Mainframe software development and maintenance
White paper
December 2009

Rational. software

IBM

Fine-tuning IBM z/OS software maintenance.

Strategies and tips for improving your maintenance experience

Jon Sayles, Rational developer for System z/enterprise modernization technical specialist, IBM Software Group

Contents

Executive summary

Decades have passed since the first business software was developed for the mainframe in your shop. While the hardware and software, platforms and paradigms, computing languages and methodologies have all changed and evolved in that time, many of those same applications may still be running your business. They represent assets of significant value, your company's intellectual property—automating, extending and encapsulating other assets of your business. Their book value, if viewed from the standpoint of income generated over time, is substantial. And software, like all corporate assets, requires reinvestment if its value is to be sustained. Investment enables continual improvement and growth in the applications that support your changing business model. Moreover, investment compounds the business value of your software, saving—and earning—you time and money. This reinvestment is called "maintenance."

> *"… [T]he majority of software costs are incurred during the period after the developed software is accepted. These costs are primarily due to software maintenance, which here refers both to activities to preserve the software's existing functionality and performance, and activities to increase its functionality and improve its performance throughout the life-cycle."*
>
> *— Barry Boehm, TRW Emeritus Professor of Software Engineering, Computer Science Department, University of Southern California*[1]

We have found that focusing on three areas of investment will help you fine-tune the maintenance of the applications that run your business:

- *Investment in people*
- *Investment in process*
- *Investment in tools*

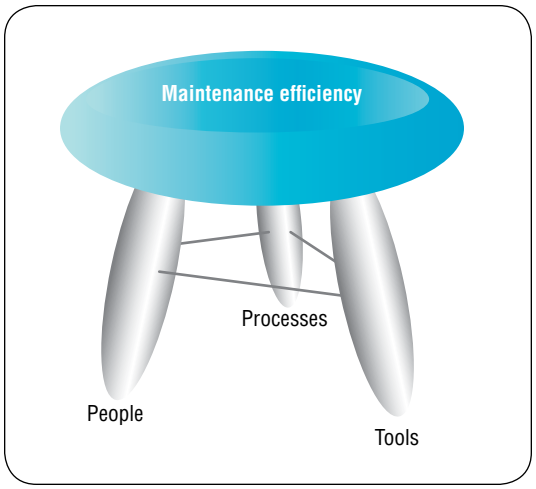

Figure 1: Three key organizational areas to renovate

Investment in people

Your business benefits by the quality of the work produced by your people. And the quality of the software maintenance they perform on your business applications correlates with the caliber of knowledge used every day by your software maintenance staff.

Key to success

Ensure that software maintenance staff has easy access to application domain knowledge.

Application domain knowledge provides an understanding of how software intersects with your business. To some degree this is vertical industry knowledge, but it also includes the unique business rules and processes that differentiate your corporation from competitors in the marketplace. The trustees of application domain knowledge are the business users. Ensuring that the software maintenance staff's analysis needs for authoritative application domain knowledge are satisfied is a fine-tuning key to success.

Key to success

Provide software maintenance personnel with access to premier technical learning.

Technical knowledge provides deep mastery of the complete technology environment underlying your application software stack as well as the specific maintenance tools and processes employed in your shop. Providing maintenance teams with premier, first-class—and often no-charge—technical learning opportunities is a fine-tuning key to success.

Application semantics refers to the custom code that realizes the unique business rules and idiosyncratic in-house processes that keep your company running. In most shops, the original developers of applications have long since moved on. Providing current maintenance teams with static and dynamic analysis tools that simplify the problem of intellectually grasping application semantics is a fine-tuning key to success.

Key to success

Automate software maintenance tasks for:

- *Application understanding/impact analysis.*
- *Dynamic code analysis and testing.*

Is increasing the caliber of knowledge of the software maintenance staff really necessary? No, it's not. But just as doctors take the trouble to learn about new medical advances and auto mechanics learn how to use new computerized diagnostic tools, keeping your people current on their knowledge of software maintenance best practices helps them keep applications producing for the company.

Investment in process

A methodology or even multiple methodologies for distinct categories of maintenance that address how to tackle the diverse and complex issues presented by IBM z/OS® software maintenance in a systematic way are critical investments. Yet at many companies, formalized processes for maintaining applications don't exist. We might find documentation, informal "cookbooks" and other collateral based on lessons learned from doing certain aspects of maintenance work. But systematic, formal software maintenance methodologies for handling adaptive maintenance are rare.

Key to success

Establish a position overseeing software maintenance.

Investment in correcting, adapting and perfecting your maintenance processes by adopting a formal lifecycle—and revising it with best practices lessons learned—is a fine-tuning key to success.

