Design and Development of an Electronic Performance Enhancement Tool for Creating and Maintaining Information Management Web Sites

Todd Hamilton Bowden

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in
Curriculum and Instruction
(Instructional Design and Technology)

March 2, 2011
Blacksburg, Virginia

Keywords: Electronic Performance Support Systems, Web Design, Programming, metaprogramming, Databases, Cold Fusion, Dreamweaver, Web Forms, FADBIT

Design and Development of an Electronic Performance Enhancement Tool for Creating and Maintaining Information Management Web Sites

Todd Hamilton Bowden

ABSTRACT

This study explored the design and development of an electronic performance enhancement tool that can assist a person with limited programming skills to create a variety of simple customized information management websites. In particular, this study was modeled after needs within an Instruction Technology department in which individuals were able to create pre-functional web pages with various elements such as textboxes and dropdown menus but lacked the programming skills necessary to add functionality to these web forms. Skilled programmers could add functionality to these pre-functioning web forms or create customized information management websites from scratch. However, programmers are not always available when needed. At the time of this study, there was no readily available way for persons to create customized information management websites without the services of a programmer or without needing to learn programming skills themselves. This study sought to determine what functionalities, characteristics and capabilities could be included in an electronic performance enhancement tool to assist non-programmers to create simple customized information management websites and how a tool with such functionalities, characteristics and capabilities could be designed and developed. A prototype version of such tool (named the Form And DataBase Interaction Tool or "FADBIT") was designed and developed in this study. This tool asks users who have created simple pre-functional web forms to answer a series of questions related to those webforms. Given the user's responses to these questions, this tool is able to form a metalanguage representation of the user's intentions for the web form and can translate this representation into useful programming code to add the desired functionality. The tool was successfully designed and developed using a generalized modular framework, and a Create-Adapt-Generalize model, with each module addressing one or more patterns common to web programming. The prototype tool successfully allowed non-programmers to create functional information websites for two structured evaluation projects, and achieved some level of success and encountered some difficulties with an unstructured project. Proposed modifications and extensions to the tool to address the difficulties encountered are presented.

Acknowledgements

I would like to thank my advisor Dr. Ken Potter for his care and guidance during this study. I am particularly indebted to him for introducing me to the Create-Adapt-Generalize paradigm.

I would also like to thank the members of my committee, Dr. Barbara Lockee, Dr. Jamie Little, and Dr. Todd Ogle, for their advice and support during this study.

I would give special thanks to the evaluators of this project for their time and effort. Privacy rules prevent me from listing their names, and this is the closest I can come to a public acknowledgement of their efforts.

Finally, I would like to thank my Mom and Dad for their love, encouragement, and patience during this endeavor.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Problem Addressed by this Study

The problem addressed by this study may be stated as follows. Individuals often need customized information management websites. Programmers can create customized information management websites but programmers are not always available when needed. Persons with a need for an information management website could potentially learn programming skills themselves but may lack the inclination or aptitude to learn programming skills in a reasonable length of time. There is currently no readily available way for persons to create customized information management websites without the services of a programmer or without needing to learn any programming skills themselves.

Information management websites that make use of electronic databases have the potential to be of value in a wide array of educational settings. Within an Instructional Technology department, as one example, there are a variety of people with information management needs. These include faculty and staff who wish to manage information related to students and classes, such as online and distance learning classes. Others need to perform information management tasks such as collecting and reporting data in order to conduct research. In some cases, existing commercially available software is available to meet those needs. However, there are cases in which the need for specific characteristics and capabilities are not met by any readily available software. In these cases a person with sufficient programming skills can create customized information management websites using broadly available electronic database software and programming languages designed to create dynamic web pages for standardized Internet web browsers such as Firefox, Safari or Internet Explorer.

However, often the persons needing such information management websites lack the necessary programming skills to create them. In these cases, instructional technologists with skills in areas such as programming, database design and web page design are sometimes called upon to create customized information management websites for others. As a researcher and computer programmer in the Instructional Technology program at Virginia Tech this has often been the author’s role. In that role, the author has been asked to numerous meetings in which other persons -- professors, clients and graduate students -- described, in abstract terms, a set of information management tasks they would like the computer to perform. The author’s responsibility was to take their descriptions and translate them into computer programs that would perform those tasks. This process of translation, from abstract idea to specific detailed steps is the job of a programmer.

While programmers can create customized information management websites when needed, programmers are not always available, or the time they can devote to a particular project is sometimes limited. In those cases, ideally, persons needing custom information management websites would be able to create such sites themselves with the assistance of a tool that would perform some of the same functions of a programmer.

Purpose of this Study

The purpose of this developmental dissertation was to explore the possibility that an electronic performance enhancement tool would be able to assist persons with limited programming skills to create a variety of simple customized information management websites that would normally require programming skill, by designing, developing, and testing a prototype version of such a tool.

Relationship of this Study to Instructional Design and Technology

The Association for Educational Communications and Technology (AECT) sets forth a set of standards for instructional technology programs divided into five categories: Design, Development, Utilization, Management, and Evaluation. The Management category includes a subcategory specifically on information management (AECT, 2000). This study addresses the information management component of the AECT standards.

Within the field of Instructional Technology, Information Management "… involves planning, monitoring, and controlling the storage, transfer, or processing of information in order to provide resources for learning" (Seels & Richey, 1994, p. 51). A variety of systems exist for information management, with various names, such as *Content Management Systems*, *Record Management Systems*, *Library Management Systems* and *Learning Management Systems* (Robinson, 2004). While these systems are designed to meet specific needs, all share one or more common characteristics. These systems provide a means to store, retrieve, modify, organize, display, search for, delete, or control access to information.

The Need for a Tool

There is a need for a tool that can reduce dependence on programmers by generating the code required to produce customized information management websites. While there is a *prima facie* case for a tool that can reduce the dependence on programmers by assisting persons without traditional programming skills to create customized information management websites, that need is further demonstrated by empirical evidence the author has gathered from working with others with information management needs in their roles within an instructional technology program. Three projects involved graduate students with information management needs related to their research, a fourth involved evaluation and reporting requirements for the Virginia Department of

Education, and the fifth is an online masters degree program offered by the instructional technology program at Virginia Tech called the Instructional Technology Master of Arts or ITMA program. The three graduate student researchers first explored suitable tools for their informational management needs, and were unable, at the time of their research, to find suitable tools with all of the characteristics and capabilities required. The other two clients, the Virginia Department of Education and ITMA, had ongoing programming needs that could not be met by their current personnel.

Larson Dissertation

The first of these projects was an online survey conducted by Miriam Larson, a graduate researcher at Virginia Tech, as part of the research for her dissertation entitled "Survey and Case Study Analyses of the Professional Preparation of Instructional Design and Technology (IDT) Graduates for Different Career Environments" (Larson, 2004). Larson had several requirements for her survey. She wanted to be able conduct the survey online, over the Internet using web pages and forms. A web page that displays fill-in text boxes, selectable drop-down lists or combo-boxes, selectable buttons, radio-buttons, and other standardized forms is usually referred to as a *web form*. Larson wanted control of the layout, fonts, and colors of each web form (page). She wanted the survey to "branch" at various points to one or more alternate sets of questions depending on how a respondent answered questions on the current page. She also wanted to access the survey responses online. Lastly she wanted the data to be stored in a secure location under the control of people she trusted. As indicated above, Larson was unable to find any adaptable software that had the characteristic and capabilities she required.

Kobayashi Dissertation

In a somewhat similar project, Michiko Kobayashi, a second graduate researcher at Virginia Tech, also needed to conduct an online survey as part of her dissertation "Facilitating Academic Achievement in High School Interactive Television Programs by Promoting Self-Regulated Learning" (Kobayashi, 2007). In addition to the requirements of Larsen listed above, Kobayashi had several additional requirements. She wanted survey participants to identify themselves with a username and password before they could access the survey. Her survey also had multiple parts, each of which had to be repeated once per week for several weeks. She wanted participants, after gaining access to the survey, to select both the part of the survey they wanted to view and the appropriate week from a menu, a feature similar but not identical to the "branching" feature required by Larson. Additionally, for one of the survey pages, Kobayashi wanted participants to view some of their previously entered responses. Once again, Kobayashi was unable to find any existing software that would allow her to create such a survey.

Yilmaz Dissertation

A third researcher, Harun Yilmaz, also required an online survey for his dissertation, "Identification of Academic Program Strengths and Weaknesses through Use of a Prototype Systematic Tool" (Yilmaz, 2007). Like the previous projects, Yilmaz wanted to conduct a survey online, control the layout, fonts, and colors of each web form, access the responses online, have the data stored in a secure location and require participants to identify themselves with a username and password before accessing the survey. Additionally, Yilmaz needed the participants to respond to various parts of the survey by selecting from a list of likely answers. He also needed the ability to create these lists himself and to allow participants to add to the lists if the response they wished to give was not among the existing answers. Yilmaz also needed a feature that allowed him to create customized reports of the data collected, which could be sorted and filtered on specific values. Once again Yilmaz was unable to find any existing software that would allow him create such a survey.

Virginia Department of Education Survey

In another project the Virginia Department of Education needed to collect data from various educational consortia in Virginia for evaluation and reporting purposes. Like the previous projects, the DOE wanted to conduct the survey online, control the layout of the page, access the responses online, store the data in a secure location and require a username and password to access the survey. In addition to the features that were required in the dissertation projects above, the DOE had some additional requirements. One requirement was a navigation menu that would appear on every survey page and would allow the user to choose any other survey page to view, a feature similar but not identical to the "branching" feature mentioned above. Another very significant requirement not present in the surveys listed above was that previous survey responses needed to be editable by the user. That is to say, when a user viewed a survey page, the various text boxes and other elements on the survey page needed to be pre-filled with any previously entered responses, which could then be edited and re-saved.

Instructional Technology Master of Arts (ITMA) Website

Another project with extensive information management needs is the Instructional Technology Master of Arts or "ITMA" program. The ITMA program is an online Masters Degree program offered through Virginia Tech. The website for the ITMA program has three main parts. One section is an interface allowing students to access course material, submit assignments, and review grades and feedback from instructors. Another section of the website allows instructors to review, and provide feedback for students. A third section is for administration of students, courses and the underlying database. Like the previous projects, ITMA needed to be available online, access to each of the three separate sections needed to be controlled by a username and password, and various pages needed navigation or "branching" capabilities. In the student section of the website, students needed to be able to select assignments from a menu, submit completed assignments for grading, and later view instructor feedback. In the grader section, individual graders needed to be able to view lists of submitted assignments, select an assignment to grade, and enter or edit feedback and numerical scores for those assignments. The administration section needed features such as creating or editing student profiles, adding or editing courses, enrolling students in courses and other general database administration.

The Tool

The empirical evidence presented above of five representative projects within an instructional technology program shows that there were information management needs that, at the time the projects were undertaken, required the services of a programmer. From the perspective of the graduate students and employees, there also was a need for a tool capable of addressing their information management needs without the services of a programmer. Such a tool would assist persons who consider themselves “non-programmers” to create similar information management websites. The purpose of such a tool is to reduce the dependence on programmers within an Instructional Technology department.

The empirical evidence suggests tool should permit sufficiently skilled persons to use the tool themselves to create their own information management websites. Secondly, the tool should facilitate the distribution of work within an instructional technology department. The tool, while making the creation of information management websites easier, will still require some entry skills for the users. Persons who have, or are willing to develop, those skills can use the tool to create information management websites, thus reducing the dependence on the programmers.

Research Questions

This study was designed to answer the following research questions:

1. What functionalities, characteristics and capabilities could be included in an electronic performance enhancement tool that would allow persons with limited programming skills to create simple customized information management websites that would normally require programming skills?
2. How could an electronic performance enhancement tool with such functionalities, characteristics and capabilities be designed and developed?

Overview of Remaining Chapters

The remaining parts of this study are outlined as follows. The second chapter identifies and reviews areas of relevant literature with the purpose of identifying functionalities, characteristics and capabilities that could be included in an electronic performance enhancement tool that would allow persons with limited programming skills to create simple customized information management websites. The third chapter examines the methodology used in creating a prototype of an electronic performance enhancement tool with the features identified in the literature review and from empirical evidence. The fourth chapter describes the results of the study and discusses the construction and evaluation of the prototype. The final chapter discusses the results as well as the contributions of this study to the field of Instructional Design and Technology.

Chapter 2: Literature and Background

The previous chapter introduced empirical evidence of a need for a tool to assist non-programmers to create information management websites that would normally require programming skill. This chapter examines whether this need aligns with needs identified in the literature as well as what the literature has to say about the design and development of such a tool. Although no mention was found in the literature that explicitly discussed the creation of a tool to assist non-programmers in creating information management websites, several areas of literature address similar (though not identical) needs and provide insight into the general features, characteristics and capabilities that would need to be present in a tool to assist non-programmers, as well as design and development approaches for creating such a tool. One area of literature to look for the identification of similar needs is the literature on Electronic Performance Support Systems.

Electronic Performance Support Systems

Electronic Performance Support Systems (EPSS) have some similarities as well as some important differences with a tool to assist non-programmers. The review presented here does not encompass the entire field of EPSS. The EPSS literature is quite large and only partially relevant to the current topic. Instead, this review is narrowly focused on identifying specific characteristics and capabilities of an EPSS that would be relevant to a tool to assist non-programmers in creating information management websites.

The Electronic Performance Support System was originally envisioned as a system that would enable persons to be more productive without prior knowledge or training (Gery, 1989, 1991). Although there are numerous definitions of Electronic Performance Support Systems (Cagiltay, 2001) the best fit for this study is the approach and definition offered by Gery (1989) who defines an Electronic Performance Support System as “An integrated electronic environment that is available to and easily accessible by each employee and is structured to provide immediate, individualized on-line access to the full range of information, software, guidance, advice and assistance, data, images, tools, and assessment and monitoring systems to permit job performance with minimal support and intervention by others” (p. 21).

The goal of an EPSS is to provide whatever support is needed to generate performance and learning at the moment of need. Gery (1991) contends that this kind of support has always required human beings in the past, but the technology now exists to provide that type of support with electronic systems. Gery also delineates several types of software that might be components in an EPSS. These include advisory or expert systems, productivity software to assist with specific tasks, application software that performs specific tasks, help systems, interactive training sequences, assessment systems and monitoring, and assessment and feedback.

Raybould (1990) described the development of an EPSS for sales and support personnel that included expert systems, databases, interactive video discs, formatting information online, information retrieval techniques, HyperCard, computer-based training, simulations, and user interface. Carr (1992) proffered artificial intelligence as a potential component of an EPSS.

Sleight (1993) described an Electronic Performance Support System as having the following characteristics: it is computer-based, it provides access to the discrete, specific information and tools needed to perform a task at the time the task is to be performed, it is used on the job, or in simulations or other practice of the job, it is controlled by the user, and reduces the need for prior training in order to accomplish the task.

Miller (1996) described an Electronic Performance Support System as "…any computer software program or component that improves employee performance by either: reducing the complexity or number of steps required to perform a task (process simplification), providing the performance information an employee needs to perform a task, or providing a decision support system that enables an employee to identify the action that is appropriate for a particular set of conditions." Raybould (1995) described an EPSS as a system that would "…enable an individual to achieve a required level of performance in the fastest possible time and with the minimum of support from other people" (p. 66). Barker and van Schaik (2010) broadly presented an EPSS as a system for enhancing human ability and overcoming human limitations.

The arrival of the EPSS also coincided with significant developments in instructional technology, among these being the performance technology movement. As a result of this movement, many instructional designers began conducting more careful analyses of the causes of performance problems, and oftentimes discovered that poor training, or lack of training, was not the cause. In many such instances, instructional designers prescribed non-instructional solutions, such as changes in incentive systems or in the work environment, to solve such problems (Dean, 1995). Thus the types of activities many instructional designers engaged in greatly expanded.

Reiser (2002) noted "…the relatively new performance technology movement, with its emphasis on front-end analysis, on-the-job performance, business results, and non-instructional solutions to performance problems, was beginning to have an effect on instructional design practices (Rosenberg, 1988,1990; Rossett, 1990)" (p. 62). It is in this context that the EPSS emerged. As noted by Cagiltay (2001), it is agreed among EPSS experts than an EPSS, while fitting within the broad field of Instructional Design and Technology, is different from instructional systems in that its main focus is on performing rather than learning.

The literature identifies five important characteristics or capabilities of an EPSS that are also relevant to a tool to assist non-programmers. An EPSS includes productivity software to perform specific tasks (Gery, 1991). It reduces the need for prior training in order to accomplish the task (Sleight, 1993). It reduces the complexity or number of steps required to perform a task (Miller, 1996). It allows users to achieve a required level of performance in the fastest possible time (Raybould, 1995). It is a non-instructional solution to performance improvement (Dean, 1995). In keeping with the performance technology niche within Instructional Design and Technology, the focus is on performing, not learning (Cagiltay, 2001).

With regard to the tool described in this study, these five characteristics or capabilities would be: productivity software to generate computer code, reducing the need for prior training in programming skills, reducing the complexity or number of steps needed to generate the necessary code for functional webpages, allowing users to achieve the required performance (creating simple websites that work) in faster time, and a focus on the required performance, not learning programming skills.

Although some characteristics of an EPSS relate directly to the needed tool, empirical evidence suggests the tool does not currently exist or is not readily available. This implies that if a tool is to be created that will incorporate the above characteristics or capabilities, that tool must be designed, developed and programmed. It is therefore worthwhile to review the field of programming itself, the difficulties and shortcomings associated with mainstream programming, and then the field of *metaprogramming* -- the writing of computer programs that write other computer programs.

Programming and Metaprogramming

The job of the mainstream programmer is to create a step-by-step set of instructions to complete some task or solve some problem. Mainstream programming can be summarized in three steps: First, given a task to program, a programmer forms a conceptual mental model of how to solve the problem. Second, a general-purpose language is chosen for writing the solution. Third, the program is written by performing the sometimes difficult mapping of the conceptual model into the programming language. The programming (mapping, translation) step is the "bottleneck" because it is not easy or natural for humans to perform this translation for complex programs (Dmitreiv, 2005).

A serious problem with mainstream programming is that there is a very long gap between the time when it is known exactly how to solve a problem and the difficult process of successfully communicating this solution to the computer as a program. It is possible to explain the problem and solution to another programmer in a matter of hours, but the process of translating this solution into a programming language takes much longer. When explaining a program to another programmer, one can just express very high-level ideas in natural language, but for the computer, every single step must be expressed in detail. In mainstream programming, most of the time spent "programming" is really just finding ways to translate concepts expressed in natural language to equivalent terms in a programming language, something "difficult, not very creative, and more or less a waste of time" (Dmitreiv, 2005, p. 2).

When there is a problem to solve, the programmer thinks of a solution represented in words, notions, concepts, and thoughts. This is the programmer's mental model of how to solve the problem. The programmer does not think in terms of a set of instructions, but instead of a set of "inter-related concepts". For example, in the case of a graphical user interface the programmer would think "*I want this button to go here, this field to go here, and this combo-box should have a list of some data in it.*" This mental construct might even be just a picture in his mind, without any words at all.

This mental construct can be explained to another programmer with sufficient specificity that the programmer could write a program which will solve the problem. It is not necessary to explain the solution in terms of a programming language. For example, if a mental model is a graphical user interface or web form, it would be sufficient, in communicating to another programmer, just to draw the form, and explain what each element on the form does.

Given the difficulty of the translation process, a critical question inevitably arises: is it possible for the computer itself to do some of this translation? Is it possible for the users/programmers to express the task they want the computer to perform in a more abstract form, and then allow the computer to perform the translation from this more abstract representation to a traditional programming language? This approach to programming is refered to in the literature as metaprogramming.

Metaprogramming

Metaprogramming is the writing of computer programs that write or manipulate other computer programs. Rideau and Ban (2001) state that metaprogramming is "the art of programming programs that read, transform, or write other programs" (p. 2). Cordy and Shukla (1992) describe metaprogramming as "[T]he process of specifying generic software source templates from which classes of software components, or parts thereof, can be automatically instantiated to produce new software components" (p. 215). Metaprogramming is also described as the "automatic selection and assembly of components on demand" (Czarnecki & Eisenecker, 1999). Lowe and Noga (2002) offered a more abstract definition: “Metaprogramming interprets a source program as data to be analyzed and transformed” (p. 106).

As outlined by Rideau and Ban (2001), in the early days of computers, a computer program was generally small enough and simple enough that a single person could understand all the elements of that program, down to the smallest details. Since that time, the size and complexity of computer programs have increased substantially. According to Brooks (1995), however great the progress made by individual programmers, the complexity level of computer programs has long since exceeded the level where any one person can conceive of the entirety of a program in all of its details that would also take full advantage of the capabilities of existing computers. In order to effectively make use of more capable computer systems, it has become necessary to make the programming process increasingly abstract so that the more routine and conceptually redundant tasks can be automated by the computer, thus letting the programmer focus on the “essence of programming” (Rideau & Ban, 2001, p. 1), that is, on higher level problems.

The concept of metaprogramming therefore is to be able tell the computer at a more abstract level what the computer should do at an equal or less abstract level. All metaprogramming is relative to some less abstract representation of the same program. All general purpose high-level programming languages, such as C, C++, Java, PHP, ASP, Cold Fusion, *etc.*, are metaprogramming languages relative to assembly code, in that they allow the programmer to give instructions to the computer at a more abstract level than the assembly code (also called machine code or language) into which those programs are ultimately translated.

Another example of metaprogramming involves code generating programs (Bartlett, 2005) that create or generate code in language such as C++ or Cold Fusion from some more abstract representation. The process of metaprogramming may be iterative and, when it is iterative, there may be many levels of metaprogramming. An abstract expression may be translated by a code generating program to C++, and the C++ code is then translated in turn into assembly code.

A theoretical type of metaprogramming would begin at an even more abstract level, taking a precise, human language description of what a computer should do, translating that description into a more familiar high-level programming language, which could, in turn be translated into assembly code. No current computer program can perform the translation required for this theoretical type of metaprogramming. Today, this type of translation, from natural human language to high-level computer code, is the function of a computer programmer. The programmer takes a human language description of a problem given by another human, and translates it into a programming language, which can in turn be translated into machine language. This approach is illustrated in Figure 1.

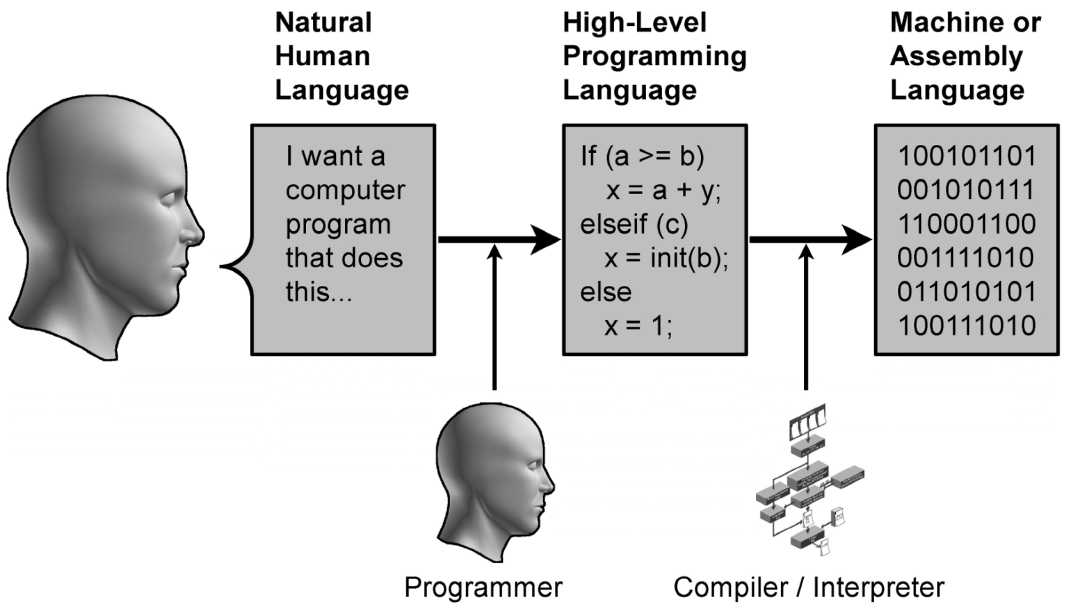


Figure 1. Traditional Programming

There are three important elements associated with the metaprogramming concept. While there are some differences in terminology used in the literature, in this review these elements will be referred to as the metaprogram, the object program and the translation program. Each of these elements has its own role. The first of these elements is the metaprogram, is written in the metalanguage. The role of the metalanguage is to provide some means of precisely specifying what the computer should do (Dmitreiv, 2005). Many languages can potentially be metalanguages. High-level programming languages are metalanguages in relation to machine code. Natural human language can be a metalanguage in relation to a high level programming language. There are also code generating programs (Bartlett, 2005) that can also be more abstract than a high-level programming language but more formal than natural human language. A metalanguage is any language in which one can precisely specify what the computer should do.

