April 2004

The healthcare industry: Vital signs for software development

Plus:
- Transitioning from waterfall to iterative development
- The challenges of enterprise transformation
- Eight steps to agile use-case generation
- Introducing the project pyramid

more inside...
Over the past year, IBM has placed greater emphasis on leveraging its industry-specific expertise as the sales and services organizations reach out to customers. You may have read about this strategy in business journals, or in the popular press. But this isn’t the only reason that The Rational Edge occasionally publishes articles about software development issues in a given vertical industry, like this month’s cover story on the healthcare industry.

There are two more interesting reasons: 1) these articles offer insight into things that affect all of us (what touches us all more universally than healthcare delivery systems?); and 2) they illustrate how software development professionals are tackling problems that aren’t necessarily unique to their business domains. Whatever industry you work within, I think you’ll find this interview with two IT experts engaging.

And there’s lots more listed in the contents below!

Happy iterations,
Mike Perrow
Editor-in-Chief

PS: Check out our archives! Click on the link at right, and you can select articles by topic category, or by issue.
business goals.

- **Come to IBM Rational Software Development User Conference 2004...**

  Information about an exciting new conference for the IBM Rational and developerWorks Live! communities.

Rational reader

- **Software Development for Small Teams (A RUP-Centric Approach)**
  by Gary Pollice, Liz Augustine, Chris Lowe, and Jas Madhur
  Reviewed by Ben Lieberman
  Lieberman appraises a book whose authors describe how they successfully adapted RUP to yield an agile process that is effective for small development teams.

- **Critical Testing Processes**
  by Rex Black
  Reviewed by Susan McVey
  McVey reviews a book outlining a four-step process for managing test projects that focuses on early planning and preparation.

- **Building the on demand business: Four imperatives for improved software development**
  Written for the business manager working closely with an IT organization, this paper describes four essential concepts for creating and managing software applications in the on demand computing era. These four "imperatives" — develop iteratively, focus on architecture, continuously verify quality, and manage change and assets — are based on IBM Rational's time-tested best practices for software development.

- **Requirements: An introduction**
  by Scott McEwen
  Accurate requirements are an essential part of the formula for software project success. This article explains why, and describes a three-fold approach to effective requirements documentation.

- **IBM Rational's Bran Selic to demo UML 2.0 at OMG**
  OMG workshop to showcase model-driven architecture (MDA), scheduled for May 17-20, 2004.

Theory and practice

- **The project pyramid**
  by Joe Marasco
  Joe Marasco proposes a model for predicting project success based on a familiar geometrical shape. Probability of success is measured by a pyramid's altitude, and four production factors — scope, quality, speed, and frugality (or limits on resources) — form the four sides.
Healthcare payers and providers: Vital signs for software development
An interview with Tony Bosselait of IBM and Eric Brown of Forrester Research

Scott Cronenweth
Reporter for The Rational Edge
1 Apr 2004

from The Rational Edge: This interview with two industry experts explores both the differences and overlaps between healthcare payers and providers with respect to software development.

The healthcare industry is driven by the symbiotic yet often conflicting needs of consumers, care providers, and both government and private insurers. As we are reminded almost daily, consumers have been seeing steady increases in healthcare costs despite strong pressure from government and business groups to improve provider efficiency. After bouncing back from its financial troubles in the '90s, the private health insurance industry may be pushing the hardest; currently it is one of the most important forces shaping the business strategies, and hence the IT operations, of healthcare service providers.

Resistance among these providers is high, however. As unfilled beds, rising labor costs, and diminishing returns from Medicare and Medicaid reimbursements continue to squeeze many hospitals, IT investments often seem too expensive — and too risky.

To find out more about the complex forces shaping the IT and software development space in this turbulent industry, The Rational Edge asked writer and consultant Scott Cronenweth to interview two industry experts: IBM’s Tony Bosselait, business executive for worldwide healthcare industry software sales, and Eric Brown, research vice president for healthcare at Forrester Research.

Scott Cronenweth: In your view, how are healthcare industry market forces playing into IT and software investment decisions for payers and providers today?

Tony Bosselait: To answer that question properly, we have to look at the two sides separately, so let’s begin with the payers. Although payers went through tough times in the mid-’90s, as a whole, these businesses are now doing fine and making money. They got their collective act together in managing reimbursements and pushing costs back on us, the consumers. For most of us, annual health insurance premiums keep increasing significantly each time we renew them. So payers have largely stabilized their businesses.

However, there will still be mergers and acquisitions over the course of this year. Companies focusing on expanding their customer base may buy their way into either the consumer or business markets. And those who want to enhance their back-office processing capabilities may buy another company that has made back-end processes a core competency. Either way, they’ll make fairly substantial IT investments. The payers are not unlike the rest of the insurance industry and banking in this respect.
Eric Brown: One result of that recent business stability is that payers now hold most of the cards. On the provider side it’s a little ironic that, particularly in competitive markets, the most potent agents of change are the demands of the payers rather than those of the patients. The payers have the power to reward or punish providers via differential co-payments and higher negotiated rates for procedures.

Now, payers want hospitals to invest in IT. They want these facilities to integrate their various isolated data repositories and begin using patient data to deliver care more efficiently. If a hospital did that, then any person sitting at any kind of terminal in any department or branch facility could see at a glance that Mr. Jones doesn’t actually need another X-ray today because he just had one yesterday.

Of course, hospitals have a different perspective on this. They feel like the payers are leaning on them to invest a lot of money in IT at a time when they’re already squeezed by disappointing revenues. They also fear that infrastructure changes like those we just described might actually reduce revenues even more. In reality, however, if providers can show payers that they’re putting technology in place to deliver services more wisely and with less waste, the payers will balance that out in future contract negotiations by compensating them more generously for the procedures they do provide.

Consumers will help pay for these changes, too. If providers can demonstrate that they’re delivering better quality care at a lower cost, that data will start working its way into the public domain. And the people who are asking, “Should I go to Hospital A or Hospital B for my knee surgery?” will say, “Oh, look, Hospital A does more of these procedures, they have better outcomes, and people tell me you don’t have to wait around for five hours in a hospital gown to get into an operating room. I’d rather spend my money there.”

Tony Bosselait: What Eric says is true, but it’s hard for providers to take a long view on investing because they’re so strapped right now. Hospitals in many communities are still suffering from over-capacity — too many beds or too many doctors. They’re also suffering from limited reimbursements for the procedures they’re doing. The federal deficit is curtailing reimbursements from Centers for Medicare and Medicaid Services, for example; and meanwhile labor costs continue to soar. In particular, the nursing shortage means hospitals have to offer more incentives to recruit and retain critical staff. So these institutions are getting squeezed from all angles, revenue-wise. These folks don’t have the cash to buy applications, or do much in the way of IT improvement that would result in more efficient care or business processes.

Scott Cronenweth: With payers in a stable position and cash-strapped providers getting pressured by payers to make changes, how are these respective IT environments likely to evolve — or not — in the near future?

Tony Bosselait: At IBM, we see payers using their IT budgets first and foremost to Web-enable their existing applications in the area of claims and, to a lesser extent, sales. It’s cost-effective for them to process as many claims as possible in a straight-through, automated manner. When a hospital or doctor submits a claim, ideally you want it to be adjudicated and paid without a person ever having to touch it or make a decision about it. Some payers are processing between 70 and 80 percent of their claims this way today, and we expect to see payers continue investing in application development that helps to automate the whole claims processing flow. Likewise, as some firms build core competencies in this area, we’ll see more outsourcing in the area of claims processing. Either path will help companies drive down people costs and substantially improve revenue cycle time.

Web-enabling the sales side of the business is also about reducing people costs. And it has the valuable side-effect of driving integration work among sales, marketing, risk analysis, and other related systems. We began to see this in 2003, and I think that trend will accelerate dramatically in 2004. Payers want to move to an online a la carte or buffet style, menu-driven selection of healthcare product choices. Using a wizard or something similar, consumers will step through queries about the size of their family, income, age, elder care needs, and so forth. Then up will pop a menu of choices with associated cost information. Humana is a leader in this area right now, and “the Blues” are catching up rapidly.

Eric Brown: Even with their tight budgets, providers are also doing integration within their IT infrastructures as a long-range cost-reduction measure. But this work is about basic integration of isolated, core business systems: tackling the kinds of problems most other businesses addressed ten years ago. I sometimes refer to this as the ERP-ization of healthcare. Hospitals very much need to integrate clinical information from a wide variety of departmental systems into a single clinical data repository, from which they can generate a master patient clinical record.

Most hospitals lack the internal IT skills and resources to create these kinds of complex systems, so instead they buy software essentially off-the-shelf from their vendor of choice for other clinical systems: Siemens, McKesson, Eclipsys, and so forth. There are probably fewer than ten good-sized vendors in this space. This “clinical database of record” then becomes the foundation for further integration involving, perhaps, emergency room systems or lab systems, also purchased from that vendor. Some hospitals want additional capability: for example, to provide graphical status information to the ER or the nursing stations, or even to doctors outside the walls of the building. So they ask their key vendor to help them solve their integration problems and build those additional functions for their master clinical record system.

Tony Bosselait: It would be a challenge to overstate how desperately some hospitals need to do the kind of ERP-style integration work...
that Eric is talking about. These companies began buying packaged applications long before that became fashionable in the rest of the business community. They have a long history of buying whatever package came with their operating room providers' system, their lab providers' system, and so on. The problem is that they’ve overdone it. Most hospitals now have a proliferation of disparate, stand-alone packages — sometimes as many as fifty — and the management problems are huge. My wife, who's a pediatrician, recently switched hospitals. On day one at her new job they gave her nineteen different system passwords! That gives you a sense of the spectacular lack of interoperability. So these folks are finally beginning to consolidate with help from their key suppliers, in an effort to make their lives simpler and bring down costs.

In most cases they’ll go to their CIS (Clinical Information System) vendors and consolidate those systems first. That’s about half the battle. Then, after they get through that initial round of consolidation, they’ll begin using systems integrators (SIs) to tackle the other 50 percent. Unlike, say, Lockheed in the government sector, there isn’t really an 800-pound gorilla in the healthcare SI arena; no one vendor will automatically take the dominant share of that remaining work. Many hospitals already have relationships with local or regional SIs, but bigger firms like IBM have a lot to offer in this area, and we view this trend as an opportunity.

What’s also interesting is that many of the ISVs with Web-based offerings have gone into hosting in a big way. Vendors like Siemens or Cerner host their own applications in their own data centers for up to 60 or 70 percent of their customers. Hospitals may be slow to adopt some technology, but they bought into this concept a long time ago. The ability to spread costs out incrementally based on usage is very appealing to them. Hosting also requires fewer people and less capital — and many hospitals are short on both.

**Scott Cronenweth:** Integration work and putting Web front ends on core systems are critical activities for IT, but do you foresee a demand for more innovative development work in the near term?

**Eric Brown:** Oh, there’s plenty of innovative work going on now. On the payer side, the big focal point for innovation right now is portals. Every health plan — whether they outsource the development work for this or do it on their own — is going to need about five portals (if they don’t already have them).

- First, they need a member portal so that subscribers can get online pre-authorizations, obtain plan and pricing information, or download forms.
- Second, participating physicians need a portal so they can file claims, check on claims data, track patient data, submit reports and special requests, and so forth.
- Third, the subscribers have administrators that pay the bills, so the benefits people in these companies need a portal to enroll and unenroll employees, check on premium payments, stay up to date on plan changes, and share information with plan administrators.
- Fourth, these health plans also have agents and brokers who need a portal so they can get up-to-date information about their customers and plan details.
- Fifth, the plan’s employees need a portal that provides visibility into all of these other groups’ activities and data, so they can field questions, provide assistance, troubleshoot, and so forth.

I’m not talking about online brochures here — these portals have to be robust, transactional sites that provide clear, seamless visibility into complex, mission-critical back-end processes. And as long as a company has back-end processes that are manual, it can’t deliver the necessary value to all of these constituent groups through its Web site. So building these portals — and configuring all the systems and hardware that sit behind them — will entail a lot of work and ingenuity on the part of IT and/or outside vendors.

**Tony Bosselait:** I agree that portals will be a growth area. Payers want to present a unified look-and-feel to customers and partners, and provide comprehensive, unified data access to all their stakeholders. Some payers that built their own portals a while back are looking closely at enhancing or replacing them with off-the-shelf capabilities, as their homegrown systems become difficult to maintain or can’t scale. This is an area where we see a great deal of interest in applying model-driven development. There’s a lot at stake, and if portal designers don’t plan their architectures carefully, they can bring misery to a lot of people’s lives.

Providers are also very interested in portals, particularly chains with upwards of twenty or thirty hospitals. They know they could deliver enormous value to patients, doctors, technicians, and clerical staff via that kind of interface. Doctors, for example, are becoming more and more technically savvy as a group, and they’re currently beating the tar out of their IT shops. They don’t want to fuss around with green screens; they’re asking for unified patient profiles and records as well as immediate access to critical test results, a decent scheduling system for critical procedures, the ability to communicate effectively with patients and colleagues, reasonable access to Web-based resources and references, and much more. Unfortunately, a lot of institutions simply aren’t equipped to deliver on these requests. They’re short on IT staff, and the people they have aren’t highly skilled integrators or developers. Nor do they have money to buy outside development services.

**Eric Brown:** It’s true that hospitals are not hotbeds of innovative, ground-up application development these days. Although a few of the leading academic medical centers have very, very strong IT departments that strive to be thought leaders in the industry. So they might write their own core systems — like a longitudinal medical record (LMR) application to consolidate patient data over time — instead of...
modifying off-the-shelf products. But for the provider side of the industry as a whole, what we’re seeing is the maturation of mainline packaged applications. And this is actually a positive trend, because it provides a stable, somewhat standardized foundation upon which the industry will eventually build more innovative systems. As Tony said, most hospitals won’t attempt to build these kinds of applications themselves; their IT people don’t have the skills, and it’s not their core business.

But it’s also common knowledge that hospitals are notorious for moving rather slowly to upgrade their technology infrastructures — even those with the means to do it. One reason is that competition in this industry is far from cutthroat. When a manufacturing company makes an IT investment, they do it because they want to beat out their competitors — slash costs, deliver shareholder value, all that gritty business stuff. A hospital, in contrast, doesn’t implement a new clinical system because they want to put the other hospital in town out of business. They’d certainly like to be more profitable than the other hospital, but that rabid thirst for new technology just isn’t there like it is in some other verticals.

Tony Bosselait: I believe there’s another touchstone for innovation among both providers and payers besides portals. We’ve mentioned in passing the ability to access consolidated, integrated patient records. This is actually a huge proposition, and the top-tier medical centers — the Dukes and the Mayo Clinics — are currently working on applications that would make providers, rather than payers, the gatekeepers for patient data. They’re building patient record data warehouses and datamarts, which they can then make available to physicians, researchers, and pharmaceutical companies, as well as payers and other providers. These institutions now recognize what retailers like WalMart and L.L. Bean have known for years: that gathering data is worthwhile if you can examine it for trends.

In addition, some of the most innovative payers are reaching out to the physician community and sharing that data — helping providers learn from it. Their attitude is, “With this data we can collaborate and both do better; we’ll know what to pay, and you’ll know what we’ll reimburse.” Of course, secure, Web-based applications are the vehicles for this kind of data exchange.

Eric Brown: And in the end, if the sharing is genuine, the question of who should “own” patient data might be a moot point. I certainly don’t view the payers as an Evil Empire. Right now, if it weren’t for the research that payers do, many hospitals might not be aware of how effective their care is relative to other hospitals.

In addition, very few hospitals have built unified clinical data repositories that provide longitudinal patient history — and here again, the payers have filled part of the gap. A customer’s claim history provides an approximation of an electronic health record — though it’s pretty incomplete. It shows when patients had their last two or three check-ups, what medications they’ve taken, tests they’ve had done — there’s even an emerging effort among payers to capture lab data when they pay for a test. So if the patient goes to a different doctor down the road, she can give her payer permission to share past test data with the doctor. This would surely save us all money.

Of course, there’s potential on both sides to provide more proactive care via electronic systems. Imagine how great it would be if your health plan could notify your co-worker with diabetes to refill his insulin prescription because he’s about to run out, and go get an eye exam because he hasn’t had one in two years. That would help keep your co-worker’s condition stable and possibly save the payer about $20,000 to $30,000 in emergency room and treatment fees. Some of the most interesting custom application development is aimed at providing these kinds of services, using data mining techniques, neural networks, and previously untouched data sources, such as prescription claims, self-report data, demographics, and pay-out data.

Scott Cronenweth: What about distributed applications for healthcare administration work? What technology — and, by extension, which vendors — do you predict will garner most of that business?

Tony Bosselait: Like everywhere else in the hospital IT realm, ISVs are paramount here, and they coalesce into two camps. Those in the...
provider camp are moving their applications to Web environments — either Java or .NET. But in the other camp you have vendors selling old-style, proprietary, client/server applications. We’re talking fat clients — code installed on 500 workstations in the hospital to do function X. It’s amazing how many of these systems they can still unload on healthcare providers simply because they sell them dirt cheap, and capitalize on the customer’s existing system investments. To providers that have no cash, an antiquated system might look like the best option.

Institutions that can afford to evolve their infrastructures make their choice between a cross-platform, Java approach and the proprietary .NET approach based primarily on which technology their favorite vendor — or systems integrator — uses to build its systems. Or how heavily they’re already invested in Microsoft. Many of the leading ISVs, such as Cerner and Siemens, have embraced Java and are well along in delivering robust distributed applications for healthcare. Others, like Eclipsys, are clearly in the .NET camp. And then there are the many ISVs who have yet to commit to either technology; people like me and my counterparts at Microsoft and Oracle invest considerable energy in courting those folks. Will one side win a decisive victory? I very much doubt it; my prediction is that both technologies will maintain a significant market presence.

Eric Brown: We should keep in mind that the premier providers are offering complex, three-tiered applications. These have a data repository, a business logic server layer, and a presentation layer. And these applications had better be well architected and their object interfaces had better be carefully designed, or the product is going to have major performance problems. Some of the early-entry ISVs in this space have already learned that lesson the hard way.

Plus, these offerings are not cheap. If I were a hospital CIO considering one of these systems, I would think about its utility to my organization using probably a ten-year time horizon. I’d want the solution to be the dominant platform for integration and systems enhancements over time in my entire IT domain. I’d want to know that the platform was extensible and consistent with best practices for object-oriented development, Web services, XML, and all those good runtime tools that I might use to extend what I have in a structured and manageable way as my needs evolve.

Certainly hospitals that can afford them will embrace these technologies because they provide the best of all worlds. They use a centralized database but multiple application servers, so you don’t have to scale up on a single box. There’s no installed executable client with its associated high maintenance costs. Nor is there a Web client with an HTML client interface; that would be fine for some workers, but not for a clinician walking around with a wireless tablet. So although there isn’t a huge installed base of these applications yet, there will be in time. The industry is definitely heading that way.

Scott Cronenweth: What will happen to the financially strapped providers who are buying those client/server applications? Will they be unable to compete and end up closing their doors?

Eric Brown: On the provider side, size matters. The largest integrated delivery systems and hospital chains obviously have bigger IT budgets, so they’re the ones who can make those investments. In stark contrast is a 70-bed community hospital with no investment dollars. So will the growing disparity in IT-driven capabilities and services mean that small hospitals will go away and big hospitals will win? Not necessarily. In many instances there are mitigating factors more important than IT. Hospitals are service businesses, so success still depends mostly on their doctors, the quality of their care, and how people feel about the institution.

