Methods and Tools for Developing Interactive Information Systems: an Historical Perspective

Anthony I. Wasserman
Carnegie Mellon West
Moffett Field, CA 94035 USA
tonyw@west.cmu.edu

Abstract

This paper describes the evolution of hardware and software technology over the past three decades, focusing on approaches for building interactive information systems and web applications. Successive generations of technology have used advances in hardware and software technology, along with increasingly sophisticated development methods and tools, to reduce development times and to produce a better user experience. After describing the key technology characteristics of each generation, this paper also describes the evolution of a specific methodology, User Software Engineering, from its origins in the 1970’s to its applicability to the development of modern applications.

1 Introduction

The processes and tools for developing interactive information systems (IIS) have changed drastically over the past three decades. Users of these systems have moved from slow, text-based systems in the 1970’s to today’s high-speed Web-based applications. Methods for developing these systems have evolved from the phased waterfall approach of the 1970’s to more agile approaches commonly followed today. Similarly, the tools for creating these systems have evolved rapidly, reflecting changes in hardware technology, user interfaces, development notations and languages, and mechanisms for collaboration among members of a team.

Looking back over the history of hardware and software technology, it is helpful to identify different generations of this technology that had a major impact on the processes used for building an IIS and the nature of the resulting system.
2 First Generation: Mainframes, Batch Processing, and Files

In the 1960’s, most computing was batch-oriented, with few interactive systems. Graphical displays were available for some highly-specialized applications, such as radar tracking. Early database management systems used a hierarchical or network oriented view of data, and were deployed on large mainframe systems. Terminal devices were slow (300 baud) and resembled teletypes, displaying text only in uppercase, and printing on rolls of paper. The slow speed severely limited the amount of interaction. Among the first interactive applications were airline reservation systems, financial applications, and military applications, such as radar tracking. These systems, in general, used file systems rather than a DBMS to store information, reflecting the need to obtain the best possible performance from the limited capabilities of the hardware. The first interactive programming environments, such as BASIC, also were developed at that time, making use of the early time-shared operating systems on minicomputers.

Development techniques for interactive systems were in their infancy, since there was almost no experience in building them. There were no analysis and design methods, no database modeling methods, or any of the other techniques that are now common. Modeling was at the procedural level, e.g., flowcharts, and at the logical level for files and data. In summary, the various hardware and software pieces for building an IIS weren’t yet in place. For software professionals about to retire, the first generation reflects the situation at the beginning of their careers.

3 Second Generation: Time-Sharing, Terminals, and Early DBMSs

The widespread presence of time-shared operating systems and alphanumeric video terminals transformed computing in the 1970’s. Furthermore, this technology was available on minicomputers, such as those made by Digital Equipment Corporation, making it practical for almost every organization to obtain one. Primitive networks were in place to provide remote access to computing, including low-speed dialup.

The user interfaces were still text-based, but gradually moved from being line-oriented (glass teletype) to screen-oriented. While the first windowing systems were being developed in research settings, they were not generally available. Database systems had matured, but they remained
very expensive. The first relational database management systems and the
SQL query language were also introduced as research projects in the
1970’s. The first analysis and design methods also emerged in the 1970’s,
including Parnas’ principles of modularity [1], Structured Design [2,3],
Structured Analysis [4], and many more. Data modeling methods also
appeared, including Chen’s Entity-Relational Modeling [5] and Bubenko’s
Inferential Abstract Modeling [6], though these were still primarily of
interest to the research community and not to the average IIS developer.

Nonetheless, all of the pieces were in place for the earliest interactive
information systems. Bank tellers used terminals to process user
transactions. Travel agents installed terminals for an airline reservation
system and learned the arcane text commands for checking schedules and
booking reservations. Similarly, companies began using interactive
systems for managing their business, including accounting and order
processing. While many of these systems were very primitive by today’s
standards, they were the vanguard of the transition to interactive
computing.

4 The Origins of User Software Engineering

The User Software Engineering (USE) methodology was conceived
during this period (1975-1980) [7]. USE added the user perspective to
function and data design for overall system design. At that time, systems
were frequently imposed on users, and users rarely participated in
definition and design of a system. Systems were architected in a top-down
or bottom-up manner.

However, users have little interest in the structure of the system; their
only concern is whether the system makes it easier for them to get their job
done. What was needed for an IIS was not top-down design, but rather
outside-in design, where "outside" represents the user's perception of the
system. Outside-in modeling is one of the most important concepts of the
USE methodology. By emphasizing the external view of a system, it
became much easier to communicate with users.

