MAXIMIZING STUDENT SOFTWARE ENGINEER PRODUCTIVITY IN HYBRID COMMERCIAL-EDUCATIONAL ENVIRONMENTS

Reduction of Software Development Learning Curves, Introduction of Best Practices and Quality Motivated Procedures

By

Jane Zielieke

A Research Paper
Submitted in Partial Fulfillment of the
Requirements for the
Master of Science Degree
With a Major in

Management Technology
Telecommunication Systems Specialty

Approved: 2 Semester Credits

Research Advisor

The Graduate School

University of Wisconsin-Stout

May, 2004

The Graduate School

University of Wisconsin Stout

Menomonie, WI 54751

ABSTRACT

Zielieke | Jane | M

(Writer) (Last Name) | (First Name) | (Middle Initial)

Maximizing Student Software Engineer Productivity
In Hybrid Commercial-Educational Environments

(Title)

Management Technology | Bruce Johnston | May 04 | 132

(Graduate Program) | (Research Advisor) | (Month/Year) | (# of Pages)

American Psychological Association, 5th edition

(Name of Style Manual Used in this Study)

Software engineering has advanced far procedurally in the last few decades, reaping tremendous benefits in terms of technology, quality, and productivity. In order to remain competitive, companies require reduced training schedules while simultaneously raising quality standards and productivity. Hybrid commercial-educational facilities face the additional challenge to provide a quality education with the intention of providing

students with experiences and knowledge applicable in their post-graduation careers. In an educational-commercial production environment with yearly workforce turnover, rapid training and reduced learning curves are essential. This study describes a new approach that simultaneously addresses software engineer training and process maturity called the CETAP "see-tap" (commercial-educational training and process) method.

Staff training and development has traditionally yielded significant results in technology related fields. The CETAP method is an example of this, specifically focused on software development in a commercial-educational environment. Research for the CETAP method involved the production of several important learning modules including:

- Graphical user interface design,
- Data storage and retrieval.
- Coding style.

These modules in conjunction with implemented process reform were tested for their overall effectiveness. Analysis reveals both significant improvements in quality and reduction in development times.

A substantial contributor to quality and productivity has been the maturity of the software development process. Quite often on a software project, expectations are unstated and unmet, quality suffers, and many man-hours are spent on rework, clarification of process, and responding to customer dissatisfaction. This study shows that implementation of quality control procedures, standardization of the development process, and the use of style guides have resulted in dramatic reductions in training time and development.

ACKNOWLEDGMENTS

I would like to thank the employees at UW-Stout's Technical Services for all their support, encouragement, and consent to participate in my research. I would also like to express my appreciation for Dr. Bruce Johnston for without whom I would most likely have chosen another path.

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER I: INTRODUCTION

Statement of the Problem

This study will address two key issues as they related to software development: training and process. While either is capable of positive returns in development timelines and quality, implementation of both strategies achieve far greater results. This study will discuss the details, rationale, and supporting research of training and process implementation at Technical Services, a small software development within University of Wisconsin-Stout.

In fast paced fields requiring high levels of technical expertise such as software development, the learning curve is a critical factor in determining how quickly an individual can meaningfully contribute to a project. The amount of information one has about doing their job has a significant influence upon the quality of and the rate at which their various tasks are completed. Higher education itself is based upon passing on knowledge to better prepare individuals for future employment situations; however, a college education can only provide a broad based foundation for further learning. Specialized training on the job will be required and will act to increase the likelihood of achieving the desired output and reduce the amount of time before measurable results are seen.

In its infancy, software development was largely ad hoc and characterized as chaotic. However, the last several decades have seen the software development process become much more structured and organized. By improving the processes by which software is created, a software developer's productivity and the quality of the software they produce can be significantly improved. It is by these improvements that software development is starting to achieve some level of maturity. The establishment of organized, repeatable software

development processes has delivered tremendous rewards in terms of production costs to quality of software produced.

Purpose of the Study

This study is intended to identify and address issues related to student software engineers in Technical Services in order to reduce the associated learning curve and increase productivity and quality. Technical Services (TS) is a comprehensive Information Technology department within Student Life Services, providing networking, system administration, and software development services to departments on campus.

Historically, student software engineers begin their employment at TS during the first semester of their junior year and require, on average, six months to one year to meaningfully contribute to the department, leaving very little time for significant project work. As the department has a heavy reliance on the use of students for the core software engineering and support, management has indicated that the time currently required to train them needs to be reduced. Management also indicated that software quality needs to be improved.

A program was designed to address systematic and programmatic issues identified during a pre-CETAP interview with the relevant supervisors. A new student software engineer participated in the new training modules including: graphical user interface design, data storage and retrieval, and code style. Upon successful completion of the training modules, an actual software development project was completed by the student programmer. This gave management an opportunity to assess the effectiveness of the training modules. Effectiveness was measured by comparing interviews with the supervisors both before and after the CETAP program.

The goal of this study is to significantly reduce the student software engineer's learning curve and enhance the quality the software developed by implementing a series of self-study learning modules for on the job training of the student software engineers.

Assumptions of the Study

This study assumes student software engineers have a basic understanding of software development processes and programming languages through coursework or other work experiences prior to inclusion in this study. In addition, it assumes that all other factors such as supervisor involvement remain constant.

Definition of Terms

For the purposes of this study, the following definitions and terms will be used:

Student software engineer – A student enrolled at University Wisconsin – Stout, employed by Technical Services for the purpose of software development.

Programmer – see student software engineer.

GUI – Graphical User Interface, a visual interface by which a computer user may interact with a software application.

Maintenance – Repairing software application defects and adding capability (Braude, 2004).

Testing - "The process of exercising or evaluating a system by manual or automatic means to verify that it satisfies specified requirements or to identify differences between expected and actual results" (Stiller & LeBlanc, 2002).

Code bloat – the growth, in terms of file size, of a code segment or application without obvious benefit (Langa, 2001).

Limitations of the Study

Limitations of this study are as follows:

1. It is not within the scope of this research to analyze the impact of changes in available technology or development utilities.

2. A comparison between learning curve reductions obtained through systematic improvements and those obtained through learning modules has not been made.

3. The limited sample size and situation specific variables may limit how well the results obtained within this study apply to other populations.

Methodology

Technical Services at University Wisconsin-Stout, committed to its role as both an educational and software development organization, appreciated the need to develop or improve their software development processes and their student software development employee training program. Key process challenges included:

- Increased managerial awareness of software project status,
- Improved quality assurance,

- Implementation of formalized workflow, and
- Utilization of standardized documentation practices.

Implementation of procedures and practices discussed in further detail chapters two and three revealed reduced development timelines, increased quality, and improved inter-departmental communication.

Some key skills necessary for effective software development at Technical Services include:

- Adherence to coding style standards as determined by campus directives and industry standards,
- Utilization of effective data storage and retrieval methods, practices, and syntax, and
- Observance of graphical user interface (GUI) design standards as determined by industry standards.

In order to reduce the time necessary for a student software developer to meaningfully contribute to a project, increase the quality of their work, and reduce the time needed for project rework, this study employed the following steps:

1. Implement the training modules for the CETAP method.
2. Give a pre-CETAP questionnaire to the software development managers in order to ascertain a historical baseline.
3. Train the new student software engineers using the CETAP program.
4. Develop a software application utilizing the CETAP method and principles.
5. Give a post-CETAP questionnaire to the software development managers for comparative analysis against the baseline.

The remainder of this study will discuss the proposed solutions and supporting research, implementation, and results contained in the CETAP method, followed by a discussion.

CHAPTER II: LITERATURE REVIEW AND BACKGROUND

Process Development

In its infancy, software development was largely task, value-added work with little in the way of process maturity and administration. As a result, software developers have developed a reputation for producing software of questionable quality, an inability to meet deadlines, and frequent cost overruns. Software developers have gradually taken this lesson to heart, learning that process directly results in product quality, decreased effort, and reduced cycle-times (Harter et al., 2000).

New software development processes are often perceived as cost prohibitive, demanding increased effort and personnel hours, or requires excessive timelines. Harter et al. (2000) empirically proved this presumption incorrect in their study finding:

> The net effect of improvement in process maturity on development cycle time and effort is negative. At the average values for process maturity and software quality, a 1% improvement in process maturity leads to a 0.32% net reduction in cycle time, and a 0.17% net reduction in development effort (taking into account the positive direct effects and the negative indirect effects through quality). These findings provide empirical support in the context of software production for theories of the cycle time and cost benefits of improved quality deriving from process improvement.

Their research dispels the presumption that process improvements have a significant cost of implementation, increased development, and monumental effort.

Quite frequently software development activities are undertaken to mitigate risk for the developers. Risks can come in the form of profits lost due to missed contractual timelines, lost business and prestige, claims of unfulfilled requirements that have gone undocumented, and

excessive personnel hours devoted to appeasing clients' seemingly endless requests for upgrades (termed "scope creep"). Beck (2000) supports the necessity of procedures and documentation as part of software development stating, "The strategy for the team is to invest as little as possible, to put the most valuable functionality into production as quickly as possible, but only in conjunction with the programming and design strategies designed to reduce risk" (p. 87).

Process maturity has proven to be a significant determinant of both quality, speed to release dates, and the productivity of programmers. The Capability Maturity Model (CMM) utilized in the Li et al. study (2003) shown in Table 1 below defines and describes levels defined in the CMM. Harter et al. (2000) states, "the CMM framework includes 18 key process areas such as quality assurance, configuration management, defect prevention, peer review, and training."

Table 1

The Capability Maturity Model

CMM Level	Characteristics	Key Challenges	Key Process Areas
1. Initial	(Ad hoc) Project Chaotic	+ Project management + Project planning + Configuration management + Software quality assurance	+ Ad hoc processes
2. Repeatable	(Intuitive) Process dependant on individuals	+ Training + Technical Practices + Process Focus	+ Requirements management + Software subcontract management + Software project tracking & oversight + Software project planning + Software quality assurance + Software configuration mgmt.
3. Defined	(Qualitative) Process defined and coordination	+ Process measurement + Process analysis	+ Peer reviews + Intergroup institutionalized + Quantitative quality plans + Software product engineering + Integrated software management + Training program + Organization process definition + Organization process focus
4. Managed	(Quantitative) Measured process	+ Changing technology + Problem analysis + Problem prevention	+ Software quality management + Quantitative process management
5. Optimizing	Improvement fed back into process	+ Still human intensive + Maintain organization at optimizing level	+ Process change management + Technology change mgmt + Defect prevention

Prior this research, Technical Services struggled with project management and planning, had little in the way of quality strategies, or workforce training. Using the CMM model, Technical Services was categorized as being in the initial, ad hoc phase of software process. Through implementation of these key processes, significant quality and cycle time improvements can be achieved, moving them from the ad-hoc phase to repeatable and defined.

Technical Services emphasized their need for process improvement through standardization and documentation coupled with an introductory training program for their new programmers. Establishment of workflow, documentation templates such as application request, requirement, use case, and user manuals address project management, planning, and quality assurance difficulties. In addition, implementation of testing procedures and peer review were identified as areas of desired and necessary improvement as a part of process reform and progression within the CMM model.

Establishment of a Formalized Workflow

The CMM model characterizes the initial phase of software development maturity as ad hoc or chaotic, struggling with basic project management tasks. Historically, software development in the department under study remained largely the responsibility of the student programmers leaving projects unplanned and undocumented resulting in protracted development timelines, poor quality, and code bloat. Providing a customized framework for good software design represents the first step in unlocking potential within the department. Ultimately, formalizing workflow procedures, detailing each step in the process from acceptance of the project to maintenance cycle, would directly result in enhanced quality and productivity by outlining expectations and methods.

Establishment of Standard Documentation Items

It is a common experience amongst users and developers to be frustrated by poor or nonexistent documentation. Documentation plays an important role in communication, quality, and continuous improvement. Providing a standard written document for each phase of the project incorporates flexibility and consistency, allowing any number of developers to easily assist, diminishing the reliance on specific experience-based knowledge of a single programmer. Ongoing document development is also important for the customers and users. As an example, users provided with manuals require less developer assistance. Moreover, requiring documentation throughout the development process provides management with a milestone of current progress, a measurable outcome that can be improved upon with each iteration. Based on management's areas of desired improvement and industry recommended standards, application request, requirements, use case, user manual, testing procedure, and peer review documents and procedures were included in CETAP method.

Application request document

Software development supervisors indicated interested in formalizing the application request process in order to increase efficiency, clarity, and customer satisfaction. Various methods for development inquiry had been in use at Technical Services, including verbal requests to administrative staff and direct requests to the programmer. Formalizing the application request process would improve development timelines, quality, and customer service. As a part of the request process, a feasibility assessment would evaluate at the outset whether the client's goals would best be met by custom or pre-packaged software and whether or not each department is technically, financially, and time-wise capable of meeting their needs.

Requirements document

A requirements document is perhaps the most significant record of a project's purpose, containing specific needs of the client. Clarification of requirements, timelines, estimated costs, responsibilities, and risks better prepare the client and the development department in order to deliver a better product. While developing a requirements document requires both time and effort, Stiller and LeBlanc (2003) site Boehm's work which asserts, "... investing more upfront effort in verifying and validating the software requirements and specifications, software project developers will see reduced integration and test costs as well as higher software reliability and maintainability." Utilization of a requirements document fulfills management's objectives, increased quality and customer satisfaction in a reasonable timeframe at a reasonable cost.

Use case documents

The utilization of use case documents has become a mainstay in the software development process because "the whole development process since most activities such as analysis, design, and test are performed starting from use cases," ultimately providing "... a systematic and intuitive means of capturing functional requirements with a focus on value added to the user," (Jacobson et. al, 1999). Technical Service's desire to formalize their processes, educate their workforce, increase product quality, and decrease development timelines can be supported through use cases.

User manuals

The department under study identifies user manuals as an opportunity for decreasing personnel costs and improving customer service management estimates over two hours of initial training per user per application plus recurring training to refresh users on infrequently used functionality. Documentation, when present, is often incomplete resulting in additional

programming team time spent on follow questions and site visits. A consistent, well-designed, user-oriented manual can represent hours of saved development time and increased user satisfaction.

Provision of a user manual template provided a standard layout, demonstrating the intent and format to the students, saved them time they had previously spent developing their own from scratch and prepared them for the differences between academic and workplace writing. As an educational institution with primarily student employees with limited technical writing experience, it is useful to emphasize the fundamental differences between styles. Barker (1991) describes the difference as the following:

> *When writing papers for class projects in an academic setting, the primary audience of a student paper is the professor who assigned it, the purpose of the paper is to show how much the student knows about the subject. However, in the workplace, readers are not teachers/critics but co-workers-people who need information rather than people who are verifying writers' grasp of the information. As a result, writers must change strategies for writing and strategies for finding out whether a document is valuable.*

With a template addressing the academic vs. workplace distinction, programmers can improve their productivity by reducing time spent on improperly directed user help and user manual design.

Previous research has been done on the content of (user) help manuals, regarding the best methods and approaches for design. Barker (1991) states that "people read for a purpose, and that their purpose affects the way that they read, absorb, and apply the information," suggesting the importance of user involvement and designing user-oriented help materials. Additionally, the

Shayo et al. (1999) study sites Panko's research, supporting user involvement stating "software trainers' efforts may be more effective if they understand their trainee's requirements before the design of the training materials, selection of training methods and actual training" emphasizing not only designing for the user, but involving the user themselves in the design of their training and help materials in order to "produce motivated, knowledgeable users who will subsequently use their software skills on the job."

Ultimately, the purpose of user-oriented assistance documentation is to increase customer satisfaction via improved instruction; however, an additional benefit manifests in the form of reduced developer involvement post-release. Research has shown that the content and style of the document can have significant impact on the understanding and problem-solving skills of the application users. Dutke and Reimer (2000) studied the effectiveness of two distinct information formats, operative and function-oriented help, to determine their effect on learning performance. Operative help indicates a list of steps necessary to achieve a desired goal. Function-oriented help informs the user of how a particular function works. Their research found that operative based help provides more assistance during the initial schema acquisition phase while function-oriented help is more useful during the model building phase. In essence, operative helps the user early on, in acquiring the basic layout of the software while function-oriented provides adaptive skills later, encouraging users to creatively and efficiently find solutions without developer assistance. Utilization of a combination of both information formats may reveal cumulative benefits.

