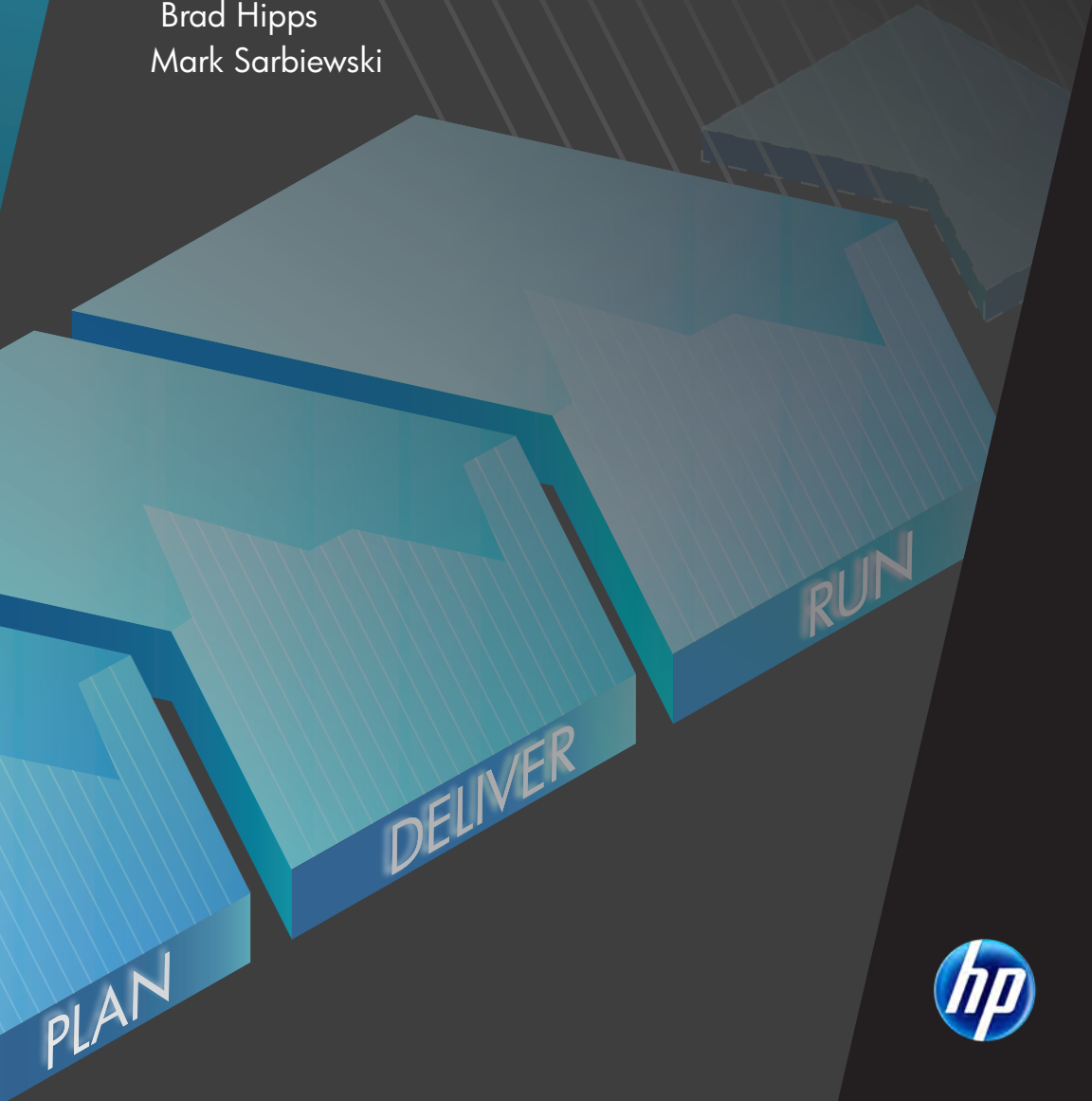
E-Book Edition

The Applications Handbook

A Guide to Mastering the Modern Application Lifecycle

Brad Hipps
Mark Sarbiewski



The Applications Handbook

A Guide to Mastering the Modern Application Lifecycle

Brad Hipps

Mark Sarbiewski

Contributors: Roger Barlow, Irina Carrel, Manu Chadha, Tracy Dedore, Kelly Emo, Colin Fletcher, Lou Harvey, Priya Kothari, Sarah Macaluso, Genefa Murphy, Silvia Siqueira

An HP Publication

3000 Hanover Street, Palo Alto, CA 94304-1185 USA

First Edition

Printed in the United States

Edited by Beverly Hanly

Contents

Introduction

Mastering the Modern Application Lifecycle

Strange but true: The modern enterprise is built from widgets and bits. So thoroughly have applications become the means to operate the business, it's easy to forget how different the world was only a decade ago.

In the not-so-distant past, even large companies relied on a small handful of "super apps" to power the major business processes (payroll, accounts receivable, etc.). Not anymore. Most of us would have trouble naming a single business process that doesn't depend on an application or series of applications—from the tiniest wiki to the grandest enterprise resource planning system.

In fact, applications are so integral to the modern enterprise that business nimbleness depends on application agility: You don't get one without the other. The business moves, changes, and expands only as fast and efficiently as its applications do.

Recognizing this, IT organizations have fundamentally changed the ways their people, processes, and technologies deliver applications. We're seeing projects with offshore or near-shore support structures; a process shift away from document-heavy methods like Waterfall to flexible ones like Agile; and new technologies such as richer clients (RIA), cloud com-

puting, and service-based architectures. These initiatives characterize modern application delivery.

What is driving these trends? HP commissioned Forrester Research to survey 206 global IT decision makers to rank today's primary business drivers, and here's the response:

- Increased agility
- Innovation
- Cost reduction
- Functional enhancement
- Richer customer experience[1]

So the characteristics and objectives of modern delivery are clear. But how have they impacted Application teams?

No Free Lunch: The Challenges of Modern Delivery

Few things worth doing are easy, and modern delivery is no exception. For each initiative to reduce cost or increase agility, an equal and opposing force often rises up.

1. *Clearing Your Path to Modern Applications and Business Agility*, Forrester Research, April 6, 2010. A Forrester Consulting thought leadership paper commissioned by Hewlett Packard.

Table 1 shows a sampling of the key aims and typical challenges of these modern trends.

	Trend	Business objective	IT challenge
People	Distributed	+ Lower global cost structures + Flexible staffing + Continuous productivity	- Program management complexity - Who owns what? - Deeper silos
Process	Flexible	+ Greater quality + Lower project risk + Improved response	- Emphasis on speed over quality - Inconsistent results - Diminished thoroughness
Technology	Composite, RIA, cloud	+ Quicker development + Richer user experience + Economies of scale, lower TCO	- Loss of standards - Security, performance - Interdependency

Table 1: Trends, aims, challenges of modern delivery

The modern application is often built from a combination of new and existing subcomponents. It's assembled by geographically distributed teams, perhaps using new or unfamiliar technologies and almost certainly using different methodologies: Agile, iterative, sequential, or none at all. The impact of this mix on people, processes, and technologies means complexity is layered on complexity.

The hard truth is that few organizations marshal a good response to these complexities. The complications tend to creep in incrementally (a pilot outsource arrangement here, an exper-

iment in new technology there) rather than exploding all at once.

This drip-by-drip approach means no single increment feels like a game-changer. As a result, the organization doesn't see a need to rethink the legacy mechanisms they've long used for managing and executing application delivery—spreadsheets, emails, document templates, heroic work efforts, etc.

While the aim is modern delivery, the means are often anything but modern. Rather than achieving greater agility, lower costs, and higher quality, these organizations experience the opposite: Rampant complexity overwhelms the operational status quo.

A Modern Approach to Delivery

World-class application organizations successfully weave two interrelated principles:

- *A commitment to enhanced management and automation*: Modern delivery warrants modern capabilities.
- *A comprehension of the true nature of the application lifecycle*: The traditional Systems Development Life Cycle (SDLC) does not represent the full life of an application; rather it is the core of a broader, more complete application lifecycle.

Why are these two principles important? We'll return to them throughout the book, but let's examine them briefly here.

Unifying Management and Automation

An effective application lifecycle—one that consistently delivers end value to the business—requires integrating strong management and automation capabilities. Management capabilities that show where and how some action is required should sponsor an automated response with results feeding seamlessly back into the management picture.

This notion of integrated management and automation is already established in disciplines such as mechanical engineering. Planes and cars have cockpits and dashboards that indicate the status and behavior of the vehicle. These trigger corrective actions through dependent automated systems (more fuel, a different angle to the rudders) the results of which (greater speed, revised direction) are instantaneously reflected in the cockpit's management console.

Software engineering has not made as much progress integrating management and automation capabilities. Part of this is a result of inertia: IT organizations grow comfortable with legacy practices and point tools, even as the systems and methods of delivery grow more complex. But also, vendors have had difficulty creating integrated solutions.

In the chapters that follow, we suggest ways companies can take advantage of unified automation and management.

Mastering the True Application Lifecycle, From Core to Complete

Clearly, the work from requirements through deployment—what HP terms the *core lifecycle*—is critical. Failure here means the application never comes into being, or it does so with poor quality and at wildly inflated costs.

However, too often companies assume that by controlling the core lifecycle, everything else falls into place. This approach overlooks all that happens before requirements and after initial deployment: portfolio planning, establishing enterprise technical policy and governance, implementing operational responsiveness and tuning, managing change, and finally retiring the application.

In other words, this approach ignores the *complete lifecycle*, that cradle-to-grave context whose effective execution is the true challenge of any application's success.

By focusing on core delivery to the exclusion of the complete lifecycle, organizations miss the vast majority of an application's total cost of ownership (TCO).

> A recent Gartner study estimates that 92% of an application's cost occurs after the initial delivery![2]

Thinking of the complete lifecycle encourages delivery teams to understand the pivotal role they play in an application.

While the initial delivery cost of an application may be small relative to its TCO over a multiyear period, decisions made in the initial delivery—architectural and design decisions, test investments (or lack thereof)—will have lasting impacts on total cost to the business. It will also affect the ability to change or adapt the application in response to business needs.

All this may sound fine in the abstract, but what does it mean for the way IT organizations work? By embracing the perspective of the complete lifecycle, we can discover some answers.

As you look at the table of contents, you'll notice that the chapters reflect this complete lifecycle view. We begin with portfolio planning and management and conclude with application retirement, which is a phase of the application lifecycle that is frequently completely overlooked.

Putting It All Together

When we speak of integrated management and automation, we refer to capabilities that are necessary across the complete lifecycle.

---

2. *A Framework for the Lifetime Total Cost of Ownership of an Application*, Gartner Research, March 30, 2010.

Table 2 shows some legacy issues that management and automation can address in the core and complete lifecycle.

	Management (Analytics, workflow, dashboard, and visibility	**Automation** (Task automation and asset reuse)
Core lifecycle	■ No single view of progress exists across multiple, geographically distributed teams. ■ No central view of available project assets (requirements, components, tests, known defects) means much reinvention of wheels. ■ A mix of team methods (iterative, Agile, Waterfall) frustrates use of common tools and practices.	■ Lack of traceability among dependent artifacts means change-impact analysis is cumbersome and manual, resulting in slower response time to the business and less thorough vetting of impact. ■ Manual tests take too long to execute, resulting in critical project latency and shortcuts that hurt application quality.
Complete lifecycle	■ Lack of integrated change management results in regular "surprise" requests (enhancements, fixes) to the application delivery teams. ■ Portfolio management lacks real-time status information from active projects, and no view of underutilized applications, resulting in a misallocation of budgets.	■ Known application defects are submitted from the Help desk, resulting in redundant change requests to the delivery teams. ■ Apps and Ops teams adopt different, redundant approaches for preventing, diagnosing, and resolving application performance problems.

Table 2: Effective management and automation in the application lifecycle address some challenges of modernization

Unified management and automation, and viewing the core lifecycle in context of the complete lifecycle, form a rubric for

evaluating how effective organizations meet the challenges of modernization.

Of course, the true proof of success is in better business outcomes. As we explore practices and capabilities in each chapter, we'll keep in mind the business objectives of modernization. Any recommended practice, any new capability, must further at least one of these objectives:

- Agility
- Innovation
- Cost reduction
- Enhanced application experience

A Parting Metaphor

Some years ago, the U.S. Air Force discovered it had reached an odd tipping point. The capabilities of the latest fighter jets, and the switches, gears, and knobs required to manage them, had become too numerous and complex for pilots to control. In the split seconds of aerial combat, even the most seasoned veterans were unable to use more than a portion of what their planes had been designed to do.

The answer to this problem was to digitize the cockpit: Management and automation systems eliminated critical latencies in decision-making and execution.

Luckily, the stakes of application delivery aren't quite as high. But fundamentally our challenge is the same. The kinds of applications the enterprise now expects, and the means and methods by which these applications are created, have outstripped many of the traditional approaches to delivery.

Distributed teams working from distributed components, according to varying methodologies, simply cannot function without unifying management capabilities for sharing assets, reporting progress, and quantifying the overall state of the program.

The scale and urgency of most delivery efforts mandates automation. Change-impact analysis, requirement traceability, workflow management, and test execution are no longer activities that can be left to point tools, phone calls, emails, herculean all-hands efforts, and the like.

"Jet speed" may never apply to the task of application delivery. But we hope something in the following pages will mean at least clearer flying and better landings for your organization.

Chapter 1

PLAN: Building the Right Applications in the Right Way

Key Questions:

- Which application projects are we currently investing in, and what's being spent on each?
- Is our investment optimized across the portfolio? Where and how might we shift resources among projects?
- Are our business policies and priorities aligned with our technical policies and priorities?
- How do we ensure that our technical standards are understood and applied the same way across the enterprise?

Applications don't start as a set of system requirements; they start as a business idea. The company wants to try a market innovation or process improvement, and this effort becomes the reason to change or create an application, or more often, a series of applications.

First, examine your priorities, as you do with any business investment.

Because applications can't be seen or touched like other business investments such as real estate or equipment, they're often

not subject to the same vetting. Applications do not undergo the same dollar-for-dollar comparisons as other kinds of asset expenditures.

For many companies, the full content of the application portfolio is a mystery.

Enterprise architects face an equivalent complexity. While the business decides its portfolio priorities, architects have to establish a set of aligned technical policies: those standards and practices that must be true for every application project. Architecture and policy governance is deciding, communicating, and making sure everyone follows the company's technical policies. It is one of the great challenges of the modern enterprise.

In this chapter, we look at:

- Portfolio planning: the way an enterprise decides how to build the right application
- Technical policy governance: how it goes about building the right application in the right way

Begin at the Beginning: Evaluate the Business Objective

The world of applications is undergoing initiatives that touch every aspect of IT: people, processes, and technologies.

Key among the impacts are the complexities of portfolio management—seeing the applications that exist, and defining and measuring their business performance against incoming candidate investments. That's on top of managing more frequent releases, parallel development cycles, distributed teams, and (as ever) ongoing constraints on IT resources and budget.

When we consider the complete application lifecycle, we find that starting right—that is, with a quantifiably justified business investment, to be measured over time—makes all the difference for business outcome.

The aim of this section is to investigate how a new approach to project and portfolio management can help ensure that an enterprise's overall modernization efforts yield better business outcomes.

Portfolio Management: Moving to an Outcome-Oriented Approach

At the top of the chapter, we outlined a few questions nobody in the enterprise wants to be asked. Each question likely involves polling different teams across the organization, manually tabulating the results, and aggregating them into something that approximates the composite view.

