Mathematical Foundations of Software Engineering: a Roadmap

Tom Maibaum

Key Research Pointers

- Representing behaviour (including concurrency and duration of activities) and being able to analyse it. There may be many different kinds of behaviour and there is no obvious necessity to have a universal representation - quite the opposite! Engineering tools are the most useful when they are specific, so different classes of problems may demand differing languages to represent them.
- Representing 'ility' properties and devising corresponding engineering theories enabling the use of the 'ility' in design.
- Systematising domain knowledge for the area of application. This is hard, long and often tedious work.
- Defining the specification and refinement patterns/architectures required to encapsulate design choices. This results in support for the 'cookbook' aspects of normal design. The work on product line architectures, if properly driven towards formal engineering systematisation, will contribute enormously to this.
- Better understanding of modularity principles. The only effective method for dealing with the complexity of software based systems is decomposition. Modularity is a property of systems, which reflects the extent to which it is decomposable into parts, from the properties of which we are able to predict the properties of the whole. Languages that do not have sufficiently strong modularity properties are doomed to failure, in so far as predictable design is concerned.
- Improved and specialised analysis tools, many more abstraction (interpretation) tools to address feasibility/tractability of analysis; more and specialised decision procedures for interesting properties (using abstractions to approximate same).

The Author

Tom Maibaum was appointed to the Chair of the Foundations of Software Engineering at King's College London in 1999. From 1981 to 1999 he was a Lecturer, Reader and Professor (of the Foundations of Software Engineering) at the Imperial College of Science Technology and Medicine, University of London, serving as head of department from 1989 to 1997. Previously, he held appointments at the University of Waterloo, Canada, from 1973-1981. He holds a BSc in Mathematics from the University of Toronto and a PhD in Computer Science from the University of London. He is a Chartered Engineer and a Fellow of the IEE and the Royal Society of Arts. He is an Honourary Professor of the Pontifícia Universidade Católica do Rio de Janeiro and has been a Visiting Professor there on several occaisions. He has been a Royal society Industrial Fellow, a Pierre and Marie Curie Fellow and a recipient of a Royal Academy of Engineering Foresight Award. His interests include the meta-theory of specification, requirements specification, semantics of object oriented specification formalisms, design of reactive systems, design methods, rule based systems and, morerecently, the epistemological foundations of Software Engineering. He has numerous publications in these areas, including joint editing of the Handbook on Logic in Computer Science and The Specification of Computer Programs, written jointly with Prof WM Turski.

Mathematical Foundations of Software Engineering: a roadmap

TSE Maibaum
Department of Computer Science
King's College London
Strand
London WC2R2LS
UK
tom@maibaum.org

ABSTRACT
Although we do not profess to be capable of defining a 'roadmap' for the mathematical foundations of SE over the next ten years, we can discern some important steps that would be extremely useful for the systematisation of design knowledge. The focal point is the concept of *normal design*, introduced by [13] to describe the systematic approach of mature engineering disciplines to standard design problems. It is worthy of note that the mathematics (and science) used by engineers is **not** the same as those used by mathematicians (and scientists). Using his categorisation of engineering knowledge and a framework based on the epistemology of science developed by the logical empiricists, we were able to outline some useful theoretical and methodological issues, which would contribute to developing a mature SE praxis.

Keywords
Foundations, engineering, normal design, radical design, epistemology

1 INTRODUCTION

Futurology is a difficult endeavour.... This is particularly the case if one is asked to produce a 'roadmap', an itinerary, a set of instructions for getting from here to there. The easier part (and it's not so easy!) is the 'here'; much more difficult is the 'there', even if it is only ten years away!

So, let me begin by stating that this will be less of a roadmap than an agenda, which will, I hope, have some relevance beyond my personal activities. But, first, I want to take us on a little detour to place my comments in context. It is an old topic for me, one that has occupied me for many years: just what is it that we are doing when we are engaging in the activity called software engineering (SE)?

SE is the activity of systematically constructing software based systems which are fit for purpose. The word 'systematic', as used here, relates SE to engineering and science, generally. Therefore, in order to make any attempt at explaining what SE theory should be about, we need to understand in what way SE is related to engineering and science and in what way it is different. Moreover, pursuing the distinction between science and engineering will enable us to distinguish agendas for Computer Science (CS) from that for SE.

Constructing software based systems from specifications, be they requirements specifications or design specifications, involves the following activities, amongst others:

- the manipulation of specifications;
- their validation;
- the construction of the initial architectural description of the system from the corresponding requirements specification;
- the verification of the latter against the former;
- the identification of components, be it because they exist in 'real life', or as an artifact to deal with complexity;
- the composition of a system from such parts;
- the interactive refinement of specifications – interactive with the software engineer and, via validation, with 'real life';
- the reification of specifications to transform them from descriptions of the components of the original system, in its own domain, into descriptions of software components that 'simulate' the original ones;
- the verification of refinement relations;
- the construction of programs from detailed specifications;
- the validation of components and systems of software (testing);

- the correction of specifications, designs and software, taking into account the negative results of tests or detection of errors during use (maintenance);
- various management and quality related issues (including version control, change management, project management, etc);
- process improvement.

