Publication Date: 10 January 2006 **White Paper**

Commonwealth Office of Technology

Seven Habits of Highly Successful Projects

Steve King
Information Systems Architect

IT development projects can be high-risk endeavors. In order to minimize risk and maximize the likelihood of success, there are a number of best practices that can be used during the course of any development project. This document highlights seven habits of highly successful projects. Organizations should consider incorporating these best practices into their development processes and procedures so they become, in essence, organizational habits that help ensure the success of their IT development efforts.

Table of Contents

Commonwealth Office of Technology
101 Cold Harbor Dr.
Frankfort, KY 40601
http://technology.ky.gov

Publication Date: 10 January 2006

Seven Habits of Highly Successful Projects

Software projects can be among some of the most unsuccessful endeavors in history, and almost all of us have sailed the RMS Titanic at least once in our careers. Just so you know you're not alone, here's a sampling of the Software Hall of Shame:

- Problems with their inventory system in 2005 caused a Canadian department store to lose approximately $33.3 million in revenues.
- In 2004, a major automobile manufacturing company abandoned a purchasing system *after* deployment, to the tune of about $400 million.
- Upgrades to a major phone company's CRM system led to problems, which led to a revenue loss for the company of $100 million.
- One state cancelled a tax system after having spent $11.2 million.
- Two other states cancelled development of their DMV systems after having already spent $40 and $44 million dollars, respectively.

If you make the analogy of software development to home construction, then the homes we build are often functional but they aren't always that comfortable to live in: the roof leaks occasionally, the toilets overflow sometimes, and perhaps the home isn't insulated well enough to keep the gas and electric bills within the owner's budget. Worse yet, we didn't account for the time when the owner might want to expand his or her home by adding additional rooms.

Everyone knows that the success of any project is measured mainly by the system's ability to meet or exceed stated expectations. As such, there are measures of quality *throughout* the project, not just at the end. Think of the overall quality as being the sum of the quality of each of the items produced along the project timeline. For instance, analysts may create use cases and other types of documentation which you can use to create class and sequence diagrams. If the use cases are of poor quality, then the diagrams you create will also be of poor quality (relative to the stated expectations).

Obviously, there are a number of factors present throughout the project's life that can contribute to the overall quality of the end product. Rather than focusing on all the factors, however, I'll focus here on factors that can be affected during the elaboration and construction phases. I'll then make recommendations on habits you can develop now to help improve your product's quality and help ensure its overall success. In so doing, we're going to keep the following four primary objectives as drivers for the formation of our habits:

- Design
- Reusability
- Accountability
- Resource Utilization

I am going to emphasize *design* so that you allow yourself enough time to properly model the software that you are designing. Modeling has the express purpose of making abstract concepts more communicable, usually in the form of one or more UML diagrams. By doing this, you will be able to catch design problems before any code is actually written.

By *reusability*, I mean that I want to identify opportunities to leverage pre-built, pre-tested components in your systems so that you don't spend unnecessary time re-inventing the wheel. You will also be able to reduce the number of defects with which you will have to contend. This may mean creating services or components internally or purchasing them from a third-party.

Accountability means that you are going to take ownership of your code and subject it to the scrutiny of your peers. Often, your fellow developers can spot potential problems or even more efficient ways of doing things more quickly than you can, i.e., "can't see the forest for the trees" syndrome. By reviewing your code, you'll also ensure that it meets all of your organization's published coding standards and best practices.

Finally, I'll look at the *utilization* of your most valuable resources: developers! I'll discuss ways to take advantage of individuals' natural strengths, talents, and interests, so as to better align them with their tasks.

Habit 1: Manage Expectations

Of all the recommendations to be made, this is perhaps the most important. As a developer, you probably don't like saying "no" when you're asked if you can have something done by a certain date; so you feel obliged to say yes, even though your instincts may be telling you something altogether different. You don't necessarily have to say "no," but you do owe it to yourself and your customers to be *honest*.