Tony Bosselait: I don’t entirely agree with Eric on this point. The lack of a competitive IT infrastructure is a huge problem for struggling institutions, and it will only get worse. Hospitals that don’t have the money to make significant IT investments are treading water right now and working with what they have. But financially stronger institutions are beginning to show not just the payers, but also their doctors, other key staff, and patients, what a Web-enabled, server-centric, componentized application environment can provide. And these stakeholders are pretty impressed. Unfortunately, it’s not uncommon even now for IT “have nots” to run out of cash and close their doors. Whether the differentials in infrastructure translate to what doctors and nurses hospitals can attract or how efficient their back-office processes are, IT can potentially make or break an organization in this industry.

Tony Bosselait

Tony Bosselait joined IBM in 1985 and has held a variety of technical, sales, and management positions. Currently, he is leading the transformation of IBM healthcare software sales from a brand-focused strategy to one focused on healthcare industry solutions.

Formerly he led the IBM federal WebSphere sales team, and his technical accomplishments range from rocket motor and submarine sonar system design to the design and delivery of complex federal, insurance, and hospital IT systems.
Tony has a B.A. in Liberal Arts from Colgate University and an M.B.A. from the University of Vermont's Kalkin School of Business. He has also studied contract law under a DoD program sponsored by Florida State University.

Eric Brown launched Forrester's healthcare practice in 1999 and now serves as a research vice president, examining the impact of information connectivity on the healthcare sector. His current research agenda focuses on healthcare claims intermediaries, the pharmaceutical industry and medical supply chains, and Web applications for physician offices.

Previously, he served as a senior analyst at Forrester, forecasting the impact of application integration technologies and Web services on business collaboration; he also led the manufacturing and business trade research team. Prior to joining Forrester, he was director of healthcare marketing for Optika’s MediPower division.

A frequent and highly rated speaker at industry events in the US and Europe, he received his B.S. in Computer Science from MIT and earned an M.S. in Visual Studies from MIT’s Media Lab.

About the author
Scott Cronenweth is a freelance writer specializing in technology and business strategy, including e-business, software development, and knowledge management (KM). His fifteen-year career includes diverse experiences as a documentation manager, technical writer, marketing copywriter, and reporter. Currently, he writes magazine articles, whitepapers, and marketing collateral for technology companies, publishers, and PR agencies nationwide. He holds an M.S. in Information Science from the University of Pittsburgh.

What do you think of this document?
- Killer! (5)
- Good stuff (4)
- So-so; not bad (3)
- Needs work (2)
- Lame! (1)

Comments?
Facing the challenges of Enterprise Transformation

Brenda Cammarano  
IBM Rational  
13 Apr 2004

from The Rational Edge: This article examines IBM's Enterprise Transformation solution, focusing on the phases required for a successful endeavor, the business drivers affecting transformation, and the ways software development tools can help align IT with business goals.

A strong and successful enterprise requires a solid IT system and application infrastructure, providing a clear linkage between IT and corporate goals and strategies. This infrastructure should support the intricacies of your business while providing an integrated and flexible architecture to support the endless business challenges of tomorrow. With an integrated and aligned IT system, you can be confident that your enterprise will run at top efficiency, enabling you to take advantage of every opportunity to strengthen your business and remain ahead of all competition.

Unfortunately, this integrated architecture does not exist at most corporations. Instead, organizations maintain a disjointed IT landscape, with applications and systems all working independently, and developed vertically to address the needs of each department rather than horizontally, across departments, to address the needs of the business. IBM is dedicated to helping these organizations transform themselves to create an on demand computing environment, which begins with the integrated architecture we just described — one that facilitates change across all business constructs.

At the same time, we recognize that tumultuous economic changes during the past few years have forced organizations to adopt tighter budgets and stringent guidelines for IT projects. Most have limited the scope of new application development, integration efforts, and legacy enhancements, not to mention capital equipment procurement.

To help organizations effectively establish a more cohesive yet cost-efficient approach to IT in this economic environment, we encourage them to evaluate existing resources that can help them address alignment and integration needs. As they go through this evaluation process and consider how to allocate budget dollars, companies need to keep the IT challenges driving today's business decisions uppermost in their minds. These include:

- **Integrating disparate IT systems across the enterprise.** One approach to the problem of disconnected systems is called "enterprise architecture." Pioneered by US-based state and federal IT organizations, enterprise architecture, by some estimates, could lead to as much as a forty percent savings in application integration costs.
- **Development staff turnover/retirement.** This equates to losing an understanding of the intellectual capital of legacy systems, which causes system maintenance costs to skyrocket — consuming as much as 80 percent of the total IT budget because systems need to be relearned.
- **Ongoing legislative mandates requiring absolute conformance.** Experts predict that Fortune 1000 companies will spend as much as $2.5 billion this year on compliance-related projects.
- **Weighing the options to outsource development, maintenance or testing.** Businesses may be able to cut labor costs 25 to 75 percent by using workers in India, China, and the Philippines.
- **Cost of application maintenance and modification.** Of all IT executives surveyed by Forrester Research, 70 percent plan to use Web services for integration of internal applications and data as a way to address this challenge.

To help companies realign their IT infrastructures in a way that will address these challenges, IBM offers a unique solution called "Enterprise Transformation." This solution offers a holistic approach to the alignment of IT with the mission, strategy, and business needs.
of the entire organization through a comprehensive phased approach. Those of you familiar with the four phases of IBM Rational’s highly successful Rational Unified Process®, or RUP®, which steps software development projects through a flexible development process, will recognize a similar pattern in the four phases of the Enterprise Transformation solution.

This article will take a look at what defines Enterprise Transformation: the phases required for a successful endeavor, the business drivers affecting transformation, and the ways software development tools can help align your IT portfolio with business goals and strategy.

Understanding your enterprise

In this article, we will use the term enterprise to refer to the systems at work within an organization. This definition was first introduced by John Zachman in the 1980s, while he was developing IBM’s information planning methodology. Zachman believes “…the design of the system is the design of the enterprise; and if the system can't change, the enterprise can't change!” In other words, for the business to operate effectively and grow, systems must be synchronized and aligned with the business.

An enterprise operates along many dimensions, with each dimension providing a portfolio of elements essential to the enterprise as a whole. The IT analyst firm Giga refers to three types of portfolio management: IT asset management (ITAM), application portfolio management (APM), and project portfolio management (PPM). Giga views these as sub-disciplines within the full IT Management Portfolio.

Let’s place these sub-disciplines in a typical business context. Applications that address functional business needs within your organization are part of Application Portfolio Management (APM). These applications, both new and legacy, can range from simple spreadsheets for tracking employee vacation and salary to more advanced applications that deliver core business processes, rules, and information. The architecture to support these applications is known as IT Asset Management (ITAM). Assets include hardware, such as servers, mainframes, terminals, laptops, and printers; data and software applications that reside and operate on the hardware; and the networking and connectivity required to facilitate communication among all hardware and software components. All these assets are part of the “architectural blueprint” for your company.

Unfortunately, most businesses have limited or non-existent APM or ITAM systems. Instead, their business units work in isolation — developing departmental applications, purchasing custom applications, purchasing supporting hardware — all to support their own requirements without considering the needs or strategy of the organization overall. The lack of collaboration, understanding, and planning among business units contributes to the following problems within the enterprise:

- Redundant and sometimes contradictory information stored in disparate databases, which can lead to reporting of erroneous results and faulty business decisions.
- High maintenance and development costs resulting from independent development efforts.
- High costs associated with unaligned capital asset purchases and underutilized servers/equipment.
- Security gaps throughout the enterprise because of non-secured assets.

To complicate the situation even further, the lack of PPM systems means that future project plans within and across departments are not tracked, communicated, or consolidated. If one department is planning to develop an inventory tracking system, and another is developing a sales tracking system, their efforts will be misaligned and will miss opportunities to integrate and leverage data and shared processes.

According to Forrester, disjointed (or missing) sub-disciplines within an overall Management Portfolio can adversely affect an organization’s value:

Only 10 percent to 15 percent of organizations effectively roll up the sub-disciplines into a higher-level process to better enable strategic decision-making — the very definition of IT portfolio management. As a result, organizations are likely wasting 5 percent to 8 percent or more of their overall budgets due to duplicated, misaligned and ineffective spending. For an organization with a $500 million per year IT budget, this translates directly into $25 million to $40 million per year of pure waste. IT complexity has reached the point where it is no longer possible to manage these resources in the haphazard ways of the past. 

On a simple level, we can say that business requirements remain unfulfilled at many organizations because IT efforts do not properly address them. But the reasons for this glaring inadequacy and the resulting waste of budget resources are complex; the solution requires a higher-level vision for managing IT efforts across the enterprise.

What is Enterprise Transformation?
Enterprise Transformation is an initiative to align business needs directly to IT actions. It involves a full assessment and revitalization of an organization’s business applications and systems — both internal processing and customer-facing applications.

Phases

There are four fundamental steps or phases involved in Enterprise Transformation (see Figure 1):

1. **Discover.** Develop a strategy for discovering the elements within your existing system and application portfolio. This involves mining, understanding, and evaluating existing applications and systems.
2. **Plan.** First, formulate a revitalization to determine what steps you must take to modernize/revitalize your portfolio. This involves stakeholders across the enterprise who assess the current landscape (as defined in the Discover phase) and determine the best mix of projects to align IT spending with organizational goals. Then, develop an execution plan for these projects that revises, extends, preserves, migrates, integrates, and builds upon your existing resources.
3. **Develop.** This involves building the assets specified in your plans.
4. **Deploy.** Transition all new systems into the operational infrastructure. Revisit and revise your plans periodically and/or as needed to reflect changes in your business needs and the competitive environment.

We will discuss these steps in more detail below, focusing primarily on essential elements in the “Discover” and “Plan” phases. The “Develop” and “Deploy” phases entail operational expertise to implement requirements determined in the “Plan” phase. A detailed explanation of these two phases is beyond the scope of this discussion but will be covered in another article.

Figure 1: The IBM Enterprise Transformation initiative

Benefits

Pursuing the four-phase transformation process we have described delivers value through:

- An infrastructure built on consolidation and collaboration among departments.
- A standardized process to support and augment the infrastructure.
- A centralized repository of all applications in the organization; components of these applications are documented, understandable to technical professionals, and available for analysis to support operational efficiency.
• A competitive edge in the marketplace based on an agile infrastructure allowing rapid change to applications and process.
• An optimized development team focused on high-value business requirements.
• Lower costs for application integrations.

Discover phase of Enterprise Transformation

The Discover phase consists of many processes, both manual and automated, which yield a robust inventory of your enterprise assets. The automated approach involves creating a repository containing meta information for multiple virtual storage (MVS) and distributed environments. Once you have collected the data, you can apply impact analysis studies against the inventory. The knowledge and capabilities you will obtain through this process are:

• A high-level view of your applications as well as an understanding of their components and architecture.
• A deeper understanding of business process flow, gained by drilling down into applications.
• Ability to identify areas of an application that will change as a consequence of other changes you make, including lines of code that use affected data items and indirect data items.
• Comprehensive insight into databases and files that comprise the application, and data flows that provide logical connections among programs, processes, and applications. This insight is gained through a process of “harvesting” data and application sources, many of which may be obscure initially. We will say more about data harvesting in the Legacy transformation section below.

Reverse engineering and visual trace are additional processes that help teams harvest existing intellectual capital. Reverse engineering provides the ability to connect to databases and source code, which you can then render into visual models. These models communicate key system characteristics to predict system qualities and the effect on specific properties when aspects of the system are changed. Visual trace[6] is used during program execution to reveal the execution path through a diagram; it depicts the control flow and how the software operates.

These processes, combined with manual efforts to record all hardware and networking assets, help define the current state of your enterprise, at both an Application Portfolio Management (APM) and an IT Asset Management (ITAM) level. This definition serves as a foundation for future IT initiatives.

Plan phase of Enterprise Transformation

The Application and System Portfolio is the starting point for the Plan phase. As you enter this phase, you are ready to begin integrating current and future project portfolio management (PPM) into your overall system.

The Plan phase involves stakeholders across the enterprise who will carry out the assessments defined in the Discover phase. During Plan, the project team may make recommendations ranging from “We need to develop a new user interface for Application X” to “It would be cost effective to outsource the maintenance for Project Y.”

To help determine the best mix of IT spending to achieve alignment with organizational goals, for each project proposal, the planning team must specify cost projections, impact analyses, ROI, risks, and benefits. This will require a decision-support tool and cost models, which are available in the IT marketplace. The planning team must then present this supporting information to the executive governing body, or IT Capital Planning Community, for evaluation and a final decision on each proposed project.

Business issues

As you formulate a revitalization and execution plan for your organization, using the PPM approach, you will weigh IT considerations against business goals. This will include consideration of the following key business issues:

• Legacy applications: Now that you’ve discovered the specifics of your applications and data your next step is to plan what legacy applications need to be migrated, made obsolete, updated with a new user interface, consolidated, or totally rewritten.
• System integration: As you evaluate all systems at work in your organization, you should consider extending and integrating existing systems and applications with advanced technologies.
• Outsourcing of labor: Consider maintaining a 24 x 7 development staff, which reduces time-to-market and thus increases your competitive strengths, and offering a significant return on investment given reduced payroll costs.
• Mandates and standards: Incorporating new standards may be required to ensure legislative compliance, depending on the countries where your business operates. In addition, many organizations will improve overall operational efficiency by adopting any number of standards that have emerged during recent years to support code reuse, Web enablement, business process engineering, etc.
Let’s take a closer look at the issues you will likely have to address during the Plan phase.

**Legacy transformation**

Enterprise application portfolios usually span several technologies for Web, distributed, and mainframe applications. Legacy applications typically perform heavy-duty processing functions within the enterprise; they may be based on any number of programming languages, including COBOL, PowerBuilder, RPG, C, C++, and Java. Maintaining and upgrading these legacy applications represents significant cost. Giga research shows that between 60 and 80 percent of the typical IT budget goes to application maintenance, because as employees come and go, new people must learn how to deal with these legacy applications.

Although these applications retain functional value for certain areas of the business, their high maintenance cost, reliance on outdated technology, and misalignment with business-wide demands add up to a need for a transformation plan. But before you begin dismantling or replacing these applications, you need a detailed understanding of each one, which you can accomplish mostly through the “harvesting” you do during the Discover phase. This process, which involves locating all source code, maintenance patches, data sources, and system interdependencies, can be quite daunting. Often the developers who wrote these applications have left the company, leaving behind only scant documentation. Those who remain may have limited insight into the application’s process flow, data architecture, or business logic. The Discover phase in IBM’s Enterprise Transformation solution helps to reduce the time, effort, and stress associated with these challenges.

When the Discover phase is complete, the primary task during the Plan phase is to address evolutionary options for each application. This process is seldom cut-and-dried; rather it involves evaluating many aspects of each application, including total cost of ownership, efficiency, competitive advantage, supporting foundation, risk factors, scalability, and business processing priority. As enterprise transformation teams evaluate these factors, they must consider the business objectives to determine the most effective way to leverage existing applications. Some options to consider include:

- **Extension.** Some systems may be effectively modernized through Web enablement.
- **Segmentation.** Teams may be able to construct new presentation logic that is segmented from the business logic, convert mainframe terminal interfaces to HTML, and distribute various components to application server machines.
- **Planned obsolescence.** Some applications can be retired if they have been abandoned by a user community because they are inefficient or if they have been replaced by other processes. Discovering obsolete applications usually reveals deserted servers hosting data and functionality.
- **Integration.** Getting multiple applications to work together more efficiently requires development teams to address points of integration between processes and data across the business. Increasingly, teams use component architecture and Web services to centralize processing for integrated applications.
- **Code rewrites/translations.** In some cases, teams may choose to totally reengineer and migrate an application to a new platform and/or language.

A legacy transformation plan usually contains a mix of the above options in order to satisfy, with available funding, the business needs and user requirements throughout the organization. To effectively migrate, integrate, and extend the application portfolio according to plan, it is important to constantly track and update application requirements throughout the development cycle, as well as to continuously verify quality and manage change.

**Outsourcing of labor**

A trend that began in manufacturing and retail production, outsourcing is now gaining momentum in IT. Cost savings have fueled this trend: A computer programmer in India making $20,000 can do the same work as an American programmer making $80,000. Companies who don’t consider outsourcing to reduce their costs may soon be out-priced by their competition.

But realizing a potentially substantial savings through outsourcing will require dedication to process and management, a clear understanding of what types of projects to outsource, and due-diligence regarding vendor selection. Your choice of software tools is also important: If you want to track and communicate project requirements, map testing plans to each requirement, and use models to visually demonstrate data and logic-flow, then the artifacts and data must be accessible to all team members everywhere. You will need clear communication across geographical boundaries to ensure that the right application is developed on time, using the right requirements and quality specifications.

Alignment of process and development tools is fundamental in achieving a high rate of return for outsourcing engagements. As a META Group analyst stated:

> ... most IT organizations save 15% - 25% during the first year; by the third year, cost savings often reach 35% - 40% as
companies “go up the learning curve” for offshore outsourcing and modify operations to align to an offshore model.

Typically, the projects that work best for outsourcing are those that require repetition and have predictable outcomes, such as data conversions. Projects that don’t require extensive decision-making during the development cycle and those that don’t require knowledge of the sponsoring country’s government regulations are also good candidates. Key to success for any project is addressing and understanding cultural and language differences of vendors overseas, as well as having local staff intimately involved in communications among different project and development groups.

The attention devoted to communication, process, and standards is an important measure of an outsourcing model's effectiveness. In particular, the Capability Maturity Model - Integration (CMMI) is acknowledged as an effective model for obtaining standardized and repeatable results; the higher the level you achieve, the more notable the cost savings.

The IBM solution for Enterprise Transformation relies on CMMI principles to improve processes teams use to create software and thereby improve the software’s quality. An effective process is vital to the success of an outsourcing program. Simply put, a process is a sequence of steps performed for a given purpose; it integrates people, methods, procedures, and tools. Tools that are critical to support a uniform process for outsourced teams include requirement tracking and visual modeling tools. These are necessary for communicating project specifications and overall application flow. Quality assurance tools and change and asset management tools are also needed to verify code development and track application modifications, respectively, throughout the project lifecycle.

Mandates and standards

Whether your organization is subject to a government mandate — for example, the USA Patriot Act, HIPPA, Basel II, TREAD — or adopting a new standard to improve business practices — Six Sigma or CMMI, for example — keeping up with the avalanche of recent requirements can be a daunting task. To be successful, organizations must attain a solid understanding of each set of requirements, and then learn how to adapt their tools, methodologies, and application portfolios to implement the requirements.

Complying with multiple standards and mandates is challenging because each set of requirements has a different purpose and uses different terminology to specify it. In complying with one mandate, organizations must avoid implementing new systems too rigid to accommodate the adoption of a second or third set of requirements. At the same time, compliance with multiple standards should not lead to inefficiencies in individual business processes.

As organizations grapple with these challenges, many are turning to software that can help them comply with the law without adding staff. These systems can help businesses identify common threads in their operating procedures, so that they can implement each set of requirements properly, and the mandated controls can become visible in the operational structure.

One of the most critical mandates affecting businesses across all industries in the US is Sarbanes-Oxley, which requires complete conformance. As META Group states:

CFOs must make financial accuracy a top priority due to regulatory conditions brought on by recent accounting scandals. G2000 companies must appropriately prioritize IT projects and ensure that their current solutions provide the required financial transparency and visibility.

In general, many companies are wondering how they can scope out the work necessary to become compliant, yet still manage all their other IT projects and deadlines. How long will compliance-related development take, and will they be able to meet the associated deadlines? The problem is magnified when applications that must be made compliant are developed to address a particular department’s needs and are not centrally managed.

The new mandates also present auditing and bookkeeping challenges. Auditors will be looking for documents on everything: customer orders, shipments, electronic transactions, customer ID verification, and more. For some businesses, this will require retooling their accounting practices and retraining their accounting staff to:

- Validate that all mandate controls are incorporated into systems.
- Ensure that those controls are fully tested.
- Maintain an audit trail and documentation for system audits.
- Foster an agile development environment that can support new compliance mandates and supplements.
- Continuously validate compliance and the organization’s ability to handle change.
- Assess overall project operations and budget controls.

