Service-Based Software: The Future for Flexible Software

Keith Bennett[1], Paul Layzell[2 *], David Budgen[3], Pearl Brereton[3], Linda Macaulay[2], Malcolm Munro[1]

[1] *Department of Computer Science, University of Durham, UK*
[2] *Department of Computation, UMIST, UK*
[3] *Department of Computer Science, Keele University, UK*
[*] *Contact for correspondence (email: paul.layzell@umist.ac.uk)*

Abstract

For the past 40 years, the techniques, processes and methods of software development have been dominated by supply-side issues, giving rise to a software industry oriented towards developers rather than users. To achieve the levels of functionality, flexibility and time to market required by users, a radical shift is required in the development of software, with a more demand-centric view leading to software which will be delivered as a service within the framework of an open marketplace. Already, there are some signs that this approach is being adopted by industry but in a very limited way. We summarise research and a research method which has resulted in a long-term strategic view of software engineering innovation. Based on this foundation, we describe more recent work, which has resulted in an innovative demand-side model for the future of software. We propose a service architecture in which components may be bound instantly, just at the time they are needed – and then the binding may be discarded. A major benefit of this approach is that it leads to highly flexible and agile software, that should be able to meet rapidly changing business needs.

1. Introduction

For many years, software engineers have striven to produce methods to address key problems which inhibit the development and deployment of increasingly sophisticated software-based systems [7]. Initially, such problems focused around the delivery of software and the management of its inherent complexity. For over 40 years, this focus of attention has ranged from a crude division of complexity into *data and process,* through an understanding of structure and decomposition leading to structured programming and stepwise refinement, into the age of process and methods (such as SSADM, JSD and OMT) and such object-centred concepts as design patterns.

Any analysis of its short history cannot fail to recognise the significant impact that software development techniques have had on the quantity, quality and complexity of delivered software and the impact that such software has had on wealth creation and improving the quality of life.

However, the internet age has ushered in a new era of highly dynamic and agile organisations which must be in a constant state of evolution if they are to compete and survive in an increasingly global marketplace. This era poses significantly new problems for software development, characterised by a shift in emphasis from producing ‘a system’ to the need to produce ‘a family of systems’, with each system being an evolution from a previous version, developed and deployed in shorter and shorter business cycles.

In 1995, British Telecommunications plc (BT) recognised the need to undertake long-term research which would lead to different, and possibly radical, ways in which to develop software for the future. BT commissioned a group of UK universities to undertake this research. Senior academics from UMIST, Keele University and the University of Durham, came together with staff at BT to form DiCE (The Distributed Centre of Excellence in Software Engineering), the body which would work towards the development of a new approach to the production of highly flexible, but robust, software to meet the needs of the new, emerging organisations that would drive economies in the 21st century. The method and outcome of this research is summarised in Section 2 of the paper. In Section 3, we express the objectives of the current phase of the research in terms of the vision for software- how it will behave, be structured and developed in the future. In turn, the vision developed by the group presented a grand challenge for software engineering- how to deliver the vision. Thus from 1998, the core group of researchers switched attention to developing a new overall paradigm for software engineering, leading to the development of a service-based approach to structuring, developing and deploying software. This new approach is described in the second half of this paper. The core technical issue is the service architecture, and this is presented in sections 4 and 5. An example is given in section 6, whilst related work, particularly addressing interdisciplinary issues, is summarised in section 7.

2. Developing a Future Vision

Software has become a critical element in all aspects of modern life, supporting wealth creation and being deployed in products and processes designed to improve the quality of life. The demands placed on the software engineering community, such as productivity, flexibility, robustness and quality, have increased at an exponential rate, leading to new development paradigms, formalisms and methods of working, the success of which have been truly remarkable.

However, in spite of the ability of the software engineering community to respond to these demands, there is still criticism of software systems and the methods employed in their development, such as high cost, long time-to-market and poor flexibility. Many of these issues have been accentuated through the widespread use of the internet and the acceleration of business cycles that is enabled by e-business and its support technology.