Each user interaction with the system was viewed as an event that
could trigger an activity and/or a response. For example, the system could
display a menu of choices to the user, with the user’s input determining the
program action, eventually leading to either program termination or
another request for user input.

The user interaction and the system behavior was modeled as a
hierarchical set of transition diagrams. This approach had two major
benefits. First, even on paper, it was possible to walk through a dialogue with a potential user of a system, validating the overall scenario. Second, and more significant, transition diagrams are a formal, executable model, making it possible to build an executable version of the emerging system.

The executable nature of transition diagrams led to the most important innovation of the USE methodology: rapid prototyping of user interfaces. The state transition diagrams, including specification of the user inputs and system outputs, were encoded in a transition diagram language TDL. This language also included the ability to specify executable program units. A tool, RAPID/USE, was built to interpret the TDL and execute the associated program units. In this way, RAPID/USE could be used both for prototyping the user interface and for running a complete program.

In this way, users could begin to work with the emerging system at a very early stage of development, to the extent that they could actively contribute to the definition of the system and the style of the user interface. This notion of user involvement in the software development process is, in many respects, the central idea of the USE methodology. The RAPID/USE system also gathered metrics on user behavior, making it possible to track error conditions, task completion times, and other measures of system usability.

In general, developers would start by designing part of the user interface, implementing it, and adding functions or pseudo-functions as place holders, as well as beginning design of the relational database model. This incremental approach to application development was a sharp contrast to the waterfall approach in widespread use at the time. This approach is very much in line with today’s agile methodologies [8].

The ability to separate the user interface component from the program operations led to another significant concept of the USE methodology: a three-tier architecture. This notion was very similar to the Model-View-Controller (MVC) approach first presented by Reenskaug [9].

![Figure 1 – The three-tier architecture of the USE methodology (from [10])](image-url)
This architecture shows the separation between the various components of the information system, anticipating client-server systems of the early 1990’s and modern n-tier architectures. In addition, it shows the possibility of associating multiple user interfaces with a set of system operations. Such an approach permitted separate interface designs for novice and expert users, as well as an application programming interface that could be used to drive test cases or to integrate the system with another system.

5 Third Generation: Personal Computers and WIMP Interfaces

The third generation focused on three dramatic changes in computing systems: graphical user interfaces (GUIs), powerful personal computers, and computer networking. Rudimentary GUIs were built for Pong, PacMan, Spacewar and other arcade games, but GUIs were not generally available for general purpose systems until the release of the Apollo and Sun workstations in 1982 and the Apple Macintosh™, with its integrated windowing system, in 1984. With the Macintosh, the user interface shifted from the alphanumeric screen of video terminals to a bitmapped display using an interface style known as WIMP (windows, icons, menus, pointer). The GEM GUI provided a similar interface for PC DOS about a year later, and supported color as well. Microsoft Windows was announced about the same time, but it was not until Windows 3.1 was released in 1992 that PC users had a widely used GUI.

The growth of personal desktop computing created a revolutionary change in applications. Previously, users had worked with “dumb” terminals, with all of the computing being done on a remote machine. Suddenly, the personal computer could run applications locally or emulate a terminal running a remote program. From a software perspective, the WIMP approach came with a set of libraries, application programming interfaces (APIs), and user interface guidelines for building applications using each windowing system, with the result that there was a great deal of commonality among GUIs on a particular computing platform. Many platform vendors offered incentives to software vendors to comply with the GUI guidelines, and users came to expect applications to comply with platform standards. By the early 1990’s, new applications were being built with GUI interfaces, though many older enterprise applications still used alphanumeric interfaces.
In the same period, DBMSs became widely available, including DBASE for the PC, and various relational DBMSs for other platforms. With the widespread adoption of Unix, C became an important programming language for IIS applications, with C-based APIs for the windowing systems and the DBMSs.

These advances led to significant changes in methods and tools for IIS development. Instead of a text-based interaction with a file system, developers now wrote C code to build GUI interfaces on personal computers and to connect their application to local and remote RDBMSs. The MVC architecture and the WIMP interface were well-suited to an object-oriented programming paradigm, for which the C++ language was widely adopted.

With these technology advances, the USE tools were no longer appropriate for prototyping of user interfaces. The WIMP interface could not be easily modeled with the hierarchy of state transition diagrams that worked for text-based systems, since there would be far too many states and transitions.

