CURRENT INDUSTRIAL SOFTWARE ENGINEERING CONCERNS

B.C. Warboys

Rapporteurs: Mr. F.L. Hughes
Mrs. A. Petrie

Extended Abstract:

1. Theme of lecture

Changes in concerns lead to changes in the required support systems. These support systems then require differing skill bases for their exploitation and this in turn implies that curriculum planning needs to take into account the new scenario.

2. Changes in Concerns

A movement from a single discipline profession very much targetted towards the production of discrete components. The new scenario reflecting the increasingly endemic nature of IT in modern society is multi-discipline. Some members are still concerned with the production of discrete components, more are engaged in the glueing together of bought-in components, and in the future most will be concerned with "Systems development in the large". That is the production of total ecosystems with the IT components providing the enabling technology.

2.1. Market Segmentation

This multi-discipline scenario is leading to a movement towards two styles of development. A so called "professional" approach attempting to improve the predictability of software production by an emphasis on formalism. Such approach is inevitably based around the development of sound language theories.

The other reflecting the "natural" development style based upon the views that human design activity is best attempted in a dialogue style. That the iterative nature of design is best achieved in an interactive style with designs being tried out, examined, discarded and then reiterated. This tends to be based around the development of "natural thought" languages with a view that the onus should be on the computer system to understand and sympathise with the habitual human reasoning process.

2.2. Software as an enabling technology

Whichever theology of development is accepted, and I believe that both will continue to flourish, the key movement in industrial software engineering is towards its use as an enabling technology in broader system development. That is it is a partial process which needs embedding in a broader ecosystem design activity.

Increasingly the IT components of total systems are formed from a set of sub-components each with a "small" set of distinct characteristics. The creative software activity is concerned with the use of software as the embedding medium for bought-in components and the customisation of this conglomerate system to satisfy end user demands.

Further the trend to accelerated cost-reduction in hardware components leads to the need for hardware component manufacturers to increase the intellectual added-value from their outputs to compensate for the lack of profit to be derived from the purely manufacturing content of the products. Thus the technology is producing both the capability and the need for higher functional content in hardware components. The speed of cost-reduction together with the automated nature of the manufacturing process leads to an ever-decreasing hardware component life-time. The higher functional content leads to an increasing reliance on software development paradigms. This, in itself, tends to a greater integration of software and hardware concerns and hence again re-enforces the trend towards functional enhancement of hardware components.

Paradoxically the increasing reliance on specific algorithms to help manage business concerns has the effect of extending software component life-times. It is the software constructed embedding medium which must take the stress of this difference in hardware and software lifetimes. It is this stress which has resulted in an increased emphasis on multi-layered system standards and increasing concern for component reuse technology in operational systems. As we shall see later there are other developmental pressures which also lead to an increased pressure for the technology of component reuse.

2.3. The Industrial Scenario

Industrial software engineering concerns are increasingly influenced by four main factors.

(a) The endemic nature of IT - leading to an emphasis on software as an enabling technology for the development of the IT components which contribute to a total ecosystem. This is reflected in the breakdown of development costs where increasingly costs associated with issues of total system integration are dominant. This leads to an increasing emphasis on the technologies of system specification, human interfacing and component integration.

(b) The nature of the IT business - the added-value chain reflects the move towards component integration. As hardware components gain added functionality so the base component supplier moves further up this added-value chain. The user is gradually succeeding in pressurising the industry into standard building works and the application generator technology to support standard component usage. These generators are less complex to utilise than the component construction languages and hence increasingly the end-user is able to participate more fully in the construction of the IT components to suit his total system needs. Thus the end-user moves down the added-value chain and increasingly the characteristics of these total systems have a greater input on supporting component design. This integration leads to the development of more "natural" component relationships and has led to a reawakening of interest in object oriented systems - a subject which to an extent has laid dormant since the short era of capability machines at the end of the sixties.