The aim of the BT-funded work conducted by the DiCE group was to form a vision of the future of software and software development, based upon systematic use of expert judgement and peer review, leading to the establishment of a long-term research agenda that could help meet the needs of society for software that is reasonably priced, reliable, adaptable and available when and where needed. The detailed rationale for this work is contained in [1].

From the outset, part of the DiCE philosophy was that the group should take a *holistic* view of software and software engineering. In particular, the group wanted to avoid the pitfalls inherent in viewing software from a specialist perspective, either in terms of technologies (e.g. formal methods, object orientation, component based approaches, agents), or in terms of life cycle phases. Therefore five hypotheses were developed, the aim of which was to postulate how software working practices might change in the period up to 2005, irrespective of how these might be realised. In summary the hypotheses were:

H1. There will be a shift in control of service development from software centres to customer and user sites.

H2. There will be a change in working practices within development and within customer sites involving greater globalisation of development teams and greater user involvement in system delivery.

H3. There will be a change from one view of quality to many different views, each having a different approach to evaluation.

H4. There will be a change in attitude towards software development and towards business practice which will improve acceptance and take-up of new technology.

H5. There will be a change from the inability to predict service behaviour to managing complexity.

From these hypotheses, the DiCE group formulated three questions about the future of software: *How will software be used? How will software behave? How will software be developed?* In answering these questions, a number of key issues emerged.

K1. Software will need to be developed to meet **necessary and sufficient requirements**, i.e. for the majority of users whilst there will be a minimum set of requirements software must meet, over-engineered systems with redundant functionality are not required. For example, users of a sophisticated word processor may only need a very small subset of its capabilities and, from the user's point of view, should only need to acquire and pay for that subset.

K2. Software will be **personalised**. Software is currently packaged and marketed as a generic product with little scope for configuration or personalisation. In future, software will be capable of personalisation, providing users with their own tailored, unique working environment which is best suited to their personal needs and working styles, thus meeting the goal of software which will meet necessary and sufficient requirements.

K3. Software will be **self-adapting**. Software will contain reflective processes which monitor and understand how it is being used and will identify and implement ways in which it can change in order to better meet user requirements, interface styles and patterns of working. It will also identify the need to commission new or changed software and decommission redundant software as and when user requirements change and thus supporting personalisation.

K4. Software will be **fine-grained**. Future software will be structured in small simple units which co-operate through rich communication structures and information gathering. This will provide a high degree of resilience against failure in part of the software network and allow software to re-negotiate use of alternatives in order to facilitate self-adaptation and personalisation.

K5. Software will operate in a **transparent** manner. Software may continue to be seen as a single abstract object even when distributed across different platforms and geographical locations. This is an essential property if software is to be able to reconfigure itself and substitute one component or network of components for another without user or professional intervention.

3. An Interdisciplinary View of Software

Having established a baseline vision of the future, the DiCE group tested its hypotheses, questions and issues in a *Forecasting the Future* workshop. At this, senior academics from a range of disciplines (organisational sociology, psychology, law, retail marketing, engineering and biochemistry) were invited to validate the work and enhance it with their own contributions.

The outcome was a significant turning point in the work of the DiCE group as it identified a fourth question: *How will software and society interact?* Analysis of this question gave rise to a number of further important issues which serve to highlight the true interdisciplinary nature of software and the software engineering process.

Trust and confidence emerged as a key issue when using software, ranging from the concept of *brand* (luxury v. utility software) to the extent to which users need appropriate mental models of software behaviour in order to have trust and confidence in its performance. In the latter case, the problem of introducing a new user to a mature and sophisticated product was highlighted, together with the need for the software to 'grow' with the user's experience.

Related to trust and confidence are the issues of **risk, responsibility, recovery and redress**; what happens when software fails and, with the emergence of component-based approaches, how can you ensure accountability in system development and evolution.

The nature of software with respect to individualism v. control opened a range of issues about the **political nature of software** and the extent to which users can be safely permitted to evolve and adapt software, relating closely to the issue of software personalisation and adaptation.