However, the process advocated in the USE methodology remained valid: use a succession of prototypes of the user interface to elicit user requirements and to implement increasing functionality of the IIS. From there, various user inputs could be associated with program actions that included the database operations for the various IIS functions. The emergence of standard mechanisms for database access and for GUI management led to common architectures for IIS development, making the development process more efficient and the resulting system more robust.

The USE methodology could be used with advanced 4GL systems, such as PowerBuilder (now owned by Sybase), a rapid application development system including screen design and database access mechanisms. PowerBuilder, as with many other 4GL systems, was designed to be used in a client-server environment, providing the GUI interface on the user’s local machine and network connectivity to a remote RDBMS.

In the area of software methods, structured approaches were being supplanted by object-oriented approaches, using hybrid notations such as OMT [11] and OOSD [12], pure OO notations such as Booch’s clouds [12], and the use case approach of Jacobson [13]. The OO approach and the use case notation worked very well for the design of an IIS, since there is a straightforward mapping between these logical concepts and the implementation.
Another key development of this generation was the growth of open source software, paralleling the development of the commercial software industry. Among the software packages released in source format were Berkeley Unix, the Ingres RDBMS, the USE rapid prototyping tools, and the GNU software (Emacs, GCC, etc.). Many of these free and open source components were suitable for inclusion in other packages, thereby reducing the time and effort needed to build new applications.

6 Fourth Generation: The World Wide Web and Multi-tier systems

The next major advance in technology for IIS was driven by the invention of the World Wide Web (including the HTTP protocol, the HTML language, and URLs) by Tim Berners-Lee and his colleagues at CERN in 1989, followed by the development and release of the Mosaic graphical Web browser, developed by Marc Andreesen and Eric Bina at the US National Center for Supercomputing Applications (NCSA), and released in late 1993. Andreesen and Bina commercialized their work at Netscape Communications, releasing the Netscape Navigator browser in 1994, a key milestone in the evolution of the Internet from a research tool to a ubiquitous resource for networked communications and services.

The Web brought forth a new generation of user interfaces and a new way of thinking about applications and their development. There was a vast investment in new Web-based businesses, virtually all of which had the characteristics of an IIS, using a Web browser as the “local” front end connected to a remote server that implemented program behavior and accessed one or more databases and other remote services. The mediation between the web browser and the server-side application code was the Common Gateway Interface (CGI), a standard way for an HTTP server to pass a user request to an application program and to receive data back to forward to the user.

For existing client-server systems, the advent of the Web provided an opportunity to replace a platform-specific GUI with a browser-based GUI as a way to modernize their systems, keeping much of the server-side infrastructure in place. However, the vast majority of Web applications were written from scratch, and therefore had no need to accommodate older technologies. What was most important for many of the new startup businesses was to get their website up and running as quickly as possible, as a way to gain a competitive advantage on potential competitors and to generate revenue for their business. In an earlier paper [14], we showed how our original implementation to the CRIS Conference Management System example [15] could be modified to present a Web interface.
Interactive web systems could be divided into those that needed support for a large number of concurrent interactions and those that didn’t. For the latter category, new scripting languages, including Perl, Python, and PHP, emerged. Clicking on an image map or on a web form “Submit” button could be associated with a server-side action implemented in one of these (or other) languages. These actions often included access to remote files and databases, with retrieved data formatted into HTML and passed through the CGI for display in the user’s browser. (Java Server Pages with Java servlets on the server side were another option for implementing the actions.)

High volume and critical systems, such as those for e-commerce, travel, and financial applications, needed a more robust and secure infrastructure, including server-side load balancing, firewalls, routers, and support for transactions. While these systems were logically three-tier systems, they were, in practice, N-tier systems, replicating the web servers and application servers, as well as adding clustered databases, content delivery systems, media streaming servers, and/or online payment gateways. Many of these applications were built using what is now called Java Platform, Enterprise Edition (Java EE, formerly J2EE), which provides built-in support for access to shared resources and many other features needed for creating these sophisticated web applications.

Over time, standard approaches for building such systems have emerged. For example, Sun Microsystems, which has controlled the Java standard until recently, has led the creation of a set of development guidelines, patterns, and blueprints for building Java EE applications [16].

As Web applications have grown to predominate older client-server applications, the term “Web application” has effectively supplanted the term “interactive information system”, even though it is slightly narrower in some respects.