We can organise (some of) these activities into a number of categories:

- elucidation, the activity of finding a requirements specification;
- illumination, the activity of finding a constructive design specification satisfying the requirements;
- (normal) design, the activity of applying standardised engineering methods to guide the refinement of design specifications to detailed designs;
- systematic observation, the activity of validating through experiments, by either explaining the behaviours or predicting them; and
- calculation using the underlying formalism, the activity of formal derivation of validation and verification tests, of refinements, etc.

(Of these, *design* is what distinguishes engineering from science.)

That SE is an engineering discipline is a simple consequence of the fact that ([10, p51]):

"Engineering refers to the practice of organising the design and construction of any artifice which transforms the physical world around us to meet some recognised need."

Engineering is different from science and mathematics. The differing objectives and methodologies of the three disciplines induce important differences in practice. Moreover, they induce significant differences in the education of practitioners. Conventional engineers are taught the principles of the 'devices'[1] they use in designing artefacts, as well as the systematic, normal design principles (as they will be called below) to be used in building instances of such devices. (In fact, modern SE curricula do not take cognisance of these kinds of foundations and design methods; nor do they provide the software engineer with the formal tools[2] for explanation, analysis and prediction required by engineers for the design of devices[3]. They focus almost exclusively on what is called below the much less often occurring activity of radical design. See also [9].)

2 WHAT ENGINEERING IS

In a very illuminating book[4] by Walter G. Vincenti called "What Engineers Know and How They Know It" ([13]), he argues the case for engineering being different, in epistemological terms and, consequently as *praxis*, from science or even applied science: "In this view, technology, though it *may apply* science, is not the same as or entirely *applied* science" ([13, p4]). In "The Nature of Engineering", GFC Rogers argues that engineering is indeed different from science. He argues this view based on what he calls "the teleological distinction" concerning the *aims* of science and technology:

"In its effort to explain phenomena, a scientific investigation can wander at will as unforeseen results suggest new paths to follow. Moreover, such investigations never end because they always throw up further questions. The essence of technological investigation is that they are directed towards serving the process of designing and manufacturing or constructing particular things whose purpose has been clearly defined. We may wish to design a bridge that uses less material, build a dam that is safer, improve the efficiency of a power station, travel faster on the railways, and so on. A technological investigation is in this sense more prescribed than a scientific investigation. It is also more limited, in that it may end when it has led to an adequate solution of a technical problem. The investigation may be restarted if there is renewed interest in the product, either because of changing social or economic circumstances or because favourable developments in a neighbouring technology make a new advance possible. On the other hand, it may come to a complete stop because the product has been entirely superseded by something else that will meet humanity's changing needs rather better."

He makes a further claim: "Because of its limited purpose, a technological explanation will certainly involve (our emphasis) *a level of approximation that is certainly unacceptable in science*." We shall see later the implications of this approximative approach to SE and to its teaching.

Going back to the distinctions between the aims of science and engineering, we have, again from [10, p55],

"We have seen that in one sense science progresses by virtue of discovering circumstances in which a hitherto acceptable

[1] See below for an explanation of this terminology as an aspect of the organisation of engineering design disciplines.

[2] Here, 'formal' is used in the sense of 'based on mathematical or scientific principles' and not simply restricted to its meaning as part of the phrase 'formal methods'.

[3] Furthermore, there are serious implications of this for SE research.

[4] I would like to thank William Newman of Rank Xerox Laboratories, Cambridge, for bringing this thoroughly enjoyable work to my attention.

hypothesis is falsified, and that scientists actively pursue this situation. Because of the catastrophic consequences of engineering failures - whether it be human catastrophe for the customer or economic catastrophe for the firm - engineers and technologists must try to avoid falsification of their theories. Their aim is to undertake sufficient research on a laboratory scale to extend the theories so that they cover the foreseeable changes in the variables called for by a new conception. The scientist seeks revolutionary change - for which he may receive a Nobel Prize. The engineer too seeks revolutionary conceptions by which he can make his name, but he knows his ideas will not be taken up unless they can be realised using a level of technology not far removed from the existing level."

So science *is* different from engineering. Proceeding on this basis, we can ask ourselves what the *praxis* of engineering is (and ignore, at least for the moment, the specifics of scientific *praxis*). Vincenti defines engineering activities in terms of design, production and operation of artefacts. Of these, design and operation are highly pertinent to SE, while it is often argued that production plays a very small role, if any. In the context of discussing the focus of engineers' activities, he then talks about *normal design* as comprising "the improvement of the accepted tradition or its application under 'new or more stringent conditions'". He goes on to say: "The engineer engaged in such design knows at the outset how the device in question works, what are its customary features, and that, if properly designed along such lines, it has good likelihood of accomplishing the desired task." (See [13, p7].)

In [7], Michael Jackson discusses this concept of 'normal design', although he does not use this phrase himself:

1 "... In this context, design innovation is exceptional. Only once in a thousand car designs does the designer depart from the accepted structures by an innovation like front-wheel drive or a transversely positioned engine. True, when a radical innovation proves successful it becomes a standard design choice for later engineers. But these design choices are then made at a higher level than that of the working engineer: the product characteristics they imply soon become well understood, and their selection becomes as much a matter of marketing as of design technology. Unsuccessful innovations - like the rotary internal combustion engine - never become established as possible design choices."

