt to solve them entirely within AI.
2. The SE/AI approach . Select a subset of software engineering problems and apply ideas, techniques and representations taken from AI in the solution of these problems.

Because of the distinctions between the process models discussed above, the AI/SE approach cannot be applied effectively as it does not recognise the importance of an agreed specification and validation against that specification. Irrespective of the technical merits or otherwise of this approach (and its technical superiority has not been demonstrated), it is simply not worth considering it further; there is no way in which it will be acceptable to the general community of engineers.

Prospects for exploitation of AI techniques and technology therefore must lie within the SE/AI approach. Rich and Waters [21] suggest that tools to support software engineering must become knowledge intensive if they are to make an improved contribution to the software development process. For the reasons discussed above, I do not believe that the software design process is amenable to such tool support. We therefore have to look at other, non-technical areas of systems engineering to see what knowledge-rich tools might be developed.

To assess the realistic potential of using AI to support the systems engineering process, we have to look at problem areas in systems engineering which are firstly, critical to the systems engineering process and secondly, where support using 'conventional' tools is limited. Three areas seem to me to be the best candidates:

1. Requirements analysis and definition.
2. Process modelling and process support.
3. Project planning.

Work in these areas of systems engineering requires the use of an incomplete, normally inconsistent and rapidly changing base of information. It is impossible to predict in advance what information will be required in a particular situation or when information may be important. Apart from very simple, routine tools, such as PERT analysers which assist with resource management and planning, the unpredictability and the complexity of the information base, makes it difficult to develop effective, conventional software tool support.

5.1. Requirements analysis and definition

The first stage of the systems engineering process is a requirements analysis and specification stage where the needs of the system procurer are analysed and a system specification produced which serves as a contract for the system development. This stage involves the following activities:

1. *Domain understanding*. The analyst has to understand the domain where the system is to be introduced. Normally, this understanding is incomplete and changes as the requirements are developed.

2. *Information collection*. The analyst has to collect information about the functional and non-functional characteristics of the system which is to be procured and locate this information in the context of the domain model.
3. *Systems modelling*. The analyst has to create an abstract model of the system which is to be procured. This stage usually involves making decisions about whether functions should be partitioned to mechanical, electrical or software components.
4. *Requirements documentation*. The analyst has to write a requirements document which expresses the requirements in a form which can be agreed between procurer and developer.

There is an obvious relationship between the first three of these activities and some of the activities involved in AI systems development. Practitioners in AI have realised for many years that successful systems must be domain specific and must incorporate a large amount of domain knowledge. The activity of information collection is essentially the same process as knowledge elicitation and has exactly the same problems. Systems modelling, in AI systems development, corresponds to the implementation phase of AI systems - in this case an executable system model is built. Systems modelling is also an iterative process with the model being refined as more information becomes available.

Applying AI technology in this area is appropriate because requirements are never 'correct'. Rather, they must adequately express the needs of the software procurer. If AI-based tools are applied to support programming and, because of knowledge-base inadequacies either fail to generate a program or (even worse) generate the wrong program, these tools will be rejected by systems engineers. On the other hand, requirements analysis and specification is an iterative process where requirements are derived from several sources all of which usually produce incomplete and often inconsistent requirements. One of these sources could be an AI-based tool. Given that the tool is functionally useful, inadequacies due to a lack of knowledge simply mean that missing information has to be discovered from another source.

Recent work on the Programmer's Apprentice project has been concerned with the development of a Requirements Assistant which is intended to help the analyst during the requirements derivation process [20, 21]. Like KBEmacs, this system is also based on clichés and uses a multi-layer model of knowledge. In this case, however, the clichés known to the system can be (incomplete) high-level domain abstractions such as the tracking system discussed above or a repository system which manages a collection of objects. Different application systems can be viewed as different cliché combinations.

The Requirements Assistant (RA) allows the analyst to introduce concepts as keywords and it searches its knowledge base to discover clichés representing these concepts. The RA does not insist on formal and complete specifications but allows informal specifications to be written. Support is provided for disambiguation and the RA keeps track of pending issues e.g. unknown concepts which have to be filled in by the analyst. Changes to the requirements can be proposed and the RA assesses the impact of these changes and updates the system knowledge to make it consistent with the proposed change. Finally, the system can generate a report in a form which is acceptable as a system specification.

Using a simplified knowledge representation, Czuchry and Harris [22] have commercialised some of the work on the Requirements Assistant and report on its practical use in real systems engineering projects. This has demonstrated the potential of this approach and I believe that applications in requirements analysis and specification will be the most significant applications of AI to systems engineering in the next few years.

5.1. Process modelling and support

The systems process is the set of activities and outputs which are involved in systems development. As discussed above, this is usually partitioned into three stages namely specification, development and validation. Within these large-grain activities, however, there are many sub-activities which depend on the system being developed, the organisation involved in the development, the people involved in the development, the tools which are available, etc.