Overall, these new regulations are forcing companies in all industries to rethink the way they manage data and business information
across the enterprise. Progressive companies are now taking the opportunity to build systems that improve real-time business process efficiency and control — beyond the scope of any specific regulatory compliance requirement.

Service-oriented architecture (SOA) and Web services

Faced with a portfolio of disparate systems, limited resources, reduced budgets, and tighter business deadlines, companies can no longer use the old “rip and replace” approach to updating legacy applications. In other words, they cannot resolve the problems of disconnected systems with new systems. Instead, businesses must find ways to make their current systems work more effectively. Many are now working to decipher common processes within their current application portfolio and then share these processes across the board. Over the past decade, advances in distributed computing and component-based software development have begun to support architectural solutions that facilitate change. But the question remains: How do you “re-architect” applications that have already been designed and deployed to support a more comprehensive business strategy?

For many organizations, the answer lies in the latest evolution of the distributed computing and component-based development paradigm: a service-oriented architecture (SOA). Based on Web services, an SOA gives an IT organization the ability to standardize common functions used among many applications as reusable components or services. This enables developers to focus their efforts on creating unique processes within an application, because they can leverage common process functionality across systems simply by calling a Web service. For several years, SOA and Web services have been helping companies improve their IT infrastructures in this way. The value of a fully realized SOA extends well beyond your firewall: It enables you to leverage Web services created by external service providers and develop Web services within your organization that you can offer to others.

Forrester Research conducted a survey of 75 IT executives at large North American companies to find out what they are doing with Web services. Internal deployment of Web services capabilities were at the top of the project list with 83 percent of firms planning to use Web service inside their firewalls. The project breakdown for these firms planning to use Web services internally is as follows:

- 70% for application integration
- 65% for accessing mainframe client/server systems
- 64% for facilitating internal Web portals.

In addition, the survey indicated that 57 percent of firms are likely to adopt customer Web services, and 44 percent plan to adopt supplier Web services.13

Research indicates that introducing an SOA to your IT infrastructure delivers benefits across the organization, including:

- **Agility to handle future change and support a more competitive business.** In the current business environment, the ability to make changes quickly is paramount. IT organizations must be prepared to handle mergers that require integrating heterogeneous applications, or to implement critical process updates that affect multiple systems.
- **Greater capacity to expand business opportunities and revenue.** An SOA environment allows companies to working more effectively with vendors and partners; it enables them to expand distribution channels, leverage trading opportunities, and improve both the quality and breadth of their supply chain.
- **Increases in reuse and return on existing assets.** An SOA offers the ability to leverage existing legacy applications, which is usually a cost-effective alternative to rewriting code.
- **Easier integration of data and applications.** With an SOA, teams can more easily develop an internal architecture of services for integration, which reduces the time and effort required for each project.
- **Internal standards for integration.** Variations in languages, technologies, and infrastructure lead to different approaches to application integration. The use of Web services within an SOA framework fosters a standardized approach to data and process integration as well as collaboration across diverse teams.

Summary

Initiating and managing an Enterprise Transformation project is by no means for the faint of heart. Rather, this is an effort requiring dedication, discipline, and a cohesive Enterprise Transformation team. The team must be willing to pioneer through the Discover and Plan phases to establish the state of the enterprise and map out a strategy for its transformation. Once these phases are complete, the team can then enter the Develop and Deploy phases, where the plan is executed and finally transitioned into business operations.

A successful transformation plan is usually accompanied by a comprehensive set of tools to facilitate process and development through each phase. The best tools guide you through a process of best practices at each step, assisting you with discovering your assets, mapping out a business model to communicate the steps of your project, and tracking requirements for each development project. The tools should also help you code the application, plus model and test all aspects of your transformation effort, while helping you manage and track changes throughout the process.

IBM’s Enterprise Transformation initiative can help you to truly modernize the way you do business today, and also prepare for rapid responses to your business needs in the future. It helps you create a well-designed structural foundation that fosters business resiliency, enables you to meet business goals, and keeps you ahead of the competition.

**Notes**


6. Visual trace is a capability unique to the IBM solution for Enterprise Transformation.


9. The Capability Maturity Model (CMM) Integration (CMMI) is a five-level framework for increased productivity and predictability, designed for guiding dramatic improvements in an organization's ability to increase productivity and quality, reduce costs and time to market, and enhance customer satisfaction.

10. The USA Patriot Act requires banks, investment firms, insurance companies, and stock and commodities exchanges to put in place procedures to collect information on customers when they open accounts. The Health Insurance Portability and Accountability Act (HIPAA) requires businesses to adopt standards for the security of electronically based health information, to be implemented by health plans, health care clearinghouses, and certain health care providers. Basel Capital Accord and Basel II (generally referred to as “Basel II”) encourages the banking industry to use more sophisticated risk management methodology and requires a capital minimum requirement for financial institutions. The Transportation Recall, Enhancement, Accountability and Documentation Act (TREAD) requires vehicle and equipment makers to submit reports summarizing information about consumer complaints as well as warranty, legal claims and field reports to the National Highway Traffic Safety Administration (NHTSA).

11. Six Sigma is a methodology that uses data to measure and improve a company's operational performance by eliminating or preventing “defects” in process. See footnote 9 for an explanation for The Capability Maturity Model Integration (CMMI).


**About the author**

As a market manager at IBM Rational, Brenda Cammarano is responsible for the Enterprise Modernization solution within the Rational brand. Currently, she is developing a complete strategy to bring this solution to market that reflects a deep understanding of clients’ needs and how IBM products can best work together to serve them. In this effort, she draws upon her experience as a former senior technical evangelist for the Rational Enterprise Suite, a programmer/analyst in object-oriented engineering, and a key player in product management and technical marketing organizations for several high-tech companies in the Greater Boston area.


4/15/2004
Come to IBM Rational Software Development User Conference 2004... Register early and save!*  

9 Apr 2004

from The Rational Edge: Information about an exciting new conference for the IBM Rational and developerWorks Live! communities.

July 18-22, 2004
Gaylord Texan Resort and Convention Center
Grapevine, Texas (Dallas/Fort Worth area)

Software runs the world…and IBM Rational can help you lead the way!

Whether you're a software development professional with years of Rational or WebSphere Studio tool experience, an IBM Business Partner, or a new client eager to learn more about IBM technologies, this conference will show you how to lead your organization toward success. With its focus on IBM Rational and the IBM Software Development Platform — including WebSphere Studio, Eclipse, and .NET technologies — you will learn how to integrate existing software and systems, create new application architectures, and transform business processes within your own organization or for your clients. If you're a former developerWorks Live! attendee, you'll find the open standards technology training you've come to expect, with sessions focused not only on Eclipse, but also Web Services, SOA, Linux, and more.

Session tracks include:

- Bridging Business and Technology in an on Demand World
- Design and Construction
- Design and Construction using WebSphere Studio and Eclipse
- Process and Project Management
- Requirements and Analysis
- Software Configuration Management (including an Advanced track)
- Software Quality
- IBM Software Development Platform 101

You'll also find:

- More than twenty different three-hour hands-on workshops, covering IBM tool support for all phases of the software development lifecycle (sessions fill early).
- Popular "Birds-of-a-Feather" sessions for sharing tips, techniques, and strategies on specific technologies.
- Solution Center, Engineering Lounges, and Exhibit Hall — network with peers, get technical advice from IBM engineers and Partners, and see the latest software solutions from companies like Microsoft, InferData, Jaczone, Inc., Network Appliance, Inc., Number Six Software, Praxis Engineering, and RTTS.

For more information, contact RSDUC@conferon.com, call toll-free: 888-889-0074 (781-278-2611 in Massachusetts), or visit ibm.com/software/rational/ruc

*Conference registration: $1,295 on or before April 30; $1,495 May 1 to June 30; $1,595 after July 1 or on site (special group savings: $100 off each registration fee for 5 or more employees from the same company).  
Hotel: Call 407-586-2000 now or visit www.gaylordtexan.com/IBMrational to book a room at Gaylord Texan Resort at the special
One of the greatest challenges for a development team of any size is to exploit the skills and talents of every team member in order to create a genuinely useful product. Meeting this challenge is especially crucial when the team is small. Unfortunately, project leaders too often let the development effort hinge almost entirely on the talents of one or two people, which results in a “hero” culture; under such circumstances, the team can achieve success only through extraordinary effort on the part of a few key individuals.

But it doesn’t have to be this way. In this book, Gary Pollice and his co-authors describe their experiences in a highly collaborative project that did engage the talents of all team members. They discuss how they developed a personal software process (PSP) data collection tool, based on the Software Engineering Institute discipline created by Watts Humphrey.1 Walking readers through the entire process, from forming the team to developing requirements, creating the initial operational architecture, completing the application, and delivering the product to the customer, the authors have created a case study of a real agile project, including all of the mistakes and missed opportunities along the way.

The basis for their process was IBM Rational Unified Process®, or RUP®. The authors discuss how they tailored the formal RUP framework, which includes far more artifacts than most projects require, to meet the needs of a small team in an agile development environment. In particular, they defined the guiding principle for small team development: Perform only those activities that will help the project deliver value.

Typically, small teams have fewer than ten members, and wasteful activities can be detrimental or even fatal to the development effort. In their project, the authors began by identifying artifacts they felt were absolutely required, including the vision statement, risk list, development case, use cases, test cases, project plan, glossary, and architecture. As the development effort progressed, the team discovered that they could create many of these essential artifacts informally. For example, a test plan might be simply a sticky note attached to the wall. As the authors state, this was “good enough” for the team’s needs. By creating many of their artifacts according to this principle, they avoided squandering time and attention on documentation that wasn’t necessary for delivering a stable, well-tested product.

**Describes lessons learned**

Much of this book is devoted to describing lessons learned by the team as the project progressed. The authors recount how they overcame challenges to operate as a team, despite being distributed across several locations. Early on, they devised ways to employ the Groove collaboration tool (http://www.groove.net/) to help team members communicate with one another, and this had a dramatic effect on productivity in later project stages. The book also provides a highly detailed breakdown of each project phase, starting with Inception, which includes establishing the project vision and initial requirements gathering. In particular, this section provides an excellent example showing how use cases can drive a development effort, as well as a very clever way to create a vision statement by producing a product sales brochure.
The book then follows the authors’ project into the Elaboration phase, during which the team discovered a gap between customer expectations and the deliverable they were building. This section points out how such difficulties can lead to major problems if they are not detected through delivery of tested functionality early in the project (i.e., by using an iterative development approach).

In the Construction phase the team discovered that a small project develops its own “rhythm.” When team members become familiar with one another’s work habits and concerns, they experience a rapid increase in productivity. It is therefore important for small teams to seek this level of trust and familiarity through frequent and open communication.

Finally, in the Transition phase, in which the team prepared the final product for delivery, they discovered that the procedure for installing the product had become a sticking point — one they could easily have avoided by recognizing earlier that end users would need an easy installation procedure. The authors point out that what seems to be a trivial problem early in a project can have dramatic negative effects later on if you ignore it and allow the problem to grow.

By walking readers through their project, the authors introduce us to most of the problems small teams typically face: pressure to deliver functionality with limited personnel; pressure to produce the product quickly (in this case, team members were doing their “regular” jobs in addition to project work); challenges in communicating across geographic locations, and sometimes overcoming language and cultural barriers; inherent project problems such as changing requirements, incorrect design decisions that require rework, and oversights that lead to problems late in the project. To their credit, the authors applied best practices — use-case driven development, focusing on architecture, frequent testing, and iterative development — to offset many of these project risks. And they show quite clearly, through their example, that small development projects can be both RUP-driven and agile at the same time.

Shows adaptability of RUP

Of course, no book is perfect; I noticed three potential problems for readers who might attempt to transfer the lessons of this book to other projects. First, the authors were fortunate to have access to several high-end professional tools, including IBM Rational ClearCase® for software configuration management, IBM Rational RequisitePro® for requirements tracking, and IBM Rational Rose XDE® for model driven development. However, many development projects must proceed without the support of such powerful tools. It would have been useful for the authors to describe how to use open source tools (such as CVS for configuration control and JUDE for UML modeling) as alternatives.

Second, although for each project phase the authors point out their mistakes and key lessons learned, they don’t provide a concise summary of these lessons that is easily referenced — for example, in a sidebar or text box. Such a feature would help the reader form an association between a particular project guideline and a phase of the development effort.

And third, the authors’ project was not as time-driven as most. It had a deadline, but missing it did not carry penalties as severe as those attached to typical development project deadlines. For most projects, schedule pressure is one of the primary risks and a frequent cause of failure. So guidelines for avoiding schedule-based risks — such as techniques for trading functionality against delivery dates — would have been a welcome addition.

On the whole, however, for both leaders and team members of small projects, this book provides valuable insight into the workings and common failings of development efforts. Moreover, it shows exactly how to tailor a RUP approach to meet the needs of a flexible, iterative development effort without introducing unnecessary process overhead. In fact, the bare bones, artifact-driven project management approach the authors’ team adopted has proven to be very successful in many projects in which I have been involved, as have many of the other suggestions and recommendations put forth in this book. I would recommend this book to anyone either contemplating or currently involved in a small team software development effort.

Notes

1 See Watts Humphrey, A Discipline for Software Engineering (Addison Wesley, 1995).

About the author

As the Principal Architect for BioLogic Software Consulting, a software architecture consulting firm, Ben Lieberman provides clients with training, mentoring, and practical advice on architectural issues for software systems. He became what he calls a “Consulting Architect” more than five years ago, having arrived at this role via a circuitous route. Prior to that, he had spent nearly ten years as a research scientist involved in numerous software projects relating to biological laboratory research data collection and analysis (e.g., Bioinformatics). He holds a B.S. in Molecular Biology from University of California at Los Angeles, and a Ph.D. in Biophysics and Genetics, from the University of Colorado.
Book review — Critical Testing Processes

Susan McVey
IBM
29 Mar 2004

from The Rational Edge: McVey reviews a book outlining a four-step process for managing test projects that focuses on early planning and preparation.

by Rex Black
Addison-Wesley, 2004
ISBN: 0-201-74868-1
Cover price: US$49.99
571 pages

Critical Testing Processes by software testing consultant Rex Black presents a four-step process for managing test projects: Plan, Prepare, Perform, and Perfect, which is a variation on the well-known Plan-Do-Check-Act cycle of quality engineering pioneer W.E. Deming. Intended for software test managers and consultants, the book, like Black's process itself, devotes a great deal of time and attention to the early planning and preparation stages of a project. Careful planning, Black says, will make the actual execution of tests go smoothly and quickly.

The book uses a fictional software project to illustrate the recommended processes and procedures: a new release of the SpeedyWriter word processor and its associated suite of office tools. Throughout the book, each discussion of a topic is followed by a section of narrative, mainly from the integration and system test manager’s point of view. This blend of instruction and story makes the book not only more readable and interesting, but also more effective, because it actually shows the reader how to apply Black’s principles. Many of the documents, charts, graphs, and records from the fictional project are available for download on the author’s Web site, so readers can use them as templates for their own work.

What I liked best about this book is that it uses an upgrade release of an established product as the main example. In the real world, most software engineers work on maintenance releases much of the time, but for some reason almost all the software quality engineering literature covers testing brand-new products during their initial development phase. It’s refreshing to see a writer address the issues of a realistic situation instead of an ideal one.

Black’s four-step process

The Plan-Prepare-Perform-Perfect process is broken down into practical and manageable steps. The first step, Plan, includes:

- Analyze risks to decide where to expend test effort.
- Estimate the time and cost of testing.
- Analyze the budget and estimate the Return on Investment from testing (reduced cost of failure versus cost of testing).
- Get agreement on the estimates from everyone involved in doing the work, so you know the estimates are realistic.
- Plan the tests.
- Assemble support for the plan.

As you can see, most of this advice is more about good project management than software test engineering per se. Black is a savvy veteran who understands office politics, and he offers quite a bit of advice about building consensus, handling problems, and avoiding conflict. He weaves most of this into the narrative rather than defining abstract rules.

In the next section about step 2, Prepare, Black addresses the issues of hiring, team building and training, creating test systems, and measuring test coverage. He also discusses how to handle common challenges such as vaguely defined requirements and environments, and the need to balance schedule, budget, and quality. The line between planning and preparation is a little artificial; in this case, preparation includes everything that happens between the day the manager gets signoff on his general plan and the day the test team
begins executing the test cases they have created.

The third step — performing the tests — is probably the longest and most difficult part of most software testing projects, but this section is the shortest part of the book. Since everything has been well planned and thoroughly prepared in advance, there is relatively little to say about the straightforward process of getting the work done. Black includes substantial information here about how the test team works with the release engineering or configuration management team and how product builds are handed off to the test team for testing. He also addresses practical personnel and scheduling issues, and the fictional narrative describes how the “Big Build” of the product arrives at the lab and is handled by the test team.

Finally, the fourth part of the book is devoted to step 4: perfecting both the product under test and the test process itself. This part deals with how to find, report, and respond to problems. It includes seemingly simple topics such as bug reporting, as well as larger challenges such as managing relationships and communication between the test group and other project stakeholders. The narrative risks becoming overly sweet at this point, as the managers of all teams pat each other on the back and beam with gratitude for a well-run project. But the point that all that planning and preparation pays off during the later phases of the project is well taken.

Valuable advice, despite the glitches

Unfortunately, Part 4 has a few production errors that can impede understanding. Two charts were swapped by mistake, so the text refers in two places to the wrong charts. In another place, several text boxes are out of order, and sometimes illustrations are several pages away from the text that references them. All the information is there, but you have to hunt for it.

As for the content itself, any book that attempts to capture the whole scope of how you do your job is bound to have parts that you agree with and parts that you don’t. To me, it seemed as if the author’s consultant role colors his point of view. His discussions of how the test manager should relate to the development team, for example, are right on the mark for a consultant but perhaps not for a permanent staffer. Black claims that the test team’s only role is to assess quality; he advises that testers leave all decisions about bug fixes and other quality issues to the development manager, lest they make themselves unpopular and get branded as “Don Quixote: Lone Champion of Quality.”

In my experience, there is a large middle ground between acting like a Don Quixote and completely divorcing yourself from development issues. On many teams, part of the test group’s charter is to speak up on behalf of the customer and push for the resolution of quality problems. Black does not acknowledge the difference in role responsibilities between a consultant and a permanent member of the team — a blind spot he shares with other consultants/writers whose books I have read. Aside from this one criticism, however, I found the book rich in wisdom drawn from experience and very readable. I recommend it.

About the author

Susan McVey is currently a software quality engineer and lead tester for the UNIX version of IBM Rational® PurifyPlus™. A testing specialist for more than a decade, she previously worked as a software developer, a technical animator for television, and a developer of puzzles. She is a senior member of the American Society for Quality (ASQ) and an ASQ-Certified Software Quality Engineer.

What do you think of this document?

- Killer! (5)  - Good stuff (4)  - So-so; not bad (3)  - Needs work (2)  - Lame! (1)

Comments?

Submit feedback
Transitioning from waterfall to iterative development

Per Kroll (pkroll@us.ibm.com)
Director, Rational Unified Process, IBM
7 Apr 2004

from The Rational Edge: The model for a perfect iterative development methodology is in many ways radically different from the perfect model for waterfall development. But, in practice, no team applies either approach strictly according to its model. This article explains why teams might decide to move gradually from a waterfall-like approach to a more iterative one, while outlining steps that can help.

Most software teams still use a waterfall process for development projects. Taking an extreme waterfall approach means that you complete a number of phases in a strictly ordered sequence: requirements analysis, design, implementation/integration, and then testing. You also defer testing until the end of the project lifecycle, when problems tend to be tough and expensive to resolve; these problems can also pose serious threats to release deadlines and leave key team members idle for extended periods of time.

