IDEA GROUP PUBLISHING
701 E. Chocolate Avenue, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com
ITP4392

The Future of Software Development

Karen Church and Geoff te Braake
Department of Information Technology, Port Elizabeth Technikon, Private Bag X6011, Port Elizabeth, 6000
South Africa, Tel: 27-41-5043433, Fax: 27-41-5043313, E-Mail: kchurch@petech.ac.za

ABSTRACT

Software development has changed dramatically in the last fifty years and will continue to change. Its future course is of particular interest to developers, in order to gain the correct skills, and to any person faced with a strategic information technology (IT) decision. It is commonly accepted that computers will play an ever-larger role in modern civilisation. There are many unknowns, but the IT decisions made today will affect the competitiveness and preparedness for tomorrow. Awareness of the central issues that will affect the future of software development is the best form of preparation. This paper presents a view of the future of software development based on the history of software development and the results of two surveys.

INTRODUCTION

Software development tools and techniques have changed considerably in the last half century, are still changing, and will continue to change in the future as hardware capabilities improve and new technologies make new methods of processing and communication possible.

The aim of this paper is to draw conclusions about the future of software development from trends that can be identified in its evolution to date. The results of two surveys will help to illustrate some of these trends. The first was a questionnaire survey aimed at software developers which compared their First and Last Project in terms of a number of criteria. The second was a survey of job advertisements in the *Computing SA* newspaper over a ten year period.

This paper addresses the advancing generations of programming languages which have gained and lost popularity over the survey period. The evolution of coding styles and software architecture will be briefly described. The growing importance of user interfaces will be explained, in addition to a brief description of the increasing complexity of applications from user and developer perspectives. The final section will describe the future trends that can be projected from these points.

LANGUAGE GENERATIONS AND USAGE

The first applications of computers were to gain some form of military advantage based on doing many mathematical calculations very quickly (Arnold, 1991, pp.32-35). Computers then began to be used in business to speed up administrative tasks (Leveson, 1997, p.130). Online transaction processing and later, the personal computer, introduced a whole new dimension to computing by allowing people without programming training to use computers.

The challenge for software developers is to create programs that enhance the lives and work of those who use them. This section begins by describing the software development evolution. The development of programming language generations and their usage is addressed.

LANGUAGE GENERATION

In the early generations of programming languages, machine and assembly languages, the code was written at the level of machine instructions. Many statements were needed to accomplish simple calculations. Programs were long and errors were easily introduced, but difficult to identify and remove.

High level languages (HLLs) were developed to hide the details of implementation from the programmer. This is known as abstraction and is a common theme in the history of programming languages (Watson, 1989, pp.4-10). Each HLL command is translated into any number of machine instructions. HLL coding is shorter, and programs are easier and quicker to write and debug. The commands are fairly easy to learn and meaningful names can be given to variables and subprograms.

HLLs differ in the amount of abstraction that they provide. Visual Basic (VB) offers a higher level of abstraction than C++, as can be seen in Figure 1, in the operation to change the mouse pointer.

Visual Basic	C++

frmMain.MousePointer = vbHourglass	HCURSOR lhCursor;
	lhCursor = AfxGetApp()->LoadStandardCursor (IDC_WAIT);
	m_bCursor = TRUE;
	SetCursor(lhCursor);

Figure 1: Levels of abstraction in Visual Basic and C++

The higher the level of abstraction, fewer lines of code are required to achieve the same goal. Less code in the program makes it easier and quicker to write and debug. However, there is usually a performance penalty when the level of the language is higher. Flexibility is also decreased as the level of the language increases because the programmer has less control over the exact way in which the processing is done (McConnell, 1996, pp. 345-368).

Non-procedural languages take abstraction even further, with the programmer coding the desired result, not the method for achieving it. Historically, procedural languages have been the most commonly used type of language, as other language types were slower and more resource intensive. However, recent improvements in computer performance and language optimisation has meant that currently, there is a greater use of the other types of languages. The most widely used non-procedural language is SQL (McDermid, 1991, p.44/3; Kimball, 1996, pp.xxi-xxii; Watson, 1989, pp.79-81).

