

The Future OF Software

Companies like British Telecommunications whose business is based on the performance of their software systems, should emphasize accessibility, adaptability, transparency, fail-safe operation, and a human face.

SOCIETY IS INCREASINGLY DEPENDENT ON LARGE AND COMPLEX SOFTWARE SYSTEMS. INDEED, if many of the current predictions about the Y2K computer problem turn out to be accurate, this dependency will be demonstrated dramatically on January 1, 2000. Users need software that meets stringent requirements, supports a range of interaction styles, can be produced quickly, and can be maintained to keep pace with the ever-increasing demand for functionality, quality, flexibility, and cost-effectiveness.

Producing such software is difficult and involves high costs and risks. Adopting the most appropriate methods, technologies, and tools at just the right time is a major problem for the software industry. Recognition of the critical role played by software in so many aspects of society has therefore led us to pursue the following goals:

- Forming a vision of software and software development based on the systematic use of expert judgement and peer review;
- Establishing the need for a long-term software-development research agenda; and
- Outlining a long-term research agenda that helps meet society's future needs for software that is

reliable, adaptable, available when needed, and reasonably priced.

Predicting the future is a popular pastime in many disciplines. In the field of software development, the February 1997 *Communications* offered many personal hopes and visions for the future of computer technology [1]. Authors expressed their personal views relating to particular technologies and applications, including databases [2], the Internet [3], and computational humanities [6]. In another personal view of future needs, Stuart Shapiro [5] examined a number of key software technology publications dating from as far back as the 1960s, concluding we need a more pluralistic approach to software engineering,

PEARL BRERETON, DAVID BUDGEN, KEITH BENNETT,
MALCOLM MUNRO, PAUL LAYZELL, LINDA MACAULAY,
DAVID GRIFFITHS, AND CHARLES STANNETT

emphasizing synthesis and trade-offs. The group perspective in the 1994 book *Software 2000: A View of the Future* [4] identified some major factors likely to influence the future of software.

We aim to extend the debate over how software will be developed and used in two ways—by stepping back from a detailed technology focus and by incorporating the views of experts from a range of disciplines. We describe not only the results of our deliberations but the process through which we generated our ideas as well. We hope to add further credence to the agenda we describe here and inspire others to undertake a similar process. Our concern is long-term research, and we suggest a number of criteria for differentiating between long- and short-term research (see Table 1).

We stress that we did not set out to establish a plan or a list of hot topics for software engineering research. Rather, by identifying the likely directions and consequences of software use, or the use context, we seek a better understanding of the axes along which research can be positioned.

The Future

The Distributed Centre for Excellence in Software Engineering (DiCE), funded by British Telecommunications P.L.C., a major U.K. telecommunications company and one of the biggest in the world, was established in 1995 with the broad aim of generating new ideas on the future direction of systems and software engineering. Its major purpose was to help meet the long-term planning needs of software-based companies, including BT. DiCE initially

included six senior academic researchers from its partner institutions: the University of Manchester Institute of Science and Technology (UMIST), Keele University, and the University of Durham, as well as a number of BT researchers.

The DiCE philosophy involves a holistic view of software and software engineering. We especially wanted to avoid the pitfalls inherent in viewing software from a specialist perspective in terms of tech-

nologies and life-cycle phases. Although the members of the DiCE team brought their own research expertise and experience to the task, we aimed to overcome some of the fragmentation often evident in software engineering research.

The evolution of the DiCE endeavor is summarized in Figure 1. Phase 1, which began at DiCE inception in 1995, followed traditional research style in that full-time research assistants pursued focused research to support periodic general discussions. These discussions led to a set of hypotheses (see Figure 2) concerning the future of software and software engineering while providing a launch pad for DiCE phase 2.

Phase 2 emphasized the expertise and experience

Table 1. Characteristics of long- and short-term research compared.