The next important element in metaprogramming is the object program, also sometimes called the target program (Trujillo, Azanza, & Diaz, 2007). The object program is written in the object language. The object language is less abstract than the metalanguage. The object language for a compiler or interpreter is machine code. The object language can also be high level programming language such as C, C++, Java, PHP, ASP, Cold Fusion, etc. The role of the object language is also to precisely specify what the computer should do. But the object language, either is, or is closer to, the machine code that a computer can understand.

The last of the important elements is translation, also called mapping. (Rideau & Ban, 2001) Translation takes the description of what the computer should do, as expressed in the metalanguage and generates an equivalent expression of what the computer should do in the object language, thus transforming the metaprogram into an object program (Visser, 2002).

In many programming projects, there may be multiple stages of translation. A compiler or interpreter is an example of a translator. A code-generating program that transforms a description of what a computer should do from a metalanguage into a more traditional high-level language (C, C++, Java, PHP, ASP, Cold Fusion, etc.) is an example of a translator. In mainstream programming the human mind, in particular the mind of a programmer, serves as the translator. Metaprogramming can be iterative where a metaprogram is translated into an object program, and that object program then becomes a metaprogram and is itself translated into yet another object program at a even lower level of abstraction.

Metaprogramming has several benefits. One benefit is that metaprogramming can enhance programming efficiency and assist in the creation of programs that share generic characteristics (Chen, 2005). Another benefit is that metaprogramming has the potential to expand the class of people who could serve as programmers. Currently, the job of a programmer requires the ability to perform the difficult, time consuming and error prone translation of a conceptual model into a programming language. Not everyone can do this or do it well. However, if the program can be expressed in a more abstract form that is closer to the problem being solved, the number of people who would be able to program the computer is potentially much greater. As the level of abstraction allowed by the metalanguage increases, the number of people who can program also increases. Very few programmers can effectively write in assembly language. More can write in a higher-level language such as C or Java. More still could make use of code generators, while even more could use some future theoretical development based on natural human language. In whatever form, the ability to generate programming code is one of the benefits of metaprogramming (Smaragdakis, Huang, & Zook, 2004).

Metaprogramming also has its downsides. One downside is there may be a limited complexity for the metaprograms (Koskinen, 2004). The number of things a computer can be directed to do when using an abstract metalanguage may be limited because it may not be known how to translate the abstract expression into a more traditional programming language. It is one thing for a programmer to translate an abstract idea into instructions that the computer can "understand" -- that is the job definition of a programmer. It is another thing to understand so well how that translation is performed that it is possible to write an automated system (another program) that will do the translation. Another limiting factor is that high-level abstract expressions often contain ambiguities and unspecified details that, at a more concrete level, must be decided. Assumptions can be built into the translation process to deal with ambiguities, but assumptions themselves are limitations. Currently one of the roles of programmers, in their jobs of translating an idea into a precise description of what a computer should do into traditional high-level programming language, is to make assumptions if the more abstract descriptions are not precise enough or to make choices when there are ambiguities.

In order to create a tool that can translate a metalanguage into an object language, that tool itself must be designed and developed. But how is this done? For an answer to this question, it is therefore useful to review literature related to the software development process as well as instructional design and development models that could be used to guide the creation of such a tool.

Design & Development Approaches

The *software development process*, also referred to as the *software life cycle*, *software engineering process* or *software process model* is a structure or methodology used to guide the development of a software product. The software development process typically divides the development of software into a series of stages. The function of a particular software development process is to specify these stages, the order of these stages, and the criteria for how to progress from one stage to another stage (Boehm, 1988). Various models for the software development process have been proposed in the published literature. Two prominent software development models are reviewed below and are compared to two other design and development approaches, the ADDIE approach and the Create-Adapt-Generalize model (Hicks, Potter, Snider & Holmes, 2004).

Waterfall Model

The waterfall model shown in Figure 2 is a software development model that sees software development occurring in a series of sequential stages that can be seen as "flowing" downward from one step to the next like a waterfall. W.W. Royce introduced this model in a paper published in 1970 entitled "Managing the Development of Large Software Systems" (Royce, 1970). Royce began by introducing two stages essential to any computer program: analysis followed by coding. He then expanded those stages into a more "grandiose approach" (Royce, 1970, p. 328) with preceding and succeeding stages. The stages in Royce's original paper are system requirements, software requirements, analysis, program design, coding, testing and operations.

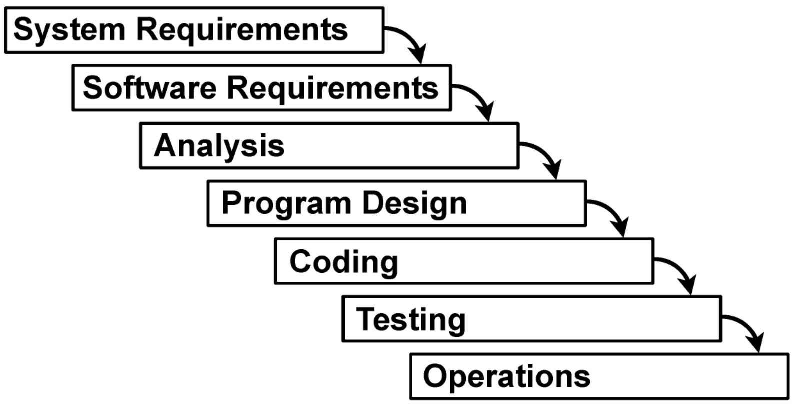


Figure 2. Waterfall Model

In a *pure* waterfall model the process proceeds from one step to the next in a sequential manner. First the requirements are completed, then the analysis is completed, then the design is completed, then the coding is completed, and then the testing is completed. The movement from one stage to the next is done only when the preceding stage is perfected. There is no jumping back and forth between stages. It follows, for example, that once the requirements are completed, they are "set in stone" and do not change.

To illustrate, suppose the requirements for a particular program are "to display the contents of a database table in a matrix format in a web browser." Once it is decided that the preceding description describes all of the requirements of the program, those requirements cannot change at a later point in the software life cycle. Thus, if it is discovered later in the development process that an edit record feature or a feature to filter the results is desirable, that feature could not be added because it was not specified in the requirements up front.

Having introduced this model, Royce later describes it as a model that is "risky and invites failure" and then proceeds to offer some alternative approaches. Various modifications of the waterfall model to address its shortcomings have been offered by others. Chief among these is the iterative development model and its variations.

Iterative Development

Iterative development, also called incremental development, presented by Basili and Turner (1975) was developed to address the weaknesses of the more traditional waterfall model. Boehm (1988) offered a variation on iterative development called spiral development. The idea of iterative development is to create software incrementally, learning during each iteration, both from the issues encountered during the development itself and from the use of the software. One approach in iterative development is to start by creating a working version of the software with only a subset of the requirements, and with each iteration, add more features until all of the requirements are implemented. If additional needs are discovered as the software is developed and used, modifications can also be made to the requirements with each iteration, Basili and Turner (1975) describe the process as beginning "…with a simple initial implementation of a properly chosen (skeletal) subproject which is followed by the gradual enhancement of successive implementations in order to build the full implementation" (p. 390).

At least one iterative approach, as seen in Figure 3, starts by creating a list of needed features of the program and then creating a basic version of the program with a subset of those features with which users can work and evaluate. Based on that evaluation, another iteration is done, and more features from the list are added. The features added at each iteration should be relativity simple and modular. An evaluation based, at least in part, on user feedback is also done with each iteration. At the evaluation stage, the list of needed features itself can be modified or added to, accommodating an evolution of the program in response to needs that become apparent during the development.

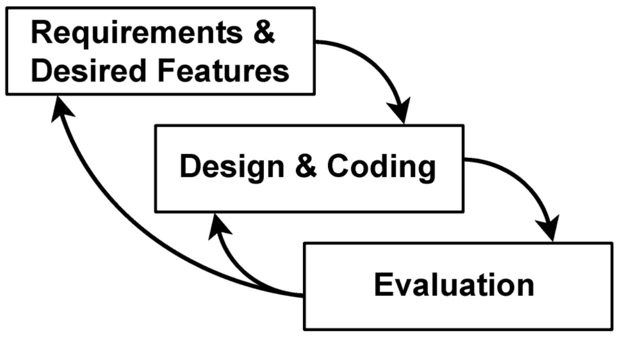
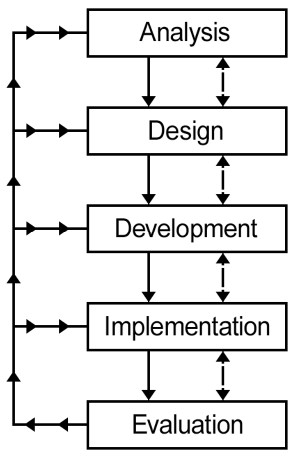


Figure 3. Iterative Model

In the example above, the iterative process might begin with list of features that include retrieving data from a database and displaying it in table form to the user. The first iteration might include retrieving the data and displaying it in some form that would verify it was successfully retrieved. The next iteration might include outputting the data in a well formatted table structure with appropriate fonts and color codes. In the evaluation stage of the iteration process, based on user feedback, the list of required features might be expanded to include an edit data option, a feature to filter the data, and a feature to authenticate the user. Each of these features can be added during successive iterations.

ADDIE

The ADDIE approach trancends the fields of programming and instructional design and corresponds with common phases in software engineering (Jonassen, 2004). ADDIE is an acronym for Analysis, Design, Development, Implementation and Evaluation. ADDIE does not have a single author, but instead evolved via oral tradition (Molenda, 2003). The phases of ADDIE follow from its name. The needs to be addressed in a particular project are considered in the analysis phase. A blueprint for a solution to the identified needs is created in the design phase. The product specified by the design is constructed in the development stage. The actual use of the product by its intended audience is referred to as the implementation phase (Visscher-Voerman & Gustafson, 2004). Finally, the resulting product is evaluated to determine if it is meeting the needs identifed from the analyis in the the evaluation stage. ADDIE, like the iterative model contains one or more feedback loops. Figure 4 shows two versions of ADDIE. In one instance of ADDIE, issues identified at any of the phases can cause one to go back and modify the previous phases (Molenda, 2003). Another instance ADDIE is circular, with evaluation feeding back into analysis.



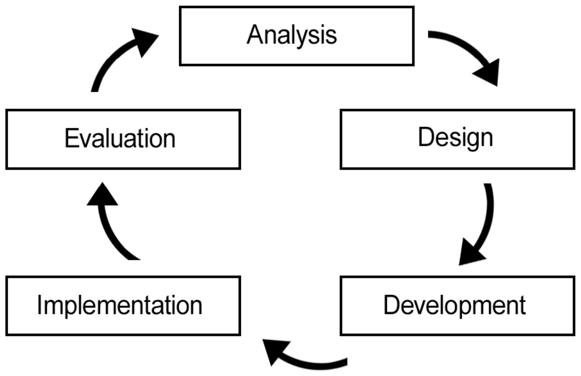


Figure 4. Two Versions of ADDIE

As noted, the phases of ADDIE correspond with common phases in software engineering (Jonassen, 2004). Analysis corresponds to *requirements and desired features* in the iterative model and *system requirements, software requirements and analysis* in the waterfal model. Design corresponds to *program design* in the waterfall model and *design* in the iterative model. Development corresponds to *coding* in both the waterfall and iterative models. Implementation and Evaluation correspond to *testing* in the waterfall model and *evaluation* in the iterative model.

The Create-Adapt-Generalize (CAG) Model

The Create-Adapt-Generalize or “CAG” software development process evolved from work on various web and database projects undertaken at Virginia Tech. The CAG process is an iterative approach that capitalizes on the presence of conceptually redundant tasks. The model was first described in the published literature in a paper entitled “The Collaborative Design and Development of a Database Management System as a Tool for Historical Inquiry” (Hicks, et al., 2004).

Many projects begin with a specific, narrowly defined, focus. A specific need is identified and it is determined that some form of technology is appropriate to address that need. While the Create-Adapt-Generalize model can actually be applied to many forms of technology, not just computer technology, here, CAG will be viewed as applying to computer technology, specifically the development of computer software.

In the Create-Adapt-Generalize software development model, the idea is to first create software that meets a specific, usually narrowly defined need. However, it often becomes clear that with some adaptation the software can be used to meet a wider variety of needs. The software is then adapted, modified and tested as needed. As more needs are identified the original software can be further adapted. Eventually it becomes clear that with appropriate adaptations, it is usually possible to make the software address a more general class of needs so that further adaptations become less and less necessary.

In their paper describing the model, Hicks et al. describe a project involving the design and development of a database management system for historical inquiry. For this project the original software focused specifically on the context of historical inquiry related to the Civil War. However, after the software had been developed, tested and was functioning in the single context, it was adapted for use with two additional contexts, the Great Depression and World War II. Finally it became clear to the developers that the software could be generalized to be used with multiple contexts and contents (Hicks, et al., 2004). In developing their system, Hicks, et al., identified several steps for each of the three steps in the Create-Adapt-Generalize process (2004). These steps are presented in Table 1.

Table 1

Create-Adapt-Generalize Process, adapted from Hicks, et al. (2004)

Stage	**Sub-stage**	**Description**
Create	C1	Identify needs in content-specific terms (CST)
	C2	Convert CST to technology-related terms
	C3	Determine role of technology in meeting needs
	C4	Design system
	C5	Select appropriate technologies
	C6a	Develop technological framework
	C6b	Populate framework with initial data
	C7	Test initial system
	C8	Modify initial system, as needed
Adapt	A1	Examine initial system for adaptability to other contexts
	A2a	Revise technological framework
	A2b	Populate framework with data from other context(s)
	A3	Test adapted system
	A4	Modify adapted system, as needed
Generalize	G1	Examine adapted system for generalizability to multiple contexts and contents
	G2	Generalize technological framework
	G3	Populate generalized framework with multi-contextual data
	G4	For each context, assess functionality of generalized framework
	G5	Modify functionality of generalized framework, as needed

CAG and Other Software Development Processes

CAG is a relatively new model appearing in the literature for the first time in the article by Hicks et al. in 2004. Nevertheless there are some similarities between CAG and the iterative software development model as well as ADDIE. The iterative model and ADDIE involve creating a basic version of the program which addresses some of the needs, as identified in an initial list of features and then adding more features in each successive iteration, including features that were added to the list of desired features as a result of evaluation and user feedback. This process is very similar to the create-adapt stages of the CAG model.

One of the shortcomings of the iterative process is that it is potentially endless with multiple iterations of small incremental changes. In this respect the CAG model differs from, and potentially has a contribution to make to, the iterative model and ADDIE. The create-adapt part of Create-Adapt-Generalize is essentially the same as the iterative process. Where CAG differs is in the explicitness of the "generalization" direction for each successive iteration/adaptation. Generalization provides some direction to the iterative process. The "generalize" part of CAG can be viewed as a way to reach an end point in iterations. In the absence of an effort to generalize, the iterative process potentially becomes the equivalent of a Create-Adapt-Adapt-Adapt-Adapt… model.

True generalization is an abstract ideal, which cannot be realistically achieved. The theoretical ultimate generalized program would be able to do anything. The "generalize" part of CAG could be more realistically viewed as a direction rather than a destination. In the abstract, if a sufficient level of generalization can be achieved at the beginning (the analysis stage) of a programming project, then the iterative process becomes unnecessary and the resulting software development lifecycle resembles the "pure waterfall model". However, since generalization is an abstract ideal -- a direction rather than a destination -- in practice, a pure waterfall model would never be practical. No matter how general a program is, short of being able to do “anything”, it can always be more general.

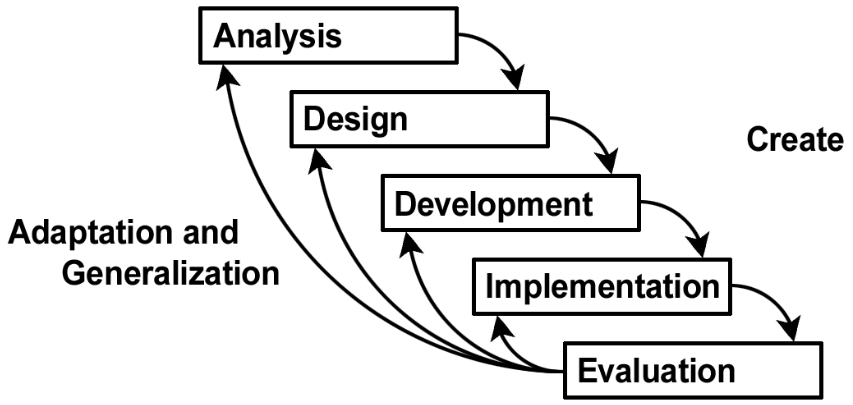


Figure 5. ADDIE and the CAG Model

While CAG can be viewed as a variation of ADDIE and the iterative model, it is also possible, as shown in Figure 5, to view CAG as a kind of “meta-model” that gives direction to the evolution of software rather than a software development model in its own right. Thus potentially, various approaches to software development can be used while using CAG as a model to guide the evolution of such programs in the direction of greater generality.

An important observation about the literature on the software development process is that descriptions of software development processes are often geared to large scale projects involving teams of many people, each creating different pieces of a larger system, all of which must function in concert. On such large-scale projects, upfront planning, documentation and specification of requirements are very important; otherwise individual software modules created by different teams may not work together. Such concerns explain why the Waterfall Model remains in use despite its shortcomings. The CAG model, on the other hand, was developed in the context of small team or individual based software development and perhaps represents an alternative model for such development, in contrast to models that were developed for and geared to large multi-team based development.

Design Patterns

As the process of adaptation and generalization progresses, often several patterns appear with regularity. These regularities are referred to in the literature as “Design Patterns”. In its most generic sense a design pattern “describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice" (Alexander, 1977, p. x). The concept of design patterns as applied to programming was introduced in “Design Patterns: Elements of Reusable Object-Oriented Software” (Gamma, Helm, Johnson, & Vlissides, 1995). In general, a pattern has four essential elements: The *pattern name* introduces a vocabulary for describing the pattern. The *problem* describes the conditions under which the pattern applies. The *solution* describes “the elements that make up the design, their relationships, responsibilities, and collaborations” (Gamma, et al.,1995, p. 817). The *consequences* are trade-offs that result when applying the pattern.

Many of the patterns identified in the literature are primarily focused on programmers designing software under the object-oriented programming paradigm. However, the general concept of patterns in programming is equally applicable to recurring patterns in the programming of information management websites. Given the premise that various programming patterns will reappear in many different information management website projects, the challenge is to identify those patterns and for each pattern, identify the information that changes with each instance of the pattern and the information that remains constant. Then by using metaprogramming principles, craft a metalanguage in which to express only the variable information for each pattern and use the information in the pattern itself to translate from the metalanguage to the object language, with the specific programming details necessary in the object language, but absent in the metalanguage taken from the information in the pattern.

Conclusion

The literature reviewed in this chapter has been focused, in one form or another, on simplifying and making possible tasks that would ordinarily be much more difficult. Electronic Performance Support Systems allow persons to perform tasks they would otherwise not be able to perform. Metaprogramming and design patterns serve to simplify the task of programming. However, Electronic Performance Support Systems design has not focused on the specific task of programming and the literature on metaprogramming and design patterns is focused on assisting persons who already think of themselves as programmers to be more efficient programmers. In this study, the author examines the use ADDIE and the Create-Adapt Generalize model described in this chapter, to extend these efforts to making some subset of the tasks that would ordinarily require programming skills accessible to persons who think of themselves as "non-programmers". A methodology for this endeavor is presented in Chapter 3.

Chapter 3: Methodology

This chapter describes the methodology used to create a tool to assist people with limited programming skills to create customized information management websites that would normally require more advanced programming skills. This tool was named the Form And DataBase Interaction Tool or "FADBIT". The methodology for creating FADBIT followed the well-known ADDIE conceptual framework. Recall that ADDIE described in Chapter 2 is an acronym describing the five major processes in Instructional Systems Design: Analysis, Design, Development, Implementation and Evaluation (Molenda, 2003). The steps of ADDIE also correspond to common steps in software engineering (Jonassen, 2004).

Within each of the five phases of ADDIE, relevant applications of the Create-Adapt-Generalize (CAG) model (Hicks, et al., 2004) were considered, either within each phase or as a guiding meta-model for successive iterations through the ADDIE steps. Given the CAG model's imperative to *generalize,* selected projects were examined from the perspective of creating components of a generalized tool that could be used to create similar projects.

FADBIT was expected to share some, though not necessarily all, of the features of an Electronic Performance Support System. Some relevant characteristics and capabilities of an EPSS were considered as guideposts to determining the general characteristics and capabilities of the tool. Also, relevant principles of metaprogramming were considered in determining how to design such a tool.

Analysis

The first phase of ADDIE is analysis. In this study, this phase analyzed the needs of five representative projects, all of which involved creating customized information management websites. This analysis was based on needs expressed or implied in conversations with clients as well as the author's programming experience in determining additional features necessary to meet the needs of clients, even when the client was unaware of those needs. The details of the needs analysis across the Larson, Kobyashi, Yilmaz, DOE and ITMA projects are presented in Appendix A, Table A1.

It should be noted that the projects above followed a Create-Adapt-Generalize model. Programming code created for one project was often adapted for other projects. The Larson project had the fewest requirements, while the Kobayashi, Yilmaz, and DOE projects generally added requirements. ITMA was the most complex of the projects and had the most requirements. The items in the list under *Needed Features* in Table A1 should be viewed as a generalized needs analysis applicable to many similar information management website projects. FADBIT was designed in a fashion that allows users to create information management websites that have the features in Table A1.

Part of the analysis in this study reflects the author's discussions with clients with specific information management needs. In addition, the analysis also incorporated features identified in the review of the literature. FADBIT was expected to share some of the characteristics and capabilities of an Electronic Performance Support System (EPSS). An analysis of these characteristics and capabilities provided additional guidelines to the characteristics and capabilities that should also be present in FADBIT. The following characteristics of an EPSS were applicable to FADBIT. An EPSS includes productivity software to perform specific tasks (Gery, 1991). It reduces the need for prior training in order to accomplish required tasks (Sleight, 1993). It reduces the complexity or number of steps required to perform required tasks (Miller, 1996). It allows users to achieve a required level of performance in the fastest possible time (Raybould, 1995). It is a non-instructional solution to performance improvement (Dean, 1995). In keeping with the performance technology niche within Instructional Design and Technology, the focus is on performing, not learning (Cagiltay, 2001). All of the EPSS features mentioned above were included in the design of FADBIT.

Design

In the design phase of ADDIE, a blueprint is created for addressing the needs identified in analysis. In reviewing the needs identified in this study, it was obvious that some of these needs could be addressed with existing software. Because such software is readily accessible, it was not necessary to design FADBIT to duplicate these features. The FADBIT design relies upon, and assumes, continued access to such software. The needs identified in the analysis that can be addressed by existing software are shown in Appendix B, Table B1.

In addition to identifying needs that could be met by existing software, it was also necessary to identify what skills users should possess to use the existing software or to use FADBIT. In the instructional design process this is called the identification of entry skills, or entry behaviors. These are skills that "…learners will already have to know or be able to do *before* they begin the instruction… " (Dick & Carey, 2001, p. 70) and therefore "… fall below the skills you plan to teach." (Dick & Carey, 2001, p. 70).

For the purposes of this study, it was assumed the following entry skills were required of users of FADBIT. Users should be computer literate, familiar with sending and receiving e-mail, and accustomed to browsing the Internet and the World Wide Web. Users should be skilled enough to create a web page using Hyper Text Markup Language or "HTML", a common presentation language for web pages, either by writing HTML themselves or by using a visual editor such as *Adobe® Dreamweaver®*. Users should have a conceptual understanding of databases and database concepts and terms such as *tables*, *fields*, *records*, and *queries.* They also should be able to appropriately respond to a series of menus, prompts and questions concerning the data they want to save to a database, or retrieve from a database and display. Ideally, users should be able to work with an existing database as well as create their own database structures using existing database management software. Users should be able to appropriately respond to a series of menus, prompts and questions related to the elements of the web forms they create and the relationship between those elements, elements on other forms, and the user's database structure.