2 "An engineering handbook is not a compendium of fundamental principles; but it does contain a corpus of rules and procedures by which it has been found that those principles can be most easily and effectively applied to the particular design tasks established in the field. The outline design is already given, determined by the established needs and products."

3 "The methods of value are micro-methods, closely tailored to the tasks of developing particular well-understood parts of particular well-understood products."

Another important aspect of engineering design is the organising principle of hierarchical design: "Design, apart from being normal or radical, is also multilevel and hierarchical. Interesting levels of design exist, depending on the nature of the immediate design task, the identity of some component of the device, or the engineering discipline required."[5] An implied, but not explicitly stated view of engineering design is that engineers normally design *devices* as opposed to *systems*. A device, in this sense, is an entity whose design principles are well defined, well structured and subject to *normal* design principles. A system, the subject of *radical* design, in this sense, is an entity, which lacks some important characteristics making normal design possible. "Systems are assemblies of devices brought together for a collective purpose"([13, p201]). Examples of the former given by Vincenti are aeroplanes, electric generators, turret lathes; examples of the latter are airlines, electric-power systems and automobile factories. The SE equivalent of devices may include compilers, relational databases, PABXs, etc. SE examples of systems may include air traffic control systems, mobile telephone networks, etc. It would appear that systems become devices when their design attains the status of being *normal*. That is, the level of creativity required in their design becomes one of systematic choice, based on well-defined analysis, in the context of standard definitions and criteria developed and agreed by engineers.

Let us now consider the particular characteristics of SE as a discipline. We want to address the question: Is the knowledge used by software engineers different in character from that used by engineers from the conventional disciplines? The latter are underpinned not just by mathematics, but also by some physical science(s) - providing models of the world in terms of which artefacts must be understood. (The discussion above illustrates this symbiosis.) We might then ask ourselves about the nature of the mathematics and science underlying SE. It is not

[5] It is quite clear from the engineering literature that engineering normally involves the use of multiple technologies. The observation that SE requires knowledge of other domains an that its teaching should be application oriented is not as perspicacious as its proponents would have us believe. This is part of the essence of engineering, whatever the 'discipline'.

surprising, perhaps, that a large part of the mathematics underlying SE is formal logic.[6].

Logic is the mathematics of concepts and abstractions. SE may be distinguished from other engineering disciplines because the artefacts constructed by the latter are physical, whereas those constructed by the former are conceptual. (After all, a program is not the printout, nor is it the physical state of some part of the memory of some computer. These are all representations of the program, in the same way that '2', '10', 'II', ... are all representations of a particular integer - a purely conceptual entity.) Hence, there should be no surprise in the close connection between logic and SE. There are some interesting and significant differences between the two kinds of mathematics and engineering mentioned above. One of these is that the 'real world' acts as a (physical) constraint on the construction of (physical) artefacts in a way which is more or less absent in the science and engineering of concepts and abstractions. There seems to be a qualitative difference in the dimensions of the design space for SE as a result.

The engineering of concepts and abstractions predates the existence of computers. Philosophical logicians, particularly, have engaged in it for some time. An obvious example, of which I have some knowledge, is deontic logic, originally developed to study legal reasoning, particularly as pertaining to concepts of duty, obligation and prohibition. *What distinguishes the theoretical computer science and SE dependence on logic is the day to day invention of theories by engineers and the problems of size and structure introduced by the nature of the artefacts with which we are dealing in SE.*

Now, the relationship between the mathematics of theoretical computer science and that of (formal methods and) SE should be analogous to the difference between conventional mathematics and its application and use in engineering. As an example, program construction from a specification has a well-understood underlying mathematics developed over the last 25 years. (We are restricting our attention to sequential programs. Concurrency and parallelism are much less mature topics.) We might expect to find a CAD tool for program construction analogous to the 'poles and canvas' model used in electronics for the design of filters. Instead, what we find is just a relaxation on the exhaustiveness requirement, i.e., we can leave out mathematical steps (proofs of lemmas) on the assumption that they can be filled in if necessary, the so-called rigorous approach. Where is the abstract model (analogous to the 'poles and canvas' one) which encapsulates the mathematics and constrains manipulation in a (mathematically/scientifically) sensible manner?

The underlying model used by the rigorous programmer (i.e., the software engineer) is exactly the same as that used by the theoretician. The only real difference is that the assumption that mathematical steps may be omitted based on the educated guess of the engineer that the omissions could actually be corrected if this proved necessary (- no pun intended!). It may be asserted that this requires the engineer to be a better mathematician than the theoretician! This is clearly very different from the experience of the conventional engineer. We may thus begin to question the thrust of much SE research over the last decade.

