

Application & Database Design Issues White Paper

Published: April 1999 (with revisions)

Application System Design and Development Using a Systematic Approach

Contents

Preface	1
Overview	2
Project Management	2
Service Level Agreements	3
Business Rules	3
Documentation	3
Database Design	4
Application Design and Development	4
User Interface	5
Security	5
Open Systems & Open Source	6
Popular Products	6
Training	6
Development Best Practices	7
Quality Management	8
System Configuration	9
Summary	10
About the Author	10
Let Us Help You Succeed!	10

Preface

Only 28% of application system projects are completely successful (source: The Standish Group “1998 Chaos Report”) – meaning that the project is completed on time and on budget, with all features and functions as originally specified. While updated versions of that report show some improvement, there is still significant room for improvement. This white paper describes, at a high level, some of the many areas to consider regarding the design and development of application systems in order to improve your chances for success.

Overview

Application systems are generally best when they are designed and developed using a systematic approach. **Seldom does a system *evolve* into something great over time without standardized processes in place.** It is critical to understand the goals, requirements, and expectations of the system before development starts. Everyone needs to understand “what success looks like” before the first specification is created or the first line of source code is written. Then it is just as important to plan for success, track the progress, proactively identify and eliminate obstacles, and then plan to test and roll-out the new system to production.

These steps are even true when using XP (Extreme Programming, something that has fallen out of favor) or “Agile” programming techniques. Many people believe that a discipline like Agile programming gives them license to do everything “on the fly” (“that is what refactoring is for, isn’t it?”). That is not true. Goals and scope are important when identifying releases. The primary benefit that we have seen from this methodology, when properly implemented, is the team practices (such as pair programming, collaboration, and the use of a “Tracker” to track progress on a daily or weekly schedule) benefit the project. The problem is that it can be difficult to be disciplined enough to do this properly.

Below are issues that are germane to any development project. This is not an all-inclusive list by any means, but it does address several critical areas for the majority of projects.

By addressing these issues in the initial stages of a project the likelihood of delivering a system that works as needed, on time and within budget is greatly increased. If you are unfamiliar with any of these subject areas then it is best to seek assistance from a consulting company with proven expertise and success.

Project Management

Is there a common vision of success? Is the scope of the project aligned with business goals? Is there a project plan? Are there defined tasks with assigned roles and responsibilities? What are the acceptance criteria (i.e., how do we know when we are *really* done)? Do we know what the available resources are, when they are available, for how long and what portion of their time? Are specific tasks mapped out with specific deliverables? It is important to identify problems as early in the process as possible, and that is very difficult if there are not “hard targets.” The value of project management cannot be overstated.

If needed, is there high-level Management support for this effort? If we require additional resources or talent, will funding be available? Has a risk analysis been performed and is there a risk management plan? Is there a communication plan? There are many aspects to project management that, if performed properly, will add significant value to systems design and development efforts.

For more information on Project Management please see our various [white papers](#) on the topic.

Service Level Agreements

It is important to determine what level of service the user community is not only expecting, but actually needs in order to be fully satisfied with the system or application, now and in the future. Service Level Agreements (SLAs) address system availability (e.g., Monday through Saturday, 5:00 a.m. - 10:00 p.m. CST), acceptable scheduled down time for maintenance, maximum acceptable unscheduled down time (usually due to hardware or software problems), and system response time (average and maximum, e.g., 90% of all online transactions will complete within 2 seconds, 10% of all online transactions can take up to 4 seconds to complete). This knowledge will be especially useful when configuring the system and when designing applications. It is much easier to design a high performance application than it is to modify a system to improve performance.

Business Rules

This is critical to the overall success in a large, integrated environment. These map the fundamental requirements and relationships in business. For example, a business rule might be "a cashier can only access a cash register while they are working." The list of business rules can be quite extensive, intricate and often not well defined. It is very important to document these rules, their interactions, and dependencies.

Exceptions to the rules should also be noted. Exception processing can often be very difficult and therefore exceptions should be minimized as much as possible. The list can then be reviewed to determine where the rules will be implemented (in the application or server). It is important to understand both the process and procedures involved with exception handling, and then make sure that the new application system is flexible enough to accommodate them.

Documentation

Never underestimate the importance of well-designed documentation in the development of an application or system – technical internal IT design and maintenance documentation as well as external end-user documentation. Documentation is the common thread that makes a system tangible, and provides a consistent perspective of what is being done versus what was requested.