After analysing the outputs from this first phase of work, it became clear that the issue of **interdisciplinarity** would be critical to developing a future vision of software. A significant proportion of software does not exist in isolation but in a political, social, economic and legal context. To fully understand software and to achieve the highest levels of productivity and quality, it is essential that the rigid boundaries between software and its environment are broken down (as shown in the figure below) so that software naturally **incorporates** an understanding of its environment and context rather than simply **interacting** with them. Clearly this view of software relates to systems which directly interact with users—the set of issues for embedded and real-time control systems are somewhat different, and are not addressed in this research.

4. The Service-Based Vision

4.1 Supporting Emergent Organisations

Most software engineering techniques are conventional supply-side methods, driven by technological advance. This works well for systems with rigid boundaries of concern, such as embedded systems, but it breaks down for applications where system boundaries are not fixed and are subject to constant urgent change. These applications are typically found in **emergent organisations**- "organisations in a state of continual process change, never arriving, always in transition" [2]. Examples are *e-businesses* or more traditional companies which continually need to reinvent themselves to gain competitive advantage [3]. An example of this may be that of a firm of stockbrokers who may have a need to introduce a new service overnight; the service may only exist for another 24 hours before it is replaced by an updated version.

The subsequent research by the core DiCE group has taken a **demand-led** approach to the provision of software services, addressing delivery mechanisms and processes which, when embedded in emergent organisations, give a software solution in emergent terms-one with continual change. The solution never ends and neither does the provision of software. This is most accurately termed *engineering for emergent solutions*.

4.2 A Service-Based Approach

This **service-based model of software** is one in which services are configured to meet a specific set of requirements at a point in time, executed and discarded-the vision of instant service, thus conforming to the widely accepted definition of a service:

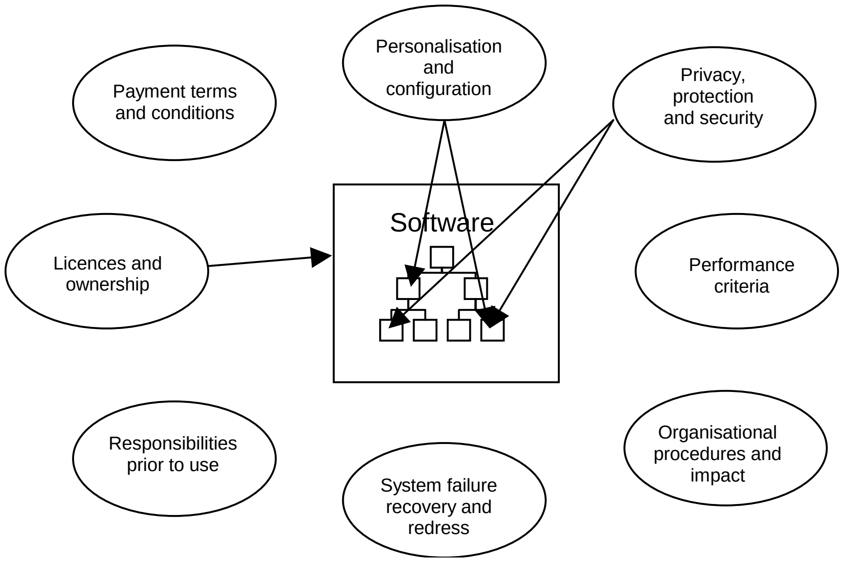
Payment terms and conditions
Personalisation and configuration
Privacy, protection and security
Software
Licences and ownership
Performance criteria
Responsibilities prior to use
System failure recovery and redress
Organisational procedures and impact

Figure 1: A Service Delivery Environment

> "an act or performance offered by one party to another. Although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in ownership of any of the factors of production" [4].

Services are composed out of smaller ones (and so on recursively), procured and paid for on demand, as and when needed. A service is not a mechanised process; it involves humans managing supplier-consumer relationships. This is a radically new industrial model for software, which could function within markets ranging from a genuine open market (requiring software functional equivalence) to a *keisetzu* market, where there is only one supplier and consumer, and both work together with access to each other's information systems to optimise the service to each other.