(c) Further the emphasis on business enabling has led to a move towards operational and decision support and away from scientific processing. This brings with it the needs of management as a user and hence in turn adds to the pressure for a move towards more "natural" human interfaces. As the "naturalness" of interfaces increases so there is added difficulty in providing the sound theory which has characterised the development of computer languages hitherto. Paradoxically as IT components have become more integrated with business and operational systems there is a need to increase their reliability. Thus the stresses of trying to cope with the, as yet, divergent demands of "natural" interfaces/rule-based object management systems and rigour in software development have led many to conclude that fault-tolerance is the only viable way forward. Whatever the ultimate answer, the trend for some time will be towards technologies which can cope with this variety of development approaches.

3. Changes in Support Systems

The changes in concerns highlighted above lead to the desire for new properties in software engineering support systems. These support systems must recognise the importance and characteristics of:-

(a) Systems development in the large - the fact that the IT component is only a part of the total system.

(b) The variety of development approaches - as the breadth of system design increases it is unlikely that a single generic development paradigm will emerge.

(c) The importance of reuse as a technology - not just at the operational component level but particularly at a process level. As system design becomes a larger concern so the time spent in process design increases as a proportion of total development cost.

3.1. What then is an IPSE?

The term IPSE, in the UK attributable to the Alvey programme, is an acronym for Integrated Project Support Environment. It is clear that given the wider definition of System Design outlined above that the IPSE must concern itself with process support issues much broader than those concerned with the mere development of software components. The lecture "IPSE 2.5 - one IPSE that is necessary" will address these issues (see next Section).

4. Changes in Skill Base

The changes in the skill base derive from the background outlined above and reflect the two enduring issues which have consistently engaged the Software Engineering community.

4.1. "The Software Crisis"

Basically and consistently over the last decade the demand for software production has outstripped the capability of the industry. The increase in the number of software developers and the improvements

in software production prcductivity have failed to keep pace with the need. This is partially a function of the huge increase in the use of IT and partially that the total cost of production is spread so widely over a range of development concerns. Attempts to improve productivity have tended to deal with single aspects of the development process and rather like stamping on rugged balloons have only resulted in moving the hot air from one part of the surface to another. How much this is due to the sixties derived notion of the development process is still a matter of much great debate. However, it is clear that it is well understood that the cost of correcting errors is proportional to the phase of the development cycle in which they are found and further that most software systems do not meet their contractors original requirements. Yet there is little progress in responding to these difficulties on an Industrial scale. This is at face value, an extraordinary state of affairs. Whatever the shortcomings of our industries, they usually solve problems once they have been identified. My contention is that the tools and methods which purport to assist these problems have taken too narrow a view of the total process and hence their solutions have had partial effects. This in turn led to more complication not less and hence the solutions have been rejected.

4.2. On the Industrialisation Problem.

The industrialisation of software engineering techniques has three major implications. First that of scale - horrendous development processes have derived from the linear scaling of acceptable techniques at small project levels. Secondly the problems of technology transfer have been underestimated. Particularly since the technology has a strong process content, it is by definition attempting to change the way people design. Since the human is a "natural" design animal this has a profound influence on the way people think. By definition the more successful software managers are the more experienced ones and hence one is trying to influence a method of thought which has evolved through relatively successful training by experience. Further the problem solving skills are developed to apply software technology in the small to the larger system concerns and little theoretical work is usually performed on this aspect of technology usage. The situation is further complicated by the variety of componentry which needs management integration and indeed many of the components are poorly identified because of the legacy of history and varying performance and integration constraints. My contention is that a complete unified solution to all these difficulties is not practicable and that instead emphasis needs to be placed on enabling infrastructures which allow differing instantiations to deal with differing problem sets.

In particular recognition must be given to both the volatile nature of the technology and the rapidly changing shape of the "added-value" chain. Considerations which lead to the need to allow infrastructure change to occur as new techniques arise and need to be embedded in rather than totally replace existing practices.

The life cycle support in the large which is implied needs to ensure that a gearing of individual effectiveness is obtained for a variety of roles within the development group. The needs of Personnel Managers, Salesmen, Researchers, Management must be addressed to ensure that any gains in productivity due to programming support are not eroded by the extra complications of fitting the new programming practices into existing and integrated business practices.