In conclusion, SE *is* an engineering discipline, but it has a special character of its own which distinguishes it from other engineering disciplines. Hence, the subject matter and curricula of SE is not simply subsumed as a speciality within some existing discipline. To put in context the remarks in the next section on SE research, we should first examine what Vincenti calls 'categories of [engineering] knowledge'. These comprise:

- Fundamental design concepts
- Criteria and specifications
- Theoretical tools
- Quantitative data
- Practical considerations
- Design instrumentalities

Fundamental design concepts include the *operational principle* of their device. According to Michael Polanyi (as per [13]), this means knowing for a device "how its characteristic parts....fulfil their special functions in combining to an overall operation which achieves the purpose". A second principle taken for granted is the *normal configuration* for the device, i.e., the commonly accepted arrangement of the constituent parts of the device. These two principle (and possibly others) provide a framework within which normal design takes place.

Criteria and specifications allow the engineer using a device with a given operational principle and normal configuration to "translate general, qualitative goals couched in concrete technical terms" ([13, p211]). That the development of such criteria may be problematic is clear.[7] However, the

[6] This is not to say that elements of conventional engineering disciplines are not also useful for software engineers. See [Parnas] for a discussion of this.

[7] I have long been puzzled by the seemingly intractable problems of systematising knowledge about the "ilities" of SE: usability, portability, dependability, etc. So much so, that I began to accept the possibility that these qualitative concepts were inherently uncharacterisable. Vincenti's book discusses at length such an "ility" relates to aeroplane design and called *flying qualities*. These

development and acceptance of such criteria is an inherent par of the development of engineering disciplines.

Engineers require *theoretical tools* to underpin their work. These include intellectual tools for thinking about design, as well as mathematical methods and theories for making design calculations. Both conceptual tools and mathematical tools may be devised specifically for use by the engineer and be of no particular use or value to a scientist/mathematician. Again, [7] states: "The methods of value are micro-methods, closely tailored to the tasks of developing particular well understood parts of particular well understood products. ...[T]he most useful context for the precision and reliability that formality can offer is in sharply focused micro-methods, supporting specialised small-scale tasks of analysis and detailed design." [8]

Engineers also use *quantitative data,* often the result of empirical observations, as well as tabulations of values of functions used in mathematical models. A good example in SE of this thoroughness in providing data useful for design is the work of Knuth on sorting and searching. There are also *practical considerations* in engineering. These are not usually subject to systematisation in the sense of the categories above, but reflect pragmatic concerns. For example, a designer will want to make use of various trade-offs in the design of devices which are the result of general knowledge about the device, its use, its context, etc.

Design instrumentalities include "the procedures, ways of thinking, and judgmental skills by which it [engineering] is done" ([13, p220]). This is clearly what the Capability Maturity model has in mind when it refers to well defined and repeatable processes in SE.

3 RESEARCH FOCUS AND DIRECTIONS

I assume that our intention is to make SE a proper engineering discipline. So, when we say "mathematical foundations" of SE, we are speaking about the mathematics required to support the categories of SE knowledge referred to above. Hence, research objectives (and curricula) should be driven by this intent. The status of SE has sometimes been described as being no more than craftsmanship[9]. In relation to the difference between craftsmanship and engineering, the problem is that, without a formal representation of the objects, rules, and guidelines involved in an engineering process, this chain is nothing but a set of prescriptive and descriptive rules of thumb. Therefore, a method built upon this basis does not possess the "exactness" and "inferential (analytical) power" that are components of every mature engineering process.

Achieving this "exactness" and "inferential (analytical) power" should be our guiding principle in deciding what SE theory should have as its aims. But we may still wish to make more precise what exactly the subject matter of this "exactness" and "inferential (analytical) power" is. The italicised sentence above (... *the day to day invention of theories by engineers and the problems of size and structure introduced by the nature of the artefacts* ...) provides a key to the nature of the endeavour and a possible basis for organising theoretical investigations.

There does exist a body of knowledge about the invention of theories, namely in epistemology as applied to the understanding of how scientists go about their business. It is common to refer to the introduction or experimental verification of scientific theories, particularly (but not exclusively) in the natural (physical) sciences. (The famous dictum that a positive result of a test does not confirm the correctness of a program, while a negative one refutes it, is obviously a late restatement of the hypothetico-deductive method of science – stated by Newton and refined by Carnap, Hempel and others – which is at the heart of the method of natural science and, we maintain, engineering ([11, 12]).) Simply stated, scientists invent theories to explain phenomena and use the mathematical language of the theory in order to:

- explain known data; and
- design experiments with the purpose of confirming or refuting the theory;
- make predictions about phenomena which are derivable consequences of the theory.

The fact that this theory invention is not a very frequent activity, though it is continual, is what will distinguish science from SE. Engineers use theories in the same ways, though the methods and tools are somewhat differently focused. Design, explanation and prediction are important tools for the engineer. For example, designs are theories, at least in the sense that they (partially) explain the function of some artefact. But, whereas the scientist focuses on existing/observed phenomena, the engineer uses a specification of the intended artefact as a theory to predict the

"comprise those qualities characteristics of an aircraft that govern the ease and precision with which a pilot is able to perform the task of controlling the vehicle". It was one of the triumphs of aeronautical engineering between the World Wars that they "translate[d] an amorphous, qualitative design problem into a quantitatively specifiable problem susceptible of realistically attainable solutions." See [Vincenti, Chap 3].

