



PathPoint Software



The PathPoint Approach

A Breakthrough Productivity Tool To
Drive Down the Time and Cost For
Mainframe Application Projects

PathPoint Software, Inc.
101 North Plains Industrial Road
Wallingford, CT 06492
Phone 203-741-0400
www.pathpointsoftware.com

Projects Involving Legacy Applications

Legacy applications typically support the majority of any large company's core business, and they are crucial to the life of those organizations. As the business responds to competitive pressures and the changes in business climate, so must these applications be continually updated and enhanced. There are two broad categories of projects involving legacy applications, application maintenance and transformation projects.

Application Maintenance

Application maintenance involves responding to the day-to-day requests that are made to enhance legacy applications. Typically, these legacy applications must change in direct response to:

- Government-mandated regulatory requirements (e.g., HIPAA, T+1, SEC, etc.)
- Internal users of applications, who are continually making business requests for enhancements to keep up with the competition, incorporate new products, etc.
- External users, such as suppliers and customers, requiring access to an organization's application files.
- New technologies such as a new database technology.

Transformation projects

Transformation projects involve leveraging existing legacy applications as part of a strategy to improve the capabilities of the portfolio of applications in supporting the business. A transformation approach may be more appealing than a wholesale legacy application replacement strategy because it tends to be less costly and, perhaps more importantly, less risky. One example is an application consolidation project where two similar applications are to be combined to achieve various efficiencies. A merger or acquisition scenario would trigger this type of project. In order to achieve such a consolidation it is necessary to know how each application works to process the business it supports. If one application is to be merged with a second, it is necessary to know what logic is unique in the first and must be moved to the second. A second challenge is to figure out where in the second application to place this transported logic.

Another example is a legacy application replacement project where a particular application in the company's suite of applications needs to be replaced. In this case, in order to re-write the application, it is necessary to understand the business rules and the flow of logic in the application. Unfortunately the only place this information resides in most cases is in the legacy application code itself. The process to extract this information is typically resource intensive, time-consuming and error-prone.

Why Do Legacy Application Projects Require so Many Resources?

Legacy mainframe applications are typically older systems with little or no documentation. The original developers are long gone, and the applications have been changed over the years by waves of transient developers and consultants. What remains is application source code that contains all the business processing logic which has become voluminous and cryptic. A small application program can have 5,000 lines of code, and a large program can easily exceed 30,000 lines of code.

Logic is often duplicated in different programs, so changing logic in one program may require making the same change in other programs. As programs are modified, parts of the original code often become "dead" code that adds to the complexity of understanding how the program works. Understanding the logic flow of the source code becomes even a bigger problem when the source code is not well organized.

A practice used frequently by programmers who were, and are, responsible for making changes to these mainframe legacy applications is the process of "branching out," which involves writing code in an isolated part of the program and "branching" to it to perform the new business rules. This technique requires

programmers to do extensive regression testing to ensure, to the best of their ability, that the new branch does not adversely affect any portion of the core application functionality. This practice leads to even more disorganized code. As a result, simply locating the logic of interest is a tedious and time-consuming undertaking.

Over the years, massive amounts of branching and other “islands of code” have resulted in increasingly complex applications whose “spaghetti code” appearance makes understanding these applications a nightmare.

Application Maintenance Example

Implementing the Typical Business Request – Where is the Time Spent?

When a business request to modify a mainframe legacy application arrives at the application manager’s desk, a project is initiated, and a project leader is designated to determine the time and resources required to implement the change. Projects can range from 2-4 weeks for a small project, to 12-18+ months for larger projects

The difficulties of estimating resources and time arise when the application is not well understood. Just finding the programs that support the business transaction to be changed can be daunting, especially when the application consists of 2,000 or more programs.

Time spent on up-front Analysis

The Analysis phase includes determining the part of the legacy application that embodies the files and logic pertinent to the business transactions of interest, and understanding how it currently works. For example, if a business request requires changing the way online international orders are processed, it is necessary to find out what programs are involved and how the application is currently processing online international orders.

