

Remediation – What to Do When Your Software is Failing

by Decision Design Corporation

Creating and implementing a successful new software system is difficult. Numerous studies have shown that companies have only a 1 in 4 chance of developing and delivering successful software, with most attempts ending in either total failure or delivery of only a marginally better system than was replaced. New software development regularly disappoints those that attempt it.

If you happened to be engaged in such a risky endeavor, chances are you are now trying to figure out what to do next. Your system or project is either not going well or has already been delivered and is not meeting your expectations. Many companies in this position refuse to admit their software is failing, instead believing that with just a little more time and money, it will finally succeed. Other companies give up completely, scrapping the project, and reverting back to whatever they had before. Finally, some companies just muddle through, rationalizing that their newly delivered system is acceptable, when everyone can see it needs to be fixed.

But there is a way out of this problem. There is a solution to failing or failed software. Even if you believe you have the worst software ever created, it can be saved. How? Software Remediation.

What is Software Remediation?

While the term Software Remediation may not be a household word, the concept is obvious. Software Remediation is the process of salvaging failed or failing software. It's about closing the application gap between a failed project and a successful project. It's about removing the cause of your users' or customers' frustration. It's about completing software that a vendor left unfinished. It's about recouping as much of the cost you already put into a project as possible. Remediation is about stopping the bleeding, returning a project to the path of success, and then delivering successful software. Most importantly, Software Remediation is about getting your software to deliver the some or all of the value you expected from it when you first started the project.

Is Your Software Failing?

The relative success of a software project, like any new initiative, exists on a continuum between Complete Failure and Total Success, and no project is on either end of this continuum. Before you conclude your software is failing and decide to throw it away - or before you conclude your software is acceptable and move forward - it is important to know where on this continuum your software actually is. Decision Design developed a Software Success Scale that visually describes this continuum and can be used to help describe when a project has succeeded, when a project should be remediated, and when it should be scrapped altogether.

First, some terms. The three categories of software success describe virtually all software projects. Every software development project ends in one of these categories.

Success – The Elusive Top Spot

A software project is a Success when most or all of the initial vision and high hopes are met. Successful software realizes the potential that was envisioned. Successful software shows obvious and demonstrable improvement to the company. There is no debate when a successful software project is delivered and implemented, since a successful project is seen by all as irreplaceable. The thought of going back to the old way is inconceivable. Successful software is like magic; its benefits are difficult to exaggerate, the success difficult to explain.

Successful projects are listed on resumes, bragged about in meetings, and used to justify personal reward. Most importantly, a successful project means the company is able to do something it could never do in the past; faster, better, cheaper.

Workable – Common and Improvable

Software projects that fail to meet the Success criteria fall into 2 categories, Failure and Workable. The Workable system is really a type of failure, that is, a Failure to Succeed. Workable software is not a Success in that it did not meet its original vision, it is not obvious that the company is better off, and there are even a few people that grumble that the old way was better. Workable software is the bare minimum. It is better than before, but barely. Many people say that the benefits of a Workable system are a wash when compared to the previous system, some saying but with the costs factored in, it is worse. Some problems were solved, but others, new ones, were introduced. Many software projects, perhaps even most, are Workable systems. At the low end of the Workable range, a software project might be argued as a failure. At the high end of the range, some might argue that it is a Success.

Failure - A software project is a failure when after its delivery its benefits clearly and obviously do not outweigh their costs. A failed system is worse than the system it was to replace. A failed system is seen as a mistake; a problem that most believe must quickly or eventually be corrected. Many times failed software is never delivered or put into production. It might never have been permitted to be delivered. A failed software project is a heartbreak. It causes enormous pain and frustration. The vendor, developers, project manager, and management all lose credibility. Money has been “wasted”. People lose desire to ever tackle another development project. Blame seems to be everywhere.

Remediation versus Improvement

Remediation

Software Remediation is the process of moving a software project from Failure to Workable. That's it. It is about taking a system that was not completed or was not delivered and improving it enough to meet the Workable threshold, taking something that was thought to be dead and bringing it back to life. In some cases, a project can be so transformed that it becomes a true Success, but this is not the objective of Remediation. Because a Failure project has the highest exhibited cost/benefit ratio, that is, all cost and little benefit, a Failure project benefits the most from the execution of a proper Remediation effort, even if it only becomes Workable. Moving from Failure to Success is extra bonus.

Improvement

If your software project is Workable, it is not a candidate for Remediation, but rather the more traditional improvement or enhancement process. In this situation, the software is working, it is delivering some value, and does not need immediate or dramatic attention. This is not to say that companies are not well justified in spending resources to improve the system to Success. They may justifiably want to push the system to the next level, but the crisis and desperation feelings common to Failure systems do not exist. Less draconian or dramatic changes than employed in Remediation can be enough to push the software far enough along the continuum that it delivers more of what is needed at costs that are acceptable. Improvement requires the right vendor, developers, management, etc, but it does not need the Remediation process.