Table 1 shows a definite trend towards higher level languages with over three quarters of the Last Project being done using fourth generation languages (4GLs). This can be attributed to increasing

This paper appears in *Managing Information Technology in a Global Economy*, the proceedings of the Information Resources Management Association International Conference.

pressure to produce systems more efficiently together with the development of more powerful 4GLs (McConnell, 1996, pp.2,345).

Table 1: Language Generation by Project

Style	First Project	Last Project
3GL	57.1%	15.4%
4GL	35.7%	76.9%
Other	7.1%	7.7%

LANGUAGE USAGE

Whilst hundreds of programming languages have been created, relatively few have been widely used. The advert survey (results below) aimed to discover which languages have been used the most in software development since 1989. A number of general trends can be seen from Table 2.

Table 2: Most Sought-after Languages by Year

	1989	1990	1992	1993	1994	1995	1996	1997	1998	1999	2000
ASP	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.4	2.5	7.9
C	12.1	8.7	23.1	28.8	22.1	13.5	9.9	8.5	6.2	7.1	6.8
C++	0.0	0.0	4.6	8.0	19.4	18.5	16.2	11.7	12.9	16.2	14.7
COBOL	26.4	34.1	18.5	16.8	17.1	14.9	17.0	21.0	12.4	6.0	1.3
HTML	0.0	0.0	0.0	0.0	0.0	0.0	0.3	2.9	2.6	3.0	8.7
Java	0.0	0.0	0.0	0.0	0.0	0.0	0.6	3.2	4.7	7.0	12.3
JavaScript	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.4	1.1	3.1
Natural	12.1	19.2	15.7	12.8	14.4	8.6	8.8	9.8	9.4	3.3	0.5
RPG	18.3	19.2	21.3	15.2	6.8	9.6	5.4	5.6	6.0	4.6	0.8
SQL	2.9	0.9	7.4	7.2	5.0	10.2	6.5	4.0	9.4	10.2	11.0
Visual Basic	0.0	0.0	0.0	0.0	10.4	19.5	17.9	10.9	16.5	20.5	18.4

Figures = percentage of skills per newspaper issue.

The most sought-after language in the early 1990s was COBOL, followed by RPG, Natural, and C. By the year 2000, the main languages were C++, VB, SQL, and the Web languages Java, HTML, and ASP.

Thus, there has been a move to higher level languages. There has been a shift away from some long established languages with the new computing environment dominated by the graphical user interface and the Web in particular. The next section will describe the evolution of coding styles.

CODING STYLE EVOLUTION

As seen in the previous section, languages have become more powerful and have raised their level of abstraction. The programming language chosen for development may either encourage or discourage certain programming practices depending on their features. This section highlights some of these coding styles.

Structured programming became the most popular programming style in the 1970s. It popularised the concept of modular programming (Yaeger, 1995, p.2). A source of many problems with structured programming was that variables could be inadvertently changed resulting in errors.

Since the late 1980s object-orientation has gained popularity amongst software developers. Object-orientation makes use of classes which encapsulate data and functions into a single unit. Object-orientation is an important paradigm for contemporary system developers and is supported in many widely used languages, such as C++ and Java (Salus, 1998, pp.5-11).

Graphical development languages (e.g. Visual Basic, Delphi) popularised the concept of component-based programming. Components should be built with standard interfaces so that they can be reused by other applications and any other language or tool that supports the interface method. This is widely used in Internet development. (Jacobson *et al.*, 1997, pp.85,156).

It is common to build components using the object oriented style just as object orientation makes use of structured concepts. Thus the different styles can be seen as an evolution of better coding practices which aim to increase productivity, maintainability, reusability, and readability of code while decreasing the number of errors, and time required for coding and maintenance.

