17 IMPLICATIONS OF A SERVICE-ORIENTED VIEW OF SOFTWARE

Paul Layzell
Department of Computation
University of Manchester
Institute of Science and Technology
Manchester M60 1QD
United Kingdom

Abstract

A change in attitudes and approaches to software development is emerging from the software engineering community, in which software is no longer regarded as a product, but as a service. This paper outlines the history of this change and reviews the implications of a service-oriented approach to information systems development. The chief impact is the need for software engineers, information systems developers, and managers to take a much broader view of the development and deployment process, with far reaching implications for traditional IS departments.

1. INTRODUCTION

For many years, software engineers and information system developers have striven to produce methods of software development that lead to the successful specification, design, implementation, and deployment of information processing systems.

Such methods abound, ranging from technical approaches to software development through to broader, socio-technical based approaches. No contemporary organization can function without the sophisticated information processing support systems these methods deliver; however, each method has its limitations and much criticism is still levelled for the general inflexibility in approach and, more often, the failure to fully understand the essential user requirements.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35489-7_33
N. L. Russo et al. (eds.), *Realigning Research and Practice in Information Systems Development*

While considerable research effort has been directed toward improving system development methods, either by taking a new technical approach (Booch et al. 1999) or by placing greater emphasis on broader user and organizational issues (Checkland and Scholes 1990), the underlying structure of software has remained broadly the same (Pressman 1997).

Researchers must challenge not only the appropriateness of contemporary information system development methods, but also whether the underlying technical structure of software is correct and whether mis-formed structures are the root cause of the continuing difficulties experienced in developing and deploying successful information systems (Brereton et al. 1999).

To address this fundamental issue, in 1995 British Telecommunications plc (BT) recognized the need to undertake long-term research that would lead to different, and possibly radical, ways in which to develop information systems of the future. BT commissioned a group of universities in the United Kingdom to undertake this research. Senior academics from UMIST, Keele University, and the University of Durham, came together with BT staff to form DiCE (The Distributed Centre of Excellence in Software Engineering), a body that would work toward the development of a new approach to the production of highly flexible, but robust, software to meet the needs of the new, emerging organizations that would drive economies in the 21st century.

The emphasis on highly flexible software arose because the Internet age was ushering in a new era of highly dynamic and agile organizations that must be in a constant state of evolution if they are to compete and survive in an increasingly global marketplace (Truex et al. 1999). The results of this work are summarized in section 2, but full details can be found in Brereton et al.

Building upon this work, a key research issue emerged—*the changing nature of software*—in which the view of software shifts from being a product to one in which it is considered a **service**.

This change in attitude has far-reaching consequences for information system software, information system users, information system development methods, and information system developers.

A summary of the key concepts of *software as a service* is presented, followed by a review of the key consequences for information systems methods and developers. It is hoped that the issues raised present a fresh perspective on information systems and their development, laying down a research agenda that will engage the wider information systems and software engineering community.

2. A CHANGE IN PERSPECTIVE

While every individual and organization is heavily reliant on sophisticated information processing systems, there is still much criticism of the development process and resulting products (Jones 1996). Criticisms include high cost, long

lead times in development, and poor flexibility of the final system. Many of these issues have been accentuated through the widespread use of the Internet and the acceleration of business cycles demanded by e-business (Reifer 2000).

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.

From the outset, part of the DiCE philosophy was to take a holistic view of software and software-based systems; in particular, 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.

To achieve this holistic view, the core research group was supplemented by a team of interdisciplinary experts who knew something about software, but whose views were driven by their own domains and expertise and would thus inevitably shed a different light on the problems of software development. Five key themes emerged from the work.

First, software will need to be developed to meet necessary and sufficient requirements, i.e., for the majority of users, while 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 and not be overburdened by the cost of ownership of features they do not use or require.

Second, building upon this initial outcome, software of the future will be personalized. Software is currently packaged and marketed as a generic product with little scope for configuration or personalization. In the future, software should be capable of personalization, providing users with a tailored, unique working environment that best suits their personal needs and working styles—an essential requirement in job design (Arnold et al. 1998).

Third, software should be adaptable. For example, software might contain reflective processes that 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. Adaptation is also concerned with the need to commission new or changed software and decommission redundant software as and when user requirements change, thus supporting personalization.

Fourth, arising from the *share and communication* philosophy embodied in the Internet, software should be fine-grained and structured into small, simple units that cooperate through rich communication structures and information

gathering. This will also provide a high degree of resilience against failure in part of the software network and allow software to renegotiate use of alternatives in order to facilitate self-adaptation and personalization.

Finally, in order to engender trust and confidence in increasingly complex software, software needs to operate in a transparent manner.

3. THE SERVICE-BASED VISION

3.1 Software as a Service

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, as in emergent organizations.