Application system projects are much more likely to fail (i.e., not meet business expectations) from end-user adoption issues than from a poor implementation. A successful end-user rollout, and new-user adoption throughout the life of an application can only happen with properly designed and thought-out documentation. Documentation design considerations include overall content, optimum delivery mediums, initial introduction and training use, routine use, and helping users deal with problems and exceptions. Remember that end-users are not only the applications users but also system administrators and operations staff that need to schedule, operate, monitor, back-up, etc. – in any way support the application system and data. This information should be widely available to people within your company and reviewed on a regular basis.

Accurate, comprehensive internal documentation will not only prove to be of great value during the development process, but also during ongoing support and maintenance. Documentation should include at a minimum the system requirements, functional specifications, business rules, a data element dictionary (that defines common data elements and provides domain information), Entity-Relationship (ER) diagrams, and Data Flow Diagrams (DFD). Object Oriented environments should describe the various business objects and their methods, as well as Object / Relational Mapping when using a relational database. The value of this documentation increases with the complexity of an environment. Documentation is as important and valuable as the software it describes!

If your company is considering outsourcing the development, maintenance, or support of application systems, the value of documentation increases even further. It is easy to make assumptions if you “know everything” about a system. In an outsourcing scenario this context is often absent. Therefore, documentation at all levels needs to be complete, provide specific examples, and be as unambiguous as possible.

Database Design

This is critical to the overall success of any large project. Design problems can limit the overall flexibility of the system, and may cause excessive table joins and/or coding work-arounds that impact performance. We recommend using a three schema approach to DB design: Business schema (ensures that all business requirements are defined); Logical schema (the normalized business schema); and Physical schema (storage structure selection for performance, controlled denormalization, security).

A design goal should always be consistent access time. This may require summary tables (such as "fact" tables in a data warehouse) to help facilitate that. The applications should contain functionality to monitor and track access times. Nothing frustrates end-users more than inconsistent access time.

Many Clients prefer to use a RAD or XP approach with new system development. While this may be great for application system design, but is not a good idea for database design. It is generally worth the extra effort to carefully and properly design the database. Even if the database is already designed it is good practice to have that design independently reviewed by a design expert.

Application Design and Development

Performance and tracability are two critical aspects of system / application design that are often overlooked. It is much easier to design performance into a system than to try to "back" it in later. Critical information as the service level agreements and anticipated user load should be used to create a high performance, highly concurrent application. This includes good transaction and query design, efficient data access and manipulation, and proper error detection / retry / restart logic. This can also improve the overall responsiveness and performance consistency that is so important in a system. Please see our [white paper on transaction management and restart logic](#) for more information.

It is also important to be able to gather performance metrics and debug / trace information. This can be used to validate performance on an ongoing basis as well as to isolate problem areas of code when an issue arises. This is especially helpful when it can be started using non-intrusive means such as an environment variable or command line option. It can also help by providing the necessary information for performance profiling and trend analysis.

Having a good foundation for an application will have an immense payback during the life of that system and is therefore a very good investment in time, effort and money. As with database design, consistent performance is a worthy and achievable goal.

User Interface

What type of user interface is being used? More and more frequently it is a web browser. Is this a graphical (GUI) or character based (CHUI) interface? In a web-based environment this could mean the difference between using HTML (which can be operated in a pseudo-GUI manner) and Java (which is a true GUI tool, but could be made to handle a CHUI application if necessary). Once the type of interface is selected it is important to select the best tool or environment for developing that interface (rather than picking a tool and then deciding on the type of interface).

When developing browser-based systems it is important to contrast the functionality and technology used with the anticipated browsers and platforms. Different browsers can render a screen in many different ways, and therefore testing must be done not only with several browsers and versions, but also on several platforms. Designing an interface that only works with a single web browser (or worse yet, a single version of that web browser) is not a good practice.

Older browsers often cause problems, but it may not be possible to specify that an end user use a specific browser or version of that browser. Furthermore, certain technologies (java, java script, active server pages, cascading style sheets) have the potential to restrict the overall functionality of the application when they don't execute properly in a given browser. NOTE: While this is not as much of a problem anymore, it can still present numerous problems when applications are intended to perform in a non-controlled environment (such as the Internet).

Security

Will the system interface with existing systems? What about external systems? What are the requirements to safeguard the data? Authentication and identification are critical for most web applications. This needs to be implemented in a way that is secure, scales well, does not place too much of a load on the system, and is easy to use (smart cards are ideal, but not widely used). Use of relative paths within protected directories is a good and simple approach to security once the session has been authenticated and the end user identified.

It is very important to have an understanding of what the potential threats are, and what the impact of a security breach or exploit is. What is the cost of such a breach? What will the loss of sensitive customer information, trade secrets, pricing and availability, and other competitive information be to your business – directly and indirectly? Security, like performance, should be design into a system.