Table 3: Coding Style by Project

Style	First Project	Last Project
Component	7.1%	23.1%
Object-oriented	28.6%	61.5%
Structured	50.0%	15.4%

The coding style used by the respondents in the First and the Last Project shows considerable difference (Table 3). In the First Project the structured style was the most common, followed by the object-oriented style. Structured programming decreased considerably in the Last Project. Object orientation was the clear leader in the Last Project, followed by the component style.

The majority of object-oriented and component-based development is done using 4GLs . Projects developed using the structured style, however, mainly use 3GLs.

There has been an evolution in programming styles to promote modularisation, data hiding, and reuse. This allows systems to be developed more quickly, to have better quality and to be easier to maintain. Software architecture has also changed considerably resulting in different development opportunities and challenges which will be discussed in the next section.

SOFTWARE ARCHITECTURE

The previous sections showed that using modern coding styles can help developers to produce and maintain systems more efficiently. These styles have been supported by different languages in different eras. The evolution in software architecture is as a result of the changing capabilities of hardware, and increasingly distributed and integrated systems.

Most early data processing applications were isolated subsystems. Each application used its own flat data files. Online transaction processing increased the number of records in files and required random access to records. However, as the number of records in files increased, inconsistencies in data and accessing of records became major problems. Therefore a more integrated solution was sought and a number of database models were developed.

The network model was the first de facto database model in the late 1960s and early 1970s. The databases were, however, dependent on the application development language and many vendors produced incompatibile products (Fortier, 1997, pp.187-188). The relational model, proposed in 1970, was independent of the application development language using the database and many applications could access the same database (Deen, 1985, p.77). This meant that the organisation was not bound to a particular language for development (Hughes, 1988, p.4-5). The relational model has become popular due to the simplicity of database structure, the flexibility of relationships, and the richness of data manipulation (Fortier, 1997, pp.207,244).

In the mid-1980s Local Area Networks (LANs) were becoming popular and each department in a company installed its own LAN and developed its own departmental client/server applications. This resulted in redundant and inconsistent data within an organisation. In the early 1990s the development of enterprise client/server IT systems which replaced or augmented legacy main-

frame systems and integrated departmental LANs, allowed companies to deliver the right information when and where it could best be used (Goldman, Rawles & Mariga, 1999, p.19).

These systems began using the three-tier application architecture (Figure 2), which is also the architecture of the Web. Applications are divided into three layers or tiers known as P-A-D, presentation, application and data. Each tier can be handled by different computers and developed in different languages. Not only can the layers of the application be split onto different computers, but each layer, especially the application and data layers, may also be split over multiple computers making it scalable. System maintenance and modification is facilitated by allowing changes to one tier or component without affecting the others (Edwards, 1999, pp.3-11). Providing the client with a Web interface greatly simplifies distribution and platform problems.

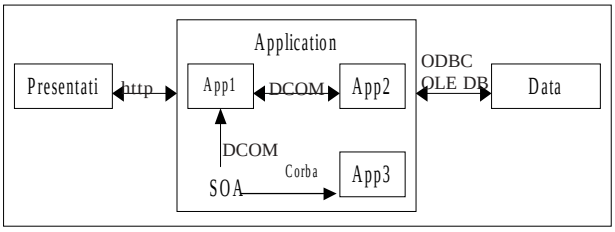

Figure 2: Three-tier Web architecture

Thus software architecture has moved from a single unit on a mainframe computer to distributed data, application and presentation tiers. Data has moved from multiple, inconsistent data sources to single integrated databases. In dealing with large systems, such as the many enterprise scale systems presently being created, it is desirable to have an architecture that allows units to be worked on simulataneously and independently (Jacobson, 1997, p.171). An area of software development that has become very important in recent years, is the user interface, which is discussed in the next section.

USER INTERFACES

IT systems are commonly developed for access in a distributed environment, giving non-IT people access to information resources and data processing power. This makes the user interface particularly important in development. Changes in common user interfaces are described below.