Testing Procedures

Testing, while obviously necessary, is often inadequately addressed because of limited time schedules, costs, and has diminishing returns upon each iteration (Conger, 1994, Barnum,

2002). Especially prevalent in small in-house development environments, testing is often delayed until the last phase of development with the tacit assumption that the users will fill the role of testers, essentially deemphasizing quality in the end product (Albin, 2003). Testing should include more than the proverbial "monkey test", a form of testing wherein a system is confirmed to act sensibly, regardless of nonsensical input (Beck, 2000). Testing should include the following elements: first impressions, first tasks, frequently and most important tasks (both based on use cases), specific problem areas, and anything new introduced to the system since the last iteration of testing (Barnum, 2002).

Technical Services management identified quality, customer service, and time invested in post-release devoted to fixes as areas for desired improvement. Putting into policy testing early, often, and thoroughly, common best practices, dramatically affect the quality of the end product resulting in increased customer satisfaction and decreased post-release rework (Braude, 2004, Hunt & Thomas, 2000). Iterative testing, throughout the development process in conjunction with a policy of quick reporting of issues and immediate correction would show a significant positive effect, directly addressing management's goals.

Technical Reviews

Technical reviews, such as walkthroughs, inspections, and formal or informal code reading, are as significant to controlling costs as testing (McConnell, 1998). The formal review process:

- notification and distribution,
- preparation,
- review meeting,
- review report, and

- follow up (McConnell, 1998),

can often be scaled down for small-shop environments in the form of peer review. Braude (2004) notes that testing can reveal defects, but never their absence. Human beings, during peer review can determine additional information such as the quality (understandability and maintainability) and efficiency of code. Such information is crucial when estimating maintainability, upkeep costs, and system requirements, as such; technical reviews are an invaluable tool for cost reduction and quality control.

In addition to tangible business goals, technical reviews can provide Technical Services with an opportunity for often overlooked benefits such as workforce development and evaluation. McConnell (1998) makes mention of a "cross-pollination" benefit obtained from code reviews. During review, less experienced programmers glean valuable techniques, procedures, and code itself from senior developers, reducing the time expended learning independently. The learning process goes upstream as well, however. Senior developers can benefit from informal evaluations of their trainees as well as the new methodologies and the challenge of old habits and assumptions brought out by the review process.

Establishment of Training Modules

The planned progression through CMM levels has been specifically adapted to Technical Services unique role in an educational institution, moving items usually delayed until later CMM levels earlier in the process. Peer review and training programs which are usually delayed until an organization as reached the third "defined" level of development were developed earlier in order to address historically documented deficiencies. Specifically, software development supervisors indicated that training modules were needed in the areas of: coding style, data

storage and retrieval, and graphical user interface design. These modules became the focal point of the training program developed during the course of this research.

Coding Style

Coding style correlates directly with the maintenance costs associated with the last phase of the software development lifecycle. Budgen (1994) notes that "many of the factors that effect maintainability are related to implementation factors or, at least, to very detailed design issues. Examples of these are the choice of identifiers, comment structuring practices, and documentation standards." Albin (2003) and McConnell (1998) found that the maintenance phase of the software development lifecycle accounts for the majority of application costs. Retrofitting quality coding has proven costly and impractical; therefore, enforced code practices early on, particularly in preliminary training will increase the quality and productivity of the department's programmers (McConnell, 1998).

Data Storage & Retrieval

Database applications comprise the majority of Technical Services' workload. As such, adequate training or database techniques for new employees are crucial to the productivity of the department. With modules developed in this study, software engineers are more likely to pick the correct tools for various database operations, resulting in more flexible applications and less accidental data loss. In addition, they will achieve higher database application development quality levels (fewer redundancies, appropriate data types, etc.). Based on software supervisors' goals, this research focused on creating a learning module designed to give the programmers basic programmatic experience with syntax and methodologies associated with database connectivity and manipulation. As database design is a part of the University's Computer Science curriculum and is a strongly encouraged elective by Technical Services, this research

excludes database design from its scope, focusing instead on the specifics pertaining to language choice and departmental requirements.

GUI Design

Graphical interfaces present a powerful first and lasting impression with clients. Consistent, clear, and user-oriented interfaces that incorporate intuitive controls based on recognition instead of recall have the following benefits:

- Increased user productivity,
- Reduced user training time,
- Increased programmer productivity via reduced decision time,
- Reduced programming time (Weinschenk, 1997), and
- Reduced reliance on external documentation sources such as user manuals.

Poor interface design ultimately results in increased personnel costs and development times. Specific consequences of poor GUI design, many of which suffered by Technical Services, are:

- Confused, panicked, bored, and frustrated users;
- Abandonment of the system;
- Misuse or direct programming (directly accessing data, circumventing the UI) of the system (Galitz, 1997);
- Increased user training time;
- Increased maintenance time; and
- Increased time devoted to corrections during testing.

CHAPTER III: METHODOLOGY

The primary purpose of this research was to identify areas of possible quality and productivity improvements in a small software development department in keeping with its mission as a hybrid commercial and educational facility and implement recommended policies and training based on those needs.

Subject Selection and Description

Subjects were selected based on their employment in the department under study and relevance to the research topic. To qualify under the scope of this research, student software engineers must meet the following criteria:

1) Be an enrolled student at the University of Wisconsin-Stout; and,
2) be a current employee of the department under study with a job title related to software development. Examples include: software engineer, software developer, web developer, programmer, and advanced programmer.

No distinction was made based on past development experience, major, percentage of degree completed, gender, or age as it pertains to subject selection. Two subjects were selected based on the above criteria.

Supervisors were selected based on the quantity of direct supervisory contact with the selected student population and job description. Supervisors with direct, formal authority or those that have significant contact or influence on the student software engineers were included based on relevance. Dominic Howard (Information Systems - Comprehensives Services Professional – Senior) and Graydon Richartz (IS Systems Development Services – Senior) were selected as a result of formal and historical authority over the area of study within the department.

Instrumentation

Supervisors were presented with a researcher-developed questionnaire containing fifteen questions. The contents of the questionnaire, available in Appendix A, covered basic concepts such as software development processes, training, and development. Specific questions were asked based on information gathered during the exploratory phase of research in the form of preliminary discussions with software development supervisors. The questionnaire was designed with open-ended questions in order to elicit a broad range of qualitative and/or quantitative responses.

The CETAP model developed in this study is an adaptation of the CMM model discussed earlier. Table 2 (below) merges the needs of Technical Services and the principles of the CMM model for a streamlined approach, discarding several inapplicable items and adding specialized requirements. As basic principles are required for later phases of software development, the "managed" and "optimized" phases are areas for further study; however, several preliminary areas are suggested.

Table 2

The CETAP Model

CETAP Level	Characteristics	Key Challenges	Key Process Areas
1. Initial	(Ad hoc) Project Chaotic	+ Project management + Project planning + Configuration management + Software quality assurance	+ Ad hoc processes
2. Repeatable	(Intuitive) Process dependant on individuals	+ Training + Technical Practices + Process Focus	+ Requirements management + Software project tracking & oversight + Software project planning + Software quality assurance
3. Defined	(Qualitative) Process defined and institutionalized	+ Process measurement + Process analysis + Quantitative quality plans	+ Peer reviews + Training program + Organization process definition + Organization process focus
4. Managed	(Quantitative) Measured process	+ Changing technology + Problem analysis + Problem prevention	+ Software quality management
5. Optimizing	Improvement fed back into process	+ Still human intensive + Maintain organization at optimizing level	+ Process change management + Technology change mgmt + Defect prevention

Data Collection Procedures

The questionnaire was administered pre- and post-CETAP implementation in order to establish a basis for comparative analysis. Pre-CETAP responses were used to establish a baseline response. Post-CETAP responses were compared to the baseline in order to ascertain the effectiveness of the training modules and improved software development process.

Data Analysis

Data analysis was conducted via a side-by-side comparison of the composite answers given by software development supervisors pre- and post-implementation of the CETAP modules and procedural reforms. Forming the basis for the assessment of the effectiveness of this study, qualitative assessment methods were used. Reponses were assessed based on stated perceived change by the respondents.

Limitations

The qualitative nature of the questionnaire itself elicits a broad spectrum of responses including past experiences, anecdotes, quotes, factual, and quantitative information making concrete quantitative analysis difficult. Subjective interpretations of the comparison between pre- and post-CETAP questionnaire responses were largely left to the researcher. Quantitative responses of an individual pre- and post-CETAP retain none of the subjectivity of the qualitative responses. In addition, the small sample size of this study limits broad applicability to a larger population. Conducting further studies with a larger sample size may provide a more broad perspective, revealing any statistically significant trends.

CHAPTER IV: RESULTS

This study identifies and addresses issues related to Technical Services student software engineer learning curves, productivity, and quality. A questionnaire was developed to establish a basis for comparison of management's assessment of departmental processes and personnel prior and subsequent to utilization of the CETAP method developed during this study.

Item Analysis

Table 3 below contains a summation of pre- and post-CETAP questionnaire responses contained in Appendices B and C, respectively, followed by an assessment of the effectiveness of each item (see Appendix A).

Table 3

Questionnaire Item 1 *How has the utilization or lack of project request documents affected the software development process specifically regarding quality of work produced and on what time frame?*	
Pre-CETAP	• time lost due to communication delays • quality suffers • frequent rework unauthorized projects
Post-CETAP	• shortened development process • fewer design meetings • decreased user and programmer frustration • increased quality

As a part of the CETAP method, project request documents have reaped significant rewards for Technical Services in terms of time spent in the development process, communication between all parties involved, and end-product quality. Formalizing the development request process has prevented developers from taking on projects on their own, increased management awareness and participation early on in the process, and educated the user on the resources necessary to accomplish their goals; ultimately increasing their satisfaction with the application and the department. Productivity increased since less time was wasted pursuing

unclear goals, attending multiple meetings between the various parties, and increased developer satisfaction. Quality improved because user goals were made clear earlier on in the development process, during conception versus the user approval phase.

Table 4

Questionnaire Item 2 *How has the utilization or lack of project design documents affected the software development process specifically regarding quality of work produced and on what time frame?*	
Pre-CETAP	• significant delays • workflow issues
Post-CETAP	• shortened development time by 60 to 70% • improved quality • Accurate "vision"

Project design documents perform a critical role in the software development process. CETAP's inclusion of design documents addressed several key managerial concerns, prolonged development schedules and workflow. Shortening the development lifecycle by an estimated sixty to seventy percent, design documents clarify user requirements and provide a project plan or "vision" for developers. Such a vision increases productivity by cutting out significant guesswork and later, less rework when the software does not match unstated user requirements. A detailed design document, such as the one used during this study, directly results in increased quality. User requirements are met, raising customer service and decreased likelihood that they would discontinue use of the application due to dissatisfaction.

Table 5

Questionnaire Item 3 *How has the utilization or lack of project use case documents affected the software development process specifically regarding quality of work produced and on what time frame?*	
Pre-CETAP	• significant delays • frequent rework • unmet user requirements • 50% of code thrown away
Post-CETAP	• Shortened development phase • Improved quality • Reduced "scope creep"

Use case documents further improve quality and shortened the development process at Technical Services. Reduction of delays, rework, and thrown away code significantly contributed to the development team's productivity.

Table 6

Questionnaire Item 4 *How has the utilization or lack of user manuals affected the software development process specifically regarding developer payroll hours devoted to post-release maintenance and user training?*	
Pre-CETAP	• Delays • Substantial time devoted to follow-ups • 2 hours devoted to training per user
Post-CETAP	• Reduced training time • Programmer familiarity • As a result of this policy, no users have requested one-on-one training.

Standardized use case templates had two distinct benefits, reduced user training and reduced developer time devoted to creation of new user manual documents for each application. Through the CETAP method, user training went from over two hours per user per application to

virtually none when the user utilized the user manuals. Decreased training time may also be a result of increased GUI design quality.

Table 7

Questionnaire Item 5 *How has the utilization or lack of project error report documents affected the software development process specifically regarding quality of work produced and on what time frame?*	
Pre-CETAP	• Significant time spent sourcing out problems • No basis for improvement • Reduced quality • Negative affects on testing procedures
Post-CETAP	• Quicker debugging • Reduced required user assistance • Increased communication • Incremental vs. batch error reporting

The CETAP method, as it relates to error reporting, was designed to address managerial concerns regarding the significant amount of time their developers spent determining the source of application errors. Requiring a standardized error report generated each time an application encounters an error provided the developer with an invaluable tool for finding the source of errors efficiently. Ultimately error report documents accomplished the following managerial objectives: quicker error detection and correction, increased communication between developers, users, and the application, and increased customer satisfaction.

Table 8

Questionnaire Item 6 *How has the utilization or lack of project approval procedures affected the software development process specifically regarding quality of work produced?*	
Pre-CETAP	• Significant time spent chasing unapproved projects • Quality suffered
Post-CETAP	• Discouraged irrelevant request submissions • Decreased development time spent evaluating whether or not to accept the development project • Clear user goal definition

Utilization of project approval procedures proved beneficial to Technical Services. Management listed fewer frivolous request submissions, decreased development time, and clear user goal definition as some of the benefits of the CETAP method.

Table 9

Questionnaire Item 7 *How has the presence or lack of established project responsibility hierarchies affected the software development process specifically regarding quality of work produced time frame?*	
Pre-CETAP	• Extended timeframes • Code bloat • Scope creep
Post-CETAP	• Increased managerial direction of process • Acceptance standards became possible • Reduced command squabbles • Defined development roles for users and developers • Increased error/issue detection

Establishment of project responsibility hierarchies represented a significant step for Technical Services, moving away from late project delivery dates, code bloat, and scope creep. The CETAP method addressed these issues and produced significant benefits, such as those in Table 9 above.

Table 10

Questionnaire Item 8 *How has the utilization or lack of a peer review process affected the software development process specifically regarding quality of work and on what time frame?*	
Pre-CETAP	• Introduction of errors • Time lost
Post-CETAP	• Detection of "bad" code • Coding practices enforced • Consistency & standardization • Increased quality • Increased collaboration • Increased communication • Overall time savings

Peer review represents perhaps the most significant contribution of the CETAP method toward Technical Services' goals of productivity, quality, and education. Peer review increased the consistency and quality of the software produced while simultaneously increasing the knowledge exchange rate between developers. Table 10 (above) gives a synopsis of the benefits of CETAP observed by management as pertains to peer review.

Table 11

Questionnaire Item 9 *How has the utilization or lack of a quality assurance "QA" process affected the software development process specifically regarding quality of work produced and on what time frame?*	
Pre-CETAP	• Release of non-functional code • Customer service suffered • Poor quality • Time lost
Post-CETAP	• Improved error detection • Estimated time saved: "a few days" • Improved customer perception of quality

Quality assurance, while an upfront expenditure of time, proved to save Technical Services overall development time. In addition, software quality improved prerelease, improving customer approval and reducing post-release rework. As a result, quality assurance processes have proved to be an effective part of the CETAP method.

Table 12

Questionnaire Item 10 *How has the utilization or lack of a user approval "UA" process affected the software development process specifically regarding quality of work produced and on what time frame?*	
Pre-CETAP	• Increased user training time • Additional programming time • Customer satisfaction suffers
Post-CETAP	• Improved assignment of responsibility and accountability • Fewer "last minute changes" • Improved user workflow • More positive release of software (fewer "glitches", positive user involvement, fewer follow up appointments, etc.)

The CETAP method includes user approval process in order to address user training and project development timelines, as well as workflow, quality, customer service. User participation earlier in the development lifecycle increases product-to-requirements matching and positively influences their perception of the product as a whole.

Table 13

Questionnaire Item 11 *How has the presence or lack of an isolated development environment affected the software development process specifically regarding quality of work and time frame?*	
Pre-CETAP	• Errors remain undiscovered until roll-out • Customer satisfaction suffers • Additional time spent reworking and debugging
Post-CETAP	• Improved user perception • Decreased development time • Likely future benefits

Utilization of an isolated development environment address brought about improved quality by tightly controlling the circumstances under which an application is created. Inclusion of this element in the CETAP method represents current and future improvements to Technical Services management.

Table 14

Questionnaire Item 12 *How has the presence of lack of an isolated test lab affected the software development process specifically regarding quality of work produced and on what time frame?*	
Pre-CETAP	• Difficulty tracing errors • Quality suffered
Post-CETAP	• Reduced time spent testing • Increased programmer confidence • Hours saved not having to reload the machine to its previous state

An isolated and consistent test lab assisted the developer in identifying specific causes of application code and integration errors, cutting down on the guesswork. Elimination of the operating system and associated software packages as a potential source for errors found during the testing process saw increased programmer confidence during the project release phase.

Additionally, Technical Services, through the use of system state restoration software maintained consistency and reduced system reset times by several hours thereby increasing their productivity.