Understandably, many enterprises choose to proceed by feel and guesswork, and often let such forces as political infighting settle competitive investments.

Clearly this isn't a repeatable, sustainable way of managing application investment.

The right approach begins with a clear, rationalized, and universal view of demand. This approach sounds rudimentary, but we find that many business requests are only informally documented, if at all. Instead demands are revealed in accidental, haphazard fashion—and often not until IT has begun to scream from overload.

Figure 1: Imagine this happening all over the enterprise

When demand requests are centralized in a single web-based solution, each request type follows a standardized workflow that specifies the process for reviewing, evaluating, prioritizing, scheduling, and approving the request.

See the Madness, Impose a Method

As requests for services and support come in, a process must be in place to prioritize and refine them. You can't make decisions about any individual request in a vacuum. You have to compare requests against other demands in the current quarter and in future periods.

Effective demand management allows the organization to consolidate and prioritize all application requests, including strategic application projects and operational activities. World-class organizations rely on strong demand management to ensure maximum focus and throughput of the IT organization.

Without such control of demand, IT departments—and business initiatives—tend to die the death of a thousand cuts. Fewer things are seen to completion because every day brings a new slew of requests, urgencies, and enhancements.

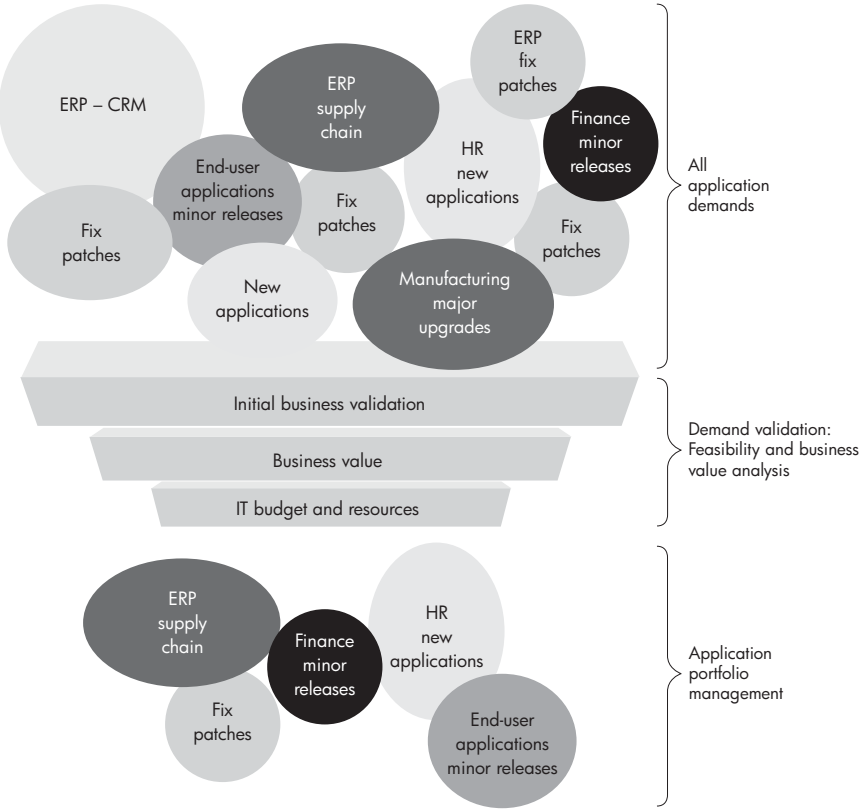


Figure 2: Winnow demand according to business priority and feasibility

It's important to understand that good demand management relies on automation. Preconfigured workflow and business rules means a standard evaluation and approval process that you can visually model, enforce, and measure.

> Businesses that automate the demand lifecycle often reduce the time it takes to resolve requests by more than 50 percent.[1]

1. Based on Gantry Group ROI Report for HP Project and Portfolio Management.

A sample workflow for new application proposals might go something like this: After a business user or IT staff submits a request, a business-relationship manager evaluates the request to see if it requires a business case. If the business case passes the respective approvals and evaluations against the portfolio, the workflow automatically prompts a new application project.

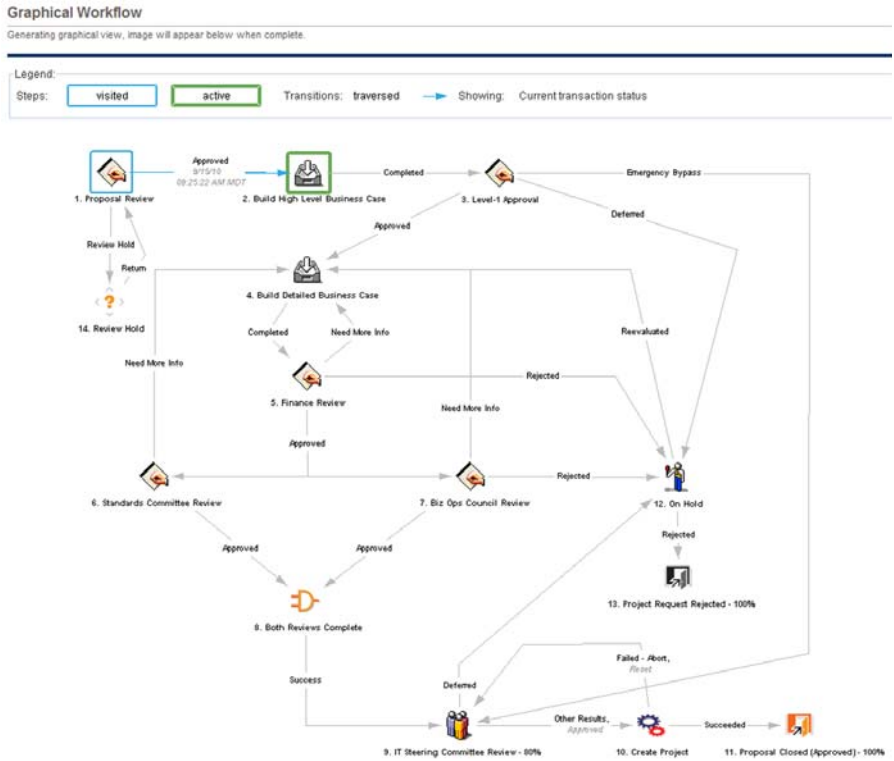


Figure 3: A customized, automated, and auditable workflow (source: HP Project and Portfolio Management Center)

By managing *all* application requests, the organization gains a comprehensive picture of past, present, and future IT demands. You can then prioritize, assign, or archive demands. This centralization of demand also permits trend analysis to identify recurrent problem areas or redundancies.

Another benefit is the creation of a system of record with an audit trail and transaction history that is part of the request and business case for regulatory compliance.

Compare Apples to Apples

Getting a comprehensive view of demand and automating the workflow as much as possible is half of the battle. But the discipline of comparing and prioritizing requests remains.

The first step is to ensure that each request is categorized in a standard way. Categorizing requests allows the organization to understand the scope of the request: Is it a defect fix, a minor enhancement, or an entirely new application? The larger the request, the greater the justification it requires. Request categories might capture this information:

- Application class: innovation, growth, efficiency, maintenance, compliance
- Business objectives: increase revenues, contain costs, kickstart new business initiatives
- Sponsor and beneficiaries: the business organization that will sponsor the request and organizations that will benefit
- Resources required (headcount, perhaps by role)
- Budget: include capital and operational estimates, as well as labor
- Business value vs. risk: essentially the cost-benefit analysis, for which many techniques are available (See Sample Value Criteria on the next page.)

Sample Value Criteria

To drive subjectivity out of the process and to limit political battles, a standard set of criteria ensure that for each large request, business value remains foremost in the discussion. Here is where the value proposition provides the means to rate each request in a quantifiable way.

Value criteria	Options
Internal rate of return	■ Zero or less ■ 1 to 2.99% ■ 3 to 6.99% ■ 7 to 9.99% ■ 10% and above
Strategic match	■ No impact to business goals ■ No impact to business goals, but increases efficiency ■ Indirect impact to business goals ■ Direct impact to business goals
Competitive advantage	■ No client or market impact ■ Improve operating efficiencies in strategic areas ■ Improve operating efficiencies affecting competitive position
Productivity	■ Zero or less ■ 11 to 25% increase ■ 26 to 50% increase ■ 51 to 75% increase ■ 76 to 100% increase

Here, as in other parts of the lifecycle, automation is paramount. Determining the ideal mix of proposed application projects, active projects, and operational work to fund in any particular timeframe is not an easy task if you're trying to eyeball spreadsheets.

The best automated capabilities allow users to model changes to dates, budgets, or resources to meet IT and business goals, whether based on strategic alignment, return on investment (ROI), net present value, benefit realization, capacity, skill availability, or a combination of these or other factors.

Figure 4: IT demands and projects by risk vs. value (source: HP Project and Portfolio Management Center)

Real-world results: A large communications company cut the time of request-to-approval in half by automating their demand-management process — and did so with the IT department operating at a third of their industry's average cost.

Add the Missing Ingredient: Reality

Of course, the best-laid portfolio plan means very little if it's not integrated with the realities of active projects: budget variances, revised resource demands, new deadlines.

Such a robustly integrated view is nearly impossible in a large enterprise without modern management capabilities. The right solution will unify live project data with the program or portfolio view, which provides management with a holistic, real-time picture of progress. Using emails, phone calls, spreadsheets, and disconnected project plans—even if a manual aggregation of these elements is possible—consumes huge amounts of staff time and the data are bound to be out of date almost as soon as the assembly is complete.

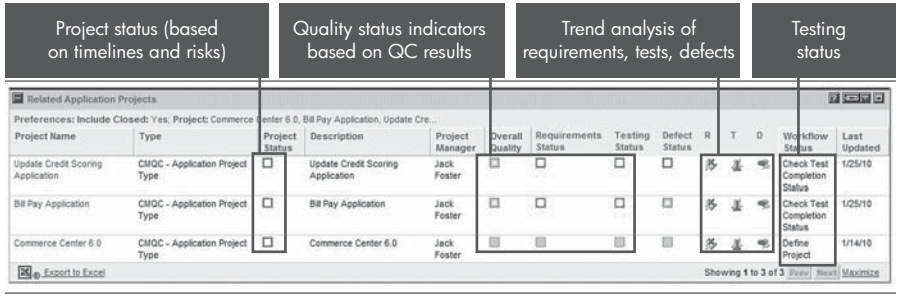


Figure 5: Real-time program view (source: HP Project and Portfolio Management Center integrated with HP Quality Center)

Program or portfolio views, in addition to those that track budget depletion and resource utilization, are paramount to proactive problem resolution and to understanding whether portfolio bets will stay aligned with the expected payoff.

Real-world results: An energy supplier, formed by the merger of two regional energy companies, wanted to rapidly lower IT costs and demonstrate the value of IT to executives. Using a portfolio-management solution, this customer was able to respond to the mandate quickly, and saved $8 million by identifying at-risk or low-value projects. The company also improved the number of healthy projects by 70 percent — all within eight months of implementation.

Keeping It Real: Portfolio Optimization

All of this information about portfolio management suggests a framework for determining how the portfolio is plotted, and how the business and IT remain aligned as requests move from idea to active project.

But, of course, portfolio planning is not a once or twice yearly event; it's a continuous activity of synchronizing new requests, active projects, and the business performance of live and long-standing applications. Consider a representative list of the questions that CxOs might ask to optimize the portfolio:

- Are we funding the right projects?
- Is our staff working on the highest priority tasks?
- Do we have the right resources in place to meet next year's goals?
- Are we getting maximum value from our IT investment?

The answers are rarely simple. Above all else, effective portfolio optimization requires a company to be able to answer "what if" questions.

- What if we borrow resources from this project for that one?
- What if we terminate this early-stage project and merge its budget with another?

- What if we delay the start of this project by two months?

These questions are particularly thorny because they pit operational risks against business value. We need the ability to model various scenarios based on reliable data and current metrics.

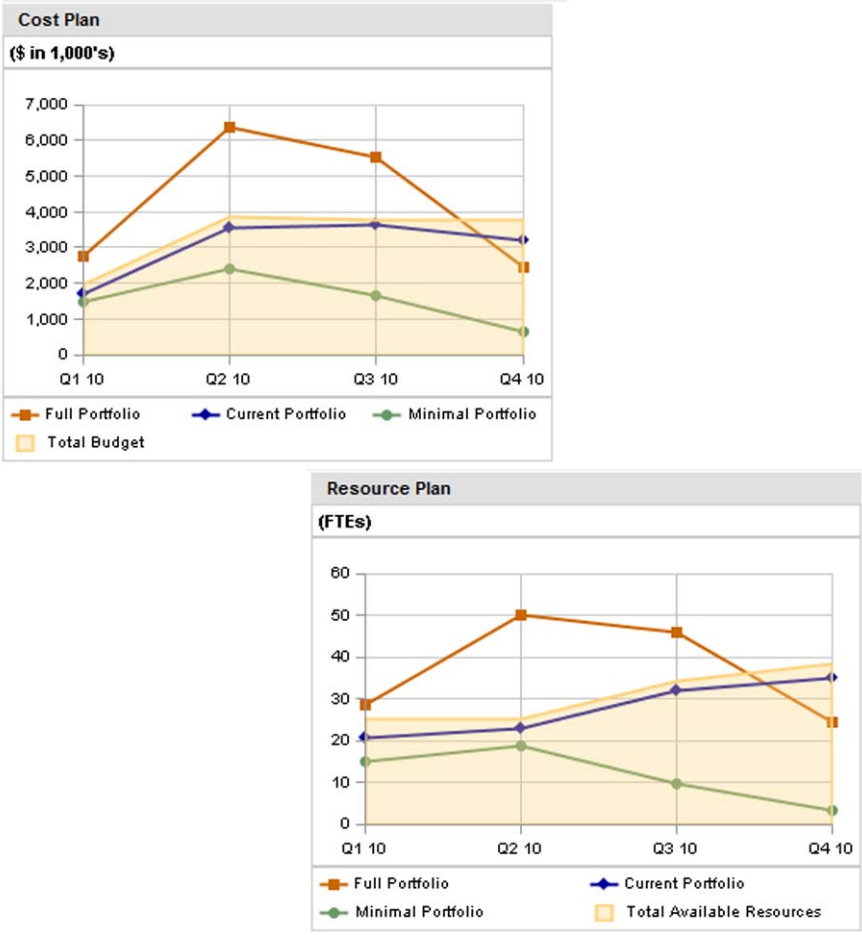


Figure 6: Demands and projects vs. budget and resources by quarter (source: HP Project and Portfolio Management Center)

These charts look at scenarios of different candidate proposals and current active projects. By changing the number or timeframe of certain proposals, planners can see in an instant the

impacts on cost and resource availability. It can also help planners:

- Revisit proposals that are not in the current planning period but are still relevant for the business within the next planning period.
- Identify proposals that cross fiscal reporting periods.

Summary: Application as Asset

An application is a business asset. It merits the same comprehensive financial treatment (ROI, depreciation, etc.) as other assets.

Portfolio management and optimization is the critical discipline in proper IT financial management, where enterprises achieve complete visibility into the delivery and support costs of the application, from proposal through retirement.

Only by gaining the necessary transparency of costs can IT hope to control them, calculate value, and direct spending for maximum innovation and business value.

The 'Other' Enterprise Planning: Architecture and Policy Governance

We've made the case that too few enterprises recognize the huge change that modernization means for their application teams. Perhaps nowhere is this more evident than in application architecture.

Gone are the days of brittle, standalone applications. Increasingly, the application results from an orchestration of pre-existing components and services.