The first few decades of computing focused on performance and functionality of applications. When millions of people began using productivity tools, it became apparent that a primary determinant of the success of an application was its ease of use for users of all levels of experience (Van Dam, 1997, p.64).

From the early 1960s through the mid-1980s text-based user interfaces were used almost exclusively. The WIMP GUI (graphical user interfaces based on windows, icons, menus and a pointing device), first began to gain popularity with the Macintosh in 1984 and later achieved its current dominance with Windows. When this event-driven paradigm was introduced it was difficult for developers to produce this type of application with the available tools. The Windows environment returned programmers to working in ways reminiscent of low-level programmers. A tool was needed to increase the level of abstraction to allow efficient Windows programming. Therefore languages such as Visual Basic and Delphi were developed to build GUI applications efficiently (Cornell, 1997, pp.xix).

Half of the First Projects reported in the questionnaire survey were text-based. Text-based systems development virtually disappeared in the Last Project whilst Web interfaces show the biggest gains, even though they are relatively new (Table 4).

Table 4: User Interface

Style	First Project	Last Project
GUI	35.7%	53.8%
Text-based	50.0%	0.0%
Web-based	14.3%	46.2%

Thus the user interface is one of the most important aspects of IT systems, especially as they are becoming more complex from a number of perspectives, which are discussed below.

GROWING APPLICATION COMPLEXITY

The user interface is one of the primary factors determining the success of a system. Applications are becoming more powerful but also more complex for developers to produce. This complexity arises from increasing integration with other systems and utilising the growing power of computers to produce better information. These trends are discussed in terms of groupware, multimedia, multiple language development, and team work.

GROUPWARE

Groupware is a relatively new set of technologies that allows for easier communication and collaborative work by means of a computer network. The Web is a very good medium for deploying groupware technologies, but needs to have enhanced security to make it viable (Goldman *et al.*, 1999, pp.177-178, 217).

MULTIMEDIA

Multimedia provides a richer experience of the application for the user. This has become possible because of increased hardware capability. The Internet provides a container for presenting rich multimedia as well as providing the means of co-ordinating its distribution. Multimedia development tools have developed rapidly due to industry focus on the Web and its mass usage (Nicol *et al.*, 1999, p.79).

An increasingly important feature of the software industry is gaming. Games tend to tax computer system resources to the maximum, making it imperative that developers access sound and graphics capabilities at low levels to increase the speed of performance. Graphics and sound are combined to create more real experiences. The simulation effects are becoming so realistic that games have large inventories of the objects and environments that are simulated. Some games require some level of artificial intelligence. Therefore, games development is driving new technologies, many of which will have applications in marketing, education and other areas (Tapscott, 1999; Walnum, 1995, pp.6-11,70-71).

Multiple Language Development

It is evident from the advert survey referred to earlier that multiple technologies for a single project is not a new phenomenon. Many adverts for COBOL programmers included required skills in CICS and some database management system. In 2000 (Table 2) SQL was the fourth most sought-after skill. In Web development there are client side scripting and markup languages and application logic languages (Edwards, 1999, pp.3-11). Therefore, multiple language development is the rule, rather than the exception.

The questionnaire demonstrated that over 90% of the last

projects (Table 5) were developed with multiple languages. This was particularly true of Web-based projects. It was less common in GUIs and the minority of text-based systems (Table 6). The component paradigm gives the possibility of being able to create a system built from components developed in the best language for the task. The components are connected using an interface protocol, the most common being COM/DCOM and CORBA (Finne, Leijen, Meijer & Jones, 1999).

