USING PERIODIC AUDITS TO PREVENT CATASTROPHIC PROJECT FAILURE

Dr. Paul Dorsey, Dulcian, Inc.

In the early part of the 17th century, Sweden endeavored to construct the grandest warship ever built. It would be larger, heavier, and carry more firepower than any ship ever built. It would be covered in beautiful carvings so, not only would it be capable of destroying any enemy vessel, it would do so in grand style. The construction of this ship consumed 5% of the country's GNP. In 1628, the VASA set out on its maiden voyage. It sank to the bottom after sailing 1000 meters from the dock in a light summer breeze, killing 50 people. The irony is that this was not an unexpected event. The project made every mistake common to IT projects (scope creep, change in project lead, using new technology, failure to follow a careful design). The worst part was that, although the ship failed when it was tested, no one informed management (the king). Not only did the project fail, but 50 innocent lives were lost. This cautionary tale exactly parallels some massive IT project failures (although rarely do IT failures kill more than careers and profits).

Year after year, we continue to build the IT equivalent of the Vasa. The main difference between the doomed historical ship and "IT Vasas" is that when a ship sinks, construction stops; on an IT project, we continue to pump millions of dollars into a system after it has already failed. We cannot prevent poor decisions, but we can prevent people from ignoring them.

Managing Successful Projects

Several years ago, I was contacted by a graduate student who was working on a thesis comparing system development methodologies. The student wanted to create a contingency model indicating which style of development was appropriate for which type of projects. I was asked to give my opinion about when each proposed development methodology should be employed. I thought it was an excellent question; however, I don't think the answer I gave satisfied the student very much. I replied that the software development method employed had very little effect upon project success. I have seen successes and failures using every method. If the development team used no formal process at all, failure was guaranteed, but it did not seem to matter which development method was used.

What I told the student was that project management is like leading a bunch of children on a wilderness hike. Periodically, you should climb to the top of a tall tree and see if you are still going in the right direction, find out if there are obstacles in the way, and make sure that the path you are on will get you to your final destination. Climb back down from the tree, gather all of the children around (most will have wandered off into the woods), and point them all in the right direction prior to moving on.

As simple as this description of project management sounds, it is shocking how often managers do not follow it. Project managers may go for months (if not years) without taking stock of the real status of a project. As a result, the architecture of the system may be unsound and failure may be inevitable, but this critical information will not be discovered in a timely manner. Frequently an unsound system foundation is not discovered until there is an attempt to put the system into production. Of the many shortcuts that are taken in the name of expediency, there is none more dangerous than failing to make a periodic assessment of the overall project status.

Auditing is the critical success factor in preventing massive failures. IT projects need to be periodically examined to make sure they are on the right track. Failure to periodically assess means that you could be building a system that will never work and be unaware of it.

These assessments can be internal, but IT professionals are notoriously poor at self-examination. The most successful way to get developers to consistently test their own code is to force them to write the tests before they write the code (the so-called test-first concept from the Agile methodology). The official audit should not be done by the project team.

A project audit is a significant, substantial effort. It is not a review of one part of the system (a security audit is not a project audit), nor should its scope be limited. The audit should answer the question "Are we building a system that will meet user requirements?"

Audit Costs

Useful IT project audits are both time consuming and expensive and will the delay the project. They are intrusive, annoying and usually create political land mines. Assume that the audit will consume 5-10% of the total cost of the system (excluding hardware). For systems that have been stuck in analysis or development paralysis, the cost may be much lower. Therefore the actual audit cost is 5-10% of the current total productive work completed on the project, which, unfortunately, may be far less than the total amount spent on the project.

Meaningful audits can't take place without the participation of the project team. Team members will need to walk the auditors though the system, prepare architectural overviews, and educate the audit team.

Audits can also be very threatening to a project team. In the rare cases when they are done at all, audits are usually only initiated when a project is known to be in trouble and management is looking for a scapegoat or a different solution.

Everyone connected to the project will declare an audit to be a waste of time and effort. Project management will feel threatened and wonder why you don't trust them. The developers will wonder why they need to help these outsiders when they should be coding.

Given the economic and political costs, it is little wonder that IT project audits are so rare.

Audit Benefits

An audit will tell you whether or not you are building a Vasa. Failing a $200 million project after $20 million (rather than $300 million after cost overruns) is a huge benefit. Of course, the second time you try to build the project may be no better than the first, but at least you will not be going down a road that is doomed to failure.