Appointing a single person to be the software maintenance architect at your shop and tasking that person with responsibility for establishing organizational controls—developing and refining the maintenance plan, etc.—is another fine-tuning key to success.

Investment in tools

As fate would have it, the software stacks running on and attached to your IBM z/OS platform have evolved and grown in size and complexity just as corporate cost cutting has inversely reduced available staff resources and maintenance time. From a technology perspective, tools are no longer nice to have—they have become a necessity. Using the same manual processes and approaches throughout the maintenance lifecycle that would have been acceptable 10, 20 or even 30 years ago no longer provides the efficiencies demanded by constrained resources.

Key to success

Employ key software maintenance tools for:

- *Team collaboration and source code management.*
- *IDE-based development.*

Fortunately, the software industry has moved beyond time sharing option–based and manual approaches to maintaining and supporting cross-platform or even complex single-platform applications. Mature and cost-effective solutions are available in the following areas:

- *Integrated development environment (IDE)—**IBM Rational Developer for System z® software***
- *Application understanding/impact analysis—**IBM Rational Asset Analyzer software***
- *Testing and dynamic code analysis—**IBM Problem Determination Tools for z/OS technology***
- *Team collaboration—**IBM Rational Team Concert™ software***
- *Source control management—**IBM Rational ClearCase® software***

Using a modern IDE, collaboration, source code management (SCM) and automated analysis tools are fine-tuning keys to success.

Outsourced maintenance

When taking a deeper look at the maintenance being done to the software assets that run your business, we should note the trend over the last five years toward outsourcing maintenance. Outsourcing presents a number of new challenges: distance, or latency; communication, or compounded problems as a result of misunderstanding and misinterpretation of complex business language; and loss of control over the maintenance practices of your outsourcer. These can be exacerbated by the loss of in-house applications or business expertise.

Overcoming these additional obstacles requires tightening down on software maintenance projects through increased project governance and adoption of technology that automates project scoping and offers real-time, collaborative team maintenance lifecycle progression.

Key to success

Employ a software maintenance methodology that takes into consideration the different categories of software maintenance.

Fine-tuning z/OS maintenance

To understand the nuances of z/OS software maintenance, let's begin by more precisely defining terms and vocabulary. Table 1 presents the software categories defined by the Institute of Electrical and Electronics Engineers (IEEE)[2] and places them in a matrix of unscheduled/scheduled and reactive/proactive maintenance.

	Unscheduled	Scheduled
Reactive	Emergency maintenance: unscheduled corrective maintenance performed to keep a system operational	Corrective maintenance: reactive modification of a software product performed after delivery to correct discovered faults Adaptive maintenance: modification of a software product performed after delivery to keep a computer program usable in a changed or changing environment
Proactive		Perfective maintenance: modification of a software product performed after delivery to improve software quality, performance or maintainability

Table 1: The software maintenance matrix (software category definitions © 1993 IEEE)

People

Who are the people at your shop who do maintenance? In our work with large z/OS shops over the past 10 years, we have looked to identify staff categories available for maintenance. While an informal effort, our results are listed in table 2.

Maintenance team organization	Used by percentage of shops
Developers who wrote the code maintain it (primarily)	7
Separate maintenance in-house team	16
Local consultants	8
Offshore/outsourced maintenance	40
Other (various combinations of the above)	29

Table 2: Software maintenance team breakdown

The first thing we noticed is that roughly 60 percent of the world's software maintenance is done by external development teams, not by a shop's own technical staff. How does that affect the maintenance challenge?

Application domain knowledge

Let's look at the importance of the business and programmer analyst. In the beginning, there were business users, systems analysts and coders—and problems. As the systems analysts were overwhelmed in translating business requirements to procedural specifications, they—and the business-process knowledge they acquired—became a software bottleneck.

The solution that evolved was that the coders inhabiting various shops became "programmer/analysts." Programmer/analysts interfaced with users and shared design responsibilities with systems analysts. The bottleneck eased. And as they accumulated business knowledge, over the years their work became even more valuable and increased in quality.

Key to success

Nurture and retain those with deep technical business analysis knowledge of your current application stacks.

In many respects, the current situation with external development teams is a throwback to the genesis of business systems development, with a split between coders and business/systems analysts. A key to success for your company is to, at all costs, nurture and retain those on staff who have deep technical business analysis knowledge of how your current application stack operates—because that intellectual capital may be the most valuable software asset you have.