Table 15

Questionnaire Item 13 *How has the presence or lack of code documentation training affected the software development process specifically regarding quality of work produced and on what time frame?*	
Pre-CETAP	• Delayed developer acclimation to an existing application • Quality loss on subsequent development
Post-CETAP	• Future benefits for consistency across personnel • Reduced documentation generation time • Improved structure, readability, and standardization

Technical Services' employee pool, primarily consisting of students, has a naturally high turnover. Consistency across "generations" of developers was of paramount concern to management. In-code documentation clearly and quickly communicated to the reader the purpose of a code segment. The benefits of standardization of code will become increasingly apparent with future enhancements of this particular software project.

Table 16

Questionnaire Item 14 *How has the presence or lack of data storage and retrieval training affected the software development process specifically regarding quality of work produced and on what time frame?*	
Pre-CETAP	• Protracted timelines • Difficulty backing up system • Data integrity concerns • Developer preference for storage medium rather than best choice
Post-CETAP	• When implemented and used consistently, savings of hours to days can be realized • Exercises developed proficiency and confidence of the programmers

Database applications comprise the vast majority of Technical Services' workload indicating that data storage and retrieval training (such as in the CETAP method) would reduce the learning curve. This study found implementation of database training produced a more proficient and confident developer on a shorter timescale when compared to those without.

Table 17

Questionnaire Item 15 *How has the presence or lack of graphical user interface "GUI" training affected the software development process specifically regarding quality of work produced and on what time frame?*	
Pre-CETAP	• Customer satisfaction suffered • Inconsistent interfaces
Post-CETAP	• Easier user adaptation to changes • Increased user understanding • Reduced the number of false error reports • Aided in transitioning of applications between programmers • Resulting GUI's were clean, straightforward, easy to use, and consistent

This study found implementation of graphical user interface design training produced a more clean, usable, and consistent interface after training than prior to implementation of the

CETAP module. Reduced user training/assistance were a direct result were a chief benefit achieved by the training.

CHAPTER V: DISCUSSION

In order to meet management's quality and productivity goals while keeping in mind the department's role as part of an educational institution, an adaptation of the Capability Maturity Model, called the CETAP method, was developed and implemented. To review, this study followed the steps below in order to ascertain CETAP's effectiveness.

1. Implement the training modules for the CETAP method.
2. Give a pre-CETAP questionnaire to the software development managers in order to ascertain a historical baseline.
3. Train the new student software engineers using the CETAP program.
4. Develop a software application utilizing the CETAP method and principles.
5. Give a post-CETAP questionnaire to the software development managers for comparative analysis against the baseline.

Conclusions

This study identified two key areas Technical Services management wanted to address within their software development department: productivity and quality. Both were addressed by process and training reforms. Cumulatively, formalization of workflow, documentation templates, testing procedures, and technical reviews shortened the development lifecycle significantly while simultaneously increasing the quality of the end-product. Process reform can reasonably be expected to have a multiplicative time savings for larger development projects. As a part of the CETAP method, coding, database, and GUI learning modules produce confident, skilled programmers who can actively contribute to development projects earlier than those who have not undergone training.

Recommendations

Based upon the results of this study, further research may be conducted to:

- Study the effects of achieving a managed or optimized state at Technical Services,
- Compare the benefits achieved through process reform with those obtained through training, and
- Move Technical Services from an ad-hoc CETAP level to that of defined or optimized.
- Study the development of additional CETAP modules for further advances up the CMM scale.

References

Albin, S. (2003). *The Art of Software Architecture: Design methods and techniques.* Indianapolis, IN: Wiley Publishing, Inc.

Bannert, M. & Reimann, P. (2000). Guest Editorial: Approaches to the Design of Software Training. *Journal of Computer Assisted Learning, 16*, 281-283. Retrieved 2/10/04 from EBSCOhost.

Barker, Thomas T. (Ed.). (1991). *Perspectives on Software Documentation: Inquiries and innovations.* Amityville, NY: Baywood Publishing Company, Inc.

Barnum, C. (2002). *Usability Testing and Research.* New York: Longman.

Beck. K. (2000). *Extreme Programming Explained: Embrace change.* Boston, MA: Addison-Wesley.

Braude, E. (2004). *Software Design: From Programming to Architecture.* Boston University, MA: John Wiley & Sons, Inc.

Budgen, D. (1994). *Software Design.* Wokingham, England: Addison-Wesley Publishing Company.

Conger, S. (1994). *The New Software Engineering.* Balmont, CA: Wadsworth Publishing Company.

Coomer, J. (2004). *Organizational Leadership.* UW-Stout, Menomonie, WI.

Coomer, J. (2004, February). *The Deming Chain Reaction: A strategy for Management.* Presented in classroom at UW-Stout, Menomonie, WI.

Dutke, S. & Reimer, T. (2000). Evaluation of Two Types of Online Help for Application Software. *Journal of Computer Assisted Learning, 16*, 307-315. Retrieved 2/10/04 from EBSCOhost.

Galitz. W. (1997). *The Essential Guild to User Interface Design.* New York: Wiley Computer Publishing.

Harter, D., Mayuram, S., & Slaughter, S. (2000). Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development. *Management Science*, 46, 451-466. Retrieved 2/10/04 from EBSCOhost.

Hunt, A. & Thomas, D. (2000). *The Pragmatic Programmer.* Reading, MA: Addison Wesley Longman, Inc.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The Unified Software Development Process.* Reading, MA: Addison Wesley Longman, Inc.

Johnson, J. (2000). *GUI Bloopers: Don'ts and do's for software developers and web designers.* San Francisco, CA: Morgan Kaufmann Publishers.

Langa, F. (2001). Rethinking 'Software Bloat.' *InformationWeek.* Retrieved May 5, 2004 from http://www.informationweek.com/story/IWK20011212S0003

Li, E., Chen, H., & Lee, T. (2003). A Longitudinal Study of Software Process Management in Taiwan's Top Companies. *Total Quality Management,14*, 571-590. Retrieved 2/10/04 from EBSCOhost.

McConnell, S. (1998). Software *Project Survival Guide.* Redmond, WA: Microsoft Press.

Shayo, C., Olfman, L., & Teitelroit, R. (1999). An Exploratory Study of the Value of Pretraining End-User Participation. *Info Systems Journal, 9*, 55-79. Retrieved 2/10/04 from EBSCOhost.

Stiller, E. & LeBlanc, C. (2002). *Project-based Software Engineering: An object oriented approach*. Boston, MA: Addison Wesley.

Weinschenk, S., Jamar, P., & Yeo, S. (1997). *GUI Design Essentials*. New York, NY: John Wiley & Sons, Inc.

Appendix A: Questionnaire Instrument

Assuming a small to medium sized software development project…

1) How has the utilization or lack of project *request* documents affected the software development process specifically regarding quality of work produced and on what time frame?
2) How has the utilization or lack of project *design* documents affected the software development process specifically regarding quality of work produced and on what time frame?
3) How has the utilization or lack of project *use case* documents affected the software development process specifically regarding quality of work produced and on what time frame?
4) How has the utilization or lack of *user manuals* affected the software development process specifically regarding developer payroll hours devoted to post-release maintenance and user training?
5) How has the utilization or lack of project *error report* documents affected the software development process specifically regarding quality of work produced and on what time frame?
6) How has the utilization or lack of project *approval procedures* affected the software development process specifically regarding quality of work produced?
7) How has the presence or lack of *established project responsibility hierarchies* affected the software development process specifically regarding quality of work produced time frame?
8) How has the utilization or lack of *a peer review process* affected the software development process specifically regarding quality of work and on what time frame?
9) How has the utilization or lack of *a quality assurance "QA" process* affected the software development process specifically regarding quality of work produced and on what time frame?
10) How has the utilization or lack of *a user approval "UA" process* affected the software development process specifically regarding quality of work produced and on what time frame?
11) How has the presence or lack of an *isolated development environment* affected the software development process specifically regarding quality of work and time frame?
12) How has the presence of lack of an *isolated test lab* affected the software development process specifically regarding quality of work produced and on what time frame?
13) How has the presence or lack of code *documentation training* affected the software development process specifically regarding quality of work produced and on what time frame?
14) How has the presence or lack of *data storage and retrieval training* affected the software development process specifically regarding quality of work produced and on what time frame?
15) How has the presence or lack of *graphical user interface "GUI" training* affected the software development process specifically regarding quality of work produced and on what time frame?

Appendix B: Pre-CETAP Questionnaire Results

1) *How has the utilization or lack of project request documents affected the software development process specifically regarding quality of work produced and on what time frame?*

Howard:

The lack of project request documents has lead to many issues in project development and end user satisfaction. Since the programmer, who is often not involved in all phases of the request meetings, does not have a blueprint to work from directly, the wishes of the user are often missed. With this process, the user's desires and inputs frequently do not get to the programmer until the presentation of the first prototype. As such weeks can be lost before the correct design is developed, and some changes at this stage also require additional programming time to change the underlying structure of the program to accommodate the user's changes. Quality may suffer during the revision stages, as the programmer attempts to make up for lost time by taking shortcuts or spending less time on aesthetics.

Richartz:

Unclear user requirements, requests being sent to multiple people (programmers, supervisors),

Not requiring the use of project request documents has negatively impacted the quality of the finished product as well as extending the project's timeline. A project request document forces the user to spend some time thinking about "what" their department's needs for this software really are. Once completed, this document is submitted to a designated person for review. Not having this document in required has caused all of the following to happen:

1. Projects being submitted verbally
2. Projects being submitted directly to a programmer
3. Unauthorized programs being submitted

Each of the previous examples may affect either the quality of the product or the timeline or both.

Verbally submitting a request requires someone in our department to revisit the request with the requestor costing us time. Also, if the project requirements are written by someone other than the requestor, the final project may not meet the requesting department's needs.

Submitting a project directly to a programmer rather than submitting it to the supervisor may delay the project's timeline. The supervisor knows which of their programmers have the skill set or time to complete the request on the timeline which is needed. If the program was given to a programmer who has other

commitments the timeline would most likely slide. If the program was given to a programmer who didn't have a certain skill set, the program's quality may suffer as a result.

Submitting an "unauthorized" project, a project that has not been approved by the department's director, wastes the programming department's resources.

2) *How has the utilization or lack of project design documents affected the software development process specifically regarding quality of work produced and on what time frame?*

Howard:

The lack of design documents will generally add two to three weeks to a project as the programmer has to decipher the desired outcomes from mostly non existent design documents, based only on assumptions given the lack of use cases used (see below).

Richartz:

Coding without designing is dangerous! Taking the time to do project design provides two major benefits. First, it allows the programming team and the requestor's to finalize the requirements before anything is coded. Together the team can design the layout of any screens as well as iron out any potential workflow issues they may run across. This allows the requestor to know what the finished product will look like in advance. Second, it prevents project requirement "creep" – the constant adding of more features during coding. Finally, project design saves time during the coding phase. The programmer has a roadmap to follow and should run into relatively few project design questions during this phase.

Failing to use project design documents could end spell disaster. Your team may spend many hours creating a solution which doesn't meet the user's needs.

3) *How has the utilization or lack of project use case documents affected the software development process specifically regarding quality of work produced and on what time frame?*

Howard:

The lack of use cases contributes to the 2 to 3 week lag arising from the previous steps. Without use cases, all programmer decisions are based solely on assumption, and must frequently be corrected.

Richartz:

I took over management of a project where the programmer and supervisor didn't sit down and flush out the use cases. A week later the programmer told me the program was ready for testing. Since I was new to the project, I went to the requestor and worked with him to write up requirements to use during testing. Shortly after starting testing, I realized that the finished product only met one of the user's requirements and didn't handle the other three to the user's satisfaction. At this time, I setup a meeting with the requestor, programmer and myself to finalize the use cases so that we were all on the same page. The result of the meeting was that the programmer had to scrap approximately 50% of the code previously written.

Implementing the practice of project use cases serves three major purposes. First, the requestor has an opportunity to input his thoughts into the project. This will almost certainly result in a superior product. Second, it gives the programmer a roadmap for coding thus saving time. Third, use cases provide the testers a guide to follow to setup testing requirements. This also will improve the quality of the product.

4) *How has the utilization or lack of user manuals affected the software development process specifically regarding developer payroll hours devoted to post-release maintenance and user training?*

Howard:

Lack of user manuals adds at least 2 hours of general training time for each user, and precludes the effective use of group training. It also means there must be substantial follow-up to refresh users in operational practices they do not frequently use.

Richartz:

In the absence of user manuals, the only source of information relating to the software is the programming team. We support a variety of software programs some have user manuals and some do not. I have found that on average we spend more time researching questions and training users for programs which are missing this documentation. I have also found that the questions which arise for programs that have user manuals are a different type of question. Often these questions pertain to the program's functionality as opposed to "how do I" and in many cases these questions turn into future enhancement requests for the software.

5) *How has the utilization or lack of project error report documents affected the software development process specifically regarding quality of work produced and on what time frame?*

Howard:

Lack of standardized error reporting means that the programmer must spend time trying to isolate vague problems, and has to track down individual users for clarifications of problems before being able to tackle them. The programmer also has no history of issues with which to look for more global problems or identify issues specific to the computer configuration of the end user.

Richartz:

The utilization of error report documents and structured testing during projects can reduce the project timeline by up to twenty percent. The use of these documents reduces the amount of project management by having errors documented in a central location and the ability to view all of the project's outstanding issues.

Formal error reporting improves the quality of the current project as well as future coding projects. It allows programmers and management to view the types of issues that arise during the current project as well as past projects and detect any trends in the issues. Once a trend is detected, the team then can focus on the root cause and make improvements to the corresponding project phase to rectify the issue. Also, formal error documentation provides "proof of testing". If an issue arises later in the development process or once the project has gone "live", a team member can easily check to see if the issue had been tested during development. This information can then be used to improve testing requirements.

6) *How has the utilization or lack of project approval procedures affected the software development process specifically regarding quality of work produced?*

Howard:

The lack of approval process means that the programmer may chase tangents of their own that should not be included in a project, or tracks and modules that were no longer required but had not been removed from the project request. These issues can add many weeks to the development time of a project.

Richartz:

Not using project approval procedures can seriously affect the quality of work produced. A bad decision made early in the project can "snowball" into huge issue during the coding phase. For example, we currently support a program

which records a person's work duties. This program was written by a "non-programmer" who knew enough to be dangerous. In the past six months there have been issues reported on this program. While researching we found that the database design has redundancies, missing relationships, and was just poorly designed. Now, the department is asking for us to upgrade their program and it is nearly impossible without redesigning the database. If the person who created this program would have taken the time to go through the software development cycle and have someone review and sign off on each step the database flaws would have been discovered and corrected.

7) *How has the presence or lack of established project responsibility hierarchies affected the software development process specifically regarding quality of work produced time frame?*

Howard:

It allows programmers to make decisions without context or approval, and adds both time and bloat to most projects. That time depends on the creativity of the programmer and the scope of the project.

Richartz:

Lack of defined responsibilities adds considerably to the project's timeline. The project timeline could be bloated from 10% to 100% depending on the severity of the issues a lack of project responsibility hierarchies causes. If no roles are defined the team inevitably runs into feature "creep". Without having people designated as "end users" and involving them during analysis and design phases of the project, I have found that programmers will almost always continue to define the requirements during the coding phase. This is a fundamental process flaw. In these cases the programming is influencing the user requirements and use cases. This is backwards! The program requirements and use cases need to guide development to ensure the program meets the end users' needs.

Without structured responsibility, who is in charge? The failure to have defined roles during a project will appear in the final product. If the project manager, end user, programmer, tester roles aren't defined at the beginning of a project quality will suffer. In extreme cases, the quality will be so poor that the finished product won't meet enough of the customer's needs and the project be scraped or totally redone. Cases that I have seen are far less severe. I have seen projects which have been deemed as "ready for use" that were missing error checking, used incorrect calculations, included misspellings, and other issues. It took anywhere from a day to a few weeks to correct all of the project's issues before releasing the software. If responsibility hierarchies were used a project manager or tester would have found these issues much earlier in the process.

8) *How has the utilization or lack of a peer review process affected the software development process specifically regarding quality of work and on what time frame?*

Howard:

The lack of peer review eliminates the first phases of quality control and allows for the introduction of errors into the process caused by the focus of the programmer overriding the logical process required for complete testing. How much time is added depends on the scope of the project.

Richartz:

It is difficult to quantify the amount of time utilizing the peer review process saves because the time saved could be during the current project or future projects. I estimate that our team has saved anywhere from a day to a week by using the peer review process on recent projects.