If you don't have the personality to deal with the repercussions of saying "no" to your Project Manager, then ask someone in authority on or outside of your project to help you. Never underestimate your instinct when deciding when to put the brakes on. More importantly, if you're asked to do something that you fear will put the project in jeopardy, then *get it in writing* from your Project Manager. Besides, it's not just enough to say "no;" you have to be ready to explain why and offer alternatives.

Keep in mind that your management has certain expectations from you as a developer. For example, you are expected to adhere to certain standards and best practices. If you're asked to do something that conflicts with management's expectations, you have the right and the *responsibility* to remind your Project Manager of your obligations, and let them work with the customer in making the decision as to whether to cut corners or not. Ultimately, you and your team will be held accountable by your own management if things go south, so be prepared to defend your actions, especially those that may jeopardize quality. The best way to do this is to require your Project Manager to obtain written and signed approval from the customer. Often when customers are required to sign something, they will reconsider. This isn't because your customers are being difficult, it's just because they may not know the risks involved in what they are asking. You have a responsibility to help identify them.

Habit 2: Model Software First

We all know that a picture is worth a thousand words. Modeling software offers an effective way to communicate what needs to be built through the use of tools such as use cases and class and sequence diagrams. Using these tools, the Program Manager can better:

- create and communicate software designs before committing additional resources
- trace the design back to the requirements, helping ensure that they are building the right system
- practice iterative development, in which models and other higher levels of abstraction facilitate quick and frequent changes

Most systems can benefit from at least some modeling. Insist that you be given ample time to design the system before the first line of code is ever written. It is important that you advise your Project Manager that the success of the project is very dependent upon the time allotted to design and modeling, so that he or she can provide enough time in the project plan.

UML (Unified Modeling Language) is the standard way to model your designs. There are several different types of UML documents. The following table lists all eight of the current UML diagrams and what role they play:

Diagram	Purpose
Activity Diagram	Modeling workflow, analyzing use cases, and dealing with multi-threaded applications
Collaboration Diagram	Shows a set of object roles related in a particular context, and in interaction, which the set of messages exchanged among the objects to achieve an operation or result
Component Diagram	Used to partition a system into cohesive components (source code files, DLLs, etc.)
Deployment Diagram	Implementation-level diagrams which show the structure of the run-time system
Sequence Diagram	Shows actors or objects participating in an interaction and the events they generate arranged in a time sequence
Statechart Diagram	Represents a state machine
Static Structure Diagram	Can represent concepts from the real world and relationships between them (conceptual) or decompose a software system into its parts (class). Unlike a conceptual diagram, the parts in a class diagram represent fully-defined software entities rather than conceptual entities.
Use Case Diagram	A diagram showing the representation of a set of events that occurs when an actor uses a system to complete a process

You can always scribble your design down on a cocktail napkin, but your team is going to appreciate your design much more if it is presented to them in a professional manner. There are a number of excellent modeling tools on the market today; and given the current move towards Model Driven Architecture (MDA), as well as Model Driven Development (MDD), these products are bound to increase as companies react to increasing demand. Following is a list of four products which are widely used today (in no particular order).

Product	Description
Microsoft® Visio®	General template-based diagramming application; supports all 8 UML diagrams
IBM Rational Software Modeler (RSM)	Customizable UML 2.0-based visual modeling and design tool
Borland® Together Architect/Designer	UML 2.0-based integrated visual modeling tools; supports forward and reverse engineering as well as synchronization between model and code
MagicDraw	UML 2.0-specific software modeling tool

Habit 3: Standardize Best Practices and Conventions

You must have standards if you want consistency and quality. Just think of what would happen if NASA's engineers couldn't agree on whether to use the metric or US system of measurement. Both units of measurement are fine—as long as everyone is working with the same one.

If your team isn't designing or constructing systems using standards-based practices, then you are definitely putting the quality of the source code, and therefore the project, at risk (the degree to which is subjective, of course).

Now, when I say standards, I'm not necessarily talking ISO 2000. In fact, unless you are already implementing ISO software development standards, I would recommend that you keep your standards simple, easy to read, and easily accessible. Otherwise, your team isn't going to bother reading them, and you'll have to work extra hard to see that they're implemented.