Long-term research	Short-term research
• Challenges existing assumptions	• Carried out within existing assumptions
• Concerned with identifying new problems	• Concerned with solving known problems
• High risk of no immediately usable results, even no results	• Low risk of no usable outcome
• Ideas-led	• Typically involves experimenting with new technologies
• Time to market unknown	• Constrained by market needs
• Involves an open approach to exploration and debate using multiple perspectives	• Closed, concerned with highly focused evaluations, feasibility assessments, and prototyping
• Criteria for success unknown and may be difficult to quantify	• Goals are typically project-led and involve planned outcomes
• Results not always immediately exploitable	• Results expected to be exploitable in development projects
• Involves originality and new insights, technologies, visions, ideas	• Involves producing prototypes and evaluations
• Defines the solution space for short-term research	• Produces solutions that can feed into development

of the contributing senior academics, since team members had proved themselves their most creative when working with a minimum of the kind of project management traditionally associated with software engineering projects. Similarly, within BT, the “success criteria” for the project changed between phase 1 and phase 2. During phase 1, which involved BT’s own researchers, the aim was to produce technical deliverables. In phase 2, BT repositioned DiCE internally, linking it to the research program management team, rather than to the researchers themselves. Unlike BT’s usual way of working, this linking to the management team allowed DiCE to directly support BT’s strategic software technology research profile—with the project being judged successful by BT management if it led to changes in that profile.

A series of meetings involving senior people from BT, together with the six academics, led to the “scenarios document,” which became the major deliverable to BT management. The agenda for the meetings was structured around the scenarios document and established three scenario categories, discussed in the following sections; each was brainstormed, with subsequent meetings reviewing

and developing the results from previous meetings and contributions made between meetings.

Following this brainstorming, phase 3 began with a multidisciplinary workshop called “Forecasting the Future” that brought together senior academic researchers from a range of disciplines, including organizational sociology, psychology, law, retail marketing, engineering, and biochemistry. The result was another (fourth) scenario focusing on the interaction between society and software.

How software and society will interact. Although system development is often analyzed mainly (or even solely) from a technical viewpoint, software systems routinely interact with people and society. It is therefore important to consider the future relationship between software and society, even to the extent of considering whether future development processes could become socially driven, rather than technically driven. This relatively radical view

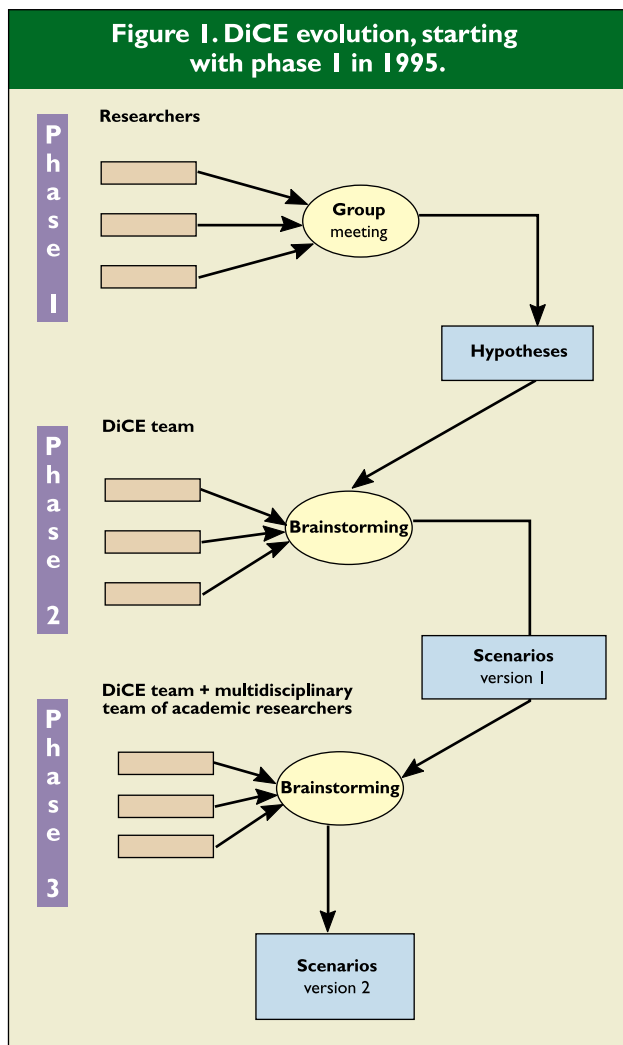
of the future of software seeks to bridge the gap between technology and society.

Software has the power to bring about significant change in both commercial and political organizational structures. As an example, communication and information technologies might permit every citizen to participate directly in government, rendering “representative” government redundant. This direct participation and representation would provide a significant challenge to existing interests, resulting in either radical social change or attempted governmental regulation.