In practice, most teams use a modified waterfall approach, breaking the project down into two or more parts, sometimes called phases or stages. This helps to simplify integration, get testers testing earlier, and provide an earlier reading on project status. This approach also breaks up the code into manageable pieces and minimizes the integration code, in the form of stubs and drivers, required for testing. In addition, this approach allows you to prototype areas you deem risky or difficult and to use feedback from each stage to modify your design. However, that runs counter to the thinking behind the waterfall approach: Many design teams would view modifying the design after Stage 1 as a failure of their initial design or requirements process. And although a modified waterfall approach does not preclude the use of feedback, it does not facilitate, accommodate, or encourage it. And finally, the desire to minimize risk does not typically drive a waterfall project. This article will explore the improvements that an "iterative" approach to the software development process offers over the traditional waterfall approach.

Advantages of an iterative approach

In contrast, an iterative approach — like the one embodied in IBM Rational Unified Process,® or RUP,® — involves a sequence of incremental steps, or iterations. Each iteration includes some, or most, of the development disciplines (requirements, analysis, design, implementation, and so on), as you can see in Figure 1. Each iteration also has a well-defined set of objectives and produces a partial working implementation of the final system. And each successive iteration builds on the work of previous iterations to evolve and refine the system until the final product is complete.

Early iterations emphasize requirements as well as analysis and design; later iterations emphasize implementation and testing.
Figure 1: Iterative development with RUP. Each iteration includes requirements, analysis, design, implementation and testing activities. Also, each iteration builds on the work of previous iterations to produce an executable that is one step closer to the final product.

The iterative approach has proven itself superior to the waterfall approach for a number of reasons:

- **It accommodates changing requirements.** Changes in requirements and "feature creep" — the addition of features that are technology- or customer-driven — have always been primary sources of project trouble, leading to late delivery, dissatisfied customers, and frustrated developers. To address these problems, teams who use an iterative approach focus on producing and demonstrating executable software in the first few weeks, which forces a review of requirements and helps to pare them down to essentials.

- **Integration is not one "big bang" at the end of a project.** Leaving integration to the end almost always results in time-consuming rework — sometimes up to 40 percent of the total project effort. To avoid this, each iteration ends by integrating building blocks; this happens progressively and continuously, minimizing later rework.

- **Early iterations expose risks.** An iterative approach helps the team mitigate risks in early iterations, which include testing for all process components. As each iteration engages many aspects of the project — tools, off-the-shelf software, team members' skills, and so on — teams can quickly discover whether perceived risks are real and uncover new risks they did not suspect, at a time when these problems are relatively easy and less costly to address.

- **Management can make tactical changes to the product.** Iterative development quickly produces an executable architecture (albeit of limited functionality) that can be readily translated into a "lite" or "modified" product for quick release to counter a competitor's move.

- **It facilitates reuse.** It is easier to identify common parts as you partially design or implement them in iterations than to recognize them during planning. Design reviews in early iterations allow architects to spot potential opportunities for reuse, and then develop and mature common code for these opportunities in subsequent iterations.

- **You can find and correct defects over several iterations.** This results in a robust architecture and a high-quality application. You can detect flaws even in early iterations rather than during a massive testing phase at the end. And you can discover performance bottlenecks when you can still address them without destroying your schedule — or creating panic on the eve of delivery.

- **It facilitates better use of project personnel.** Many organizations match their waterfall approach with a pipeline organization: Analysts send the completed requirements to designers, who send a completed design to programmers, who send components to
integrators, who send a system for test to testers. These multiple handoffs not only create errors and misunderstandings; they also make people feel less responsible for the final product. An iterative process encourages a wider scope of activities for team members, allowing them to play many roles. Project managers can better use available staff and eliminate risky handoffs.

- **Team members learn along the way.** Those working on iterative projects have many opportunities during the development lifecycle to learn from their mistakes and improve their skills from one iteration to another. By assessing each iteration, project managers can discover training opportunities for team members. In contrast, those working on waterfall projects are typically confined to narrow specialties and have only one shot at design, coding, or testing.

- **You can refine the development process along the way.** End-of-iteration assessments not only reveal the status of the project from a product or scheduling perspective; they also help managers analyze how to improve both the organization and the process in the next iteration.

Some project managers resist adopting an iterative approach, seeing it as a form of endless, uncontrolled hacking. However, in RUP the entire project is tightly controlled. The number, duration, and objectives of iterations are carefully planned; and the tasks and responsibilities of participants are well defined. In addition, objective measures of progress are captured. Although the team does rework some things from one iteration to the next, this work, too, is carefully controlled.

### Four steps for a transition

Most waterfall projects divide the development work into phases or stages; we can also view this as a first step toward iterative design. But then, to move to an iterative approach, we would apply different process principles, using the following four steps:

2. Divide the detailed design, implementation and test phases into iterations.
3. Baseline an executable architecture early on.
4. Adopt an iterative and risk-driven management process.

Let's examine each of these steps more closely.

**Step 1: Build functional prototypes early**

As a first step toward iterative development, consider one or more functional prototypes during the requirements and design phases. The objectives of these prototypes are to mitigate key technical risks and clarify stakeholders' understanding of what the system should do.

Start by identifying the top three technical risks and the top three functional areas in need of clarification. The technical risks might relate to new technology, pending technology decisions that will greatly affect the overall solution, or technical requirements that you know will be hard to meet. Functional risks might relate to areas in which stakeholders have provided fuzzy requirements for critical functionality, or to several requirements that are core to the system.

For each of the key technical risks, outline what prototyping you need to do to mitigate the risks. Consider the following examples:

**Technical risk:** The project requires porting an existing application to run on top of IBM WebSphere Application Server. Users are already complaining about the application's performance, and you are concerned that porting it might slow performance even more.

**Prototype:** Build an architectural prototype to try out different approaches for porting your application. Ask an expert WebSphere architect to help you. Evaluate the results and write architectural and design guidelines providing the team with dos and don'ts. This will increase the likelihood that your ported application's performance will be good enough to avoid rework late in the project.

**Technical risk:** You are building a new application for scheduling and estimating software projects. You know that a key differentiator for this application versus off-the-shelf products will be how well it supports iterative planning. However, that is also one of the fuzziest areas in your requirement specification.

**Prototype:** Build a functional prototype based on your assumptions about how to support iterative project planning. By demonstrating the prototype to various stakeholders, you will encourage them to pay more attention to planning and tell you which of your assumptions they agree or disagree with. The prototype will help you clarify the planning requirements and also provide you with useful information about the user experience and look and feel for your application. It might even yield some reusable code.

**Step 2: Divide the detailed design, implementation and test phases into iterations.**
Many project teams find it hard to divide a project into meaningful iterations before they know what the project is really about. But when you are ready to enter the detailed design phase, you typically have a good understanding of what the requirements are, and what the architecture will look like. It's time to try out iterative development!

You can use two main approaches to determine what should be done in what iteration. Let's discuss the pros and cons of each approach.

**Approach 1: Develop one or more subsystems at a time.** Let's assume that you have nine subsystems, each with increasingly larger numbers of components. You can divide the detailed design, implementation and test phase into three iterations, each one aiming at implementing three of the nine subsystems. This will work reasonably well if there are limited dependencies among the different subsystems. For example, if your nine subsystems each provided a well-defined set of capabilities to the end user, you could develop the highest priority subsystems in the first iteration, the second most important subsystems in the second iteration, and so on. This approach has a great advantage: If you run out of time, you can still deliver a partial system with the most important capabilities up and running.

However, this approach does not work well if you have a layered architecture, with subsystems in the upper layers dependent on the capabilities of subsystems in the lower layers. If you had to build one subsystem at a time, such an architecture would force you to build the bottom layer subsystems first, and then go higher and higher up. But to build the right capabilities in the bottom layers, you typically need to do a fair amount of detailed design and implementation work on the upper layers, because they determine what you need in the lower layers. This creates a "catch-22"; the second approach explains how to resolve it.

**Approach 2: Develop the most critical scenarios first.** If you use Approach 1, you develop one subsystem at a time. With Approach 2, you focus instead on key scenarios, or key ways of using the system, and then add more of the less essential scenarios. How is this different from Approach 1? Let's look at an example.

Suppose you are building a new application that will provide users the ability to manage defects. It is a layered application built on top of WebSphere Application Server, with DB2 as the underlying database. In the first iteration, you develop a set of key scenarios, such as entering a simple defect, with no underlying state engine. In the second iteration, you add complexity to these scenarios — for example, you might enable the defect to handle a workflow. In the third iteration, you complete the defect entry capability by providing full support for atypical user entries, such as capability to save a partial defect entry and then come back to it, and so forth.

With this approach, you work on all the subsystems in all iterations, but you still focus in the first iteration on what is most important and save what is least important or least difficult for the last iteration.

Approach 1 is more appropriate if you are working on a system with a well-defined architecture — on an enhancement of an existing application or developing a new application with a simple architecture, for example. Most projects building complex applications should use Approach 2, but they should plan the iterations in such a way that they can cut the scope of the last iterations to make up for possible schedule delays.

**Step 3: Baseline an executable architecture early on.**

You can view this step as a much more formal and organized way of doing Step 1: *Build functional prototypes early on.* But what is an "executable architecture"?

An executable architecture is a partial implementation of the system, built to demonstrate that the architectural design will support the key functionality. Even more important, it demonstrates that the design will meet requirements for performance, throughput, capacity, reliability, scalability, and other "-ilities." Establishing an executable architecture allows you to build all the system’s functional capability on a solid foundation during later phases, without fear of breakage. The executable architecture is an *evolutionary prototype*, intended to retain proven features and those with a high probability of satisfying system requirements when the architecture is mature. In other words, these features will be part of the deliverable system. In contrast to the *functional prototype* you would typically build in step 1, the evolutionary prototype covers the full breadth of architectural issues.

Producing an evolutionary prototype means that you design, implement, and test a skeleton structure, or architecture, of the system. The functionality at the application level will not be complete, but as most interfaces between the building blocks are implemented, you can (and should) compile and test the architecture to some extent. Conduct initial load and performance tests. This prototype also reflects your critical design decisions, including choices about technologies, main components, and their interfaces; it is built after you have assessed buy versus build options and after you have designed and implemented architectural mechanisms and patterns.

But how do you come up with the architecture for this evolutionary prototype? The key is to focus on the most important 20 to 30 percent...
of use cases (complete services the system offers to the end users). Here are some ways to determine what use cases are most important.

- **The functionality is the core of the application, or it exercises key interfaces.** The system's key functionality should determine the architecture. Typically an architect identifies the most important use cases by analyzing many factors: redundancy management strategies, resource contention risks, performance risks, data security strategies, and so on. For example, in a point-of-sale (POS) system, Check Out and Pay would be a key use case because it validates the interface to a credit card validation system — and it is critical from a performance and load perspective.

- **Choose use cases describing functionality that must be delivered.** Delivering an application without its key functionality would be fruitless. For example, an order-entry system would be unacceptable if it did not allow users to enter an order. Typically, domain and subject-matter experts understand the key functionality required from the user perspective (primary behaviors, peak data transaction, critical control transactions, etc.), and they help define critical use cases.

- **Choose use cases describing functionality for an area of the architecture not covered by another critical use case.** To ensure that your team will address all major technical risks, they must understand each area of the system. Even if a certain area of the architecture does not appear to be high risk, it may conceal technical difficulties that can be exposed only by designing, implementing, and testing some of the functionality within that area.

The first and last criteria in the above list will be of greater concern to the architect; project managers will focus mainly on the first two.

For each critical use case, identify the most important scenario(s) and use them to create the executable architecture. In other words, design, implement and test those scenarios.

**Step 4: Adopt an iterative and risk-driven management process.**

If you were to follow Steps 2 and 3 as described above, then you would come very close to the model for "ideal" iterative development. Then, your next step would be to fuse Steps 2 and 3, adding a management lifecycle that supports risk-driven and iterative development. That is the iterative lifecycle described in RUP.

RUP provides a structured approach to iterative development, dividing a project into four phases: Inception, Elaboration, Construction, and Transition. Each phase contains one or more iterations, which focus on producing the technical deliverables necessary to achieve the business objectives of that phase. Teams go through as many iterations as they need to address the objectives of that phase, but no more. If they do not succeed in addressing those objectives within the number of iterations they had planned, they must add another iteration to the phase — and delay the project. To avoid this, keep your focus sharply on what you need to achieve the business objectives for each phase. For example, focusing too heavily on requirements in Inception would be counterproductive. Below is a brief description of typical phase objectives.

- **Inception:** Establish a good understanding of what system to build by getting a high-level understanding of all the requirements and establishing the scope of the system. Mitigate many of the business risks, produce the business case for building the system, and get buy-in from all stakeholders on whether or not to proceed with the project.

- **Elaboration:** Take care of many of the most technically difficult tasks: design, implement, test, and baseline an executable architecture, including subsystems, their interfaces, key components, and architectural mechanisms (e.g., how to deal with inter-process communication or persistency). Address major technical risks, such as resource contention risks, performance risks, and data security risks, by implementing and validating actual code.

- **Construction:** Do a majority of the implementation as you move from an executable architecture to the first operational version of your system. Deploy several internal and alpha releases to ensure that the system is usable and addresses users' needs. End the phase by deploying a fully functional beta version of the system, including installation, supporting documentation, and training material; keep in mind, however, that the functionality, performance and overall quality of the system will likely require tuning.

- **Transition:** Ensure that the software addresses the needs of its users. This includes testing the product in preparation for release and making minor adjustments based on user feedback. At this point in the lifecycle, user feedback should focus mainly on fine-tuning the product, and on configuration, installation, and usability issues; all the major structural issues should have been worked out earlier in the project lifecycle.¹

Many ways to apply these steps

In this article, we have described how you can gradually transfer from a waterfall approach to an increasingly iterative approach, using four transitional steps. Each step will add tangible value to your development effort, with minimal disruption. Some teams may take on more than one step at a time; others may run a few projects based on one step and then take the next step. However you choose to use this step-wise approach, it can help you reduce the risks associated with process changes in a development organization.
Notes


About the author

Per Kroll is the director of the Rational Unified Process development and product management teams at IBM Rational Software. He's been working with customers as a trainer, mentor, and consultant on the RUP and its predecessors since 1992 and was the original product manager for the RUP when the product team was launched in 1996. He's also been heavily involved in certifying partners and training Rational personnel to deliver services around the RUP.

What do you think of this document?
☐ Killer! (5)  ☐ Good stuff (4)  ☐ So-so; not bad (3)  ☐ Needs work (2)  ☐ Lame! (1)

Comments?

Submit feedback

developerWorks > Rational

About IBM  |  Privacy  |  Terms of use  |  Contact


4/15/2004
Fast tracking the RUP Inception phase: Eight steps for agile use-case generation

David Kohrell
Technology As Promised, LLC
8 Apr 2004

from the Rational Edge: In this article, an experienced consultant describes how he successfully streamlined RUP's Inception phase for a major government project.

Rational Unified Process,® or RUP,® is a leading methodology for managing software projects. RUP unifies the activities of four software development phases (Inception, Elaboration, Construction and Transition) across the following disciplines:

- Business Modeling
- Requirements
- Analysis and Design
- Implementation
- Testing
- Deployment
- Configuration and Change Management
- Project Management

Those unfamiliar with the methodology typically ask: Is RUP agile or heavy? Can it be applied to short-term as well as long-term projects? How does it work in the real world? The answers to the first two questions depend greatly on how you address the third question. This article focuses on how my team used RUP as an agile process to conduct a public sector project in a real-world setting.

Project background

Between September 22 and November 21, 2003, I worked on the State of Nebraska's Workforce Development (NWD) Unemployment Insurance Benefit System Modernization project. Specifically, I worked on the Inception phase of this project, applying the disciplines of business modeling and requirements gathering. In sixty days, we progressed from a blank slate to a validated set of requirements. How did we accomplish that so quickly?

First, as president of Technology As Promised (www.aspromised.com), I work frequently with fellow contractor Mitchell Ummel, president of UmmelGroup International (www.ummelgroup.com) on various project activities. In parallel with my work on requirements, Ummel created a request for proposal (RFP) to solicit bids from vendors who could develop a system that addressed those requirements. This parallel work was consistent with the "fast-tracking" approach specified in RUP — doing things in parallel that are done sequentially in traditional development methods.

Requirements gathering

We also used a fast-tracking approach to accomplish our business modeling and requirements specification goals. Before we began our work, Allan Amsberry, executive director of the NWD Unemployment Insurance Office and executive project sponsor, stressed the need to conduct the requirements gathering meetings efficiently and with buy-in from the participants. He stated that "...a project of this size requires staff support and a plan focused on a final product that can be used by the consumer. We need to make sure our business needs are cataloged."

To ensure that we would fulfill these expectations, we selected a core team of business and information technology professionals from the...
NWD, along with forty-two additional subject matter experts, to participate in sixteen meetings that lasted one to three days. We facilitated each meeting using a derivative of joint application development (JAD) called joint requirements planning (JRP). It focused on gathering requirements, not on creating a whiteboard prototype. Eleven of the sixteen sessions concerned unemployment insurance claims and benefits, and five concerned unemployment taxes paid by employers. I won't bother describing the predictable difficulties I encountered in scheduling busy professionals for several different sessions; instead I will focus on the process the team used, the results we gathered, and the lessons we learned.

**Process for generating use cases**

Each JRP session focused on a business process area, such as filing an unemployment claim. I generated use cases, following eight basic steps — plus a few bonus steps — for each business process area. We reproduced these steps in large print on posters in the JRP facilitation room shown in Figure 1. Each day concluded with a review of what went well and what we could have improved.

During each session, I captured the group's initial thoughts on the dry-erase board, or "sandbox," as it became known. Once an idea (use-case description, basic flow, alternate flow) was fairly well-formed, it was transferred to posterboard by a team member with excellent penmanship, Larry Clark. Meanwhile, another team member, Becky Massey, typed notes into MS Word that were then exported to IBM Rational Requisite Pro® at the conclusion of each session. Did we really need a three-person facilitating and recording team? Yes. It actually allowed for a near real-time capture of results while also enabling the entire group time to review and own those results. It provided speed and supported consensus building.

**Figure 1:** The Joint Requirements Planning (JRP) facilitation room. Basic steps for each business process were reproduced in large print on posters placed at the top of the wall.

The eight basic steps we used to generate use cases for each business process area are described below.

**Step 1: Confirm actors and goals.**  
Have all actors and their goals been identified?  
Which actors can be generalized (combined)?  
Which goals are potential use cases?

**Step 2: Develop an outline of the use case(s).**  
For the goals identified as potential use cases, what are the key pieces?  
For each outline level, what are key data?  
Outline all use cases.  
Prioritize the use-case flows.  
Decide on a final use-case list (for initial pass).

**Step 3: Write a brief description of the use case(s).**  
What two or three sentences describe all actors and the basic flow?  
Generate content first, and worry about wordsmithing later.
Step 4: Detail the basic flow.
What event starts the use case?
How does the use case end?
How does the use case repeat some behavior?
What is the "happy" (best case) path?
There is one and only one basic flow.

Step 5: Detail the alternate flows.
Are there optional situations for the use case?
What might go wrong?
What might not happen?
Which resources might be blocked?
Which alternate flows are special — perhaps nonfunctional — requirements (i.e., they apply to this use case only)?

Step 6: Review the use case(s).
Are there more use cases?
Should some use cases be redefined?
Which ones can be combined?

Step 7: Record pre- and post-conditions.
What was the previous state before this use case comes into play?
What happens once the use case is complete?

Step 8: Develop generalizations for all use cases.
Determine shared content and process for the use cases.
What items have been noted for the glossary or as global business rules?
Who has the most recent and accurate source document?
Where is it located?

Additional process tools:

- **Sandbox**: This is where first thoughts and discussion notes go. In our case, it was a simple dry-erase board for recording the idea and reformulating it if it evolved into something significant.