This is next-generation architecture and it's redefining the way applications are designed, built, and delivered. However, adopting these new architectural strategies doesn't just happen by embracing buzzwords like "service-oriented architecture" (SOA) or "composite applications." Getting the maximum

effect from these strategies requires an important change in the way that companies plan, direct, control, and measure their enterprise architecture.

IT's Industrial Revolution

In essence, service-based architectures allow you to build applications the way modern manufacturers produce goods. Very few commercial products are handcrafted custom items. Instead, most are built from prefab components on an assembly line.

Application delivery can happen the same way in the age of reusable services: Use reliable parts to assemble applications (rather than build from scratch). The benefits are the same—more consistency, better quality, and a much greater ability to scale.

It's an elegant strategy, but requires real investment to become reality. Adopting a services-oriented approach brings challenges:

- Organizational behavior
- Cross-team collaboration and information sharing
- Consistent processes and policies that drive compliance and guide decision-making
- Modern application architectures mean more moving parts

These changes run counter to the way IT has delivered applications, systems, and solutions over the past 40 years.

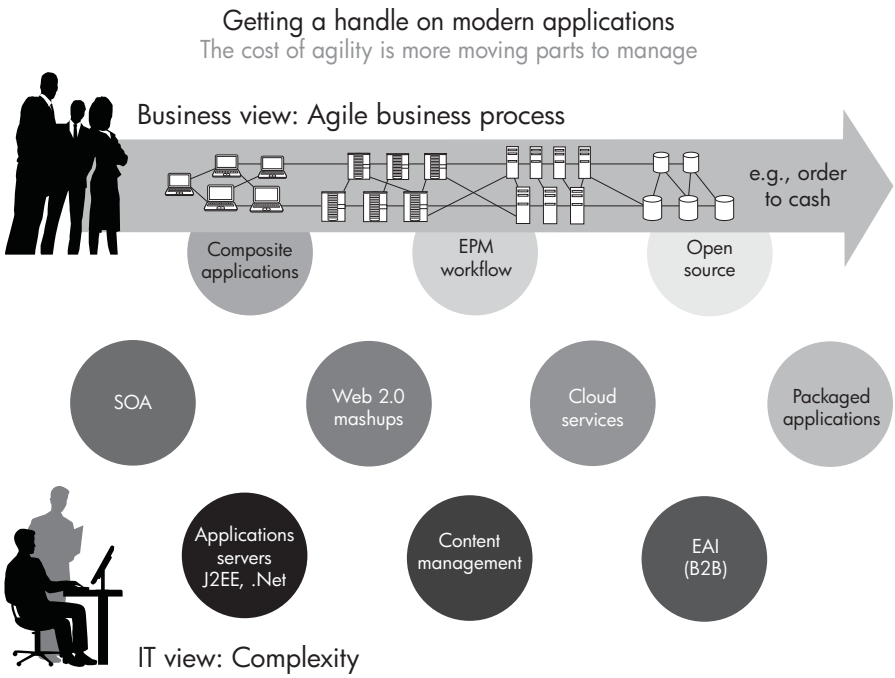


Figure 7: The reality of modern application architecture

Much of the business logic that goes into each application consists of components that can be reused and shared. This architecture promises quicker assembly of new user-facing, composite applications or automated business processes.

However, with this evolution comes an explosion in the sheer amount of information you need to track:

- Who is building out the shared components
- Who is using or consuming them
- Whether they are being developed in accordance with architectural standards
- How they are (or are not) meeting their service level agreements (SLAs) with increased use
- What changes are planned

And so on. It can get overwhelming.

To make it all work, application teams must have good cross-team visibility, so that the right information is in front of the right people at the right time.

> The good news is that we can use what we've learned through adopting SOA and shared web services during the past decade.

We can now leverage our experience across all types of modern application styles to guide IT through these changes faster and more efficiently.

Enter technical policy governance.

This explosion in moving parts and the need for information about the behavior and trustworthiness of those parts began with the rise of SOA in the mid-'90s. Now we expect to reuse a component or service instead of writing the same code over and over again—it's hardly architecture-specific. It's simply a best practice of modern application organizations.

Whatever acronym you use to describe it, your architectural strategy is very likely one that cries out for effective governance.

Think of what you want to know if you're a developer working to create or enhance a modern application:

- Is there a component that already exists that will deliver the business logic I need?
- Can I discover that component with a minimum of hassle? (If not, I'll write it again myself!)
- Do I need to let the owner/administrator of that component know that I plan to use it? If so, is this easy to do? (If not, I'll just ...)
- Can I trust the component? (Can I be sure it will function and perform, and not expose my application to security risks?)

- What are the key technical standards I need to know and abide by for the components I'll create? Are these easy to discover? (If not, I'll just do the best I can; I'm extremely busy.)

Technical policy governance, when done right, provides an affirmative answer to those questions. It delivers:

- **Visibility:** A supported and consistent way to publish and discover application artifacts, shared services, and their dependencies, which usually involves a "metadata catalogue."
- **Timely metadata:** "Data about data" means that the right information can be available to the right people at the right time, which supports application change and use and reuse decisions.
- **Policies:** These ensure compliance throughout the life-cycle of an application.
- **Measurement:** The ability to constantly collect and report on adoption of enterprise goals and standards.

When you invest in policy governance, you're building best practices across IT.

As modern application projects scale up, technical policy governance ensures a stable foundation to guide new projects both inside and outside (such as partners and outsourcers) the company walls.

How to Manage What You Can't See

Modern applications are built for agility: Loosely coupled components can be rapidly assembled to form new composite applications and business processes. While this sounds promising, it means that what was once an M + N problem (one application, one user) now becomes an M x N problem (many users, many shared services).

This rapid proliferation of moving parts, users, and dependencies means the old, ad hoc approaches of spreadsheets, wikis, and email are no longer scalable.

> A "wild west" of components and services can quickly emerge, with no one knowing who created what, what each one does, whether it's been properly tested, and so on.

Policy governance tames this unmapped country with a system of record (metadata repository or catalogue). You get:

- Visibility into the existing application components and into any dependencies among them.
- Information from other areas, such as design-time documents, source-code management systems, quality-management and testing systems, and run-time systems.
- An integrated view of the status and compliance of application components at their various stages of lifecycle progress.

As a result, policy governance delivers the information that distributed teams need to so that they can collaborate. You also gain awareness of the application architecture, and the assurance of architectural- and standards-compliance through policies.

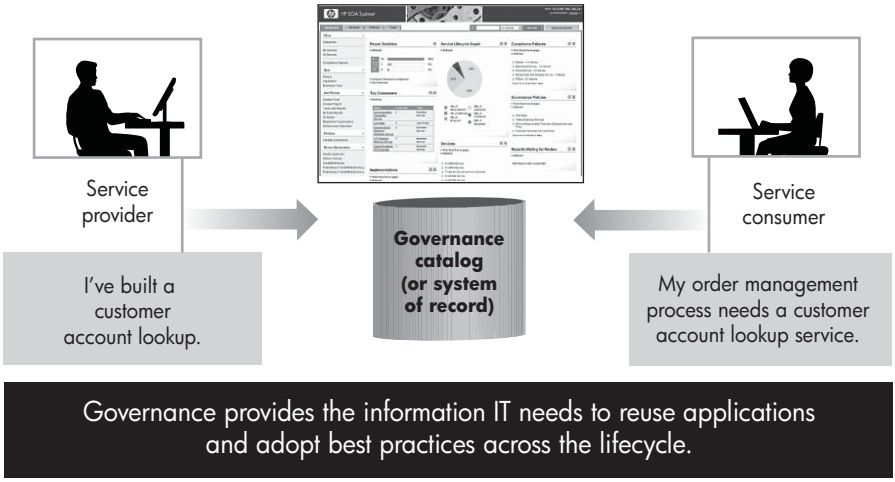


Figure 8: Visibility: You can't manage what you can't find

Pay Now or Pay More Later: The Dilemma of Technical Debt

In 1992, at the Object-Oriented Programming, Systems, Languages and Applications (OOPSLA) Conference, pioneering programmer Ward Cunningham introduced the concept of technical debt. In Ward's original formulation, technical debt refers to decisions made during coding that reduce the current delivery effort, but will necessitate a later (post-delivery) investment in refactoring.

Here the assumed "debt" is taken on deliberately. That is, a programmer, presumably in concert with his manager, makes a decision to forgo some technical best practice with the aim of correcting it at a later time.

Several important assumptions are made here:

- A formal record is kept of how much debt has been absorbed into the application, and where.
- Time is set aside later to "pay off" the accumulated debt, before it swells to crippling proportions.
- The decision to take on some debt is a conscious one.

The third assumption is the most pivotal. Why? We know that rarely do all the developers know an organization's technical standards, and those standards are never applied 100% of the time.

There are numerous reasons for this situation:

- Policies change with little notice.
- New hires may not be trained on the company's latest technical standards.
- Design and code reviews become casualties of project deadlines.
- Reviewers may not know all the latest policies.

Technical policy and compliance is too often a kind of tribal knowledge that is conveyed by email, advice passed over cube walls, and so on. Even for co-located teams, this is an impractical way of managing technical policy. For distributed teams, it's nearly impossible.

Without ready knowledge of all the company's technical policies and best practices, developers can only go by what they know. The upshot of this is applications that become bloated by significant quantities of unconscious technical debt.

And, the longer the debt goes untended, the greater the costs. These costs manifest in a variety of ways.

One way is functional rigidity in the application: Complex, noncompliant code is much harder to enhance. The same is true for basic maintenance and bug fixes. Given the rate of application change in the modern enterprise, rigidity from

technical debt amounts to a particularly expensive form of "interest payment."

Then there are the costs to things like application performance and security.

Ultimately, the application may reach a state where the only option is to throw it out and start over, which is the development equivalent of bankruptcy.

Managing Debt with Automated Policy Governance

Our message isn't that application teams should never take on debt. Properly managed, technical debt can do what the right kind of debt does—provide flexibility and leverage in tight times.

But, given the steep difficulties of conveying technical policies and best practices consistently across the enterprise, how can you identify and manage debt, instead of take it on unintentionally and recklessly?

The answer again lies in automation.

By automating policy governance, an enterprise can achieve consistent, cross-project observance of its technical standards. Automated governance allows you to establish policies and assign them to the relevant stage in the application lifecycle. Delivery team members then validate their work against such policies using their native tooling environments (for example, the developer uses IDE and the tester uses the test solution).

Let's look at an example.

Say the enterprise-architecture team has decided that classes of more than 4,000 lines of code indicate unnecessary complexity. Using automated governance, a class-size policy is set accordingly, and its compliance check is configured to run at the end of the development stage of the lifecycle. Before the application can be promoted to its next lifecycle stage, the development manager or project architect would know whether and where classes exceed the size limit.

The decision then becomes explicit and "public": Do we accept some unwieldiness in certain classes for the sake of pushing to the next phase, or, perhaps given the number of overlarge classes, do we delay to pursue some refactoring now?

Instead of counting on tribal knowledge, policy compliance becomes an automated, integrated part of every project. If a team elects to abridge or bypass some policy for the sake of delivery time, the exception becomes part of a system-generated compliance report, which is essentially a ledger of the technical debt assumed by the team.

This ledger can then feed the backlog for the application, which helps to frame the objectives and priorities of the next release.

> Real-world results: A large retailer of mobile phones implemented automated policy compliance around some simple coding standards and saved nearly a million dollars in the first year.

How to Reduce Technical Debt

Besides preventing the accumulation of unmanageable quantities of technical debt, automated policy governance can identify and quantify the sources of debt in existing applications. This means you can even vet applications that were developed before automated governance for policy compliance.

You need only run the automated policy checks against the existing application. Doing so allows an organization to determine where the greatest concentrations of technical debt are and the type of debt incurred. Then, you can plot a path to debt reduction.

Automated policy governance allows the enterprise to avoid undue technical debt.

Giving all teams real-time access to the latest standards, and enabling automated verification of these standards whenever possible significantly reduces noncompliant development.

Instead of forcing huge interest payments of unknown size onto future releases, you're reining in technical debt. Teams are given a pragmatic, controlled way to balance delivery deadlines and business objectives.

Frenemies? Technical Policy and Business Policy

A software architect at a very large technology organization was once told by another architect at the same organization: "We don't use Enterprise JavaBeans." When asked why, the architect answered, "I'm not really sure. We just don't."

Too often, technical policies are the result of personal prejudice or whim. Nothing stalls an application-governance effort faster than rules for the sake of rules—that is, an overly rigid enforcement of technical policies that seem to have no bearing on the goals and objectives of the business.

Technical policies require strong, clear business justification. Good governance begins with understanding what is most important to the organization, and then selecting an initial shortlist of maximum-impact policies that reflect these goals.

You can apply most critical best practices and standards, even across distributed and virtual organizations by completing these tasks:

- Align your business and IT teams.
- Work to define and agree on the most important standards and practices to govern.
- Effectively translate these standards and practices into business and technical policies.
- Measure and reward compliance.

'Governance' Is Not a Four-Letter Word

Governance is associated with heavy-handed procedures and rules, so delivery teams may distrust it. Those using Agile methods tend to distrust it the most.

> Too often, Agile development is seen as another name for "cowboy coding."

The truth is, *automated* policy governance can further Agile objectives in much the same way that it can be used to manage technical debt. How?

An enterprise can effectively establish the "guardrails" for development by selecting a few key standards and automating the compliance check.

- Developers can be as entrepreneurial as they like within the established rules.
- Managers can be confident that standards haven't been abandoned in the name of Agile speed.

Automation remains paramount in this scenario. Attempting to enforce governance without automation is like trying to test without automation: It makes working in Agile sprint-time nearly impossible.

In practice, each iteration includes automated validation of policy prior to closure. If certain policies fail validation, they are either corrected by extending the iteration, or they become part of the backlog and are adjusted in the next iteration. Either way, a cadence is developed so that the Agile teams remain quick on their feet but also don't go "off the edge" of architectural integrity and interoperability.

Reuse: Haven't We Been Down This Road Before?

Yes, we have. But only now is the destination— true economies of scale—in sight.

In the days of 3GL and procedural languages, we had artifacts, such as functions or subroutines, and copybooks or "data layouts." In theory, these artifacts could be reused, but for the most part they became locked into siloed applications.

The industry later evolved to object-oriented languages, where data and behavior merged into individually reusable objects. However, a lack of open standards and interfaces made the resulting implementations proprietary, and thwarted the goal of modularity and broad-scale reuse.

But with modern application architectures, we are realizing reuse and modularity. We have seen "code-level" reuse in larger-grained entities like components (such as open source projects). And, SOA adoption has allowed IT to move from code-level reuse to "run-time reuse" via live services.

Looking ahead, cloud computing has the potential to expand this reuse even further and outside the walls of the individual organization. Architecture and policy governance ensures that as available services and components multiply, developers and delivery organizations can:

- Find the components and make educated decisions around the impact of reuse to their specific project goals (something source-code-centric solutions can't do).
- Formalize the relationship between the provider and consumer of a shared component, ensuring that dependencies are visible and that components are trustworthy.
- Abide by technical standards and policy without guesswork or undue burden.

In sum, the effective governance of modern architectures is what makes reuse and grand economies of scale a reality, at long last.

Chapter 2

DELIVER: Modernizing the Core Lifecycle

Key Questions:

- Everyone talks about being more agile and responsive, but what does this mean in practical terms?
- Managing a delivery organization today feels like flying a plane over mountains in thick fog, with the instruments out. How do we see the work and track all that's asked of us?
- Why does it seem that every project is a new adventure? How do we start getting real consistency in our results?
- Given the size and sophistication of our applications, and the pace of change, how can we be sure complexity won't trump all our work on quality?