There are hundreds of articles quoting industry project failure rates that vary from 20% to 80%. These articles differ about what failure means, but regardless of the source of the statistics, there are enough project failures to conclude that audits make good economic sense.

Periodic assessments can help prevent several common problems:

1. **Ensuring that project milestones have actually been achieved.** Assessments should be made:
   - when determining project scope
   - when the data model is thought to be relatively stable
   - after the basic user interface design is complete
   - after each major architectural decision.

   Such assessments are best made with the assistance of an outside auditor who is not otherwise connected to the project. The act of defending a decision to a third party often helps to focus the thinking of the team.

2. **Averting unforeseen disasters.** It is very easy to think that a project is going along smoothly when many people already know that it is doomed. Asking developers and users involved in the day-to-day project development process how things are going on a regular basis can help reveal problems looming on the horizon.

3. **Inappropriate resource allocation.** At the outset of a project, it is not always possible to accurately determine the number of hours or all of the resources required for each phase of the project. By taking time to frequently assess project status, tasks that have been omitted will be discovered and a reassessment of the outstanding issues will ensure that they are being addressed.

Audits do more than simply prevent disasters. They can also help to identify missed opportunities and find places to improve even a very successful project. An audit will help the existing team to take stock of the project and provide them with opportunities view the project from another perspective that they might otherwise never have seen.

The Politics of the Audit

The political impact of a project audit must always be taken into consideration. If the project development team or the project manager resists an audit, it is time to worry. A competent IT professional should be proud of his/her work and be happy to get another team's perspective. If they resist an audit, this usually means that they already know what the audit will find and that the result will not be positive.

It is important to ensure that the auditor is truly independent and brought in only to audit. There should be no chance that the auditor can work on the project after the audit. There should be no conflict of interest. The auditors should neither be partnered with, nor have an adversarial relationship with the development team. On one audit, (where litigation was likely) I was not even told what consulting organization the previous team came from and I had to perform most of the audit from existing documentation.

Who does the auditor work for? This is a very important question. On one audit, I was brought in by the prime contractor who had a bad relationship with the sub-contractor responsible for all of the IT work on the project. I was told to make the report to the client as general and positive as possible. In the report for the prime contractor, I was told to list everything that was wrong with the system requiring correction. The point is that I was hired by the prime contractor as their agent. Therefore, only in my interactions with the prime contractor was I a totally independent auditor.

An audit is never totally objective. Biases always exist among the members of the audit team. They bring their own way of doing things and their own view of the world. However, if you are building a Vasa that will never float, even a biased audit will reveal that valuable information. An audit team with a different approach will help to bring new ideas to the project.

Finding the Right Auditor

This may be one of the more challenging aspects of performing a successful audit. IT projects are so rarely audited that auditing is not a widely discussed or advertised service.

You may need to get a team to perform the audit whose members have never done a project exactly like the one being audited. However, any reputable consulting firm should have some experience of reviewing existing work as well as creating plans for assuming responsibility for an existing project to go forward, which is effectively the same task as an audit.

You will not need a large audit team, but each member should be highly skilled. You will need to make sure that each area of the audit has a technical lead who can evaluate the quality of the system. Minimally, you will need the following resources:

1. Project Manager – This individual must have managed projects similar in size and scope to the one being audited. If he/she has never managed anything larger than a $1 million dollar project, that person is not a good choice to audit a $100 million dollar project.
2. Database Administrator (DBA) – A DBA is needed to audit the underlying database and make sure that the system is set up to conform to industry standards. This resource can also help evaluate the system backup and recovery procedures.
3. Application Architect – User Interface design and development is its own specialty. This area is just as important as the database. In Object-Oriented (OO) or Service Oriented Architecture (SOA)-driven systems where most of the logic is not coded in the database, this individual may be more important than a good DBA. Note that SOA and UI architecture are not two different areas that can be audited independently. The application audit must be done as a whole.
4. Security Specialist – Assessing system security (particularly web-based systems) is usually not within the skill set of most designers. You need to have a security specialist for this task. Experience with high-security military, medical, or financial systems is useful in selecting the appropriate specialist since security in those industries is vital.

When selecting the audit team, be sure to evaluate the actual individuals performing the audit and not just their firm's reputation. This is a very specialized operation that requires very high-quality talent. You don't need an army of junior developers reviewing each Java class for adherence to documentation and coding standards. An audit is not a total quality evaluation. It should indicate the health of the project, not identify every undocumented class.

Structuring the Audit

An audit should not be an isolated effort. In addition to substantial periodic audits, large projects should have some level of independent continuous monitoring to make sure that the ongoing management of the project is on track.