Highlights

Technical knowledge

Ensure that your own in-house staff has open access to technical learning content and expertise—especially any of the groups dedicated to emergency maintenance. Start by establishing a set of links to the kinds of quality materials that are freely available on the Internet, including the IBM Redbooks® library, language manuals and other collections of technical articles. Finally, it's important to note that an opportunity for saving time and money exists in educating maintenance developers well in traditional z/OS platform–specific areas of highest impact.

Static and dynamic analysis tools are useful for in-house teams but critical when outsourcing maintenance.

Application semantics

The breakdown in table 2 brings this key factor into focus:

- *For in-house maintenance done by developers who developed the original code, application semantics should not be an issue. And for maintenance teams from within your organization, including local consultants, application semantics will be a straightforward learn.*
- *For externally-supported maintenance, using static and dynamic analysis tools is critical, and not just by lead project analysts during the analysis and scoping phases of a project. It is critical to have these tools available during the technical construction and testing phases of projects. It's useful to note that in-house teams also benefit from static and dynamic analysis tools.*

Highlights

A typical process flow organizes the software maintenance process in seven steps.

Process

A typical process flow has been captured by IEEE in IEEE Std 1219-1993[3] and is adapted below in figure 2. It organizes the software maintenance process or lifecycle in seven steps. You may recognize some or all of these steps as occurring in your shop's approach to software. Or you may have similar, fewer or more steps depending on the formality of your methodology and how it is applied by different teams.

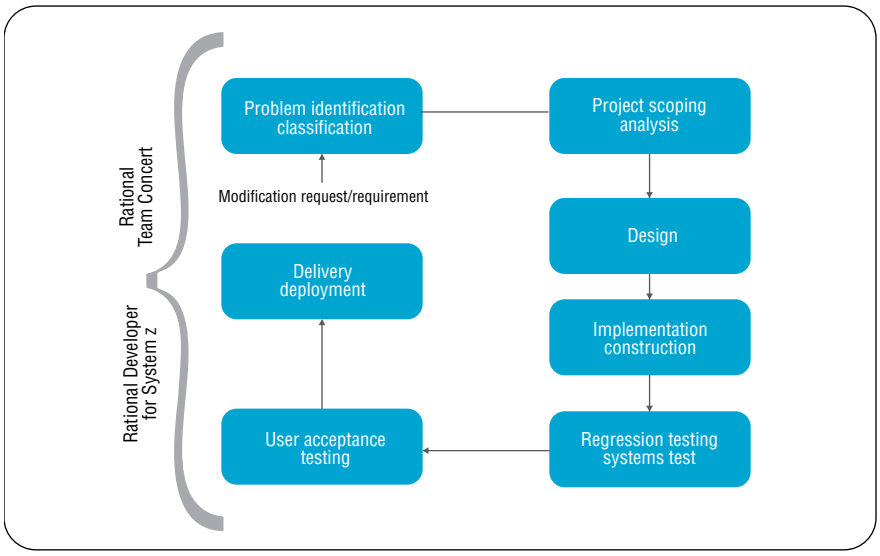

Figure 2: The IEEE software maintenance process has been adapted to show critical subprocesses (process flow © 1993 IEEE).

Highlights

Keep in mind that standard maintenance processes are significantly affected by unique needs and preferences.

However, keep in mind:

- *As described earlier, it is the experience of most of the shops that the needs of each category of maintenance (emergency, corrective, adaptive and perfective) expand out to additional subprocesses for most of the steps in the IEEE process (see table 2).*
- *Your shop's own experience, priorities and prerogatives, even your approach to implementing methodologies, always supersede a generic, one-size-fits-all approach.*
- *Working toward an explicitly formalized methodology might be an exercise that will not provide the return on your investment you anticipate. It must be considered in the context of your shop's inclination toward results-based, versus process-driven, activities and project management.*

Tools and steps in the process

When considering the required, optional and effective use of tools throughout the software maintenance lifecycle, consider that there are two categories:

- ***Tools that bring value to each and every step*** *—including cross-platform collaboration products and source control management tools*
- ***Tools that have step-specific advantages*** *—including the static and dynamic analysis tools described below*

Maintenance tools need to adapt to new technologies and new problems.

Your shop might create its own maintenance-tools-process-driven matrix—which will change over time—as new technologies and new problem domains appear on the landscape. However, a typical correlation of software maintenance lifecycle phases and tools that can help fine-tune your maintenance projects is shown in table 3.