You can start by standardizing or conventionalizing one or more of the following:

- **Development Environment**. Describes the needed development and support tools, recommended hardware requirements, location of source code, partitions, project and directory structure, etc.
- **Naming Conventions**. Describes naming conventions for files, variables, classes, interfaces, controls, etc.
- **Coding Conventions**. Recommends standards or conventions for coding, such as the use of literals, exception handling, commenting, indentation, etc.
- **Database Naming Standards**. Standardizes the naming of tables, columns, primary keys, foreign keys, indices, stored procedures, functions, etc.
- **Look and Feel**. You even want to standardize, to some degree, the look and feel of your applications. One way to implement this is through the creation of standard cascading style-sheets for web applications or template or base forms for desktop applications.[1]

If you've never had well-defined standards before, then you may find some resistance from your team. They may feel that standardization will stifle their creativity. Depending on the size of your shop, you will want to invite some of, or perhaps all of, the developers to participate in the standardization process. Be careful, however, not to get too many people involved, or you may never get anywhere. Select a team, decide what you will be standardizing, set a goal date, and then schedule a regular time that your team will meet. Don't wait on a unanimous vote to include (or exclude) something in your standards or you may hit a brick wall, which can quickly take the steam out of your effort. Instead, build consensus (general agreement) among the team, and then put a stake in the ground by codifying the standard and publishing them on your company's intranet. Also make sure to include these standards part of the training process for new developers.

It won't take your team very long to incorporate these standards into their work; it will very quickly become habitual. Also make sure there's a process for changing your standards as technology changes. You'll soon find that your developers will *demand* that standards and they'll wonder how they ever got along without them!

Habit 4: Implement Application Development Frameworks and Pre-Built Libraries

An application framework is a set of libraries or classes that are used to implement the standard structure of an application. Frameworks can reduce time and help the team reproduce successful systems by using well-tested components. The most apparent advantages of using a framework are that they:

- Reduce development time
- Reduce total cost of ownership
- Reduce failure rate

Depending on your programming platform, there may be a variety of application frameworks open to you either commercially or as open-source. Some examples of application frameworks are:

- **Struts**— an open-source framework for developing J2EE web applications
- **PDSA**—an application generator and framework for the .NET environment
- **Telon**—an application development system for COBOL that generates applications

[1] Some modern application development languages, such as .NET, Visual FoxPro, and Delphi, allow visual controls, such as forms, to be subclassed.

Code libraries are reusable source-code components that provide proven solutions to common development challenges. In fact, some application frameworks are nothing more than a collection of code libraries. Using pre-built code libraries greatly reduces the amount of time needed to code, and can therefore help reduce the number of defects in your software.

Take Microsoft's Enterprise Library, for example. This library is a free collection of application "blocks" that provides a consistent way to do any of the following:

- Access a database (SQL Server, Oracle, DB2, or any other ODBC-compliant database)
- Implement security
- Handle exceptions
- Manage system configuration
- Apply cryptography
- Manage cache efficiently
- Perform logging and instrumentation

Microsoft Enterprise Library and various commercially available code libraries are thoroughly tested and often extensible—so there is almost no need to write the same functionality from scratch.

On a side note, code generators can be useful to reduce development time. Be sure that these generators allow you to customize the code that's created, if necessary, especially if you want the resulting source code to match your own internal standards. Most importantly, don't use a code generator in lieu of good modeling practices.

You and your team should decide for yourselves which libraries and frameworks work best for the type of work you do, and then make the decision, with the support of your management, to either purchase or download these components, and enforce their use in the systems that you develop.

Habit 5: Enforce Code Walkthroughs and Audits

Your developers may not be used to having their work critiqued yet it's necessary to ensure that their work meets the enterprise's standard for quality. A couple of ways to review the work being done are **code walkthroughs** and **code audits**.