There are various sections within a system audit:

- The first step for the audit team is to understand the system. The auditors should have a thorough knowledge of the system prior to critiquing it. This initial phase of the audit follows the same process as having a new team take over a project. The auditors formulate a project plan, begin reviewing documentation, and walk through the system architecture and features.

- The audit should include scalability and performance tests. IT development teams are notorious for hiding system performance and scalability flaws.
- Part of the review should also include direct interviews with users and observations of the working system.

A. Knowledge Transfer

This step allows the auditors to understand the entire system architecture as if they were taking over system development. Declaring that the level of understanding of the auditors must be sufficient to take over development of the system helps to clarify the detailed knowledge needed by the auditors. Many auditors do not try to really understand a system prior to commenting on it . This makes their audits superficial and necessarily flawed. The goal of this phase is to gain an understanding of the complete system prior to working with any or all of the individual system areas.

The following areas should be reviewed for the knowledge transfer portion of the audit:

1. System Overview – Review the existing Strategy Document. This will allow the auditors to understand the project charter, roles, responsibilities, and core use cases to be supported.
2. Data Model Walkthrough – Review or create documentation describing each database table and attribute.
3. Review/Identify Transaction Use Cases – Review the requirements documentation or interview current personnel to identify how the use cases are supported by the current system and to understand how users interact with the existing system.
4. Review User Interface Code Architecture – This will help the auditors understand how the existing applications are built and maintained. Of particular interest is the amount of information being moved from the client to the application server and from the application server to the database.
5. Review Reporting Requirements/Architecture – Examine how the reporting requirements are met in the existing system
6. Review System Architecture – This includes the information transfer/synchronization mechanism in distributed systems.
7. Review System Installation/Upgrade Mechanism. This requires installing the entire system on the auditor’s computers.
8. Internal Control review – Identify known exposures and match them to existing internal controls. This is a much broader review than simply examining what computer security controls are in place and includes business exposures such as human data entry errors and collusive fraud.
9. User Access review – Examine architecture controlling user names, passwords, roles, privileges and how these are controlled in the system.
10. Review process for handling system bugs and enhancement requests. Review the existing bug handling and configuration management processes.
11. Multi-Lingual capabilities (if applicable) – Review the current system's multi-lingual architecture and capabilities.
12. Basic System Requirements – The number of users, bandwidth restrictions, transaction volumes, performance requirements should be quantified and evaluated.
13. Process Flows – Determine the current system process flows and how they are implemented
14. Custom Framework –Review the application development framework and documentation for any custom frameworks.
15. Performance –Audit system performance to determine the number of users in the system, number and types of transactions performed, and the effectiveness of existing performance testing. Identify performance bottlenecks.
16. Standards – Review user interface coding standards and how these standards are implemented and enforced
17. Training – Review existing training methods and trainer qualifications

B. Infrastructure Audit

In this phase of the audit, examine each of the following areas from a technical soundness perspective. Compare existing system practices to current industry best practices and document any discrepancies.

1. System and User Documentation
   - Includes code commenting, training manuals

2. Data Model Audit – Examine the system model for the following:
   - Violations of database normalization rules

- Appropriate level of abstraction
- Consistency of naming conventions
- Adequate documentation of the model
- Consistency and appropriateness of data types
- Flexibility to accommodate future changes to requirements

3. Database Review – Examine the following aspects of the database structure and design:
    - Appropriate usage of server-side code, particularly for batch routines
    - Quality/performance of SQL
    - Database performance including creation of appropriate indexes

4. User Interface (UI) Architecture Review – Examine the following aspects of the user interface code and design:
    - Binding framework
    - UI complex rule enforcement
    - Resulting network traffic
    - DML Operations
    - State management
    - Row caching
    - Appropriate use of relevant architecture
    - Code and documentation

5. Distributed System/ETL Audit
    - Complexity and maintainability of code
    - Quality of architecture

6. Security Audit
    - User interface susceptibility to code injection and cross-scripting
    - Control/exposure matrix review
    - User privilege security review

7. Hardware/Software/Networking Review
    - Review existing hardware/software and networking equipment and assess suitability to meet system requirements

8. Backup/Recovery Procedures
    - Review plans for system backup/recovery in the event of system failure or damage

9. Appropriateness of system upgrade mechanism

C. System Ability to Meet Current and Future Requirements

Examine the current set of system requirements, identify all of the use cases, and review a sample of the use cases for suitability, specifically:

- Compare documented requirements to the existing use cases and how they are handled.
- Interview users to assess satisfaction with the existing system.
- Determine whether or not the existing backup and recovery procedures are sufficient to meet maximum downtime requirements.
- Assess flexibility of the system to meet new requirements.