Highlights

	Tools used at specific lifecycle steps	Tools used throughout the lifecycle
Problem/modification identification, classification and prioritization	**Dynamic testing tools** for initial breakdown of emergency and corrective maintenance problem definition: Problem Determination Tools for z/OS software	Rational ClearCase z/OS Extensions software Rational Team Concert software Rational Developer for System z software Rational Asset Analyzer software
Project scoping analysis	**Dynamic testing and static source code analysis tools** for trustworthy/accurate results and efficient/cost-effective analysis activities: Problem Determination Tools for z/OS and Rational Asset Analyzer software	
Design	n/a	
Implementation Construction	**Static source code analysis tools** for solving inconsistencies in developer understanding: Rational Asset Analyzer software	
Regression testing System testing	**Dynamic testing tools** for revealing problems during test and debugging: Problem Determination Tools for z/OS software **Static source code analysis tools** for solving problems stemming from an inconsistent developer understanding: Rational Asset Analyzer software	
Acceptance testing	**Dynamic testing tools:** Problem Determination Tools for z/OS software	
Delivery	n/a	

Table 3: Typical lifecycle phases and tool usage correlation

Tool selection

The first consideration that must be addressed is, "Why tools?" And the second is, "Which tools?"

IBM offers tools to be used at specific stages of the lifecycle, as well as tools to be used throughout the lifecycle.

For decades, software developers cogitated over listings, inserting paper clips and sticky pads as bookmarks, or they used manual Interactive System Productivity Facility (ISPF) search operations from IBM, which slow analysis down to manual typing speed. Perhaps your developers still practice these ancient rituals. Let's discuss the value proposition for using the four categories of tools shown in table 3 and discuss the must-have features that offer enough value for you to invest in and use them.

Highlights

Look for the must-have features that can justify tool investment.

Modern IDE

IBM initially introduced green-screen tools for edit, compile and debug to build the first generation of z/OS applications. And they did an admirable job, as evidenced by the billions of lines of COBOL, PL/I and even HLASM application source code still running your production workloads. But the needs of today's optimized maintenance demand facilities not intrinsic to develop-from-scratch projects, including deep language-editor semantics, fluent navigation, extended source content real estate and especially the benefits of deeply integrated analysis/edit/compile/debug. These features, all of which are available within Rational Developer for System z software, create an unlevel playing field compared to tools that were state of the art nearly four decades ago.

Source code management tools

This is the easy subcategory. Given the size, complexity and dynamic nature of modern z/OS development scenarios, especially with the introduction of offshore development, it is simply axiomatic that without a robust SCM product that automates the version control process—and offers revision management, file locking and the ability to quickly return to previous versions and merge development and maintenance source code—deltas would quickly grind to a halt.

Team collaboration tools

Unlike traditional source code management, you may not yet have been introduced to team collaboration software such as IBM Rational Team Concert software, which is essentially technology that:

- *Allows project leaders to define, organize and track your software maintenance, support and development teams and their projects (not program source, people and processes).*
- *Provides simplified interaction and communications between subject matter experts (SMEs) and the technical programmer/analysts and architects who interact with them. This interaction can be captured electronically for reuse, refinement and documentation.*
- *Accommodates management reporting and governance of project milestones and deliverables.*
- *Enables project technical developers to share development artifacts (source and model-based) in the context of doing project work.*

Highlights

Revisiting our previous points on challenges introduced by externally-supported maintenance projects, team collaboration was born to solve the latency, communications and logistics issues that are part and parcel of split project efforts. Distributed development versus central development presents organizational challenges that must be addressed to gain the understood benefits.

Static and dynamic code analysis tools

Analysis tools (both static and dynamic) assist developers in "gaining intellectual control" over an application, as Larry England from the IBM Santa Theresa Lab puts it.

Analysis tools can help you gain intellectual control over an application.

Research has shown that there is a *diseconomy* of scale in productivity when maintaining large systems. This is a result of the large number (often in the hundreds to thousands) of variables and "functions of interest" scattered throughout large amounts (often in the hundreds of thousands to millions of lines) of software.

Historically only expert z/OS developers have been capable of building exact internal/procedural models of the code execution paths, key variables and variable state transitions. Over the past 20 years, many of these individuals have retired or moved on to management positions, and now many maintenance teams are in the untenable position of using relatively inexperienced developers to make technical judgment calls on changes to core systems—systems that have only gotten older, more complex, brittle and more difficult to fathom than ever before.

Static code analysis

One of the unintended consequences of Y2K was the birth of static code analysis tools. Passing up the opportunity to employ what worked for Y2K means missing the benefits of automated, comprehensive and electronically precise static analysis tools, increasing risk, and spending considerable and additional time and money.