This strategy enables users to *create, compose and assemble* a service by bringing together a number of suppliers to meet needs at a specific point in time. An analogy is selling cars: today manufacturers do not sell cars from a pre-manufactured *stock* with given colour schemes, features etc.; instead customers configure their desired car from series of options and only then is the final product assembled. This is only possible because the technology of production has advanced to a state where assembly of the final car can be undertaken sufficiently quickly.

Software vendors attempt to offer a similar model of provision by offering products with a series of configurable options. However this offers only extremely limited choice - consumers are not free to substitute functions with those from another supplier since the software is subject to *binding* which configures and links the component parts and makes it very difficult to perform substitution. The aim of this research is to develop the technology which will enable *binding* to be delayed until immediately before the point of execution of a system. This will enable consumers to select the most appropriate combination of services required at any point in time.

However *late binding* comes at a price, and for many consumers, issues of reliability, security, cost and convenience may mean that they prefer to enter into contractual agreements to have *some early binding* for critical or stable parts of a system, leaving more volatile functions to late binding and thereby maximising competitive advantage. The consequence is that any future approach to software development must be interdisciplinary so that non-technical issues, such as supply contracts, terms and conditions, certification, and redress for software failure are an integral part of the new technology.

4.3 Current Approaches

The term "software as a service" is beginning to gain acceptance in the market-place; however the notion of service-based software extends beyond these emerging concepts. We have identified three notions of software service that are currently in use.

The **rental model** is based upon the rent or hire of software from a producer, as a means of reducing upfront costs. For example, more than half of UK companies are planning, within the next year, to use services that allow them to rent certain software items rather than buying them [5]. Strictly, the rental model does not imply any change to the physical structure or installation location of software, and so is merely a change in payment method.

The **server model** is based upon the use of thin clients to offer software from a central server with a charging regime based on pay-per-use, typically to avoid upfront procurement costs by user organisations and achieve up-to-the-minute maintenance through access to the latest release of software. However this model does not necessarily require any change to the basic structure of the software and relies on achieving user flexibility through the distribution network. The problem of maintenance and delivering flexibility is passed to the host organisation and provides little scope for easily delivering software variants and personalised solutions.

The **service package model** is based on a well established trend for products to be *packaged* with a range of services designed to support and enhance product use. For example, an airline offering seats as its core product, may offer a range of additional, value-adding services as a package. Similarly, some software producers offer *business solutions* comprising product and service elements. Again, this concept does not imply any change in the nature of the underlying software product itself, although users may be provided with different *experiences* through the service layer surrounding the product.

A concept that bears some relationship to that of the service is that of the *component* [8]. Indeed, component technology may well be an important interim step on the way to developing a realisation of the service concept. Component-based development already includes such technical concepts as composition, substitution and evolution, as well as more consumer and market-oriented issues such as supplier confidence [9]. However, components are very much a system implementation concept, and both constructional issues such as binding mechanisms and architectural forms as well as conceptual issues such as characterisation of components require to be resolved in order for components to realise their full potential.

4.4 Bind Once-Execute Once

A truly service-based role for software is far more radical than current approaches, in that it seeks to change the very nature of software. To meet users' needs of flexibility and personalisation, an open market-place framework is necessary in which the most appropriate versions of software products come together, are bound and executed as and when needed. At the extreme, the binding which takes place prior to execution, is discarded immediately after execution in order to permit the 'system' to evolve for the next point of execution. Flexibility and personalisation are achieved through a variety of service providers offering functionality through a competitive market-place, with each software provision being accompanied by explicit properties of concern for binding (e.g. dependability, performance, quality, licence details etc), covering both technical and non-technical properties of binding.

5. Key Challenges

In the service-based model of software, two key issues must be considered: (i) the nature of the service supply chain and (ii) the anatomy and structuring of services. Each highlights a range of research problems covering both *technical issues,* such as how system functionality is delivered, standardisation of interfaces and performance, as well as *non-technical issues,* such as contractual relationships, the role of markets, industry and economic models and perception (brand, quality etc.).