[8] Jackson is discussing formal methods and arguing, persuasively, that universalist views of formal methods are misguided (*my* word) and that their obvious role is in micro-methods.

[9] Craft: a skilled trade.

characteristics of a 'new' reality, i.e., one in which the world has been changed by the existence of the posited artefact. See the figure at the end of the paper, illustrating the main objects, relationships and processes involved in the design of software based systems. (The figure is borrowed from [3] and is based on the ideas discussed in [4].) The upper horizontal plane corresponds to the theoretical level of discourse, while the lower horizontal level corresponds to the (old and new) realities, which represent the original problem and the realisation of its solution, both in the context of the extant realities. (Note that the old dictum of SE that the existence of the new system may alter the environment of the system itself, is reflected by a possibly changed context in the figure.

Accepting that scientists build theories and that much of SE activity is related to such invention of theories, can we learn anything from epistemology about the nature of such invention? If we can, then we might be able to apply this 'engineering' part of science to the everyday activities of software engineers. In fact, there is much to learn from the so-called logical positivists about these issues and much remains to be done to adapt them to deal with the issues raised by SE. See [4] for a more detailed explanation.

The clear separation of and determination of the relationship between the language and logic of observation, on the one hand, and the theoretical language and logic of scientific theories, on the other, underpins the systematisation of engineering praxis. A simple(?) and immediate consequence of this separation is the justification for having (formal) specifications. The normal explanation of specifications is that they serve a very pragmatic purpose in aiding an engineer to bridge the large gap between initial and informal requirements and eventual executing system: this gap is too large to achieve in one go, so it must be modularised. But, there is actually a fully formal explanation of why they are necessary and not just helpful. For example, take the problem of determining if a program, executing on some machine, will halt on some input or not. Firstly, there is no experiment we can design which will determine this fact: we might wait forever to see if halting does occur. (This is simply the requirement on experiments to have a definite outcome.) But we can use the theory of the programming language to reason in the logic of the theory about the termination, a theoretical property, of the program on that input. Then, this reasoning gives us a time interval that we have to wait for this termination. We can now conduct an experiment to see if halting does occur at the predicted time or not. The 'trick' we have used is to replace an observationally undecidable property, halting, with a theoretical one, termination, and then relate the two.

Hence, this theoretical plane of discourse is a necessary one and not just an aid to thinking. Moreover, to the extent that SE is largely about design, theory construction and manipulation are important activities to characterise and turn into engineering tools. To the extent that explanation, experiment, analysis and prediction are the are important for operational aspects of SE, the knowledge of how theory relates to observation becomes relevant and crucial. The theory of SE, and so its mathematical foundation, is fundamentally related to these points and the road ahead must provide signposts to relevant work.

As to the formalisations of engineering versus those for science/mathematics, we start by quoting again from [7]. "...formalisation in traditional engineering is applied locally to well-understood characteristics. The context for applying the formalism is almost fixed, and the calculations to be made are almost standardised."

In some areas of SE there are terms, usually of 15 syllables and ending in 'ility', belonging to the class of informal, semi-scientific concepts that bedevil the subject (and clients who want enforceable contracts). On the other hand, the definition of terms known in engineering as 'ilities' are crucial to it. Let us take, for instance, the concept of *flyability* in Aeronautical Engineering ([13]). Flyability is a concept relating the ability of the pilot to control the aeroplane with some physical properties of the aeroplane. It is an important design tool for aeronautical engineers and has been so since the 30s. We do not need to spend too much time in realising that if Aeronautical Engineering deserves the status of being a branch of Engineering, with a body of knowledge of its own, flyability is a crucial concept. These 'ilities' are at the heart of engineering praxis. They encapsulate what we have referred to as *normal design* above, thus capturing the essence or focus of an engineer's everyday activities.

The problem with 'ilities' is that they are dispositions, i.e., theoretical properties that are not immediately observable, but which have observational consequences. (The advice of some epistemologists to non-philosophers for guessing if a property is a disposition or a plain (observational) one is exactly 'see if the English word designating it ends with 'ility'". It has been observed that trying to introduce dispositions naïvely in the language of science is very risky, because the naïve way of defining dispositions is by using operational definitions and operational definitions are inherently flawed. The natural reaction against this flaw, based on non-strict readings of the material conditional, are dangerous because one needs to 'bear in mind' the subjunctive reading of the material conditional. And, even if one is sufficiently lucky to be able to 'bear in mind' the appropriate subjunctive properties (i.e., properties of counterfactuals), one will be disappointed, because, as we know, counterfactuals are not truth functional.)

Now, one of the observable goals of the SE endeavour is the construction of software development environments based on solid conceptual architectures. It is too dangerous to deal with the construction of a huge software artefact relying on an environment which forces us, for instance, to 'bear in mind' dozens of 'non strict readings'. Consider that software development environments are the tools of SE, as wind tunnels are for Aeronautical Engineering. Moreover, many of the 'ilities' of SE are ill defined or not defined at all. As an example, consider more or less any of the definitions of *usability*, including ones to be found as part of international standards, such as ISO9001. Even worse, perhaps, are uses of 'ilities' in such security standards as the common criteria for security of smart cards.