Features of static code analysis tools

The information needs of maintenance and support staff vary by the type of application learning/comprehension model employed. It is commonly understood that you learn a business application by combining top-down and bottom-up study. Top-down study consists of reading documentation, talking with SMEs, etc. Bottom-up study starts with reading code in specific functions, often working from program listing cross-references and tracing backward and upward toward higher levels of abstraction.

Highlights

Automating and simplifying analysis can boost team productivity and lower risk.

Tools that automate or simplify analysis top-down and bottom-up will fast-track your teams to improved productivity and lowered risk. Such tools should at a minimum contain the functionality described here—and come with a graphical (mouse-based) IDE—largely because the task of understanding code is keyboard-less, and driving through diagrams and hyperlinks with a mouse instead of typing FIND commands into green-screen panels is actually a critical element of the usage model.

Dynamic code analysis

For certain problems, static code analysis can be an indispensable aid in helping to solve many classes of software maintenance. This is especially true for some in the emergency-corrective maintenance category where you need another view of the production application—a view that is provided by actually running the live code—at the source level and following the execution sequence. This is called dynamic code analysis.

Dynamic code analysis tools such as the IBM Debug Tool allow you to inspect the following at the detail level without assumption:

- ***The instruction sequence***—*every statement is stepped through and animated running against the software stack that is the deployment platform*
- ***Variable values***—*in program storage, file buffers, database records temporary storage: anything to which the code has addressability*

This type of real-life monitoring at the source code level (which is the logical currency developers trade in) is one of the best methods for solving some of the most obstinate and bewildering software analysis problems. A table that lists some of the more prominent and useful dynamic code analysis features is available here.

Dynamic code analysis provides many more advantages beyond just debugging.

You probably own a product, like the IBM Debug Tool, that can be used for dynamic code analysis. But are you taking advantage of these tools for dynamic code analysis—and not just debugging? The table's feature/function column lists some of the ways that dynamic code analysis can provide teams with automated assistance in areas that are traditionally labor intensive and error prone because of the size and scale of complexity that concerns z/OS legacy systems.

How to benefit from software maintenance

Software is like other knowledge-based research and development products: an intangible corporate asset. The value of intangibles is based on the income they are expected to generate in the future.

It is naïve to think that organizations have buckets of money to throw at the dilemma of software maintenance; however, in this white paper we have attempted to provide solutions that incorporate:

- *Internal process reengineering.*
- *Web-based knowledge acquisition.*
- *Modest investment—or repurposing of development software with definitive maintenance value.*

Key to success

Endorse the business value of your production software and its maintenance as a capital asset across your organization.

Implementing one or all of these solutions will help you see:

- ***Fewer defects**, which lowers costs and raises the overall quality of your data center operations.*
- ***Improved productivity**, which can help balance the demand for changes and queued enhancements with new systems functionality.*
- ***Savings in hardware and software costs**, through more effective and productive developer usage models.*
- ***An increase in the shelf life (and respective business value) of your z/OS applications**, a business value that, unlike other intangibles, grows over time.*

Keys to success

With the software maintenance practices available today, there's no need to let your z/OS legacy applications flatline. It doesn't require a diagnostic genius to make things better, just IT leadership that appreciates the value of "systems that work."

- *Endorse the business value of your production software and its maintenance as a capital asset across your organization.*
- *Employ a software maintenance methodology that takes into consideration the different categories of software maintenance.*
- *Establish a position overseeing software maintenance.*

- *Ensure that software maintenance staff has easy access to application domain knowledge.*
- *Provide software maintenance personnel with access to premier technical learning.*
- *Employ key software maintenance tools for:*
  - *Application understanding/impact analysis.*
  - *Dynamic code analysis and testing.*
  - *Team collaboration.*
  - *Source control management.*

For more information

To learn more about software maintenance for the IBM System z® operating system, contact your IBM representative or access these additional resources:

- *Collections of technical articles are available at the COBOL Café at* **ibm.com**/software/rational/cafe/docs/DOC-3024
- *Rational Developer for System z software:* **ibm.com**/software/awdtools/rdz/
- *Rational Asset Analyzer software:* **ibm.com**/software/awdtools/raa/
- *Problem Determination Tools for z/OS technology:* **ibm.com**/software/awdtools/deployment/
- *Rational Team Concert software:* **ibm.com**/software/awdtools/rtc/
- *Rational ClearCase software:* **ibm.com**/software/awdtools/clearcase/

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
December 2009

---

1 Barry W. Boehm, *Software Engineering Economics*, (Prentice-Hall PTR, 1981), 533.

2,3 The Institute of Electrical and Electronics Engineers, *IEEE Standard for Software Maintenance*, IEEE Std 1219-1993, 1993.

RAW14180-USEN-00