- **Posters**: This is where a topic gets recorded once it graduates from the sandbox.

- **Parking lot**: This is where we can capture initial ideas that are tangential to the sandbox discussions in order to address them later in the session. A parking lot is a vital tool to help keep the group focused and avoid sidebar conversations, but also to validate a valuable off-topic comment.

- **Business rule log**: This is where notes regarding alternate flows or non-functional requirements that may be global are written. This includes things that apply to the entire Benefit Payment System (BPS) or employer unemployment tax collection system (TAX). Figure 2 shows how we displayed business rules so that the teams could more easily critique them.

- **Bonus step review**: We gathered all initially derived business rules, printed them, and displayed them on a table for review by forty-two participants at a summary meeting (see Figure 2).

- **Terms**: JAD — structured discussions that helped us compose the requirements for the new BPS.
  Actor — a person or system that will interact with the system under development, usually by requesting or sending data.
  Use case — a sequence of actions performed by a system that yields an observable result of value to a particular actor.
Figure 2: Display of derived business rules at the summary meeting. Conducted on Halloween as a "Cat in the Hat Haunted House," this meeting allowed people to look at other groups' work and review their own.

Results of the Inception phase

At the end of the Inception phase for this project, we had results in two forms: 1) the content created, and 2) transformation of that content into a request for proposal (RFP).

Content created

In our sixteen sessions, we created forty-eight use-cases, eighty-seven business rules, thirty-one actors, and 456 glossary definitions. Table 1 provides a list of the use cases and functional groups. We generated use case models using the IBM Rational® XDE product.

Table 1: Use cases listed by functional group.
<table>
<thead>
<tr>
<th>Use Case</th>
<th>Functional Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Initial Claim</td>
<td>Claim Management</td>
</tr>
<tr>
<td>2. Modify Claimant Data</td>
<td></td>
</tr>
<tr>
<td>3. Modify Claim Data</td>
<td></td>
</tr>
<tr>
<td>4. Modify Employer Data for Benefit Claim</td>
<td></td>
</tr>
<tr>
<td>5. Take Reopen Claim</td>
<td></td>
</tr>
<tr>
<td>6. View Wage Data for Benefit Claim</td>
<td></td>
</tr>
<tr>
<td>7. File DUA Initial Claim</td>
<td></td>
</tr>
<tr>
<td>8. File TRA Basic Claim</td>
<td></td>
</tr>
<tr>
<td>9. File TEUC Claim</td>
<td></td>
</tr>
<tr>
<td>10. File EB Claim</td>
<td></td>
</tr>
<tr>
<td>11. Determine Non-Monetary Eligibility</td>
<td>Benefit Eligibility</td>
</tr>
<tr>
<td>12. Enter Benefit Appeal</td>
<td></td>
</tr>
<tr>
<td>13. Create Benefit Issue</td>
<td></td>
</tr>
<tr>
<td>14. Schedule Appeal Hearing</td>
<td></td>
</tr>
<tr>
<td>15. Issue Automated Orders</td>
<td>Benefit Monetary and Payment</td>
</tr>
<tr>
<td>16. Enter Appeals Decision</td>
<td></td>
</tr>
<tr>
<td>17. Claimant/Employer Call-in Notice (E3) for Benefit Claim</td>
<td></td>
</tr>
<tr>
<td>18. Calculate Benefit Monetary</td>
<td>Benefit Monetary and Payment</td>
</tr>
<tr>
<td>19. Transfer Wages Out for Benefit Claim</td>
<td></td>
</tr>
<tr>
<td>20. Transfer Wages In for Benefit Claim</td>
<td></td>
</tr>
<tr>
<td>21. Special Programs for Benefit Payment</td>
<td></td>
</tr>
<tr>
<td>22. Pay Benefits</td>
<td></td>
</tr>
<tr>
<td>23. Modify a Paid Benefit</td>
<td>Benefit Payment Control and</td>
</tr>
<tr>
<td></td>
<td>Charging</td>
</tr>
<tr>
<td>24. Process Benefit Charges</td>
<td></td>
</tr>
<tr>
<td>25. Charges &amp; Credits on Transferred Out Wages for Benefit(J4)</td>
<td></td>
</tr>
<tr>
<td>26. Process Benefit CP Reimbursements</td>
<td></td>
</tr>
<tr>
<td>27. Create a Benefit Overpayment</td>
<td></td>
</tr>
<tr>
<td>28. Establish a Benefit Overpayment</td>
<td></td>
</tr>
<tr>
<td>29. Benefit Overpayment Agent Intercept</td>
<td></td>
</tr>
<tr>
<td>30. State Income Tax Intercept for Benefit Overpayment</td>
<td></td>
</tr>
<tr>
<td>31. Produce Federal Reports</td>
<td>Interfaces</td>
</tr>
<tr>
<td>32. Register with ES</td>
<td></td>
</tr>
<tr>
<td>33. Profiling</td>
<td></td>
</tr>
<tr>
<td>34. RES Cat 2</td>
<td></td>
</tr>
<tr>
<td>35. Claim Status with SASi</td>
<td></td>
</tr>
<tr>
<td>36. Create ES QTR monetary list</td>
<td></td>
</tr>
<tr>
<td>37. External Interfaces</td>
<td></td>
</tr>
<tr>
<td>38. Establish Employer Record</td>
<td>Tax</td>
</tr>
<tr>
<td>39. Establish Investigation</td>
<td></td>
</tr>
<tr>
<td>40. Maintain Suspense</td>
<td></td>
</tr>
<tr>
<td>41. Establish Liable Employer</td>
<td></td>
</tr>
<tr>
<td>42. Maintain Status</td>
<td></td>
</tr>
<tr>
<td>43. Transfer Employer</td>
<td></td>
</tr>
<tr>
<td>44. Maintain Year Quarter Data</td>
<td></td>
</tr>
<tr>
<td>45. Compliance</td>
<td></td>
</tr>
<tr>
<td>46. Employer Appeals</td>
<td></td>
</tr>
<tr>
<td>47. Update Tax with Quarterly Benefit Charges (67A)</td>
<td></td>
</tr>
<tr>
<td>48. Maintain Collections Activity</td>
<td></td>
</tr>
</tbody>
</table>

Transforming content into an RFP

We conducted two validation sessions with participants from each topic area (unemployment insurance benefits and employer tax). The benefits validation session occurred on Halloween, and David and the core team called it a "Use case Cat in the Hat Haunted House." This humorous approach provided a much-needed respite from the six weeks of intense, daily discussions we had just concluded.

Figure 3: During the "Use Case Cat in the Hat Haunted House," each of the thirty-seven benefit payment system use cases was placed on the table for the entire group's review and critique.

The process of casting requirements into an RFP allowed us to further validate those requirements. This intense and focused review produced some changes. We whittled down the more than two hundred business rules to eighty-seven by grouping together multiple rules concerning employer tax identification data and other topics. Also, we rearranged and grouped use cases into an order that reflected the normal process more closely.1

Conclusion

At the end of the Inception phase, Colleen Cochrane, an unemployment insurance manager and a core team member, concluded that, "This process gave structure and form to what very well could have been a swamp of disjointed ideas and information. It really was our 'yellow brick road' to a successful and tangible result."

Naturally, it's great to complete a project with clients who have positive things to say about your team. As for our next project, we will apply the following lessons learned:

- RUP can be light or heavy, agile or SDLCish.2 The form that RUP assumes for your project depends on how you choose to adapt the process for your purposes.
- Business audiences can understand use-case terminology, although it does take about five to six hours for them to become comfortable with the new language.
- Teams need plenty of healthy food and refreshments to make it through a long series of workshops.
- JRP or JAD rooms do not have to be perfect — just accessible and dedicated for your purpose.
- Virtual collaboration is great, but it has not eliminated the need for people to talk together, live and in-person. Although the bounty of available automated, virtual team tools offers essential support and bridging function for team members located in different places, when JRP sessions are conducted in person, they stimulate creating, sharing, collaborating, debating, and sweating over the requirements for a new system. Best results are achieved when you use both virtual and face-to-face techniques.
Notes

1 To view copies of this RFP go to: http://www.das.state.ne.us/materiel/purchasing/80z1.htm

2 SDLC, or system development lifecycle, is a process for developing information systems through investigation, analysis, design, implementation, and maintenance. Also known as information systems development or application development. Source: Webopedia.

About the author

David Kohrell is the president of Technology As Promised, LLC, and a professor at Bellevue University in Omaha, Nebraska. He also serves as president for the PMI Mid-Nebraska Chapter (2002-present). In Technology As Promised LLC (www.aspromised.com), he has created a company focused on developing technology leaders and leading technology into organizations. He has consulted and led teams for projects involving product development, software delivery, network infrastructure, and process engineering at Ameritrade, FiServ, Gallup, Microsoft Press (iUniverse.com), Nebraska State Patrol, the Omaha Police Department, and other organizations. He holds an M.A. in MIS management, an M.A. in community and regional planning, and a B.A. in science from the University of Nebraska. He holds certification as a Microsoft Solutions Framework Practitioner and for MS Project 2000; he also has Project Management Professional certification from the Project Management Institute.
Building the
On Demand Business:
Four Imperatives for
Improved Software
Development

Mike Perrow
IBM Rational Communications
IBM Software Group
Executive Summary

One of the major factors in determining a company’s success in the on demand era is its software development capability. New business needs and consumer expectations that define e-business on demand™ will require a higher level of responsiveness and agility from software development teams. The software applications these teams maintain and develop must be ever-more innovative to keep businesses ahead of the competition.

To support the on demand business and its operating environment, business software applications must:

• Adapt quickly to changing business needs.

• Capture and maintain a strategic advantage.

• Be reliable, scalable, and manageable for the growing on demand business.

To ensure these advantages, businesses must rely on software development organizations to extend and upgrade legacy applications, customize and extend commercially available applications, and develop new applications that offer competitive differentiation in the marketplace. Yet, to leverage maximum value from software through customization, extension, and new development, most organizations must improve their software development capability with techniques suitable to the on demand world.

Specifically, they must embrace four software development imperatives:

1. Develop iteratively — Teams must use a results-oriented process that yields increasingly improved iterations of a software system until it is ready for deployment. This reduces project risk, increases predictability, allows proper scope, and reduces design flaws.

2. Focus on architecture – Teams must create a sustainable architecture that enables development to react to new business requirements without breaking the systems that are already in place. This requires reusable components and service-oriented architectures that can be maintained, upgraded, and replaced without compromising overall system function. A sustainable architecture allows applications to be designed for change, with reduced complexity and integrity.
3. **Continuously ensure quality** – Teams must ensure application fitness at each stage of the development lifecycle. Responsibilities include a sensitivity to the validity and accuracy of evolving requirements, a rigorous commitment to architectural and code quality, and thorough test planning and test execution. Once applications are deployed, teams must continue their vigilance to ensure rapid reaction and proactive, responsive customer service.

4. **Manage change and assets** — Teams must track all changes to the software under development, manage the team's activities, and protect the development assets that are strategic and unique to the business. This shortens development lifecycles by allowing multiple teams to develop in parallel, protects critical assets of the development process, and improves confidence in deployed software.

IBM Rational offers a comprehensive software development solution based on these four imperatives. This solution combines software engineering best practices, market-leading tools, and expert professional services, all of which drive rapid and continuous improvement in software development capability for on-demand businesses.

In addition, IBM Rational offers extensive experience in promoting and delivering *integrated* and *open* software systems, both of which are key characteristics of the on-demand operating environment, -- the means by which business increase their flexibility and IT strengths.

Thousands of companies around the world have realized the benefits of the approach advocated by IBM Rational. Their processes are results-oriented, the artifacts they produce are well-designed and reusable, and they are working at higher levels of capability now required by the on-demand era.
Improving software development capability

Contents

2 Executive Summary

4 Introduction

6 Creating the on demand e-business

9 Driving business value through software applications: The need for software development

10 Four imperatives for improving software development capability

11 The development process for on demand software applications

22 Software development solutions from IBM Rational

28 Conclusion

Introduction

Remember the frenetic pace of the mid-1990s, when nearly every business was adopting an Internet strategy? The possibilities seemed endless. Websites sprung up overnight, promoting every conceivable good, service, and category of information. Banks began offering corporations and individuals access to account information and soon were giving customers the ability to transfer funds between accounts and pay bills. Airlines started moving from simple online flight display to online booking. Businesses of all sorts were busy augmenting their sales channels with secure, interactive storefronts.

Since the dawn of the World Wide Web, companies seeking to exploit its business value have had a lot to keep up with. They’ve not only had to adopt — and adapt to — rapidly changing technologies, they’ve also had to be innovative in order to outshine the competition. In making the first step to basic Internet access, businesses hired Web-savvy teams who could help them quickly respond to a international wave of enthusiasm for the Web. The second step, integration of internal systems to enable Web-based transactions, presented even more challenges. Companies had to automate essential business processes that would take customers beyond the hype of virtual billboards and into the realm of real value exchange.

What transformations will the next few years bring? A new era of business automation has begun, and more and more companies are racing to become a part of it. As one of the pioneers of “e-business,” IBM has a vision for this next phase. It’s called e-business on demand.

In the same way that businesses made the transition from simple Internet access to integrated, Web-enabled business process integration, companies must take a critical third step. They must create an enterprise based on business processes that are integrated end-to-end across the company and across multiple enterprises, linking businesses with key partners, suppliers, and customers. The result? On demand e-businesses will be able to respond with flexibility and speed to any customer demand, market opportunity, or external threat.
To get there, organizations are leveraging the power of software. They must be able to build new applications, integrate, modernize, and extend existing ones, and deploy these applications more quickly than ever before. For this reason, a company’s software development capability is one of the major determining factors in how successfully it makes the leap to on demand computing. Their software development tools and techniques must support the new business demands and consumer expectations that define e-business on demand™. And the applications they use must be ever-more innovative to keep their business ahead of the competition.

This paper will explain how an emerging on demand organization can improve its software development capability. We will examine 1) the relationship between the defining characteristics of e-business on demand™ and the applications that automate and integrate business processes; 2) the software requirements that support the essential operating environment of on demand computing; and 3) how software development is essential in creating strategic advantage. We will consider the characteristics of successful software architectures for organizations that thrive in this new era. Finally, we will explain the unique benefits of using the IBM Rational solution.
I. Creating the on demand e-business

What is an on demand e-business? IBM defines it as one whose leaders can see and manage their company as an integrated whole. This means that all sectors of the business must engage each other in a dynamic transformation of formerly isolated departmental operations into full business processes integrated across the company and outside to their customers.

An on demand business has four essential characteristics\(^1\). It is:

**Responsive** — intuitively responsive to dynamic, unpredictable changes in demand, supply, pricing, labor, and competition.

**Variable** — flexible in adapting to variable cost structures and processes associated with productivity, capital, and finance.

**Focused** — concentrated on core competency, differentiated tasks and assets, with tightly integrated strategic partners.

**Resilient** — capable of managing changes and threats with consistent availability and security.

To attain these characteristics, companies need an operational environment that ties their business processes together, a means to integrate and automate their business processes. But to differentiate your business from those you compete with, these processes cannot simply be based on “out of the box” capabilities provided by packaged applications. Instead, each business must customize their systems in concert with their unique competitive strengths. The benefits of this customization are many. Primarily, it is the key to achieving strategic, competitive advantage over companies locked into vertical, siloed business processes.

---

\(^1\) For more information from IBM about e-business on demand, go to: http://www-3.ibm.com/e-business/index_fl.html
Improving software development capability

A company’s software applications are a key element in becoming an on demand business.

Software development: Essential to the on demand enterprise

The most common types of business applications at work in today’s companies drive a variety of strategic capabilities, including customer relationship management (CRM), supply chain management (SCM), and human resource management (HR). These remain valuable resources in the on demand business. In fact, in the on demand era, these applications are integrated end-to-end across the company; they are also integrated with key suppliers and/or distributors outside the company, delivering greater business value to customers. And the key to creating these valuable, customized integrations is the capability of a business’s software development organization.

In fact, not only is software development key to integrating and automating business processes, but it is a strategic business process in itself. As such, the process of software development benefits from the same type of horizontal integration long applied to supply chain management, customer relationship management, and human resources management processes.

When we compare software development with other business processes, we see that each is composed of activities which were once considered distinct, and supported by different IT system. Over time, business integration has evolved to the point where these activities are now recognized as components of a single, horizontally integrated business process supported by a single integrated application: HR, CRM, etc. (See Figure 1.)

Figure 1: Just as important to running a competitive business as CRM, SCM, and HR systems, software development itself is a strategic business process -- “A collection of activities that takes in one or more kind of input and creates an output that is of value to the customer.”

2 Source: Hammer & Champy, reengineering the Corporation - A Manifesto for Business Revolution, p. 93
An organization’s ability to integrate their applications is a direct function of their software development capability.

Software development is at the beginning of this transformation. Companies are increasingly realizing the value of an integrated software development platform in improving the efficiency of interrelated software development activities. With the introduction of IBM® Rational Suite® software development tools in 1999, IBM was the first to provide this integrated software development solution, and we remain the industry leader.

Let’s examine the role that software applications play in integrating and automating a competitive business. While the on demand business consists of much more than its software applications, the applications ultimately integrate and automate essential business processes. How, then, do a company’s software applications help it to become an on demand business? Essentially, business software applications must meet the following requirements:

Adapt quickly to changing business needs. In the on-demand world, business requirements change more rapidly than in traditional businesses. As business conditions change — new opportunities emerge or new threats appear — a company must be able to modify their business applications even more rapidly to exploit the opportunity or defend against the threat.

Capture and maintain a strategic advantage. A business must leverage the unique qualities that differentiate itself in its market. To fully leverage those differentiators, on demand companies are leveraging the power of software.

Remain reliable, scalable, and manageable. An application only delivers value when it’s running and performing well. Critical applications must function continuously and flawlessly, even while the business is growing, and they must allow for routine maintenance — such as software upgrades and updates — as technology advances continue.
II. Driving business value through software applications: The need for software development

What kinds of software systems are we talking about? In the on demand era, businesses need to leverage all forms of applications to create competitive advantage. They need to maximize the value of their existing legacy systems, customize and deploy commercially available packaged applications, and build new, custom software and applications:

**Legacy applications** — Businesses can’t afford to “rip and replace” legacy systems that still offer unique business rules and valuable custom logic. Their IT departments need to leverage what they have today by modernizing, extending, upgrading, and exposing tried and true business logic to other applications — including today’s ever-more-powerful database management technologies, middleware, and client-side applications.

**Packaged applications** — If businesses purchase commercially available software, they must ensure those applications support the essential, strategic demands of the business. While purchasing a packaged application can save a company time and resources, it is rarely the case that a purchased application can be deployed “out of the box” for any key business function. Businesses must customize and extend these off-the-shelf applications to support the core competencies of their business in competitive ways.
Improving software development capability

New development — Companies seeking true differentiation in the marketplace — especially those operating as on demand e-businesses — will need to develop new software systems that are unique to their business. That software must be designed, built, tested, and deployed, and it must automate and integrate essential business strengths.

All of these activities — modernizing legacy applications, extending packages, and building new applications — are forms of software development. In the on demand era, successful organizations will integrate and automate their business processes by integrating all of their applications. And an organization’s ability to integrate their applications and thus leverage their value through modernization, extension, and new development and deployment is a direct function of their software development capability.

In other words, for a business to succeed in the on demand era, a strong commitment to software development — including the skills associated with integration, customization, and new development — is essential.

Figure 3: To ensure that all applications are strategically connected to business value, software development organizations must modernize, extend, and build them. Each of these types of applications require different modes of software development activity.
III. Four imperatives for improving software development capability

For an organization's software development capability to meet the needs of e-business on demand, software development teams must embrace four specific "software development imperatives":

1. Develop iteratively — Teams must use a results-oriented process that yields increasingly improved iterations of a software system until it is ready for deployment. This reduces project risk, increases predictability, allows proper scope, and reduces design flaws.