4 THE ROAD AHEAD

So, what should we be doing in order to reach the next arbitrary point in the space-time continuum from the present one? (I am writing this part on the first day of the new millennium!) I will simply list, without any extensive explanation, the theoretical ideas and related tools (intellectual or the corresponding artefacts) which I would find helpful in doing SE. The topics are organised by means of the scheme described by Vincenti for organising engineering knowledge.

Fundamental design concepts

We need many and varied in-the-small theories to support design. Examples include:

- representing behaviour (including concurrency and duration of activities) and being able to analyse it. There may be many different kinds of behaviour and there is no obvious necessity to have a universal representation – quite the opposite! Engineering tools are the most useful when they are specific, so different classes of problems may demand differing languages to represent them. Moreover, the theoretical languages for representing behaviour are certainly not going to be the effective tools for engineers – the engineering notations largely remain to be discovered.
- representing 'ility' properties and devising corresponding engineering theories enabling the use of the 'ility' in design. This is a difficult process and I do not expect much progress. After all, it took 25 years for 'flyability' to attain this status.

We need to organise engineering design knowledge and, hence, we require principles for its organisation. Of obvious importance is the concept of 'normal configuration' put forward by Vincenti. In SE, the main ideas related to 'normal configuration' seem to be captured in the concepts of problem frames, patterns and software architectures. Systematising these ideas and putting them on a proper theoretical and pragmatic basis is extremely critical for the future of design. In this regard, work on standardised architectures/problem frames and product line architectures would appear to be important. However, not a lot of progress has been achieved in systematising these ideas to the level of rigour required for engineering.

Criteria and specifications

It is my firm belief that a very large proportion (perhaps the large majority) of software based systems being built today could be built using normal design principles. The fact that they are not being built in this manner is a reflection of the immaturity of the SE discipline and, also, of the natural resistance to change amongst individuals and supplier organisations who find craftsmanship and its mystique comforting. The dearth of many companies able to achieve higher ratings on the CMM scale is just one obvious piece of evidence for this.

To turn this around and apply normal design principles to the construction of most systems, we need to systematise design knowledge for these systems. This in turn involves two distinct steps:

- systematising domain knowledge for the area of application (e.g., financial systems, telecommunications, automobile control, etc.). This is hard, long and often tedious work. Nevertheless, it is essential for normal design (or, for that matter, for radical design).
- defining the specification and refinement patterns/architectures required to encapsulate design choices. This results in support for the 'cookbook' aspects of normal design. The work on product line architectures, if properly driven towards formal engineering systematisation, will contribute enormously to this.

Out of this kind of work will emerge systematic ways of turning client requirements into normal design solutions, i.e., cookbook methods for defining requirements for typical systems.

There is not much corresponding work on refinement patterns. (Fourth generation languages were an early attempt at doing just this for information systems.) It is clear that such things exist; e.g., a well known telecommunications company uses a fixed (small) number of refinement 'levels' in producing software for small switches from their specifications. The production of scheduling algorithms has been reduced at the Kestrel Institute to the application of cookbook refinement transformations. Obviously, there should be standard ways to implement certain fragments of systems within normal applications. It is not too much to hope that such normal refinement steps are actually reduced to transformations in the formal, technical sense. In fact, it would appear possible today to perform code generation by transformational methods (as opposed to compilation) as part of normal design. The benefits for systematic design are clear: correctness, traceability of code to detailed design, semantics based code optimisation, specialisation to specific hardware, etc.

Theoretical tools

As noted above, we require intellectual tools for thinking about design, as well as mathematical methods and theories for making design calculations. Both conceptual tools and mathematical tools may be devised specifically for use by the engineer and be of no particular use or value to a scientist/mathematician. This is a consequence of the fact that the theories used by scientists and engineers are not generally the same (though, of course, there should be a connection).

Amongst the many such tools we need are included:

- better understanding of modularity principles. The only effective method for dealing with the complexity of software based systems is decomposition. Modularity is a property of systems, which reflects the extent to which it is decomposable into parts, from the properties of which we are able to predict the properties of the whole. Languages that do not have sufficiently strong modularity properties are doomed to failure, in so far as predictable design is concerned. Much of the extant work on modularity is concerned with what might be called presentation modularity, i.e., presenting the text of a specification or program in a step by step fashion. It has nothing to do with modularity in the sense of architecture: how the behaviour of the functioning system is built from the behaviours of its parts. Furthermore, modularity in specification and design languages is unlikely to be the same as that in programming languages. Simply, this is because specifications and programs are ontologically very different. We need to understand the relationship between these differing notions of modularity, so as to systematise what is almost always a 'magic' step in SE, i.e., the step from detailed design specifications to executable code in some programming language. (The best work in this area is, of course, that of myself and my colleagues, such as José Fiadeiro... :-))
- better understanding of behavioural principles (concurrency, distribution, timing, ...). Behaviour is basic for design of software based systems. It may be asserted that behaviour plays the same role in software that continuous phenomena play in physical systems. There are many notions of concurrent behaviour and there is perhaps no universal characterisation of them (in the sense that differential equations may be considered a universal language for characterising physical phenomena). We may have to admit the fact that there will be a multitude of concurrent/behavioural/real-time 'horses for courses'. There is much theoretical work to be done in this area, particularly in relation to scalable models. The problems of modularity again rear their heads; feature interaction is an illustration of the phenomenon of emerging properties, i.e., properties which are ascribable to a component when it is part of some specific system, but which the component was not specified to have when it was independently designed. (So, a corresponding program that is correct with respect to the original specification may no longer be correct in relation to the specification component as part of the specific system, as it does not conform to the emergent property!)
- improved and specialised analysis tools (model checking and related tools). Theorem proving and model checking tools are often difficult to use in their full generality. Engineers require specialised tools that will enable them to analyse specific classes of properties of systems using standard techniques. An obvious existing example is type checking, which has efficient implementations for specific languages and whose results are extremely useful for design. (The analogy with specific methods to solve varying classes of differential equations, numerically, symbolically, or approximately, is a good one.) I believe that it will be the specialisation of these powerful tools into specific analysis algorithms that will enable engineers to make everyday use of them. General use will be pursued as part of radical design by a relatively few, highly trained scientists/engineers.
- many more abstraction (interpretation) tools to address feasibility/tractability of analysis. Type checking works efficiently because there is a useful abstraction of programs which is tractable for analysis and that provides useful design information. There should be many properties of specifications, designs and programs that are analysable, in analogy with type checking, via abstraction and the appropriate (algorithmic) analysis. (We may be able to relate this use of abstraction to the "levels of approximation" of [10], referred to above in Section 2.)
- more and specialised decision procedures for interesting properties (using abstractions to approximate same).
- better systematisation of testing. It is often recognised that testing in engineering is related to experiment in science. If this is the case, then ideas from science about experimental design may be applicable. This may not be the end of ideas about testing, but it is a good starting point. Recent work by Haeberer and colleagues on using experimental design principles to extend and systematise specification based testing is bearing fruit. Related to testing, we have simulation and animation, both

extremely important tools for analysis of the behaviour of systems. The work of Magee and his colleagues on generating animations from behavioural models is an outstanding pointer to what may be possible.

- developing the theories of 'ilities' and related experimental science. SE requires a systematisation of our 'ilities' into engineering design principles. This area is of great importance for the future effectiveness of design and the creation of normal design principles, which, at the same time, has the least promising position from which to start. If funding authorities are looking for investment opportunities, sound work in this area is likely to bring the greatest rewards in the future.

Quantitative data

My knowledge of this area is weak. What I have in mind may be illustrated in analogy with Knuth's attempt to organise knowledge about algorithms. Similar compilations of appropriate data would be useful for understanding the usefulness of architectures and patterns. Ditto in relation to testing schemes.

Of course, another aspect of data is related to engineering processes. SE is not that good on adopting engineering principles of measurement in this data gathering.

Practical considerations

Theory is simply a guide here to aid organisation of practical knowledge, which may itself not be formalisable in a purely mathematical sense. Here, development of (meaningful) standards and organisational/management principles (such as CMM) are important contributory factors. What *is* required is awareness of and respect for theoretical issues and advances. Similarly, and perhaps more fundamentally, respect for engineering principles and their proper application is required. Again, CMM and quality standards are related to this.

Design instrumentalities

Ditto above re practical considerations.

5 CONCLUSIONS

Although we do not profess to be capable of defining a 'roadmap' for the foundations of SE over the next ten years, we can discern some important steps that would be extremely useful for the systematisation of design knowledge. The focal point is the concept of *normal design*, introduced by [13] to describe the systematic approach of mature engineering disciplines to standard design problems. Using his categorisation of engineering knowledge and a framework based on the epistemology of science developed by the logical empiricists, we were able to outline some useful theoretical and methodological issues, which would contribute to developing a mature SE praxis.

Of fundamental importance are the following distinctions:

- between mathematics, science and engineering;
- between engineering in the classical disciplines and SE;
- between Computer Science (a science, or at least an applied mathematics).

Of great importance to the future development of SE as a discipline is the invention of appropriate iconic notations, which are well founded and which have corresponding pragmatically useful analyses. (I agree wholeheartedly with much of what is said in [7] in relation to analysis.)

6 REFERENCES

1. Carnap, R. (1950) Empiricism, Semantics, and Ontology, *Revue Internationale de Philosophie*, 11, 208-228.

2. Carnap, R. (1956) The Methodological Character of Theoretical Concepts, *Minnesota Studies in the Philosophy of Science, Vol. II.* University of Minnesota Press, 33-76.

3. Cengarle, M.V. and Haeberer, A.M. (2000) Towards an Epistemology-Based Methodology for Validation and Verification Testing, Technical Report, Institut für Informatik, Ludwig-Maximilians-Universität München

4. Haeberer, A.M. and Maibaum, T.S.E. (1998) The very Idea of Software Development Environments: a Conceptual architecture for the ARTS Environment Paradigm, *Proc. of ASE'98*, Redmiles, D. and Nuseibeh, B., eds, IEEE Computer Science Press.