Open Systems & Open Source

In general, Open Systems and Open Source are preferable to proprietary systems (no matter who makes the proprietary system or how open they claim it to be). There are usually multiple avenues for support, increased options for tools and plug-ins that enhance the features or maintainability of the system, and readily available reference materials to assist those working on the systems.

Keep in mind that existing “legacy” systems may need to interface with the new system, so look for exposed, powerful APIs (application program interfaces). [XML](#) is becoming a more commonly used and accepted interface, and should therefore be considered.

It is now fairly common to find [Unicode](#) support in software and hardware products. In a global environment with language differences (traditional single-byte versus double-byte character sets) and numerous localization issues (such as currency) it is important to ensure that your system will support a diverse global user base.

Popular Products

Keep in mind that choosing popular products can be beneficial when looking for third party products and tools, reference and training materials, as well as staff and consultants. The “best choice” is not always the *best product*. While popularity should not be your primary consideration, it is something that should be taken into consideration, especially for proprietary products.

Training

It will be important to provide training to the development and administrative staff in order to fully utilize their capabilities. Training is a sometimes underestimated cost of implementing new systems, and can negatively impact a development or production rollout schedule if not properly addressed.

The Project Manager should be intimately familiar with project management concepts and their application. System Designers should understand the technology that they are working with, and be familiar with various modeling methods. Database Administrators (DBA) and Developers should have a good working knowledge of SQL and an understanding of ANSI transaction semantics. The DBA will usually benefit from a Performance Tuning course, and can pass that knowledge along to Developers. Having the right skills at all levels is important. Good training will help quickly bridge those gaps.

Spending time and money on good, relevant training is generally much more cost effective than allowing people to learn from their own mistakes. Typically it is good to either mix training with hands-on experience or have the two occur within a close time frame. A benefit of working with knowledgeable Consultants early in the design and development process is the knowledge transfer that occurs, jump-starting the process.

Development Best Practices

Best practices are a way to increase quality by utilizing approaches that have been proven at time. There are basic things, such as:

- Making sure the team understands the goals, requirements, and constraints.
- Making sure that the team understands the environment. For example, security issues would likely be different for an internal order processing system versus an external order processing system accessible from the Internet.
- Making sure that the team understands the importance of lost data and/or inconsistent data. While this is never good, it can be far worse in a regulated environment (e.g., FDA, SOX, HIPAA).
- Making sure that the team understands the cost and impact of downtime. Knowing that “downtime currently costs our organization \$4 million/hour” might motivate the team members to be more careful or make the system a little more robust in order to avoid that scenario.

Is the source code being managed by a tool like SCCS or RCS? Is only one person modifying a source module at a time? Are “make” files properly designed and implemented so that they *know* when a dependent object has changed? Are versions for each successful system compile tracked? Could you re-make any previous version of the product at any time? Is all of this information (assets) being checked-in and backed-up on a frequent basis? Are there separate and distinct staging areas for key areas (such as for a project team, an integration area, a QA area, and a production staging area). Are you sure?

Does the source code test for anticipated errors and have generic traps for unknown / unanticipated conditions? Does it log that information or create an event notification so that others are aware that it happened? Are consistent naming conventions and data types used throughout the system? Are there enough comments in the code itself?

There are many areas involved with the design and support of application system environments. For example, production control / release management, database administration, systems administration, data administration, etc. Each area should be analyzed to ensure that best practices are implemented and consistently utilized. The saying that “a chain is only as strong as its weakest link” is a great metaphor for the complexity of most information technology (IT) application systems.

Quality Management

The Quality Management Plan really sets the tone for everything done within the project. It identifies the relevant quality standards, defines the processes and best practices, identifies tools and procedures to be used, and much more. The emphasis is usually on prevention over inspection, with a goal of identifying problems as early in the development cycle as possible. Six Sigma and Total Quality Management (TQM) techniques are often the foundation for a quality management plan.

Testing is one aspect of Quality Management, and is key to success of any software development effort. Tests are designed to validate various aspects of the system, for example: functionality (i.e., the output of the process), the process itself, performance, and integration of various components. Ideally validation testing should include the following both low-level (unit/module testing, integration testing) and high-level (usability testing, functionality testing, system testing, acceptance testing) testing.

In addition to validating functionality, it is important to validate performance and concurrency goals or requirements. It is much better to identify a problem in a pre-production environment than it is when there are hundreds of production users relying on the new system. Testing should be performed against representative sets of data and with a realistic load.

In order for a test to be useful it is important that the desired results are known, and that the test is repeatable (both under success and failure conditions). Test cases and test results should always be saved for future reference, and whenever possible performance metrics should be gathered. Additionally, specific source code module information should be tracked along with the tests to ensure that the environment can be reconstructed at a later date if necessary.