It is clear that not all persons who wish to create information management websites would have the entry skills necessary to use FADBIT. Ordinarily an IT department would rely on a programmer to create information management websites for such clients. However, within an IT department there are others who, while not skilled in programming, either have, or can acquire, the FADBIT entry skills with much less time and effort than acquiring programming skills. Such persons can then use FADBIT to create information management websites either for themselves or for those persons without the requisite entry skills, lessening the reliance on the programmers. This follows one of the rationales for FADBIT of distributing of work within an Instructional Technology department.

The needs in the analysis not addressed by existing software were the focus of FADBIT. The methodology for addressing these needs was found in metaprogramming. As discussed in the review of literature, metaprogramming is the writing of computer programs that write or manipulate other computer programs. The basic idea in metaprogramming is for the user to tell the computer what function to perform at a more abstract level relative to some other expression of the same functionality. This higher-level expression is called the metalanguage. The metalanguage is then translated or mapped to another language called the object language. Any language, even natural human language, is potentially a metalanguage as long as it precisely specifies what the computer should do. In the traditional programming approach illustrated in *Figure 6*, the mind of a human programmer performs just such a translation.

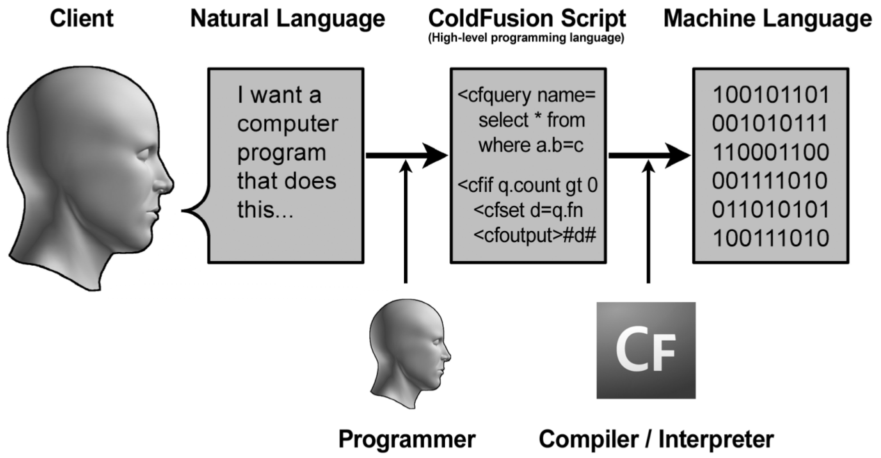

Client
Natural Language
ColdFusion Script
(High-level programming language)
Machine Language
I want a computer program that does this...
<cfquery name= select * from where a.b=c
<cfif q.count gt 0 <cfset d=q.fn <cfoutput>#d#
100101101 001010111 110001100 001111010 011010101 100111010
CF
Programmer
Compiler / Interpreter

Figure 6. Traditional Programming Approach

In the traditional programming approach in Figure 6, there are two levels of translation occurring, a translation from natural language to a high-level programming language (in this case *Cold Fusion* Script) by the mind of a human programmer, and a translation from *Cold Fusion* Script to machine language by the *Cold Fusion* interpreter and the web browser.

In this study it was not plausible to design a metaprogram to replace a human programmer by translating from natural human language to a programming language like *Cold Fusion*. However, it was possible to design a metalanguage that, while more formal than human language, was more abstract than *Cold Fusion* by using the following approach. The metalanguage representation of what the user wants the computer to do is formed by aggregating the user's responses to a series of questions. Questions can take the form a specific questions such as "*What table do you want to use?*" or a directive such as "*Select the field you want to use,*" followed by a list of choices. While it takes a skilled programmer to translate from natural human language to *Cold Fusion,* a sizable number of non-programmers can potentially translate a natural human language description given by a client into appropriate responses to FADBIT metalanguage questions. If the metalanguage description created by the responses to those questions can, in turn, be translated into *Cold Fusion* by a translator program, the human programmer in Figure 6 can be replaced by a two-step process, thus allowing a non-programmer to perform tasks that would ordinarily require programming skill. This approach is illustrated in Figure 7.

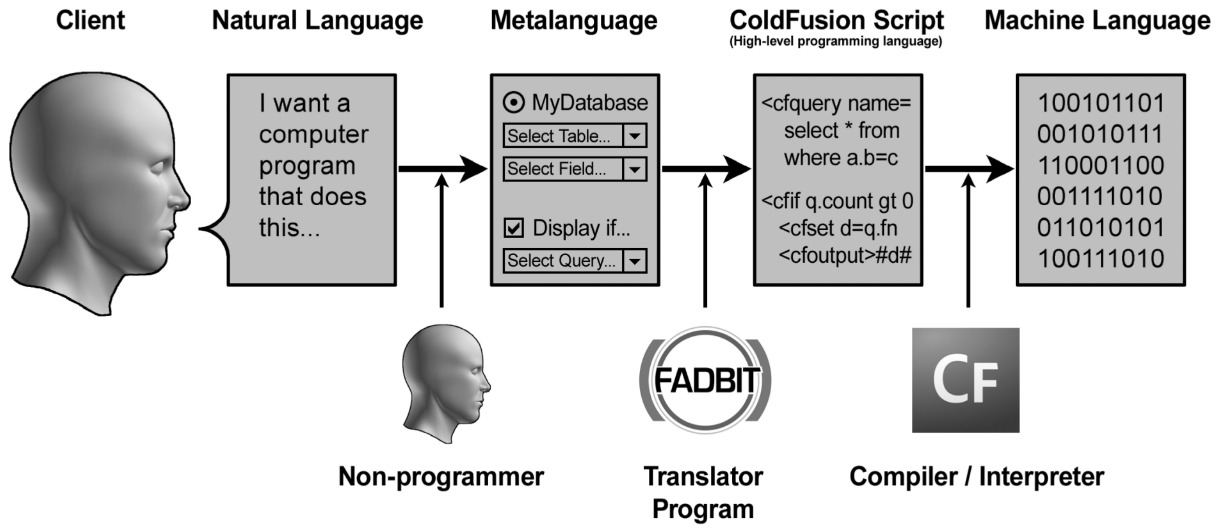


Figure 7. FADBIT Metaprogramming Approach

The needs that FADBIT should address (those needs not addressed by existing software) can be understood as *patterns* that frequently appear in the programming of information management websites. In order to create an appropriate set of metalanguage questions to address these needs it was necessary, for each feature/pattern combination, to identify the *essence of the* *feature,* or the core -- essential information that changes with each instance of the pattern and is therefore needed from the user to create programming code to perform the functions that made up that feature. The remainder of the needs identified in the analysis, an underlying pattern for each need, and a description of the essential information required from the user (the essence of the feature) to create computer code for that feature based upon the underlying pattern, are presented in Appendix C, Table C1. This information formed the basis of the metalanguage questions. For each feature/pattern combination the essential information provided by the user in response to the metalanguage questions was translated to an object language (*Cold Fusion* Script) by combining the essential information with information common to the underlying pattern.

In addition to the specific features above, the analysis noted that FADBIT was expected to share some, though not all of the characteristics and capabilities of an Electronic Performance Support System. One of the capabilities of an EPSS is *productivity software to perform specific tasks* (Gery, 1991). In order to transform a pre-functioning form to a functioning form, it was necessary to insert computer code into the pre-functioning form. This process was used to create functioning forms from pre-functioning forms in the five projects discussed earlier. Thus, the design of FADBIT had to include productivity software to perform the specific task of writing computer code.

The idea of *reducing the need for prior training* in order to accomplish a given task is another characteristic of an EPSS (Sleight, 1993). Without a tool to provide assistance, the task of creating a working form from a pre-functioning form required programming skills. By creating the appropriate code to make a set of functional web forms which combine to make up an information management website, FADBIT was expected to reduce the need for prior training in programming. Users may still need training in the creation of HTML pages, training in database concepts, or instructions in the use of FADBIT itself, but the training normally required in programming skills was expected to be reduced.

Reducing the complexity or number of steps required to perform a task is another capability of an EPSS (Miller, 1996). The programming involved in transforming non-working forms to working forms normally requires a number of complex steps. Consider one of the items in the list of needs above: retrieving data from a database. Information in a database is accessed by a programming language called **S**tructured **Q**uery **L**anguage or "SQL". SQL, first introduced in 1974 as "SEQUEL" (**S**tructured **E**nglish **QUE**ry **L**anguage) is a language "…which can be used for accessing data in an integrated relational database" (Chamberlin & Boyce, 1974, p. 250). SEQUEL/SQL, allows programmers to store, retrieve, edit, search for, and delete the data in a database or modify the structure of the database through the use of *queries* written in the SQL programming language. A query is a command to the database to perform some action such as inserting a new record, retrieving records that meet certain criteria, modifying the contents of an existing record, or deleting a record from the database. Common SQL query commands are *SELECT* for retrieving ("selecting") data from a database, *INSERT* for adding new records to a database, *UPDATE* for modifying the contents of a existing record, and *DELETE* for removing records from the database. An example of a SQL query is shown in Figure 8.

```
SELECT * FROM crn_merge,crn_table,master_student_list
WHERE crn_merge.crn = crn_table.crn and crn_table.index_no =
master_student_list.index_no and crn_table.term in ('200801','200806','200807','200809')
ORDER BY lname,fname
```

Figure 8. Sample SQL Query

Creating queries in SQL, such as the one in Figure 8, and inserting those queries along with other programming code into a form is a common task a programmer would perform in transforming a pre-functioning form to a functioning form, and is required to interact with a database. Writing SQL involves a number of complex steps which a non-programmer would have difficulty performing. The FADBIT design had to reduce the complexity and number of steps required in this process through metaprogramming as described above. A specific metaprogramming module was designed to achieve these goals. This module was called *query builder*. Query builder asks the user a number of questions based on the outline of essential information in the needs table above. Based on those answers (which form a metalanguage) query builder translates or maps them to an object language (in this case SQL) thereby creating a query without the user having to write (or even see) an SQL statement such as the one in Figure 8.

Queries are just one example of a programming task where the design of FADBIT reduced the complexity and number of required steps. All of the items listed above in the table of needs involve a number of complex steps, the complexity and number of which were reduced in a way similar to query builder described above using metaprogramming. Based on the user's responses to a series of questions (the metalanguage), FADBIT translates that metalanguage to an object language, either SQL code, *Cold Fusion* code or HTML that otherwise would be created by a programmer. Responding to questions is far less complex and requires fewer steps than writing code by hand.

Another characteristic of an EPSS is to allow users to achieve a required level of *performance in the fastest possible time* (Raybould, 1995). Programming is a skill that can be learned, but acquiring that skill takes time. By eliminating the need to acquire programming skills to create functional information management websites, the FADBIT design allows users to perform, that is, to create a functional product that meets their needs, much faster than they would otherwise be able to do.

An EPSS is also a *non-instructional solution* to performance improvement (Dean, 1995), *focusing on performing*, not learning (Cagiltay, 2001). The purpose of FADBIT is to improve performance, not to teach programming skills. The FADBIT design allows users who are not skilled in programming to perform tasks that would ordinarily require programming skill.

Development

In the development phase of ADDIE, the product is created based on the design. In developing FADBIT, each of the features listed in Table 2 had to be addressed. For each required feature, it was necessary to translate a metalanguage representation of the specific task to be performed into *Cold Fusion* code. The translation for each feature is handled by a series of FADBIT *modules*. Each module was designed to transform a specific aspect of a form. This version of FADBIT was designed around a generalized modular framework that allows an unlimited number of modules to be created. This modular framework also accommodates the CAG model in that it is possible to adapt FADBIT by creating new modules, or updating existing modules to add additional functionality as new needs are identified.

As developed, each FADBIT module consists of three parts. The first part creates underlying code that takes the user's metalanguage responses and combines those responses with pattern information to translate the metalanguage into *Cold Fusion* code. The second part is a user interface that, based on the specific elements of a pre-functioning web form, selects the set of questions the user needs to answer, asks those questions, and provides the responses to the code that performs the translation. The third part takes the translated Cold Fusion code, determines where in the pre-functional web form the code should be inserted, and inserts the Cold Fusion code along with any necessary HTML code into the pre-functional web form, thereby adding functionality to that form.

Implementation and Evaluation

In a developmental study implementation and evaluation often overlap. Implementation in ADDIE is the actual use of the product by its intended audience (Visscher-Voerman & Gustafson, 2004). While the product was used by its intended audience and is consistent with the implementation phase of ADDIE in that respect, in this study the implementers and the evaluators were the same people. In this study, a prototype version of FADBIT was implemented and evaluated by six independent evaluators. FADBIT consists of a generalized framework that allows users, interacting via a web-based interface, to login, select projects, and upload pre-functioning forms. In practice FADBIT consists of series of modules that add functionality to pre-functioning forms by inserting appropriate programming code. Using the web-based interface the user can login using a name and password. The user is then presented with an option to select a project, either selecting an existing web form or uploading a new web form from a file on the user's local computer. The user has a choice of using either a *Novice Mode* with a reduced set of more frequently used options, or an *Advanced Mode* with a full range of options. The decision of which options to reserve for the Advanced Mode was made after completion of the first evaluation of FADBIT.

The evaluation of FADBIT followed its implementation. FADBIT was evaluated to determine if it meets the needs identified in the analysis and is capable of addressing each of the patterns in Table 5 derived from those needs. In order to evaluate how well FADBIT meets its stated goal of assisting non-programmers to create information management websites that would normally require significant programming skills, the author created two sample projects, each consisting of a database and a set of pre-functional forms. The six independent evaluators were then asked to transform these pre-functional forms into a working website using FADBIT.

The first of these projects was a guided project that was used to manage a movie collection online. The purpose of the guided project was twofold. The first purpose was to show users how to use FADBIT by walking them through, in an explicit step-by-step fashion the procedure for transforming a set of pre-functional form to functional forms with explanations for each step. In the guided project the user worked with all of the FADBIT modules and became accustomed to the look and feel of the FADBIT interface. The second purpose of the guided project was to determine if FADBIT, when used correctly, was able to accomplish its stated purpose to transform a set of pre-functional forms to functional forms, thus creating an functional information management website.

A second unguided project was used to manage a list of students online. The purpose of the unguided project was to test the usability of FADBIT, that is, the ability of people, having become familiar with FADBIT by completing the guided project, to transfer that knowledge to transform a different project without explicit step-by-step instructions.

Since the purpose of the evaluation was to evaluate FADBIT and not the ability of the users to create HTML forms or databases, the author created the databases and the pre-functional forms. A standard set of databases and pre-functional forms provided a control for these factors and allowed the evaluation to focus on FADBIT itself.

In addition to completing the two projects just described, the evaluators were asked to complete the evaluation questionnaire shown Appendix D about their experience with FADBIT and questions about their experience with programming or databases. In accordance with the CAG model described in Chapters 2 and 3, FADBIT was expected to undergo continuous adaptation and generalization as it was evaluated. Given the evolving nature of FADBIT, all evaluations were considered to be formative evaluations. Results and proposed modifications and extensions to FADBIT identified in the tests conducted by the author and the evaluations performed by the independent evaluators are reported and discussed in Chapters 4 and 5.

Chapter 4: Results

This study sought to answer two research questions. The first research question was: What functionalities, characteristics and capabilities could be included in an electronic performance enhancement tool that would allow persons with limited programming skills to create simple customized information management websites that would normally require programming skills? The second research question was: How could an electronic performance enhancement tool with such functionalities, characteristics and capabilities be designed and developed?

The first question was answered by analyzing empirical data and by conducting a review of relevant literature to identify desired functionalities, characteristics, and capabilities. These tasks were followed by operationalizing the desired features through the design and development of a functioning prototype tool. The functionalities, characteristics and capabilities included in the developed prototype tool are:

The tool allows users, interacting via a web-based interface, to login, create and manage projects, and upload pre-functioning forms. The tool allows users to select a non-working form and is able perform an automated analysis of that form, identifying relevant elements the tool needs additional information about in order to generate appropriate programming code. The tool asks a series of questions of the user which include universal questions that apply to all forms as well as questions related to relevant elements the tool has identified in the automated analysis of individual forms. By acting upon the user's responses to these questions, the tool is able to form a metalanguage representation of what the user wants the form to do. The tool is capable of translating the metalanguage representation of what the user wants the form to do into a useful object language representation of the same functionality. The tool has the capability to perform this translation for all of the patterns identified in Chapter 3. These are: saving data, navigation from one form to another, login, authorization, querying a database, displaying information in table form, remembering data selected or entered, populating a dropdown menu, pre-selecting an option in a radio button set, pre-filling a textbox, and displaying text from a query or remembered data. The tool is able to create a functioning form from the pre-functional form.

Additionally, the tool was designed to reduce the need for prior training in programming in order to create a functioning form from a pre-functional form, to reduce the complexity or number of steps required to create a functioning form, to enhance users performance such that they can achieve a required level of performance in the fastest possible time, and to focus on performance, not learning. However, since no widely available tools possessing all of the above features were located, it was not possible to conduct direct comparisons of the developed prototype tool with existing tools relative to these factors.

The second research question was: How could an electronic performance enhancement tool with such functionalities, characteristics and capabilities be designed and developed?

The tool was successfully designed and developed using a generalized modular framework, the Create-Adapt-Generalize model, and a metaprogramming approach. The generalized framework developed for this study supports a number of modules, each of which can generate computer code relevant to one or more of the patterns identified above. The modules are independently adaptable and expandable. New modules can be created and added to the framework as needed. Each module addresses one or more patterns. For each pattern the core essential information that changes with each instance of the pattern and the information that remains constant were identified. Then by using metaprogramming principles, a metalanguage representation was crafted based on the user's responses to questions related only to the variable information in each pattern. This metalanguage representation was then translated into more familiar programming code by combining the information in the metalanguage representation with information from the pattern.

The Create-Adapt-Generalize model is iterative in nature and evaluation is an important part of the adaptation and generalization steps. In this study, timely testing of the individual modules and the entire tool by the author and subsequent evaluations of the relative success of the individual modules and the tool as a whole by independent evaluators were an important part of the approach. Based upon these tests and evaluations, the overall tool and specific modules were adapted and generalized in response to difficulties encountered by the creator or other evaluators.

This modular approach used to actually develop the initial prototype version of FADBIT is described below followed by the results of the evaluation of FADBIT. Specific adaptations and generalizations, both completed and proposed, in response to difficulties encountered by evaluators are discussed in Chapter 5.

The FADBIT Prototype

FADBIT consists of a generalized framework that allows users, interacting via a web-based interface, to login, create and manage projects, and upload pre-functioning forms. The FADBIT framework is organized around a series of *modules*. This current version of FADBIT has ten modules, Login, Authorization, Queries, Text, Textboxes, Dropdown Menus, Radio and Checkboxes, Reports, Save Data, and Navigation, as can be seen on the left in the previous figure. Each module examines the selected non-functioning form by asking questions of the FADBIT user and making appropriate directives. Once collected, this essential information forms a metalanguage description of the functionality the user wishes to add to the pre-functional form. The module then inserts into the pre-functional form FADBIT code containing this metalanguage description. The FADBIT code, when executed, translates or transforms this metalanguage description into Cold Fusion code. An overview of the FADBIT prototype and each of the ten modules with a logic diagram illustrating the flow of logic through the module are presented in Appendix E.

Implementation and Evaluation of FADBIT

In this study, a prototype version of FADBIT was implemented and evaluated by six independent evaluators with the entry skills listed in Chapter 3. The central question was whether or not users with the appropriate entry skills can, in a reasonable amount of time, successfully use FADBIT to add functionality to a researcher-provided set of pre-functioning forms. In the independent implementation and evaluation, the six evaluators each attempted, using only FADBIT, to create a functioning information management website for both the guided project and unguided project described in Chapter 3. In addition, the evaluators were asked to complete a questionnaire about their experience with FADBIT.

Prior to the independent evaluation, a "user's guide" to FADBIT was created. This guide was prepared in written form in a loose-leaf notebook for use by an evaluator. Representative pages of this guide can be found in Appendix F. The main contents of the guide were two pre-functioning webform projects alluded to above. The first project was presented as step-by-step instructions to the guided project in which the user was to imagine a movie collection. This project consisted of six pre-functioning webforms (pages), login.html, choices.html, selectflix.html, editflix.html, addflix.html, and viewflix.html, and was designed to allow a user to login with a user name and password, add, view, select, edit, and save movie information to and from a database. These source files were provided to each evaluator on a flash drive. The database used in this project was listed in written form, including names of tables, fields, and complete records fields, in the guide. The actual database was located on the FADBIT web server. The purpose of the first project was to familiarize an evaluator with the look and feel of the FADBIT interface as it cycled through each of its modules to add functionality to the pre-functioning webform. This was done with explicit written instructions and declarations in the notebook guide.

The user's guide then asked an evaluator to add functionality using FADBIT to the unguided project. This project consisted of six pre-functioning webforms (pages), login.html, mainmenu.html, selectstudent.html, editstudent.html, newstudent.html, and viewstudents.html. As before, the files were provided to the evaluators on a flash drive. The purpose and intent of each of these files was described to a user in a brief narrative much in the same manner that the designer of the project would give to a colleague who was to use FADBIT to add functionality to the webforms. The narrative also outlined the database to be used in this project. The database was located on the FADBIT web server. Without further instruction, and with no coaching during the process, each evaluator was asked to use FADBIT to add functionality to the unguided project. The purpose here was to test whether an evaluator, having become familiar with FADBIT in the guided project, could proceed without instruction in a similar, but different project.

Each evaluator was invited to attempt to use the pre-functioning webforms before using FADBIT to verify that indeed the webforms did not function, and then to use the resulting webforms after using FADBIT to test the final product. Each evaluator was able to complete his or her evaluation within a single day and only one evaluation was conducted on the same day. The six evaluations were conducted over a three month period.

Guided Project Results

The first three questions in the questionnaire concern the guided project. This tutorial walked the evaluator through the process of converting pre-functional forms to functional forms. The first question asked the evaluators to indicate if they were able to complete various actions within FADBIT for this project. The responses are shown in Appendix G, Table G1.

All six evaluators were able to transform the pre-functional forms into functional forms for the guided project, resulting in functional information management websites for managing a movie collection. In response to the first question in the questionnaire, there was some discrepancy between the tasks the evaluators completed and the self-reports indicated on the questionnaire. The first question asked the testers to indicate whether or not they were able to perform various tasks using FADBIT. Although it was possible for the author to inspect the final project created by each evaluator and determine what tasks had been successfully completed, this question gave the evaluators the opportunity to self-report whether or not they were able to complete the task and describe any difficulties. Several evaluators reported "no" or otherwise did not indicate "yes" for the tasks *Populate drop-down menus?, Pre-select an option from a dropdown menu?, Remember data from a dropdown menu?,* and *Pre-select an option from a set of radio buttons?*. However, on inspection of the finished project, the author was able to determine that the evaluators had completed these tasks correctly. One of the evaluators indicated that it was unclear what these questions were asking, stating for one task "I'm not sure what you are asking." and for another task "The option was available, but I don't think I used it".

In addition to the self-reports from the questionnaire, the author inspected the final project created by each of the six evaluators. After each evaluator completed the guided project, the author inspected the code inserted by FADBIT into the six files comprising the project and did a test run. Upon logging in, and selecting the appropriate form via the main menu page, the author attempted to use all of the functions of the website: adding new movies to the database, selecting and editing an existing movie from the database and viewing all movies. This testing and inspection by the author proved successful for all six of the tutorial projects. Additionally, the author used a functionality checklist to determine if each type of functionality was present in the final product created by each of the six testers. The results of the functionality checklist for the guided projects are shown in Table 2.