Design and development of Web applications presents some new technical challenges, which have been addressed by a broad variety of new tools. As with older applications, the skills needed for design of the GUI are quite different from those needed to implement the functionality of the system. Over the past decade, tools for design of the Web interface have evolved from text-based HTML editors to WYSIWYG design tools (e.g., NVU and FrontPage) to web site design tools capable of applying a template, cascading style sheets, and links to CGI actions (e.g., Dreamweaver). Developers without graphical design experience can select from a vast number of prebuilt site templates, making it quite straightforward to create a prototype of the user interface and to test it with
several different web browsers on popular platforms (Windows, MacOS, Linux).

The design process for the conceptual data model, now often called “information architecture”, has remained relatively unchanged over the years. The ideas pioneered by Chen, Bubenko, and others not only remain in widespread use, but have also been incorporated into numerous other modeling notations, such as the Unified Modeling Language (UML) [17].

The widespread use of scripting languages for Web applications is very helpful for rapid prototyping and for the application of agile methodologies. One can iteratively define the database schema, adding new tables and columns to the schema as needed, and writing the scripts to connect the actions specified in the HTML code to the appropriate computational and database modification actions. For example, one could start developing a Web application by initially creating the page(s) needed for a user to register, login, and/or request a lost password.

7 Fifth Generation: Richer User Interface, Application Generation Tools, and Open Source

Among the primary difficulties in building Web applications are building usable Web user interfaces and making the application robust, scalable, and secure. Web application performance is critical to the success of a site, since users quickly grow impatient with slow response times. Poor performance may be caused by any or all of the following:

- large images or streaming media transferred to the user’s browser;
- the nature of the HTTP protocol, which requires a separate call to the server for each item to be displayed in the browser
- inefficient database design or coding
- high traffic volume for the web site or a service used by the application, as well as high overall Internet traffic

While this list is not complete, it serves to illustrate the complexity of building Web applications and the importance of the user interface design. Over the past few years, there has been a great deal of effort expended on overcoming these problem areas. We discuss three of these efforts in this section.
7.1 AJAX (Asynchronous JavaScript and XML)

The technologies comprising AJAX are intended to reduce the amount of data traffic between the server and the browser by avoiding the need to reload an entire web page each time the user requests a change, with the result being a richer user experience. Many “Web 2.0” websites, such as Gmail, Basecamp, and Flickr, use this approach, which also facilitates the development and use of Web applications offered as a service, such as the Zoho Office Suite and productivity tools. There are more than 100 toolkits to aid in the development of these interfaces, including toolkits developed by IBM (Dojo), Yahoo, Google, and numerous startups.

Use of such a toolkit is certainly a requirement for rapid prototyping of user interfaces, since hand coding of the Ajax interface can be extremely time-consuming and error-prone. Since the primary goal of prototyping is to gain a better understanding of the user requirements, it is difficult to justify additional time and effort on GUI design at the earliest stages of a project, and it is often better to defer tuning the GUI until the use cases and functional requirements have been determined.

7.2 Application Generation Tools

Automatic generation of applications is a longstanding goal of the software development community, and many development tools (e.g., Software through Pictures [18]) and 4GLs (e.g., PowerBuilder) have aimed to generate part of all of an application from high level or visual development tools. The Object Management Group has been the primary sponsor for the Model Driven Architecture project, which allows specification of the high-level architecture of a system independent of its implementation technology. Many companies have built products intended to generate an architecture from such a specification [19].

Automatic generation of a Web application is not easy, as it requires generation of the user interface, the database schema, and the application logic in a way that links the separate pieces to one another. However, a vast body of knowledge about construction of Web applications has emerged, and there are now a variety of approaches. For many common types of Web applications, there are now website builder tools (e.g., CityMax and Caspio Bridge) with which one can quickly create a customized site. Similarly, Wiki tools (e.g., MediaWiki) and content management systems (e.g., Drupal) can be used to rapidly build specialized applications. Zou and Zhang [20] have described a framework for automatic generation of e-commerce web applications.
The Rails framework [21], built for the Ruby language [22], follows a different approach. Its creators describe Rails as “a full-stack framework for developing database-backed web applications according to the Model-View-Control pattern.” When you create a new application with Rails, it generates the application framework automatically, using a database schema that you provide. It generates Ruby code to produce a basic web-based interface. Powerful Web applications, such as the Basecamp collaborative project management tool (http://www.basecamphq.com), have been developed with Rails.