5.1 Nature of the Service Supply Chain

Services are supplied through a *service supply chain*.. At the top of the chain are consumers with needs that are satisfied through the provision of software services. These services are provided through a hierarchy of service providers, initially through a consumer-supplier (retail) market and subsequently by sub-contracting to other suppliers through supplier-supplier (wholesale) markets. At the bottom are primitive services which provide basic system functionality, with higher level providers adding increasing value to these primitives.

In a totally flexible world, consumers would be free to renegotiate service provision every time they required a service, giving *ultra-late* binding between the business problem and solution. However if this approach were followed literally, performance with current technology would be unacceptable. A more pragmatic approach is to use market forces in which service providers would form alliances jointly to provide popular services. Speculative partnerships would form to promote certain new services and brokerages would form to identify such partnerships. There would be market pressures to attain continuing service improvement. In these cases, there would also be an aggregation of existing services, plus the introduction and creation of new ones. Management of such partner relations would become central, involving negotiation, trust and co-operation; financial arrangements and legal responsibilities would need to be unambiguous.

To the consumer, issues like branding, quality, cost and delivery will take on the same role as in other service industries. For a service provider, the service functionality must be described, its quality attributes made clear, and the interface well-defined. The functionality is a business interface, not just a technical interface. To summarise, the solution addresses how to configure, assemble, compose and bind, on demand, a hierarchy of services which, together, meet the customer requirement for some top-level service. This process covers technical and non-technical service elements. There may also be a need, for long-lived service partnerships to evolve aspects of the service agreement while the service is operational. Similarly, each service may offer a range of configuration and personalisation options which may be specified by the customer when requesting an instance of the service.

The anatomy of a service provider comprises a range of human skills and technology. A *service delivery* layer interacts with customers or other service providers for the delivery of required services. A *service personalisation* layer then defines the architecture by which services will be configured. A critical success factor is the need for a negotiated and agreed architecture by which services can be configured, provided and used by the consumer with the minimum change to either the service provision or recipient's system. Success will depend upon the management of both and non-technical issues.

Service providers will conduct *service acquisition* to use sub-services from other providers, as well as adding value through *service development*. These will typically be software based, but need not be. Finally, orthogonal to the key service provider skills, will be the need to manage and control the service provision process, both at a technical level ("are we supplying services correctly?") and a business level ("are we supplying the correct services?").

Existing work on service definition, enterprise modelling, software components and agent technologies play a key role, along with interfacing and negotiation protocols. However these are not sufficient to deliver software as a service as they do not address the non-technical issues which will arise. Much of the existing enterprise modelling work and brokerage mechanisms assume simple *anonymous* market models in which consumers and suppliers come together and match requirements with supply. Real markets within a supply chain are more complex than this, with the consumer-supplier relationship being two-way: consumers influence the type and method of supply and the supplier influences a consumer's business processes in order to adapt to their product. Issues such as ability to deliver, responsibility, redress and recovery are critical to a long-term, lasting relationship between consumers and suppliers, and require formalisation within the overall service provision context.

5.2 The Anatomy and Structuring of Services

In order to achieve the flexibility required of a service-based environment, the nature and structure of the underlying software itself must change, to become *finer-grained* (K4) and *operate transparently* (K5) in order to allow seamless evolution. This is necessary in order to permit a software architecture in which non-technical, service-level issues can be assigned to precisely the element of the software to which they relate, as opposed to the current situation where service-levels (payment terms, conditions of use etc.) relate to a configured and bound software artefact. Thus in Figure 1, whilst differing non-technical issues may relate to specific parts of a software product, the fact that an entire product is bound together means that, at most, non-technical service-level issues apply to the entire software product.

Thus a service can be defined as a highly cohesive software component, which requires minimum coupling [6] *wrapped* by a service-level agreement which defines all the terms and conditions of its use.