2. Focus on architecture — Teams must create a sustainable architecture that enables development to react to new business requirements without breaking the systems that are already in place. This requires reusable components and service-oriented architectures that can be maintained, upgraded, and replaced without compromising overall system function. A sustainable architecture allows applications to be designed for change, with reduced complexity and integrity.

3. Continuously ensure quality — Teams must ensure application fitness at each stage of the development lifecycle. Responsibilities include a sensitivity to the validity and accuracy of evolving requirements, a rigorous commitment to architectural and code quality, and thorough test planning and test execution. Once applications are deployed, teams must continue their vigilance to ensure rapid reaction and proactive, responsive customer service.

4. Manage change and assets — Teams must track all changes to the software under development, manage the team's activities, and protect the development assets that are strategic and unique to the business. This shortens development lifecycles by allowing multiple teams to develop in parallel, protects critical assets of the development process, and improves confidence in deployed software.
Improving software development capability

The relationship of these software development imperatives to the applications and operating environment they support is shown in Figure 4:

**Figure 4**: Software development imperatives support adaptive, strategic, and reliable applications within the on demand operating environment.

---

**IV. The development process for on demand software applications**

As shown in Figure 4, e-business on demand is based on a logical relationship of 1) business fundamentals, supported by 2) an operating environment and business applications, supported in turn by 3) software development imperatives that drive the business applications. To express this relationship more simply:

To improve your business...⇒...exploit the power of software...⇒...by improving software capability.

But as a means toward improved software development capability, what is entailed in these software development imperatives? If an organization is developing software, what, specifically, is required from development teams and processes? Let's examine each of the software development imperatives outlined in the previous section in more detail, and explore the benefits for the software which supports an on demand business.
1. Develop Iteratively

An iterative development process yields increasingly improved versions of a software system until it is ready for deployment. Each iteration includes a mix of analysis, design, construction, and testing, which results in a demonstrable form of the software that can be validated and refined. Development teams start by addressing a limited set of functions pertaining to the high-risk areas of the project. With each iteration, teams add functionality and produce a working version of the application. Each iteration verifies the system architecture, the application’s ability to satisfy its requirements, and the quality of the software.

By adopting an iterative development process, software development teams are able to keep the ultimate users of the application apprised of progress, since a working version of the system under development is always available (demonstrable) for inspection. This is an important concept: Software teams stay focused on results. Moreover, teams using a mature iterative development framework, such as the Rational Unified Process® methodology, or RUP®, employ a flexible process — one that is customizable to any size project. This flexibility means that teams can adjust their development process quickly as business needs change and requirements for new applications arise.

An iterative, “results-oriented” process also represents a significant advance over “activity-oriented” processes, such as the traditional “waterfall” method, which mandates that each phase of the development process — e.g., design — is finished before the subsequent phase — coding — begins, so the output of one phase “washes down” to the next. This traditional process also requires teams to work separately on different parts of the system until the final phases of assembly and system testing — and these phases are when mistakes are discovered, code must be scrapped and reworked, and disappointment becomes inevitable (see Figure 5).

By contrast, an iterative development process offers significant benefits for building applications for the on demand operating environment, as follows:

Reduces project risk — i.e., the most difficult aspects of system performance and meeting end-user requirements are tackled early. Teams start with a limited set of requirements at the outset, which allows them to focus on the hard things first. With all major
Improving software development capability

14

Figure 5: In a traditional waterfall project, quality can only be assessed according to details of the plan, not product functionality, because the software isn't assembled and tested until the end of the project. At that point, teams typically start over and rework code, which means missed deadlines and high costs, or they deliver sub-optimal applications.

Figure 6: With an iterative process, each iteration (dark gray dot) is a working version of the software under development, which allows the development team to understand "where they are" in the process and make corrections as needed.

risks addressed and mitigated early on, fewer “show-stopper” mistakes are likely in the later phases of the project, and deadlines — which are no longer negotiable in on demand business — are more easily met.

Increases predictability — Using a traditional “waterfall” process, software development teams cannot fully assess product quality until the final stages of assembly, when a working version of the software becomes available. If errors are discovered at that late stage — and they always are — teams must either rework the application (incurring high costs and missing customer deadlines in the process), or deliver an application that does not meet customer expectations. With iterative development, teams can more accurately predict outcome because each iteration is designed, coded, and tested, and the team “zeros-in” on the correct result. If the project strays off-course at any point in the process, teams are better able to make adjustments in the next iteration. (See Figure 6)
Improving software development capability

Allows proper scope — Iterative development allows project managers to scope each new software iteration with its incremental improvements, instead of scoping the entire project from its inception to completion. This ensures that the teams focus on the most important aspects of the application and ensure the end-result is a high-quality system.

Reduces design flaws — By addressing and mitigating the highest risks first, and by continuous verification of software quality through iterative testing, software projects — including integration efforts, new applications, and extensions to legacy systems — yield higher quality applications that are more reliable.

Figure 7: An iterative development process supports the major objectives of applications running in the on demand operating environment.
For an on demand business, software architecture becomes doubly important as the business adapts its applications to meet changing needs.

Iterative Development: The Benefits to Business Applications

An iterative development process is essential in meeting the major objectives of applications running in the on demand operating environment. Because iterative development reduces risk and increases predictability, software development teams are better able to quickly adapt applications — whether these are pre-existing, pre-packaged, or new applications created by the business — to meet rapidly changing business needs. Thus return on application investment is more rapidly achieved. Because an iterative process allows teams to better manage the scope of their project, organizations can ensure they are delivering applications that meet the most strategic, high-leverage aspects of their business. And with reduced design flaws at the end of an iterative project, IT managers can be much more confident that the applications they deploy will remain reliable and scalable for growth over time. Figure 7 shows the relationship between iterative development and the major objectives of applications running in the on demand operating environment.

2. Focus on Architecture

An application’s architecture is the most critical determinant of its success or failure. A properly designed architecture ensures the application will meet the business needs, perform acceptably, scale with the business, and be adaptable over time. A poor architecture results in applications that are inflexible, fragile, unreliable, and expensive to maintain.

Software architecture becomes doubly important as a company seeking to become an on demand business adapts its applications to meet changing needs. A solid, well-understood architecture provides the foundation for the critical twenty percent of all artifacts (requirements, components, etc.) that will drive the overall success of a project. It affords a stable development environment so that teams are able to add new functionality over time without risking breakage of the overall system. It also allows for changing various components within the development environment as new technologies need to be added and as old ones need to be replaced.

For software development teams, a proper architectural focus allows the benefits of component-based design, including service-oriented architectures, which helps teams achieve higher levels of code reuse, project
Improving software development capability

after project, than ever before. Moreover, they can make better use of rapid application development techniques within a more structured production environment.

How does sound architecture afford these benefits?

A component is a cohesive set of pre-existing lines of code, either in source or executable format, with a defined interface and behavior. A software architecture based on components with well-designed interfaces allows more rapid change, because any of its components can be modified without affecting the rest of the application. A rapidly growing trend in software architecture design is the “Service-Oriented Architecture” (SOA) model. SOAs allow companies to integrate their internal systems according to component-based design principles. As these same companies design and build systems for their customers and partners, SOAs allow existing components to be reused with selected functionality exposed to inside and outside parties.

Without a sound architecture to back them up, teams using rapid application development (RAD) techniques typically produce limited results. However, an “architected” RAD (ARAD) approach provides a unified environment for all team members that allows integration of legacy and enterprise systems and development of scalable n-tiered applications. Teams are able to take advantage of templates and patterns for industry based solutions, as well as foundation classes that help jump start development projects.

Designing an effective architecture can be a challenging task that typically involves a number of participants. Effective communication, unambiguous definition, and a reliable means for capturing and modifying the design are critical. The industry has produced a standard — the Unified Modeling Language (UML) — to satisfy these needs. Using the UML, teams can produce an accurate, graphical representation of an application’s architecture. These models can be shared with others who will immediately understand them because of the precision in UML notation. The quickest and most effective way to produce these models is to use a visual modeling tool, which is essential to building a quality architecture within complex technology environments. They are far more than just drawing tools; they provide semantic language properties to enforce rules providing consistency and functional integrity of the models. They can also
Improving software development capability

For the on demand business itself, a focus on architecture yields the following important benefits:

**Design for change** — Component-based architectures, including emerging service-based architectures (SOA) designed to support Web Services and grid computing, allow the software to be changed quickly, because components can be switched readily, or modified, without compromising overall system integrity. This is a fundamental requirement to support responsiveness and resiliency for the on demand business.

**Reduced complexity** — Visual modeling allows project managers to focus on the core requirements of a software application, thus leaving the functional details to the coding specialists on the team. Component-based architectures allow interactions to occur between the various parts of a system without requiring the exposure of methods and details between all functional “chunks of code.” Together, visual modeling and component-based design allow all teams to work at the right level of “abstraction” — i.e., the level of complexity best suited to their role in the project.

**Integrity and quality** — Experience shows that, more than anything else, an application’s architecture determines its quality. A good architecture allows the application to be modified over time. Thus, for businesses competing in the on demand era, the ability to deliver quality applications with the flexibility to handle rapid change will help ensure long-term business success.

**Figure 8:** A focus on architecture drives key benefits for applications driving the on demand business.
Improving software development capability

Architectural Focus: The Benefits to Business Applications
An architectural focus supports the major objectives of applications running in the on demand operating environment. Because a well-designed architecture allows applications to be designed for change, they are able to adapt more quickly to the rapidly evolving needs of the business. Applications that can be rapidly changed and integrated with high confidence allow the organization to focus on its strategic differentiators as they evolve in the competitive business arena, ensuring that organizations get the highest ROI from their business applications.

And because good architecture improves an application’s integrity and quality, they naturally become more reliable and scalable. Figure 8 shows the relationships between a strong architectural focus and the major objectives of applications running in the on demand operating environment.

3. Continuously ensure quality

Software testing accounts for at least 30 percent of total software development costs, but despite this high expenditure, software bugs -- mistakes discovered after product deployment -- cost the US economy an estimated $59.9 billion annually. One reason for this high “cost of quality” is the tendency for development organizations to conduct testing late in the project lifecycle, often after isolated teams have completed their work, modules are assembled, and testing teams begin the long process of discovering defects or noncompliance issues. This eleventh-hour testing strategy usually leads to code being scrapped, reworked, recompiled, then retested, and it naturally causes projects to run over time and over budget. Another reason for the high cost is the difficulty organizations have tracking and reproducing errors. Often, finding a bug itself is quite simple, but communicating all of the details necessary to reproduce and fix that bug can take hours or days. Sometimes, a developer cannot reproduce the bug based on the information provided, which can lead to operational downtime -- including unproductive finger-pointing between quality assurance and development teams.

3 Source: National Institute of Standards and Technology (NIST) news release, “Software Errors Cost U.S. Economy $59.5 Billion Annually,” June 29, 2002
To remedy this, project teams must work to prevent, detect, diagnose, and remove defects throughout the application development and deployment lifecycle, not just at the end of it. We refer to this as the need to continuously ensure quality. The emphasis here is to address quality concerns earlier in the project life cycle -- when errors are significantly less costly to fix -- and to close the loop between the operations team that finds errors, and the software development team that diagnoses and repairs them.

Continuously ensuring quality also means stopping defects before they are coded. Many software applications fail to satisfy the client’s requirements. Validation of requirements before coding even begins helps prevent significant rework that often occurs when the client and development teams realize that requirements were misstated or misunderstood. Also, after software is deployed for production, the need to ensure that it continues to function as intended in areas like performance is significant to the project’s success. In other words, continuously ensuring quality means “even after the software is up and running.”

Commercially available tools and techniques can help teams discover and track errors, improve test coverage, and conduct regression tests (test that ensure bug fixes or other improvements to code under development do not themselves introduce errors). Yet the tools themselves are only effective when put to use by a team that has adopted a philosophy of quality, one which is shared and practiced across the entire development organization. The more software development teams work to bridge the gap between analysis, development, testing, and operations, the more it will function as a quality-aware organization capable of reducing organizational downtime, rapidly detecting faults, and automating subsequent rebuilding and redeployment.

By continuously ensuring quality, an on demand business derives the following benefits:

- **Detect and respond to changing business drivers** – Through iterative reassessment of the quality of customer use cases, requirements, and architectural models as well as code, teams remain sensitive to evolving customer needs and thus can rapidly respond. This ensures that the software not only functions as specified, but also continues to solve the right problem.
Improving software development capability

• **Delivers timely, optimized realization of unique differentiators** – Because systems are continuously changing, a process of continuous validation is required to ensure that software continues to differentiate effectively. The more component-based and interdependent our systems become, the more important it becomes to continuously validate functionality through an automated regression testing program.

• **Maintains and exceeds SLA-compliance** (“service-level agreement” established with customer) – Many of the factors that influence user experience can and do change once the application has been deployed – for example, external Web services, operating environments, and partner and supplier integrations. Only by verifying application performance post-deployment can we ensure the quality of the end user’s experience over time.

**Figure 9:** Too often, software development organizations do not test for software quality until late in the application lifecycle. By continuously ensuring quality from the beginning, teams are better able to deliver expected results to internal and external customers.

---

**Continuously Ensure Quality**

- Build quality from the beginning
- Automate testing
- Bridge development, testing, and operations teams

**Business applications**

- Adapt quickly to changing business needs
- Create strategic advantage
- Are reliable, scalable, and manageable

**Continuously ensure quality to...**

- Detect and respond to changing business drivers
- Deliver timely, optimized realization of unique differentiators
- Maintain and exceed SLA-compliance

**Continuously ensuring quality: The benefits to business applications**

Essentially, continuously ensuring quality means that organizations must strive to build quality into their products and IT systems from the beginning of the iterative lifecycle. This effort pays off at the highest levels of the business, because business systems that are tested “early and often” are more likely to satisfy customer needs, and function as intended.
At the technical level, software development organizations dedicated to continuously ensuring quality experience better communication and understanding across development, testing, and operational teams, which speeds overall time to value for business applications. As a result, all business applications benefit from improved architectural models, a better understanding of user requirements, and more sound and reliable code. Figure 9 shows the relationship between continuously ensuring quality and on demand business applications.

4. Manage Change and Assets

Embedded in the very concept of e-business on demand is the notion of change. Quick response to new opportunities, customer demands, or threats to security or overall business stability is an essential, defining capability in the on demand era. But “responsiveness” in terms of on demand software development does not mean “reactionary.” Instead, a managed response to change, which also guards against corruption of assets created during the development process, enables project managers to keep the process of successive iterations running smoothly. Software configuration management (SCM) is a key capability in modern software development practice. It allows teams to carefully trace requirements over the project lifecycle, during which numerous changes — including changes to the requirements themselves — occur.

Change and assets management offer the following key advantages for on demand software development:

**Enables virtual teams and parallel development** — Advanced SCM systems allow multiple, sometimes overlapping, branches of a project to be worked on by different development teams simultaneously, so more work can be accomplished faster, on demand, without sacrificing quality.

**Protects critical assets** — A company’s software development assets — requirements documents, design models, source code, automated test suites, etc. — are unique, strategic resources that cannot be purchased or recreated from outside sources. Just as valuable as a corporation’s business assets, these software development artifacts must be managed and protected. Effective change management systems ensure that no unit of code or component under
Improving software development capability

development is ever lost or over-written. This affords an important safeguard against the threat of security breaches or disaster.

Allows confidence in software deployment — Change and assets management ensures that teams who are building and maintaining complex systems remain in sync as they combine multiple versions and various pieces of software code. Change management systems also allow all requirements to be traced throughout the project lifecycle, so that the high-level architecture translates to a software system focused on user expectations.

Change and Assets Management: The Benefits to Business Applications Managing changes and assets during the software development process is required to meet the major objectives of applications running in the on demand operating environment. Virtual teams and parallel development capability means faster project turn-around times to meet rapidly changing business needs. Because the software development assets are as strategic to the business as the applications they create, protecting and managing them is central to delivering strategic value. And greater team confidence in deployed software speaks for itself. Figure 9 shows the relationships between change and asset management and the major objectives of applications running in the on demand operating environment.
In all areas of the software development life cycle, our customers claim improvements. Customers claim reductions in time to value, report improved project management capabilities, and claim tremendous increases in productivity.

Conclusion

For a radical transformation of business systems to succeed, technology managers need a clear understanding of the business goals their applications and operating environment must support. The IBM on demand vision is compelling, and its achievement will transform not only business systems, but also the ways in which customers and partners across the globe interact with corporations and with each other. Every industry will benefit, with higher quality, more affordable products and services available worldwide.

To leverage the full potential of e-business on demand, the leaders of software development organizations must incorporate proven principles that will ensure their success in the on demand era. That means they must focus on four imperatives for successful software development:

- Develop iteratively
- Focus on architecture
- Continuously ensure quality
- Manage change and assets

Deeply committed to the IBM Software Development Platform, IBM Rational delivers software engineering best practices, development tools, and professional services to today’s organizations seeking the software development capability that these four imperatives represent. Thousands of companies around the world have realized the benefits of this approach. Their processes are results-oriented, the artifacts they produce are well-designed and reusable, and they are working at higher levels of capability than ever before. These companies are realizing the promise of the on demand era.
IBM software integrated solutions

IBM Rational supports a wealth of other offerings from IBM software. IBM software solutions can give you the power to achieve your priority business and IT goals.

- **DB2® software** helps you leverage information with solutions for data enablement, data management, and data distribution.

- **Lotus® software** helps your staff be productive with solutions for authoring, managing, communicating, and sharing knowledge.

- **Tivoli® software** helps you manage the technology that runs your e-business infrastructure.

- **WebSphere® software** helps you extend your existing business-critical processes to the Web.

- **Rational® software** helps you improve your software development capability with tools, services, and best practices.
Requirements: An introduction

Scott McEwen
Metasys Technologies, Inc.
2 Apr 2004

from The Rational Edge: Accurate requirements are an essential part of the formula for software project success. This article explains why, and describes a three-fold approach to effective requirements documentation.

A Fortune 100 company embarked on a project to design and build a sophisticated software package that it would ultimately deploy to its offices throughout the world. Two years and about $10 million later, the field offices refused to use the software because it didn’t do what it was intended to do. Instead of helping to streamline an important business process, the software actually hindered it.

In another case, a leading software integrator was awarded a contract by the procurement organization for a major utility company. Later, that organization was shocked when the integrator informed it that, based on “true” client requirements, the project’s scope had increased twofold.

Do these stories sound familiar? Why do things like this happen? According to a recent survey by the Standish Group of more than 352 companies reporting on more than 8,000 software projects:

- 31 percent of all software projects are canceled before they are completed (a waste of $81 billion).
- 53 percent of projects cost 189 percent of their original estimate.
- In large companies, 9 percent of projects are on time and within budget.
- In small companies, 16 percent of projects are on time and within budget.

The Standish Group survey also asked respondents to identify the causes of these failures. Table 1 shows the top three reasons why projects are “impaired.”

<table>
<thead>
<tr>
<th>Project impairment factors</th>
<th>% of Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lack of user input</td>
<td>12.8%</td>
</tr>
<tr>
<td>Incomplete requirements and specifications</td>
<td>12.3%</td>
</tr>
<tr>
<td>Changing requirements and specifications</td>
<td>11.8%</td>
</tr>
</tbody>
</table>

As this table shows, poor requirements are the biggest problem. If it isn’t clear what you are supposed to build, how can you estimate the...
cost of building it? How can you create a project plan, assign resources, design system components, or create work orders? You need accurate requirements to perform these activities. Of course, requirements evolve as a project proceeds, but carefully worded basic requirements provide a starting point. Then, as the project progresses, you can fill in details and update planning documents as the requirements evolve.

So what is a requirement? This article attempts to explain this commonly misunderstood term. Rather than supplying a definition up front, we will discover one by understanding first why we need requirements, and then examining the content of the different documents we use to capture them.