The increasing complexity of systems means that software failures will have a more significant and visible influence on communities. A topical example is the Y2K problem. Consequently, systems developers need to address issues of trust and confidence in their systems in terms of the quality of both the systems themselves and the information they generate. Systems need to be more transparent, supporting many different mental models of a software system’s behavior and operation.

Many applications of information technology are perceived as involving the transfer of knowledge and

Figure 1. DiCE evolution, starting with phase I in 1995.



skills from humans to machines. This nominal dumbing-down of society has implications on the perception of the roles of software in society and ultimately the ability of society to innovate. A new form of Ludism may yet emerge whereby technology is regarded more as a threat than as an enabler. This view has implications for the ways systems should be developed, with greater emphasis on software's life-enriching properties.

The traditional view of software as a product is rapidly changing to a view of software as a service. This new view implies new responsibilities for developers, particularly in light of the risk of software failure. As systems become more complex through distributed component use and reuse, component use will have to be assessed for risk, along with technical, organizational, and legal mechanisms for redress and recovery after system failure. Mechanisms will be required to ensure greater accountability in system development and evolution.

Software, like any other technology, can be used by the state (possibly as a tool of oppression) and by the

individual (sometimes to avoid state control). The only thing certain is that it will, in the future, play an important role in socio-political relations.

As systems become more complex, ensuring their accessibility becomes more important. Even relatively simple products, such as word processors, have sophisticated underlying models of structure and behavior that can be daunting for novice users. Software systems may therefore need adaptive interfaces that allow users to progress as they gain experience in the product.

Software has a major effect on the way its users behave in terms of the work they perform and the social context in which they perform it. While developers involve users in the specification of systems, they are less ready to involve them in the system's technical development, thus excluding the influence of social issues on design choices. In the future, there will be even greater need to allow social issues to influence and possibly drive the design process.

How software will be used. This scenario was addressed from the perspectives of who, what, when, where, and why. For example, it is no longer sufficient to think of users in such technology categorizations as, say, "naive" and "expert" or "frequent" and "infrequent." Many more classification axes are needed, possibly including:

- Economic grouping (such as low- and high-spending);
- Type (such as individual, family, and international);
- Use type (such as military, entertainment, social, and commercial);
- Use level (such as frequent and infrequent and minimal and full functionality); and
- Ability (such as level of expertise and physical limitations).

Each is important for different reasons, and the future use of software may depend on understanding not only the implications of each category but the interplay among them.

Software will be used when there is no other practical alternative (such as in large payroll/billing systems and high-risk locations) and when it is considered the most desirable solution (such as due to volume and safety concerns or as a means of displaying wealth or status). It will be used increasingly in entertainment, health care, security, finance, military affairs, e-commerce (with related far-reaching consequences for taxation), education, and travel. In addition, it will play an increasing human/humane role in various applications, as in those:

Figure 2. These hypotheses identify some of the ways software design, development, and use may change over the next 10 years.

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

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

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

There will be a change in attitude toward software development and business practice to improve acceptance and use of new technology.

There will be a change from an inability to predict service behavior to managing complexity.

- Dealing with customers (such as in banking, travel agencies, and medicine);
- Involving menial robots (such as for information gathering and domestic chores); and
- Building communities (such as for strengthening local communities and overcoming isolation)

The locations of the software, its execution, and its data will be flexible and, typically, transparent. For example, consider what would happen to the concepts of teleworking based on the following premises:

- Applications, bandwidth, and other functionality improve to such an extent that teleworking becomes viable, and face-to-face contact is no longer needed for most, if not all, high-value-add professions;
- There is a shakeout of bit-transporter bandwidth down to commodity pricing, so there is no pricing differential between data transport over local and long distances; and
- City and suburban quality of life degenerates in older centers of commerce and their commuter belts.

Such situations could lead to the export of wealth creation and, by implication, wealth itself away from the current centers of communication. For example, when stock market trading in the U.K. is removed from the trading floor and handled instead through electronic communication, there is no longer any real reason for the physical stock market and its office hinterland to exist. Traders can work from

anywhere in the country or even from anywhere in the world. Wealth creation could migrate to areas noted for a good quality of life at a more reasonable cost.