Table 5: Number of Languages by Project

Style	Multi-language	Single Language
First Project	35.7%	64.3%
Last Project	92.3%	7.7%

Table 6: Number of Languages by Interface

Style	Multi-language	Single Language
GUI	66.7%	33.3%
Text	28.6%	71.4%
Web	87.5%	12.5%

TEAM WORK

A team can be defined as a group of people whose complementary skills, common purpose and approach enable them to complete a task for which they are mutually accountable. Team work has always been important, especially now with multiple languages and having to deal with the intricacies of networks and other technologies. This range of skills can only be provided by teams (Jacobson *et al.*, 1997, p.54). Table 7 shows that the percentage of projects on which developers worked as a team, as opposed to doing the project on their own, rose from 78.6% to 100%. Thus while team work has been important in IT development in the past, it is has now become vital.

Table 7: Team work by Project

Style	First Project	Last Project
Team member	78.6%	100.0%
Working solo	21.4%	0.0%

Thus applications are becoming more complex, both in terms of functionality offered and consequently in their development. Now that some of the important factors of the past and the present of IT systems development have been discussed, some thoughts on the future are presented.

THE FUTURE

With the rapid rate of change in the IT field it is very difficult for developers to see what the future trends might be. After analysing the past changes and current situation the following points are suggested as likely directions for the future of software development in the short term.

The trend of moving to higher level languages is sure to continue in the effort to produce quality systems efficiently. Hardware advances make the processing overheads incurred by these languages less significant.

There needs to be some consolidation in Web development and there are likely to be numerous tools and languages developed that attempt to do this. One technology that may prove important is Microsoft's ASP+ Web Forms, which will allow the development of Web applications in a similar way to Visual Basic. The ease with which these developers can produce complete Web applications and the increasing usage of ASP (see Table 2) makes this a technology to watch in the coming months. (Microsoft, 2000a; Microsoft, 2000b).

Java has progressed from experimental to implemented systems faster than any language except VB. Considering its rise in popularity and wide usage and successive releases to remedy the slowness in execution, Java can be expected to remain a mainstream programming language for some time to come (Berst, 2000a; Babcock, 2000).

Developers will experiment with other types of languages. Non-conventional languages may be used to produce specific components or applications in the areas for which the language is intended..

Object-orientation appears to remain dominant, but the component paradigm is likely to gain ground, especially with the importance of the Web.

The Web is likely to play a role in most systems development projects, especially as XML is developed to allow for more powerful applications. A specific example is the Simple Object Access Protocol (SOAP), a protocol that could provide the interface between virtually any two systems as long as they support both hypertext transfer protocol (HTTP) and XML. (Skonnard, 2000).

An emerging area of software development is that of mobile devices. The second generation of mobile phones, using digital networks, were introduced in the early 1990s and experienced exponential growth in numbers of users and services associated with them. The next generation of mobile telecommunications will include many more wireless data services (Väänänen-Vainio-Mattila & Ruuska, 1999, pp.24-25). It will be an important area of software development. It will create new requirements and limitations while still providing a rich multimedia experience for an even less computer literate audience than the Internet.

Thus the future of IT systems development will have increasingly stronger tools that allow developers to produce systems that address ever more complex functionality, thereby building applications that will enhance the user's productivity, not restrict it. As the tools become more powerful more of the technical correctness will be supplied by the tool, but more creativity will be required of the developer to adapt to and to use new technologies to produce better IT systems.

CONCLUSION

IT systems play a vital role in modern civilisation. There is virtually no industry that does not use some form of computerisation and many are totally dependent on computers to control their operations. Software development will change unrecognisably in the future, as it has in the past, and it is not possible to predict how with any certainty (Leveson, 1997, p.129). This paper attempts to present the future of software development, in terms of the factors from its past.

Programming languages have seen to be continually raising the level of abstraction, hiding the details of implementation from developers. This allows them to focus their efforts on achieving the best solution, rather than how to do it. The languages used have changed with the type of the majority of applications that are developed. Currently, as well as in the near future that means the most widely used languages will be visual development languages that produce Web applications. Coding styles have evolved methods for making programs easier and quicker to develop and to maintain by building them out of units which can be changed independently and reused in many systems. The independent units include the splitting of the application into data, logic and presentation tiers with interfacing protocols to make applications flexible

and scalable. The user interface has become an increasingly important part of applications as they become more powerful and are used by people of all levels of experience to improve their efficiency.