Why do we need requirements?

We use requirements for a variety of purposes, including:

- Project scoping
- Cost estimating
- Budgeting
- Project scheduling
- Software design
- Software testing
- Documentation and training manuals

Individuals throughout an organization have a vested interest in producing solid requirements. Whether you're a client or involved in procurement, finance and accounting, or IT, you are a major stakeholder in the requirements management process.

Many project teams treat requirements as a statement of purpose for the application and express them in very general terms, such as: "The system should have the ability to create problem tickets for outage notifications." But is this a solid requirement? To answer this question, let's look at how we document requirements.

Classifying and documenting requirements

Requirements are not requirements unless they are written down. In other words, neither hallway conversations nor "mental notes" constitute requirements. We typically capture requirements in three separate documents:

1. Stakeholder Needs
2. Software Features
3. Software Requirements Specification

At my organization, we associate about a half dozen attributes (e.g., priority, status, etc.) with each requirement to help with decision making, scheduling, and so on. The information contained in one requirement document should be referenceable in the others. For example, the information recorded in the Software Features document should support and be traceable to one or more items listed in the Stakeholder Needs document.

To better understand the relationships among these documents, let's return to my earlier question about whether the statement, "The system should have the ability to create problem tickets for outage notifications" is a valid requirement. The answer is, "Not yet." What this statement expresses is a need. Capturing this need is a step toward formulating a solid requirement, but the statement cannot stand alone; you must first translate it into one or more features that you capture in a Software Features document. Those features, in turn, must then be detailed in the Software Requirements Specification document.

Using these three separate documents also helps to simplify the process of requirement reviews. Although an executive manager might be a reader/approver for the Stakeholder Needs and Software Features documents, he/she may delegate responsibility for reading and approving the more detailed Software Requirements Specification. Maintaining separation among these different documents allows specific readers to understand specific parts of the system. It also promotes better accountability — a key element for a successful software development process.

Documenting stakeholder needs

Let's look at what each of these documents contains (see Figure 1). We'll start with the Stakeholder Needs document.
As we describe what to capture in each document, keep in mind that whatever needs and requirements you formulate at the outset of your project will evolve as your project proceeds. If you are using an iterative development approach, you should assess your requirements after each iteration, and if you make changes in one document, you should update the others as well to maintain consistency.

Figure 1: Requirements categories

Stakeholder needs, which are part of the problem domain, describe what stakeholders require for a successful project. In other words, needs describe what the application should do to help improve or lower the cost of a business process, increase revenue, or meet regulatory or other obligations.

Documenting stakeholder needs involves identifying, understanding, and representing different viewpoints. Often, users and stakeholders don't know how to solve the entire problem but are experts at explaining what they need to do their job better. Each stakeholder sees the problem from a different perspective. Therefore, you must understand the needs of all stakeholders in order to understand the entire problem domain.

The first step is to identify all stakeholders. Users represent a class of stakeholders, but by no means do they represent the interests of the whole organization. Other classes of stakeholders may come from finance and accounting, procurement, and IT, as well as from other departments or organizations that directly or indirectly support or benefit from the project.

You should identify (and recruit) at least one representative from each stakeholder class who will speak for the entire class. Also, document your list of stakeholders so that everyone knows who is representing each class.

You can elicit needs from stakeholders using various techniques, including one-on-one meetings, questionnaires, storyboarding, and Joint Application Development (JAD) sessions. Explanations of these specific techniques would be beyond the scope of this article, so for now, just be aware that how you ask questions and the format you use are important aspects of the process.

Let's look at a hypothetical project aimed at streamlining a help desk application for a major corporation's IT department; we'll use this project as an example throughout the remainder of this article. Imagine that you, a project team member, have met with the help desk manager and formulated a requirement that says, "He needs to be able to increase the number of support calls his team can handle by 30 percent, without increasing headcount."

Note that this need requirement provides little detail, but it clearly conveys what the client wants at a high level. Ambiguity is expected at this stage; you will capture more detail later.

But not all the needs you gather will describe system functionality. For example, a stakeholder from procurement or finance might say, "The budget for the initial implementation of the application help desk project cannot exceed $350 thousand." Of course, this perfectly valid need might conflict with other stakeholders' needs that might cause the budget to exceed $350 thousand; resolving conflicting needs is a normal part of the requirements management process. However, in the beginning, you should focus on eliciting and recording the perspective of each stakeholder; conflict resolution can come later in the process.

**Documenting software features**

After you have defined stakeholder needs, you must translate them into a set of distinct system features. What's the difference between...
needs and features? Needs do not indicate a particular solution; they simply describe the business need. For example, if a stakeholder says, "We need to streamline the help desk’s application support process because we can't keep up with the calls," that person is expressing a need that the development team can translate into a feature.

However, if the stakeholder says, "We need a Web-enabled system so that customers can enter their own support requests," the stakeholder has already translated the need into a feature. It is perfectly fine for stakeholders to express themselves in any way they wish; often, you will want to ask additional questions to clearly understand both needs and features. I'll explain why in a moment. For now, let's define what a feature is.

A feature is a service that the system provides to fulfill one or more stakeholder needs.2

It is important for the development team to understand the distinction between needs and features and to record them in separate documents. Why must they separate needs from features? Needs are part of the problem domain, and features are part of the solution domain. It is critically important to fully understand the problem domain before deciding on a solution; often, you will find opportunities to generalize the solution once you fully understand the problem. In other words, by separating needs from features, you can find a common set of features that will meet multiple needs. Like the Stakeholder Needs document, the Software Features document should be available to all team members throughout the process. And it is important to maintain traceability from each feature to its corresponding need(s).

Let's return to our example of a help desk support application. Table 2 shows three stakeholder requests expressed as needs.

Table 2: Stakeholder needs

<table>
<thead>
<tr>
<th>ID</th>
<th>Need</th>
<th>Stakeholder</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>&quot;Need to notify support manager when a 'support request' is initiated.&quot;</td>
<td>Support manager</td>
</tr>
<tr>
<td>N2</td>
<td>&quot;Need to assign support request to appropriate support engineer.&quot;</td>
<td>Support manager</td>
</tr>
<tr>
<td>N3</td>
<td>&quot;Need to keep customer informed of the progress of a support request.&quot;</td>
<td>Customer (user)</td>
</tr>
</tbody>
</table>

Table 3 shows the corresponding features mapped to these needs.

Table 3: System features mapped to stakeholder needs

<table>
<thead>
<tr>
<th>ID</th>
<th>Feature</th>
<th>Description</th>
<th>Maps to</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>The system will be workflow oriented.</td>
<td>The 'support request' will go through a series of stages and assignments.</td>
<td>N1, N2, N3</td>
</tr>
<tr>
<td>F2</td>
<td>Email notification capability</td>
<td>A centralized email notification system that will be utilized by the workflow engine.</td>
<td>N1, N2, N3</td>
</tr>
</tbody>
</table>

Keep in mind that this is a highly simplified example. Complex systems can involve lots of stakeholders, external system interfaces, complex workflows and analytics, and other elements that make translating needs into features far more difficult.
Documenting software requirements

After you analyze and generalize needs and features, it's time to move deeper into the solution domain by analyzing and capturing the system requirements. Now we have enough understanding to define a requirement as:

…a software capability that must be met or possessed by a system or a system component to satisfy a contract, standard, or desired feature.3

Simply put, requirements must satisfy one or more of the following criteria:

1. Contract obligations
2. Standards
3. Desired needs and features

We can classify the requirements themselves into two categories: functional requirements and non-functional requirements.

Functional requirements present a complete description of how the system will function from the user's perspective. They should allow both business stakeholders and technical people to walk through the system and see every aspect of how it should work — before it is built.

Non-functional requirements, in contrast, dictate properties and impose constraints on the project or system. They specify attributes of the system, rather than what the system will do. For example, a non-functional requirement might state: “The response time of the home page must not exceed five seconds.”

Here are some qualities that should characterize the descriptions in your Software Requirements Specification document:

1. Lack of ambiguity. The software development team will be unable to produce a product that satisfies users' needs if one or more requirements can be interpreted in multiple ways.
2. Completeness. In the beginning of your project, you should not expect to know all the system requirements in detail; the development team should not waste time trying to specify things that are bound to evolve. As the project proceeds, however, you should keep your Software Requirements Specification document up to date; as you gain more knowledge about the system, the specification document should grow more complete.
3. Consistency. You cannot build a system that satisfies all requirements if two requirements conflict or if the requirements do not reflect changes that were made to the system during the iterative development and functionality testing.
4. Traceability. The team should track the source of each requirement, whether it evolved from a more abstract requirement, or a specific meeting with a target user.
5. No design information. As long as requirements address external behaviors, as viewed by users or by other interfacing systems, then they are still requirements, regardless of their level of detail. However, if a requirement attempts to specify particular subcomponents or their algorithms, it is no longer a requirement; it has become design information.

Capturing functional requirements

To document functional requirements you must capture three categories of information:

1. Use cases
2. Functional capabilities
3. Business rules

Use cases define a step-by-step sequence of actions between the user and the system. Organizations are rapidly adopting use cases as a means to communicate requirements because they:

- Are easier to create, read, and understand than traditional functional specifications.
- Show how the system will work from the users' perspective rather than the system's perspective.
- Force us to think about the end-game: What is the user trying to accomplish by using the system?
- Require us to define how the system should work, step-by-step.
- Provide an excellent basis for building test cases and helping to ensure that these are built before the code is written.
- Provide a common requirements "language" that's easy for stakeholders, users, analysts, architects, programmers, and testers to understand.
The end result of a use case is a complete requirement. In other words, when you communicate via uses cases, you don't leave it up to the developers to determine the application's external behavior. Specifying the format and details for creating a use case goes beyond the scope of this article, but it is important to capture use cases using a standard template that contains all the components of a complete specification. These include a use case diagram, primary and assisting actors, triggering events, use case descriptions, preconditions, post conditions, alternative flows, error and exception conditions, risks and issues, functional capabilities, and business rules.

Note that use cases do not result in requirements until you define functional capabilities and any business rules that apply to the use case.

**Functional capabilities** define what specific action the system should take in a given situation. You can relate functional capabilities directly to a specific use case or define them globally for the entire system. A functional capability for our example application might be, "When creating the support request, populate the "created by" field with the user's logon id."

**Business rules** state the condition under which a use case is applicable and the rule to be applied. For instance, a business rule related to a use case might state, "Only the system administrator may modify the name of the customer in use case UC01." Like functional capabilities, business rules can be directly related to a use case or defined globally for the entire system.

**Capturing non-functional requirements**

Non-functional requirements are attributes that either the system or the environment must have. Such requirements are not always in the front of stakeholders' minds, and often you must make a special effort to draw them out. To make it easier to capture non-functional requirements, we organize them into five categories:

1. **Usability**
2. **Reliability**
3. **Performance**
4. **Supportability**
5. **Security**

**Usability** describes the ease with which the system can be learned or used. A typical usability requirement might state:

- The system should allow novice users to install and operate it with little or no training.
- The end user shall be able to place an order within thirty seconds.
- The end user shall be able to access any page within four seconds.

**Reliability** describes the degree to which the system must work for users. Specifications for reliability typically refer to availability, mean time between failures, mean time to repair, accuracy, and maximum acceptable bugs. For example:

- The system shall meet the terms of a Service Level Agreement.
- The mean time to failure shall be at least four months.

**Performance** specifications typically refer to response time, transaction throughput, and capacity. For example:

- All Web pages must download within three seconds during an average load, and five seconds during a peak load.
- While executing a search, the system must be able to display 500 search results per page.

**Supportability** refers to the software's ability to be easily modified or maintained to accommodate typical usage or change scenarios. For instance, in our help desk example, how easy should it be to add new applications to the support framework? Here are some examples of supportability requirements:

- The system shall allow users to create new workflows without the need for additional programming.
- The system shall allow the system administrator to create and populate tax tables for the upcoming tax year.

**Security** refers to the ability to prevent and/or forbid access to the system by unauthorized parties. Some examples of security requirements are:

- User authentication shall be via the corporate Single Signon system.
- Only authorized payroll administrators shall be permitted to access employee pay information.
Conclusion

In a software development project, requirements drive almost every activity, task, and deliverable. By applying a few key skills and an iterative development approach, you can evolve requirements that will help ensure success for your project. Use separate documents to record needs, features, and requirements, and improve the accuracy of your requirements by sharing responsibility for review. With these documents you can also establish traceability between needs, features, and requirements to ensure that your Software Requirements Specification will continue to match up with business objectives.

It is typically very costly to fix requirement errors that remain undiscovered until all the code has been written. Use cases can help you avoid such errors by communicating requirements to all project stakeholders; developing the system in iterations will help you identify what requirements you might need to specify in more detail or change.

Remember: When gathering stakeholder input, ask about non-functional considerations as well as requests for specific functionality. And keep in mind that solid requirements are themselves the most important requirement for producing useful, high-quality software.

Notes


3 Dean Leffingwell, op.cit.

About the author

Scott McEwen, director of business solutions at Metasys Technologies, is an expert in project and requirements management. He has more than fifteen years of experience in software project management, requirements management, and software engineering. He can be reached at smcewen@metasysinc.com.

Metasys Technologies is an Atlanta-based consulting firm that helps clients improve the quality and reduce the cost of building software. It offers expertise in requirements management as well as training, coaching, and seminars designed to quickly improve the productivity of software development teams. For more information, visit www.metasysinc.com.
IBM Rational’s Bran Selic to demo UML 2.0 at OMG

15 Apr 2004

from The Rational Edge: OMG workshop to showcase model-driven architecture (MDA), scheduled for May 17-20, 2004.

Next month, Unified Modeling Language guru and IBM Rational thought leader Bran Selic will conduct a tutorial in UML 2.0 as part of the Object Management Group’s upcoming "Model Driven Architecture® Implementers' Workshop — Succeeding with Model Driven Systems."

Set to take place May 17-20, 2004 in Orlando, FL, USA, this OMG-sponsored workshop will feature dual tutorial tracks, new case studies, and business solutions centering around a model-driven approach to designing software systems. Since MDA was publicly announced three years ago, the concept has gained traction as more organizations need to ensure that their mission-critical information systems are rooted in standards that will adapt to ever-changing hardware capabilities and software platforms.

Open to all, OMG's MDA Implementers' Workshop offers an excellent opportunity to learn more.

To see a complete program schedule and to register to attend, visit http://www.omg.org/news/meetings/MDA2004/. For more information contact Kevin Loughry, Director of Event Management at loughry@omg.org.

What do you think of this document?

- Killer! (5)
- Good stuff (4)
- So-so; not bad (3)
- Needs work (2)
- Lame! (1)

Comments?
The project pyramid

Joe Marasco
Former senior vice president, Rational Software
12 Apr 2004

from The Rational Edge: Joe Marasco proposes a model for predicting project success based on a familiar geometrical shape. Probability of success is measured by a pyramid's altitude, and four production factors — scope, quality, speed, and frugality (or limits on resources) — form the four sides of the base.

I recently read about "the iron triangle" and its extensions on Max Wideman's excellent Web site on project management wisdom, www.maxwideman.com. I've long been fascinated by the now famous "scope, resources, time — pick any two" paradigm, which states that trying to maximize scope while simultaneously minimizing resources and time will impose too many constraints and inevitably lead to project failure.1

Max makes the important point that we need to add a critical fourth dimension — quality — to the paradigm. As he wrote to me,

Interestingly, quality ultimately transcends all else, whether in terms of performance, productivity, or final product. But a remarkable number of people in the project management industry don't seem to have latched onto that. Who cares if last year's project was late and over budget? That's all lost in last year's financial statements. But the quality [of the product] is enduring.

It is hard to argue with that point of view. Most of us software developers can recall some time when, in our zeal to make our commitments and ship on time, we let stuff get out the door that caused us heavy regrets later on.

So Max extends the iron triangle to a star, as shown in Figure 1.

Figure 1: Max Wideman's extension of the "iron triangle" — resources, scope, and time — introduces "quality" as a fourth element.

As an alternative to this star, Max's correspondent Derrick Davis suggests using a tetrahedron to illustrate these relationships. This allows you to maintain the original triangle but create a third dimension to depict the quality aspect. The nice thing about the tetrahedron is its...
intrinsic symmetry; the four attributes populate the vertices, and any three can be used as the base. Max has illustrated this in a thoughtful way, tying the vertex pairs together with another descriptor (see Figure 2).

Figure 2: The tetrahedron model allows any three attributes to serve as a base, placing the fourth attribute in the third dimension.

Five, not four

Although I agree with Max's insistence on quality as a critical fourth factor, I believe that his model still leaves something to be desired. When thinking about a project prior to beginning work on it, management is typically interested in the "shape" of the project — an interest that maps nicely to the four parameters illustrated in Figures 1 and 2. That is, we can state how much we intend to do (scope); we can describe how well we are going to do it (quality); we can predict how long we will take to complete the project (time); and we can estimate how much it will cost (resources). But then are we done with our project description?

I don't think so. Management is always interested in a fifth variable: risk. That is, given the previous four parameters we've identified and the plan that goes with them, management wants to know whether the project represents a high, medium, or low risk to the business. We know from vast experience that projects have different risk profiles, and good managers try to balance their project portfolios by planning a spectrum of projects with different risk levels. The more risky ones have a greater probability of failure, but they might have bigger payoffs, too. Just as it is judicious for individuals to have diversified financial investment portfolios, it is smart for a company to diversify its portfolio by having many projects with different risk/reward profiles. Statistically, such an enterprise is bound to prosper.

Now, how can we use geometry to visualize this new, important, and — I believe — final, parameter?

Enter the pyramid

I propose a model that represents the first four variables as the four sides of the base of a pyramid. We'll assign extensive properties to the sides so that the lengths are meaningful. Note that this is different from the Davis tetrahedron model, in which the attributes occupy the vertices.

For simplicity, let's assume that all sides of the base are of equal length, so that the base forms a square. This is reminiscent of Max's star, except that we have moved the attributes from the corners to the edges. Of course, these lengths can be independently adjusted, so the base is actually an arbitrary quadrilateral. Conceptually, however, we lose nothing by assuming for now that the base is a square.

Now let's redefine the length of each side of the base. We will also adjust our terminology slightly to reflect more accurately what the sides represent. Bear with me, and you will see why.

1. **Scope.** More "things to do" represents a larger scope, so the length of this side increases as the scope increases.
2. **Quality.** Higher quality standards mean a tougher job, so the length of this side increases as our quality metrics increase — in other words, as we "raise the bar" on quality.
3. **Speed.** This is our way of capturing the time element; we increase the length of this side as the speed increases. Conversely, the slower you go — the more time you have — the shorter this side becomes. Completing five function points per month is harder than completing two function points per month; think of this side as work accomplished per unit time.
4. **Frugality.** (Max suggested this term instead of my original *parsimony.*) When we consume fewer resources, we are being more frugal. So higher frugality corresponds to a longer length for this side. If we use up more resources, then this side gets shorter.
Notice that if we use these definitions for scope, quality, speed, and frugality, the project becomes easier if the sides are shorter. That is, the project is easier if we do less, lower our quality standards, proceed more slowly (take more time) and can afford to be less frugal (have more resources.) Thus all four variables "move in the same direction."

Note also that with these definitions we increase our profitability as we increase the area of the base. This is because the value of the product goes up as we make it bigger, better, and get it sooner, while at the same time are the most frugal in producing it. Maximizing value while minimizing cost optimizes for profitability. It's perfectly logical that attempting to make our profit larger also makes the project harder and more risky.

The altitude variable

Now let's build a pyramid on this base, keeping in mind that no matter what lengths the sides are for a given project, the volume of the pyramid will be proportional to the area of the base times the altitude. The altitude abstractly represents the *probability of project success*, which is the inverse of its risk. That is, a high-risk project will have a low probability of success and a low altitude. A low-risk project will have a high probability of success and a correspondingly higher altitude.