There are many ways of classifying software by its purpose, such as to:

- Automate and eliminate mundane tasks (such as word processing);
- Help transfer traditional skilled tasks to the less skilled (such as tax assessment and primary medical diagnosis);
- Support specialists performing tasks that would otherwise be difficult or impossible (such as process plant control, medical information systems, computer-aided drafting); and
- Improve technological infrastructure in order to promote innovation and learning (such as global communications, databases, operating systems, and data mining).

How software will behave. The behavior of software is being explored by DiSC team members through the hypotheses in the following paragraphs:

All interaction will be conducted through natural forms. The interface style and methods of interaction between software and users will continue to evolve until they employ the most natural forms (such as voice recognition and automated speech output). Software may anticipate users' needs, sparing them from having to explicitly interact with systems.

Software will meet necessary and sufficient requirements. Techniques for gaining a better understanding of essential requirements will lead to more targeted software systems that are not over-engineered with redundant functionality. Moreover, as requirements change, the underlying software will also change—seamlessly.

Software will be personalized. Software is currently marketed as generic products with little capacity for configuration or personalization. Future software will permit greater personalization, providing users with working environments best suited to their personal needs and working styles.

Software will be self-adapting. Software will include reflective processes that monitor and understand how it is being used and can identify and implement ways it can adjust to better meet user requirements, interface styles, and work patterns. It will also identify the need to commission new or changed software and decommission redundant software.

Software will be fine grained. Future software will be structured in small simple units that cooperate

Table 2. The four scenarios and related research topics.

Scenarios	Research Topics
Interactions between software and society	Empirical studies to improve problem understanding Economics of software Software acceptability Accountability User-oriented models of software behavior
Software use	User interaction and usability Psychology of technology-driven change Requirements capture, especially for good-enough and just-in-time systems E-commerce Distributed cooperative work
Software behavior	Self-fixing and self-adapting software Reflective software Good-enough and just-in-time software Personalized software Luxury vs. basic software
Software development	Multiskilled, geographically distributed development Componentry (reuse and recycling) Development and evolution models, including biological analogies Interdependence among design, business, and evaluation Agile software manufacture Empowering the domain expert (vs. maintaining integrity) Nonscripting development languages

through rich communication structures and information gathering. This structure will provide a high degree of resilience against failure in part of the software network while allowing the system to renegotiate to form new groupings.

Software will operate transparently. Software will continue to be viewed as a single abstract object even when distributed across different platforms and geographical locations. This property is essential if software is to be able to reconfigure itself and accept the substitution of one component or network of components for another.

How software will be developed. The future of software development was addressed from the perspectives of people, technology, process, and quality and standards.

We expect that most software (in a very general sense) will be developed by end users who may not realize that it is even software they are developing. They may, for example, be developing software in multiskilled teams while working on several sites simultaneously. Providing them with sufficiently powerful yet easy-to-use products will require a core

set of professionals of the highest ability, possibly working as multiskilled, multisite teams.

Dependence on core professionals may, however, result in a single supplier dominating the market. This situation could lead to vulnerability as a consequence of commercial monopoly, although the consequent evolution of standards may make it easier to integrate diverse applications.

Meanwhile, software will be increasingly component-based, that is, components will be customizable and flexible, rather than rigid. Powerful “glue” technology will be a key.

There will also be enormous potential for visualization and virtual reality technology to provide powerful assistance for the purpose of both software comprehension and domain exploration.

But developing large-scale systems will still be a problem. It may be that systems development will become a process within which needs are formu-

lated (desired behavior) and potential solutions (providing the desired behavior) are then selected from a large solution space.

Software evolution will also remain a major difficulty. Extending the needs-formulation-and-solution-selection idea, it may be possible to evolve software by learning from biological models, through which evolution of incredibly complex structures has been achieved.

Advances in technology will need to be incorporated into existing systems. The legacy problem is likely to get worse when software consists of diverse components obtained from many different sources through the Internet. Continual product churn and planned obsolescence may lead to a lack of confidence in the software industry—and a resulting lack of investment by potential users.

At one end of the development spectrum, software could simply become a by-product of organizational policy whereby business strategy determines overall business processes, and software is simply a mechanism for implementing some part of that business process. There will be an increasing interdependence

between software and business; with a good business model, it may be possible to generate the software directly, that is, automatically.

An intermediate view is that software development will concentrate increasingly on the design process, focusing on architecture. Populating this architecture could then be a dynamic process, obtaining components, connectors, and more, as needed. The movement to commercial off-the-shelf system components will add momentum to this approach.