D. Financial review

Auditors should review how resources have been spent over the lifetime of the project including budgeted expenses versus actual expenditures for each year of the project.

Audits of Commercial-Off-The-Shelf (COTS) Software Projects vs. Custom System Development Projects

Given that COTS projects fail with the same regularity as custom development projects, the need for an audit is just as great; however, the structure of a COTS audit is a bit different. If the project uses mainstream COTS packages (e.g. Oracle or SAP), there is little need for a data model audit because the strengths and weaknesses of those structures are well known.

Very serious attention should be given to any customization of the COTS. Were the customizations necessary? Were they appropriate? What will be the impact of the customizations on the ability to move to the next release of the COTS? Most COTS projects have enough customization that they need the same level of scrutiny as a custom development project.

The Audit Reports

A comprehensive audit should result in multiple reports. There is usually a high-level finding report for top management to answer the big questions. This report should be written in terms that non-technical professionals can understand. This high-level report should be no more than 2-5 pages.

A detailed audit report should be written for the project manager. It should describe all major system flaws in detail and discuss the architectural strengths and weaknesses of the system design. The executive summary of the detailed report should be no more than 10-15 pages. A complete listing of all observations, flaws, and suggestions might easily require 100 pages or more.

The existing team should have access to the audit report and be given a chance to respond to it. Existing team members should have participated in many parts of the audit and senior team members should have been included in evaluative discussions of the system. The existing team may not agree with the findings of the audit but, by the time the report is released, there should be no surprises.

The existing team should be provided with an opportunity to write a response to the report to be included with the final audit report.

Acting on the Audit

The most difficult part of the whole audit process is determining how to take corrective actions. These actions may potentially result in discarding large portions of the existing system.

If an audit brings significant architectural weaknesses to light, the findings of the audit must be validated. Even if the existing team agrees with the findings of the audit, major shifts in the direction of a large project should not be undertaken lightly.

It might even be prudent to bring in another outsider to review the audit report to determine whether or not its findings are legitimate. If the project being audited costs hundreds of millions of dollars, some extra prudence is appropriate.

Once the audit report has been accepted, the basic sunk cost argument must be applied. The amount of money already spent on the project is not relevant information. The decision about how to proceed should be based on the lowest estimated cost to complete the project. Technology may have advanced since the project's inception. Therefore, starting over may be the easiest way to proceed.

Sometimes calling something an "upgrade" rather than a "refactoring" or a "rewrite" is much more palatable, especially to management. However, when some or all of an existing system is not capable of meeting user requirements, the system must be modified whether the changes cost $10 or $10 million.

Conclusions

Stopping from time to time, stepping back and asking "Is this system ever going to work?" is the key to not being surprised by a massive system failure.

There are a few key points to keep in mind:

1. Audits need to be performed by outside resources. Keeping them truly independent is a critical success factor.
2. Audits will not take the place of good design and should only comprise a part of the overall quality assurance strategy of the project.
3. Audits are large, substantive efforts. A superficial audit is worse than no audit at all since it tends to provide a false sense of security.
4. Both COTS and custom developed systems require audits.

Audits, though unpopular, expensive and intrusive, are the best and last defense against massive project failure.

About the Author

Dr. Paul Dorsey is the founder and president of Dulcian, Inc. an Oracle consulting firm specializing in business rules and web based application development. He is the chief architect of Dulcian's Business Rules Information Manager (BRIM$^{®}$) tool. Paul is the co-author of seven Oracle Press books on Designer, Database Design, Developer, and JDeveloper, which have been translated into nine languages as well as the Wiley Press book *PL/SQL for Dummies*. Paul is an Oracle ACE Director. He is President Emeritus of NYOUG and the Associate Editor of the International Oracle User Group's SELECT Journal. In 2003, Dr. Dorsey was honored by ODTUG as volunteer of the year, in 2001 by IOUG as volunteer of the year and by Oracle as one of the six initial honorary Oracle 9*i* Certified Masters. Paul is also the founder and Chairperson of the ODTUG Symposium, currently in its eighth year. Dr. Dorsey's submission of a Survey Generator built to collect data for The Preeclampsia Foundation was the winner of the 2007 Oracle Fusion Middleware Developer Challenge and Oracle selected him as the 2007 PL/SQL Developer of the Year.