Utilizing the knowledge their peers possess allows programmers to save time and improve the quality of work. Examples of the peer review process include:

- saving time include collaborating to solve tough issues
- reuse of existing code
- peer reviewer gaining knowledge of project which can be applied during the testing phase.

Examples of improved quality include:

- having the peer review provide alternate "better" solutions
- programmers work together to give their projects "consistency"
- review the design for improvements

9) *How has the utilization or lack of a quality assurance "QA" process affected the software development process specifically regarding quality of work produced and on what time frame?*

Howard:

This has resulted in the release of code that was not completely functional because the programmer was certain of their completeness and accuracy, but had overlooked some flaw in the process because they were testing against the expected outcomes rather than the user factors of ignorance, laziness, and incompetence. This adds time to fix issues, and can create friction with the end user if they feel the issue is severe enough. The actual severity of the issue may not actually play into the equation, only the user perceptions.

Richartz:

QA is probably the most commonly skipped step in the development process yet I have found that the QA step is the greatest factor in determining customer satisfaction. Software can have a strong concept, great workflows, and time-saving functionality but if the software goes to your customer with bugs that could have been found in a few hours of QA you will have lost credibility in your customers' mind. I remember reading a quote a few years ago on a customer satisfaction survey which pretty much sums up how important QA is..."How is this software going make my job easier and save me time if I have to reboot my machine five times a day because of it." I use this as a reminder that the customer is the reason all software projects are implemented.

The only time when QA is noticed is when it fails. If QA isn't done thoroughly the customer will let the project team know it.

The lack of QA may cause the release of software that has "bugs", doesn't meet the customers' needs, or is of poor quality.

The use of the QA process saves both time and improves quality on projects, both present and future. All of the issues a user would find in using the software would be found before its release. Not only does this negatively reflect on you and your department, it costs TIME! Take an example where you release the software to 10 users and QA wasn't used. Now, imagine that each user finds five issues with the software. After hearing their complaints and trying to figure out what the "true" issue is, how many issues do you have? 50? 20? Most likely you will have only five to ten but they were all reported in different ways. Using QA you would have had a technically knowledgeable find the errors and be able to report them to the programmer in a manner where the error could be quickly corrected. Without QA, there's no telling how much time you will spend working with the users and correcting the issue.

10) *How has the utilization or lack of a user approval "UA" process affected the software development process specifically regarding quality of work produced and on what time frame?*

Howard:

This also relates to the user perception of the problems. If they know they are testing the program and can make suggestions that will be more cooperative than if it is dumped on their computers and they are told to go with it. This may impact training time and follow-up issues as much as programming time to restructure to the users desires after the fact.

Richartz:

Recently, the projects we have been working on have been for a small group of users, between 1 and 10, so the UA process hasn't been as viable to use. The benefit of having a small group of users is that the entire group can be your "UA

group" but in reality you just released the software without going through a formal UA process.

The use of UA is important when the finished software will be released to a large user group. The users involved this testing are a litmus test for how the software grades out. If the response is less than positive, the project team can work closely with t he users involved in UA to modify the software to meet their approval. If the response is positive, a buzz and excitement for the software will be created by the UA group to their peers.

11) *How has the presence or lack of an isolated development environment affected the software development process specifically regarding quality of work and time frame?*

Howard:

The lack of isolated environments adds to the number of errors not found until roll-out, as external conditions and interactions are not tested until post deployment. This makes debugging more difficult and time consuming, as well as frustrating the user and embarrassing the programmer.

Richartz:

Our department is currently developing software without an isolated development environment as well as a standard method for code versioning. The implementation of an isolated development environment would resolve the issue of not having of a standard method of code versioning. In reality, an IDE would resolve two department issues.

An isolated development environment would save our team time on projects for a couple of reasons. First, it would give our developers a central location to do coding. Collaboration and cooperation would be much easier for our development team if they all worked in the same "sandbox". Second, you can erase your mistakes quickly. If something were to go wrong or if they wanted to rollback changes in a certain file they could "reload" the previous version within minutes. Finally, time would be saved in project management because the project manager could easily look at the sandbox and find out a project's progress.

12) *How has the presence of lack of an isolated test lab affected the software development process specifically regarding quality of work produced and on what time frame?*

Howard:

Lack of an isolated lab compounds the extent of the isolated platform issue, as it makes it more difficult to trace issues across multiple platforms simultaneously.

Richartz:

> The lack of an isolated test lab has affected the quality of software released because we don't have multiple platform and operation system combinations to test on. This results in testing being performed on a single user machine and passing with flying colors. The issue is that one machine is not representative all of the machines or operating systems we support around campus. In the last year we ran into an issue where a program we created did not run a group of machines with a certain operating system. Since we did not have an isolated test lab, we needed to go out into the field to troubleshoot this issue. The testing ended up taking around two hours and we still hadn't resolved the issue so we created a test box. If we would have had a test box as part of the lab we could have done extensive testing to rule out potential causes before we went on-site. This would have made better use of our resources, made us look more knowledgeable to our customers, and ultimately saved us lots of time.

13) *How has the presence or lack of code documentation training affected the software development process specifically regarding quality of work produced and on what time frame?*

Howard:

> Lack of consistent comments in the coding require that each new programmer relearn all of the existing programs, and their functionality, before being able to fix or alter it. Standardized coding documentation would easily remove 1 to 2 months from the acquisition cycle of medium to large scale projects.

Richartz:

> Code documentation doesn't affect the quality of work produced on the initial release. However, it has great effect on the quality of subsequent versions of the software because in most cases the programmer writing the updated code isn't the same programmer that wrote the initial version. The best documentation for the new programmer is what exists in the code. If a programmer can't figure out what his predecessor's code is doing or how he/she can modify it then this costs our team time. An even greater consequence is that the changes implemented reduce the value of the software or make the software harder to maintain. Code documentation training would greatly reduce the time spent trying to figure out "what this code does" and provide consistency in documentation over the project's lifespan.

14) *How has the presence or lack of data storage and retrieval training affected the software development process specifically regarding quality of work produced and on what time frame?*

Howard:

This makes it harder to find, organize and retrieve in a timely manner, historic versions of code, makes back-ups more difficult, and impedes the flow of knowledge is the resources cannot be located. In addition, it is a risk to the data and process integrity if backups are not being made, or cannot be located when needed.

Richartz:

Our department has a definite need for training on how to store and retrieve information in databases. Past projects were developed using the programmer's preferred database rather than the database best-suited for the project. Over the past six months we have been focusing on making improvements during design phase of projects including database selection and design to improve software quality. The logical next step is to carry the improvements made in the design phase into the programming phase. Creating a training program and documentation on data storage and retrieval would be an integral part of the next step. Databases are the foundation of all programs and all great programs need a solid foundation.

15) *How has the presence or lack of graphical user interface "GUI" training affected the software development process specifically regarding quality of work produced and on what time frame?*

Howard:

Lack of consistency in the user interface hampers customer satisfaction and impedes their ability to flow from application to application or between versions, as they may have to learn a new interface with every version of the program.

Richartz:

I do not have anything to discuss on this topic so… No comment.

There needs to be a consistent look and feel of products that are produced by a single department or company. This doesn't mean that the programmer can't use any creativity or that all programs must use the exact same colors. Here are a few things that are expected:

- Design of similar screens in different programs need to be consistent. For example, if you have a demographics screen in both programs they should look close if not exactly the same.
- Naming conventions on buttons or actions should be carried through the suite of programs. One program shouldn't have a "submit" button and

another have an "execute" button. They perform the same action and should be named accordingly.

Appendix C: Post-CETAP Questionnaire Results

1) *How has the utilization or lack of project request documents affected the software development process specifically regarding quality of work produced and on what time frame?*

Howard:

The lack of project request documents has lead to many issues in project development and end user satisfaction. Since the programmer, who is often not involved in all phases of the request meetings, does not have a blueprint to work from directly, the wishes of the user are often missed. With this process, the user's desires and inputs frequently do not get to the programmer until the presentation of the first prototype. As such weeks can be lost before the correct design is developed, and some changes at this stage also require additional programming time to change the underlying structure of the program to accommodate the user's changes. Quality may suffer during the revision stages, as the programmer attempts to make up for lost time by taking shortcuts or spending less time on aesthetics.

Having request documents available significantly shorted the development process, as all parties involved knew what they are trying to achieve, and as a result there were usually fewer design meetings, and may fewer changes in process. Also, the quality is better as the programmer was actually working on the desired outcomes from the beginning, which also lessened the frustration of the programmer, which also aided in quality.

Richartz:

Requiring the project initiator to fill out the project request documents helped in two aspects. First, filling out the documents forced the initiator really think about project details. Second, the software project manager was able to review the project request before meeting with the project initiator thus making the initial project meeting more productive. For this project, the project initiator and project manager were able to resolve any issues and agree on the high-level requirements during their first meeting. The utilization of the project request documents before and during the project "kickoff" meeting saved upwards of a week in the project timeline. Previous projects of similar size have required anywhere from two to four meetings scheduled over a week or two before the project could move into the design phase.

2) *How has the utilization or lack of project design documents affected the software development process specifically regarding quality of work produced and on what time frame?*

Howard:

The lack of design documents will generally add two to three weeks to a project as the programmer has to decipher the desired outcomes from mostly non existent design documents, based only on assumptions given the lack of use cases used (see below).

Having the documents available to all involved personnel at the beginning easy cut 60 to 75% of the development time, and significantly improved the quality of work since all efforts were pointed in the same direction.

Richartz:

The utilization of project design documents has improved the quality of work produced as well as shortened the project's timeline. The design document which was created for this project was reviewed by the project team and the project initiator. During the review, a few changes were requested by the team. The document was revised and then reviewed a second time before coding started. Had a design document not been created for this project, the design changes wouldn't have been caught until the program was being tested. The estimated savings in making the changes requested before coding started was at a minimum of 15 hours and nearly a week on the timeline. In addition to the time savings, the design document gave the project initiator a good idea of the look and feel of the program before a line of code was written. At the completion of the project, the project initiator told me how nice it was to have a vision of what the program was going look like early on and how happy he was to see a finished product that reflected his vision.

3) *How has the utilization or lack of project use case documents affected the software development process specifically regarding quality of work produced and on what time frame?*

Howard:

The lack of use cases contributes to the 2 to 3 week lag arising from the previous steps. Without use cases, all programmer decisions are based solely on assumption, and must frequently be corrected.

Having use cases available has easily halved the development time, as the programmer was able to "see into the user's mind" and know what expectations were for the program instead of wondering, or having to schedule follow up meetings in order to resolve these questions. This savings alone can represent weeks on a mid sized project.

Richartz:

By creating a use cases document for this project we eliminated function "creep". Once the use cases were written, the project team and project initiator met and discussed every use case. During this meeting we found and corrected several minor "workflow" issues. If we would have found these issues during testing we would have added around 20 hours to this project (8-12 programming, 4-6 testing, and 2-4 management). This meeting to review the use case document was invaluable. During the meeting, we found some issues that would have given us problems later in the project and at the end the project manager and project initiator signed off on the use cases. Doing this step improved the quality of the finished product as well. We were able to make changes on paper and keep the programming and workflows straightforward instead of trying to "squeeze in" more functionality in after the bulk of the program has been coded. This resulted in a much cleaner solution.

4) *How has the utilization or lack of user manuals affected the software development process specifically regarding developer payroll hours devoted to post-release maintenance and user training?*

Howard:

Lack of user manuals adds at least 2 hours of general training time for each user, and precludes the effective use of group training. It also means there must be substantial follow-up to refresh users in operational practices they do not frequently use.

The savings here can be marginal, as the users frequently refuse to "RTFM". For those who are willing to read the manual, almost all training issues were removed from the timeline.

Richartz:

This programming project produced both a user manual and a technical specifications document. The user's manual was sent to the users of this program in lieu of training. If the users had questions or needed training after reading the user's manual then our department would help them one-on-one. So far none of

the users have asked for one-on-one training. The technical specifications are especially important for post-release maintenance. This document allows a new programmer to get a feel for the application without reading every line of code. This document will become more valuable when an enhancement request is submitted for this program. To date, the time spent creating the documentation has paid for itself a few times over and the savings will continue to grow.

5) *How has the utilization or lack of project error report documents affected the software development process specifically regarding quality of work produced and on what time frame?*

Howard:

Lack of standardized error reporting means that the programmer must spend time trying to isolate vague problems, and has to track down individual users for clarifications of problems before being able to tackle them. The programmer also has no history of issues with which to look for more global problems or identify issues specific to the computer configuration of the end user.

Standardized error reporting made debugging faster, as the form required the user to provide specific information, and as they adapt to the format, they began to know what information to look for when an error occurs. It also aided in the communication between programmer and user, so that they worked together more efficiently, and users were encouraged to report all of the issues as they arose, rather than ignoring "the little ones" or reporting in large bunches.

Richartz:

Error reports were created by both the programmer and QA'er during this project. Being able to view these error reports at any point during the project from my desk gave me as a project manager a fast way to see the project's status, the progress made, and any hang-ups encountered. The error reports saved me a couple of hours of management time during this project. Determining how much of an effect the utilization of project error reports had on finished product is hard to quantify. I will say that no "errors" have been found by users working with the finished application. Also, the QA'er utilized the programmer's error reports when creating testing scenarios. Overall, the use of error reports played a major role in creating such a solid finished product.

6) *How has the utilization or lack of project approval procedures affected the software development process specifically regarding quality of work produced?*

Howard:

The lack of approval process means that the programmer may chase tangents of their own that should not be included in a project, or tracks and modules that were no longer required but had not been removed from the project request. These issues can add many weeks to the development time of a project.

Requiring approvals eliminated a lot of tangential issues, and also discouraged users from submitting irrelevant requests, so the both the programmers and administrators spent less time reviewing requests, and can concentrate on the actual program and its completion. Depending on the idiocy of the end users, this process can easily cut 20 to 30 percent off the development run.

Richartz:

The formalization of approval procedures this project allowed the groundwork to be laid for a successful project. Using the project request document in conjunction with the approval procedures forced the project initiator and project manager to explicitly define the project's goals, high level requirements and scope. The approval procedures ensured that the entire project team had the same understanding going into the design phase of the project.

7) *How has the presence or lack of established project responsibility hierarchies affected the software development process specifically regarding quality of work produced time frame?*

Howard:

It allows programmers to make decisions without context or approval, and adds both time and bloat to most projects. That time depends on the creativity of the programmer and the scope of the project.

As with the approval process, the existence of hierarchies aided in direction of the process, and also made setting acceptance standards possible. Without the hierarchies, any gains in approval can be lost to chain of command fights.

Richartz:

Defining the roles of every person involved with project was beneficial in many ways. First and probably most important was the fact that everyone knew their role and responsibilities. They also knew who they should go to if they had a question or an issue. This project utilized each project member's strengths which resulted in a product of higher quality. For example, the "end users" were involved during analysis, design, and user approval phases. If a programmer had an issue, he knew which team member could answer his questions. This project also included a team member whose sole responsibility was to do quality

assurance and a group of user testers. Having this group of people test every button, option, and workflow and report their findings back to the development team greatly improved the quality of the finished product. The testing found between 15-20 "minor" issues and 2 "major" workflow issues. Had the team not had defined workflows these issues might never have been found.

8) *How has the utilization or lack of a peer review process affected the software development process specifically regarding quality of work and on what time frame?*

Howard:

The lack of peer review eliminates the first phases of quality control and allows for the introduction of errors into the process caused by the focus of the programmer overriding the logical process required for complete testing. How much time is added depends on the scope of the project.

Peer review allowed for bad code and coding practices to be detected before there was a large impact on the process, and had significant carry over in savings as the coding practices became both standardized and consistent across programmers.

Richartz:

The peer review process greatly improved the quality of the finished product. Two programmers performed peer review during this project. Each programmer started their peer review a week after the project had started. Once their individual assessments were completed, the peer reviewers, project programmer, and project manager met to discuss the results. The benefits even exceeded my expectations as a project manager. The project programmer benefited from his peer's review by being able to reuse some existing menu code and received a few alternative approaches to some workflows. The peer reviewers liked the process because they were able to look at the project from a different perspective. Even after the formal peer review was completed, the programmer continued to use the other programmers as resources. This collaboration was great to see.

It is difficult to quantify the amount of time utilizing the peer review process saves because the time saved could be during this project or future projects. I estimate that our team has saved around 15-20 hours worth of programming time by using the peer review process during this project.