It's time to sideline the silo-per-discipline structure that has defined the IT shop for decades.

Modern applications, and their delivery, demand a level of collaboration and orchestration that might have been laughed off as utopian even five years ago. But this is not your father's SDLC. The days of monolithic applications built by a rigidly sequential process, with one team completing its work largely in isolation before handing off to the next, are fast fading.

The objectives and how-to's of 50+ years of IT practice in the core disciplines are still important: good project management, sound requirements, effective development, thorough testing, and hiccup-free deployment.

How those core disciplines function in context with one another and with the broader, complete lifecycle is critical.

If you need proof that team collaboration has come to the fore, witness the rise of application lifecycle management (ALM) as a discipline. Definitions of the term continue to evolve, but generally speaking, the traditional SDLC disciplines are players in the orchestra and ALM is the conductor.

These definitions rightly highlight the need for improved traceability among project artifacts, better enforcement of process and workflow, and dynamic, real-time access to project state in the form of key performance indicators (KPIs) and other metrics.

However, automation gets short shrift: The *M* in ALM has indeed been "management" only. But mastering the application lifecycle requires both management and automation. Working together, each triggering actions in the other, it's much like the cockpit dashboard and dependent systems (fuel injection, hydraulics, etc.) do in an airplane.

Traditional ALM has been an SDLC-only business. The *L* has stood for a lifecycle that begins with requirements and ends with deployment.

But this core application lifecycle exists in a wider context, that of the complete lifecycle, which includes planning, delivery, running, and retiring the application. From this perspective, lasting application success means effective coordination among multiple IT disciplines.

Let's set disciplines and acronyms aside and speak plainly about what it takes to become a world-class delivery organization.

In our experience with customers, the best delivery teams share four traits:

- They are *change-ready* and understand that the only constant in the modern application is change. These low-latency, high-speed shops respond quickly to business needs.
- They have achieved a state of *predictability* and are able to accurately forecast a project's progress and anticipate future demand.
- Their delivery methods show high *repeatability*. While variables change (project size, available resources, timeframes, etc.), the organization is machine-like in the consistency of its results.
- Their output is of consistently high *quality*. The applications work as the business wants, perform against the SLAs, and are secure.

We'll look at each of these traits in turn, and focus on the ways management and automation make success possible.

Nimble Organizations Are Change-Ready

Not long ago, companies had only a few major applications to help their business run. The application lifecycle was a relatively rigid and simple affair:

1. First, the business came to an investment decision, and after some negotiation with the delivery team, signed off on the functional scope (in blood).
2. The delivery teams, contract in hand, went to work for months and months; any change requests along the way sent all parties back to the negotiation table.

3. Finally, the application was ready for production: It moved into the live environment, and the operations teams were instructed not to touch it.

Cut to the world of today, where nearly everything the business does is enabled by one or more applications. This means the business can't so much as twitch without requiring change in an application. The responsiveness of the business to market opportunity, to competitive threat, and so on, depends entirely on the responsiveness of the application teams. The upshot for those teams is that the new norm is change and flexibility.

> Change is the new norm.

The rate of change in the modern organization is vexing for application teams. The trouble begins with gauging the impact of any single change, to say nothing of many changes at once.

Change-impact analysis must take into account a myriad of project artifacts spread across multiple teams. Usually these artifacts have been developed in different point tools (or no true tool at all, leveraging Microsoft Word or the like) and exist in different data stores (including various local hard drives).

Multiple artifacts in multiple locations means no single view into the relationships and dependencies among them is possible. Impact analysis, then, becomes a days-long game of telephone. Meanwhile the business waits, tapping its foot impatiently.

> Comprehensive *requirement traceability* is a key trait of the change-ready organization.

What does this mean?

First, you must effectively capture the application requirements and put them under change control for purposes of

audit and rollback. Instead of a disparate set of Word documents scattered across file servers, there is a single source of stakeholder truth.

With requirements centralized, all dependent project artifacts are linked to their sponsoring requirement, which includes tests, known defects, and even source code. It becomes possible to see at a glance what effect changing a requirement will have on the downstream artifacts. Teams can see how many tests, for example, are affected by the change. They even have information about which are the highest priority tests and how long they took to run in previous cycles, providing much more accurate insight into how long the change will take.

Impact analysis also works in reverse. Changes to code, tests, and defect status can be traced back to requirements to ensure that these changes are understood and evaluated in the context of the requirements elicited from the business.

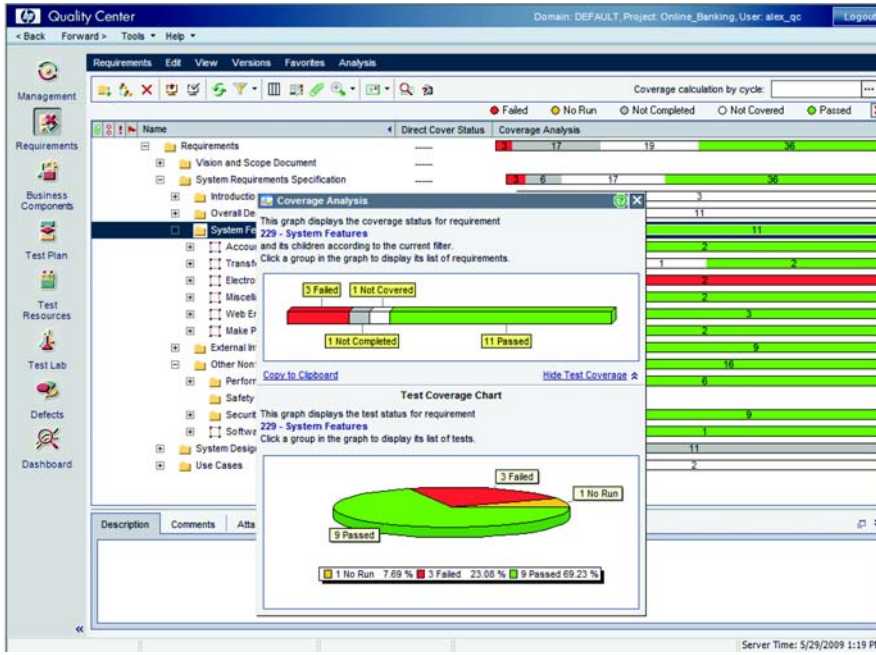


Figure 9: Comprehensive traceability at a glance (source: HP Quality Center)

Of course, impact analysis is only the first step. There remains the not-so-small matter of actually implementing the change. Here again, traceability is critical to both speed and efficiency. Any change must be complete: Requirements must reflect the latest stakeholder position, all impacted modules must be modified, and all related tests must be changed or augmented.

Ensuring the thoroughness of any change is a long, laborious process, which is prone to oversights and other human error, so traceability is essential.

Few things are more uncomfortable for a delivery organization than to discover that some urgent, high-profile change that was pushed into production does not, in fact, work as intended because of a missed enhancement in a test condition or some other element.

Note that change-readiness is not the responsibility of only the delivery teams. Here again, we must see the core lifecycle in its wider context. The ability to identify, evaluate, approve, implement, and deploy change indeed encompasses the complete lifecycle. The right strategy integrates the service desk, portfolio planning, and requirements-management solutions.

So, a traditional silo approach to application change—one marked by ad hoc email and phone requests, and fraught with lack of transparency and miscommunication—becomes a single, integrated process for change evaluation, approval, and scheduled delivery.

Agility is the top-ranked business reason that drives modernization initiatives. In the modern organization, business agility and application agility are one and the same. The business' dependence on applications means that you can't have one kind of agility without the other.

What better justification could there be for investing in the mechanisms that will speed the throughput of change?

Better Predictability, Fewer Surprises

This anecdote may sound familiar. In the course of working with an external consultant, the business and IT managers of a large professional trade group present diametrically opposed views of IT's responsiveness to the business. Their statements (taken independently) go something like this:

Business manager: "IT is almost always telling us 'no,' or, 'not now, we're too busy.' I always want to ask: What could you possibly be busy with? You say no to everything!"

IT manager: "We're killing ourselves because we say yes to everything. It puts us in a constant reactive mode, and all the reprioritization and context switching makes it impossible to see anything all the way to finish."

Who's right?

They both are, of course.

The problem here is the lack of shared view into (1) delivery capacity and (2) business demand. Without these views, the delivery organization is doubly blinded: unsure of its own run rate, and prone to "surprise" requests from other parts of the enterprise.

In this blinded state, the decision to accept (or reject) a work request becomes personalized and prone to escalations and bruised feelings.

We looked at the challenge of evaluating delivery capacity in Chapter 1. For the mystery of business demand, delivery leaders require a real-time management view into pending requests as they arise—a kind of early-warning system for demand).

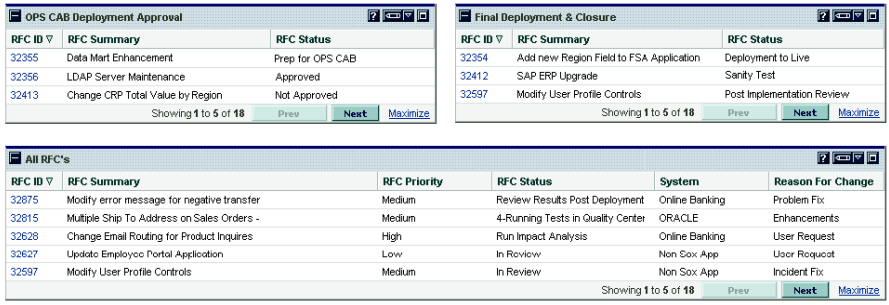


Figure 10: A centralized view of requests for application change (source: HP Project and Portfolio Management Center)

With such a capability, delivery leaders are able to better anticipate demands on their teams and work with requesters to develop mitigation strategies based on objective data instead of by feel.

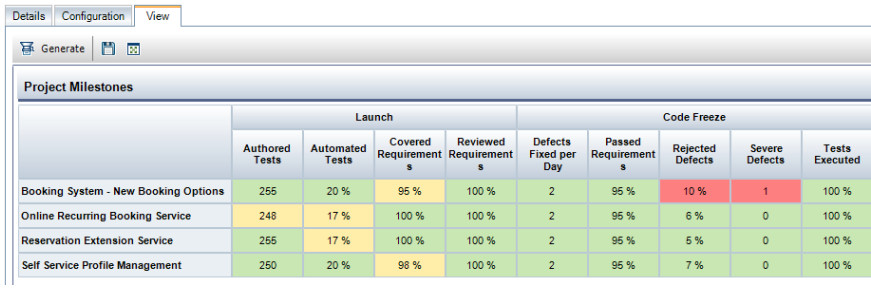
The harder and more pervasive challenge for delivery leaders is arriving at a reliable, real-time view into the state of active projects. Anticipating new demand is one thing; knowing the state of a myriad of projects is something else altogether.

Modernization has made this problem especially acute, and outsourced and virtual teams are often in the mix. The common standard is for project managers to interview team members and collate the resulting estimates (hours worked, days remaining, etc.) into some sort of reasonable composite view. With teams geographically distributed, this cumbersome, manual assembly of status becomes unworkable: By the time a picture of progress has been assembled, the data are obsolete.

Clearly, the right kind of management solution provides an integrated, real-time view of status across projects. And, you should measure "status" by binary, provably completed milestones. Measuring hours worked may be necessary for cost purposes, but it shouldn't be mistaken for a progress metric.

> Hours worked reflects *effort*, not progress.

Better to indicate progress with data like "successfully tested requirements," which reflects a truer state of how much business functionality has been implemented.


Details Configuration View

Generate

Project Milestones

	Launch				Code Freeze				
	Authored Tests	Automated Tests	Covered Requirements	Reviewed Requirements	Defects Fixed per Day	Passed Requirements	Rejected Defects	Severe Defects	Tests Executed
Booking System - New Booking Options	255	20 %	95 %	100 %	2	95 %	10 %	1	100 %
Online Recurring Booking Service	248	17 %	100 %	100 %	2	95 %	6 %	0	100 %
Reservation Extension Service	255	17 %	100 %	100 %	2	95 %	5 %	0	100 %
Self Service Profile Management	250	20 %	98 %	100 %	2	95 %	7 %	0	100 %

Figure 11: A real-time, consolidated view of program state (source: HP Application Lifecycle Management)

The days of "management by walking around" may not be completely over, but the realities of the modern organization have made it a difficult way to arrive at a true understanding of project progress. Predictable delivery counts on real-time, reliable metrics.

Repeatable Lifecycle: Consistency in the Face of Change

While the word "lifecycle" connotes a circular practice, moving from beginning to end and starting over, for many application organizations, the experience of delivery is anything but a same-and-again process.

Each new application initiative is an adventure fraught with uncertainties. Will the results be better or worse than last time? Will the staff of this project be up to the task? Will the business stakeholders know their own minds? This may sound exciting, but it's an excitement IT would happily live without.

> When it comes to application delivery, boring is better.

Why is repeatability so difficult to achieve? Because in application delivery, change is the norm.

The scope of work from project to project varies widely; resources roll on and off projects and in and out of the company, new technologies and tools come and go, and methodologies are endlessly tweaked.

Yet, despite these shifting sands, world-class organizations find a way to deliver on their obligations to deadline, quality, and cost, time and time again.

Instrumental to their success is an investment in automation and a commitment to asset reuse. But in practice, the difference between an organization that does not use comprehensive automation and asset reuse and one that does is like the difference between a cobbler and Nike.

> If the objectives are scale, performance, functional richness, speed to market, and cost efficiency, there is no place for the label that reads "Made entirely by hand."

Automation

Say the word "automation" inside most delivery organizations, and we think of test automation. It's true that test automation has been a singular breakthrough in eliminating costly, intensive manual work and giving a key IT function scale and consistency. (We'll look at some of its latest uses in "Threats of Complexity and the Quality Response.") But opportunities for automation exist for a host of other complex activities.

Consider workflow management. It's easy to forget the degree of coordination—the sheer frequency of work handoffs —required to make an application come to life.

How does a development manager working in Bangalore know that a requirement worked on by a business analyst in Texas is ready for task planning? How does a tester in North Carolina know that a defect fixed by a developer in California is ready for test? What happens when there's a delay? How is project management kept in the loop?

If your answer to these questions is "a team meeting" or "phone calls," you've just uncovered an area primed for automation.

Effective workflow management lubricates the gears of delivery, using triggering mechanisms (a change in status of a defect, or the entry of a new requirement) to automatically notify affected team members that a new task awaits some action on their part (review, code modification, retest).

The right workflow-management solution will also raise any issues, like the delay in some work item, to project management's attention using preconfigured escalation criteria.

Multiple Cooks in Multiple Kitchens

The labor to deploy an application into the production environment is another area long in need of automation. Those who have lived through application deployment know it's a challenging proposition involving multiple cooks in multiple kitchens. An application has many moving parts: components in multiple tiers (web, application, database) that allow for flexibility in development, availability, and load balancing.

Preproduction environments further complicate the picture: Regardless of investments made to mirror production, they are never exact replicas of the live environment. The behavior of the various application components and tiers in one environment is seldom identical to the behavior in another.

These complications highlight the central difficulty of deployment. Developers know which application components must go where, and the optimum configurations for each. Production personnel know the environmental destinations and their optimum configurations. The two sides must make their knowledge of these complexities mesh seamlessly and quickly. How is this done? Frequently, through manual information exchange: documentation, emails, phone calls, meetings.