With little or no documentation to rely on and minimal subject matter expertise available, the team usually has to resort to studying source code line-by-line. Although there are tools that show what the source code looks like in different formats, they do not show program logic flow through the various modules as it pertains to a specific business transaction, e.g., processing an international order. Thus, analysts have to “play computer” and identify the “business paths” as they wind their way through the complex code. This effort is a time-consuming and error-prone process. As a result, this Analysis normally constitutes about 30-40% of the total project time.

Time spent fixing unanticipated problems in later phases of the project

Any omissions or errors in the Analysis phase can lead to unanticipated problems, typically found during the coding and testing phases. Some problems do not even show up until the project is finished, and the changed programs are in production.

When these downstream problems occur:

- 1) It is usually necessary to go back to the Analysis phase to figure out how to incorporate the “unforeseen” problem in the current design.
- 2) Code changes are then made and tested to verify that the problem is corrected; and
- 3) Significant regression testing is then done to ensure that the “fix” didn’t affect any other code.

As additional problems are uncovered, the original approach to resolving the business request can become compromised, and it becomes increasingly more difficult and time-consuming to go through the cycle of

analysis, code/test, and regression test to resolve each new problem. Problem resolution typically consumes a significant amount of time and resources for the project to be completed on time.

Estimating Project Resources and Costs

Application managers and project managers well know this scenario, and they are familiar with existing tools available to them as well as their shortfalls. Skilled applications managers who are able to accurately estimate the time and resources to complete both large and small-scale projects involving legacy applications usually accept, and plan for, contingencies such as unforeseen problems and repeated regression testing. This contingency often represents up to 50% of the total project budget. That is why implementing changes in legacy applications is a costly and time-consuming process. As a consequence of this practice, even an apparent small change is costly to do.

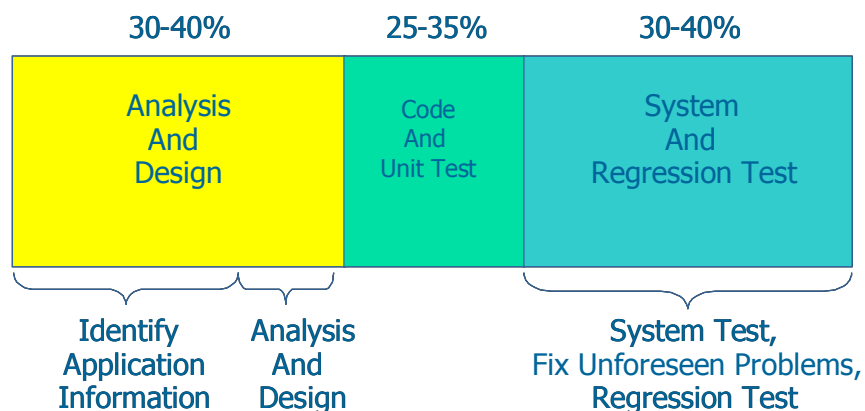
Transformation Example

One transformation project example outlined earlier describes where two companies have merged and there is a desire to consolidate two applications in order to reduce costs and increase consistency, for example, two claims applications. Once it is decided which claims application is the more robust and will be the survivor, there are two major analysis tasks that must be undertaken. The first is to determine what functionality from the application to be replaced does not exist in the selected application but is required. The second task is to determine where in the selected application to place the new logic required from the application to be replaced.

One way to figure out what functionality from the application to be replaced is required and does not exist in the surviving application is to run scripts for those transactions that process the required functionality in the application to be replaced and document the results. Then run those scripts customized for the surviving application and compare the results. Where there are material shortcomings in the logic of the surviving application based on the results produced, document them. Now you know what areas of logic need to be added.

Now comes the tricky part. The Analyst needs to figure out where to look in the application that will be replaced to gather the processing paths and business rules to be re-written. But in an application with hundreds or even thousands of modules this becomes a resource intensive, time consuming and error-prone task. Further, once this information has been extracted and documented, a similar dilemma occurs when the Analyst attempts to replace the logic in the surviving application. Where do I insert the new logic? How do I not disrupt the existing logic flow? Again, even using existing tools, this is a resource intensive, time consuming and error prone effort that will require a great deal of regression testing.