A *code walkthrough* is a manual testing technique where program source code is traced manually by a group with a small set of test cases to analyze the programmer's logic and assumptions. These walkthroughs *must be part of the project plan* and appear as regularly-scheduled activities. The frequency with which they are conducted and the audience to which they are presented should be determined by the project's lead developer. Code walkthroughs are beneficial because they

- Reduce defects by finding problems early
- Provide a platform for developers who get "stuck" on a problem
- Encourage collaboration among team members

A *code audit*, on the other hand, is a review of source code by a person, team, or tool to verify compliance with software design and programming standards. Correctness and efficiency may also be evaluated. This is where the standards you developed in Habit 3 come in very handy.

Program Managers must insist that they be allowed to perform walkthroughs on a regular basis throughout the development process. If not, it is their responsibility to notify their Project Manager of the risks incurred when walkthroughs are not performed. If your Project Manager (usually at the behest of the customer) insists that you not perform walkthroughs or audits, get this decision in writing. If the product is delivered with an "unexpected" number of defects, then you'll be in the position to defend yourself and your team if you have something in writing.

Habit 6: Manage the Build Process

Experience has shown that systems often do not have a well-defined or well-managed build and deployment strategy. This usually results in a situation where a particular release has been deployed and a critical defect has been identified, but the developers have already begun working on the next release. If the current version is pulled from the repository and changed, then the modification must be merged into the code for the next release.

If versioning does not take place, then the situation becomes even worse: the programmers have to comment all of the changes they make to the code for the next version, so the program will compile and also correct the defect. This type of reactive development is *very* risky, and has to be mitigated by relaying these risks to the Project Manager, who can then communicate these risks to the customer and get approval before proceeding. The chances of introducing new defects into the break-fix release increases dramatically, since the programmers have to rush to disable blocks of code.

There are several actions that can be taken to help reduce the number of problems encountered during this important phase.

1. Insist that the delivery of each version is specified clearly in the project plan, and allocate time for all of the activities required to build and deploy the software.
2. Treat break-fixes as mini releases. That is, if, after a version has been put into production, a critical defect is found, then the team should either a) fix the defect in the current version so as to be delivered on the next release (as scheduled), or b) use the current (production) version code to fix the problem and fix the problem in the next version code as well, so that it is not repeated in the next release. To some extent, this merging of changes may be aided through some type of automation, but regardless of the method of execution, the changes need to be made in at least two different places.
3. Create a team of junior developers who can be devoted to building the applications. This would not be their primary duty, of course, but a pool of several developers could be leveraged to help package the system for delivery, under the guidance of the Program Manager or someone he or she designates. This would have the benefit of allowing the product team to begin working immediately on the next release as soon as they have versioned the current version in source control.

Figure 1, below, helps illustrate these concepts. Notice that the development cycle for version 2.0 can remain uninterrupted, provided there are adequate resources to work on both the break-fix and the pending version. If, on the other hand, there are not enough resources to address the break-fix while continuing version 2.0, then work on version 2.0 should be frozen until the defect has been corrected, tested, and versioned (1.0a). Also notice that even the break-fix has been placed into the timeline and given sufficient time for integration testing, sign-off, versioning, and building.