4.3. Possible solution strategies.

Clearly the problem in the large is comparable to that of replacing a component in an operational system. We wish to reuse much of what exists and we wish to integrate some new component into this existing structure. The same desires for process change as we have for product change. The sensible approach is therefore to apply the same principles of component design which lead to software component reuse to the problem of components in the large. Clearly technology is needed to create both an infrastructure for component reuse and for sound component production. Clearly component is used here in the widest sense of the word to cover both operational components at specification, design and implementation levels and the process for their development.

Clearly such a flexible approach allowing for the instantiation of an endless-variety of local/personal practises needs some coherent frame work which I like to call Departmental House Style. Issues which need such stylistic control are those concerned with Representation, Modularity, the Continuity of the development process and "Testing" I believe that most other issues are amenable to a great variety of local "taste".

4.4. Some trends.

As was stated earlier it is inevitable that as business comes to rely more heavily on specific software so it will become more difficult to change them. Operational networks and corporate databases, even Departmental Mail systems, once installed will only allow the embedding of new components not their total replacement. This phenomenon applies equally to new techniques and to products obsoleted by differing product lifetimes. The stability of much of the infrastructure will lead to a greater capability both to measure the system and the improvements due to new components. Thus the science of quantification will grow.

Discontinuities will abound due to business integration, the endemic nature of IT, new technologies and the integration and convergence of many components. Thus we are likely to see a growth in techniques rather than a convergence towards a single generic solution. This growth, coinciding with a broadening of system concerns will represent a great stress on the definition of the scope of Computer Science. They indeed may lead to the creation of a broad church of Information Technology together with a narrow definition of those concerns which should be studied under the heading of Computer Science. I believe such a division will also apply to Engineering Support Environments with the current software development style being but a small part of the process support.

5. Changes in Curriculum Planning.

These "in the large" concerns coupled with rapid change in "in the small" technology have many implications for the planners of the computer science curriculum. It suggests a greater concentration on a set of basic theories which will apply irrespective of technological change. As the world gets more complex and varied the art of recognising commonality becomes more effective. It is indeed an important attribute of the Component Engineering theme which this paper is suggesting. Thus I should like to see the Basic Themes of:-

- Whole Process Concerns
- Abstraction and Representation
- Cumulative experience with ideas
- Recurrent Notions such as Hypothesis and Test, Problem Solving, Analysis and Synthesis, Abstraction and Realisation and Inductive Reasoning

become the core of the Computer Science Curriculum.

Lastly I should like to remind everybody about the biggest obstacle to Software Engineering progress over the last twenty years. The issue of Technology Transfer - we, as researchers and teachers should not forget that the technology when delivered always has a sociological component and that, as yet, tools remain the only method of technique enforcement. These active attributes of the technology are usually poorly addressed and reflect the fact that our profession contains two self-contained cultures. One the researcher needs research success to justify new research support. The other the developer needs project success to justify new project opportunity. Natural selection and time (reliant as they are on experience derived criteria) will not improve this situation. Perhaps new organisational relationships are needed before in the large component engineering becomes a reality.

DISCUSSION

Q: Are you an employee of ICL working for a University or vice versa?

A: This is not a problem. It is not difficult to resolve, just don't get too obscessed.

Q: Is it difficult to translate ideas into curriculum? Should live systems be used? What can be done to help the students?

A: The science of prototyping should be taught. The students should be allowed to play with operational systems. Both individual and group projects should be taught. However, more technology is required.

Q: Do you work with mechanical engineering?

A: Yes. There is multidiscipline in many universities. However, there should be more thoughtful investment in IT. Currently it is a case of CS versus IT.

Q: Many problems are not directly related to CS, for example, imbedded systems which are part of a larger environment. CS attempts to find solutions to these problems.

A: Yes there are many broad environments, but I am making my plea to IPSE suppliers, not just CS Departments.