There is very little margin for error in deployment. One forgotten or overlooked parameter variable can mean the application doesn't work.

While we've brought some automation to bear on the problem, often these measures serve only one side or the other. Server-automation tools will ensure that the server environment is ready for the application server; they may even install the application-server software. However, these solutions benefit server administrators, not the application teams.

Conversely, solutions that automate the management of application servers may enhance some deployment capabilities, but inevitably they leave deployment of the actual physical server out of the picture.

The right deployment automation solution should connect the worlds of the applications and operations, and serve each without losing the unified view. This means that operations must be allowed to define their standard OS builds, enforce specific configurations, and manage server policies.

Application teams should be able to leverage these standards into their application-server frameworks, and define which code components must come from which build systems and how to configure the application-specific parameters.

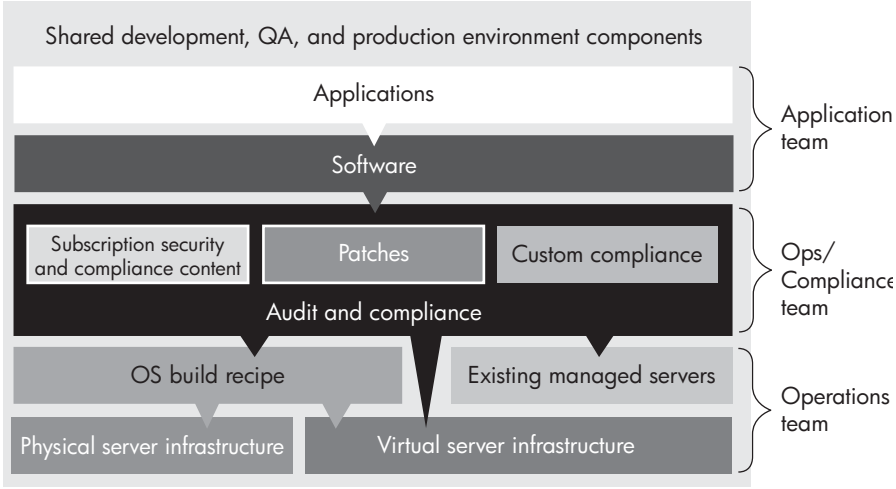


Figure 12: A unified approach to application deployment automation

The goal is to eliminate as much as possible of the manual manipulations that allow for mistakes in the complex deployment process.

Automating deployment in a unified fashion provides an approximate of cloud computing. All dependencies and procedural requirements are automated, which allows "push button" creation of services to meet business demand with the same right result time and again.

Asset Reuse

It sometimes seems that no matter how hard delivery teams work to increase reuse—consider the push for service-based architectures, as one example—the forces of modern delivery drive them in the other direction.

The geographic distribution of teams and the natural proliferation of repositories work against anyone having a single, simple view into the assets created by another team. The pace of delivery hardly allows a tester to spend hours (or days) seeking out existing test cases that might be put to use on her project.

When wheels are regularly reinvented, the problem is not only one of time and cost. Inconsistency is also an issue. The same customer need may be stated three different ways by three different analysts, and so are addressed slightly differently in three different applications.

Delivery teams understand this better than anyone. They know the value of reuse; they don't need to be told to "work smarter." What they need is a way of actually doing so.

The solution response should be a central view into available project assets: requirements, components, tests, and even known defects.

If they have the appropriate permissions, project members should be able to see—at a glance and without resorting to phone calls, emails, and other lengthy detective work—the complete inventory of existing artifacts available to them.

Automation and asset reuse are not the only engines to a repeatable lifecycle (process standardization and centers of excellence come to mind), but they are two of the most powerful. With them, a delivery organization puts itself in excellent standing to become a consistent and reliable partner to the business.

Threats of Complexity, and the Quality Response

In the modern organization, complexity exerts an ever-steady pressure on quality.

Consider the Quality Assurance (QA) department that invests in GUI test automation, only to see SOA services—which lack the GUI layer necessary for standard test script record-and-playback—become the application norm. Or, think of the testers accustomed to Waterfall methods who learn that their next project will embrace Agile, without adequate consideration of the impact on legacy test tools and mechanisms.

The continual change wrought by modernization may be why, despite years of acknowledging the importance of quality, most IT organizations still uncover far too many defects after the application is already live.

> The average IT organization uncovers more than 25% of defects after the application has gone live. World-class IT organizations report fewer than 5% post-production defects.[1]

Without an aggressive quality response to the complexities of modernization, the enterprise risks putting end-users into the role of testers, which effectively makes quality the customer's responsibility.

Our investigation of modern quality is framed by three simple-sounding questions:

- Does it work?
- Does it perform?
- Is it secure?

These questions represent the fundamental business expectation of every application. Let's look at how certain modernization trends have made answering these questions anything but simple.

Does It Work?

Peel back the GUI layer of any modern application, and you see an architecture teeming with shared services and subcomponents. Yesterday's application was a self-contained entity fired by components built by the project team for the work at hand. Today's application is a vessel for linked, independent moving

1. Caspers Jones, Software Productivity Research, LLC, 2008 Software Quality Study

parts—services, even entire subapplications—many of whose creators are external and anonymous to the project team.

In this world of the composite application, functional behavior must be validated. QA teams are required to test more layers and more integrations, to run tests more frequently and with greater emphasis on the end-to-end business impact of each transaction.

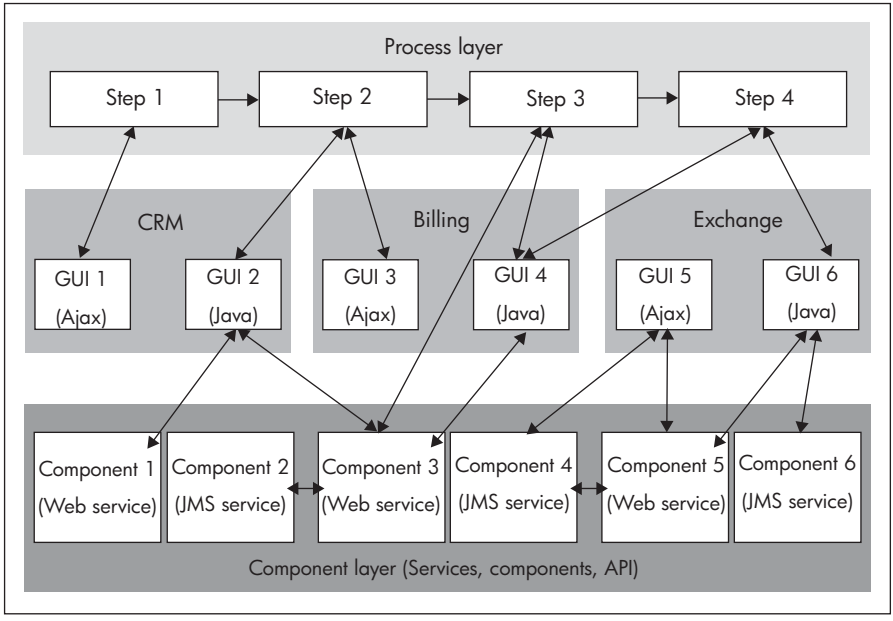


Figure 13: Cross-section view of a composite application (source: HP surveys)

Composite applications require you to expand the principles of functional testing to incorporate testing of "headless" services, which are those with no GUI.

Headless services thwart standard test-automation tools, which expect a GUI for script record-and-playback. However, manually testing a service requires a higher level of technical know-how, such as submitting data via command prompts and writing database queries to validate results. A more expensive resource must be used, and even with that skill level, the complexity of test execution chews up significant project time.

Composite applications also involve downstream processes that are external to the application under test, but that the application depends on. This often means defects that are discovered in one project have their root cause in another.

Consider, for example, the process of changing a customer address on an invoice. The billing application cross-references the address in the customer relationship management (CRM) system; if the address hasn't been updated there, the billing transaction fails. In this case, the defect appears as an error in the billing system, yet it is attributable to an entirely separate application.

This proliferation of external system dependencies is a significant complication for the test environment.

All these constituent systems and services must be in place and available for QA to validate the application under test. If they cannot be made available (because they are still under development, or belong to external systems) their behavior—inputs, outputs, response times, etc.—must be mimicked via stubs, convincingly enough that the QA team gets a full picture of the application's functionality.

For a composite application, this stub work can be exceptionally labor-intensive, so that enterprises will spend significantly to create production mirror environments. Test teams are provided with an environment to vet application functionality without having to stub for every dependent system. Such environments mean sizable capital and operating expenses, and even these investments don't solve the problem of dependencies on services (such as credit validation) that are outside the enterprise.

All of this means that to efficiently test a composite application, QA teams need a way to:

- Organize and develop a test plan for the process layer. It's a way to structure all the tests on dependent applications and services, and to produce from their results a composite view of process success or failure.

- Validate the behavior of both GUI and non-GUI components, without resorting to heavy manual effort.
- Access or otherwise mimic the behavior of the other systems and components that inform the application under test.

Happily, there's good news on all three fronts.

The right modern solutions recognize that with composite applications, the whole is greater than the sum of its parts. This means that the overall business process must be tested and validated, not just the underlying applications and subsystems.

The best solutions allow QA teams to create test plans using a business-process model, which ensures that what gets verified is, in the end, the process being enabled.

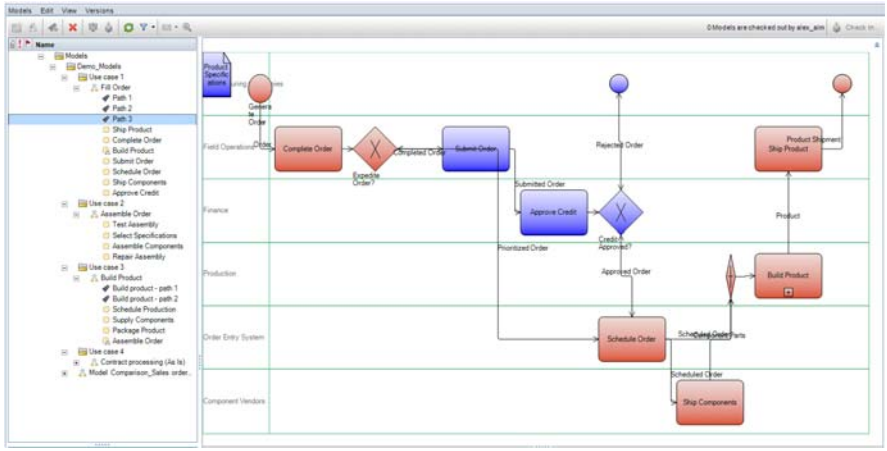


Figure 14: An efficient means of deriving the test model from the business process model (source: HP Quality Center)

Now that the test plan mirrors the business process, you must define and execute all the tests across all the systems and components involved in each process step.

Test automation is certainly the right response, but the solution must solve the problem of headless components. And, it should provide a single, visual interface for development of automated scripts for both GUI and non-GUI components. The

resulting unified test modules should be easily modifiable and reusable across multiple application environments.

This leaves the difficulty of stubbing for all dependent systems and services. Fortunately, this challenge has drawn significant innovation breakthrough in the form of service virtualization.

Service virtualization solutions work by "listening" to all the communications among the enterprise's production systems and learning the behaviors, down to average response times. The virtualization solutions automatically replicate these communications and behaviors in an entirely virtual environment. No new hardware, no physically mirrored environments, and no stubs are needed.

Does It Perform?

In the Introduction, we cited a recent survey by Forrester Research that showed a richer end-user experience to be one of the key aims of organizations as they modernize their applications.

Enter Web 2.0 and the rise of the rich Internet application (RIA), whose technologies transform the sluggish click-and-wait model of traditional web-based applications into a robust, dynamic front. The new user experience is so compelling that major enterprise-application vendors like Oracle and SAP are already taking advantage.

RIA technologies (Ajax, Microsoft Silverlight, JavaFX, Adobe Flex), along with connectivity enablers (RSS, blogs), work by delivering independent updates to select web components instead of refreshing the entire page. In a sense this marks a return of the "fat" client, but without the need for heavy-duty client software. RIA requires only a web browser and plug-ins.

However, the return of client-side logic complicates application performance. RIA technologies exchange granular amounts of data continually with a server. Hundreds of users interacting continuously with the application can mean significant new strain on servers and networks.

While these technologies may not change the underlying application infrastructure, they do often call upon multiple, diverse backend data sources, which makes diagnosis of performance issues more difficult. There are a few reasons that the new interface technologies complicate performance testing, problem diagnosis, or both:

- Mashups, one of the more popular Web 2.0 innovations, come in a wide variety of customizable architectures and frameworks with little consistency of standard.
- RIA presentation frameworks usually require more client-side libraries to deliver the richer interfaces. This means new latencies, which makes performance-test scripting more difficult.
- These frameworks offer cross-platform support and portability across heterogeneous client types (such as mobile), and some deliver offline capabilities "out of browser," which allows users to continue to work through disconnected sessions.
- RIA often means an abundance of dynamic behavior in the application interface, which again complicates test scripting.

Richer applications are fast becoming the new norm, so delivery organizations should be prepared for the performance challenges. The right performance-optimization solution should allow for these actions:

- Performance validation of both RIA client and server components.
- Interrogation of the time required for multiple webpage HTTP calls.
- Interrogation of the time required to complete common actions in an RIA client, such as dragging and dropping elements, or viewing and deleting web-based email.
- Verification of application responsiveness to client activity, including the speed of data updates and changes.

- Asynchronous behavior of the application.
- Breakdown of transactions to specific client components and isolating performance issues therein.
- Diagnosis and monitoring of performance bottlenecks in highly distributed, heterogeneous server systems.

The aim of Web 2.0 and RIA strategies is a richer customer experience, not timeouts and system crashes. The right performance strategy will help to ensure the business' aim isn't turned on its head.

You Know There's a Problem: Now What?

If testing for performance is tricky, consider the work of root-cause analysis.

Application performance invites potential trouble at many layers: web, application, database, infrastructure, etc. Determining the precise point of a bottleneck is complex, and it requires expertise in both the application and deployment environments.

Many organizations just throw more hardware at the problem. Aside from being a costly strategy, this only masks the problem, it does not fix it.

Performance root-cause analysis is another fertile ground for automation. The right diagnostic solution gives organizations a powerful method to peer into the application tiers and code in relationship to the individual systems and subcomponents.

If deployed and used continuously, such solutions provide historical context and application-specific details at the exact time of an issue. You can trace issues to arrive at root cause more quickly. This greatly shortens mean time to resolution.

The right diagnostic solution also helps monitor, isolate, and resolve issues and anomalies across complex distributed applications. It discovers bottlenecked components quickly. Most critically, this means visibility and actionable data that enable developers, QA teams, operations and application support

teams to efficiently collaborate, diagnose, and improve application performance.

What Automated Diagnostics Should Deliver

Diagnostic solution need	Examples
Monitoring and data collection	■ CPU usage and service request latency data down to the application subcomponent level (class and method level) ■ Historical resource utilization captured over time (heap memory, threads, resource pools, queue depths, etc.) ■ Capture of application-flow instances (outliers, for example) including multicontainer and distributed flows along with their exceptions
Transaction traceability and application triage	■ High-level business transaction data correlated with low-level service request flows to show problem occurrence in the performance test or synthetic transaction, as well as the corresponding method or SQL query ■ Pursuit of service requests across servers and application container boundaries ■ Identification of memory leaks, hung threads, and equivalent composite application problems
Ease of use	■ Integration with testing and end-user monitoring tools to merge diagnostics data with preproduction testing and production monitoring data. Tests do not need to be repeated for further actionable information ■ Efficient enough to be left on continually during production without degrading application performance

Successful application performance and availability management requires strong collaboration between the preproduction and production teams—and shared diagnostic information is no small part of this. Here again, we see the importance of a complete lifecycle approach.