The existing low-level structure of software, which simplistically separates data from process, supported by a user interface and early binding must also be enhanced. The software kernel of a service must be described in terms of higher-level constructs if consumers are to be able to switch easily between service providers: data must be modelled as *information* - a higher level concept which *hides* issues of representation and focuses on *content*. A clearer distinction must also be made between *business rules*, the policy by which a service will operate and *process*, the means by which a business rules are achieved. The *personalisation* of a service, to meet the specific needs of consumers must also be achievable. One approach to be considered is the employment of domain-specific languages to allow users to personalise a service, although this must be constrained within the business rules of the service in order to ensure that the service retains integrity. Finally, the issue of ultra-late binding must be addressed and will require identifying pragmatic means of addressing functional equivalence between services in order to allow them to be substituted.

In terms of the service-level agreement which *protects* the software kernel, issues concerned with service marketing and promotion, service negotiation, service delivery standards and mechanisms and post-service management (such as billing and accounting) need to be addressed.

6. User Benefits- An Example

The key user benefits of the service-based approach to software can most easily be seen through an example. Consider a simple payroll system which is required to register hours worked per month by each employee, calculate monthly pay, calculate tax and social costs, initiate bank transfer payment to each employer and tax collection service, issue payslips and charge salary costs to the appropriate cost centres in a company's ledger.

Traditionally, such payroll software will be built as a standard product, employing a range of configuration options to tailor specific processes to each user organisation. Each element of even a simple payroll system, requires a range of expertise (in pay calculation, taxation laws, electronic funds transfer etc.) and whilst it may be possible to select 'best-in-class' suppliers for each element, typically a user is presented with a specific combination of function, likely to have been written by the same organisation and hence not guaranteed to be using best-in-class. The 'bound' software product with its *internal* interfaces and non-technical service-level issues linked to the entire product, also makes it difficult to substitute alternative component parts. It is like the experience of buying consumer goods in which users are warned 'opening the box invalidates the warranty' - to try and replace a single, internal software component invalidates the implied (or explicit) service-level agreement which surrounds the software.

In a service-based approach to software, service-levels agreements are bound to individual software services, which can be procured, linked, executed and subsequently replaced on an individual basis, but without needing to renegotiate an entire service-level agreement bound to a single, assembled system.

Thus if such an architecture were to be employed for providing the payroll system, individual software services could be changed as necessary. Tax calculation services could be replaced as different methods of taxation or calculation are enforced or where employees of subsidiaries or branches come under a different tax regime. Methods of electronic funds transfer could be changed to take advantage of new payment techniques offered by different financial organisations. Similarly, the printing of payslips could be replaced by electronic notification of salary. In the extreme, the payment for each employee might utilise a different set of services, while maintaining the integrity of the whole.

7. Current and Future Status

The work outlined in previous sections outline a radical and ambitious interdisciplinary research programme which proposes a general architecture for a service-based approach to software.

It is recognised that a basic proof of concept for the technical core of this approach is essential to give an experimental framework in which the detailed concepts and ideas can be tested. Currently, research is in progress to develop this, using industry standard tools. Implementation of a simulation model for exploring the potential speed up of time to market using a service-approach is also under development.

Although it is possible to develop models of service supply chains and to define the anatomy of a service provider, it is important to recognise that there is no grand design, methods or set of tools that will achieve highly flexible, service-oriented software. Whilst there will be such artefacts, they need to reside in a broader social, economic and legal framework, which makes this approach interdisciplinary as a fundamental pre-requisite. As the proof of concept demonstrator evolves, further work can be conducted on these interdisciplinary issues.

Two specific areas are under development. Firstly, a range of management specialists are contributing to the development of a service-based model of software, whilst other disciplines are assisting in the development of the non-technical models necessary to deliver service-based software.

Secondly, specific application domains are being used to demonstrate and evaluate the research outputs. This bottom-up approach will help identify common problems and issues across domains, from which new styles of software delivery processes can be built. Each domain has a clearly identified user community whose role it is to articulate problem characteristics of their domain and help in assessing the relevance of the service-based model of software to meeting their software needs.