To counter the problems of rapid evolution of software, particularly for emergent organizations (Truex et al. 1999), an alternative view of software is required, in which systems can be rapidly composed initially from primitive components and subsequently from application frameworks composed of numerous primitive components.

At first, this appears to be component-based software engineering; however, component-based software engineering only delivers part of the answer and in itself is not sufficient. Neither is the component-based approach wholly appropriate (Brereton and Budgen 2000). There is considerable evidence that current component-based approaches are still relatively cumbersome and inflexible and do not necessarily give the levels of ultra-flexibility required for extremely dynamic organizations with modification to conventional technologies (Fingar 2000).

A more radical view of software is required, namely a demand-led approach to information system development in which the primary focus of concern is not software—the *product* of the development process—but rather the **service** provided by the software.

For developers, the issue is no longer how to build information processing systems that have properties of configurability, maintainability, and longevity. Instead it is the ability to rapidly compose information processing systems so that they can be used and then discarded, to be replaced by a new system composition that meets a new set of requirements, moving the focus of development from the product to the service provided. At the extreme, the service-oriented view discards the costs of ownership of a system because every instantiation of a set of requirements is executed only once and then discarded, before the original requirements evolve and a new system is deployed.

From a user's perspective, software *is* a service, with the outcomes from its use bringing business benefit, while the software itself has little intrinsic value, and thus conforming to the widely accepted definition of a service:

> 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 (Lovelock et al. 1996, p. 6).

Software thus becomes a series of services that are delivered through a supply chain with development effort shifting from the initial creation of basic services, which the software engineering community can do reasonably well, to the assembly of services through supply chains, which the information systems community can probably manage better than the software engineers.

Software vendors attempt to offer a similar model of provision by offering products with a series of configurable options through products such as SAP. However, this offers choice limited only to that which is built into the overall software offering and the manner in which it can be assembled and integrated (Sprott 2000). 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 difficult to perform substitution.

Therefore, the goal must be to develop 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 maximizing 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.

The key focus of the software as a service concept is, therefore, not about components or system configuration options. Instead, software as a service is about mass markets of specific services that are progressively aggregated into useful information systems.

3.2 Related Concepts

The concept of software as a service is not new and variants are already emerging. For example, the *rental model* is based upon the rent or hire of

software from a producer, as a means of reducing upfront costs (*Financial Times* 2000). However, strictly speaking, 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 alternative *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 organizations 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 organization and provides little scope for easily delivering software variants and personalized solutions.

Finally, 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.

3.3 Extending Components

Service-oriented software clearly relates to that of the *component* (Szyperski 1998). Component-based development includes such technical concepts as composition, substitution, and evolution, as well as more consumer and market-oriented issues such as supplier confidence (Brereton and Budgen 2000). However, components are essentially a system implementation concept and both constructional issues, such as binding mechanisms, and architectural forms, as well as conceptual issues, such as characterization of components, require resolution in order for components to realize their full potential.

A truly service-oriented 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 personalization, 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 personalization 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, license details, etc.), covering both technical and non-technical properties of binding.

In order to deliver the required flexibility through interoperability of software components, existing technology must be extended so that components have both a technical and non-technical interface.

The technical interface, which is addressed in current technologies, allows for the passing of data and control between components.

However, the use of any component is accompanied by a set of implied terms and conditions, which are typically built into the design and implementation of the component and are never made explicit.

For systems that evolve slowly over a period of time, such implicit terms and conditions can be managed through informal processes of program understanding, software maintenance impact analysis, and software maintenance change design. However, in a rapidly changing environment, such informality is a hindrance—the implicit must be made explicit—so that components can be selected, combined, executed, and released in an ultra-short timescale.

Examples of the informal issues implied in software include payment terms and conditions, personalization and configuration, privacy, protection and security, licenses and ownership, responsibilities prior to use, system failure, recovery and redress, performance criteria, and organizational procedures and impact.

These issues can be regarded as a *service level agreement* defining the terms and conditions of use of an individual software component and it is the explicit modeling of these issues which gives rise to the change in the nature and structure of software from product to service.

4. THE SERVICE ARCHITECTURE

Figure 1 shows the architecture of a software service and how the core software functionality is extended to incorporate the necessary service-level related issues in order to contract and receive supply of the service. The key non-technical issues of service marketing, service negotiation, service delivery, and post-service management each contain a variety of critical management issues, necessary to enable rapid deployment and replacement of software components and together form part of a standard service delivery model (e.g., Gebrauer and Scharl 1999). For example, service marketing is concerned with the interface between the software service and service brokers who "introduce" services to users. Service negotiation concerns the process by which agreement is reached to use a service under a set of terms and conditions, ensuring that services are properly deployed and used in appropriate contexts. Service delivery is concerned with ensuring that services are available at the appropriate