Table 2

Functionality Checklist for the Guided Project

		Evaluator					
Module	Functionality	1	2	3	4	5	6
Login	Successful login	✓	✓	✓	✓	✓	✓
	Unsuccessful login	✓	✓	✓	✓	✓	✓
	Remembered data	✓	✓	✓	✓	✓	✓
Authorization	Successful authorization	✓	✓	✓	✓	✓	✓
	Unsuccessful authorization	✓	✓	✓	✓	✓	✓
Queries	Simple query	✓	✓	✓	✓	✓	✓
	Filtered query	✓	✓	✓	✓	✓	✓
	Multi-table query	✓	✓	✓	✓	✓	✓
Text	Show remembered text	✓	✓	✓	✓	✓	✓
Textboxes	Pre-fill textboxes	✓	✓	✓	✓	✓	✓
Dropdown Menus	Populate dropdown menus	✓	✓	✓	✓	✓	✓
	Pre-select dropdown option from query	✓	✓	✓	✓	✓	✓
	Remember dropdown selection	✓	✓	✓	✓	✓	✓
Radio Button Sets	Pre-select radio set option from query	✓	✓	✓	✓	✓	✓
Reports	Report based on multi-table query	✓	✓	✓	✓	✓	✓
Save	Save new data from textbox	✓	✓	✓	✓	✓	✓
	Save new data from dropdown	✓	✓	✓	✓	✓	✓
	Save new data from radio button set	✓	✓	✓	✓	✓	✓
	Update record with textbox data	✓	✓	✓	✓	✓	✓
	Update record with dropdown data	✓	✓	✓	✓	✓	✓
	Update record with radio set data	✓	✓	✓	✓	✓	✓
	Save buttons save	✓	✓	✓	✓	✓	✓
	No Save buttons do not save	✓	✓	✓	✓	✓	✓
Navigation	Simple navigation	✓	✓	✓	✓	✓	✓
	Conditional navigation	✓	✓	✓	✓	✓	✓

✓ = Working

The evaluators were asked how much time they spent in completing the guided project. The lowest evaluator estimate for time spent was 40 minutes and the longest was 3 hours. Three of the six evaluators estimated a time of one hour or about one hour. The individual responses are shown in Appendix G, Table G2.

The evaluators were asked how easy the instructions to the guided project were to follow and to list their difficulties. Five of the six evaluators indicated that the tutorial was easy to understand with one evaluator saying the tutorial was initially a bit difficult. This evaluator did, however, successfully complete this project. The full responses are shown in Appendix G, Table G3.

Unguided Project Results

Immediately after completing the guided project, each evaluator then attempted to use FADBIT to transform the pre-functional forms of the unguided project into a functional information management website. These efforts produced results that were generally successful, although the testing did reveal some areas of difficulty.

The evaluators were asked to indicate if they were able to complete a variety of tasks within FADBIT for the unguided project. These results are shown in Appendix H, Table H1. Similar to the responses to the first question, there was also some discrepancy between the tasks the evaluators completed, based on inspection and testing by the author, and the self-reports on the questionnaire. A functionality checklist for the unguided project is shown in Table 3.

Table 3

Functionality Checklist for the Unguided Project

		Evaluator					
Module	**Functionality**	**1**	**2**	**3**	**4**	**5**	**6**
Login	Successful login	✓	✓	✓	✓	✓	✓
	Unsuccessful login	✓	✓	✓	✓	✓	✓
	Remembered data	✓	✓	✓	✓	✓	✓
Authorization	Successful authorization	✓	✓	✓	✓	✓	✓
	Unsuccessful authorization	✓	✓	✓	✓	✓	✓
Queries	Simple query	✓			✓		
	Filtered query	N	✓	✓	N	✓	✓
	Multi-table query	✓	✓	✓	A	✓	✓
Text	Show remembered text	✓	C	✓	✓	✓	✓
Textboxes	Pre-fill textboxes	✓	✓	✓	✓	✓	✓
Dropdown Menus	Populate dropdown menus	P	✓	✓	✓	✓	C
	Pre-select dropdown option from query	N	✓	✓	P	✓	P
	Remember dropdown selection	✓	✓	✓	✓	✓	✓
Radio Button Sets	Pre-select radio set option from query	✓	✓	✓	✓	✓	✓
Reports	Report based on multi-table query	✓	✓	✓	C	✓	✓
Save	Save new data from textbox	✓	✓	✓	✓	✓	✓
	Save new data from dropdown	N	✓	✓	✓	✓	✓
	Save new data from radio button set	✓	✓	✓	✓	✓	✓
	Update record with textbox data	✓	✓	✓	✓	✓	A
	Update record with dropdown data	N	✓	✓	✓	✓	A
	Update record with radio set data	✓	✓	✓	✓	✓	A
	Save buttons save	✓	✓	✓	✓	✓	✓
	No Save buttons do not save	✓	✓	✓	✓	✓	✓
Navigation	Simple navigation	✓	✓	✓	✓	✓	C
	Conditional navigation	P	✓	✓	C	✓	✓

✓ = Working
C = Choice by evaluator is different than intended, but still works
P = Partially Working
N = Not Working
A = Attempt not made
(Blank) = Not Required

Some problems did surface in the attempt to transform the pre-functional forms of the unguided project into a functioning information management website. The most significant issues surfaced with the first evaluator. This effort resulted in a website that, while it did not produce any error messages, was only partially functional, and uncovered a few areas of difficulty with the usability of FADBIT. As a result, some modifications were made to FADBIT between the first and second evaluation. These modifications successfully addressed the particular areas of difficulty encountered by the first evaluator. The modifications included the selection of certain features to reserve for the Advanced Mode. A discussion of the specific difficulties encountered and how those difficulties were addressed is found in Chapter 5.

The evaluators were asked how much time they spent in completing the unguided project. Times ranged from 40 minutes to 4 hours. The individual responses are shown in Appendix H, Table H2.

The evaluators were asked if the guided project tutorial prepared them to complete the unguided project. Five of the six evaluators replied in the affirmative and one evaluator offered some constructive suggestions. The full responses are shown in Appendix H, Table H3.

While it is important for FADBIT to be able to complete its stated objective of transforming pre-functional forms to functional forms, in order to be of value, it must also be relatively easy to use. The questionnaire also asked evaluators to compare the ease of use of FADBIT to other popular software. Evaluators rated FADBIT as being approximately as easy to use as Microsoft Word, Excel and PowerPoint, and easier to use than Adobe Dreamweaver, the software (or its equivalent) that FADBIT relies on to create the pre-functional forms. The complete responses are shown in Appendix I, Table I1.

FADBIT is intended to assist people who have limited programming and database experience to create functional information management websites given a set of pre-functional forms. The questionnaire, therefore, asked evaluators about their programming and database background. Of the six evaluators, only one had any significant programming or database experience, three had essentially no experience and two had minor experience. The complete responses are shown in Appendix J, Table J1 and Table J2.

The evaluators were also asked if they would consider using FADBIT if they needed to create a website similar to the ones in the test projects. Five of the evaluators said they would, and one said possibly. The exact responses are shown in Appendix K, Table K1. Lastly, the evaluators were given an opportunity to make any additional comments about FADBIT. Three of the evaluators had some additional positive comments on their experience with FADBIT. Their responses are shown in Appendix K, Table K2.

FADBIT in an Unstructured Setting

While FADBIT performed well in the formal evaluation, it was recognized that the evaluation was highly structured with the author creating both the database and a set of pre-functional forms. The author was also interested in how FADBIT would perform in a less structured situation in which a set of forms was created by a client with little knowledge of the current required structure, inner workings, or limitations of FADBIT. Such an opportunity presented itself soon after the completion of the formal evaluation.

After completing the formal evaluation, interest was expressed by a client in using FADBIT to create an information management website needed as part of a dissertation project. This project provided an opportunity for the author to more informally evaluate FADBIT under less structured conditions.

For this project, the client created the pre-functional webforms and attempted, with guidance first from the guides mentioned above and later from the author, to use FADBIT to add functionality to the webforms. Several functions of FADBIT worked quite well. The client was able to successfully create a login page, add authorization code to pages, create queries, display remembered text and populate queries. FADBIT did, however encounter a few difficulties. The initial difficulties were related to the formatting of the HTML codes and the names of various form elements. FADBIT works by parsing the HTML code in the pre-functional forms to identify specific form elements. This parsing assumes the HTML is properly formatted. Improperly formatted HTML code in the pre-functional forms can cause FADBIT to fail.

In addition to the improperly formatting of HTML, two other difficulties presented themselves. In HTML, form elements, such as text boxes, radio buttons and dropdown menus, generally have names associated with them. These names are designated using the "name" parameter inside of a tag, for example, <input type=text name=student>. In order to function, FADBIT needs each form element to have a unique name, or in the case of radio buttons or checkboxes, for each set to have a unique name. In the prototype version of FADBIT developed for this study, it was assumed that the creator of the pre-functional forms would name all of the form elements appropriately when creating the pre-functional forms. In actual practice and in a less structured setting, this may not happen and could cause FADBIT to fail.

Another issue uncovered was difficulty with the way a particular HTML creator editor program, Adobe Dreamweaver®, inserts a critical tag, the <form> tag, within the pre-functional form. The <form> tag, and its corresponding </form> tag encapsulates all of the elements within a page, the data from which will be transmitted when the user presses a submit button. Any data entered in tags outside of the enclosing <form> and </form> tags will not be transmitted and will be lost. In at least some the pre-functional forms created by the researcher in the unstructured test, each? of the form elements were encapsulated within their own individual <form> and </form> tags. This arrangement of tags would cause the form to fail.

Another problem that presented itself was the creation of complex queries in the query module. In one iteration of the project, one of the pre-functional forms required a query that combined many different tables. From a technical programming viewpoint, combining multiple tables is not problematic. One of the purposes of the query module is to make the creation of such queries less difficult and less prone to human error. In many cases the query module does accomplish this goal very well. However, the one area of difficulty is the step in the query module in which the user is asked to match equivalent key fields in multiple tables. This step presents a user who is only moderately familiar with relational database design on a conceptual level with a daunting menu of options, all of which need to be correctly selected or else the resulting query will not retrieve data from the database as expected or intended, or even fail to retrieve any data at all.

Another issue uncovered in this project was a feature for the drop down menus that was not available in the prototype. The standard dropdown menu in HTML contains two values for each option. First there is the display value, the text that is presented in the list of choices seen by the end user, and the hidden value associated with each displayed item. This hidden value is the value that is actually transmitted when the user on a form presses a submit button. These values can be the same but do not have to be. An example would be a list of names of people in which the person's name is displayed in the dropdown menu but the hidden value (the one actually submitted) is a unique id number associated with that person. One of the features in FADBIT is the ability to insert code into a form to remember items selected from a dropdown menu. This code can, however, only remember data actually submitted by the form, which means that the value displayed on the dropdown menu cannot be remembered. In the referenced project there were several pages that needed to remember both the hidden value and the display value. Finally, the issue of structuring of the project needs to be addressed. For example, trying to do too much on one page, particularly attempting to save multiple records to the same table, can be problematic with the prototype version of the Save Module. This and other issues related to the structuring of projects need to be carefully thought out. A discussion of these issues and proposed modifications and extensions to address them is found in Chapter 5.

Chapter 5: Discussion

This study was an effort to design, develop, and evaluate an electronic performance enhancement tool that can assist people with limited programming skill to create a variety of simple customized information management websites. The first part of this chapter provides a brief summary of the study and draws conclusions about the effectiveness of the tool in meeting its stated goal. The next part of this chapter presents the authors experiences with use of the tool in well structured and less structured settings. The chapter concludes with suggestions for extensions to the existing tool and a discussion of the contribution of this study to the field of Instructional Design and Technology.

Summary and Conclusions

This study was an effort to design, develop, and evaluate an electronic performance enhancement tool that can assist non-programmers in the creation of a variety of simple customized information management websites. Information management websites that make use of electronic databases have the potential to be of value in a wide array of educational settings. This study includes a review of several instances in which the author was asked (as part of his job as a programmer) to modify webform projects or add to their functionality. People using Adobe Dreamweaver® created parts of the projects, but did not have the requisite skills to make the projects fully functional. The author completed each of these projects by manually adding the appropriate code. While having a programmer add the required code is effective, programmers are not always available when needed. The need for a tool that could perform some of the functions of a programmer follows naturally.

A review of the literature suggested that certain aspects and characteristics of Electronic Performance Support Systems (EPSS) should be considered in the design and development of this tool. The same review also suggested that metaprogramming, that is, the writing of computer programs that write or manipulate other computer programs, is applicable for this tool. Of the many software development approaches listed in the literature, Create-Adapt-Generalize (Hicks, Potter, Snider & Holmes, 2004) and ADDIE are used in this study. The idea of the CAG model is to first create software for a specific need and then adapt and generalize as needed. ADDIE is an acronym for Analysis, Design, Development, Implementation and Evaluation and evolved via oral tradition (Molenda, 2003). The phases of ADDIE are used in Instructional Design and Development and also correspond with common phases in software enginerring (Jonassen, 2004).

The study identified needs based on the author's experience with the information management website projects alluded to above and a review of relevant literature. These needs were incorporated into a series of patterns that formed the core of the design of the tool. These patterns were encapsulated in a series of ten modules. Upon users uploading their pre-functioning webforms, each module collects the responses to questions asked of the users (the metalanguage), creates the necessary code (*Cold Fusion* and/or HTML) and inserts it into the users' webforms to add functionality to them. This is all implemented with a web-based interface. An evaluation of the tool was conducted by asking independent evaluators to follow a combination tutorial and user's guide through two sample projects and answering a short questionnaire.

The results of the study demonstrated that the tool was generally able to meet its stated objective of assisting people with limited programming skill to create simple customized information management websites for the structured evaluation outlined at the end of Chapter 3. Six evaluators were able to successfully transform the pre-functional forms for a guided tutorial into a functioning information management website using only the tool. The successful transformation of the guided tutorial project, using detailed step-by-step instructions provides empirical evidence that the tool is capable of creating a functioning website if used correctly. The creation of a non-guided project was also generally successful. A majority of the evaluators were able to add each of the features in a functionality checklist, although some evaluators did have difficulty with individual features. Since all of the features were successfully added by some of the evaluators, it seems that difficulties are with the use of the tool rather than the capabilities of the tool itself.

In addition to evaluating its ability to transform pre-functional forms, the tool was expected to share some characteristics and capabilities of an Electronic Performance Support System (EPSS). The first of these identified characteristics was the inclusion of productivity software to perform specific tasks. The tool includes productivity software for the task of writing computer code. The tool also was expected to reduce the need for prior programming training in order to accomplish required tasks. Two-thirds of the evaluators had little experience with programming or database design, yet even without programming experience, these evaluators were able to create functioning information management websites using the tool. The tool was expected to reduce the complexity or number of steps required to perform required tasks (Miller, 1996). Through the use of relevant prompting questions and metaprogramming, the modules of the tool reduce the complexity and number of steps required to create programming code. All modules of the tool have this feature. The tool was expected to allow users to achieve a required level of performance in the fastest possible time. Although the study did not determine if the tool achieved this goal, it was clear that the perceptions of the evaluators of the time required to create the projects using the tool were considerably less than the time required to create the project by direct coding. Finally, the tool was expected to provide a non-instructional solution to performance improvement. In keeping with the performance technology niche within Instructional Design and Technology, the focus of the tool was on management and performance, not learning. The tool was designed to be a performance enhancement tool.

Discussion

Difficulties Encountered by the First Evaluator

The most significant difficulties in the use of the tool surfaced with the first evaluator. This effort resulted in a website that, while it did not produce any error messages, was only partially functional, and uncovered a few areas of difficulty with the usability of the tool. The most significant areas of difficulty were connected to drop down menus. The drop down menu box is a frequently used input element on web forms. Within the tool there are four types of functionality connected to drop down menus: populating the list of options, pre-selecting one of the options from the list, remembering an item selected, and saving data selected from a box to the database. The first evaluator had difficulty correctly populating and saving data from some of the dropdowns. While the tool is fully capable of correctly populating and saving a selection from a drop down menu, as shown by the successful completion of the guided tutorial project, the procedure can be confusing to users. The standard procedure for populating a dropdown menu is to first create a query in the Query Module to retrieve data needed to populate the menu, then in the Dropdown Menu Module, select that query and appropriate fields. This procedure works if used correctly. Nevertheless, the procedure requires some pre-planning on the part of users. Users must plan ahead and in the query module, create a query to retrieve data for the dropdown. While this seems simple enough, in practice many users may not do this, particularly users who are just learning to use the tool.

In the guided project, the need for pre-planning was effectively bypassed, as the tutorial provided explicit step by step instructions, while in the unguided project, appropriate pre-planning in the initial version of the tool was required to make the project function as intended. The tool guides users through the modules in a linear fashion with the Query Module preceding the Dropdown Menu Module. Users may not think about what data they need for their dropdown menus until they get to the dropdown menu module and a question box appears below the relevant dropdown menu on their form. In the initial version of the tool, if the user did not add the necessary queries in the Query Module, the list of queries in the question box in the Dropdown Menu Module was either empty or missing the needed query. This is what happened when the first evaluator attempted to use the tool.

In an effort to address this difficulty and reduce the complexity for the remaining evaluators of the tool, some modifications were made after the first evaluation. These modifications were based on the author's observation that, in many instances, it is not really necessary to create a query within the Query Module to populate a dropdown menu. The Query Module allows users to retrieve information from the database with many robust and powerful options. For example, users can retrieve data from a combination of tables, filter the results to contain only certain records out of many present in a table, or sort the result in a particular order. For some dropdown menu applications, these features are absolutely necessary. However, in many instances all the user wants to do is retrieve all the records from a *single* table and fill the dropdown menu with those records. In this instance, all the information necessary to populate the dropdown can be specified directly in the Dropdown Module, without the user having to explicitly create a query. In response to the need to reduce the complexity of the tool in the simpler case, but still preserve the robust options for more complex cases, two versions of the dropdown menu options were created. A Novice Mode was created in which the user would only have the option to populate the dropdown from a single table. The original approach then became an Advanced Mode, in which the user would be able to populate the dropdown from a query created in the Query Module. The remaining evaluators in this study all used Novice Mode, as neither of the test projects required Advanced Mode features.

The other area of difficulty encountered by evaluators was with saving data. Two issues with saving data presented themselves. The first issue was that some of the options in the save module were potentially confusing to novice users. Dropdown menus are often populated from a list of items taken from a particular table in the database, and this table and the textual label associated with that dropdown menu on the form are often similar. To the novice user, the intuitive option is to save the data to the table with the most similar name, but in most cases this is incorrect. The correct table is typically one storing a collection of data for the form the user is working on, not the table storing the options for one dropdown menu. The first evaluator made this mistake, and after the first evaluation a change was made wherein the Novice and Advanced Modes mentioned above also included important modifications to the Save Module. In Novice Mode, when saving data from a dropdown menu, the list of possible tables to save data to was changed so it did not include the table used to populate that dropdown menu. For the rare cases where this table is the correct option, it will appear in Advance Mode. All of the remaining evaluators in this study used Novice Mode.

The second issue concerns the final page of the Save Module which asks the user to select options for every field in the tables that are to have data saved to them. Some of these options proved confusing. The solution employed was also to provide different options in

Novice Mode and Advanced Mode. All evaluators in this study except the first used the Novice Mode.

It should also be noted that all computer software has a learning curve. Because this was the evaluators' first contact with the tool, some difficulties in the use of the tool are to be expected. It should be pointed out that the evaluators in this study saw the projects on which they used the tool for the first time when they started their evaluation. However, users who have actually designed the web form projects and are familiar with the data source to be used and know the purpose and flow of each element within the projects would mostly likely have a less error prone experience with the tool.

Future Modifications and Extensions

The various difficulties encountered with the tool in the informal unstructured evaluation presented an opportunity to extend and modify the tool using the CAG model of adaptation and generalization. Such extensions are described below. Some of these extensions and modifications have been at least partially implemented, while most remain proposed extensions that can be implemented but lie outside the current study.

The first difficulty related to the formatting of the HTML in the pre-functional form. The proposed extension is a new module called the Format Module. This new module would come first in the list of modules above the Login Module and would check the formatting of the HTML, fixing errors, if possible, or alerting the user to fix issues in an external HTML editor if necessary. The Format Module would also correctly place the critical <form> and </form> tags within the HTML while removing extraneous <form> tags, would verify that all form elements have appropriate names, and would give the user the opportunity to view and modify form element names as necessary or desired.

Another proposed extension to the tool is an interface for managing the database. In the prototype version of the tool, the creation, modification, and management of the database was identified as a task that could be done with existing software and was not part of the design. However, in observing the creation of complex queries, it became apparent that the ability to provide some additional information about the database structure, particularly the relationships between key fields in the various tables could reduce, or even eliminate, the complexity of one critical step in the process of creating queries in the Query Module.

One adaptation to the tool was made to address the need described above in regards to dropdown menus. The existing Dropdown Menu Module was adapted to allow both the display value, as well as the hidden value to be transmitted when the user presses a submit button. The adaptation required modifying the Drop Down Menu Module to insert additional code, particularly some JavaScript code in the pre-functional form for each Drop Down Menu.

The current version of the Save Module has certain limitations that do not allow multiple records to be saved to the same table on the same form at the same time. For example, a user might have a list of items to be edited. If the user were to create a pre-functional form with each item in a text box, all of which could be edited, the current version of the tool would not be able to accommodate that format. Structuring a project in a particular way can bypass such limitations. If the user were to place the list of items in a drop down menu, one of which could be selected and then edited, the tool would be able to accommodate that format. The overall functionality would be the same but the structuring of the project would be different. A couple of approaches to adaptation are possible. A first approach is to provide users with a style guide on how to structure their projects in a way compatible with the tool. Another approach is to further adapt the existing modules, *e.g.,* the Save Module, to accommodate additional possibilities, or to create entirely new modules. One possible new module is a Repeat Module, to address the pattern of displaying and editing a variable number of items from the same table on a single form.

The proposed extensions described above are not all encompassing. However they do represent a logical starting point for improvement and extension of this study. The CAG model does not require a complete list of all possible adaptations, only that when adaptations are made, they should be made in the direction of increasing generalization. Each of the proposed extensions above represents an adaptation in the CAG model. Each would also move the tool in the direction of being a more generalized tool. As the tool continues to be tested against the project described above as well as future projects, it is expected that additional adaptations will be required, moving the tool further in the direction of greater generalization.

Contributions of this Study to Instructional Design and Technology

The contributions of this study to the field of Instructional Design and Technology are both practical and theoretical. The development and successful use of the prototype tool described in this dissertation provides empirical evidence that it is possible for such a tool to do at least some of the work of a programmer. The prototype tool works as conceived in structured situations but requires additional development for less structured situations.

This study can also be viewed as an interdisciplinary study pulling concepts such as metaprogramming and software design patterns from the field of Computer Science and successfully integrating those concepts with principles and practices from the field of Instructional Design and Technology such as the ADDIE approach of Analysis, Design, Development, Implementation, and Evaluation. Jonassen (2004) noted that the steps of ADDIE correspond to common steps in software engineering and this study provides a example of software created using the ADDIE appraoch.

From a theoretical viewpoint this study has adapted the concept of metaprogramming from the field of computer science, where it has been focused on assisting programmers to become more efficient programmers, to the field of Instructional Design and Technology with an expanded focus on assisting non-programmers to perform some tasks that ordinarily require programming skills.

Also from a theoretical viewpoint, the identification of the patterns, the use of the Create-Adapt-Generalize (Hicks, et al., 2004) model and the modular design approach introduces a new and fruitful concept in software development. In programming a webform there are numerous ways to write programming code for different types of required functionality. Another theoretical contribution of this study is to distil this vast sea of possibilities down to a finite number of repeating patterns and, for each pattern, identify the essence of the feature addressed by that pattern, the minimum essential information that is needed from the user in order to write programming code for that feature. If a future researcher were to attempt to create a tool similar to the one in this study, the approach, programming language, interface and specific code inserted into a pre-functional form could be very different, but the patterns, and the minimal information needed by each user for each pattern identified in this study would be very similar. This study also provides a proof of concept for using the Create-Adapt-Generalize model to identify patterns and a modular design approach to develop code to address those patterns. Use of this approach to identify additional patterns or modifications to the patterns identified in this study remains for future research. Use of this model as an approach to developing additional software projects unrelated to the tool developed for this study stands as a direction for future research.

References

Association for Educational Communications and Technology. (2000). *NCATE Program Standards Initial and Advanced Programs for Educational Communications and Technology.* Retrieved from http://www.aect.org/affiliates/national/Standards.pdf

Alexander, C., Ishikawa, S., & Silverstein, M. (1977). *A Pattern Language.* New York: NY: Oxford University Press.

Barker, P., & van Schaik, P. (2010). *Electronic Performance Support: Using Digital Technology to Enhance Human Performance*. Burlington, VT: Gower Publishing Company.