Thus this paper has drawn some conclusions about the future of software development in order for current developers to make themselves better prepared to meet the challenges that lie ahead.

REFERENCES

Arnold, D. O. (1991). Computers and Society Impact!. New York: N.Y. : McGraw-Hill.

Babcock, C. (2000). Java: Can Sun control the flood? Inter@ctive Week [Online]. [cited 3 July 2000] URL http://www.zdnet.com/enterprise/stories/main/0,10228,2581701,00.html

Berst, J. (2000a). Scott McNealy's Java Jive [Online]. [cited 3 July 2000]. URL http://www.zdnet.com/anchordesk/stories/story/0,10738,2582432,00.html

Cornell, G. (1997). Visual Basic 5 from the Group Up. Berkeley, Ca. : Osborne/McGraw-Hill.

Deen, S. M. (1985). Principles and Practice of Database Systems. Hampshire : Macmillan.

Edwards, J. (1999). 3-Tier Client/Server at Work (Revised ed.). New York : John Wiley & Sons.

Finne S., Leijen, D., Meijer E. & Jones S.P. H/Direct: A Binary Foreign Language Interface for Haskell. ACM SIGPLAN NOTICES, 1999, Vol. 34, No. 1, pp. 153-162.

Fortier, P. J. (1997). Database Systems Handbook. New York : McGraw-Hill.

Goldman, J. E., Rawles, P. T. & Mariga, J. R. (1999). Client/Server Information Systems. New York : John Wiley & Sons.

Hughes, J. G. (1988). Database Technology A Software Engineering Approach. New York : Prentice Hall.

Jacobson, I., Griss, M. & Jonsson, P. (1997). Software Reuse. Reading, Ma. : Addison Wesley Longman.

Kimball, R. (1996). The Data Warehouse Toolkit. New York : John Wiley & Sons, Inc.

Leveson, N. G. (1997). Software Engineering: Stretching the Limits of Complexity. Communications of the ACM, February 1997, Vol.40, pp.129-131.

McConnell, S. (1996). Rapid Development. Redmond, Washington : Microsoft Press.

McDermid, J. (1991). Software Engineer's Reference Book. Oxford : Butterworth-Heinemann.

Microsoft (2000a). Microsoft Primes Millions of Developers for the Next-Generation Web [WWW Document]. [cited 3 July 2000]. URL http://www.microsoft.com/presspass/press/2000/feb00/nextgenerationpr.asp

Microsoft (2000b). Visual Studio Enables the Programmable Web [WWW Docuement]. [cited 3 July 2000] URL http://msdn.microsoft.com/vstudio/nextgen/technology/Webforms.asp

Nicol, J.R., Getfreund, Y.S., Paschetto, J., Rush, K.S. & Martin, C. (1999). How the Internet Helps Build Collaborative Multimedia Applications. Communications of the ACM January 1999 Vol.42 No.1, pp. 79-85.

Salus, P. H. (1998). Handbook of Programming Languages Vol. 1. : Macmillan Technical Publishing

Skonnard, A. (2000). SOAP: The Simple Object Access Protocol. [WWW Document]. [cited 1 August 2000]. Microsoft Internet Developer, January 2000.

Tapscott, D. (1999). The Power of Electronic Play. Computer World, May 24 1999, p.32.

Väänänen-Vainio-Mattila, K. & Ruuska, S. Designing Mobile Phones and Communicators for consumers' Needs at Nokia. Interactions, September 1999, pp.23-26.

Watson, D. (1989). High-Level Languages and Their Compilers. Wokingham, England : Addison Wesley.

Walnum, C. (1995). Windows 95 Games SDK Strategy Guide. Indianapolis, In. : Que.

Yaeger, J. (1995). Programming in RPG/400 2nd ed. Loveland, Co. : Duke Press.