There is a great deal of interest in the software engineering community in modelling, controlling and standardising the software engineering process [23]. This work has mostly focused on the technical activities of software development and the outputs of development processes. However, empirical process studies, such as that carried out by Curtis *et al.* [24] have shown that the factors which are most critical for project success are human and organisational rather than technical factors.

The characteristics of real software processes rather than idealised process models are:

1. They involve a 'working' rather than a formal division of labour [25]. Individual engineers are assigned personal tasks but a project team often reallocates these tasks implicitly to suit current circumstances. For example, if an engineer is temporarily unavailable, some of his or her work may be taken on by others without explicit managerial intervention.

2. They are often exception-driven. While there may be a prescribed process expressed as a sequence of activities, so many exceptions occur that this process is continually changing as the team cope with whatever exceptional circumstance has arisen. The project manager and the development team will often spend a significant amount of time discussing how to cope with the exception.

These characteristics suggest to me that the use of AI techniques may be valuable in process support. Again, we are not looking to support systems to provide a 'correct' answer. Rather, such systems have to provide informed suggestions how to proceed in a process or how to adapt a process to a particular set of circumstances. The use of AI techniques with a process knowledge-base means that the range of possible support is potentially greater than that possible with conventional data-bases.

Hardly any software process research has taken developments in AI into account (again an illustration of the rift between disciplines). Mi and Scacci [26] describe a prototype knowledge-based environment which provides process definition facilities and Jarke *et al.* [27] have also developed a prototype system which provides knowledge-based support for the software process.

Like other process support systems, these systems allow a model of the software process to be expressed and modified. However, because they use a knowledge-base of process information, they can also provide support for assessing the impact of changes to this model and, in some cases, automatically propagating changes through the model. Thus, when the process changes (as it always does), the process model can be adapted quickly to reflect the new situation.

Mi and Scacci's system (the Articulator) is particularly interesting because it recognises systems and not just software engineering as a process and also the fact that processes take place in an organisational context. Relations between process agents, tasks and organisational agents can be established and organisational information used to resolve process problems.

5.2. Project planning

In a large systems development project, there may be an overall project manager plus project managers for individual sub-systems making up the system. These managers may be

concerned with mechanical, electrical or software systems development and may not understand much about the problems faced by project managers from other disciplines. As Hakami [28] points out, simple project management tools such as PERT tools are useful for an individual manager but, when project management is, essentially a group responsibility, these tools do not address the problems which project managers must face.

Existing project management tools have a very simplistic view of the process. They consider resources to be independent and generally only allow a single type of dependency relationship between tasks to be modelled. In fact, resources such as the time of specialist staff, available computers and the travel budget are all inter-dependent and teasing out the dependencies between them is a difficult task.

The general problem faced by project managers is planning work in an uncertain situation. He or she must often attempt to satisfy conflicting organisational, project and human objectives and, in general, must minimise project risk. That is, the consequences of whatever decisions are made should be predictable. Project planning goes on throughout a project and plans must be regularly recast as milestones are reached, resource availability changes and organisational priorities are modified. To plan effectively, managers need to be able to assess the impact of plan changes and propose and compare alternative plans. Yet again, correctness is not the issue; rather, informed support which managers can augment is what is required.

There has not been much research into how to provide support for project planning. Hakami [28] and Bimson and Burris [29] have developed systems to assist with the project planning process. Hakami's system is perhaps the more interesting as it explicitly recognises that project management is a cooperative process and it provides facilities for modelling process cooperation. The system is an agent-based system with different agents allocated to each planning role in the project. A set of constraints governs agent operation.

Conclusions

Although techniques with their roots in AI such as object-oriented development have entered mainstream systems engineering, the application of AI in support of the systems engineering process has not been successful. I have argued that this is partly a consequence of AI researchers misunderstanding the central role of the specify-build-validate process model in systems engineering. Attempting to recast systems or software engineering problems into the more familiar exploratory programming model of AI is a futile approach. While it may allow interesting studies into the nature of software engineering activities, it will not provide for effective activity support.

I do not think that any kind of automated software design system is likely to be useful in the foreseeable future. In specialised domains, hardware design systems have more potential, but the practical problems of integrating these systems with existing CAD tools have still to be addressed.

Unlike, perhaps a majority of systems and software engineers, I am optimistic about the role that AI could play in future tool developments for systems engineering. The contribution of AI will be in supporting non-technical rather than technical activities. These activities are characterised by solutions to problems which are neither right not wrong but which are more or less appropriate for a particular situation. Conventional algorithmic approaches cannot tackle this kind of problem effectively. Particular activities where AI techniques can be applied are requirements specification and analysis which involves extensive consultation with domain experts and in project management.

References

[1] N. Habermann and W. Tichy, Future Directions in Software Engineering, Dagstuhl Seminar Report, 32, February 1992.