Bartlett, J. (2005). The art of metaprogramming. Retrieved from http://www-128.ibm.com/developerworks/linux/library/l-metaprog3/?ca=dgr-wikiaMetaprogP3

Basili, V.R., & Turner, A.J. (1975). Iterative Enhancement: A Practical Technique for Software Development. *IEEE Transactions on Software Engineering*, 390-396.

Boehm, B.W. (1988). A Spiral Model of Software Development and Enhancement. *IEEE Computer*, 21(5), 61-72.

Brooks, F.P. (1995). *The Mythical Man-Month, Twentieth Anniversary Edition*. Addison-Wesley.

Cagiltay, K. (2001). A design and development model for building electronic performance support systems. *Annual Proceedings of Selected Research and Development [and] Practice Papers Presented at the National Convention of the Association for Educational Communications and Technology.*

Carr, C. (1992). *Smart Training: The Manager's Guide to Training for Improved Performance.* New York: McGraw-Hill, Inc.

Chamberlin, D. D., & Boyce, R. F. (1974). SEQUEL: A Structured English Query Language. *Proceedings of the 1974 ACM SIGFIDET Workshop on Data description, Access and Control.*

Chen, C., & Hongwei, X. (2005). Meta-programming through typeful code representation. *Journal of Functional Programming, 15.* 167-175.

Cordy, J.R., & Shukla, M. (1992). Practical metaprogramming. *Proceedings of the 1992 conference of the Centre for Advanced Studies on Collaborative research, 1.* 215-224.

Czarnecki, K., & Eisenecker, U. W. (1999). Components and generative programming. *Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering.* 2-19.

Dean, P.J. (1995). Examining the practice of human performance technology. *Performance Improvement Quarterly, 8*(2), 68–94.

Dick, W., Carey, L., & Carey, J. O. (2001). *The systematic design of instruction (5th ed.).* New York: Addison Wesley Longman.

Dmitriev, S. (2005). Language Oriented Programming: The Next Programming Paradigm. Retrieved from http://www.onboard.jetbrains.com/is1/articles/04/10/lop/

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patters: Elements of Reusable Object-Oriented Software.* Reading, Massachusetts: Addison-Wesley.

Gery, G. (1995). Attributes and behaviors of performance-centered systems. *Performance Improvement Quarterly 8*(1), 47-93.

Gery, G. (1991). *Electronic performance support systems: How and why to remake the workplace through the strategic application of technology.*

Gery, G., Malcolm, S., Cichelli, J., Christensen, H., Raybould, B., & Rosenberg, M.J. (2000). Performance support in internet time: The state of the practice. *Performance Improvement, 39*(6), 7-17.

Hicks, D., Potter, K., Snider, R., & Holmes, G.A. (2004). The collaborative design and development of a database management system as a tool for historical inquiry. *Journal of Computing in Teacher Education, 20*(4). 167-175.

Hunag, S.S., Zook, D., & Smaragdakis, Y. (2008). Domain-specific languages and program generation with meta-AspectJ. *ACM Transactions on Software Engineering and Methodology, 18*(2).

Jonassen, D.H. (2004). *Handbook of Research on Educational Communications and Technology (2nd Ed.)*. Mahwah, NJ: Lawrence Erlbaum, Inc.

Kobayashi, M. (2007). *Facilitating Academic Achievement in High School Interactive Television Programs by Promoting Self-Regulated Learning.* Retrieved from Dissertations and Theses database.

Koskinen, J. (2004). Metaprogramming in C++. Retrieved from http://www.cs.tut.fi/~kk/webstuff/MetaprogrammingCpp.pdf

Larson, M. (2004). *Survey and Case Study Analyses of the Professional Preparation of Instructional Design and Technology (IDT) Graduates for Different Career Environments.* Retrieved from Dissertations and Theses database.

Lowe, W., & Noga, M. L. (2002). Metaprogramming applied to web component deployment. *Electronic notes in theoretical computer science, 65*(4), 106-116.

Mileva, N., & Stanchev, I. (2000). From web-based instructional systems to browser-based electronic performance support systems. *International Journal of Continuing Engineering Education and Life Long Learning, 10*, 286-295.

Miller, B. (1996). EPSS: expanding the perspective. Retrieved from http://www.pcd-innovations.com/infosite/define.htm

Molenda, M. (2003). In search of the elusive addie model. *Performance Improvement, 42*(5), 34.

Raybould, B. (1995). Making a case for EPSS. *Innovations in Education and Training International, 32*(1), 65-69.

Raybould, B. (1996). What is an electronic performance support system? *Technical & Skills Training*, February/March.

Raybould, B. (2000). Performance support engineering: Building performance-centered web-based systems, information systems, and knowledge management systems in the 21st century. *Performance Improvement, 39*(6), 32-39.

Reiser, R. A. (2001). A History of Instructional Design and Technology: Part II: A History of Instructional Design. *Educational Technology Research and Development , 49*(2), 57-67.

Rideau, F.R., & Ban, D.V. (2001). Metaprogramming and free availability of sources. Retrieved from http://fare.tunes.org/articles/ll99/index.en.html

Rosenberg, M.J. (1988). The role of training in a performance-oriented organization. *Performance and Instruction, 27*(2), 1-6.

Rosenberg, M.J. (1990). Performance technology: Working the system. *Training, 27*(2), 42–48.

Rossett, A. (1990). Performance technology and academic programs in instructional design and technology: Must we change? *Educational Technology, 30*(8), 48-51.

Royce, W. (1970). *Managing the Development of Large Software Systems. Proceedings of IEEE WESCON, 26*(8). 1-9.

Seels, B. B., & Richey, R. C. (1994). *Instructional Technology: The definition and domains of the field.* Bloomington, IN: Association for Educational Communications and Technology.

Sleight, D. A. (1992). The potential of electronic performance support systems in schools. Retrieved from http://www.msu.edu/~sleightd/school.html

Sleight, D. A. (1993). Types of electronic performance support systems: Their characteristics and range of designs. Retrieved from http://www.msu.edu/~sleightd/epss_copy.html

Sleight, D. A. (1993). What is electronic performance support and what isn't? Retrieved from http://www.msu.edu/~sleightd/epssyn.html

Trujillo, S., Azanza, M., & Díaz, O. (2007). Generative metaprogramming. *ACM: Proceedings of the 6th international conference on Generative programming and component engineering,* New York, NY (pp. 105-114).

Visscher-Voerman, I., & Gustafson, K. L. (2004). Paradigms in the theory and practice of education and training design. *Educational Technology, Research and Development, 52*(2).

Yilmaz, H. (2007). *Identification of Academic Program Strengths and Weaknesses through Use of a Prototype Systematic Tool.* Retrieved from Dissertations and
Theses database.

Visser, E. (2002). Meta-programming with concrete object syntax. *ACM SIGPLAN/SIGSOFT: Proceedings of the Generative Programming and Component Engineering Conference,* Pittsburgh, PA (pp. 299-315).

Appendix A: Needs Analysis of Information Management Website Projects

Table A1

Needs Analysis of Five Information Management Website Projects

Needed Feature	Larson	Kobayashi	Yilmaz	VA DOE	ITMA
Accessible online from any Internet connection with a web browser	✓	✓	✓	✓	✓
Creator of the web forms can control the “look and feel” (layout, color, fonts)	✓	✓	✓	✓	✓
Users can enter data via text boxes embedded in a web form	✓	✓	✓	✓	✓
Users can select a response from a set of radio buttons	✓	✓			✓
Users can select a response from a drop down menu of predetermined options	✓	✓	✓	✓	✓
Users can create a database with appropriate tables, records and fields			✓		✓
Ability to “branch” or navigate to other web forms based on user input	✓	✓	✓	✓	✓

Needed Feature	Larson	Kobayashi	Yilmaz	VA DOE	ITMA
Require users to identify themselves with a username and password before gaining access to web forms		✓	✓	✓	✓
Determine if a user is authorized to view a web form before displaying it		✓	✓	✓	✓
Web forms can perform a database query to retrieve data		✓	✓	✓	✓
Ability to remember data retrieved from a database				✓	✓
Ability to remember data entered by the user for use on other web forms				✓	✓
Drop down menus can be populated from a database			✓	✓	✓
Drop down menu can pre-select a response based on data retrieved from a database			✓		
Drop down menu can pre-select a response based on remembered data					✓
Radio button and checkbox sets can be created from a database					✓

Needed Feature	Larson	Kobayashi	Yilmaz	VA DOE	ITMA
Ability to display remembered data on a web form				✓	✓
Text boxes can be pre-filled with responses taken from a database or from remembered data				✓	✓
Ability to save user input to a database	✓	✓	✓	✓	✓

Appendix B: Needs Addressable by Existing Software

Table B1

Needs Addressable by Existing Software

Needed Feature	**Existing Software**
Accessible online from any Internet connection with a web browser	Internet connection and a web browser such as *Firefox*, *Safari* or *Internet Explorer*.
Creator of the web forms can control the "look and feel" (layout, color, fonts)	Creator can use existing webpage editing software such as *Adobe Dreamweaver®* to create (pre-functional) web forms.
Users can enter data via text boxes embedded in a web form	Textboxes can be embedded in a web form using *Adobe Dreamweaver®* or other webpage editing software.
Users can select a response from a drop down menu of predetermined options	Dropdown menus can be embedded in a web form using *Adobe Dreamweaver®* or other webpage editing software. (Populating the list of possible responses from a database is addressed as a separate need.)
Users can select a response from a set of radio buttons or checkboxes	Radio buttons and checkboxes can be embedded in a web form using *Adobe Dreamweaver®* or other webpage editing software.
Users can create a database with appropriate tables, records and fields	Databases with appropriate tables, records and fields can be created with existing database management software such *Microsoft Access®* and *Navicat®*.

Appendix C: Patterns, Required Features and Essential Information

Table C1

Patterns, Required Features and Essential Information

Pattern	Description / Required Feature	Essential information that changes with each instance of the pattern
SAVE	Ability to save user input to a database	Which database? Which tables? For each table, which fields? For each field, what data should be saved to that field? Options are: • User responses from form elements (textbox, dropdown, radio button, checkbox) • Remembered data • Constant data • Incremented number (such as a primary key)
NAVIGATION	Ability to "branch" or navigate to other web forms based on user input	For each input element and value combination: What web page to go to? If no input element and value combination is found: What (default) web page to go to?

Pattern	Description / Required Feature	Essential information that changes with each instance of the pattern
LOGIN	Require users to identify themselves with a username and password before gaining access to web forms	In what web form textbox is the username input? In what web form textbox is the password input? In what database is the username/password stored? What table is the username/password stored in? What field is the username? What field is the password? If the username/password combo is found: • What web page to go to? • Any fields from the table to remember? If so assign a name to recall/use that data If the username/password combo is not found: • What web page to go to? • Set an error message? If so, what message?
AUTHORIZATION	Determine if a user is authorized to view a web form before displaying it	This feature is implemented by checking remembered data with a particular variable name. Which variable should be checked? If the variable is null (authorization fail), what page should the user be redirected to?

Pattern	Description / Required Feature	Essential information that changes with each instance of the pattern
QUERY	Web forms can perform a database query to retrieve data	Which database? Which tables? For each table, which fields? If more than one table selected, which fields in one table are related to which fields in another table? Which fields to filter the query by? For each field to filter by, what values should it equal or not equal? Options can be: • Constants (taken from the database) • User responses from form elements • Remembered data Of the fields, which fields to include in the final query? What fields to sort by? For each field, sort ascending or descending?
	Ability to remember data retrieved from a database	Remember data from this query? Which fields? For each field, assign a name to recall/use that data

Pattern	Description / Required Feature	Essential information that changes with each instance of the pattern
TABLE	Web forms can retrieve data from a database and display it in a table format Ability to remember data retrieved from a database	This feature depends on the QUERY feature. Which query to use? Which fields from the query to display? Which order to display them in? Which style sheet to use for the table?
REMEMBER	Ability to remember data entered by the user for use on other web forms	Which user response to remember? Assign a name to recall/use that data
DROPDOWN	Drop down menus can be populated from a database	This feature depends on the QUERY and/or REMEMBER features. For populating the dropdown: • Which query to use? • Which field from the query to use for the dropdown value? • Which fields from the query to use for the dropdown display?
	Drop down menu can pre-select a response based on data retrieved from a database	Pre-select a response from a query? If so, • Which query? • Which field?

Pattern	Description / Required Feature	Essential information that changes with each instance of the pattern
	Drop down menu can pre-select a response based on remembered data	Pre-select a response from remembered data? If so, Which variable?
RADIOSET	Radio button and checkbox sets can be created from a database	This feature depends on the QUERY feature. Which query to use? Which field from the query to use for the radio button value? Which fields from the query to use for the radio button display? How many columns in the radioset display? How many rows in the radioset display?
TEXT	Ability to display remembered data or data from a database on a web form	This feature depends on the REMEMBER, QUERY or LOGIN feature. Which variable? Where on form to display?
TEXTBOX	Text boxes can be pre-filled with responses taken from a database or from remembered data	This feature depends on the QUERY or REMEMBER feature. For queries: Which query? Which field? For remembered data: Which variable?

Appendix D: Questionnaire for FADBIT Evaluators

Questions for FADBIT Evaluators

1. Think of your experience with the tutorial project "Flix Crib". By following the instructions in the tutorial, were you able to do each of the following actions?

Actions	**Able To do?**		**Comments or difficulties**
	Yes	No	
Add a login?			
Create queries?			
Have the user's nickname appear on the screen?			
Pre-fill text boxes?			
Populate drop-down menus?			
Pre-select an option from a dropdown menu?			
Remember data from a dropdown menu?			
Pre-select an option from a set of radio buttons?			
Create a report from a query?			
Save information entered on the form?			
Navigate to other pages by pressing the correct button?			

2. Approximately how much time you spend on "Flix Crib”?

3. Were the instructions in the tutorial easy to understand and follow? If not, what difficulties did you have with the instructions in the tutorial?

4. Think of your experience with the test project “Student Management”. Using what you learned in the tutorial, were you able to do each of the following actions?

Actions	**Able To do?**		**Comments or difficulties**
	Yes	No	
Add a login?			
Create queries?			
Have the user’s first and last name appear on the screen?			
Pre-fill text boxes?			
Populate drop-down menus?			
Pre-select an option from a dropdown menu?			
Remember data from a dropdown menu?			
Pre-select an option from a set of radio buttons?			
Create a report from a query?			
Save information entered on the form?			
Navigate to other pages by pressing the correct button?			

5. Approximately how much time did you spend on "Student Management"?

6. Did the tutorial for Flix Crib prepare you to complete the Student Management project? If not, what difficulties did you have transferring what you learned in the tutorial to the Student Management project?

7. Compared to the list of computer software below, how easy or difficult did you find FADBIT to use?

	Much easier	Somewhat easier	Somewhat more difficult	Much more difficult
Web Browsing				
Email				
MS Word (word processing)				
MS Excel (spreadsheets)				
MS Access (databases)				
MS Powerpoint				
Adobe Dreamweaver (HTML Editing)				
Adobe Photoshop				
Adobe Flash				
3D modeling software				
Audio editing software				
Movie editing software				

8. How much web programming experience or training do you have?

9. How much database experience or training do you have?

10. If you were given the web forms for the Student Management project and asked to write computer code to transform those forms into a functioning project, do you think you would be able to do so?

11. If you answered “no” to question 10, do you think you would be able to learn to do so in a reasonable length of time? Given your current responsibilities, would you have the time or inclination to learn programming skills?

12. If you answered “yes” to question 10, how long do you think it would take you to write the computer code for the Student Management project?

13. If you needed to create a website similar to the “Flix Crib” or “Student Management”, would you consider using FADBIT?

14. Do you have any other comments you would like to make about FADBIT?

Appendix E: The FADBIT Prototype

FADBIT consists of a generalized framework that allows users, interacting via a web-based interface, to login, create and manage projects, and upload pre-functioning forms. The opening web page (http://www.FADBIT.com) of the interface is shown in Figure E1.

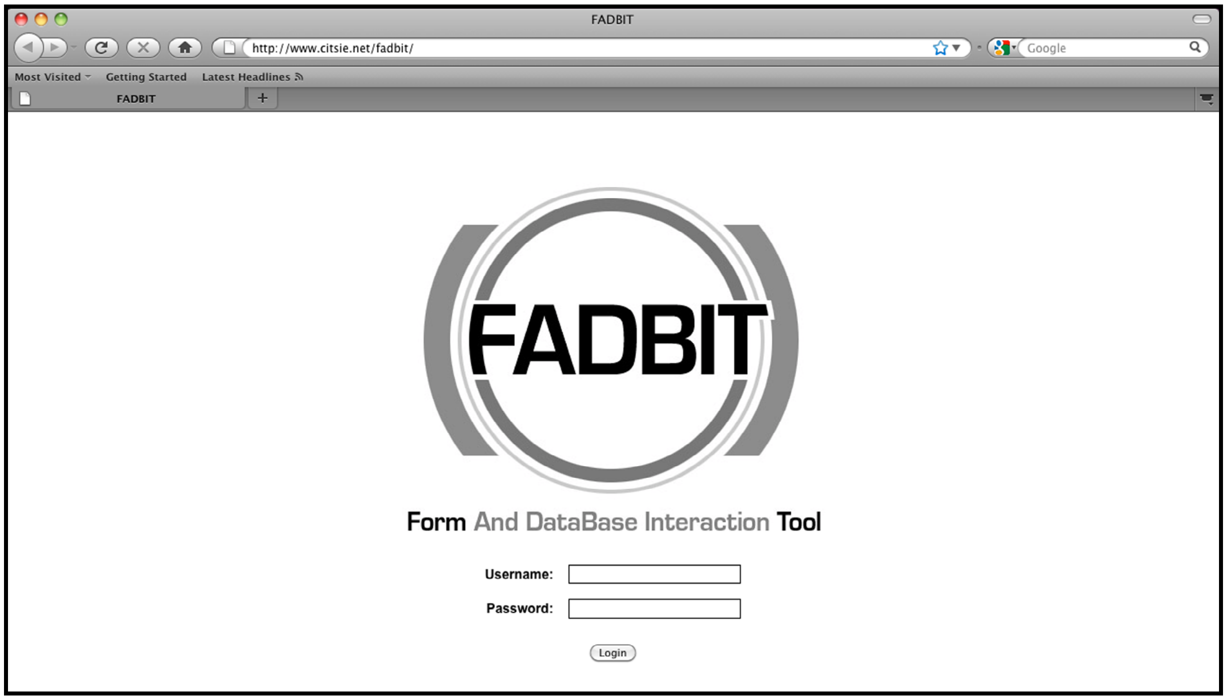


Figure E1. Opening Page of FADBIT Website

The user can login using a proper name and password combination. If the login is successful, the user is presented with the web page similar to that shown in Figure E2.

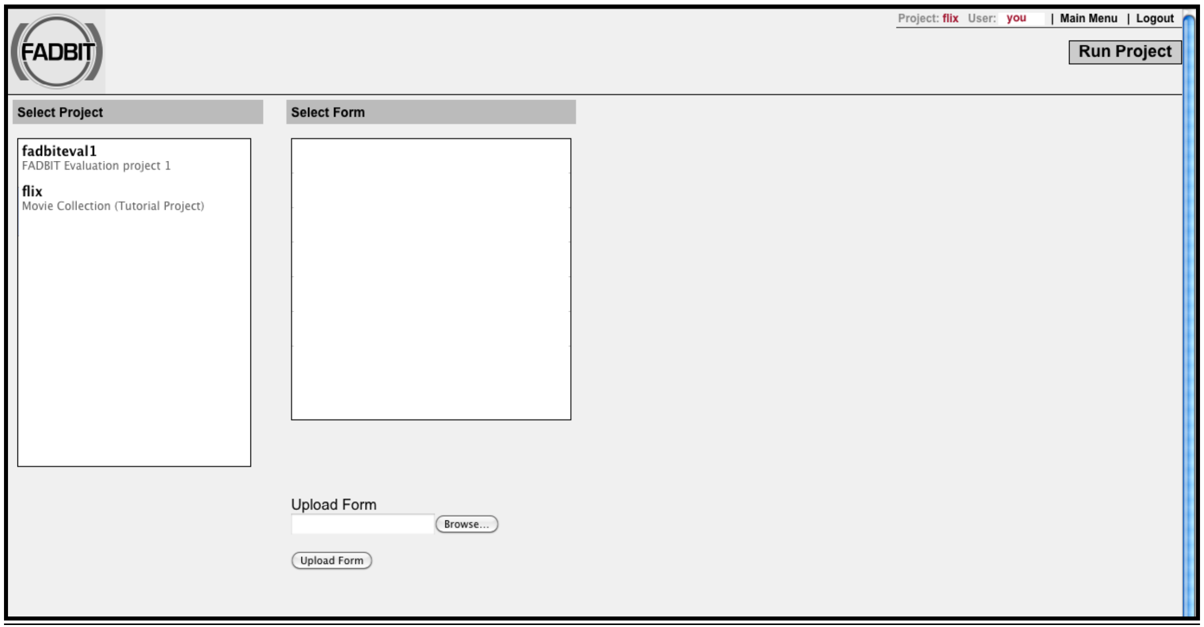

Project: flix User: you | Main Menu | Logout
Run Project
Select Project
fadbiteval1
FADBIT Evaluation project 1
flix
Movie Collection (Tutorial Project)
Select Form
Upload Form
Browse...
Upload Form

Figure E2. The Select Project Web Page of FADBIT

Here the user is presented with an option to select a project. Having made a selection of a project, the user is presented with a web page similar to that shown in Figure E3.

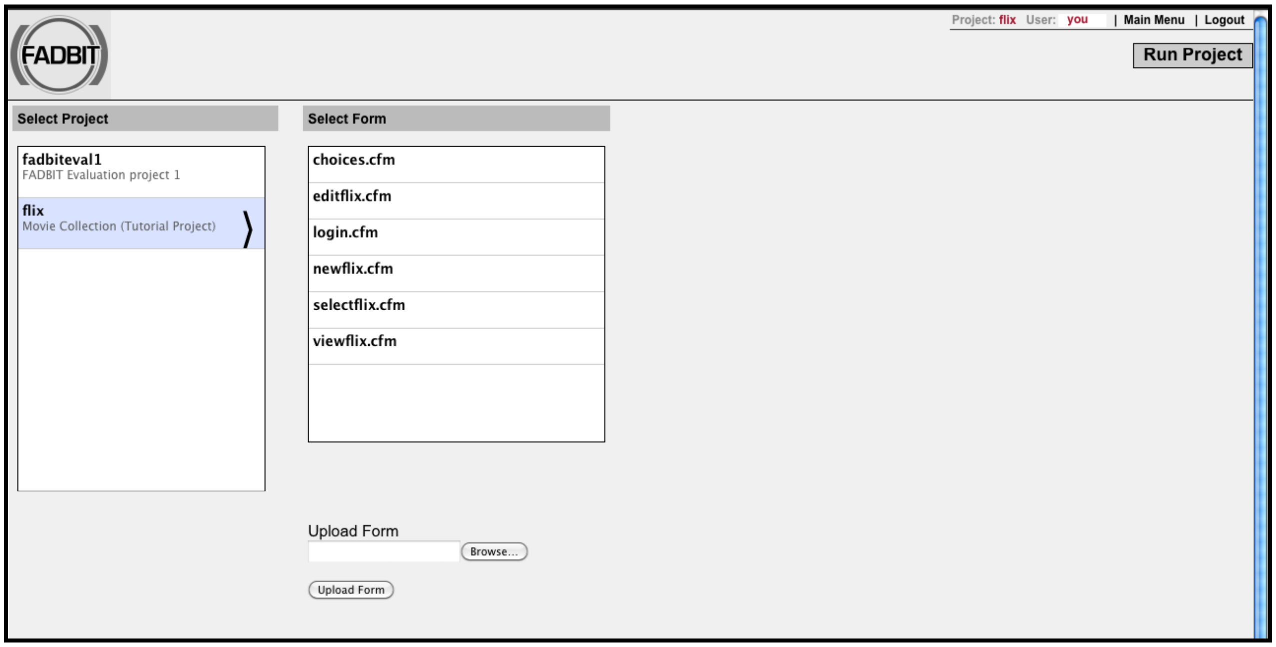


Figure E3. The Upload Form Web Page of FADBIT

On this page the user is given the option to select an existing web form for that project or upload a new web form from a file on the user's local computer. At this point in the process, either of these web forms is a non-functioning form, and after uploading, physically lies on the FADBIT server. Once the forms are situated on the FADBIT server, the user can now select one of these forms to add functionality. Because adding functionality to one form may depend on functionality having been added to a previous from, the order in which this selection process occurs is important. After the user makes a selection to add functionality, the FADBIT web page will look similar to that shown in Figure E4.