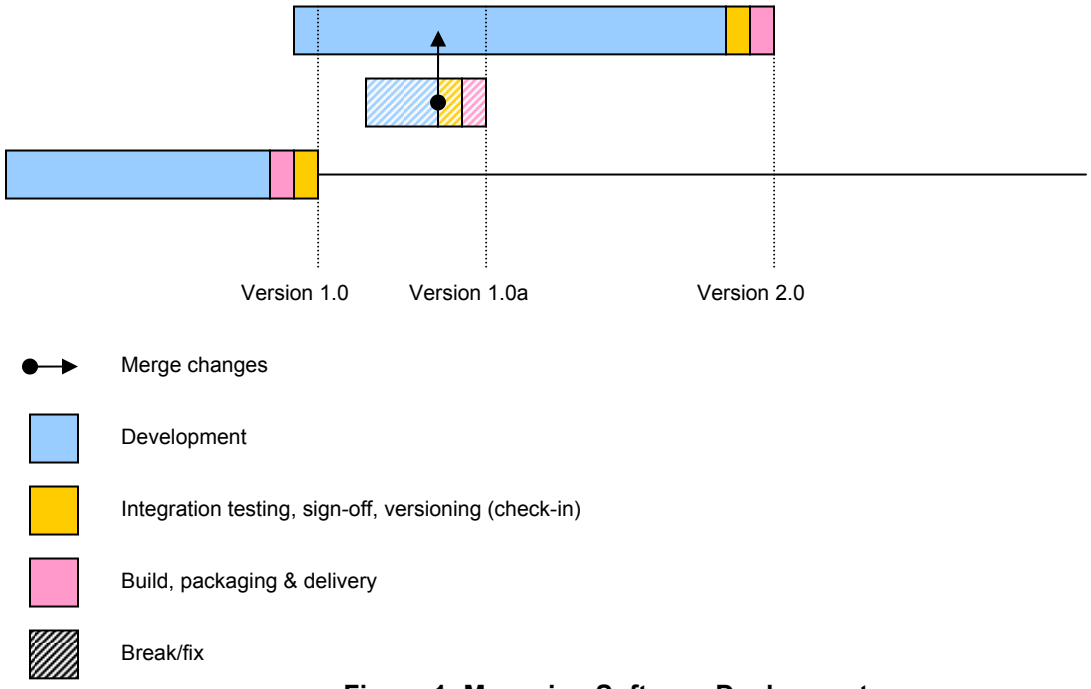

Figure 1: Managing Software Deployment

By following these recommendations, developers:

- Force the PM to accommodate the entire implementation cycle into the project plan rather than it being a reactive activity
- Alleviate the development team of the added responsibility of performing the build, packaging and deployment of the software[2]
- Strengthen our ability to quickly adapt to changing situations after deployment

[2] The development staff *is* responsible for documenting the implementation process for the system. It should never be assumed that those who deploy our solutions know the idiosyncratic tasks that sometimes must be done to successfully install or otherwise implement a large, complex application.

Habit 7: Organize Development around Skills and Tasks

Any complex task requires people with different skill levels. Let's return to the house analogy. You'll have plumbers, electricians and carpenters all doing different, yet interrelated, tasks on the same project—your home.

Ideally, constructing complex software systems should be no different. An organization that has many application developers will find that not everyone has the same strengths or even the same interests. Try to utilize people by skills or tasks for which they are best suited, to harmonize strengths and compensate for weaknesses. This will lead to an increase in productivity and effectiveness, not to mention morale.

If we compartmentalize the different areas of development on any typical distributed system, we find that there are basically three. These correspond to the different activities that are concurrently being performed during any such project: presentation, business modeling, and database programming. By aligning your staff with their natural talents, skills, and interests, you are able to get the biggest bang for your buck!

Take, for instance, a developer with very strong HTML, CSS, or graphical skills. He or she may find the task of creating user-friendly, accessible, aesthetically pleasing web pages very rewarding. Someone who prefers working with stored procedures, triggers, and data modeling, on the other hand, would find such work boring, frustrating, and unrewarding. By allowing individuals to work in their niche, we produce an environment in which each team member is working with his or her best skills and doing something he or she enjoys. No one sits down to produce bad work. We all want to be proud of what we do and go home at the end of the day feeling that we've accomplished something.

Most modern software applications can generally created in a layered approach. That is, each component to be developed will fall into one of three main categories: presentation, business, or data.

The **Presentation Layer** is, of course, what the user sees. It's the face of the product being designed. Developers working with this layer should have an appreciable aptitude not only for aesthetics, but also for things like accessibility, cross-browser support, and JavaScript.

The **Business Layer** is the plumbing, or machinery, of the application. This is where business rules and logic are implemented. Developers working with this layer need to have more experience working with object-oriented programming, implementing or creating design patterns and practices, modeling software, and interpreting use cases.

The **Data Layer** is responsible for retrieving and persisting data to any data stores used by the application, i.e., one or more RDBMSs, XML files, text files, message queues, etc. Developers working with this layer may need to be familiar with technologies such as Structured Query Language (SQL), modeling databases (logical), conversion and transformation of legacy data stores, etc.