9) *How has the utilization or lack of a quality assurance "QA" process affected the software development process specifically regarding quality of work produced and on what time frame?*

Howard:

This has resulted in the release of code that was not completely functional because the programmer was certain of their completeness and accuracy, but had overlooked some flaw in the process because they were testing against the expected outcomes rather than the user factors of ignorance, laziness, and incompetence. This adds time to fix issues, and can create friction with the end user if they feel the issue is severe enough. The actual severity of the issue may not actually play into the equation, only the user perceptions.

QA easily detected a couple dozen problems before they ever make it to the end users. Not only did this save time, it also improved the end user's perception of quality because they didn't see the bugs in the software prior to the release.

Richartz:

QA on this project found between 15-20 "minor" issues and 1 "major" workflow issue. The total amount of time spent on QA in this project was under 20 hours which encompassed three rounds of QA. As a project manager, this time spent paid for itself multiple times over and actually probably saved our team a few days worth of work.

10) *How has the utilization or lack of a user approval "UA" process affected the software development process specifically regarding quality of work produced and on what time frame?*

Howard:

This also relates to the user perception of the problems. If they k now hey are testing the program and can make suggestions, that will be more cooperative than if it is dumped on their computers and they are told to go with it. This may impact training time and follow-up issues as much as programming time to restructure to the users desires after the fact.

User approval allowed for the assignment of responsibility and accountability in the approval process. By requiring the user to approve steps, they no longer had the ability to throw in last minute changes to structures they have already approved and blame it on the process. This is especially useful when coupled with the use cases and design documents.

Richartz:

The user approval process allowed a few "power users" to test out the finished product before we released it to the public. During this testing one of our UA'ers found an issue with one of the workflows. They commented on how the current

workflow didn't quite mesh with how they would be using the program in their daily work. We took this feedback to the programmer and revised the workflow. Taking this time for UA improved the quality of the software which resulted in a more "positive" release of the software.

11) *How has the presence or lack of an isolated development environment affected the software development process specifically regarding quality of work and time frame?*

Howard:

The lack of isolated environments adds to the number of errors not found until roll-out, as external conditions and interactions are not tested until post deployment. This made debugging more difficult and time consuming, as well as frustrating the user and embarrassing the programmer. An isolated development environment improved user perception and cut down on time spent in the development process.

This statement remains true post training.

Richartz:

The IDE was helpful during this project. It allowed the peer reviewers and QA'er a central location to test the project without interference. This project wasn't the best indicator of how an IDE would benefit the software development process for a few reasons. First, only one programmer was working on this project so we didn't need to worry about contention. Also, this project was new so we didn't have to worry about changing code that was already in production. I am confident that future development projects using an IDE will better demonstrate time savings and quality improvement.

12) *How has the presence of lack of an isolated test lab affected the software development process specifically regarding quality of work produced and on what time frame?*

Howard:

Lack of an isolated lab compounds the extent of the isolated platform issue, as it makes it more difficult to trace issues across multiple platforms simultaneously.

Programmer training does not affect this process, except where that training convinces the programmer of the need to use the isolated lab to improve their projects.

Richartz:

The isolated test lab shortened the amount of time it took to test this project. Both the QA'er and UA'er spoke on how great it was to have a dedicated test box that they didn't have to worry about "messing up". Also, the ability to "reload" the test box to its original state saved hours in setting up the testing.

13) *How has the presence or lack of code documentation training affected the software development process specifically regarding quality of work produced and on what time frame?*

Howard:

Lack of consistent comments in the coding require that each new programmer relearn all of the existing programs, and their functionality, before being able to fix or alter it. Standardized coding documentation would easily remove 1 to 2 months from the acquisition cycle of medium to large scale projects.

Effective and standardized code documentation can allow for an extensive project to be turned over to a new programmer or team in a matter of days instead of months that could be required without it.

Richartz:

Code documentation training provided the programmer the tools to create more useful documentation in less time. I compared this project code to a previous project from this programmer before the training and the improvements were immediately noticed. The code was more structured, easier to read, the variables had a standard naming convention. The time savings will continue to be reaped in future versions of this application.

14) *How has the presence or lack of data storage and retrieval training affected the software development process specifically regarding quality of work produced and on what time frame?*

Howard:

This makes it harder to find, organize and retrieve in a timely manner, historic versions of code, makes back-ups more difficult, and impedes the flow of knowledge is the resources cannot be located. In addition, it is a risk to the data and process integrity if backups are not being made, or cannot be located when needed.

Training in this issue only improves quality of work if followed through by the programmer, and can result in the savings of hours to days, depending on how much information would have been lost if not backed up.

Richartz:

The data storage and retrieval training was extremely beneficial for this programmer. Since the programmer had not worked with SQL as the underlying database in a project before all of the information presented in the training was new to him. The documentation and training exercises allowed him to learn about and "play" with SQL before using it in a "live" program. The training was time well spent. The programmer didn't have any questions pertaining to data storage or retrieval during the project. In reviewing the code written I could see that the programmer applied this training to the project.

15) *How has the presence or lack of graphical user interface "GUI" training affected the software development process specifically regarding quality of work produced and on what time frame?*

Howard:

Lack of consistency in the user interface hampers customer satisfaction and impedes their ability to flow from application to application or between versions, as they may have to learn a new interface with every version of the program.

Having a consistent approach to the UI made it easier for the user to adapt, understand the operation of the program, reduced the number of false error reports, and aided in the transition of a project to another programming team.

Richartz:

One of the main goals of our team is to create consistency throughout the applications we write. Graphical user interfaces are the most visible items to the user so creating consistency between GUIs is extremely important. Getting a consistent look and feel to all applications written by different programmers can be difficult. This document is a great tool for accomplishing consistency.

The benefits of the GUI training this programmer participated could be seen throughout this project. In previous projects, this programmer liked to be a little too creative in the GUI screens he created. He often sacrificed ease of use for aesthetics. In this project, the programmer used the training document as a guide for GUI development. As a result, the program's GUIs were clean, straightforward, easy-to-use, and consistent throughout. Also, this program has a similar "look" of other programs our team has created.

Appendix D: Coding Standards - VB Style Guidelines

Coding Standards VB Style Guidelines

Student Life Services - Technical Services
Revised: May 12, 2004

Version History			
Date	**Name**	**Description of Revision**	**Revision #**
3/25/2004	Jane Zielieke	New Document	1.0

Overview

Coding standards promote consistency, readability, and team ownership of code. Standards are not about personal preference they are about assuring consistent, low-overhead access to by developers and clients. The only true coding standard is whatever standard your current project/team/company/client is using, so please leave non-Technical Services standards behind. Most developers see inconsistent code as a sign of incompetence. Given that many developers will see your code, we must strive for a consistent, high-quality look to facilitate future development ventures.

This document contains the current Visual Basic programming standards for Technical Services. As this document evolves, additions and deletions will be made. Approval signatures and documenting these changes are important for consistency and clarity and are addressed within sections of this document.

To add sections to the table of contents:

1. Follow the instructions for updating the table of contents below
2. Go to the location in text that you would like to add a level item
3. Select the appropriate level
4. Add level item
5. Begin paragraph
6. Set paragraph to "Body text"
7. Begin body text

To update the table of contents:

1. View > Toolbars > Outlining
2. On the new tool bar, click "Update TOC"

Standards

Technical Services, as a member of a Microsoft campus, adopts coding standards similar to those used by Microsoft. These are available at:
http://msdn.microsoft.com

We adopt these standards because:
As an industry standard, outside developers will be familiar with the overall look and style of Technical Services, keeping us in line with System mandate.
As an educational institution, we are committed to providing our employees with an informative work experience applicable post-graduation.

Naming Conventions

Constant and Variable Names

Scope and data type are important descriptors when naming constants and variables. Remember, it is likely that you will not be the only one updating, fixing, and maintaining your code so it needs to be as easy to understand as possible, as quickly as possible.
Scope describes to anyone looking at the code whether the variable is used within a procedure (sub or function), a module, or a whole program. A more detailed description of scope is as follows [1]:

Scope	Declaration	Visible in
Procedure-level	'Private' in procedure, sub, or function. Cannot be accessed from anywhere else.	The procedure in which it is declared
Module-level	'Private' in the declarations section of a form or code module (.frm, .bas)	Every procedure in the form or code module
Global	'Public' in the declarations section of a code module (.bas)	Everywhere in the application

Global variables should be used only when there is no other convenient way to share data between forms. Ideally, global variables should be declared in a single code module (.bas). It is good practice to limit the scope of a variable to only where it is absolutely necessary.
Data type indicates the general purpose for the variable. Including both aspects is important for readability and maintainability. The name of a variable is a combination of scope then data type. The naming conventions for constants and variables used at Technical Services are listed below [1]:

Suggested Prefixes for Scope:

Scope	Prefix
Global	g
Module-level	m
Local to procedure	None

Suggested Prefixes for Variable Data Types:

Data type	Prefix
Boolean	bln
Byte	byt
Collection object	col
Currency	cur
Date (Time)	dtm
Double	dbl
Error	err
Integer	int
Long	lng
Object	obj
Single	sng
String	str
User-defined type	udt
Variant	vnt

The names you give to variables and constants:

- ✓ Must begin with a letter
- ✓ Must contain only letters, numbers, and the underscore character (_); punctuation characters and spaces are not allowed
- ✓ Must be no longer than 40 characters
- ✓ Must use capital letters for the first letter in each word. Exceptions include: scope and data type prefixes, constants.
- ✓ Must be in all capital letters if declared as a constant.
- ✓ Must be named in the following way: *<prefix for scope><prefix for data type><descriptor>*. Examples:
- ✓ A global variable of type string may be called gstrConnect.
- ✓ A module-level variable of type double may be called mdblLength.
- ✓ A procedure level variable of type error may be called errFileNotFound.

Object Names

As with variables, its important to name controls in a consistent and descriptive way so that everyone, including yourself, can easily identify in the code, firstly, that an object is a control, secondly, what type, and thirdly, any other pertinent information such as its essential purpose. Toward that end, the names you give to objects [2]:

- ✓ Must begin with a letter
- ✓ Must contain only letters, numbers, and the underscore character (_); punctuation characters and spaces are not allowed
- ✓ Must be no longer than 30 characters
- ✓ Must use capital letters for the first letter in each word except for type prefixes (mixed case convention)
- ✓ Naming conventions for new objects must be discussed with your software development supervisor for subsequent inclusion in this document. The names you

give to objects will contain the suggested prefixes listed below in alphabetical order (commonly used controls are highlighted for quick reference) [3]:

Suggested Prefixes for Controls:

Control type	Prefix	Example
3D Panel	pnl	pnlGroup
ADO Data	ado	adoBiblio
Animated button	ani	aniMailBox
Check box	chk	chkReadOnly
Combo box	cbo	cboEnglish
Command Button	cmd	cmdExit
Common dialog	dlg	dlgFileOpen
Communications	com	comFax
Control (used within procedures when a specific type is known)	ctr	ctrCurrent
Data	dat	datBiblio
Data-bound combo box	dbcbo	dbcboLanguage
Data-bound grid	dbgrd	dbgrdQueryResult
Data-bound list box	dblst	dblstJobType
Data combo	dbc	dbcAuthor
Data grid	dgd	dgdTitles
Data list	dbl	dblPublisher
Data repeater	drp	drpLocation
Date picker	dtp	dtpPublished
Directory list box	dir	dirSource
Drive list box	drv	drvTarget
File list box	fil	filSource
Flat scroll bar	fsb	fsbMove
Form	frm	frmEntry
Frame	fra	fraLanguage
Gauge	gau	gauStatus
Graph	gra	graRevenue
Grid	grd	grdPrices
Hierarchical flexgrid	flex	flexOrders
Horizontal scroll bar	hsb	hsbVolume
Image	img	imgIcon
Image combo	imgcbo	imgcboProduct
ImageList	ils	ilsAllIcons
Label	lbl	lblHelpMessage
Lightweight check box	lwchk	lwchkArchive
Lightweight combo box	lwcbo	lwcboGerman
Lightweight command button	lwcmd	lwcmdRemove
Lightweight frame	lwfra	lwfraSaveOptions
Lightweight horizontal scroll	lwhsb	lwhsbVolume

bar		
Lightweight list box	lwlst	lwlstCostCenters
Lightweight option button	lwopt	lwoptIncomLevel
Lightweight text box	lwtxt	lwtxtStreet
Lightweight vertical scroll bar	lwvsb	lwvsbYear
Line	lin	linVertical
List box	lst	lstPolicyCodes
ListView	lvw	lvwHeadings
MAPI message	mpm	mpmSentMessage
MAPI session	mps	mpsSession
MCI	mci	mciVideo
Menu	mnu	mnuFileOpen
Month view	mvw	mvwPeriod
MS Chart	ch	chSalesbyRegion
MS Flex grid	msg	msgClients
MS Tab	mst	mstFirst
OLE container	ole	oleWorksheet
Option button	opt	optGender
Picture box	pic	picVGA
Picture clip	clp	clpToolbar
Progress Bar	prg	prgLoadFile
Remote Data	rd	rdTitles
RichTextBox	rtf	rtfReport
Shape	shp	shpCircle
Slider	sld	sldScale
Spin	spn	spnPages
StatusBar	sta	staDateTime
SysInfo	sys	sysMonitor
TabStrip	tab	tabOptions
Text box	txt	txtLastName
Timer	tmr	tmrAlarm
Toolbar	tlb	tlbActions
TreeView	tre	treOrganization
UpDown	upd	updDirection
Vertical scroll bar	vsb	vsbRate

Suggested Prefixes for Data Access Objects:

Database object	**Prefix**	**Example**
Container	con	conReports
Database	db	dbAccounts
DBEngine	dbe	dbeJet
Document	doc	docSalesReport
Field	fld	fldAddress
Group	grp	grpFinance

Index	idx	idxAge
Parameter	prm	prmJobCode
QueryDef	qry	qrySalesByRegion
Recordset	rec	recForecast
Relation	rel	relEmployeeDept
TableDef	tbd	tbdCustomers
Workspace	wsp	wspMine

Function & Event Names

Function and event names follow many of the same naming conventions that objects and variables do. Technical Services utilizes the following guidelines for naming functions and events:

- ✓ Utilize names dictated by the Visual Basic editor. Double-clicking on an object on a form creates an event in the code.
- ✓ Use mixed case naming when creating functions or subroutines.
- ✓ Use descriptive and appropriate names.
  - o Ex: Appropriate: ResetValues
    Inappropriate: CleanUpStuff
- ✓ Names are not to exceed 30 characters in length.

Code Style

Just as there are millions of programmers, there are millions more ways of putting together code to accomplish any task. Programmers, on the average, tend to consistently choose a particular pattern to laying out code. While the definition of good programming style is largely subjective, two factors determine what good code is: simplicity and readability. Simplicity refers to the clarity of the code and the ease with which others are able to understand it. Readability refers to the ease with which someone can scan the code and understand it.

In the end, these two factors determine how easily programmers that follow you can maintain what you've written. Following a standard set up guidelines prepares you for future commercial software development and produces quality applications. The general-purpose style standards used by Technical Services are intended to provide the minimal requirements necessary to accomplish the purposes discussed above while allowing you, as the programmer, free to create the flow of the application. The style elements Technical Services considers necessary are as follows:

General Guidelines

- ✓ Simplify code as much as possible [4]
- ✓ Avoid excessive nesting logic
- ✓ Declare only one variable per line
- ✓ Execute only one command per line
- ✓ Assemble calculations or strings prior to use/display. This technique is useful in the debugging process.
  - o Ex:

```
strMessage = "This string may be assembled " _
& "in different procedures. Putting it together beforehand " _
```

```
& "will allow you to debug and trace errors back to its source."
msgbox strMessage
```

- ✓ Keep an eye on system resource usage. Some of the systems your application will run on may not have the resources to handle a "resource hog" in a timely manner.
- ✓ Remove "dead" code immediately upon discovery. After many revisions, threads of code may become isolated and never have the potential to run. Removal of this code promotes readability, simplicity, and cuts down on application size.
- ✓ Avoid deeply nested function calls.