There are five primary types of testing that Comprehensive Solutions recommends: white-box testing, black-box testing, regression testing, and static code analysis, and stress testing. The following is a brief description of each of these testing methods.

White-box testing tests and validates the process as a whole. This testing requires that appropriate test cases be designed based on the object's internal coding structure. Appropriate input and expected output must be devised, and the actual output compared with the expected output. This is also good for exception testing, range testing, and domain testing. Exception testing and handling is an important and often overlooked aspect of testing.

Black-box testing is requirements based testing. It verifies that, based on its specification, an object generates the correct output(s) from a given set of input parameter values. This can be a complex activity and requires that multiple combinations of input parameter values be generated and used to ensure that the object produces consistent and correct results while input values are varied. The output of black-box testing is typically saved for use during regression testing.

Important Note: As part of both white-box and black-box testing, exercise of a given object's source code should be tracked. This ensures that all logic and execution paths are tested. This also helps to find unused code that could cause future errors and/or confusion during maintenance. This coverage analysis should be performed down to the source-code line level. Typically this is accomplished by having someone familiar with the source code develop comprehensive test scenarios, and then having someone familiar with the business process alter those tests. Documentation of the tests performed should indicate the revision of source code tested.

Regression testing uses the results of black-box testing to compare new results with saved results. If an object has been tested thoroughly and no errors found, and a change is later made to the object, it must be verified that the change did not introduce any new defects. Testing to make sure the object has not taken a step backwards is called "regression testing."

Running different tests after a change will not allow verification that no new defects were introduced to the code. Therefore, the same tests must be run after each change. New tests may be added during the development cycle, but the existing tests are kept as part of the testing process. It is highly recommended that regression testing be performed on a daily basis in an XP development environment, which should occur after the daily source code build (following the daily source code check-in). Timing information (metrics) should be gathered during the regression tests and analyzed as well. This will help identify negative trends and performance issues early in the process.

Static code analysis determines whether the object being tested conforms to a set of coding rules (standards). Some of the items tested for are unused code, uninitialized variables, design errors, run-time optimization issues, and errors in naming and coding practices. This type of testing generally goes hand-in-hand with white box testing.

Stress testing helps to determine how the object will behave under varying service loads. This is commonly performed when a functional subset of the overall software system has been produced, and again when the entire system has been produced. Stress testing generates varying input loads that are fed to the software under test. This helps to find and eliminate coding errors and system bottlenecks before the software is placed into production. This is one area where an automated tool is especially useful.

System Configuration

Does the selected hardware platform provide more performance and resources than is anticipated? Is there room for growth or is the machine at its peak capacity? Is there redundancy? What about clustering and failover support?

Hardware configuration and capacity planning is an art itself. The best advice is to over-anticipate the system requirements, over-configure the system, and make sure that there is plenty of room for growth when the demand on the system is greater than the capacity. Having the best application system in the world will still seem inadequate when run on an underpowered computer!

Summary

New system development can be complicated, especially when using new technology. Costs are minimized when the entire process is comprehensive and controlled. The value proposition of using good consultants that process can be streamlined and improved, minimizing and controlling risk while maximizing the probability of success.

About the Author

Chip Nickolett, MBA, PMP is the President of Comprehensive Solutions. He has been a consultant (Technical and Management Consulting) since 1994, and started Comprehensive Solutions in 1999. He started his career as a Developer more than 20 years ago, and has managed many development efforts since then, including large projects with distributed teams and outsourcing efforts.

Let Us Help You Succeed!

Call today to discuss ways that Comprehensive Solutions can help your organization save money and achieve better results with your IT projects. We provide the *confidence* that you want and deliver the *results* that you need.

[Download our "Confidence" Brochure](#)

[Back to White Papers](#)

[Back to Services](#)

Comprehensive Solutions
4040 N. Calhoun Road
Suite 105
Brookfield, WI 53005
U.S.A.

Phone: (262) 544-9954

Fax: (262) 544-1236

Copyright © 1999-2008 Comprehensive Consulting Solutions, Inc.

All Rights Reserved

No part of this document may be copied without the express written permission of Comprehensive Consulting Solutions, Inc., 4040 N. Calhoun Rd., Suite 105, Brookfield, WI 53005.

This document is provided for informational purposes only, and the information herein is subject to change without notice. Please report any errors herein to Comprehensive Consulting Solutions. Comprehensive Consulting Solutions, Inc. does not provide any warranties covering and specifically disclaims any liability in connection with this document.

All product names referenced herein are trademarks of their respective companies. Use of these names should not be regarded as affecting the validity of any trademark or service mark.