The table below summarizes the types of skills needed in each layer:

Presentation Layer	Business Layer	Data Layer
Skills • HTML • Accessibility • CSS • Browsers • Programming (Novice to Intermediate) • Class diagrams	**Skills** • Programming (Advanced to Expert) • Software modeling • Use cases • OO design	**Skills** • SQL • Data modeling • Conversion • Transformation
Duties • Client-side validation • Screen design • Enforce UI standards	**Duties** • Business object models development • Server-side validation • Web services • Security • Session management	**Duties** • Assist with query development • Enforce database security and performance best practices • Ensure compliance with enterprise database standards

Table 1: Software Construction Compartments

In **Figure 2** below, a rectangle represents each layer. The arrow on the left represents the order of dependencies. For example, the Presentation Layer developer will consume components from the Business Layer in order to construct a meaningful application; otherwise, he or she has simply created a flashy, yet useless, user interface.

Similarly, the developers at the Business Layer will need to consume code written by the Data Layer developers in order to retrieve and to persist data from the data stores.

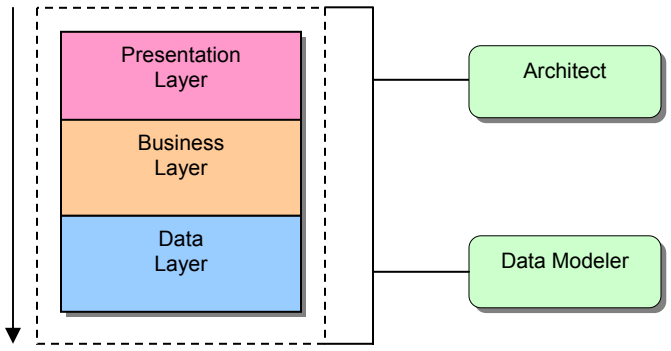

Figure 2: An ideal project organization

The idea here is to let each group of developers (at each layer) work on what they are good at, and, as a team, put together the entire solution, just like building a house. You may also find a few developers who will cross boundaries, as their natural talents and abilities transcend any attempt to box them into a particular development layer.

Finally, there are two special roles who work alongside every project. They are the Architect and the Data Modeler. Both of these roles work on a project using a consulting model by offering technical advice and guidance, especially concerning security, quality, reusability, performance, and scalability. Both of these roles should be filled with experienced personnel who with good leadership qualities, as they will also help to enforce your enterprise design standards. Don't worry if your organization doesn't formally define these two roles. A *de facto* architect or data modeler work just as well.

Get in the Habit!

I've identified seven "habits" for improving software design and development so now all that's left to do is to start practicing them. Experts say it takes about twenty-one days to form a new habit so in less than a month you and your team will be well on your way to becoming a well-honed, professional development shop.

I will leave you with a few recommendations to help you get started. What do you have to lose?

- Perform a skills assessment to identify the strengths and interests of team members. Do not, however, let your developers tell you their skills. It is human nature to inflate our own ability, so such an assessment would ultimately be of little use. Consider hiring an outside agency to perform the assessment.
- If you need to, provide software modeling training. You can't expect your programmers to create (or understand) UML diagrams if they've never seen one.
- Adopt one or more application development frameworks (and consider the idea of outsourcing this task as one option). If you adopt a framework, be ready to spend some money. You may find a good free framework, but you should make sure it's supported.
- Develop a small committee for developing standards and give them a hard and fast deadline. Make sure they have a place to publish these standards once they are finished and process to make changes to the standards as needed, i.e., a monthly review meeting.
- Enforce adherence to enterprise standards through code walkthroughs and audits. There should be consequences if code walkthroughs or audits are not performed. Hold Program Managers accountable by making them responsible for communicating the need for walkthroughs to their Project Managers, and by identifying risks when asked to forgo them.
- Enforce the use of code libraries such as *Microsoft Enterprise Library* for standard data access, exception handling, etc., unless there is a good reason to do otherwise.
- Work with the Project Manager to ensure that the enough time has been allotted in the project plan for the deployment process.