In spite of the work-in-progress, a large number of other research issues remain and part of the purpose of this paper is to help define a longer-term research agenda for the software engineering community. Key issues include:

- How do consumers know what services are available and how do they evaluate them?
- How do consumers express their requirements?
- How are services composed?
- How are services tested?
- What is the appropriate, high integrity, service delivery infrastructure?
- How must consumers' data be held to enable portability between different service suppliers?
- What standards can be used or must be defined to enable portability of service?
- What will be the impact of branded services and marketing activities (high quality v low price)?
- How can organisations benefit from rapidly changing services and how will they manage the interface with business processes?
- How will individuals perceive and manage rapidly changing systems? What is the limit to the speed of change?

- What payment and reward structures will be necessary to encourage SME service suppliers?
- What will be the new industry models and supply chain arrangements?
- How can we evaluate the research outcomes?

8. Conclusion

This paper presents a collaborative research method which, over a period of three years has given rise to a radical and innovative vision for future software. From 1995 to 1998, attention was focused on creating this vision and validating it within a multi-disciplinary workshop. Five hypotheses were formulated which were then used to frame key questions, the answers to which have led to a vision of service-based software.

Since 1998, work has addressed the implementation of this vision. A preliminary architecture has been defined and is being used as the basis of a proof of concept demonstrator.

It is pertinent to ask how relevant does the vision remain, now that five years of the original 10 year timescale has passed. When starting in 1995, the internet was at a very early stage. The explosive growth of the internet has been in a narrow form, mainly restricted to data interchange (for example, in XML, business to business e-commerce etc.). The proposed vision foresees the internet being used along a 'second dimension', to support an open demand-led marketplace for software based on ultra-late, as-needed binding. Thus the internet has not overtaken the vision; in contrast, it has increased the urgency for its implementation.

Expressed in a different way: our ability to change software to meet new business needs (i.e. software evolution) improves by only a few percentage points each year. Even very radical projections of current technologies, tools and processes (for example, COTS, reuse etc) seems unlikely to accelerate this rate very significantly. Yet businesses developing in *internet time* need orders of magnitude improvement in change rates to bring ideas to market far faster than is currently possible.

The basic thesis put forward here is that, unlike most work in the field, software engineers cannot look to technology alone to produce such magnitudes of improvement. To realise a vision of highly flexible service-based software, a holistic and interdisciplinary approach to software engineering is essential, bringing together disciplines such as law, business and economics with software engineering.

This is a huge challenge for the software engineering community, which needs to readily and wholeheartedly embraced. For if user needs cannot be met, software engineering will be deemed to have failed, and many of the significant benefits claimed by the internet will be lost.

Acknowledgements

The authors would like to acknowledge the support and input from BT plc, especially David Griffiths and Charles Stannett. The authors also wish to acknowledge the contribution of Peter Henderson and Pieter Hartel, University of Southampton, UK. K.H.Bennett would like to thank the Leverhulme Trust and EPSRC for support during this research.

References

[1] P.Brereton, D.Budgen, K.Bennett, M.Munro, P.Layzell, L.Macaulay, D.Griffiths and C.Stannett, "The Future of Software: Defining the Research Agenda", Comm. ACM, Vol.42, No.12, December 1999

[2] D. Truex, R.Baskeville and H.Klein, "Growing Systems in Emergent Organizations", Comm.ACM, Vol.42, No.8, August 1999

[3] M. Cusumano & D. Yoffe, "Competing on Internet Time – Lessons from Netscape and its Battle with Microsoft", Free Press (Simon & Schuster) 1998

[4] C.Lovelock, S.Vandermerwe, B.Lewis, *Services Marketing*, Prentice Hall Europe, 1996

[5] "Software Rentals to Increase", News Digest Report, Financial Times, 13 April 2000, p8 (UK edition)

[6] E.Yourdon and L.Constantine, *Structured Design*, Yourdon Press, 1978.

[7] S. Shapiro, "Splitting the Difference: The Historical Necessity of Synthesis in Software Engineering", IEEE Annals of the History of Computing, Vol. 19, No. 1, January 1997, pp20-54.

[8] C. Szyperski, *Component Software – Beyond object-oriented programming*, Addison-Wesley, 1998.

[9] O.P. Brereton and D. Budgen, "Component Based Systems: A Classification of Issues", accepted for publication in IEEE Computer.