At the other end of the development spectrum, the software development process may become completely informal. People will simply “code-up” through dynamic software creation right then and there, and test in an ad-hoc way.

Finally, it may be that too much professional software development will still be based on hype, and that more repeatable, empirical results backed by metrics will be required for decisions to be based on reason, rather than fashion.

Meanwhile, we expect consumers, users, and developers to keep seeking improved levels of software quality and standards. The relationship between suppliers and vendors may change as a consequence, and the marketplace may become more sophisticated and differentiated, as in, say, its ability to distinguish luxury software from ordinary software.

As a final step, the DiCE team has extracted a number of key research topics from these scenarios. Although the topics (see Table 2) were generated under the scenario headings, many have significance no matter what your view is of the future of software development and use.

Conclusion

The process DiCE finally adopted proved to be effective and enjoyable, in part because team members established good working relationships and relied on only lightweight management techniques to help organize their creative processes.

Although we used copyboards and post-it notes to considerable advantage during the brainstorming sessions, we didn't always capture adequately the material and ideas we were generating. It may be beneficial to instead make greater use of technology in the form of computer-supported cooperative work tools. Alternatively, it may be better to use a senior, experienced recorder with an appropriate technical background to keep track of the ideas being generated.

DiCE certainly succeeded in influencing BT's research profile. For example, it added new areas of strategic interest to BT's annual internal call for research proposals; one DiCE-led workshop, timed to coincide with this call for proposals, helped motivate

often heated debate within BT's own research community, yielding the positive result of aligning internal groups that had previously operated in isolation. The number of software technology and methodology proposals generated was approximately 50% greater than those delivered for BT's other research domains. Among the DiCE recommendations, two—“economics of software” and “society's understanding of software systems”—were accepted and are now formal projects within BT.

We also found that vision, especially collective vision, does not come in a flash but can be built incrementally if a suitable model of cooperative planning and development is established first. We encourage others to pursue a similar process. ■

REFERENCES

1. *Commun. ACM* 40, 2 (Feb. 1997). The Next 50 Years, special issue.
2. Korth H. and Silberschatz A. Database research faces the information explosion. *Commun. ACM* 40, 2 (Feb. 1997), 139–142
3. Leiner B., Cerf, V., Clark, D., Kahn, R., Kleinrock, L., Lynch, D., Postel, J., Roberts, L., and Wolff, S. The past and future history of the Internet. *Commun. ACM* 40, 2 (Feb. 1997), 102–108
4. Randell, B., Ringland, G., and Wulf, W., Eds. *Software 2000: A View of the Future*. Quote Reference No. P4265. ICL, Stevenage, U.K., 1994.
5. Shapiro, S. Splitting the difference: The historical necessity of synthesis in software engineering. *IEEE Ann. Hist. Comput.* 19, 1 (Jan.-Mar. 1997), 20–54.
6. Wulf, W. Look in the spaces for tomorrow's innovations. *Commun. ACM* 40, 2 (Feb. 1997), 109–111

PEARL BRERETON (P.Brereton@cs.keele.ac.uk.) is a senior lecturer in the Department of Computer Science at Keele University in Staffordshire, U.K.

DAVID BUDGEN (db@cs.keele.ac.uk.) is a professor of software engineering in the Department of Computer Science at Keele University in Staffordshire, U.K.

KEITH BENNETT (Keith.Bennett@durham.ac.uk.) is a professor of computer science in the Department of Computer Science at Durham University in the U.K.

MALCOLM MUNRO (Malcolm.Munro@durham.ac.uk.) is a professor of computer science in the Department of Computer Science at Durham University in the U.K.

PAUL LAYZELL (Paul.Layzell@umist.ac.uk.) is a professor of software management in the Department of Computation at UMIST in the U.K.

LINDA MACAULAY (lindam@co.umist.ac.uk.) is a senior lecturer in the Department of Computation at UMIST in the U.K.

DAVID GRIFFITHS (dave.g.griffiths@bt.com) is a software research program manager in the BT Group Technology in Adastral Park, Martlesham Heath, U.K.

CHARLES STANNETT (charles.stannett@bt.com) works in the BT Advanced Communications Technology Centre at Martlesham Heath, U.K.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