Indentation

Indents are used to logically organize code, to visually appealing and indicate logical groupings of code. For instance, code subject to an *if* statement are indented to quickly and easily indicate to the reader under what conditions the code segment is used. The following list of guidelines detail where, when, and how to use indentation to make your code more readable and maintainable [5]:

- ✓ Indent all code and comments within a procedure at least one tab stop (Visual Basic Editor uses a four-space tab stop by default). The only code lines that are not indented are the beginning and ending of the procedure and line labels used.
- ✓ If you use line breaks to format a procedure's argument list, use tab stops to indent the arguments and their data-type declarations so that they are aligned with the first argument on the list.
- ✓ Indent declared variables to one tab stop. Declare only one variable per line. Indent each variable's data-type specifier so that the variable data types are aligned.
- ✓ Indent control structures at least one tab stop from the preceding control structure (nested ifs, loops, etc).
    - o Ex:

```
If State = "Wisconsin" then
        If City = "Menomonie" then
                Location = "local"
        End if
End if
```

- ✓ If you use a line-continuation character (_) to break a line of code, indent the new line one extra tab stop. This indicates to readers that the lines belong and function together.
    - o Ex:

```
strQuery = "Select Username, FirstName, LastName, Street, State," _
& " City, ZIP from Customer where ClientID = 4"
```

- ✓ Indent line continuations one further than the beginning of the code.
- ✓ Indent comment lines to the same degree as the code it refers to.

Line Breaks

Line breaks contribute to the readability of the code. If used appropriately, code segments are easier to read because the reader needn't scroll horizontally to view it in its entirety [6].

- ✓ Code must not extend beyond the right edge of the code window. Taking resolution differences into account, a single line of code should not contain more than 70

characters. The Visual Basic editor indicates the column of the code at which the cursor is located.

- ✓ Use the underscore (_) and ampersand (&) characters to continue lines that exceed 70 characters.
  - o Ex:

```
If strLocation <> "Canada" and strLocation <> "United Kingdom" and _
        strLocation <> "Mexico" then
        ...
end if
```

  - o Ex:

```
strQuery = "Select Username, FirstName, LastName, Street, State," _
& " City, ZIP from Customer where ClientID = 4"
```

Commenting Code

Along side structure and formatting, commenting code greatly contributes to the readability of code. Comments are useful at the form header, procedure, and line levels describing in increasing specificity what each segment does [7].

General Guidelines

- ✓ Comments should be accurate and descriptive. While tempting, cute or imaginative comments must be avoided. Others following you will not find them informative or useful to their task as software developers.
- ✓ Blocks of code yet to be completed should be denoted as "TBD" (to be developed) at the beginning of the block. With this convention, a simple search of the code for "TBD" using the find function would easily reveal areas for necessary development and reduce the likelihood of incomplete code reaching the users.

Module Level

Each module or form must include a header comment block that summarizes it. The header should follow this format:

Original author:
Purpose:
Input modules:
Output modules:
Modifications:
Date:
Author:
Description:

Procedure Level

- ✓ Each procedure must include a brief summary of the purpose of the code segment.
  - o Ex:

```
'Convert Celsius to Fahrenheit

Sub CelsiusToFahrenheit (temp as Double)
        temp = (1.8 * temp) + 32
```

End sub

Line Level

- ✓ Comments should describe the functional characteristics of the procedure (what it does).
  - Ex: 'set counter to read the next resource in the resource list
    i = i + i
- ✓ Comments should *not* describe the implementation details of the procedure (how it works). Anyone reviewing your code will presumably be qualified to do so and will find this unnecessary.
  - Ex: 'increment counter
    i = i + i
- ✓ Every important variable declaration should include an inline comment describing its use and/or purpose.
- ✓ Blocks of code with a similar purpose or ones that need clarification need comments. Not every single line needs a comment. Use reasonable discretion when considering whether or not to comment a particular line. Use the following rule of thumb: if you have doubts, it most likely needs one.

Error Handling

Ideally, users would always do what they should and programs would always work without any concern for errors of any kind. However, this is obviously unrealistic hence the need for error handling. Errors may be handled in many ways, depending on the function and error raised; however, some general principles should be followed when addressing errors. Technical Services adopts the following standards for error handling:

Error Trapping

- ✓ Use the "On Error GoTo <form name><function name/sub name/object name + event>Err" naming convention for the error handler. This must be the first line of code within a procedure.
- ✓ Avoid "On Error Resume Next". This method restricts the applications ability to report problems to the user and to you for debugging.
- ✓ Error handling should follow the following format:
  1. Function declaration
  2. Error handling instruction - On Error GoTo
  3. Function code
  4. Exit sub line label
  5. Exit sub code
  6. Error handling line label
  7. Error reporting function call (called "LogError" in the example below)
  8. Error handling code - Clean up variables or other results of the error
  9. Jump to exit sub line label to resume operation (if appropriate)

  - Ex:

    Private Sub txtName_KeyPress(KeyAscii As Integer)

```
'Instruct application what to do upon encountering an error
On Error GoTo frmPersonnel_txtName_KeyPress_Err

   (sub function here)

'End of sub (this code is reached upon successful and unsuccessful
'execution of the procedure
frmPersonnel_txtName_KeyPress_Exit:
   Exit Sub

'Report and handle error here as appropriate
frmPersonnel_txtName_KeyPress_Err:
   Call LogError(Err.Description, Err.Source, Me.Name, "txtName_KeyPress")
   Resume frmPersonnel_txtName_KeyPress_Exit
End Sub
```

Error Reporting

Error reporting is made easier by the err object in Visual Basic. Properties of the err object are: Description, HelpContext, HelpFile, LastDLLError, Number, and Source. Parts, or all, of the information held in the err object may be invaluable in reporting errors in and debugging your application. Toward that end, error reporting:

- ✓ Must output a descriptive and useful error to the user
- ✓ Must contain (when available):
  - Date & Time
  - The form or module name
  - The procedure name
  - A description
  - User information
  - Application version
- ✓ Must export the above information to a text file located in the application folder
- ✓ Where possible, report error information to an appropriate database

Open Issues

The following table provides insight into any unresolved problems or questions. These are the things that seem to apply but could not be fit into this use case on this pass.

Issue ID	Issue Description	Status
	Version Control	Yet to be discussed

References

[1] Microsoft Corporation. Microsoft Office 2000/Visual Basic Programmer's Guide: Constant and Variable Naming Conventions. Retrieved from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon98/html/vbconconstantvariablenamingconventions.asp on 3/26/04.

[2] Microsoft Corporation. Microsoft Visual Basic Programmers Guide: Control Naming Conventions. Retrieved from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon98/html/vbconcontrolnamingconventions.asp on 3/25/04.

[3] Microsoft Corporation. Microsoft Office 2000/Visual Basic Programmer's Guide: Object Naming Conventions. Retrieved from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon98/html/vbconobjectnamingconventions.asp on 3/25/04.

[4] Stiller, E. & LeBlanc C. (2002). Project-Based Software Engineering: An object-oriented approach. Boston, MA: Addison Wesley.

[5] Microsoft Corporation. Microsoft Office 2000/Visual Basic Programmer's Guide: Indentation. Retrieved from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/odeopg/html/deconindentation.asp on 3/28/04.

[6] Microsoft Corporation. Microsoft Office 2000/Visual Basic Programmer's Guide: Line Breaks. Retrieved from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/odeopg/html/deconlinebreaks.asp on 3/28/04.

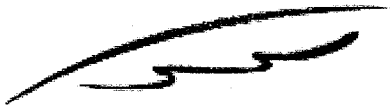
[7] Microsoft Corporation. Microsoft Office 2000/Visual Basic Programmer's Guide: Using Comments Effectively. Retrieved from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/odeopg/html/deconusingcommentseffectively.asp on 3/28/04.

Approval Signatures

Below are the approval signatures for this document.

Date	Version	Name	Comment

Appendix E: GUI Standards – VB Style Guidelines



STOUT
UNIVERSITY OF WISCONSIN

GUI Standards
VB Style Guidelines

Student Life Services – Technical Services
Revised: May 12, 2004

Version History			
Date	**Name**	**Description of Revision**	**Revision #**
3/29/2004	Jane Zielieke	New Document	1.0

Overview

Graphical user interface (GUI) standards improve usability through a clean and consistent look and functionality. Standards are not about personal preference they are about assuring consistent, easy to use applications for our clients. Most users react to cumbersome and inconsistent interfaces with confusion, frustration, and often avoidance. For the users, the interface is the first and only point of contact with the program; because of that, we must strive for a consistent, high-quality look and function to facilitate the users' objectives.

This document covers two basic contributors of good interface design, good human and technical requirements. Human requirements are an often overlooked facet of user interface design. In the end, the most important question becomes "Does this program help them do their jobs efficiently and with satisfaction?" You may often find that meeting the human factor objects are most thoroughly learned through experience; however, this document will contain some guidelines to promote the process. The human requirements section will cover issues such as designing for the user and user assistance. Technical requirements are certainly easier to cover in detail and will follow the human requirements section. Technical requirements will cover: controls, layout, visual effects, and display attributes.

This document contains the current Visual Basic GUI standards for Technical Services. As this document evolves, additions and deletions will be made. Approval signatures and documenting these changes are important for consistency and clarity and are addressed within sections of this document.

To add sections to the table of contents:

8. Follow the instructions for updating the table of contents below
9. Go to the location in text that you would like to add a level item
10. Select the appropriate level
11. Add level item
12. Begin paragraph
13. Set paragraph to "Body text"
14. Begin body text

To update the table of contents:

3. View > Toolbars > Outlining
4. On the new tool bar, click "Update TOC"

Standards

Technical Services, as a member of a Microsoft campus, adopts coding standards similar to those used by Microsoft. These are available at:
http://msdn.microsoft.com

We adopt these standards because:

1. As an industry standard, users will be familiar with the overall look and style of Technical Services applications, keeping us in line with System mandate.
2. As an educational institution, we are committed to providing our employees with an informative work experience applicable post-graduation.

First, a brief word about GUI guidelines and standards… creation of a set of specifications provides a reasonable level of consistency with both the visual and the behavioral aspects of an application. If done correctly, GUI design and implementation can negate many of the potential negative consequences symptomatic of poor design. Poor GUI design can result in difficulties for both the user and yourself as the developer. A user faced with poor design may experience confusion, frustration, and aversion to using the application. As a result, the developers will deal with the fallout: excessive (and preventable!) amounts of time spent on training, needless complexity, and an unsatisfied clientele which may refuse to use the software. The end result is wasted energy by everyone involved. In order to produce a clean, professional application, the phrase "get it right the first time" summarizes the purpose of Technical Services GUI standards document.

Human Requirements

General Guidelines

In order to deliver a productive, user-friendly, interactive interface to our applications, standards have been developed to cover the human requirements for this department. The topic covered in this section is: designing for the user – philosophy and user assistance.

Designing for the User

Philosophy

The primary purpose of a developer at Technical Services is to better help the user accomplish their tasks. If this primary principle is kept in mind, designing for the user will be a relatively easy task. To expand, here are the guidelines for designing for the user:

- ✓ Sit down and talk with users about what they do, how they do it, what they would like to improve on, and what is most important to them.
- ✓ Utilize the use case documents when designing an interface.
- ✓ If given a choice between making your life easier by sacrificing some usability for the user and making it a little more difficult for yourself, choose to put in the work and give them the best interface possible.
- ✓ Expand your knowledge by learning to use new controls and features of Visual Basic.

- ✓ Research GUI design in articles, news groups, and books.

Each of these can dramatically improve how you design a user interface, together the difference is extraordinary.

User Assistance

- *Message boxes*

Informing the users on a high level of the tasks that the application is doing behind the scenes reduces confusion and the likelihood of errors. An informed user can more competently interact with the system if they know what is required of them.

- ✓ Use to convey additional information.
- ✓ Use to inform the user of a desired action.
- ✓ Use to convey errors.
- ✓ Be concise.
- ✓ Be friendly.
- ✓ Use proper grammar and spelling.

  - Ex. – Printing

    When printing a large report, the application will be unavailable for additional input from the users. Informing them of this followed by a completion notice will prevent them from attempting to interrupt, potentially causing a problem within the application and ultimately resulting in frustration.

- *Tool Tip Text*

Often, a little extra effort placed in describing a feature can create an informed, self-sufficient user base. In order to have a concise name for a control, sometimes the name will need to be abbreviated. Adding tool tip text to these will help create a user-friendly, cleaner application.

- ✓ Always use on command buttons.
- ✓ Where available, use on features that need additional detail.
- ✓ Use to further describe purpose.
- ✓ Do not use to restate the obvious.

  - Ex: A command button with "Save" as its text should not have a ToolTipText message "Save". A more appropriate message would be describing what is being saved such as "Save customer information."

- *Automation*

Automation is a useful tool in any developer's kit. The basic principle is if there is a way programmatically to reduce the amount of effort the user has to expend to achieve their goals,

do it. The benefits are tremendous: reduced training time explaining or teaching a new feature, fewer points of failure, and a slick feature that users will most likely appreciate.

- Ex. – Updating the application
  - No automation used – When updating an application from the web, the user will need to shut down the system, type in the URL, download and run the update patch.
  - Automation used – Application notifies the user of an available update, launches the browser with the necessary URL, and shuts itself down. The user is only responsible for download and running the update.

In this example, the application has simplified the user's task by cutting the number of things they need to do from four to two. In addition, the user is no longer required to remember the URL. As you can see, this makes the application easier to use.

Technical Requirements

General Guidelines

In order to deliver a consistent, high-quality interface to our applications standards have been developed to cover the technical requirements for this department. Topics covered in this section are: controls, layout, visual effects, and display attributes.

Controls

Below are a list of the most commonly used controls and guidelines for use. If there is a control not listed here that you would like to use, consult the section titled *Items Not Discussed.*

Command Buttons

- ✓ Use to perform a user requested action.
- ✓ Use an ampersand (&) prior to a letter in the "Text" attribute to enable keyboard shortcuts for that letter.
- ✓ Use tool tips to elaborate purpose.
- ✓ Must be concise.
- ✓ Default property must be set to false.

Labels

- ✓ Use to display information unavailable for user input.
- ✓ Set size slightly in excess of font size, users may have windows large font setting enabled.
- ✓ Must be descriptive.
- ✓ Set backstyle to transparent unless good reason to do otherwise exists.

Text Boxes

- ✓ Must be length appropriate.

- ✓ Use masked textboxes for standardized input for things such as date or time.
- ✓ Prohibit invalid input in KeyPress event.
- ✓ Set multi-line to true if length will exceed horizontal space and extend vertically.

Combo Boxes

- ✓ Must be length appropriate.
- ✓ Prohibit invalid input in KeyPress event.
- ✓ Order appropriately for the task. For example, alphabetically, by date, frequency of use, etc.

Option Buttons

- ✓ Use to indicate an "or" condition.

Check Boxes

- ✓ Use to indicate a potential "and' condition.
- ✓ Never use for a strictly "or" condition.

Frames

- ✓ Use to group visually or functionally (option buttons within a frame operate together).
- ✓ Whenever possible label appropriately.

Menus

- ✓ Grey out unavailable options.
- ✓ Set consistent keyboard shortcuts.

Grids

- ✓ Use consistent behavior throughout an application.
- ✓ Do not use for database input purposes.

Timers

- ✓ Minimize use. Timers use significant system resources.
- ✓ If time to measure exceeds 65,535 milliseconds (a control limitation), use a loop to continue counter.

Scroll Bars

- Vertical
  - ✓ Use in conjunction with picture boxes to conserve vertical space.
  - ✓ Set large and small step appropriate to the size covered.

- Horizontal
  - ✓ Do not use.

Picture Boxes

- ✓ In order to conserve system resources, do not use excessively large picture files.
- ✓ Use in conjunction with vertical scroll bars to conserve vertical space.
- ✓ See guidelines for visual effects for additional rules.

Items Not Discussed

- ✓ Consult industry standards.
- ✓ Consult lead developer for inclusion in this document.

Layout

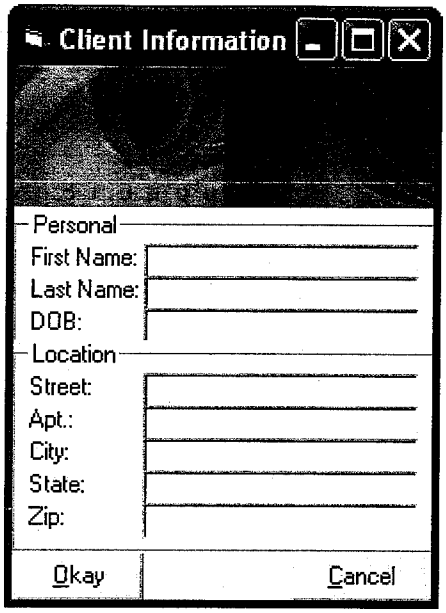
Layout preference is a highly subjective topic matter; therefore, what follows are more generalized guidelines than in the rest of the text. The most important things to consider when designing layout is how well it will function, how well the user will understand it, and yes, how visually appealing it is. In this section, white space and screen flow will be discussed in greater detail.