Assessing your Specific Situation

Accurately assessing where your software is on the Scale is important to knowing how and when to take action. Depending on whether a project is completed or not impacts the decision making process. Other factors such as costs, commitment, and technical capability are also important in deciding what if any action is required.

Once a project is completed and delivered, it is always obvious where it belongs on the Scale. A successful project is obvious. A Failed project is equally obvious. Hindsight provides this answer with clarity, with certainty.

This is not the case, however, when the project is in progress and appears to be failing. In the middle of the chaos and finger pointing of a failing project, it can be difficult or impossible to know which side of the Failure/Workable threshold it will end up on. There are many indicators that a project is not going well:

- excessive technical problems
- missed milestones
- cost overruns
- changing expectations
- increasing blame

During a potentially failing project, many organizations are able to mask these symptoms with the addition of manpower, schedule slipping, and increased management attention. While these can appear to “solve” the problem this “solution” can be a very costly mistake. Sinking more money and resources into a system that will ultimately fail is a poor decision. The trick is knowing if what appears to be a failing project will actually end up failing.

A solution around this problem is simply to assume that the project is going to fail and begin a Remediation effort immediately. From our experience, this is the hardest thing for companies to do. No one likes to admit that a project isn't going well. But it helps to think of Remediation from the context of a safety net. If the project is truly failing, Remediation will solve the problem and ensure that the software ends up in Workable, maybe even in Success. If the project would have ended up in Workable anyway, Remediation will have been a good insurance policy and will likely provide tangible improvements anyway. At the very least, if the Discovery process of the Remediation methodology identifies the project as having little chance of ever meeting Workable, the costs of Remediation will likely be less than the continual investment into a failed system. Remediation will at least offer a lower cost alternative when scrapping the system altogether.

Remediation is Not Always a Solution

Remediation is a process of Examination, Diagnosis, and Resolution. During the Examination and Diagnosis phases of Remediation, it may be found that a system cannot or should not be remediated. Not all failing or failed software systems can be saved. In 10 years of performing Remediation projects, we now estimate that in about 90% of all efforts we have been able to salvage all or most of the failing software and deliver either Workable or Successful systems. In about 10% of the situations, it was impossible or impractical to complete remediation. In these “failures to remediate”, we have identified 2 main software deficiency categories, Technical and End-use Integration, that contributed to this failure.

Technical Deficiency

Technical deficiencies stem from the inability to correct bad code, solve performance problems, or otherwise permit the software from operating in a manner that meets certain basic standards of use, reliability, integrity, security, or performance. Generally, lack of documentation, access to previous developers, or underlying technology used are not causes of remediation failure. The following factors have been shown to increase the likelihood of a remediation failure:

Missing Source Code – Significant missing source code, either from disappearance or incompleteness. Incomplete software can happen when a vendor failed to finish a project in the middle or when the budget overrun became too high and the company pulled the plug. In that case Remediation is not about fixing, but rather more about original development. Also, if a project is found to have little or no source code, because of vendor contractual issues or simply not knowing how to obtain the source code, little can be done to remediate.

Overly Complex Source Code - Software can fail when it has become overly complex and remediation can fail when the overly complex software cannot be untangled. If it is found that changing the code cannot be accomplished without the significant introduction of problems, it may be impractical to remediate. Overly complex code is usually the result of inexperienced developers. Good code is simple, elegant, and short. Sometimes it is not practical to make complex code into elegant code. In these cases, rewriting a simple application that meets the need may be easier and cheaper than stripping out the complexity in the existing code.

Wrong Use of Technology – A system built upon an incorrect database, language, or topology can prevent Remediation from succeeding. During the project, an overly complicated language or database architecture was selected and is now hampering effective operation. We see this most often when a vendor committed to a particular technology is selected instead of choosing a good vendor who can then choose the right technology. The vendor maybe excellent at writing software in that technology, but it might have been the wrong technology for your software. In this case, it might be cheaper to rewrite part or all of the application using the proper technology rather than attempt to correct an uncorrectable technology choice.

End Use Integration

While software is traditionally thought of as the combination of source code, database architecture, user interfaces, hardware, networks, and communications, it is equally about workflow, business process design, data integration, solving the right problem. In certain cases, poor execution of these elements can adversely affect the successful outcome of a software project, as well as the ability to Remediate.

Software Does Not Solve the Business Problem – Even if the software is well designed and executed, reliable and fast, if it does not solve the business problem that it was intended to, it will fail. Remediation may not work if the magnitude of this problem is too big, the corrections looking more like a full rewrite.