Is It Secure?

We've stated that too many organizations are trying to meet the modern application world with legacy mechanisms. Perhaps this is truest in the realm of security.

Historically, security was a network threat to be solved by firewalls and the like. It was an Operations problem.

But the modern hacker knows that the easiest route to a company's data stores isn't a frontal assault on the network. Instead, it's to blend in with the gigabytes of trusted, firewall-approved traffic that come through the public-facing applications every day. According to the SANS Institute, attacks via web applications constitute more than 60% of the total attack attempts observed on the Internet.[2]

The Hard Costs of a Security Failure[3]

Average cost per compromised record	Average number of compromised records per breach	Average total cost per data breach
$202	30,000	$6.06 million

Although most IT organizations recognize that the hacker's attention has shifted from network to application, too many hold on to the historical view that security is Operations' problem to solve. This means security assessments frequently do

2. SANS Institute, "The Top Cyber Security Risks," September 2009.
3. Ponemon Institute, "$U.S. Cost of a Data Breach," 2008.

not occur until just before the application is due to launch, and sometimes not until after an application goes live.

As a result, security flaws in the application source code (a significant bug, by any definition) aren't discovered until the latest possible moment. The costs to fix these bugs at that point are sky high. Furthermore, pressures to go live often result in the aggressive backlogging of known security holes that seldom get filled.

The right way to manage application security is to see it in terms of the complete lifecycle. Where the legacy view believes that security tasks are the responsibility of the security team, the modern approach is to make the entire application team accountable. Security vulnerabilities are like any other defect, and prevention and early detection are the watchwords.

What does it mean to thread security throughout the application lifecycle? Consider the following ideal organizational approach:

- **Planning:** Stakeholders assign a security-risk score to each candidate-application investment, based on factors such as audience (internal or external), data sensitivity, network and server placement (proximate to other confidential data?), and any regulatory standards for privacy or security. The risk level drives the necessary security requirements for the application.

  Security teams work with application teams to develop a standard risk-level matrix and a set of standard detailed security requirements per level. For example:

	Risk level 1	Risk level 2	Risk level 3	Risk level 4
Failure to secure	Low risk	Medium risk	High risk	Critical risk
Assessment performed by	Novice individual	Skilled individual	Certified individual	Security professional

	Risk level 1	Risk level 2	Risk level 3	Risk level 4
Teams that must participate	Security	Development and Security	Development, QA, and Security	Development, QA, and Security
Review by Security required	No	No	Yes	Yes

- **Requirements:** Sound security requirements account for authentication and authorization mechanisms, data-access controls, and the validation of application inputs (e.g. disallowing SQL syntax in data fields). The team develops functional requirements that take into account the intentional misuse of an application.

  Techniques to help discover these requirements include *threat modeling* and *misuse cases*:

  - **Threat modeling** examines how the application might be a source for a particular vulnerability (for example, a denial-of-service attack that impacts enterprise application availability).
  - **Misuse cases** document the malicious behaviors an attacker might take to compromise the application.
- **Development:** Developers conduct security-defect source-code reviews as part of the standard code-review process and use automated source-code security-analysis tools to discover vulnerabilities prior to testing.
- **Testing:** Instead of a separate, specialized security test team, all QA members test for security. This generalization is made possible by the right automated security-test tools, which provide for dynamic scans during system and integration testing. As people execute security tests, they log vulnerabilities just as any other software defect, and they follow the same standards for reporting and remediation.

One unique aspect of security defects is their quantity. Automated scans will often unveil long lists of vulnerabilities. However, the right security-test solution includes automated grouping and prioritization of vulnerabilities.

- **Deployment and production:** Pre-deployment security tests ensure that you configure operating system and platform components with secure defaults enabled and all file permissions set appropriately using the secure settings of the application's configuration. For the highest risk applications, perform penetration tests to evaluate the application's ability to thwart a simulated attack from a malicious source. Penetration tests help to identify not only any remaining application vulnerabilities, but also problems in the application infrastructure and configuration.

  After deployment, continually monitor and reassess application security with scheduled scans to account for changing hacker tactics over the (long) life of the application.

Application security bears a close resemblance to application performance. Years ago, performance was seen as a specialized remit (some might say an afterthought) in application delivery. Now, we understand that good performance is a whole team initiative, one that begins with clear and explicit statements about what the business expects. Application security is no different: given the consequences of failure, sound security is an indelible a part of the entire lifecycle.

The Promise (and Pitfalls) of Agile

The renovation by Agile methods of traditional, sequential delivery is clearly one of the main modernization storylines of the last few years. And, justifiably so.

The promise of Agile is (say it with us) to enable application teams to deliver higher quality software faster, and to collabo-

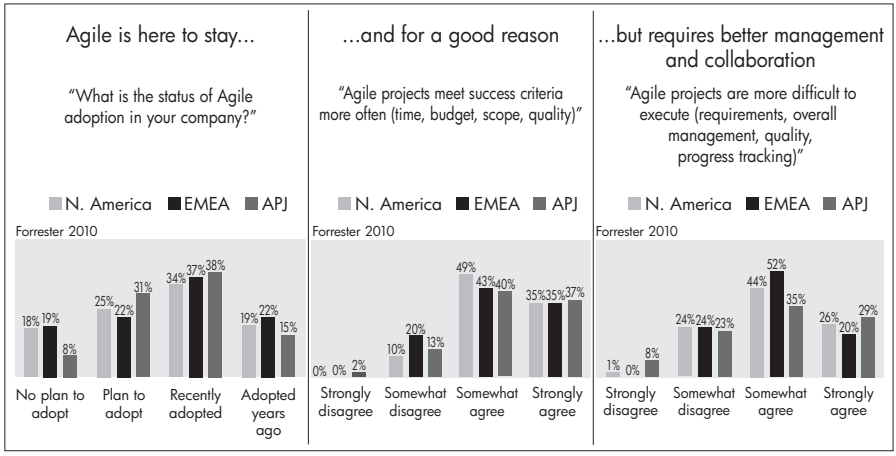
rate closely with the business to ensure the software meets expected business requirements. Agile delivery makes these goals possible:

- Discovery of code defects earlier in the development cycle
- Decreased project risk
- Quicker response to changing business priorities

Few modernization trends could be more supportive of the mantra of predictability, repeatability, quality, and change-readiness.

However, some customers report that the full promise of Agile remains elusive or, where realized, has involved more extensive surgery than expected.

More Than a Fad[4]



4. Figures based on a sample of 206 decision makers with insight into budgeted modernization activities. Source: "Clearing Your Path to Modern Applications and Business Agility." Forrester Research, April 6, 2010. A Forrester Consulting thought leadership paper commissioned by Hewlett Packard.

A Full Embrace

In the race to adopt Agile practices, many organizations suffer from piecemeal adoption. Often developers are first to move into sprint-like iterations, while the business analysts and testers default to their old, sequential places in the project plan. It could be said that while these organizations have moved to Agile development, they have not yet achieved delivery.

As a consequence, the key objective of Agile—early discovery of issues—is thwarted. Completing the code faster doesn't get the application to market faster, or with higher quality, if the rest of the delivery organization (business analysts, QA engineers, project managers) continue following Waterfall practices and timelines.

In fact, this problem of part-Waterfall and part-Agile within a single project is common enough to have earned its own nickname: "scrummerfall." (The joke is that scrummerfall projects fail twice as fast as they would have with Waterfall alone.)

> Agile methods yield results only when the entire delivery team is involved.

Agile as It Should Be	Agile as It Too Often Is
Time-boxed for focus: Brief iterations help the team to prioritize and concentrate work plans.	Unchanged
Hands-on with stakeholders: Instead of existing at a removed, "contractual" relationship, stakeholders are actively involved team members.	Unchanged
Surfaces issues sooner: Comprehensive functional, performance and security testing by iteration reveals risks and issues early enough for remediation.	Unit test is accepted as a proxy for system test; true comprehensive validation doesn't occur until late in the project. The delay leaves the project open to the same Waterfall problems of surprise issues and limited time to fix.
Rigorous, cumulative testing: Testing each new iteration validates the new functionality of that iteration. It also regression-tests the preceding iterations and any legacy-application functionality.	The parts are mistaken for the whole. Each iteration is tested in and of itself, without regression tests to vet the work of previous iterations. The team does not know until late in the project the cumulative application behavior—functionality, performance, security—and has little time to react to problems.
Designed for change: The team works in expectation of change, instead of seeking to paint stakeholders into a corner from which they can't subsequently move without a lengthy request-for-change process.	The spirit is willing, but the flesh is weak: The team is open to change without being prepared for it. The limitations mean that change is introduced with too little appreciation for its impact.
True measure of progress: The complete vetting of application functionality by iteration shows the truest possible measure of progress, which is working code.	The triumph of hope over experience: Without successful test results, progress is a mirage.

Agile Principles and Traditional IT Aims: Friend or Foe?

Perhaps one of the reasons organizations are slower to fully embrace Agile is the fear that it will engender more bad habits than good ones. But if you embrace Agile's premium on things like flexibility and responsiveness, you should not abandon traditional IT aims like consistency and thoroughness.

The truth is, these goals are mutually supportive. For example, proper requirements traceability (as discussed earlier) empowers swift change-impact analysis.

Consider a few other key areas where "traditional" IT objectives that are properly executed—in large part by leveraging integrated management and automation solutions—help to further Agile aims:

- **Speed *and* quality:** To meet sprint deadlines, Agile development teams often short-circuit the testing effort. However, this undermines the goal of discovering defects early in the development cycle, which is perhaps the primary benefit of iterative over sequential methods.

  Companies can balance speed and quality through the use of test-automation tools. Test automation helps validate the functional, performance, and security requirements of each software iteration. Automation allows you to increase test coverage and keep the project moving in sprint time.

- **Flexibility *and* consistency:** Unlike traditional methods that rely on cumbersome and often obsolete project plans, Agile encourages teams to be flexible and responsive to change. However, this can lead to ad hoc project management.

  Companies can strike the right balance by using a management solution that supports sequential projects, as well as Agile methods. This approach facilitates effective user-story definition and management; release,

sprint, and backlog management; and burn-up, burn-down, and cross-sprint velocity charts.

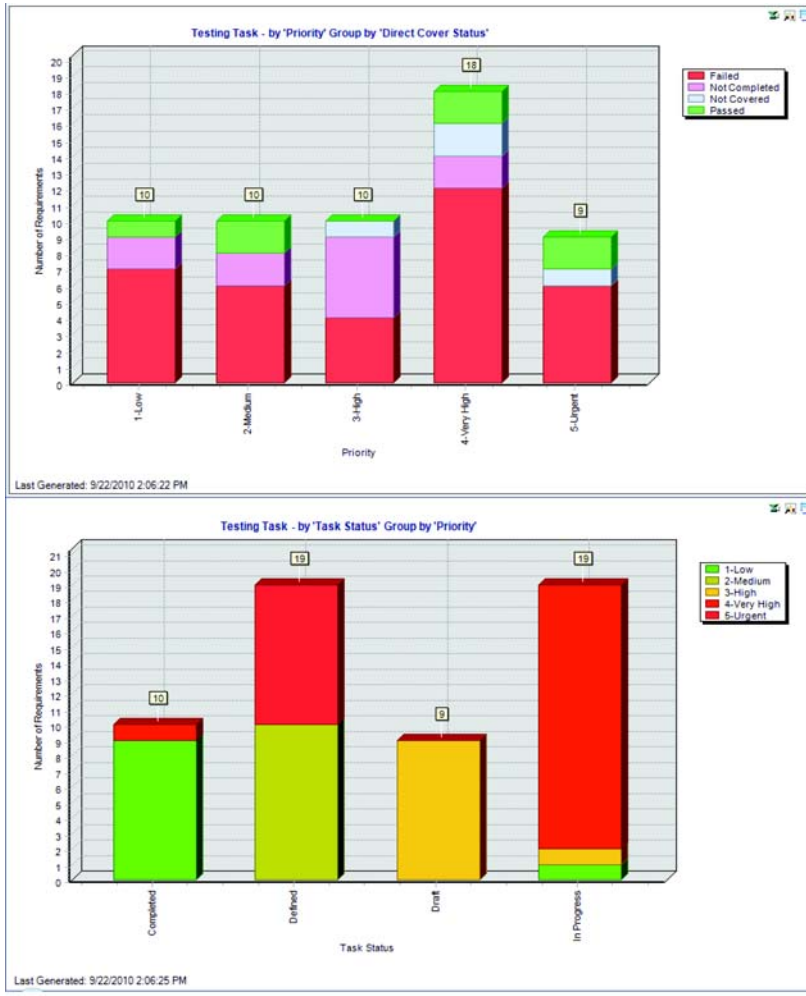


Figure 15: A cross-project view of Agile team progress (source: HP Agile Accelerator)

- **Entrepreneurship *and* economies of scale:** Agile encourages smaller teams and greater autonomy. These objectives are good, but they can trend toward limited cross-team collaboration and higher duplication of effort. To preserve IT's embrace of economies of scale,

delivery teams should use a common repository to streamline the sharing and reuse of project assets and artifacts (user stories, tests, components, known defects).

Have You Fallen Into the Scrummerfall Trap?

An organization can ask three key questions to gauge its overall Agile effectiveness:

- Are my Agile projects discovering code defects earlier in the lifecycle than my traditional projects?

  With Agile, you should always surface issues sooner.

- Am I seeing fewer defects in finished products when I compare my Agile projects with non-Agile projects?

  Agile should improve the quality of a software project, while decreasing the costs of fixes.

- Are business stakeholders generally more satisfied with Agile projects?

  Agile should help business and IT better communicate expectations.

Given the rapid, ever-changing nature of technology, it's probably true that application organizations will always be challenged by complexity. But complexity can be managed. It requires only the right, aggressive response—one that recognizes what modern delivery demands of application teams, and brings integrated management and automation solutions to the challenges.

Chapter 3

RUN: Maintaining the Responsive Live Application

Key Questions:

Application performance and availability

- Will the application function, perform, and be secure when it goes live?
- Are performance problems resolved quickly?
- How do we keep up with changing security threats?

Application change management

- When will the problem be fixed, and when will the enhancement be available?
- Do we have the resources to make the requested change, or will doing so derail a more urgent request?
- How do we know if a problem is really an application problem or an infrastructure issue?

Successful deployment of an application into production is reason for celebration. Whatever the sweat and tears to get it there, the labor of delivery is at an end. But this metaphor of birthing pains is appropriate for another often overlooked reason: The launch of an application represents a beginning, not an end.

When we look at the complete lifecycle, it's obvious that launch may signal the conclusion of the application team's major effort, but for the production teams who must work with their pre-production peers to ensure that the application is responsive to the business, it's only the start.

To qualify as "responsive," the application has to perform well technically, with good response times for all users. It also has to serve the business by changing as needed easily, efficiently, and with minimal risk.

In this context, application performance management and application change management are critical disciplines in running an effective live application.

- Application performance management assures that applications are of consistently high quality.
- Application change management is the key to running applications that are change-ready.

Application Performance Management

For a vivid example of why so many want to dissolve traditional IT silos, consider the usual approach to confirming application performance.

- Preproduction teams work to develop complex, comprehensive test scripts that will realistically mirror end-user behavior.
- When the apps go live, these scripts go into a drawer, saved for future releases and regression tests.
- Production teams create performance scripts to verify application response times in the live environment.