For the CRIS conference management application, one could start by defining the mailing list table (using the syntax for the MySQL RDBMS) as follows:

```sql
CREATE table mailing_list
(
    name varchar (30) not null primary key,
    affiliation varchar (63),
    detail_address varchar (255),
    postcode varchar (15),
    city varchar (30),
    country varchar (30)
);
```

The generation process creates a web-based form with the fields linked to the columns of the table, with the needed HTML code being generated by the Rails framework. Showing the use of Rails for the entire conference management system is beyond the scope of this paper; a significant amount of hand coding in Ruby is needed above and beyond the parts that can be generated automatically. Nonetheless, Ruby on Rails goes a long way toward automated generation of a Web application.

### 7.3 Open Source Software

Free, libre, and open source software (FLOSS) products have been widely used for many years. Among the most prominent of these open source products are the GNU Emacs text editor, the GNU Compiler Collection, Berkeley Unix, the Apache web server, the Eclipse development framework, the MySQL and PostgreSQL DBMSs, the JBoss Java EE application server, various distributions of Linux, and scripting languages, including Perl, Python, PHP, and Ruby. Today, there are
hundreds of widely used open source products and thousands of open source projects.

FLOSS products, both commercial and non-commercial, have made significant inroads into companies to support their software infrastructure. The Apache HTTP server, for example, is used for more than 60% of all websites. Beyond that, many FLOSS components have been integrated into leading software products. For example, SugarCRM is used with the Apache HTTP server, MySQL or PostgreSQL, and PHP. In addition, many of the most heavily used websites, including Google and Yahoo, are built on FLOSS technologies, and the Windows operating system includes licensed open source components.

More than 60% of Java developers now use the Eclipse development environment, causing vendors of traditional closed source environments, such as BEA and Borland, to link their development tools to Eclipse. As FLOSS products have grown in use, they have found their way into virtually every organization, either intentionally or accidently.

Early FLOSS projects were focused on development tools, small components, and middleware. In many cases, the project would create a FLOSS replacement for a piece of commercial software, matching the external specifications so that the open version could serve as a replacement for the closed one. More recently, there has been a growth in FLOSS applications, such as SugarCRM and OrangeHRM, that build upon an open source infrastructure.

Rather than designing and writing the CRIS conference management application from scratch, one now has the option of using an open source conference management system. The SourceForge repository (http://www.sf.net) contains more than 140,000 open source projects (of varying quality and maturity). Among them is the IAPR COMMENCE Conference and Meeting Management System [23]. COMMENCE uses an HTTP web server, along with MySQL and PHP. Since those components were already running on my server, it was possible to download COMMENCE, install it in the HTTP server directory, run the setup script to create the database, and immediately use the application. All of the software was freely available and the COMMENCE system comes very close to meeting the original requirements for the CRIS system. A sample page of the running COMMENCE application is shown in Figure 2.
The availability of this existing and freely usable open source project completely transforms the application development process. This unchanged version of the COMMENCE application can serve as a working prototype for the desired application, and the source code can be modified to improve the user interface, as well as to address missing requirements and remove unwanted functionality from the original version. The total time required to find and install the application was less than an hour. Methodologies such as User Software Engineering and the Rational Unified Process \[24\] become essentially irrelevant in this setting.

Open source software is transforming the software industry and shifting the traditional build vs. buy decision process. There are a growing number of open source alternatives to traditional closed source applications and tools. As their numbers grow, developers will increasingly look toward these open source packages as key components for systems that they are designing and developing. This growing inventory of high quality FLOSS components may have a bigger impact on Web application development than any other aspect.

8 Conclusion

The nature of interactive systems has changed drastically over the lifetime of today’s computer science community. Severe limitations in storage capacity and processing speed are rarely an issue today. The nature of the user interface is much more intuitive, though also much more complex. The body of existing tools provides much more powerful application development capabilities than was previously. Furthermore, the presence of high quality existing applications and application frameworks means that fewer new applications need to be built from
scratch, but rather by reusing and modifying existing software. Such an approach implies a revolutionary change in system development methodologies. For a method such as User Software Engineering, the key concepts remain valid, but may be less relevant as new development builds on existing code.

Future applications will certainly take advantage of new technologies in user interface development. Work is already well underway in developing web applications for mobile devices, addressing the challenge of providing a good user experience with the limited display area, restricted input mechanisms, and relatively low bandwidth of current mobile devices. Among the approaches for overcoming these restrictions are voice-based applications, which remain quite limited in scope, but which could eventually overcome many of the current difficulties in using web applications.

Other advances are harder to predict, but continuing change is a certainty. It would have been nearly impossible to predict today’s Web applications from the information systems of the 1960’s described in Section 2. Forty years from now, future generations of information systems specialists will likely be using technologies that we cannot easily imagine.

References