Now all we have to do is link it all together.

![Project Pyramid Diagram](image)

**Figure 3: The project pyramid. A high-risk project will have a low probability of success and a low altitude. A low-risk project will have a high probability of success and a higher altitude.**

The pyramid's volume is constant

We can now posit that the volume contained in the pyramid is a constant for a given team. That is, reality dictates that only so much "stuff" will fit into the project pyramid, based on that team's capabilities. This makes sense, because the pyramid's volume is proportional to

\[ \text{volume} \propto \text{difficulty} \times \text{probability of success} \]

As one goes up, the other must go down. This is another way of saying that there is a "conservation law" at work here: the product of the base area — which represents the project difficulty due to the specification of the four parameters — times the altitude (representing the probability of success) is proportional to a "conserved" volume.

What determines the pyramid's volume? Two things. First, the capabilities of the project team, as we have already mentioned. And second, the degree to which the project team members are grappling with unfamiliar problems. A highly capable team implies a larger volume:

more capacity = more "stuff" = more volume
and lots of new problems and unknowns implies a smaller volume:

- more unknowns = higher risk = less volume

So, given a constant volume corresponding to the project team, what do you have to do if you want to make the altitude higher — that is, if you want to increase the probability of success? By the logic of elementary solid geometry, you must make the base smaller. You do this by reducing the lengths of one or more sides of the base, thereby making the project easier.

Remember: Volume is proportional to base times altitude, regardless of the base’s shape.

**A statistical interlude**

At this point, we can attempt to figure out the right "scale" for the altitude. We can measure the edges along the base in familiar units:

- Scope — function points or features
- Quality — inverse of number of defects allowed
- Speed — function points or features/month
- Frugality — "inverse" dollars or person-months

But what about that pesky probability of success, our altitude?

We know that "longer is better," that a higher altitude corresponds to a higher probability of success. But there is a slight problem with using probability — a percentage-based measurement — as the scale. For example, if we have a pyramid with an altitude corresponding to a 60 percent probability of success, we cannot, under the constant volume assumption, improve that percentage by cutting the area of the base in half in order to double the altitude. That would give us an absurd answer of "120 percent probability of success," and we know that probabilities must be between zero and 100 percent.

To resolve this conundrum, we must investigate how the outcomes of software development projects are distributed. Can we assume that these project outcomes are distributed according to the standard normal distribution — the well-known "bell curve"? The diagram in Figure 4 is worth a thousand words.

![Diagram showing the relationship between project outcomes and the standard probability bell curve.](http://www-106.ibm.com/developerworks/rational/library/4291.html)
For those of you who are rusty on what a probability distribution function is, recall that the x-axis represents the outcome, and the y-axis represents the number of events with that outcome, which, properly normalized, is the probability of that outcome. If we start from the left edge and sweep out the area under the curve, we measure the cumulative probability of attaining that outcome. In Figure 4, the percentages below the x-axis show us how much area is contained between the x-axis coordinates that are spanned.

Note that the distribution is "normalized" here, with the midpoint called µ, and the "width" of the distribution characterized by the standard deviation: sigma (the Greek letter in Figure 4 that resembles a small "o" with a tail on its upper right). The distribution extends to both plus and minus infinity, but note that the "tails" of the distribution past the plus and minus 3 sigma limits are quite small; the two tails share less than 0.3 percent of the entire area under the curve. The graph tells us that 68 percent of the projects will be either somewhat successful or somewhat unsuccessful, that only about 27.5 percent (95.5 percent minus 68 percent) will be either very successful or very unsuccessful, and that only 4.2 percent (99.7 percent minus 95.5 percent) will be either extremely successful or extremely unsuccessful. To get the relevant percentages for each of these, we can just divide by two, as there is symmetry around the middle. For example, we can predict that around 34 percent — approximately a third — of all projects will be somewhat successful.

In most applications we assume that µ is zero, so the outcomes range from minus infinity to plus infinity. We can think of the x-axis as the payoff or reward. Although I am sure many software development projects have had zero payoff, it is hard to conceive of a project having a very large negative payoff, red ink notwithstanding. And surely all projects will be cancelled long before management lets them get to minus infinity! So the symmetrical standard normal distribution with tails to infinity in both directions seems to be the wrong model. What we'd prefer is a distribution that we can use with positive outcomes only, or at least with a finite limit on negative outcomes.

**Right idea, wrong distribution**

For this purpose, my dear friend and colleague, Pascal Leroy, suggested the skewed lognormal distribution, which more accurately reflects many phenomena in nature.

Unlike the standard normal distribution, the lognormal distribution is asymmetrical and lacks a left tail that stretches to infinity. It describes phenomena that can have only positive values. Figure 5 shows what it looks like.


**Figure 5: The lognormal distribution for depicting positive outcomes only**
We still use a sigma to represent the standard deviation, but we interpret it differently for the lognormal distribution, as we will explain below. Note that μ is now coincident with 1 sigma. Half the area under the curve is to the left of 1 sigma and half is to the right; if we believe the universe of projects has this distribution, then we want our project to fall to the right of the 1 sigma line, which means its reward will be above the average. This is the equivalent of saying that we are willing to invest μ (or 1 sigma) to do the project; any outcome (payoff, reward) less than that represents a loss (red ink), and anything above that a win.

Unlike the standard normal distribution, the lognormal distribution clumps unsuccessful projects between zero and 1 sigma, and successful projects range from 1 sigma to infinity, with a long, slowly diminishing tail. This tells us that we can have a small number of projects with very large payoffs to the right, but our losses are limited by the zero on the left. This seems to be a better model of reality.

The meaning of sigma is different in this distribution. As you move away from the midpoint, which is labeled here as 1 sigma, you accrue area a little differently. Each confidence interval corresponds to a distance out to \((1/2)^n\) sigma on the left, and out to \(2^n\) sigma on the right. This means that 68% of the area lies between 0.5 sigma and 2 sigma, and 95.5% of the area lies between 0.25 sigma and 4 sigma. This is how the multiplicative nature of the lognormal distribution manifests itself.

Mathematically, the distribution results from phenomena that statistically obey the multiplicative central limit theorem. This theorem demonstrates how the lognormal distribution arises from many small multiplicative random effects. In our case, one could argue that all variance in the outcomes of software development projects is due to many small but multiplicative random effects. By way of contrast, the standard normal distribution results from the additive contribution of many small random effects.

**Implications for real projects**

What are the implications of this distribution for real projects? Because the peak of the curve lies at around 0.6 sigma, we see that the most likely outcome (as measured by the curve's height) is an unsuccessful project! In fact, if the peak were exactly at 0.5 sigma, your probability of success would be only around 16 percent:

\[
50\% - 1/2(68\%) = 50\% - 34\% = 16\%.
\]

Since the peak is not at 0.5 sigma but closer to 0.6 sigma or 0.7 sigma, the probability of success is a little higher — around 20 percent.

Now this is starting to become very interesting, because the Standish CHAOS report, of which I have always been skeptical, documents that around four out of every five software development projects fail. This is a 20 percent success rate. I will have more to say about this report later on. But it is interesting to note that the lognormal distribution predicts the Standish metric as the most likely outcome, which may mean that most development projects have a built-in difficulty factor that causes the lognormal distribution to obtain.

**What does it take to get to a coin flip?**

What project manager wants to start with a less-than-even chance of success? At the very least, we would like to get the chances up to 50/50 for our projects. So, using our pyramid model, what do we have to do to the base to increase the altitude?

Using units of sigma for our pyramid's altitude, we begin with a plan that gives us a starting point at the most probable outcome, at the distribution peak of 0.66 sigma. To get to a 50 percent probability of success, we need to accumulate half the area under the curve, which we know is at the 1 sigma point. So we need to go from 0.66 sigma to 1.0 sigma, which is an increase of 50 percent. That says we have to increase the altitude of the pyramid by a factor of 1.5, which means decreasing the area of the base by 1.5, or multiplying it by two-thirds.

In turn, this implies that we must multiply the lengths of sides of the square base by the square root of two-thirds, which is about 0.82. Therefore, to go from a naive plan with only a 20 percent chance of success to a plan with a 50 percent chance of success, we must simultaneously:

- Reduce scope by about 18 percent
- Reduce quality standards by about 18 percent
- Extend the schedule by about 18 percent (i.e., reduce speed by 18 percent)
- Apply about 18 percent more resources (i.e., reduce frugality by 18 percent)

relative to what we had planned in the original scenario.

We could, of course, change each of these parameters by a somewhat different amount, as long as we reduced the area of the base by a
third.

Let's call this new plan, the one that gets us to a 50/50 footing, "Plan B." We'll refer to the original, most likely, and somewhat naïve plan as "Plan A."

**More confidence**

Can we do better? Suppose we wanted to go out to the 2 sigma point. This would then lead to a probability of success of around 84 percent:

\[
50\% + \frac{1}{2}(68\%) = 50\% + 34\% = 84\%
\]

This would bring up our odds to five-to-one, which any project manager would gladly accept. In fact, this would be standing Standish on its head: five successful projects for every unsuccessful one.

What would it take to get us there?

Well, we can do the math both ways, either starting from our original Plan A or from the 50/50 Plan B. For consistency's sake, let's begin with Plan A. The math is pretty much the same. We now have to go from 0.66 sigma to 2 sigma, increasing our altitude by a factor of 3. That means we must multiply the area of the base by a third, which in turn means that we must multiply each side by the square root of 0.333. And in our previous list of things we'd need to change simultaneously to achieve better results, we'd have to replace 18 percent with 42 percent.

Let's now summarize, using rough numbers so that we don't assign spurious precision to the model. Plan A has a probability of success of only around 20 percent. As we have seen, if we simultaneously reduce the difficulty of all four of the base parameters (scope, quality, speed, and frugality) by about 20 percent, we get Plan B, which has a 50 percent probability of success. To achieve an 85 percent success rate, we'd need to reduce the difficulty on the base parameters by around 40 percent relative to Plan A. Table 1 summarizes these relationships.

**Table 1: Results of using the pyramid model and lognormal distribution**

<table>
<thead>
<tr>
<th>Plan</th>
<th>Description</th>
<th>Location on lognormal curve</th>
<th>Probability of success</th>
<th>Values for base parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Naïve and most likely starting point</td>
<td>0.67 sigma</td>
<td>20%</td>
<td>Per Plan</td>
</tr>
<tr>
<td>B</td>
<td>More realistic</td>
<td>1 sigma</td>
<td>50%</td>
<td>Reduced by 20% relative to A</td>
</tr>
<tr>
<td>C</td>
<td>High efficiency</td>
<td>2 sigma</td>
<td>85%</td>
<td>Reduced by 40% relative to A</td>
</tr>
</tbody>
</table>

Clearly we've gone way out on a thin limb here, but the numbers in Table 1 represent the pyramid model's predictions, based on the lognormal distribution for project outcomes and a constant volume assumption.

**Important caveats**

At this point it is important to step back for a moment and consider the limitations of this model. We have made many implicit assumptions along the way, and now we must make them explicit.

- Let's begin with what we mean by "success." Remember, we said we would define success as an outcome greater than 1 sigma in the lognormal distribution, which would mean that about half of our projects would be successful.

  But the Standish report says that four out of five projects fail. Does this mean they are so constrained and therefore so difficult that this is the result? Perhaps. Many software development projects are doomed the instant the ink dries on the project plan. But I think there is more going on than that.
I have always had a problem with the Standish report, because I think it overstates the case, and in so doing it trivializes the real problem. If we were to take all original project plans and then apply our four base metrics to assess the projects at their conclusion, Standish would probably be right. And the lognormal distribution seems to support this scenario. But do we really have four failures for every success?

Here is what I think really happens: Along the way, as a project progresses, management realizes that the original goals were too aggressive, or the developers were too optimistic, or that they really didn't understand the problem. But now the project has incurred costs, so that scrapping it would seem wasteful and impractical. So instead the project gets redefined, and the goals are reset. This may involve scaling back the feature set, deferring some things to a subsequent release. Sometimes, especially if the team discovers problems near the end of the development lifecycle, they will sacrifice quality and ship the product with too many defects. And even then, they are likely to exceed their schedule and budget. But does this mean that the project is a failure? Not necessarily.

I maintain that lots of these projects fall into the "moderately successful" bucket, and some into the "only somewhat unsuccessful" bucket. So, as in everything else in the world, we revise expectations (usually downward) as we go, so that when we are done we can declare victory. This is important both politically and psychologically. It avoids what the psychologists call "cognitive dissonance." No one likes to fail, and you can always salvage something. So we tend to gently revise history and "spin" actual results — and the usual distribution applies. In reality, the Standish metrics apply only if you use the original project plan as a measuring stick. But no one actually does.

- The parameters scope, quality, speed, and frugality are not all independent of each other. For example, as the project slips and takes more time, it also incurs increased cost because of the increased resources consumed, so both speed and frugality tend to suffer in parallel. You could try to offset one with the other — for example by spending more money to hire more people and go faster. But, as Brooks so clearly pointed out almost thirty years ago, adding people to a software project usually has the effect of slowing it down! If you wanted to make this kind of trade-off, counter-intuitively, you would do better to spend less money per unit time by having fewer people and going more slowly. You might not even lose that much time, because, as Brooks pointed out, smaller teams tend to be more efficient.

- The different parameters do not have perfectly equal impact. Time, or the inverse of speed, appears to play a more critical role than the other three, although this is always open to debate. Typically, managers resist reducing scope and quality, and they are always in a big hurry. From their point of view, the only parameter they can play with is resources. So often they opt to throw money at the problem. This usually fails, because they don't take the time to apply the money intelligently and instead spend it in ineffective ways. This is exactly Brooks' point. In the end, he said, more projects fail for lack of time than for all other reasons combined. He was right then, and I believe he is still right.

- In general, you cannot trade off the four parameters, one against another — at least not in large doses. That is, you cannot make up for a major lacuna in any of them by massively increasing one or more of the others. Projects seem to observe a law of natural balance; if you try to construct a base in which any one side is way out of proportion in relation to the others, you will fail. That is why we opted to assume our base was a square, with all sides (parameters) conceptually on a somewhat equal footing. We acknowledge that you can adjust the sides up and down in the interest of achieving equivalent area but caution against the notion that you can do this indiscriminately. Again, you can't increase one parameter arbitrarily to solve problems in one or more of the other parameters. Max Wideman likes to think of the base as a "rubber sheet." You can pull on one corner and adjust the lengths, but eventually the sheet will tear. Geometrically, of course, one side cannot be longer than the sum of the other three sides, because then the quadrilateral would not "close."

- To some extent, we have ignored the most important factor in any software development project: the talent of the people involved. Over and over again, we have seen that it is not the sheer number of people on a team that matters, but rather their skills, experience, aptitude, and character. Managing team dynamics and matching skills to specific project tasks are topics beyond the scope of this article. However, the pyramid's volume to some degree corresponds to the team's capabilities.

- We should be careful not to specify product quality based solely on the absence of defects. Quality needs to be defined more generally as "fitness for use." A defect-free product that doesn't persuasively address an important problem is by and large irrelevant and cannot be classified as "high quality."

- What about iterative development? Unfortunately, this treatment looks at the project as a "one shot," which goes against everything we believe in with respect to iterative development. But perhaps the unusually high failure rate documented by Standish is caused by a lack of iterative development. That is, by starting with an unrealistic plan and rigidly adhering to it throughout the project, despite new data indicating we should do otherwise, we bring about our own failures.

However, if we are smart enough to use an iterative approach, then we can suggest a workable model. We start out with a pyramid of a certain volume and altitude during Inception, based on our best knowledge of the team and the unknowns at that point. As we move into the next phase, our pyramid can change both its volume and shape. The volume might shift as we augment or diminish the team's capability, or as we learn things that help us mitigate risks. This is a natural consequence of iterative development. In addition, the shape of the pyramid may change, as we adjust one or more sides of the base by reducing scope, adding resources, taking more time, or relaxing the quality standard a bit — or by making changes in the opposite direction.
This should happen at each of the phase boundaries; our goal should be to increase the altitude each and every time. As the project moves through the four phases of iterative development, we should see our pyramid not only increase volume but also grow progressively taller as we reduce risks, by whatever means necessary. If this does not happen through an increase in volume, we must accomplish it by decreasing the base area.

- The issue of whether projects follow the lognormal probability distribution is debatable. I agree with Pascal that it makes more sense than a standard normal distribution. But we have no fundamental underlying mechanism that says the distribution must be lognormal.
- Finally, the conservation law expressed as a constant volume pyramid is just a model. It provides a convenient visualization of the phenomenon, but it is a guess — and the simplest geometric model I could come up with. To determine whether it reflects reality, we’d need to examine empirical data.

Although it is long, this list of caveats does not negate the value of the model; I think its predictions are valid and consistent with my previous experience. Indeed, many midcourse corrections that teams make during a development project to improve their probability of success turn out to be mere band-aids and don’t come close to addressing the real issues. We have demonstrated over and over again that to improve your chances of success substantially, you need to do more than relax a single constraint by 10 percent, and this model underscores that point. Therein lies its greatest value; I believe it represents a fundamental truth.

It’s all about risk

Risk is perhaps the most important parameter to consider in funding and planning a project. That is why the simple model we have defined in this article correlates four traditional project parameters — scope, quality, speed, frugality — and then adds risk as a fifth variable. If you were planning to paddle a canoe down a river, you’d want to know whether the rapids were class three or class five. The latter would be a lot riskier, so you might decide to spend a little more money on your boat. The same is true for software development investments: It is worthwhile to assess the risks before deciding what resources to allocate to a project. But remember, resources are only one leg of the square base; you must consider them in concert with scope, time, and quality. It is the combination of these four parameters, along with the quality of the team, which ultimately determines the risk profile.

The simple pyramid model also shows how much you must trade off to improve your probability of success. Although it is speculative, the model helps us to soberly decide whether we are willing to invest the resources required to raise our probability of success above the minimum threshold acceptable for our business, given the scope, quality, and time constraints that we specify.

Notes

1 The relevant links are [http://www.maxwideman.com/musings/irontriangle.htm](http://www.maxwideman.com/musings/irontriangle.htm) and [http://www.maxwideman.com/musings/triangles.htm](http://www.maxwideman.com/musings/triangles.htm). In this article, we use the term resources to mean all costs, including burdened people costs.

2 Part of this has to do with the finite horizon of a project. If we ship a defective product, the company will suffer huge support costs post deployment. But these costs are rarely charged back to the project. This is rather unfortunate, because it shifts the burden away from the place that originated the problem. True project costs would include post-deployment support.

3 See [http://www2.inf.ethz.ch/personal/gutl/lognormal/bioscience.pdf](http://www2.inf.ethz.ch/personal/gutl/lognormal/bioscience.pdf)

4 Actually, the math is a little more complicated than that. For the standard normal distribution, the mean and the median are identical, because of the symmetry of the distribution. For the lognormal distribution, they are not. So taking the 1 sigma point here is a little off, but the effect is small. We will ignore it in all that follows, as the effect is on the order of a few percent, and our overall model is not that precise anyway.

5 See, for example, [http://www.costxpert.com/resource_center/sdtimes.html](http://www.costxpert.com/resource_center/sdtimes.html)

6 Remembering also the detail we ignored earlier about the mean and median not being identical for the lognormal distribution. Here is where we can bury some of that approximation.

responsibility in marketing, development, and the field sales organization, overseeing initiatives for Apex and Visual Modeler for Microsoft Visual Studio. In 1998 he served as Senior VP of operations. He retired from Rational in 2003. He holds a Ph.D. in physics from the University of Geneva, Switzerland, and an M.B.A. from University of California, Irvine.

What do you think of this document?
☐ Killer! (5)  ☐ Good stuff (4)  ☐ So-so; not bad (3)  ☐ Needs work (2)  ☐ Lame! (1)

Comments?