But this lack of sharing works both ways. Real user behavior has the uncanny ability to reveal critical application weaknesses and faulty assumptions that the development teams

would never dream of, much less know to test against. Production teams regularly grapple with this behavior, but the valuable information that they learn is kept in the production silo, out of reach of the preproduction teams.

The delivery team has worked hard to create high quality applications: The applications work as the business wants them to; they perform and are secure. Now it's up to the production team to make sure that the applications continue to meet these high standards.

The first step is to knock down the silos and take a complete lifecycle view of application quality.

Recycling for Fun and Profit: Lowering Rework in Performance Testing

Application performance management focuses on integration, collaboration, and resource-sharing from development to production and back again. This focus means that the hard work invested in building test scripts, as well as performance and quality efforts, are reusable in the management of production performance.

More than just reusing scripts to measure technical performance, preproduction and production teams share logical assets including KPIs and service levels. This sharing ensures that both teams concentrate on the intended business outcomes of the applications they deliver.

For example, if the QA team interviews business management to understand what to test and what the KPIs should be, production support teams should use this same KPI information going forward as their performance baseline to measure against. Such a process saves countless hours and trouble establishing reasonable measurement thresholds.

The information about how real users interact with the application can also highlight functionality and usability issues. If users are taking unexpected or strange paths through a web-

site, for example, this may indicate confusing pages or unanticipated user requirements.

Nothing Like the Real Thing: Optimizing Performance with Real-World Usage

In most cases, the business process under test for scalability and functionality in pre-production is the same business process you have to monitor in production. But in production, there is an additional critical element at play—real users. Until recently, getting user-behavior data in production has been at best impractical and at worst impossible.

Application performance management enables the production team to directly monitor real-user session information of all users to understand how the application is performing for them. If performance is poor or users behave in unexpected ways, the system can generate test scripts from this behavior to feed back into the preproduction testing process. This feedback ensures that lessons learned from user interaction enhance the testing process, which results in applications that perform better in the real world.

Clear as Glass: Fixing Performance Bottlenecks via Shared Analysis

If applications were still centralized, self-contained entities running on one server somewhere on the same network segment the users are on, life would be much simpler. Root-cause analysis on performance bottlenecks could be done very quickly and likely by one person.

Today's applications, though, are constellations of components spread across any number of systems, which makes it extremely difficult and resource-intensive to identify, much less resolve, performance bottlenecks.

A comprehensive application performance management approach can help: It provides a common, detailed view of applications that quickly detects, definitively diagnoses, and

effectively repairs performance problems. Using a shared set of diagnostic and monitoring tools, your team spends less time deciding which tools to use, and more time resolving the performance problems at hand.

The Best Defense Is a Good Offense: Attacking Security Risk

It doesn't matter how strong your locks are if expected visitors do the unexpected. IT has spent untold resources strengthening networks, firewalls, and operating systems only to metaphorically leave the door wide open for hackers to attack at the application layer.

Applications must traverse infrastructure and organizational boundaries to do what they need to do, so they should be subject to security analysis before they go live.

However, the security challenge doesn't stop there. Because threats evolve and as malicious hackers learn new tricks, an application that was secure yesterday may be vulnerable today. Not only is it necessary to scan production systems, you must update the scanning technology to address the latest threats.

A regular audit and remediation of production security issues leads to a consistent understanding of how applications *should* behave and how they actually *do* behave in production. These efforts to secure applications in production also reap continual improvement benefits as resulting actions are again fed directly back into the development and testing processes.

Application Change Management

> The only constant in modern applications is change.

World-class IT groups respond quickly to business needs by keeping their production applications, their processes, and their teams change-ready.

First, Do No Harm: Business Impact of Change

Twenty-five years ago, application changes or fixes were performed without any formal process. The results were unpredictable, and often late nights were spent fixing a change that either didn't work or had unintended consequences.

Unfortunately, many organizations still do not have a robust change-management process.

When the process is ineffective, changes come in the back door, are not formally managed, and have poor or no alignment with business objectives. Improperly implemented or conflicting changes can bring about a further breakdown in quality.

The first step in any change-management process is to make sure that the change will do no harm to the business—either directly by causing problems in business processes or indirectly by diverting resources needed for more urgent issues.

Application changes typically fall into two categories: operational and strategic.

- **Operational changes** address service interruptions, perform standard maintenance, keep the operation running, and institute regulatory compliance.
- **Strategic changes** tend to be major changes that are aligned, feasible, and cost-justified. An example of a strategic change would be a request from the business to IT to provide a new application to manage a new line of business.

It is important to classify whether a change is considered operational or strategic. That classification defines the decision on whether to move forward with the change, as well as the execution and delivery processes.

The change management system can and should help enforce and accelerate the processes you want your teams to follow. The process of modeling, automating, measuring, and enforcing rules is known as "digitizing" the change-management process. Operational and strategic changes will go through different processes, but both can be digitized to enforce and accelerate a best-practice approach to evaluating, prioritizing, and scheduling the change.

Digitization also feeds a demand-management process that gives the organization visibility into the resources available to accomplish application changes. Demand visibility is critical to prioritizing the IT workload and determining the relative business value of various alternatives.

With digitized processes in place and a smooth flow of data about changes, projects, and resources, the production team has the information and real-time visibility necessary to effectively manage status, SLAs, and trends. Tracking the costs and resources of an approved change throughout its lifecycle also provides the information you need for sound IT financial management.

Making Change: How to Manage Application Changes

There are many change-management processes, including those prescribed by the IT Infrastructure Library (ITIL) and other IT best practices. All are designed to reduce risk and rework by identifying dependencies, involving groups that might be affected, documenting changes, and enabling them to be easily undone if they fail.

Without getting into the details of these specific practices, an effective application change-management process should look like a smaller version of the application lifecycle:

1. First, determine how a change will affect business outcomes. If the change will not affect the business or the request has medium to low priority, it can be part of

a larger release managed through program- and project-management processes. If changes may cause critical issues to the business, prioritize and execute the request within a specific change-management workflow, such as an emergency change process.

2. Next, weigh the benefits of the change against the costs of implementing it. Sometimes the right thing to do is to not make the change!
3. Key stakeholders should prioritize the change alongside other change requests within the application portfolio. Checking priorities assures that the most important changes are implemented first, which makes the best use of limited IT resources.
4. When the change is approved, the next step is to update the application's requirements. When requirements are effectively captured and put under change control, it helps throughout the lifecycle—especially for change management in production. When all dependent project artifacts are linked to their sponsoring requirement, it's possible to see at a glance what effect changing a requirement will have on the downstream artifacts.

> Save yourself a lot of heartache with properly managed application requirements — find out *before* implementation if a change will break something downstream.

5. Next, plan the implementation to reduce rework and minimize the impact on users. At this point, you should take steps to comply with applicable regulations and audit requirements. In parallel, start the testing and operational rollout planning processes to define how you will test and deploy the application change. It is critical to have the QA and Operations teams aware of the application changes early in the process to avoid problems or delays in testing or deployment. This

means testing and validating the overall business process, as we described in the previous chapter, not just the underlying applications and subsystems.

6. As you implement the change, test it against its new, updated requirements. You can return it to production when you've verified that it still functions, performs, and is secure enough to meet the needs of the business.
7. Before moving to production, evaluate and compare the deployment of the change with all other changes moving to production to avoid collision and impact to production. Generally, require an authorization to move to production before deployment.

Change with the Times: Automating Change Management

Many handoffs occur in the application change-management process, and each is fraught with peril. Even minor miscommunication may keep a critical change out of the next release.

Automating the handoffs assures that the information exchanged is accurate and timely. A change request that originates from the service desk should automatically update the demand-management system and should trigger notification of the appropriate team members to review change requests.

If the change is approved, the management system should trigger updates to the requirements management or defect management systems that initiate the workflow to expedite the change activities.

Following and automating application change management processes helps lower costs while reducing the risks and impact to the business. It helps meet compliance requirements and provides better visibility into what needs to be delivered so you're using finite IT resources effectively to meet the needs of the business.

Untangling the Knot: Application Mapping

When users experience a problem with an application, the application is not always at fault; sometimes the problem is with the underlying infrastructure. In many organizations, these problems are hard to locate and diagnose, and there is often a lot of finger pointing between the Applications and Operations teams. Meanwhile, the users become impatient while they wait for the problem to be resolved.

Change-ready organizations have this figured out. They avoid "blame-storming" sessions by having processes and technology in place to help them quickly locate the root cause of a problem. They have mapped applications to their dependent infrastructure, which provides the information they need to accelerate problem isolation. This leads to faster problem resolution and happier users.

Application mapping also feeds a configuration-management system that houses a wealth of information for the delivery team. The team can see a list of all live applications and services and compare it to a list of what the team thinks should be there. If there are any discrepancies, they can address them before users experience problems.

The team can view the real configuration of the production applications and infrastructure, which is extremely valuable for the delivery and validation of new versions and releases.

Chapter 4

RETIRE: Letting Go of the Application at the Right Time

Key Questions:

- How do we reduce the cost of maintaining all our applications?
- How can the business retrieve the information it needs if we retire an application?
- Can we retire an application and still comply with legal and financial data-retention requirements?

Today, more than 70% of IT spending goes toward maintaining and managing existing applications. This saps resources that could otherwise help to innovate and grow the business.

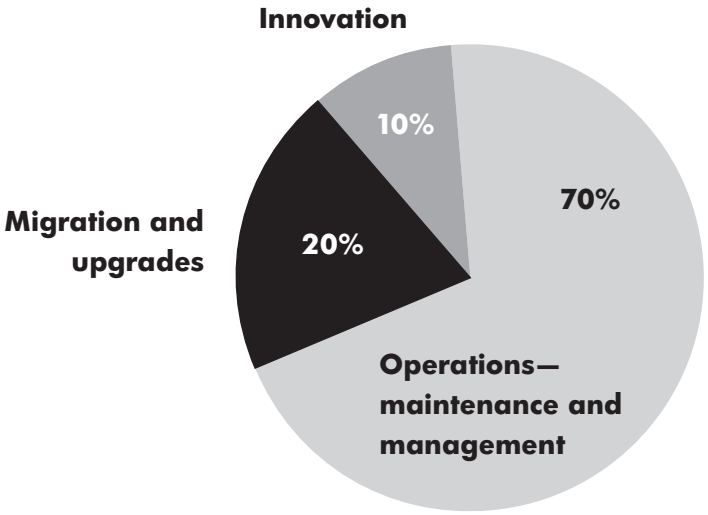


Figure 16: Keeping the lights on at the expense of moving the business forward (source: IDC/Alinean data)

Why does so much money go to operational maintenance? The answer lies almost entirely in the sheer proliferation of applications in the average portfolio. According to some estimates, application portfolios expand at an annual compound growth rate of 4% to 7%!

The costs associated with these growth rates become apparent when we consider the basic relationship among business information, applications, and infrastructure. As we launch new applications to support new business initiatives, we need more infrastructure to support them.

Because applications are repositories of critical business information, data storage puts its own demands on infrastructure. Gradually, the demand for information and applications leads to a self-perpetuating sprawl.

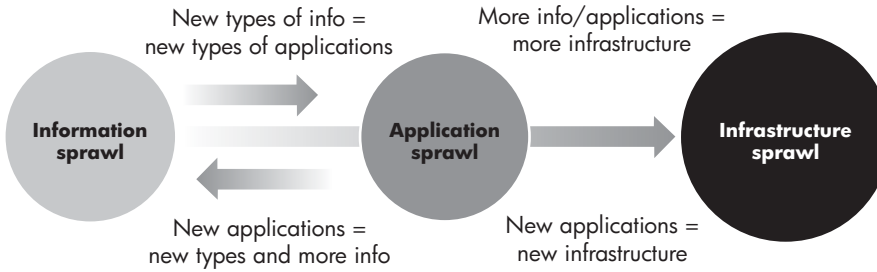


Figure 17: The sprawl that ate IT

Enterprises hold on to applications well beyond their useful life.

A Basement Full of Junk

Most organizations are fairly efficient at managing infrastructure assets. We understand that equipment should be disposed of when it becomes obsolete.

But for many organizations, this principle seems to apply only to hardware. The rigor of knowing what we're using, understanding its value, and retaining it for the right period of time hasn't yet translated into applications. More and more new applications are brought into production, while legacy applications seem to never go away.

Hanging onto applications beyond their useful lifespan creates challenges throughout the organization:

- **For business users:** Legacy applications make it difficult to change to new business processes, which can limit agility in the marketplace. Older applications often have a higher probability of failure, and they pose a high risk to the business as a result of legal discovery.
- **For the IT organization:** Legacy applications are difficult to support. The IT organization may no longer have the skill sets to maintain older applications, and inflexible application architectures make it hard to keep up with changes the business requests. Disaster recovery of

application data is riskier for older applications. It is increasingly difficult to bring older applications into compliance with changing security and user access requirements.

- **For the legal department:** Legacy applications increase risk. Searching for data related to legal matters in legacy applications can be very difficult. It's also challenging to manage legal hold-compliance orders for e-discovery.
- **For the comptroller's office:** Legacy applications contribute to information sprawl. The comptroller's office, which is responsible for managing data for compliance purposes, is overwhelmed with huge quantities of information. It's concerned about losing legacy data through records that can't be searched. There is often no lifecycle management of information in legacy applications and no documented destruction of data. All of these issues can lead to regulatory agency fines for noncompliance.

The corollary benefits of regular application retirement are also clear.

- Reduced risk
- Better data access and security
- More flexibility and lower overall costs

Not surprisingly, application retirement usually pays for itself. This begs the question: Why doesn't the enterprise just turn off outmoded applications?

There's a one-word answer: Data.

Companies retain the majority of legacy applications to provide access to information: historical data that the business may need, data for legal e-discovery holds, or records for governance and compliance purposes. If the enterprise has no clear strategy for the appropriate preservation of this data, the application remains.

Breaking the Hoarder Mentality

To retire an application, you first have to extract records into either a production database that is logically suited to receive them or into an archival data store. Either way, the records should be written so they are correctly preserved for audit and compliance. After all the application's records have been migrated, the application can be shut down.

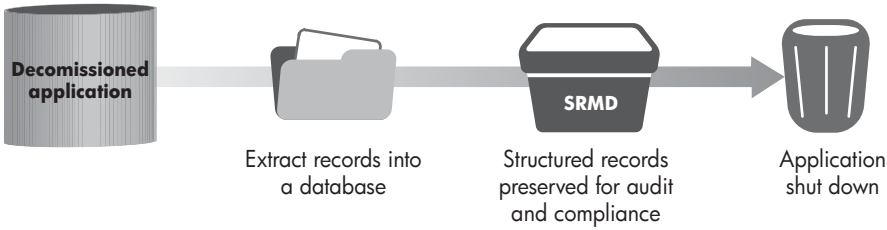


Figure 18: Extract and preserve records before retiring an application

To avoid information and application sprawl moving forward, you must have information-management processes in place, particularly for archived data.

Information in the organization has three primary uses: It has business value, compliance value, and possibly legal value. If it has value in any of these areas, IT needs to ensure that the information is available, searchable, secure, protected, and managed in a cost-effective way. If it no longer has value in any of these areas, then it represents risk and cost with no value to the organization, and it should be defensibly disposed of as soon as possible.

The key here is that information that has business, compliance, or legal value today will likely not retain that value forever. Thus, managing information is not a one-time process, but rather an ongoing best practice.