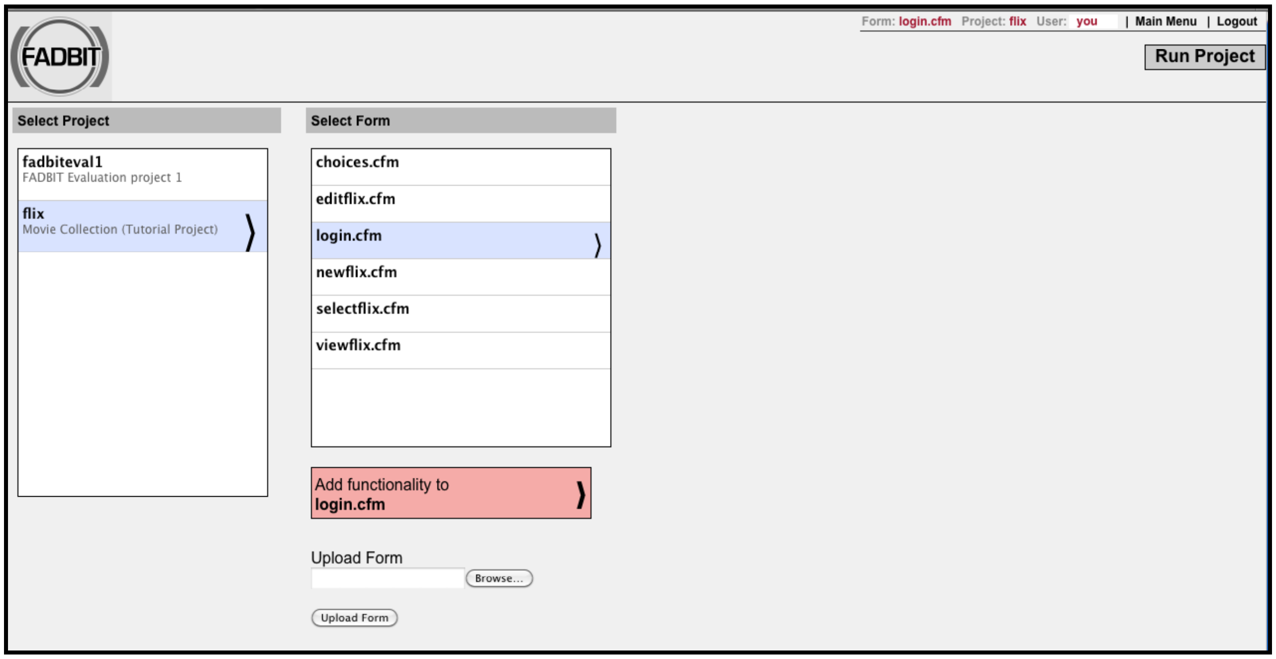
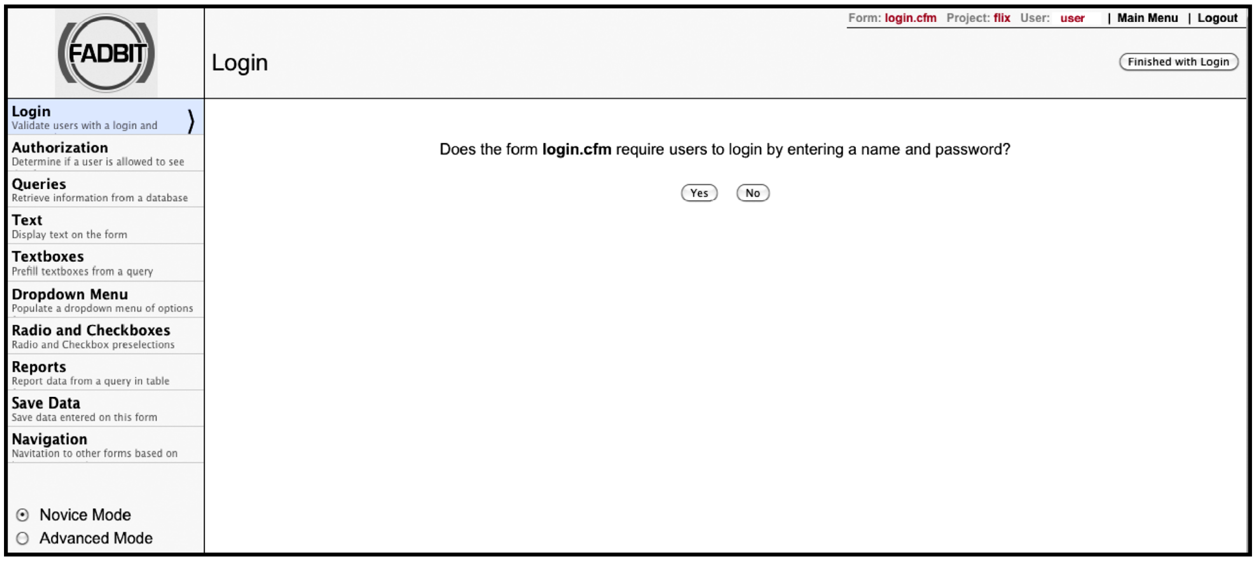


Figure E4. Web Page Showing Selection to Add Functionality

Having selected a form to which functionality will be added, the user is presented with a web page similar to that shown in Figure E5. This figure illustrates two important aspects of the current implementation of FADBIT. It clearly shows the user the list of modules (shown on the left) that are able to act on the selected, pre-functioning form. It also shows how the user may select either, novice or advanced mode.


Form: **login.cfm** Project: **flix** User: **user** | **Main Menu** | **Logout**

FADBIT

Login

Finished with Login

Login
Validate users with a login and

Authorization
Determine if a user is allowed to see

Queries
Retrieve information from a database

Text
Display text on the form

Textboxes
Prefill textboxes from a query

Dropdown Menu
Populate a dropdown menu of options

Radio and Checkboxes
Radio and Checkbox preselections

Reports
Report data from a query in table

Save Data
Save data entered on this form

Navigation
Navitation to other forms based on

[✓ Novice Mode] Advanced Mode

Does the form **login.cfm** require users to login by entering a name and password?

Yes No

Figure E5. Web Page from FADBIT

The FADBIT framework is organized around a series of *modules*. This current version of FADBIT has ten modules, Login, Authorization, Queries, Text, Textboxes, Dropdown Menus, Radio and Checkboxes, Reports, Save Data, and Navigation, as can be seen on the left in the previous figure. Each module, in the order seen above and discussed below, will examine the selected non-functioning form by asking questions of the FADBIT user and making appropriate directives. Once collected, this essential information forms a metalanguage description of the functionality the user wishes to add to the pre-functional form. The module then inserts into the pre-functional form FADBIT code containing this metalanguage description. The FADBIT code, when executed, translates or transforms this metalanguage description into Cold Fusion code. Each of these modules is briefly described below with a logic diagram illustrating the flow of logic through the module, the information collected and code written to the pre-functional form.

Login Module

The ability to require users of a web project to identify themselves with a username and password is a recurring pattern in the design of web-based information management projects. The login module is used to determine if the selected non-working form is a login page by asking the user a series of questions. If the page is supposed to be a login page, the information collected is used to add login functionality to the pre-functional form. The logical flow of the login module is shown in Figure E6.

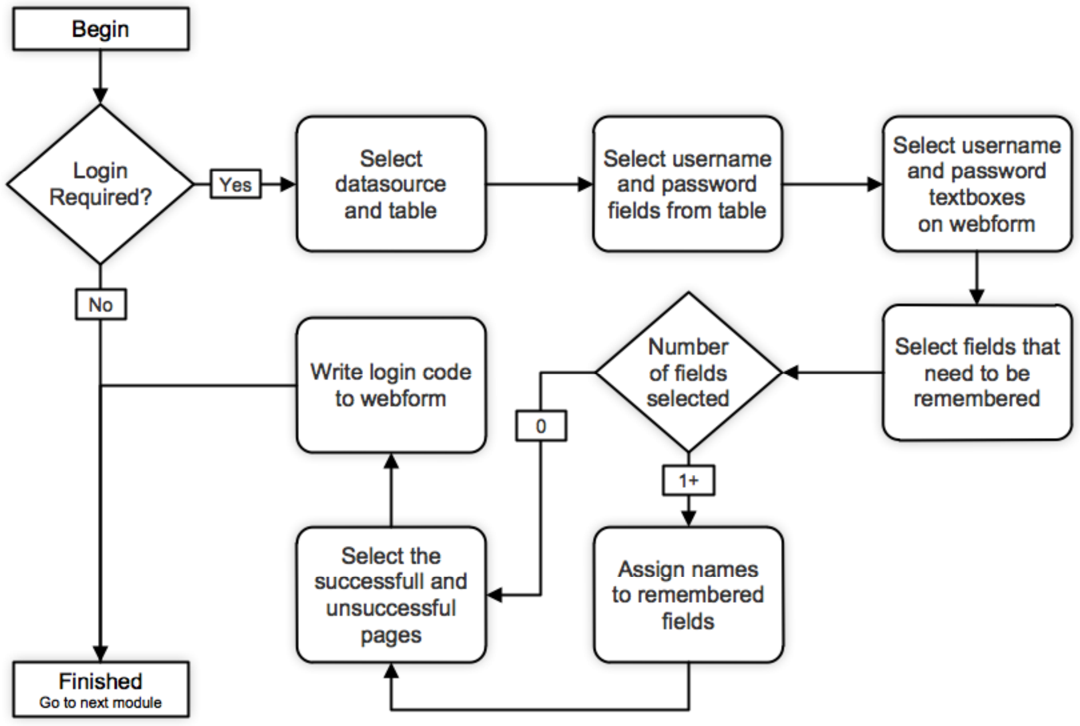

Begin
Login Required?
Yes
No
Select datasource and table
Select username and password fields from table
Select username and password textboxes on webform
Select fields that need to be remembered
Number of fields selected
0
1+
Write login code to webform
Select the successfull and unsuccessful pages
Assign names to remembered fields
Finished
Go to next module

Figure E6. Login Module Logic Diagram

Authorization Module

Another recurring pattern is the need to check if an end-user is authorized to view a specific form. There are many instances where the designer of an information management website would like to restrict access to a page to specific users. The authorization module asks the user specific questions to collect the minimum information necessary to create a functioning authorization check. The module then inserts code that checks if an end-user should be able to view the selected form or should be directed to an alternate form. The logical flow of the authorization module is shown in Figure E7.

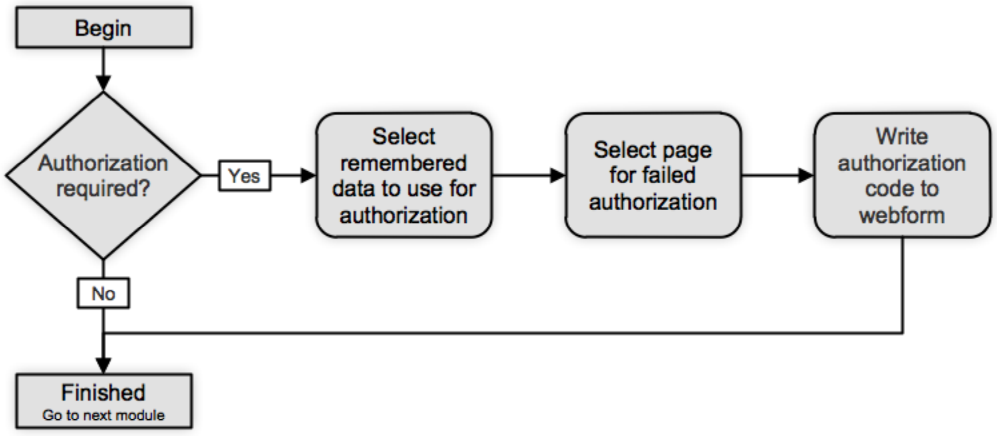

Begin
Authorization required?
Yes
Select remembered data to use for authorization
Select page for failed authorization
Write authorization code to webform
No
Finished
Go to next module

Figure E7. Authorization Module Logic Diagram

Query Module

Retrieving data from a database is a pattern found in every information management website that uses a database. Retrieving data from a database is referred to as a database *query*. In this tool, queries are written in a programming language called *Structured Query Language*. The query module allows users who are not familiar with Structured Query Language to add one or more queries to a pre-functional form by responding to a series of questions concerning what data the end-user will want to retrieve. The query module then inserts code into the pre-functional form that will retrieve that data. This inserted code ultimately translates the FADBIT user's answers to the query module questions into Structured Query Language programming code. The user, however, never sees such code. The query, once inserted becomes available for use in other modules, such as populating a drop down box of items, pre-filling data in text box, or displaying data in a report. The user can add multiple queries to the same form for different purposes. The logical flow of the query module is shown in Figure E8.

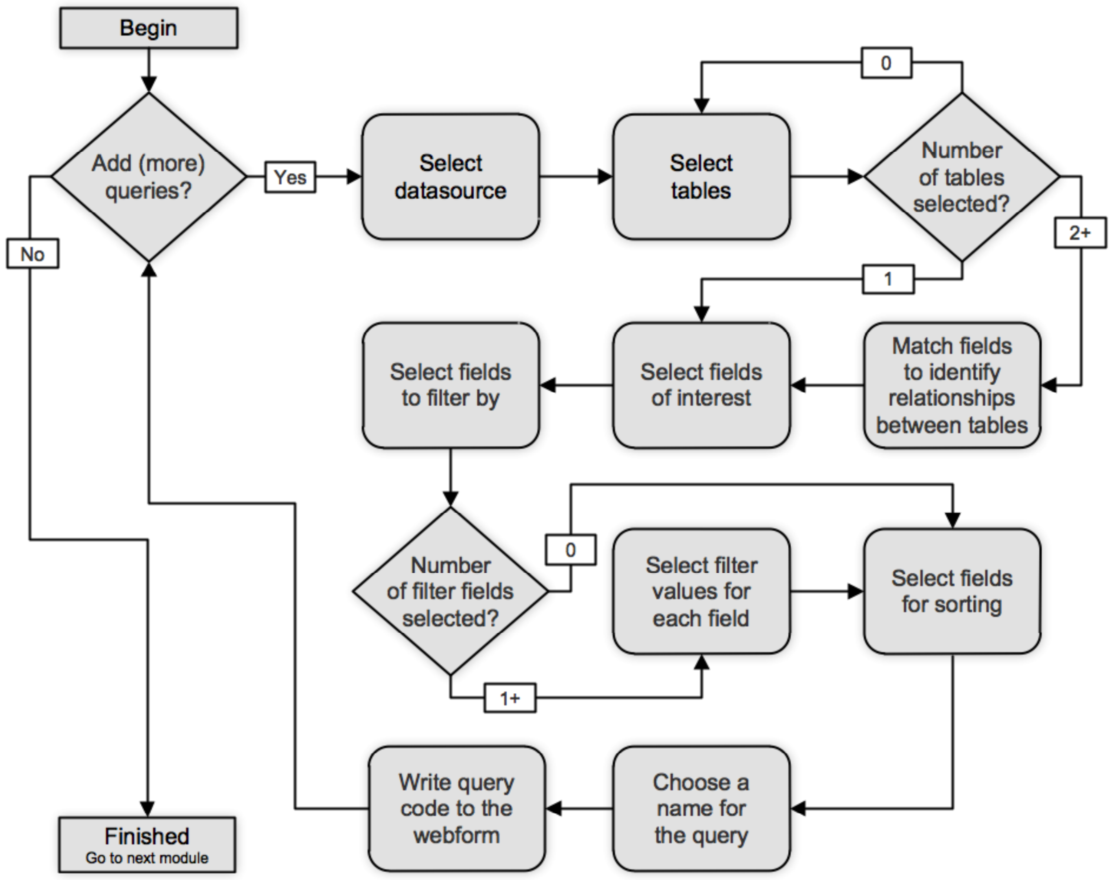
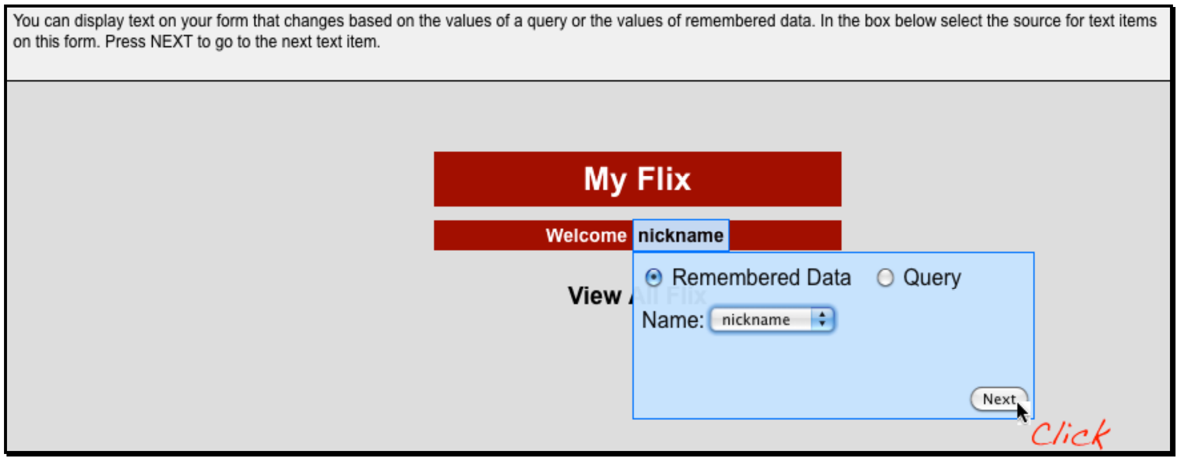

Begin
Add (more) queries?
Yes
No
Select datasource
Select tables
Number of tables selected?
0
1
2+
Match fields to identify relationships between tables
Select fields of interest
Select fields to filter by
Number of filter fields selected?
0
1+
Select filter values for each field
Select fields for sorting
Choose a name for the query
Write query code to the webform
Finished
Go to next module

Figure E8. Query Module Logic Diagram

Text Display Module

A pattern often found in an information management website is the need to display text that may change from end-user to end-user. The text display is based on the result of a query or, as coined in FADBIT, "remembered" data. A partial screen shot of the Text Module in action is shown in Figure E9, and its logic diagram in Figure E10.



You can display text on your form that changes based on the values of a query or the values of remembered data. In the box below select the source for text items on this form. Press NEXT to go to the next text item.

Figure E9. Partial Screen Shot of the Text Module in Action

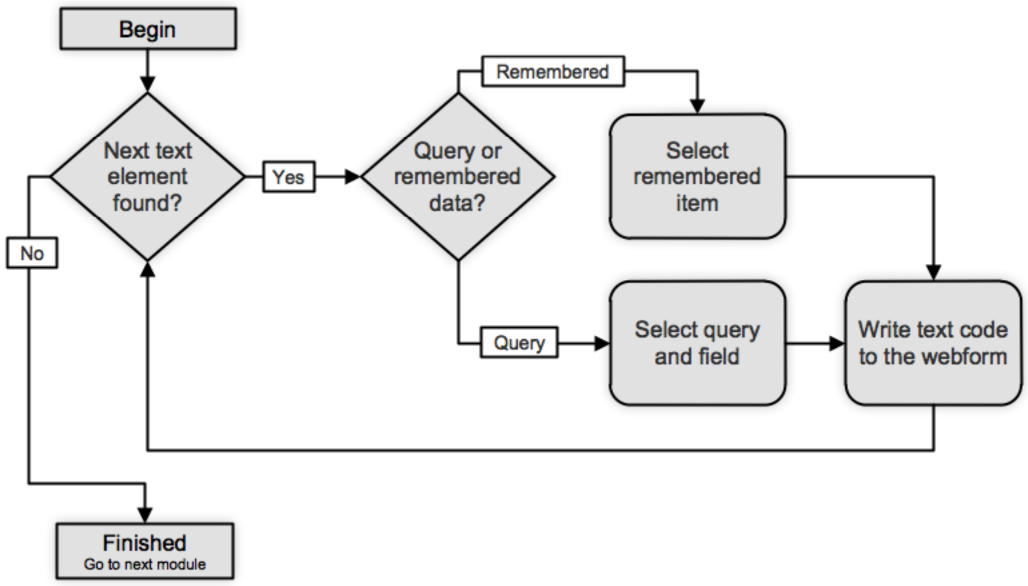


Figure E10. Text Display Module Logic Diagram

Textbox Entry Module

A common element in webforms is the textbox. Textboxes can be used to enter data for a variety of purposes. In the programming of webforms, three common programming patterns related to textboxes are pre-filling textboxes with data, setting variables to a value entered in a textbox (what FADBIT calls remembering data) and saving data entered in a textbox to a database. The FADBIT textbox module addresses the first two of these patterns. The FADBIT Save Module addresses the third of these patterns, saving data to a database. The textbox module works by parsing the pre-functional form, identifying each of the textboxes and asking the user for the essential information needed for each box to add the functionality the user wishes to add. A blue interface box appears underneath each text box where the user can enter information about the functionality required for that textbox. A partial screen shot of an example of the textbox interface box is shown in Figure E11. The logical flow of the Textbox Module is show in Figure E12.

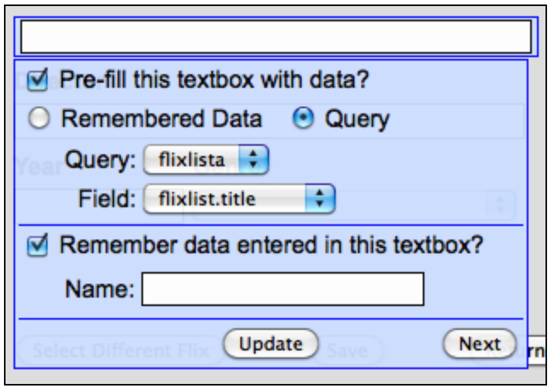

☑ Pre-fill this textbox with data?
○ Remembered Data [✓ Query]
Query: flixlista
Field: flixlist.title
☑ Remember data entered in this textbox?
Name:

Figure E11. Textbox Entry Module Interface Box

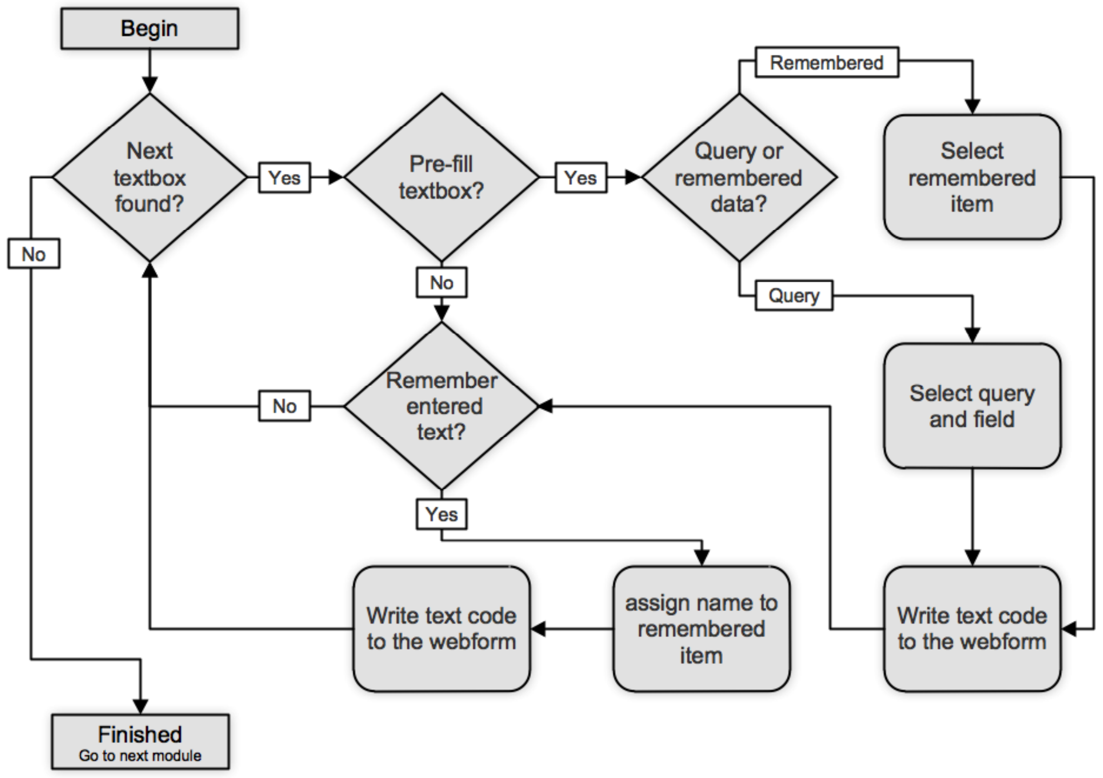
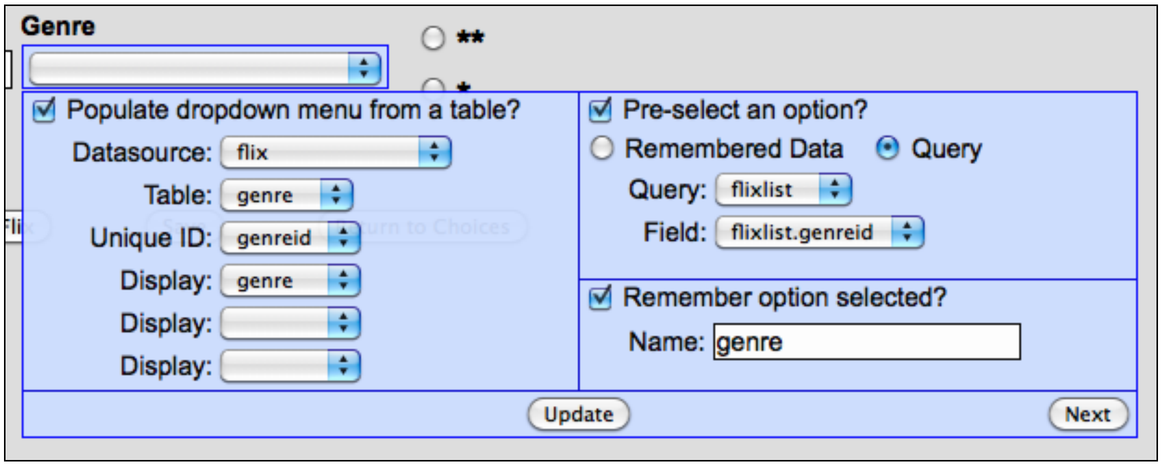


Figure E12. Textbox Entry Module Logic Diagram

Dropdown Menu Module

Another common element in webforms is the dropdown menu. There are instances when the designer of an information management website would like a dropdown menu to be populated with a list of options when the webform is displayed at runtime and one of the options in that menu to be pre-selected. The dropdown menu module asks the FADBIT user relevant questions to collect the information necessary to populate the dropdown menu and pre-select an option. The module then inserts code that will fill the dropdown menu as the web browser displays the page to the end-user. Another pattern is to remember the option selected from a dropdown list for use elsewhere in a project. A partial screen shot of an example of the dropdown menu interface box is shown in Figure E13. The logical flow of the dropdown menu module is shown in Figure E14.