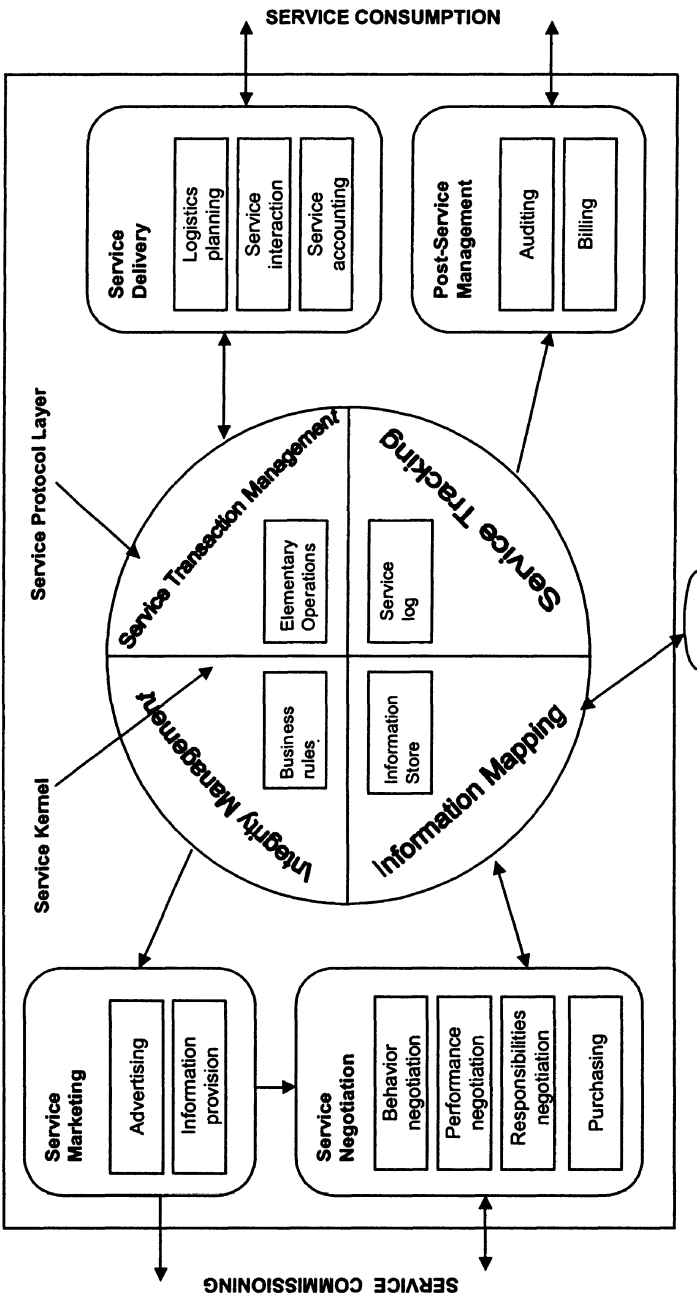

Figure 1. A Software Service

Figure 2. Service-Based Architecture

time and that a check is made to ensure that "delivery" is made according to the negotiated contract. Finally, post-service management conducts a variety of after-sales services.

Figure 2 shows how different services can be combined to deliver an "end-service" to users. For example, the end-service might consist of a *calculator service*, i.e., the ability to perform a simple arithmetic operation on two numbers. End-users employ a brokerage service to identify possible calculator services, from which one specific service is selected. This service in turn may sub-contract another service which then uses further brokerage services to procure services to input numbers, perform a calculation, and output the result. In turn, an arithmetic broker will identify and select a specific arithmetic service, in this case, a subtraction service.

Each service is one of a specific type of service: specialist end-service provider (input, output, addition, etc.), a service wholesaler (calculator), a value-adding service (calculator sub-contractor, input-output etc.), a service broker (broker, input-output broker, etc.) or a service integrator (any service that combines others services).

The service marketing, negotiation, delivery, and post-service management are clearly critical to ensuring the successful operation of the service supply chain.

5. USER BENEFITS

The key user benefits of the service-oriented approach to software can most easily be seen through an example. Consider a simple payroll system that 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 employee and tax collection service, issue pay slips and charge salary costs to the appropriate cost centers 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 organization.

Each element of even a simple payroll system requires a range of expertise (in pay calculation, taxation laws, electronic funds transfer, etc.) and, while 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 organization 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-oriented approach to software, service-level 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 organizations. Similarly, the printing of pay slips could be replaced by electronic notification of salary. In the extreme, the payment for each employee might utilize a different set of services, while maintaining the integrity of the whole.

6. CURRENT AND FUTURE STATUS

The work outlined above develops a radical and ambitious interdisciplinary research program that proposes a general architecture for a service-oriented approach to software.