[2] H.A. Simon, Whether Software Engineering Needs to be Artificially Intelligent, *IEEE Trans. on Software Engineering,* 12 (7), 1986, 726-732

[3] I. Zualkernan, W.T. Tsai, and D. Volovik, Expert Systems and Software Engineering: Ready for Marriage?, *IEEE Expert,* 1 (4), 24-32.

[4] G. Arango, I. Baxter and P. Freeman, A Framework for Incremental Progress to the Application of Artificial Intelligence to Software Engineering, in *Artificial Intelligence and Software Engineering,* Ablex Publishing, New Jersey, 1991.

[5] D. Partridge, *Artificial Intelligence and Software Engineering,* Ablex Publishing, New Jersey, 1991.

[6] C. Rich and R.C. Waters, *Artificial Intelligence and Software Engineering*, Morgan Kaufmann, Ls Alton, 1986.

[7] R.C. Waters, The Programmer's Apprentice: A Session with KBEmacs, *IEEE Trans. on Software Engineering,* 11 (7), 1985, 11296-1320.

[8] D. M. Steier and E. Kant, The Roles of Execution and Analysis in Algorithm Design, IEEE Trans. on Software Engineering, 11 (11), 1985, 1375-1385.

[9] R. Balzer, A 15 Year Perspective on Automatic Programming, *IEEE Trans. on Software Engineering,* 11 (7), 1985, 1257-1267.

[10] D.R. Smith, KIDS: A Semiautomatic Program Development System, *IEEE Trans. on Software Engineering,* 16 (9), 1990, 1024-1043.

[11] L. Abraido-Fandino, An Overview of REFINE 2.0, *Proc. 2nd Int. Symp. on Knowledge Engineering,* Madrid, 1987.

[12] E. Soloway and K. Ehrlich, Empirical Studies of Programming Knowledge, *IEEE Trans. on Software Engineering,* 10 (5), 1984, 595-609.

[13] B. Adelson and E. Soloway, The role of Domain Experience in Software Design, *IEEE Trans. on Software Engineering,* 11 (7), 1985, 1351-1360.

[14] Expert Systems in Engineering, *IEEE Computer* , Special issue, 19 (7), 1986.

[15] M.A. Jabri and D.J. Skellern, PIAF: Efficient IC Floor Planning, *IEEE Expert,* 4 (2), 1989, 33-45.

[16] D.C. Brown and B. Chandrasekaran, *Design Problem Solving: Knowledge Structures and Control Strategies*, London, Pitman, 1989.

[17] S. Mittal, C.L. Dym and M. Morjaria, PRIDE: An Expert System for the Design of Paper Handlers, *IEEE Computer*, 19 (7), 1986, 102-114.

[18] V. Oh,I. Sommerville, M. French and A. Taleb-Bendiab, Incorporating a Cooperative Design Model in a Computer-Aided Design Improvement System, *Proc. AISB Conference* Birmingham, 1993.

[19] A. Taleb-Bendiab, V. Oh, M. French and I. Sommerville, Knowledge Representation for Engineering Design Product Improvement, *Proc. Applications of Artificial Intelligence in Engineering VII,* Elsevier, New York, 1992.

[20] H.B. Rubenstein and R.C. Waters, The Requirements Apprentice: Automated Assistance for Requirements Acquisition, *IEEE Trans. on Software Engineering,* 17 (3), 1991, 226-240.

[21] C. Rich and R.C. Waters, Knowledge-Intensive Software Engineering Tools, *IEEE Trans. on Knowledge and Data Engineering*, 4 (5), 1992, 424-430.

[22] A.J. Czuchry Jr. and D.R. Harris, KBRA: A New Paradigm for Requirements Engineering, *IEEE Expert,* 3, (4), 1988, 21-35.

[23] W. Humphrey, *Managing the Software Process,* Addison Wesley, Reading, Mass., 1989.

[24] B. Curtis, H. Krasner, and N. Iscoe, A Field Study of the Software Design Process for Large Systems, *Comm. ACM.*, 31 (11), 1988, 1268-87.

[25] R.J. Anderson, J.A. Hughes, and W.W. Sharrock, W. W. , *Working for Profit: The Social Organisation of Calculability in an Entrepreneurial Firm*, Aldershot: Avebury, 1989.

[26] P. Mi and W. Scacci, A Knowledge-based Environment for Modeling and Simulating Software Engineering Processes, *IEEE Trans. on Knowledge and Data Engineering*, 2 (3), 1990, 283-294.

[27] M. Jarke, J. Mylopoulos, J.W. Schmidt and Y. Vassilou, DAIDA–An Environment for Evolving Information Systems, *ACM Trans. on Information Systems,* 10 (1), 1992, 1-50.

[28] B. Hakami, ISM: A Knowledge-based Project Support System, in *Software Engineering Environments,* ed. K.H. Bennett, Ellis Horwood, Chichester, 1989.

[29] K.D. Bimson and L.B. Burris, Assisting Project Managers in Project Definition, *IEEE Expert,* 4, (2), 1989, 66-76.