White Space

White space has both aesthetic and pragmatic uses when designing a user interface. First, aesthetics, if properly used white space can enhance the visual appeal of the program. Improperly used white space can look pinched or sparse. Keep the following in mind when developing a GUI.

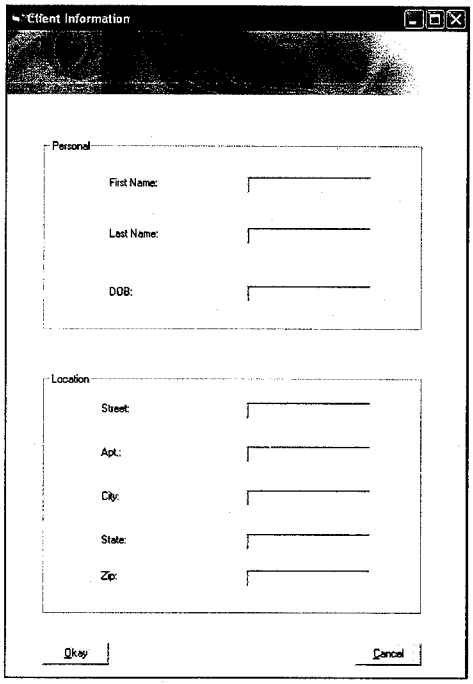
- ✓ Be watchful of crowding.
- ✓ Keep labels and accompanying controls reasonably close to one another.
- ✓ Use more white space between different sections or categories than between related items.
  - o Ex. – Bad Use of White Space #1

    In this example, bad use of white space is demonstrated by the lack of space between labels and text boxes, edges of controls and the edge of the form, and between the command buttons and other elements of the form. Poor use of white space can result in a crowded and pinched look which is displeasing to the eye and, worse yet, make lines indistinct and disorganized making the job of the user more difficult.



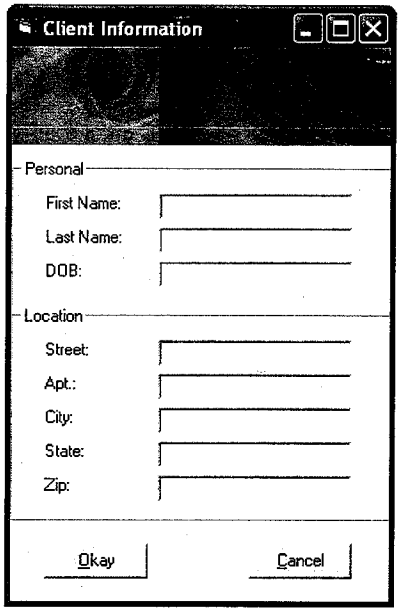
- Ex. – Bad Use of White Space #2

In this example, bad use of white space is demonstrated by overuse of space between labels and text boxes, edges of controls and the edge of the form, and between the command buttons and other elements of the form. Overuse of white space can give the impression of inadequate content and results in a sparse and disconnected look which makes the user work to find connections between labels and accompanying boxes making their job more difficult.



- Ex. – Good Use of White Space

This example demonstrates good use of white space by appropriately using white space as an organizational tool, separating the lines so that the eye is better able to distinguish one line from the next and one category from another, creating a form that is both visually appealing and well organized.



Screen Flow

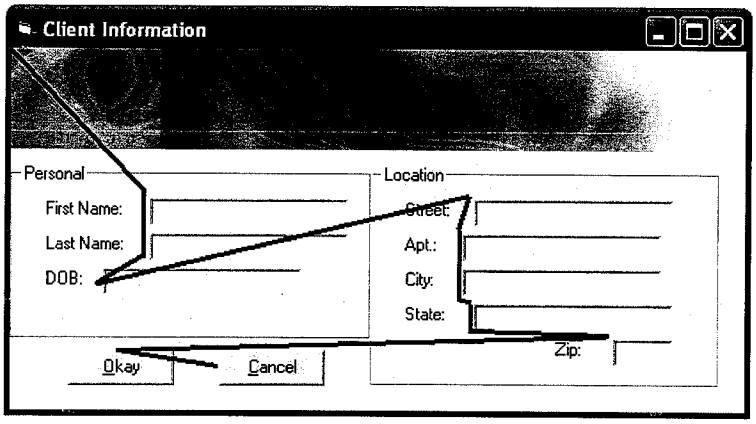
- *Visual*

Visual screen flow directly affects how easy it is to use an application, how quickly it can be learned, and the user's productivity while using it. For that reason, the following are screen flow guidelines:

- ✓ Make visual flow logical and compatible with the user's needs.
- ✓ Line up vertically controls that will be used in sequence.
- ✓ The path the user's eye will follow should contain as few changes of direction as possible.
- ✓ Line up the left edges of labels
- ✓ Size text boxes identically; exceptions may include small fields such as state abbreviations, Zip codes, etc.
- ✓ Line up the left and right edges of text boxes
- ✓ Indent items identically that belonging to a category; example shown below – First name, last name, and date of birth all relate to personal information
- ✓ If information exceeds screen space, use vertical scroll bars or multiple screens to accommodate size.
- ✓ Do not force the user to scroll horizontally on any form, this is often disruptive and cannot be accomplished by a mouse wheel making it difficult to use.

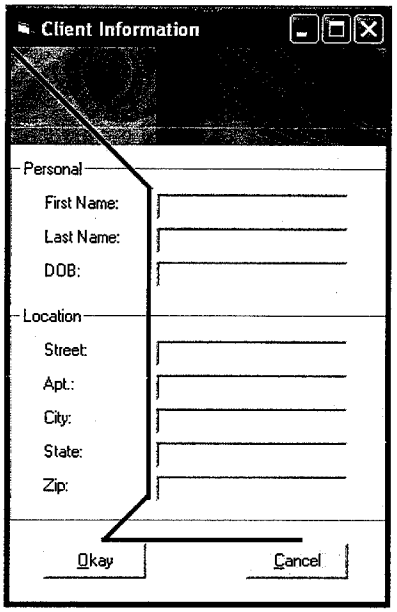
- Ex. – Bad Visual Screen Flow

In this example, bad screen flow is demonstrated by the jagged line the eye needs to follow in order to input or read information off of the screen, left and right edges do not line up.



- Ex. – Good Visual Screen Flow

This example demonstrates good screen flow by minimizing eye flow direction, uniform text field widths, and uniformly indented fields within a category. If more information were to be included, vertical scrolling or organizational controls such as Microsoft's tab control may be used.



- *TabStops*

TabStops provide a valuable component to functional flow of a form. It sets the order of manipulation within the form. If the functional flow is out of synch with the visual flow, the usability of the application suffers. Use the following rules when applying TabStops:

- ✓ Always use TabStops to control data entry and form manipulation.

- ✓ Make functional flow logical and compatible with the user's needs.
- ✓ Match functional flow (TabStops) to visual flow (eye flow).
- ✓ Check for unnecessary TabStops, for instance, non-user responsive controls such as frames, by running the program, disable when found.

Visual Effects

Blinking

Blinking of controls, text, or other items has tremendous attention getting power, so much so that often it is distracting or, worse yet, annoying. Guidelines for use:

- ✓ As a general principle, do not use blinking as an attention getting feature.
- ✓ Blinking may only be used as a blinking system tray icon when all other forms are closed.

Fading

Fading, such as between forms, consumes system resources. Some of the systems under Technical Resources may prove inadequate to the task and slow down to an unacceptable performance level.

- ✓ Do not use fading.
- ✓ Utilize another technique, such as indicating a "wait" status by calling for an hourglass icon and displaying a splash screen asking for the user's patience while the application retrieves the requested information.

Animation

Animation is not only unprofessional in an educational setting; it consumes an unacceptable amount of resources on an already taxed system.

- ✓ Do not use animation.

Display Attributes

Color

Depending on use, color can either be an enhancement or a problem. Technical Services uses the following guidelines regarding color:

- ✓ Use VB defined system colors wherever possible, this accounts for different desktop themes. System colors are found by accessing a color property's drop down box and selecting the "System" tab.
- ✓ Take into account the possibility color blindness; therefore color should only be an accent, not a functional requirement.

✓ Use no more than four colors at a time on a screen [1].

Text

Text is perhaps the most important method for conveying information to users. Color, clarity, size all affect the legibility and interpretation gotten from an application. General guidelines serve to assist the user by providing a friendly, consistent, and easy to understand tool.

- *Fonts*

  The most critical characteristic when choosing text is the font. Fonts have inherent, and subjective, interpretations the moment someone looks at them. For example, free flowing Script MT Bold has a casual, hand-drawn feel. However, as a developer at Technical Services, fonts should be chosen with the following in mind:

  ✓ Fonts, ideally, should go unnoticed by the user. Distracting or unusual fonts detract from the application and should not be used.

  ✓ Font must be clear and legible.

  ✓ Choose a font with 'professional' connotations such as: Times New Roman, Arial, etc.

  ✓ Do not use graphical fonts such as Wingdings, ZaphDingbats, WebDings, etc.

- *Size*

  Many of the same guidelines that apply to fonts carry over to size. Below are some guidelines when choosing font size:

  ✓ Use a standard 12 point font wherever possible.

  ✓ Do not exceed 14 point size. Font size can be used to add emphasis or attract attention. However, excessive use is both distracting and unprofessional.

  ✓ Do not use less than 8 point size. Keep in mind that visual acuity varies between users and they may not be able to read small type.

- *Dimming/Graying*

  Dimming or graying texts via either their font color or through disabling the control is a powerful indicator to users about the availability of that item. Guidelines for dimming and graying text and controls are as follows:

  ✓ Use only when making the item unavailable.

- *Emphasis*

  Text emphasis can be accomplished in various ways such as underlining, increasing or decreasing size, bolding, indentation, etc. In excess text emphasis, either individually or in conjunction with another technique, can create a jumbled and difficult to use interface. Guidelines for textual emphasis are as follows:

  ✓ Be watchful for overuse of emphasis. Emphasis on the majority of lines decreases its effectiveness and creates a "loud" and confused look.

- ✓ Only use emphasis on particularly important information or headings.
- ✓ Be particularly aware of legibility.

Open Issues

The following table provides insight into any unresolved problems or questions. These are the things that seem to apply but could not be fit into this document on this version.

Issue ID	Issue Description	Status
1.	Icons	yet to be discussed
2.		
3.		
4.		
5.		

References

[1] Galitz. W. (1997). The Essential Guild to User Interface Design. New York: Wiley Computer Publishing.

Approval Signatures

Below are the approval signatures for this document.

Date	Version	Name	Comment

Appendix F: VB Data Storage & Retrieval – VB Style Guidelines

STOUT
UNIVERSITY OF WISCONSIN

Style Guide: Data Storage & Retrieval

Student Life Services - Technical Services
Revised: May 12, 2004

Version History			
Date	**Name**	**Description of Revision**	**Revision #**
4/5/2004	Jane Zielieke	New Document	1.0

Overview

This module was designed to give the first-time Visual Basic programmer experience with the procedures and syntax of data storage and retrievals most often encountered at Technical Services. This module comes in two sections: a basic instructional segment detailing syntax and approaches followed by an exercise section designed to test the programmer's knowledge. Later, when the programmer feels confident, the instructional segment may be used as a quick reference guide, skipping the exercise(s) that follow.

Principles

This section will provide you with basic syntax for working with basic data types you will encounter. It is not intended as a comprehensive query manual. It will give you the most basic start in building database applications in Visual Basic. Three types of data storage are discussed in this section: Access, SQL, and text files. For other types, consult the software development supervisor for possible inclusion in this document.

General

Data Validation

When working with databases, Technical Services uses the following validation guidelines:

- ✓ Validate input information in code rather than relying on the database.
- ✓ Data validation functions should be contained within a module (.bas).

Data Types

- ✓ Plan ahead.
  - o Ex:

  Somewhere down the road, the users may wish to sort phone numbers by area code. If the phone "number" was stored as a string, sorting would be more difficult and resource intensive.
- ✓ Utilize correct data types.
  - ▪ Do not use strings to store numerical values.
  - ▪ Do not store Boolean as strings.
  - ▪ Etc.
- ✓ Keep space in mind. In fields permitting variable lengths, determine the absolute maximum and size accordingly. Field size can always be enlarged.
- ✓ Keep speed in mind. For instance, numbers sort more quickly and with less effort than strings. If application design permits, utilize the lowest cost data type.

Queries

- ✓ Avoid "Select *". Grab only the fields needed.
- ✓ Utilize stored procedures in SQL Server whenever possible. This reduces the resources required to process the request. This method is preferred for frequently used queries.

Database Connections

- ✓ Close data connections whenever they are not in use.
- ✓ Set connection variables to "Nothing" after use.

- ✓ Store database connection location/information in an external text file for added flexibility.
- ✓ When creating a new SQL security login or password protecting a database, notify project lead of program, purpose, location, and password.

Access Database

- ✓ Use Access versions 2000 or later.
- ✓ Add reference "Microsoft DAO 3.6 Object Library."

Queries

Insert

- String
  - ➢ Direct:

    The query should be in this format:
    <Query name> = "Insert into <table name>(<Field name 1>, <Field name 2>, …, <Field name n>) Values('<Value 1>', '<Value 2>', …, '<Value n>')"

    Note: Double quotes before and after query.
    Singular apostrophe before and after values.

    - o Ex:

      strQuery = "Insert into Building(Name) Values('Antrim Hall')"
      db.Execute strQuery

  - ➢ In a variable:

    The query should be in this format:
    <Query name> = "Insert into <table name>(<Field name 1>, <Field name 2>, …, <Field name n>) Values('" & <Value 1> & "', '" & <Value 2> & "', …, '" & <Value n> & "')"

    Note: Double quotes before and after query.
    Singular apostrophe, double quote, space, ampersand, and space before variable.
    Space, ampersand, space, double quote, and singular apostrophe after variable.

    - o Ex:

      strQuery = "Insert into Building(Name) Values('" & strBuilding & "')"
      db.Execute strQuery

- Number

- Direct:

  The query should be in this format:
  <Query name> = "Insert into <table name>(<Field name 1>, <Field name 2>, …, <Field name n>) Values(<Value 1>, <Value 2>, …, <Value n>)"

  Note: Double quotes before and after query.

  - Ex:

    strQuery = "Insert into ZipCode(Number) Values(54751)"
    db.Execute strQuery

- In a variable:

  The query should be in this format:
  <Query name> = "Insert into <table name>(<Field name 1>, <Field name 2>, …, <Field name n>) Values(" & <Value 1> & ", " & <Value 2> & ", …, " & <Value n> & ")"

  Note: Double quotes before and after query.
  Double quote, space, ampersand, and space before variable.
  Space, ampersand, space, and double quote after variable.

  - Ex:

    strQuery = "Insert into ZipCode(Number) Values(" & intZip & ")"
    db.Execute strQuery

- Date

  - Direct:

    The query should be in this format:
    <Query name> = "Insert into <table name>(<Field name 1>, <Field name 2>, …, <Field name n>) Values(#<Value 1>#, #<Value 2>#, …, #<Value n>#)"

    Note: Double quotes before and after query.
    Pound symbol before and after values.

    - Ex:

      strQuery = "Insert into TimePunch(LogDate, LogTime)
      Values(#12/12/04#, #12:00:00 PM#)"
      db.Execute strQuery

  - In a variable:

    The query should be in this format:

<Query name> = "Insert into <table name>(<Field name 1>, <Field name 2>, …, <Field name n>) Values(#" & <Value 1> & "#, #" & <Value 2> & "#, …, #" & <Value n> & "#)"

Note: Double quotes before and after query.
Pound symbol, double quote, space, ampersand, and space before variable.
Space, ampersand, space, double quote, and pound symbol after variable.

- Ex:

strQuery = "Insert into TimePunch(LogDate, LogTime) Values(#" & strDate & "#, #" & strTime & "#)"
db.Execute strQuery

Update

- String
  - Direct:

The query should be in this format:
<Query name> = "Update <table name> set <Field name 1> = '<Value 1>', <Field name 2> = '<Value 2>',…, <Field name n> = '<Value n>' where <where clause>"

Note: Double quotes before and after query.
Singular apostrophe before and after values.

  - In a variable:

The query should be in this format:
- Ex:

<Query name> = "Update <table name> set <Field name 1> = '" & <Value 1> & "', <Field name 2> = '" & <Value 2> & "',…, <Field name n> = '" & <Value n> & "' where <where clause>"

Note: Double quotes before and after query.
Singular apostrophe, double quote, space, ampersand, and space before variable.
Space, ampersand, space, double quote, and singular apostrophe after variable.