Improper or Non-Existent Data Integration – Most software systems work within an environment of other systems and databases. If software is built and deployed into an environment that needs connections to other systems or data from other databases and these connections have not been built or are not effective, the software will fail. If the magnitude of this problem is large enough, remediation will not be effective and the system may need to be rewritten.

Incorrect Type of User – Software is commonly used by people to solve problems. If a software product is written for one type of audience and the user community ends up sufficiently different, the software can fail. Examples of this might include software that requires a factory line operator to use a Windows computer to enter large amounts of text data, or creating a touch screen interface for a keyboard intensive application. In certain extreme cases, remediation will not be able to correct for this problem.

Remediation Methodology

Decision Design has developed a highly effective 3 part Remediation Methodology that was specifically designed for Remediation projects. It is based on the 4 elements of our Client Service Methodology, but with specialization of the Discovery, Design, and Delivery elements important to helping a failed or failing system.

The methodology includes examining the current software, identifying what is thought to be the problem, learning how the system works, and then identifying the actual problem. Learning about the previous vendor, technologies, company organization, and project management team is a key component, as is understanding the original vision, the problem the software is to solve, and the expectations the system has with users, stakeholders, and management. In the end, our Remediation Methodology is about delivering a system that meets the original vision and solves the original problem, with as seamless a transition as possible.

Discovery

Discovery is the first step in any of Decision Designs projects, whether new development or Remediation. Specific to remediation, Discovery is divided into 2 phases, Examination and Diagnosis, each playing an important role in learning what is wrong, what is right, and deciding if remediation can succeed.

Examination

Examination involves learning about the problem and assessing the projects location on the Success Scale. We analyze how the code was written, how the data is stored, and how decisions were made in the early stages of the failed or failing project, ascertaining how the software works, what the bottlenecks are and what the real problem to be solved is. We learn about the previous vendor, technologies, company organization, and project management team and emphasize understanding the original vision, the problem the software is to solve, and the expectations the system has with users, stakeholders, and management.

Finding the one or many sources of the problems is the most important part of Examination. Examination is a process that collects information, evaluates causes, talks to stakeholders, and finds missing information.

Diagnosis

Diagnosis is about drawing conclusions, making decisions, and communicating results to a the client. In this step we conclude what is wrong and what needs to be done to correct the software, providing careful cause and effect conclusions. Diagnosis is about making decisions and establishing a remediation plan.

At the end of Diagnosis, we provide the answer, that is, what went wrong, what can be done to fix it, and how the remediation effort will be executed. In the rare case that the problem cannot be fixed, options will be identified.

Discovery in Remediation is about focusing on the original expectations of the system and not about discussing the ways the project should have been conducted in the first place. Discovery is everything about keeping focus and not being distracted by unimportant technical or project management issues. It's not about finding blame and to pointing fingers. How software got to the state it is in is less important than the state itself – what is wrong.

At the end of Discovery, we will have a list of the changes that must be made that will overcome the problem. If the system has not been completed, it will include a plan for completion. If the system is broken or underperforming, it will include a list of actions that must be taken. In either case, at the end of Discovery, we will be able to quote the cost to resolve the problems and provide a schedule for doing so.

The next 2 steps are about executing and finishing the job.

Design

In Design, the Remediation plan is executed, bugs corrected, databases redesigned. Design is the process of implementing the changes that will solve the problem by planning the work, identifying the resources, and performing the tasks that will fix the software. Programming happens at this stage with an emphasis on showing results as quickly as possible to building the confidence of an anxious client. In Design, we take the previous software, make changes, and transform it into the software that it should have been in the first place, turning the ugly duckling into the swan.

Delivery

The Delivery process includes the testing and rollout of the corrected or newly completed system. The goal is finally realized. What had been hoped for a long time ago becomes real. When doing a Remediation project, special care is given in the Delivery process to account for previously failed expectations by end users. Either they have been waiting for long promised software that was to correct a problem or they have been living with software that was suppose to make things better but didn't. In Delivery, users finally see software they had been promised or they see dramatic changes from what they were frustrated with for so long. As with any software that Decision Design is involved with, Delivery is the best part because it is when all the hard work is realized. In Remediation projects Delivery is even sweeter because clients are coming from a place of initial discouragement. Unlike with new development projects, where expectations are high and the dream is alive and well, in Remediation, clients are often exhausted, exasperated and disillusioned with the original dream. Delivery is the point where the client now knows they are on the Workable to the Success side of the software success continuum.

Summary

Software Remediation may not be a household word, but it solves a most common problem. Whether you engage an outside vendor like Decision Design or choose to remediate in house, it must be performed with objectivity, timeliness, and with a commitment to a methodology that will specifically succeed at Remediation of the Software. At Decision Design, we are ready and able to tackle these difficult projects and can salvage your investment and realize your vision. Remediation is an essential tool to resolving the most difficult of problems and leveraging, not wasting your considerable investments.

Software does not have to fail.