Time Allocation for Typical Application Maintenance Project



The PathPoint Approach

The examples cited in the white paper all have one common characteristic- they require the need to understand **how** legacy applications work to process business transactions from a business processing standpoint. When the user hits the enter key what is the logic path that is followed to complete that transaction? PathPoint is a collection tool for application analysts that targets this up-front analysis effort. It is unique in its ability to quickly and accurately capture application activity for selected business transactions as the application is processing them. The PathPoint value proposition is to significantly reduce this analysis time while simultaneously improving its accuracy, which reduces downstream problems that are more costly to resolve.

What PathPoint does

Unlike other tools that require the analyst to know which program modules to look into and where in the module to look, PathPoint collects application information for selected business transactions entered by the end user and executed by the application programs. This real-time approach to data collection quickly follows each business path (such as processing an online international order) as the application makes its way through the transactions initiated, the programs called and the I/O statements actually executed as they are issued in sequence. PathPoint also determines the activity, (i.e., create, read, update or delete), against files and base tables that occurred down to the field level.

PathPoint also captures the input and output screens involved so the analyst can clearly see what the application did to carry out the business transactions of interest. The old trial-and-error approach is replaced by clear, accurate visibility of how the application actually works for any given business transaction.

PathPoint reduces the time and cost for application analysis

The capability of PathPoint to quickly capture how the application processes each business transaction is extremely important in the Analysis phase of a project. PathPoint replaces the time-consuming, error-prone, source code approach to Analysis with a real-time collection solution. ***Analysts now can do in minutes what normally takes months of effort. This dramatically reduces both the time and resource requirements for application analysis, design and system testing, and increases the accuracy.***

PathPoint reduces and avoids problems in later phases

The ability to understand what the application is really doing to process each business transaction drastically reduces the number of unforeseen problems that were common before PathPoint was available. PathPoint serves as a roadmap or radar device to see through “spaghetti” code, avoid “dead” code and enable the analyst to identify problems early in the Analysis phase that would have been uncovered in the testing phase. ***Problems found early in the Analysis phase are a lot cheaper to resolve than the problems that occur in later phases.***

PathPoint enables better use of existing tools

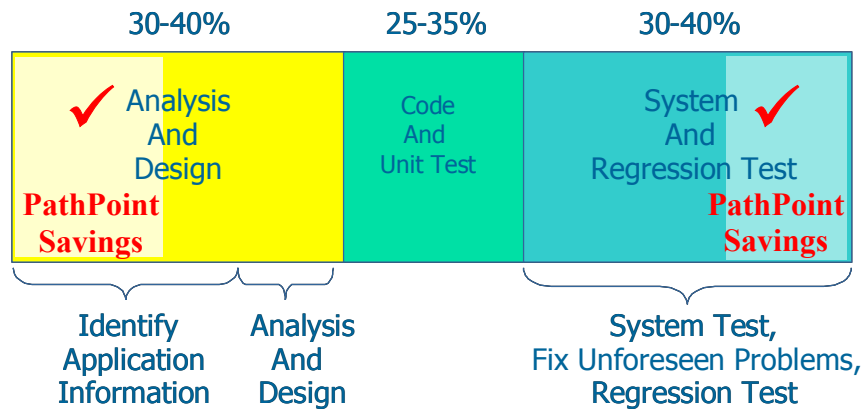
PathPoint is an **application** analysis productivity tool, while existing **program** analysis tools require that you already know which programs you are interested in and where in the programs you want to look. In an application that may have 2,000 programs, finding the right ones to drill down into for a particular business transaction can be daunting. PathPoint will identify the programs accessed by the transaction in question and show the call path that was followed through these programs. The analyst can follow the call path and then use tools such as Xpediter to drill down into the business rules at specific locations.

The End Result

PathPoint quickly shows how an application really works. This visibility makes it easy to perform better impact analysis, resulting in fewer unforeseen problems.

The bottom line: With PathPoint, IT mainframe application analysis time can be reduced by 50% or more.

The Value of PathPoint to Analysts



PathPoint® is a registered trademark of PathPoint Software, Inc.