5. Hempel, C. G. (1965) *Aspects of Scientific Explanation and Other Essays in the Philosophy of Science,* The Free Press, New York

6. Hesse, M. (1966) *Models and Analogies in Science,* University of Notre Dame Press.

7. Jackson, M. (1997) Private Communication.

8 Jackson, D. and Rinard, M. Reasoning & Analysis: a roadmap. In this volume.

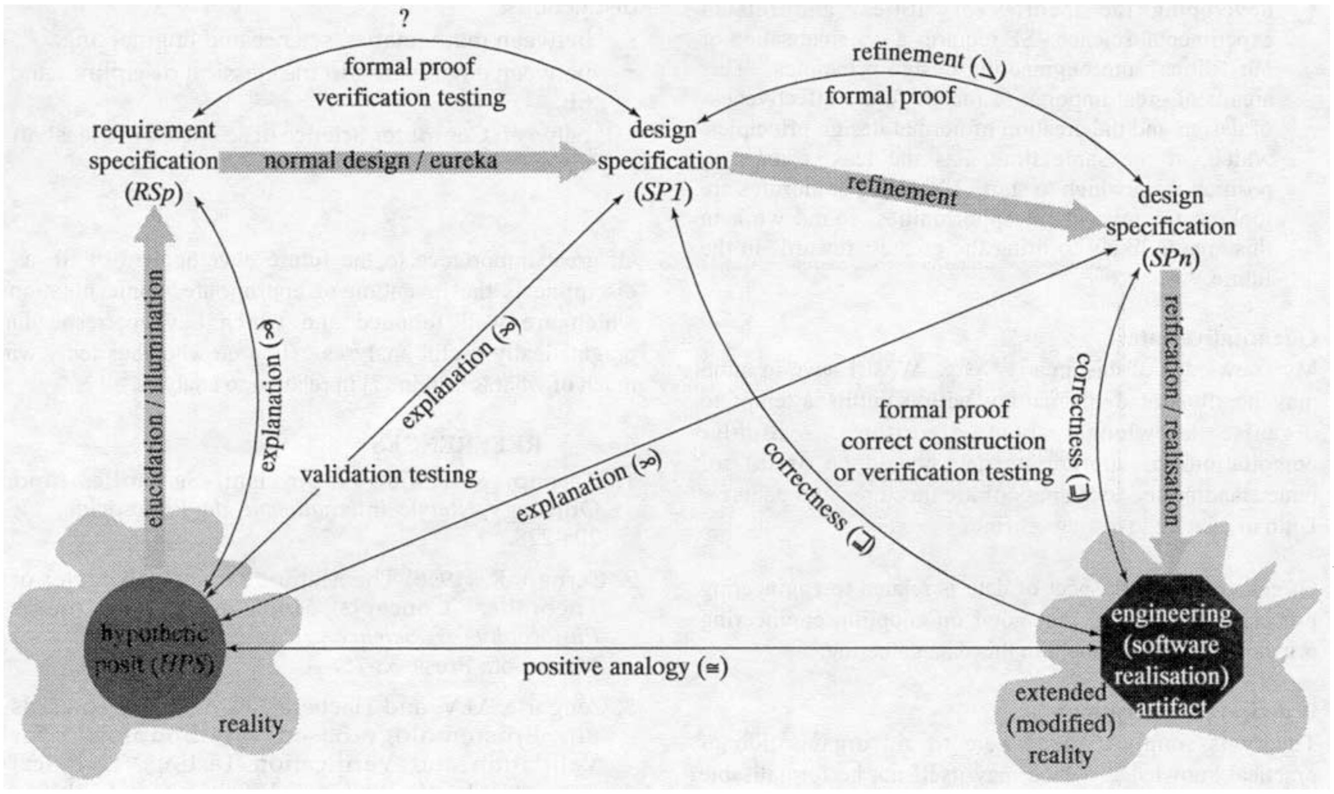
9. DL Parnas, Education for Computing Professionals, *IEEE Computer*, Vol 23, No 1, 17-23, 1990.

10. Rogers, G.F.C. (1983) *The Nature of Engineering,* The Macmillan Press Ltd.

11. Stegmüller, A. (1979) *Probleme und Resultate der Wissenschafstheorie und Analytischen Philosophie Band II: Theorie und Erfahrung,* Springer-Verlag.

12. Suppe, F. (1979) *The Structure of Scientific Theories*, University of Illinois Press.

13. Vincenti, W.G. (1990) *What Engineers Know and How They Know It*, The Johns Hopkins University Press.


?
formal proof
verification testing
refinement (⊵)
formal proof
requirement specification (*RSp*)
normal design / eureka
design specification (*SP1*)
refinement
design specification (*SPn*)
elucidation / illumination
explanation (≫)
explanation (≫)
validation testing
explanation (≫)
formal proof
correct construction
or verification testing
correctness (⊑)
correctness (⊑)
reification / realisation
hypothetic posit (*HPS*)
positive analogy (≅)
reality
engineering (software realisation) artifact
extended (modified) reality