- Ex:

strQuery = "Update Building set Name = '" & strBuilding & "' where ID = 4"
db.Execute strQuery

- Number
  - Direct:

    The query should be in this format:
    <Query name> = "Update <table name> set <Field name 1> = <Value 1>, <Field name 2> = <Value 2>,..., <Field name n> = <Value n> where <where clause>"

    Note: Double quotes before and after query.

    - Ex:

      strQuery = "Update ZipCode set Number = 54751 where TownID = 1"
      db.Execute strQuery

  - In a variable:

    The query should be in this format:
    <Query name> = "Update <table name> set <Field name 1> = " & <Value 1> & ", <Field name 2> = " & <Value 2> & ",..., <Field name n> = " & <Value n> & " where <where clause>"

    Note: Double quotes before and after query.
    Double quote, space, ampersand, and space before variable.
    Space, ampersand, space, and double quote after variable.

    - Ex:

      strQuery = "Update ZipCode set Number = " & intZip & " where TownID = " & TownID & ""
      db.Execute strQuery

- Date
  - Direct:

    The query should be in this format:
    <Query name> = "Update <table name> set <Field name 1> = #<Value 1>#, <Field name 2> = #<Value 2>#,..., <Field name n> = #<Value n># where <where clause>"

    Note: Double quotes before and after query.
    Pound symbol before and after values.

    - Ex:

      strQuery = "Update TimePunch set LogDate = #12/12/04#, LogTime = #12:00:00 PM# where UserID = 4"

db.Execute strQuery

- In a variable:

  The query should be in this format:
  <Query name> = "Update <table name> set <Field name 1> = #" & <Value 1> & "#, <Field name 2> = #" & <Value 2> & "#, …, <Field name n> = #" & <Value n> & "# where <where clause>"

  Note: Double quotes before and after query.
  Pound symbol, double quote, space, ampersand, and space before variable.
  Space, ampersand, space, double quote, and pound symbol after variable.

  - Ex:

    strQuery = "Update TimePunch set LogDate = #" & strDate & "#, LogTime = #" & strTime & "# where <where clause>"
    db.Execute strQuery

Delete

The query should be in this format:
strQuery = "Delete from <table name> where <where clause>"

- Ex:

  strQuery = "Delete from Building where Name = 'JTC'"

Note: Access accepts the "Delete * from" convention but for consistency, use the "Delete from" syntax.

Warning! Test your queries on a backup first! You don't want to damage/lose live data.

Other

The examples shown here pertain to data type manipulations vs. instruction in various types of queries. For further information on different types of queries consult the following sources: text references, news groups, professional development websites, MSDN, etc.

Recordsets

- ✓ Open as briefly as possible
- ✓ Close and set to nothing when you're done using it.
  - Ex:

    'Declare objects
    Dim dbClient As Database
    Dim rsClient As Recordset

```
Dim ws As Workspace

'Set database
Set db = ws.OpenDatabase("C:\Biblio.mdb")
'Set query
strQuery = "Select ClientName, Address from Clients where ID = 4"

'open recordset cursor
Set rsClient = dbClient.OpenRecordset(strQuery)

'close
rsClient.close
Set rsClient = Nothing
db.Close
Set db = Nothing
```

SQL Database

- ✓ Add reference "Microsoft ActiveX Data Objects 2.5 Library."
- ✓ Utilize a .dsn file as your method of connection.
  - o Ex:

    The .dsn file should contain a single line similar to the following:
    Provider=SQLOLEDB;Data Source=<ServerName>;
    uid=<SecurityLoginName>;pwd=<password>;
    Catalog=<database name>;

Queries

Insert

- String
  - ➢ See Access syntax for updating a date field in SQL
- Number
  - ➢ See Access syntax for updating a date field in SQL
- Date
  - ➢ Direct:

    The query should be in this format:
    <Query name> = "Insert into <table name>(<Field name 1>, <Field name 2>, …, <Field name n>) Values('<Value 1>', '<Value 2>', …, '<Value n>')"

    Note: Double quotes before and after query.

Singular apostrophe before and after values.

- Ex:

```
strQuery = "Insert into TimePunch(LogDate, LogTime)
Values('12/12/04', '12:00:00 PM')"
db.Execute strQuery
```

➢ In a variable:

The query should be in this format:
<Query name> = "Insert into <table name>(<Field name 1>, <Field name 2>, …, <Field name n>) Values('" & <Value 1> & "', '" & <Value 2> & "', …, '" & <Value n> & "')"

Note: Double quotes before and after query.

Unlike Access, SQL does not use the pound symbol for date; therefore:
Single apostrophe, double quote, space, ampersand, and space before variable.
Space, ampersand, space, double quote, and single apostrophe after variable.

- Ex:

```
strQuery = "Insert into TimePunch(LogDate, LogTime) Values('" &
strDate & "', '" & strTime & "')"
db.Execute strQuery
```

Update

- String
  - ➢ See Access syntax for updating a date field in SQL
- Number
  - ➢ See Access syntax for updating a date field in SQL
- Date
  - ➢ Direct:

    The query should be in this format:
    <Query name> = "Update <table name> set <Field name 1> = '<Value 1>', <Field name 2> = '<Value 2>',…, <Field name n> = '<Value n>' where <where clause>"

Note: Double quotes before and after query.
Singular apostrophe before and after values.

- Ex:

```
strQuery = "Update TimePunch set LogDate = '12/12/04', LogTime =
'12:00:00 PM' where UserID = 4"
db.Execute strQuery
```

➢ In a variable:

The query should be in this format:
<Query name> = "Update <table name> set <Field name 1> = '" & <Value 1> & "', <Field name 2> = '" & <Value 2> & "',..., <Field name n> = '" & <Value n> & "' where <where clause>"

Note: Double quotes before and after query.

Unlike Access, SQL does not use the pound symbol for date; therefore:
Single apostrophe, double quote, space, ampersand, and space before variable.
Space, ampersand, space, double quote, and single apostrophe after variable.

- Ex:

```
strQuery = "Update TimePunch set LogDate = #" & strDate & "#,
LogTime = #" & strTime & "# where <where clause>"
db.Execute strQuery
```

Delete

The query should be in this format:
strQuery = "Delete from <table name> where <where clause>"

- Ex:

```
strQuery = "Delete from Building where Name = 'JTC'"
db.execute strQuery
```

Note: SQL does NOT accept the "Delete * from" convention; use the "Delete from" syntax.

Warning! Test your queries on a backup first! You don't want to damage/lose live data.

Other

The examples shown here pertain to data type manipulations vs. instruction in various types of queries. For further information on different types of queries consult the

following sources: text references, news groups, professional development websites, MSDN, etc.

Recordsets

- ✓ Open as briefly as possible
- ✓ Limit lock type strictly. For instance, if all you're doing is retrieving information, lock to read only.
- ✓ Close and set to nothing when you're done using it.
  - o Ex:

```
'Declare objects
Dim dbClient As ADODB.Connection
Dim rsClient As ADODB.Recordset

'Set query
strQuery = "Select ClientName, Address from Clients where ID = 4"

'Instantiate recordset
Set rsClient = New ADODB.Recordset

'open recordset cursor
'format - <recordset object>.open <query>, <active connection>, <cursor
type>, <lock type>, <options>
'only source and active connection are required, cursor type and lock type are
'recommended
rsClient.Open strQuery, dbClient, adOpenStatic, adLockReadOnly
rsClient.Close
set rsClient = Nothing
db.Close
set db = Nothing
```

Text File

- Input

```
Open <File path and name> For Input As #1
Print #1, <Input>
Close #1
```

  - o Ex:

```
Open App.Path & "\Defaults.txt" For Input As #1
Print #1, "Theme=0"
Close #1
```

- Append

Open <File path and name> For Append As #1
Print #1, <Input>
Close #1

- Ex:

```
Open App.Path & "\ErrorLog.txt" For Append As #1
Print #1, "Error – File not found"
Close #1
```

Approval Signatures

Date	Version	Name	Comment

Appendix G: Requirements Document

STOUT
UNIVERSITY OF WISCONSIN

Requirements Document:
<Project Name>

Student Life Services - Technical Services
Revised: May 12, 2004

Version History			
Date	**Name**	**Description of Revision**	**Revision #**
4/15/2004	Jane Zielieke	New Document	1.0

Project Overview

Describe the background and context for the project and why it is being undertaken. Speak to the business value of the work being performed. Put enough information here so that the rest of the sections in the project definition make sense. (Remove this comment section from final document.)

Functional Requirements

Objectives are statements that describe what this project will achieve and deliver. Objectives should be "SMART": Specific, Measurable, Achievable, Realistic, and Time-Based. To be specific and concrete, objectives should be deliverable-based. The priority of requirements will roughly be in descending order. The completion of an objective should be evident through the creation of one or more deliverables. If the statement is at a high level and does not imply the creation of a deliverable, it may be a goal instead. If the statement is too low-level and describes features and functions, then it may be a requirement statement instead. (Remove this comment section from final document.)

The XXX project will meet the following objectives:

Deliverables

1. **<Name of highest ranked requirement>**
   1.1. **Description**

   A full description of the requirement.

   1.2. **Criticality**

   Describes how essential this requirement is to the overall system.

   1.3. **Technical issues**

   Describes any design or implementation issues involved in satisfying this requirement.

   1.4. **Cost and schedule**

   Describes the relative or absolute costs associated with this issue.

   1.5. **Risks**

   Describes the circumstances under which this requirement might not able to be satisfied, and what actions can be taken to reduce the probability of this occurrence.

1.6. **Dependencies**

Describes interactions with other requirements.

1.7. **... others as appropriate**

2. **<Name of next highest ranked requirement>**

Project Scope

In this section, you should clearly define the logical boundaries of your project. Scope statements are used to define what is within the boundaries of the project and what is outside those boundaries. Examples of areas that could be examined are data, processes, applications, or business areas. The following types of information can be helpful:

- *The types of deliverables that are in scope and out of scope (Business Requirements, Current State Assessment)*
- *The major life-cycle processes that are in scope and out of scope (analysis, design, testing)*
- *The types of data that are in scope and out of scope (financial, sales, employee)*
- *The data sources (or databases) that are in scope and out of scope (Billing, General Ledger, Payroll)*
- *The organizations that are in scope and out of scope (Human Resources, Manufacturing, vendors)*
- *The major functionality that is in scope and out of scope (decision support, data entry, management reporting)*

(Remove this comment section from final document.)

The scope of this project includes and excludes the following items.

In scope:

-
-
-
-

Out of scope:

-
-
-
-

Organizations Affected

Specify areas or groups affected by, or that may participate in, the project. This is meant to be comprehensive but high level. Individual names should not appear, but the organizations they represent are included here. (Remove this comment section from final document.)

Organization	How are they affected, or how are they participating?

System Requirements

System requirements are statements that describe what this project will require. Requirements should be stated so as to be specific and measurable.

Hardware

Hardware requirements include items such as disk drives, memory, CPU, storage, etc.

Required:

Desired:

Operating System

Operating system requirements include information such as Windows, Linux, UNIX, etc. Version information must be included.

Required:

Desired:

Support Software

Support Software includes any software dependencies necessary for the application to meet functional requirements. Examples include Microsoft's Office Suite, database support, web server, etc. Version information must be included.

Required:

Desired:

Other

Required:

Desired:

Design Constraints

Standards Compliance

Development Language Requirements

Non Functional Attributes

Security

Portability

Compatibility

Others

Project Responsibilities

Client Director:
Name and contact information for the primary stakeholder for this project.

Primary Client Contact:
Name and contact information for the primary client-side contact person. The software development team will contact this individual first regarding questions, scheduling, and other pertinent issues.

Project Department Director:
Name and contact information of the director of software development department.

Project Manager:
Name and contact information for the software development supervisor.

Developers:
Names and contact information for the developers.

Primary Project Contact:
Name and contact information for the primary project-side contact person. The client will contact this individual first regarding questions, scheduling, and other pertinent issues.

Project assumptions

Project assumptions are circumstances and events that need to occur for the project to be successful but are outside the total control of the project team. They are listed as assumptions if there is a HIGH probability that they will in fact happen. The assumptions provide a historical perspective when evaluating project performance and determining justification for project-related decisions and direction. (Remove this comment section from final document.)

In order to identify and estimate the required tasks and timing for the project, certain assumptions and premises need to be made. Based on the current knowledge today, the project assumptions are listed below. If an assumption is invalidated at a later date, then the activities and estimates in the project plan should be adjusted accordingly.

- Assumption #1
- Assumption #2
- Assumption #3, etc.

Project risks

Project risks are circumstances or events that exist outside of the control of the project team that will have an adverse impact on the project if they occur. (In other words, whereas an issue is a current problem that must be dealt with, a risk is a potential future problem that has not yet occurred.) All projects contain some risks. It may not be possible to eliminate risks entirely, but they can be anticipated and managed, thereby reducing the probability that they will occur.

> Risks that have a high probability of occurring and have a high negative impact should be listed below. Also consider those risks that have a medium probability of occurring. For each risk listed, identify activities to perform to eliminate or mitigate the risk.

Project risks are characteristics, circumstances, or features of the project environment that may have an adverse affect on the project or the quality of its deliverables. Known risks identified with this project have been included below. A plan will be put into place to minimize or eliminate the impact of each risk to the project.

Risk Area	Level (H/M/L)	Risk Plan
1. Project risk #1		Risk plan activity #1 Risk plan activity #2, etc.
2. Project risk #2		
3. Project risk #3		

Project approach

This section is used to describe how the project will be structured and the important techniques that will be utilized. The project approach is intended to encourage the project manager to think about the project from the top down instead of the traditional bottom-up method. Including the approach in the project definition compels the project manager to both consider the dependencies of the project and to incorporate the project management necessary to plan and manage the project. (Remove this comment section from final document.)

Project estimated effort/cost/duration

The estimated effort hours and project costs may be depicted in many ways, including cost by team member, cost by deliverable, cost by milestone, or cost by category (internal labor, external labor, travel, training, supplies, etc.). Also include a chart showing the project start date, major milestones, and end date. The deliverables included in this milestone chart should all have been described in the requirements section. (Remove this comment section from final document.)

Estimated cost:

Estimated effort hours:

Estimated duration:

Milestone	Date completed	Deliverable(s) completed
Project planning	Mm/dd/yy	Project definition Workplan
Milestone 1	Mm/dd/yy	Deliverable 1 Deliverable 2
Milestone 2	Mm/dd/yy	Deliverable 3
Milestone 3	Mm/dd/yy	Deliverable 4
Milestone 4	Mm/dd/yy	Deliverable 5
Project conclusion	Mm/dd/yy	

Project approvals:

Add any signatures that are important for the approval of the project. (Remove this comment section from final document.)

Customer Project Manager–xxxxx xxxx	Date
Project Director–xxxx xxxx	Date
Project Manager–xxxxx xxxx	Date

Appendix H: Use Case Document

Use Case Worksheet

Student Life Services - Technical Services

Title	*[3-4 word verb phase]*
Version	
Date	*[Date of this version]*
Author	*[Name and contact details of the person who wrote the Use Case]*
Summary	This use case begins when …This use case does …This use case concludes (or ends) when …
Assumptions / Constraints	
Primary Actor	*[Roles of people and/or systems that will use this Use case]*
Secondary Actors	*[Roles of other people and/or systems that will use this Use case]*
Outstanding Issues	
Pre-condition(s)	*[The necessary conditions that have to be met before the use case can be performed]*

Typical Sequence of Events

User Stimulus	Interface	System Response
1.		2.
3.		4.
5.		6.
7.		8.
9.		10.
11.		12.
13.		14.

Post-condition(s)	*[The state of the system after the use case has successfully completed]*

Appendix I: Testing Document

Testing Document:
<Project Name>

Student Life Services – Technical Services
Revised: May 12, 2004

Testing History

Date	Programmer	Reviewer	Changes Needed?

Unit Testing

Module 1

Description	Input	Expected Test Results	Actual Test Results	Pass or Fail
1.				
2.				
3.				

Module 2

Description	Input	Expected Test Results	Actual Test Results	Pass or Fail
1.				
2.				
3.				

Module 3

Description	Input	Expected Test Results	Actual Test Results	Pass or Fail
1.				
2.				
3.				

Module 4

Description	Input	Expected Test Results	Actual Test Results	Pass or Fail
1.				
2.				
3.				

Integrated Testing

Workflow 1

Description	Input	Expected Test Results	Actual Test Results	Pass or Fail
1.				
2.				
3.				

Workflow 2

Description	Input	Expected Test Results	Actual Test Results	Pass or Fail
1.				
2.				
3.				