☑ Populate dropdown menu from a table?

Datasource: flix
Table: genre
Unique ID: genreid
Display: genre
Display:
Display:

☑ Pre-select an option?

○ Remembered Data ● Query

Query: flixlist
Field: flixlist.genreid

☑ Remember option selected?

Name: genre

Figure E13. A Partial Screen Shot of the Dropdown Menu Interface Box

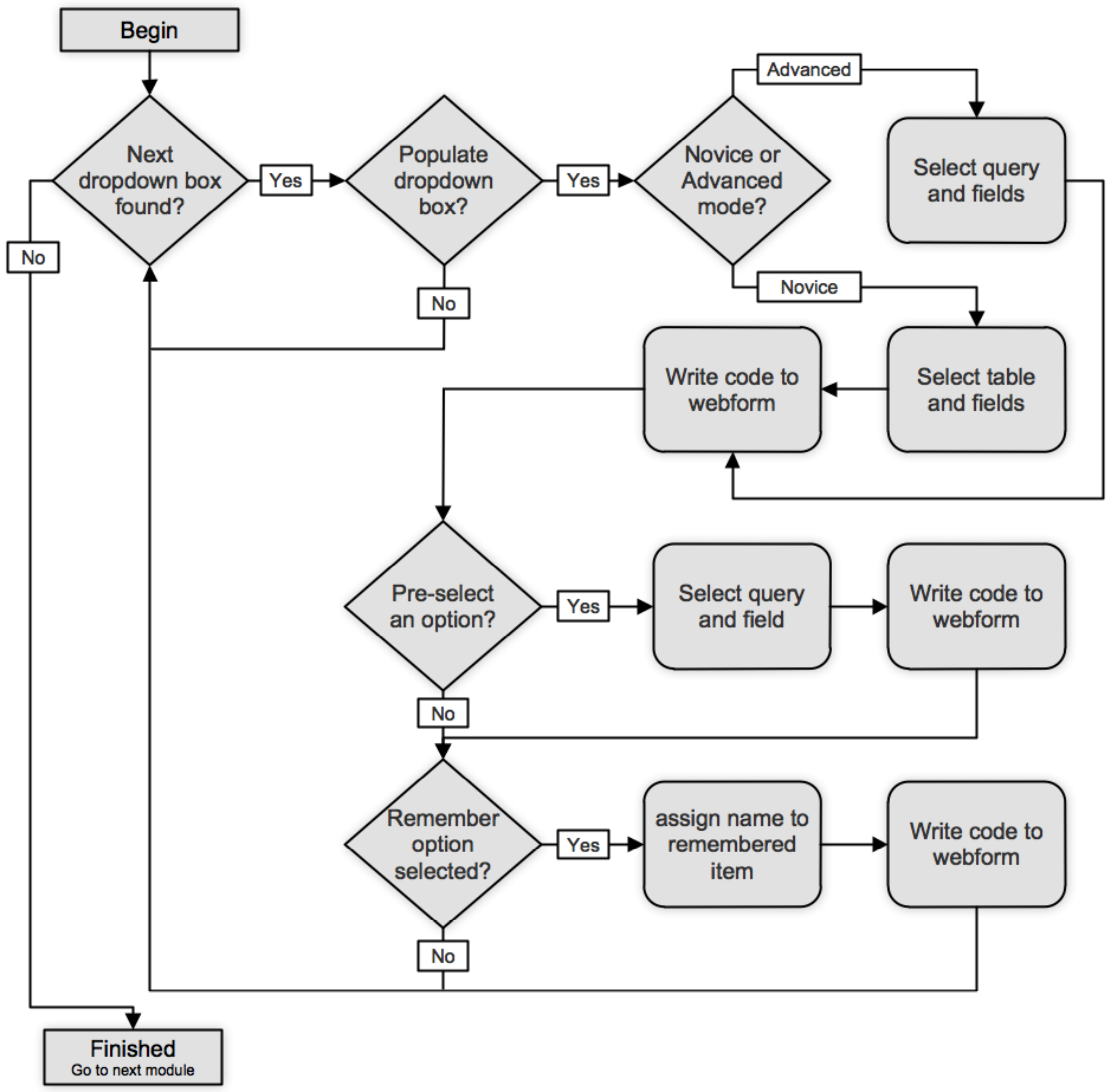


Figure E14. Dropdown Menu Module Logic Diagram

Radio Checkbox Module

The designer of an information management website may like a radio checkbox to be pre-selected when the webform is displayed at runtime. The radio checkbox menu module asks the FADBIT user for the query and field that are to be used in that pre-selection. The module then inserts code that will make the proper selection when the web browser displays the page to the end-user. A partial screen shot of an example of the radio checkbox module interface box is shown in Figure E15. The logical flow of the radio checkbox module is shown in Figure E16.

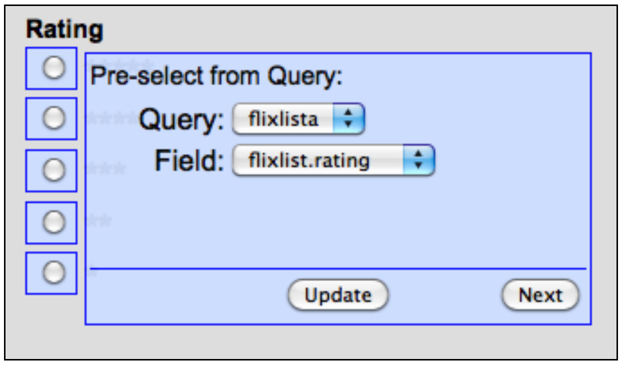


Figure E15. A Partial Screen Shot of the Radio Checkbox Interface Box

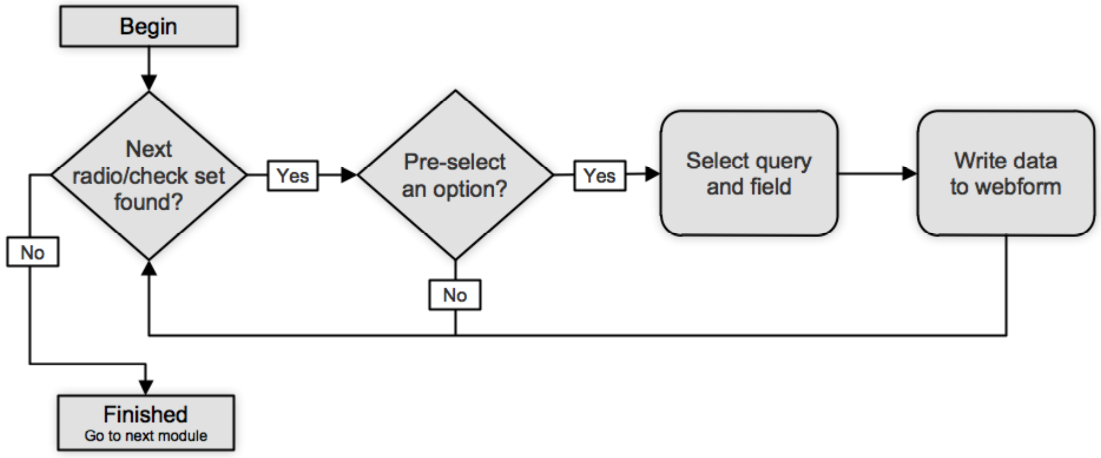


Figure E16. Radio/Checkbox Set Module Logic Diagram

Reports Module

The designer of an information management website may like certain data to be displayed in a table form. The term "reports" is used in FADBIT for this situation. In this case, the designer must explicitly place a "reports tag" in the original HTML coding to make FADBIT aware of the intent to display these data. Unlike some of the form elements described above, such as the text box or dropdown menu, there is no standard HTML tag for "report", so FADBIT introduces a report tag of <report> using the standard HTML tag convention of angled brackets. The reports module then looks for this tag in the HTML code as an indicator of where the user wishes to insert a report. The reports menu module asks the FADBIT user for the query and fields that are to be used in the report. The module then inserts code that will write the report when the web browser displays the page to the end-user. The logical flow of the report module is shown in Figure E17.

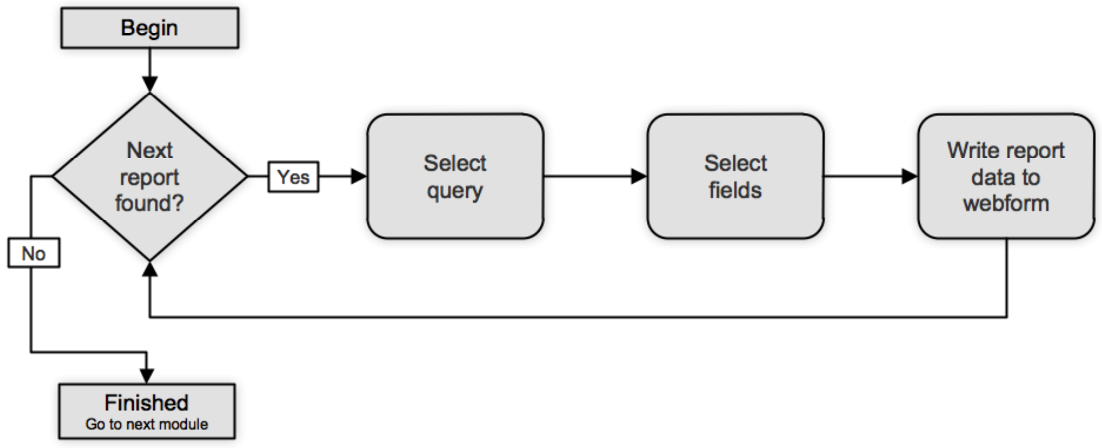


Figure E17. Report Module Logic Diagram

Save Module

Saving data to a database table is another recurring pattern in the programming of information management websites. The need to save data that have been entered, edited, altered, or collected is obvious. The save module asks the FADBIT user to indicate the appropriate tables and fields to which data is to be saved. The module then inserts code that will save data to the appropriate tables. The logical flow of the save module is shown in Figure E18.

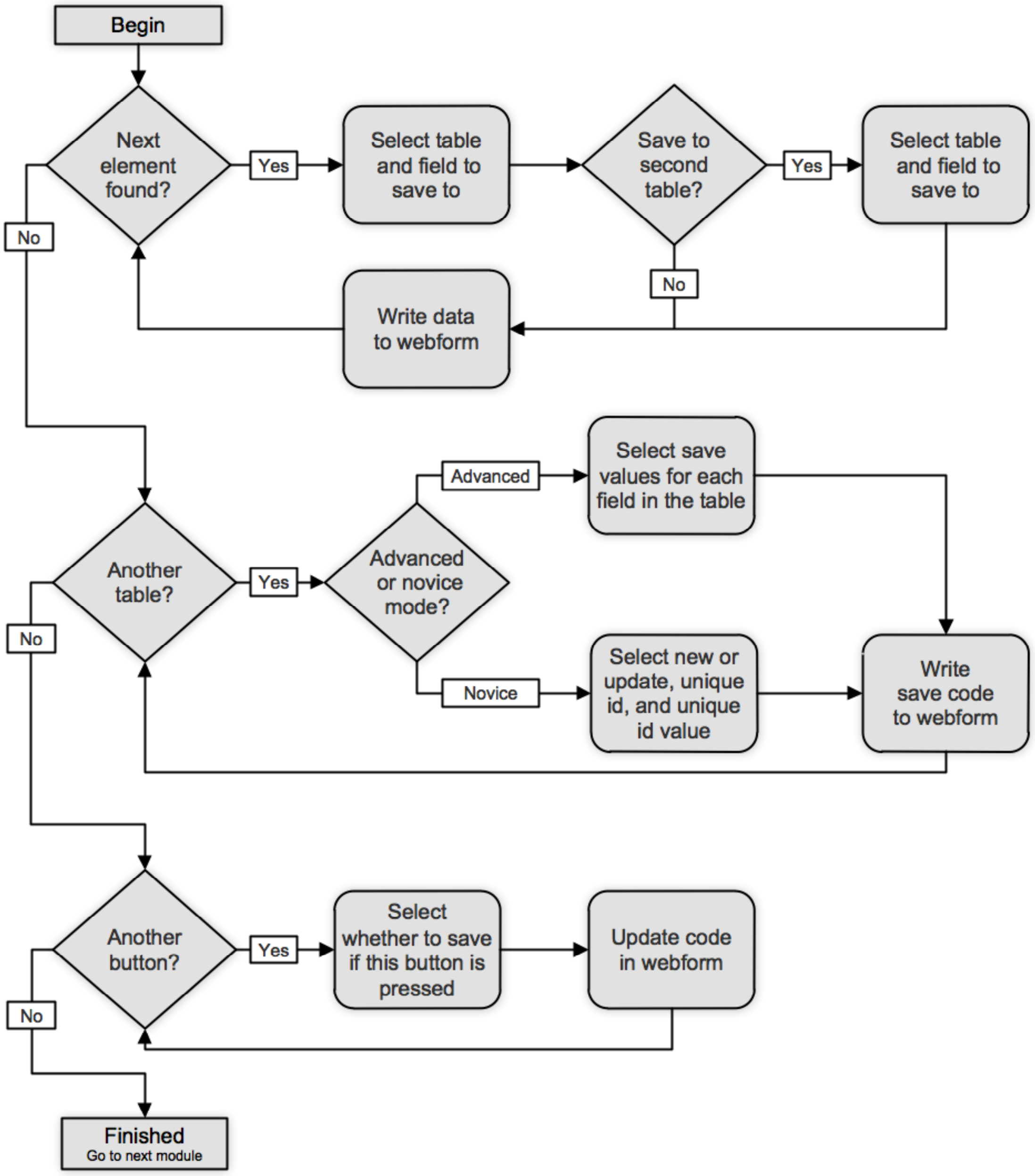

Begin
Next element found?
Yes
Select table and field to save to
Save to second table?
Yes
Select table and field to save to
No
Write data to webform
No
Another table?
Yes
Advanced or novice mode?
Advanced
Select save values for each field in the table
Novice
Select new or update, unique id, and unique id value
Write save code to webform
No
Another button?
Yes
Select whether to save if this button is pressed
Update code in webform
No
Finished
Go to next module

Figure E18. Save Module Logic Diagram

Navigation Module

A typical information management website (project) consists of several related webform (pages). The navigation module asks the FADBIT user to indicate which page the web browser should display when a navigation button is pushed. The navigation module then inserts code that directs the flow appropriately from page to page. The logical flow of the navigation module is shown in Figure E19.

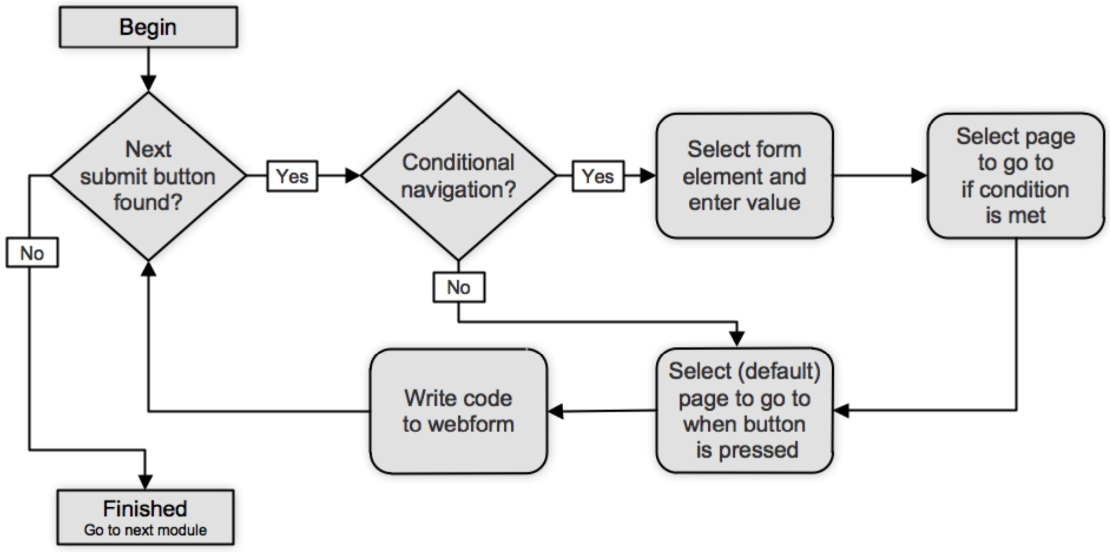
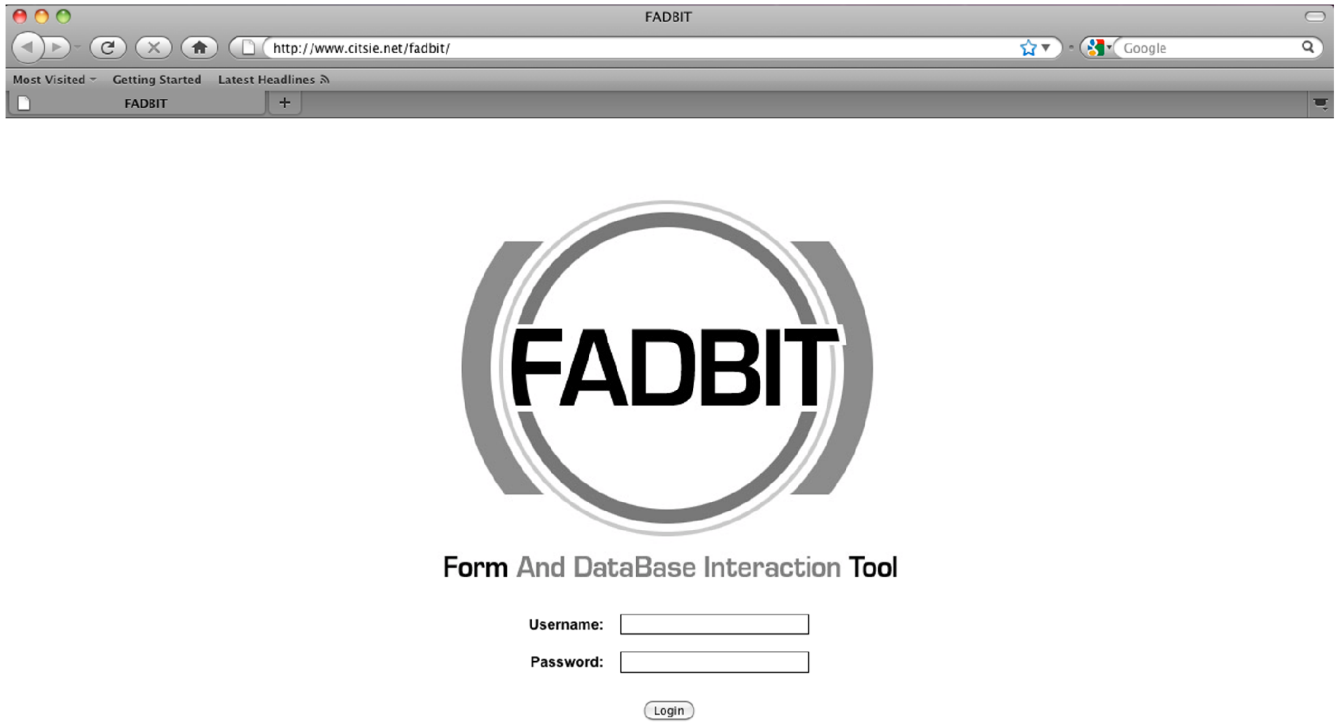

Begin
Next submit button found?
Yes
No
Conditional navigation?
Yes
No
Select form element and enter value
Select page to go to if condition is met
Select (default) page to go to when button is pressed
Write code to webform
Finished
Go to next module

Figure E19. Navigation Module Logic Diagram

Appendix F: Representative Pages of the FADBIT User's Guide

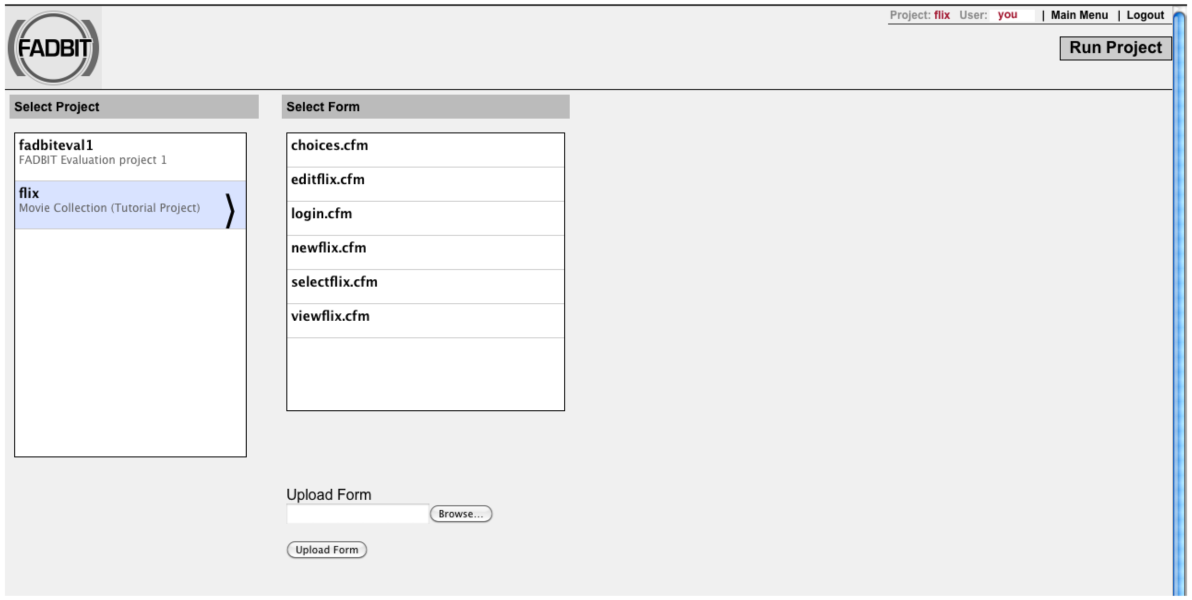
Step 4. Opening FADBIT

Open a web browser and type the address of the FADBIT web site into the address slot: http://www.fadbit.com. You will see a screen that looks like the following.