The lifespan of a typical business record generally has four stages:

1. **Transactive:** Initially, when a business creates a record, the business is the primary user. It accesses the record frequently and modifies it often. In an online shopping-cart transaction, for instance, the record will be very active as an order is booked, payment is made, and the product is shipped. The record will need to remain active during the time that the customer may have an inquiry, but after that, the business will need to access it much less frequently.
2. **Reporting:** At this point, the transaction is over. The information has become inactive, and it is static and unchangeable. The business still needs the information for reporting purposes, but accesses it much less frequently than during its transactive stage.
3. **Compliance:** When the record enters the compliance stage, business users no longer need day-to-day access to the information. Instead, specific events triggered by compliance, audit, tax, and legal requirements dictate whether the business accesses the information at all. At this stage, records managers, tax, and legal users need a simple search mechanism to find this information when their job calls for it.
4. **Destroy:** At the end of the compliance phase, the organization no longer has an obligation to keep this information, and it should defensibly dispose of it.

Most organizations keep the record of a sales transaction forever, and they keep it available on their most costly, tier-one infrastructure. Forward-thinking organizations take the opportunity to leverage different storage tiers for information, depending on its stage in the lifespan. You can store static information that is accessed less frequently on less expensive tier-two or tier-three infrastructure. When the business no longer needs the information, the information should be destroyed.

However, organizations can no longer just delete information. Disposal must be based on a documented policy, preferably highly automated, with a full audit trail.

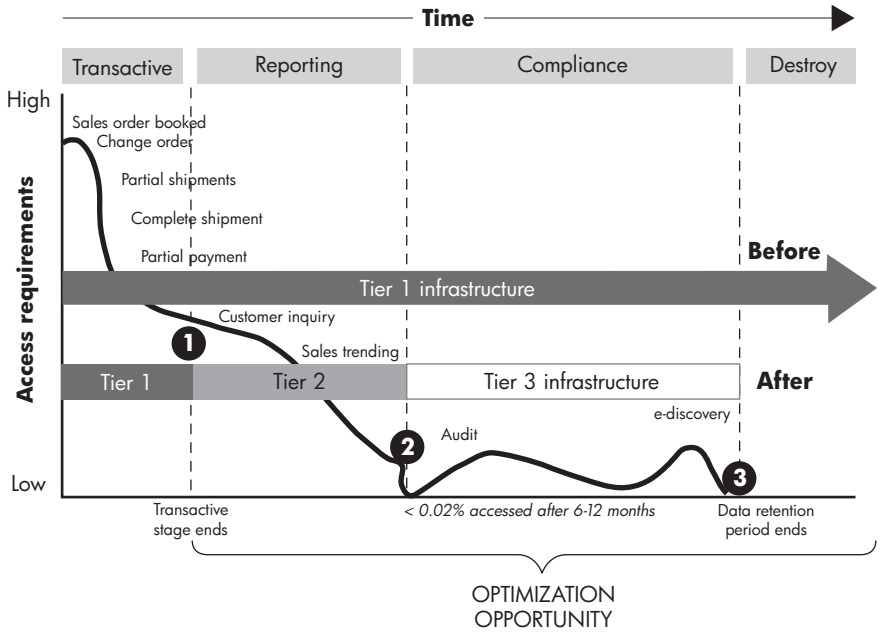


Figure 19: Matching record-retrieval frequency to infrastructure cost

When an organization decides to retire applications as part of a broader application portfolio rationalization initiative, it also chooses to keep a number of applications in production. The applications it keeps may need to take on additional functions of the retired applications. Or, if applications with duplicate functions are eliminated (such as after a merger or acquisition), the applications it keeps may need to serve a wider audience of end users.

Thus, the applications that remain in the portfolio may need to be updated to function, perform, and be secure enough to meet the needs of the business.

And this returns us to the beginning of the book, where the application lifecycle begins anew.

Conclusion

Application Lifecycle Maturity: How to Get There From Here

You've probably noticed a couple of themes in the preceding chapters. One theme is that without effective, efficient application delivery, real business outcomes are hard to achieve. Another theme is that organizations gain effective, efficient applications through modern methods, but only when the organization lets go of legacy mechanisms for delivery, and embraces integrated management and automation.

So, how does an organization begin to assess its own application lifecycle maturity?

Lifecycle Maturity Affects Business Outcome

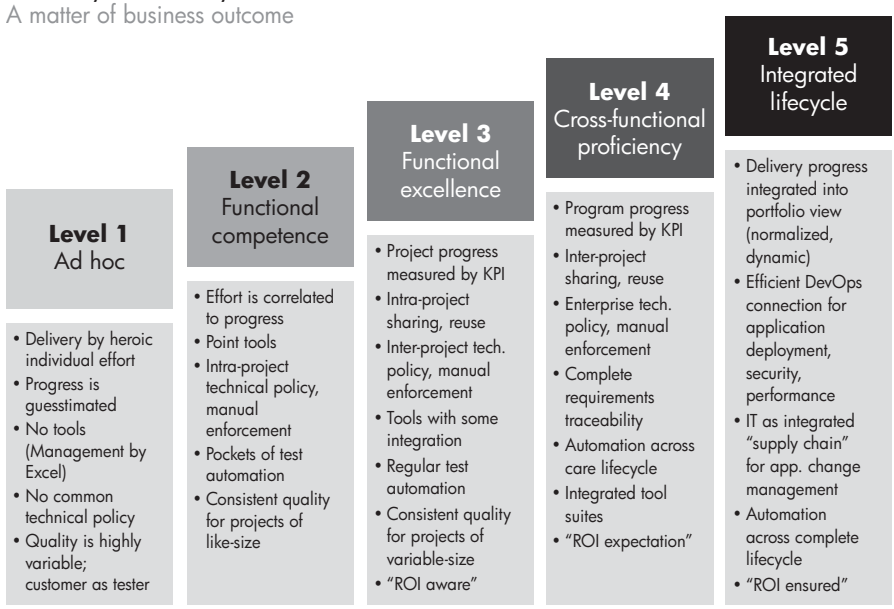
You'll find plenty of certifications and scoring mechanisms to help your organization benchmark its application delivery against best practices (ISO, CMMI, Six Sigma, etc.). What's often missing from that score is business outcome.

Consider an organization that ranks in the upper quadrant of delivery efficiency, but time and again builds the wrong appli-

cation. "Wrong" here might mean a new application with functionality largely redundant with another somewhere in the enterprise. Or, the money spent would've shown better return on a different application.

The real measure of application lifecycle maturity is not purely better, faster, cheaper. It must also include some calibration against business result.

HP has experimented with its own model for gauging application lifecycle maturity. It's not heavily prescriptive, and it's oriented around what we believe are the ultimate aims of world-class application teams.



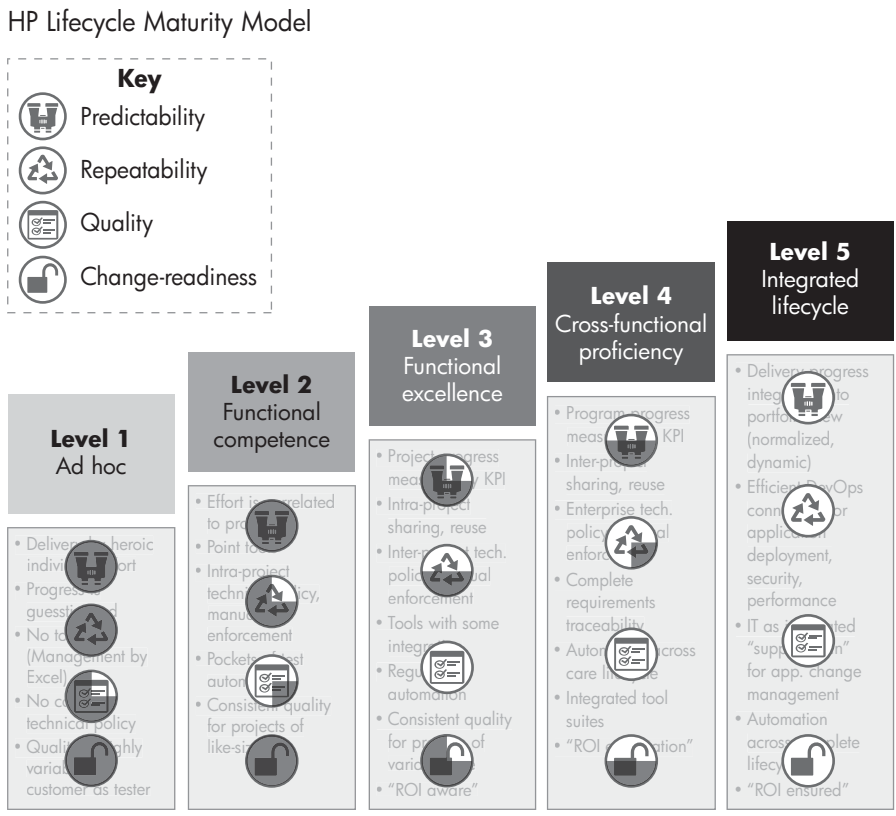
Though the details of each level matter, the trajectory represented by the different levels is important. What's happening with the business is more than what the individual bullets represent.

Organizations need to develop basic competence in the key functional disciplines (requirements, development, test, project management, etc). With persistence and the right investment,

the organization moves from functional competence to functional excellence.

We know, however, that functional excellence alone is not enough. Modern delivery demands a move away from legacy silos, and toward integrated, cross-functional teams. Of course, this means breaking down not only intra-organization silos like business analysts and testers, but also inter-organization silos, such as those that exist between Delivery and Operations (as the "DevOps" movement has recognized).

In the Chapter 3, we established that world-class delivery organizations share four traits: predictability, repeatability, quality, and change-readiness. Applying these objectives to the maturity model, we see a progression something like the following.



The first objective, rightly, is application quality. Little else matters when the application fails to behave as its sponsors expect.

Generally, as the organization matures, we see repeatability is the next trait the organization achieves. That is, we find less dependency on individual heroic effort, greater consistency in process, more automation, and more asset reuse.

Predictability and change-readiness enter the picture shortly thereafter.

At the highest end of maturity, we see what might be described as an integrated "supply chain" for application change. This means the walls separating the teams that plan, deliver, and run applications have come down.

Companies at the summit consistently deliver superior returns on IT investment.

Some Good Indicators that You've Arrived

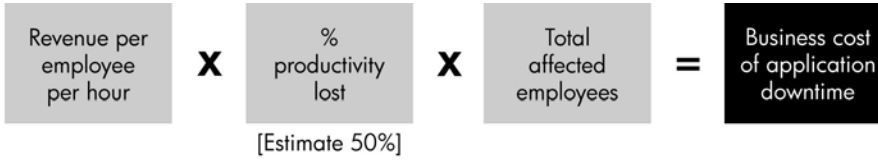
Fortunately, you can begin to assess your organization's application lifecycle maturity without complicated models around ROI or percent of IT spent per revenue dollar earned. A few top-line indicators help predict maturity (or lack thereof) in an organization's lifecycle approach.

- **Business satisfaction.** Personal relationships go a long way in cementing IT's connection to the business. However, no amount of friendliness can cover up slow, misconceived, poorly executed, or poorly supported applications. A relatively simple survey of the customer will tell the tale.
- **Defect leakage.** This refers to defects discovered in production. Because quality is an encompassing measure, which points to the work not only of testers but the entire delivery team, a low rate of defect leakage reflects fairly directly to core-lifecycle effectiveness. Despite

this, we find many organizations cannot reliably assess their rate of defect leakage. (And frequently the delivery organizations and service desks give very different estimates!) As a useful, if rather worrying, point of reference, the industry's average rate of defect leakage is just over 40%. In other words, in the average company, more than 40% of the defects raised against a given application are first encountered in production. By contrast, world-class organizations achieve defect leakage rates of less than 5%—a reduction of more than 80%.

- **Revenue impact from application mistakes.** Nearly everything the business does is powered by applications, so application downtime impacts revenue, directly or indirectly. For a back-of-the-napkin calculation that helps determine this impact, use A Rough Guide to Application Downtime, below. Note that this rough math does not include the time to find, fix, and retest the problem. Understanding revenue impact that arises from application failures, even if the numbers are only directional, can help application organizations to justify improvement initiatives.

A Rough Guide to Application Downtime



- **Mean time from change request to production.** This is a good way to start gauging the efficiency of your complete application lifecycle. Why? Because changing a production system touches Operations (usually both the service desk and deployment team), as well as the affected line(s) of business that generally must prioritize

and finance the change. To further simplify the metric, you might begin measuring only the mean time for enhancements of a certain size, for example, 50 staff days or less.

Taken together, these indicators can form a simple scorecard for lifecycle maturity.

It Wouldn't Be a Business Book Without Next Steps

We've tried for a comprehensive a view of the application, looking not only at how you can improve the core delivery lifecycle, but also at what happens before requirements and after deployment. The long life on either side of development forms the complete application lifecycle.

Maybe you've found a few principles here that practically cry out for immediate use in your own enterprise. Maybe the immediate way forward is clear. We hope so! But it's also possible that you've found yourself thinking: So many possibilities for improvement, where do I start? If so, we recommend that you:

- **Think big.** This isn't a vote for endless whiteboard utopias, but it does highlight the imperative of starting with the end in mind—and making sure that key stakeholders agree with the destination. If you had to pick one business priority above all others, what would it be? Greater agility? Improved throughput? The old standby of lower costs? You might fix on a metric, however rough, to gauge the problem and targeted improvement. For example: We will reduce the average time-to-production of change requests by 10% for large changes, 20% for medium changes, and 40% for small changes.
- **Start small.** This is important, and it is an antidote for analysis-paralysis. Don't let establishing the perfect goal stand in the way of improvements today. Maybe testing

applications for cross-browser compatibility is maddeningly labor-intensive work with a lower yield than other manual activities, and it is one more thing that stands in the way of improved throughput. Such work is ripe for elimination by automation. (If you're not sure of the full range of automation possibilities that exist, challenge vendor partners like HP to provide some.)

- **Scale quickly and adapt.** This connects thinking big and starting small. By picking a few relatively easy first steps and knowing roughly where the path ends, you can plot the remaining course. The important thing is not to let the perfect be the enemy of the good. To borrow from our friends in the Agile community, favor adaptability over endless planning. Concentrate on moving pragmatically toward the goal, adjusting for reality when necessary, and know what stands between you and the next step. Application teams care more about measurable progress than the elegance of end-strategy or tactical brilliance.

Where once the enterprise's overriding concern for the application was stability, the new demand is for agility. The prevalence of software means the modern enterprise is only as nimble as its applications.

Modern delivery trends hold the promise of agility, but only when modern solutions help you master them. Legacy means won't produce modern ends.

We hope that something in these pages has struck a chord, sparked a thought, or otherwise shown that application organizations need not drown in complexity. Remember, your importance to the business has never been greater. Indeed, there's no business without you.

About the Authors

Brad Hipps is senior manager of solutions marketing for software at Hewlett Packard. Before signing on with HP, he was senior vice president of service delivery for EzGov, a company that builds and manages enterprise web solutions for national governments. Previously, as a manger for Accenture, Brad worked on large-scale process re-engineering and systems development in the United States and Europe.

Mark Sarbiewski, is vice president of product marketing for HP Software. Before joining HP, Mark was senior director of products at Mercury Interactive and vice president of marketing for InterTrust Technologies Corporation. He was a principal consultant for Pittiglio, Rabin, Todd & McGrath, and put in his time as an application software engineer for IBM. In 2008, Mark co-authored *Optimize Quality for Business Outcomes: A Practical Approach to Software Testing for HP*.