A basic proof of concept for the technical core of this approach has been developed using standard industry tools and implementation of a simulation model for exploring the potential speed up of time to market using a service-approach has been developed and is under evaluation (Bennett et al. 2001).

Although it is possible to develop models of service supply chains and to define the anatomy of a service provider, it is important to recognize that there is no grand design, methods, or set of tools that will achieve highly flexible, service-oriented software. While 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 prerequisite.

While it is clear that a technical platform for the rapid composition of services will be possible, the interdisciplinary issues relating to the service-level agreements surrounding software services present the greatest challenge.

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 to ensure that rights and obligations are properly handled?

- How are services tested in order to provide trust and confidence in services?
- 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 vs. low price)?
- How can organizations 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?

These issues will have a significant impact on the future methods of information system construction, deployment, and use, as well as changing the very nature of the information systems department. Key skills will shift from programming and component assembly, to much broader issues where technical problems are embedded in socio-economic issues of supply chain management, trust and confidence, flexibility and empowerment, payment and reward.

The software engineering community in its work on component-based systems, object brokerage, distributed systems management, and portability of data has moved a long way to delivering the technical environment necessary for software services. It remains a challenge to the wider information systems community and cognate disciplines to develop techniques for the modeling and deployment of the service-level layer that must surround software components if they are to meet the needs of highly flexible information systems.

7. ACKNOWLEDGMENTS

The author would like to acknowledge the support and input from the following colleagues in the development of the concept of "software as a service": Professors Keith Bennett and Malcolm Munro, University of Durham, UK; Professor David Budgen and Dr. Pearl Brereton, Keele University, UK; Professor Linda Macaulay, Dr. Nik Mehandjiev and John Keane, UMIST, UK.

8. REFERENCES

Arnold, J., Cooper, C. L., and Robertson, I. T. *Work Psychology* (3rd Edition), Harlow, England: Financial Times/Prentice Hall, 1998.

Bennett, K., Munro, M., Gold, N. E., Layzell, P. J., Budgen, D., and Brereton, P. "An Architectural Model for Service-Based Software with Ultra Rapid Evolution," in *Proceedings*

of the IEEE International Conference on Software Maintenance, Florence, Italy, November 2001 (forthcoming).

Booch, G., Rumbaugh, J., and Jacobson, I. *The Unified Modeling Language User Guide*, Reading, MA: Addison Wesley, 1999.

Brereton, P., and Budgen, D. "Component Based Systems: A Classification of Issues," *IEEE Computer* (33:11), November 2000, pp. 54-62.

Brereton, P., Budgen, D., Bennett, K., Munro, M., Layzell, P., Macaulay, L., Griffiths, D., and Stannett, C. "The Future of Software: Defining the Research Agenda," *Communications of the ACM* (42:12), December 1999.

Checkland, P., and Scholes, J. *Soft Systems Methodology in Action*, Chichester, England: J. Wiley and Sons, 1990.

Financial Times. "Software Rentals to Increase," News Digest Report, *Financial Times* (UK edition), April 13, 2000, p.8.

Gebrauer, J., and Scharl, A. "Between Flexibility and Automation: An Evaluation of Web Technology from a Business Process Perspective," *Journal of Computer-Mediated Communication* (5:2), 1999.

Fingar, P. "Component-Based Frameworks for E-Commerce," *Communications of the ACM* (43:10), October 2000.

Jones, C. *Patterns of Software Systems Failure and Success*, Boston: International Thomson Press, 1996.

Lovelock, C., Vandermerwe, S., and Lewis, B. *Services Marketing*, London: Prentice Hall Europe, 1996.

Pressman, R. *Software Engineering*,(4th Edition), New York: McGraw-Hill, 1997.

Reifer, D. J. "Requirements Management: The Search for Nirvana," *IEEE Software* (17:3), May/June 2000, pp. 45-47.

Sprott, D. "Componentizing the Enterprise Application Packages," *Communications of the ACM* (43:7), April 2000, pp. 63-69.

Szyperski, C. *Component Software: Beyond Object-Oriented Programming*, Reading, MA: Addison Wesley, 1998.

Truex, D., Baskerville, R., and Klein, H. "Growing Systems in Emergent Organizations," *Communications of the ACM* (42:8), August 1999.

About the Author

Paul Layzell has worked in IT for over 20 years and is currently Professor of Software Management in the Department of Computation, UMIST (University of Manchester Institute of Science and Technology), UK. His key research interests are in the processes, management, and support technologies for the development of software and software-related products. He has worked on a number of projects concerned with improving software development productivity, both for new software products, as well as in the maintenance of large-scale, legacy systems. He has also been concerned with the introduction of new technologies, such as geographical information systems, and their impact on development processes and end-user working practices. Professor Layzell is a fellow of the British Computer Society and member of the IEEE Computer Society and Association for Computer Machinery. He can be reached by e-mail at paul.layzell@umist.ac.uk.