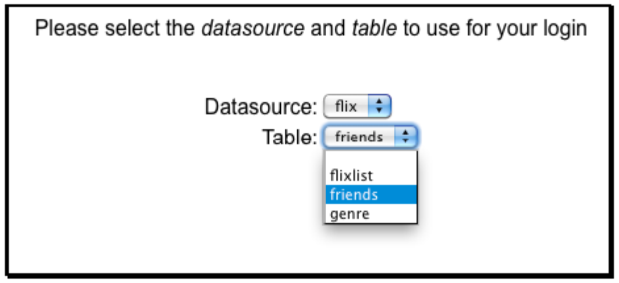
Type in __________________ as your username and _________________ as your password and then click on the "Login" button. (This username and password pair is unique to you.) You will now see a screen that looks something like the following.

You now need to upload your project files. Click on the “Browse” button and locate, say, the login.html file. Remember that this file is in the “tutorial” folder on your desktop. Now click the “Upload Form” button. Repeat this procedure until all of the project files are uploaded. Your screen should now look something like the following.



Note that the file extension of each of the files has been changed to .cfm. This stands for cold fusion markup (language) of which html is a subset. This is the markup language that FADBIT uses. Nothing else at this point has been changed. The procedure now is to let the FADBIT modules add functionality to each of the pre-functional forms. Because some of these forms depend on other forms, you, as the imagined designer, have “ordered” the forms according to the following designer sketch (diagram).

Screen 2 Adding Functionality to login.cfm. (*Login Module*)


Please select the *datasource* and *table* to use for your login

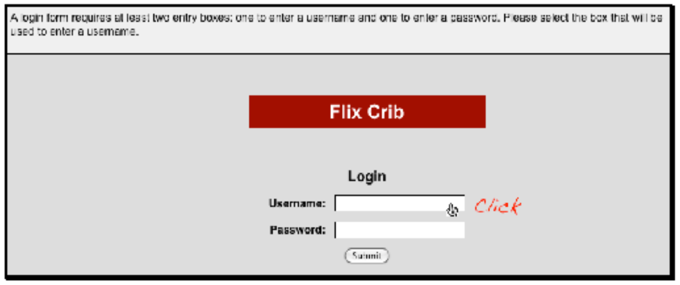
Datasource: flix

Table: friends

Instruction: Make the selections indicated above.

Explanation: A login works by comparing the username and password entered on a form to list of valid username/password combinations in a database. In order to add login functionality to the form, FADBIT needs to know which *datasource* and *table* contain the username/password combinations. As indicated above, the datasource (database) for the "flix" project is named "flix" and the *table* **friends** contain the data to check for users.

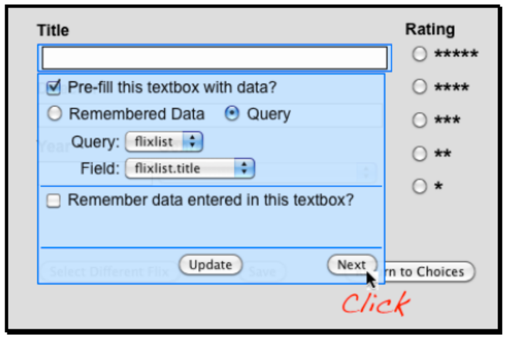
Screen 3 Adding Functionality to login.cfm. (*Login Module*)


A login form requires at least two entry boxes: one to enter a username and one to enter a password. Please select the box that will be used to enter a username.

Instruction: Make the selection indicated above

Explanation: On a login page, users will enter their usernames in a textbox and their passwords in another textbox. FADBIT needs to know which box is the username box and which box is the password box. FADBIT will ask you to identify the boxes by clicking on them.

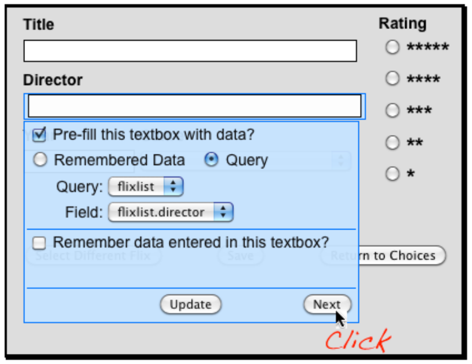
Screen 15 Adding Functionality to editflix.cfm. (*Textboxes Module*)


Title
Rating
☑ Pre-fill this textbox with data?
○ Remembered Data ● Query
Query: flixlist
Field: flixlist.title
☐ Remember data entered in this textbox?
Update
Next
rn to Choices
*****
****
***
**
*
Click

Instruction: Make selections indicated above.

Explanation: The textbox labeled “Title” is pre-filled from the *query* flixlist with the *field* title.

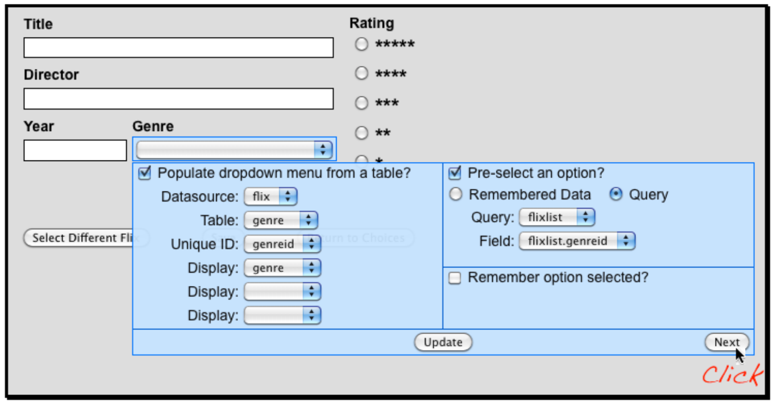
Screen 16 Adding Functionality to editflix.cfm. (*Textboxes Module*)



Instruction: Make selections indicated above.

Explanation: The textbox labeled “Director” is pre-filled from the *query* flixlist with the *field* director.

Screen 19 Adding Functionality to editflix.cfm. (*Dropdown Menu Module*)



Title

Director

Year

Genre

Rating
○ *****
○ ****
○ ***
○ **

☑ Populate dropdown menu from a table?
Datasource: flix
Table: genre
Unique ID: genreid
Display: genre
Display:
Display:

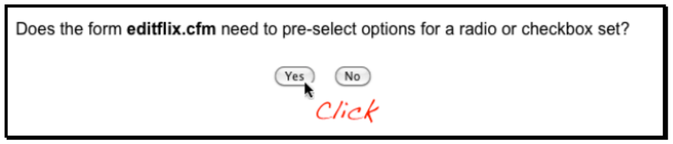
☑ Pre-select an option?
○ Remembered Data [✓ ● Query]
Query: flixlist
Field: flixlist.genreid

☐ Remember option selected?

Instruction: Make the selections indicated above.

Explanation: The menu needs to be populated with a list of genre so that the end-user can select one to edit. Checking the "Populate this dropdown menu box from a table?" opens a list of options. The *datasource* for "Flix Crib" is named flix. The *table* containing the list of genre is named genre. The *unique ID* for genre is name genreid. The end-user will select the genre. The designer (or FADBIT user) needs to pre-select an option from the *query* flixlist and the *field* genreid. Because the form **editflix.cfm** is intended to edit an existing *record* in the *database*, the genre dropdown menu should pre-select the option already in that *record*.

Screen 20 Adding Functionality to editflix.cfm. (*Radio and Checkbox Sets Module*)

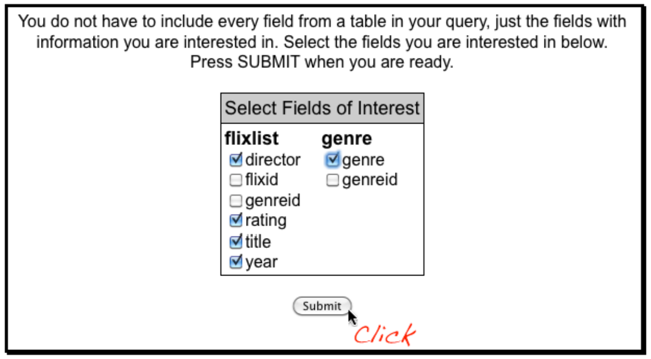


Does the form **editflix.cfm** need to pre-select options for a radio or checkbox set?

Instruction: Make the selection indicated above.

Explanation: The "Rating" radio button set needs to be pre-selected

Screen 9 Adding Functionality to viewflix.cfm. (*Queries Module*)

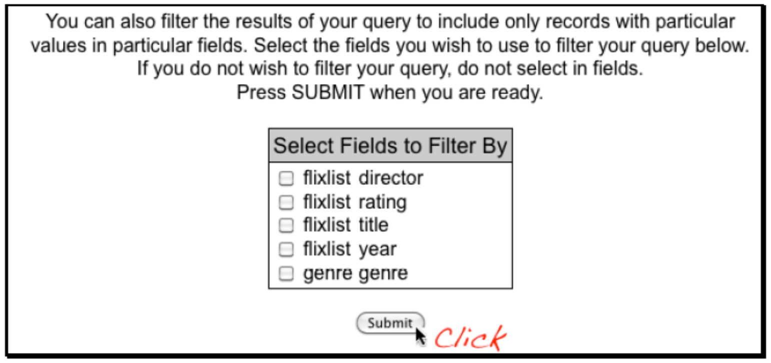


You do not have to include every field from a table in your query, just the fields with information you are interested in. Select the fields you are interested in below.
Press SUBMIT when you are ready.

Instruction: Make the selections indicated above.

Explanation: These are the *fields* to be displayed in a "Report".

Screen 10 Adding Functionality to viewflix.cfm. (*Queries Module*)



You can also filter the results of your query to include only records with particular values in particular fields. Select the fields you wish to use to filter your query below.
If you do not wish to filter your query, do not select in fields.
Press SUBMIT when you are ready.

Instruction: Make the selections indicated above.

Explanation: There is no need to filter records here.

Appendix G: Guided Project Responses

Table G1

Experience With the Guided Project

Think of your experience with the tutorial project “Flix Crib”. By following the instructions in the tutorial, were you able to do each of the following actions?						
	Evaluator					
	1	2	3	4	5	6
Add a login?	Y	Y	Y	Y	Y	Y
Create queries?	Y	Y	Y	Y	Y	Y
Have the user’s nickname appear on the screen?	Y	Y	Y	Y	Y	Y
Pre-fill text boxes?	Y	Y	Y	Y	Y	Y
Populate drop-down menus?	Y	N	Y	Y	Y	Y
Pre-select an option from a dropdown menu?	Y	Y	Y		Y	Y
Remember data from a dropdown menu?	Y	Y	Y		Y	Y
Pre-select an option from a set of radio buttons?	Y	N			Y	Y
Create a report from a query?	Y	Y	Y	Y	Y	Y
Save information entered on the form?	Y	Y	Y	Y	Y	Y
Navigate to other pages by pressing the correct button?	Y	Y	Y	Y	Y	

Y = Yes
N = No
(Blank) = Not Answered

Table G2

Time Spent on the Guided Project

Approximately how much time did you spend on "Flix Crib”?	
Evaluator 1	“About 1 hour”
Evaluator 2	“45 minutes”
Evaluator 3	“1 hour”
Evaluator 4	“… about an hour”
Evaluator 5	“40-50 min”
Evaluator 6	“3 hrs”

Table G3

Ease of the Guided Project Tutorial

Were the instructions in the tutorial easy to understand and follow? If not, what difficulties did you have with the instructions in the tutorial?	
Evaluator 1	"Yes, they were easy."
Evaluator 2	"Yes, very detailed"
Evaluator 3	"Quite easy to follow"
Evaluator 4	"I felt the instructions in the tutorial were initially a bit difficult to understand. I missed the information/instructions at the top of the screen. However, this could be in part because the font was small, and because I was focused on the main part of the window." [a]
Evaluator 5	"The tutorial was so clear and easy to follow"
Evaluator 6	"Instructions were easy to understand and follow, and once you get to understand the work flow, there may be no need for instructions because you build everything based on what outcome you'd like."

[a]This evaluator did, however, successfully complete the tutorial.

Appendix H: Unguided Project Responses

Table H1

Experience with the Unguided Project

Think of your experience with the test project "Student Management". Using what you learned in the tutorial, were you able to do each of the following actions?

	Evaluator					
	1	2	3	4	5	6
Add a login?	Y	Y	Y	Y	Y	Y
Create queries?	Y	Y	Y	Y	Y	Y
Have the user's nickname appear on the screen?	Y	Y	Y	Y	Y	Y
Pre-fill text boxes?	Y	Y	Y	Y	Y	Y
Populate drop-down menus?	Y	N	Y	Y	Y	Y
Pre-select an option from a dropdown menu?	Y	Y	Y		Y	Y
Remember data from a dropdown menu?	Y	Y	Y		Y	Y
Pre-select an option from a set of radio buttons?		N	Y		Y	Y
Create a report from a query?	Y	Y	Y	Y	Y	Y
Save information entered on the form?	Y	Y	Y	Y	Y	Y
Navigate to other pages by pressing the correct button?	Y	Y	Y	Y	Y	

Y = Yes
N = No
(Blank) = Not Answered

Table H2

Time Spent on Unguided Project

Approximately how much time did you spend on "Student Management"?	
Evaluator 1	"About 2 hours"
Evaluator 2	"50 minutes"
Evaluator 3	"2 hours"
Evaluator 4	"At least an hour, if not more."
Evaluator 5	"40-50 min"
Evaluator 6	"4hrs"

Table H3

Transfer from the Guided Project to the Unguided Project

Did the tutorial for Flix Crib prepare you to complete the Student Management project? If not, what difficulties did you have transferring what you learned in the tutorial to the Student Management project?	
Evaluator 1	"Yes."
Evaluator 2	"Yes, the Flix Crib set a very good foundation and prepared me for the Students Management project."
Evaluator 3	"Yes. I was just not very clear on how to create a report from a query."
Evaluator 4	"I followed the steps in the manual while working through the tutorials. I would have liked to see more contextual information provided in the manual for both projects. I believe this would have made the process more relevant and helped me understand exactly what the tool was doing to help me."
Evaluator 5	"Yes."
Evaluator 6	"The "Flix Crib" tutorial was a great help, at times I referred to the diagrams on "Flix Crib" to help me navigate and make appropriate choices for Student management"

Appendix I: FADBIT Compared to Other Software

Table I1

FADBIT Compared to Other Software

Compared to the list of computer software below, how easy or difficult did you find FADBIT to use?

	Evaluator						
	1	2	3	4	5	6	Mean
Web Browsing	3	4	3	3	2	1	2.7
Email	3	4	3	3	2	1	2.7
MS Word® (word processing)	3	3	3	3	2	1	2.5
MS Excel® (spreadsheets)	3	3	2	3	2	1	2.3
MS Access® (databases)	2	3	2	2	2	2	2.2
MS PowerPoint®	2	3		3	2	1	2.2
Adobe Dreamweaver® (HTML Editing)	2	2	1	2	2	1	1.7
Adobe Photoshop®	2	2	1	2	2	1	1.7
Adobe Flash®	2	2	1	2	2	2	1.8
3D modeling software		2	1		2	1	1.5
Audio editing software		3	1	3	2	1	2.0
Movie editing software		3	1	3	2	1	2.0

1 = Much easier
2 = Somewhat easier
3 = Somewhat more difficult
4 = Much more difficult

Appendix J: Evaluators' Programming and Database Experience

Table J1

Evaluators' Programming Experience

	How much web programming experience or training do you have?
Evaluator 1	"Very little."
Evaluator 2	"Very few."
Evaluator 3	"Not much. Can just create simple web pages."
Evaluator 4	"I was fairly well versed in HTML and CSS, but haven't used it much in the past 5 years."
Evaluator 5	"About 10 years"
Evaluator 6	"Basic"

Table J2

Evaluators' Database Experience

	How much database experience or training do you have?
Evaluator 1	"Very little."
Evaluator 2	"Very few."
Evaluator 3	"I have gone through some training but have not really created databases. I have experience in database administration, and installation of open source database Library database management system which consists of dependencies such as mysql server, apache and some perl modules."
Evaluator 4	"3 days of training with Microsoft Access. Because of this training 10 years ago, I have been able to interact with databases to extract information. Primarily, my experience has been with the needs for a database and the users' interaction with the database. I only played with building one."
Evaluator 5	"Around 5 years"
Evaluator 6	"Intermediate"

Appendix K: Evaluators' Additional Responses

Table K1

Evaluators' Consideration of FADBIT for Similar Projects

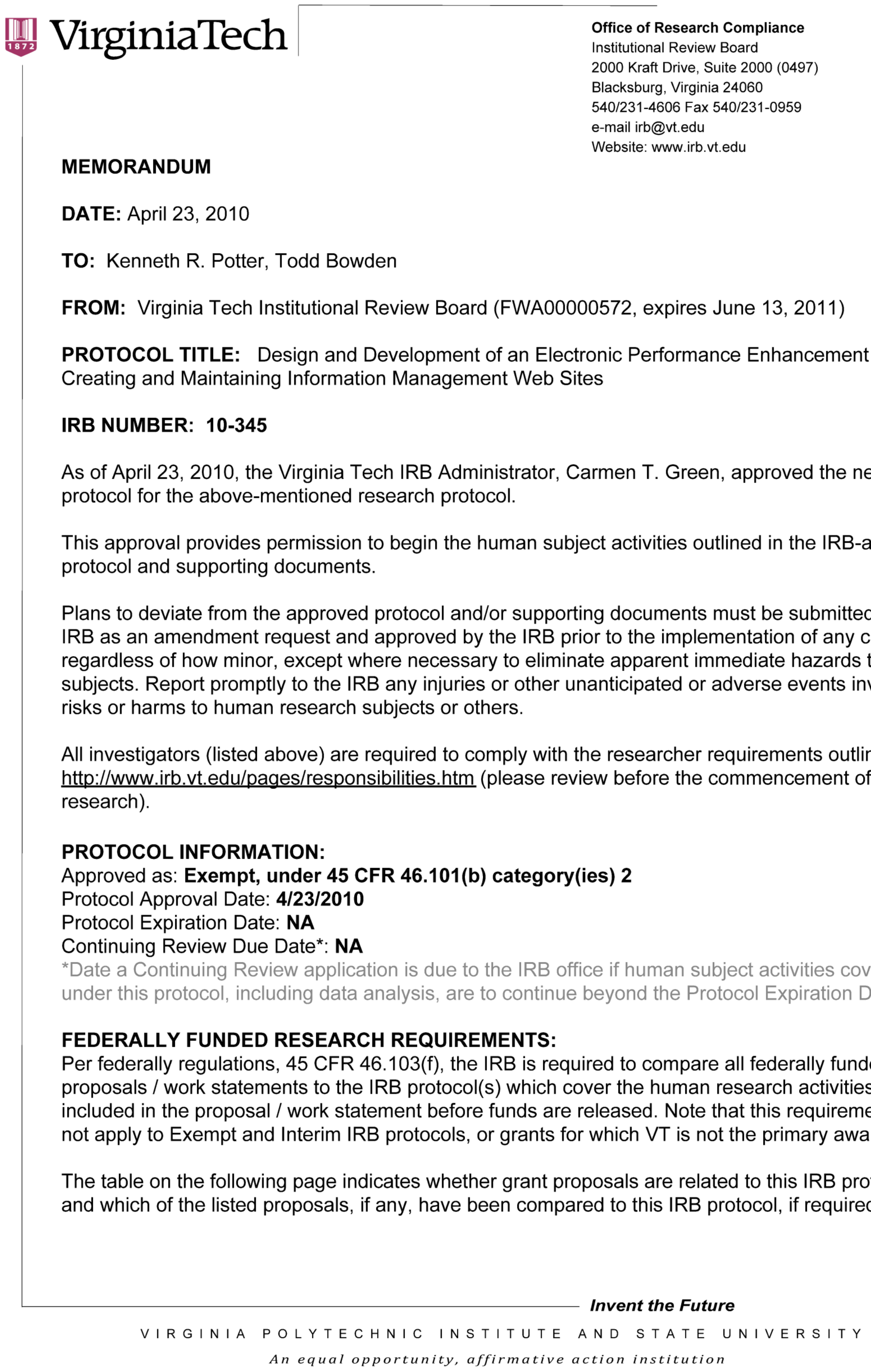
If you needed to create a website similar to the "Flix Crib" or "Student Management", would you consider using FADBIT?	
Evaluator 1	"Yes."
Evaluator 2	"Possibly. But I need to get more familiar with the FADBIT. I don't think one-time experience would help me a lot and convince me to use it."
Evaluator 3	"Yes"
Evaluator 4	"Absolutely!"
Evaluator 5	"I will."
Evaluator 6	"Yes"

Table K2

Evaluators' General Comments on FADBIT

Do you have any other comments you would like to make about FADBIT?	
Evaluator 1	
Evaluator 2	
Evaluator 3	"FADBIT is a very good and very user friendly. I feel that it just needed a little improvement with the interface."
Evaluator 4	"As mentioned previously, FADBIT is an impressive piece of software. I gained an appreciation for all the detail involved in setting up a database and making the interfaces functional. Now that I've been through the tutorials and had time to think about what the tool is really doing, I think I would be able to use it."
Evaluator 5	
Evaluator 6	"The program is easy to manipulate once you get the hang of it."

Appendix L: IRB Approval Letter



VirginiaTech

Office of Research Compliance
Institutional Review Board
2000 Kraft Drive, Suite 2000 (0497)
Blacksburg, Virginia 24060
540/231-4606 Fax 540/231-0959
e-mail irb@vt.edu
Website: www.irb.vt.edu

MEMORANDUM

DATE: April 23, 2010

TO: Kenneth R. Potter, Todd Bowden

FROM: Virginia Tech Institutional Review Board (FWA00000572, expires June 13, 2011)

PROTOCOL TITLE: Design and Development of an Electronic Performance Enhancement Tool for Creating and Maintaining Information Management Web Sites

IRB NUMBER: 10-345

As of April 23, 2010, the Virginia Tech IRB Administrator, Carmen T. Green, approved the new protocol for the above-mentioned research protocol.

This approval provides permission to begin the human subject activities outlined in the IRB-approved protocol and supporting documents.

Plans to deviate from the approved protocol and/or supporting documents must be submitted to the IRB as an amendment request and approved by the IRB prior to the implementation of any changes, regardless of how minor, except where necessary to eliminate apparent immediate hazards to the subjects. Report promptly to the IRB any injuries or other unanticipated or adverse events involving risks or harms to human research subjects or others.

All investigators (listed above) are required to comply with the researcher requirements outlined at http://www.irb.vt.edu/pages/responsibilities.htm (please review before the commencement of your research).

PROTOCOL INFORMATION:
Approved as: **Exempt, under 45 CFR 46.101(b) category(ies) 2**
Protocol Approval Date: **4/23/2010**
Protocol Expiration Date: **NA**
Continuing Review Due Date*: **NA**
*Date a Continuing Review application is due to the IRB office if human subject activities covered under this protocol, including data analysis, are to continue beyond the Protocol Expiration Date.

FEDERALLY FUNDED RESEARCH REQUIREMENTS:
Per federally regulations, 45 CFR 46.103(f), the IRB is required to compare all federally funded grant proposals / work statements to the IRB protocol(s) which cover the human research activities included in the proposal / work statement before funds are released. Note that this requirement does not apply to Exempt and Interim IRB protocols, or grants for which VT is not the primary awardee.

The table on the following page indicates whether grant proposals are related to this IRB protocol, and which of the listed proposals, if any, have been compared to this IRB protocol, if required.

Invent the Future

VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY

An equal opportunity, affirmative action institution

Date*	OSP Number	Sponsor	Grant Comparison Conducted?

*Date this proposal number was compared, assessed as not requiring comparison, or comparison information was revised.

If this IRB protocol is to cover any other grant proposals, please contact the IRB office (irbadmin@vt.edu) immediately.

cc: File