Java Pays -- Positively

Evan Quinn, Chris Christiansen

Bulletin #16212 - May 1998

Table of Contents - Abstract - Summary - Full Document - Print Format - Order - Research Feedback

Full Document
Highlight ☐

"This bulletin was adapted from a study and White Paper sponsored by Sun Microsystems."

IDC Opinion

What benefits and challenges should early Java adopters expect?

International Data Corporation (IDC) found that projects built with the JavaTM language, targeting the 100% Pure Java standard yielded an average of 25% savings when compared to C++ for development projects requiring multiple platform deployment. Coding phase savings were even more dramatic, averaging nearly 40%. Java technology adopters realized a similar savings benefit on an annual basis in terms of code maintenance. Large-scale projects that did not make use of component architectures, such as JavaBeans, exhibited less favorable payback results than smaller and medium size projects.

IDC reviewed the Java software development projects of early adopting organizations. The clear majority of positive reaction by the subject adopters impressed IDC, especially given that the projects studied represented the initial Java efforts of the companies interviewed. Not only are such initial projects typically fraught with learning-curve mistakes, but all projects studied commenced on the foundation of the immature Java Development Kit (JDK) version 1.0.2. Sun has since released a far more robust version of the Java language and platform via JDK version 1.1. IDC therefore concludes that the Java language and 100% Pure platform deliver on the promised primary value propositions.

Introduction

During the pre-JDK 1.1 days spanning late 1995 through early 1997, several independent software vendors (ISVs) and corporations decided to test their development mettle using the Java language and 100% Pure Java platform. IDC decided to study early Java technology adopters who started their projects before the JDK 1.1 release, but finished their projects after the JDK 1.1 release. Soon IDC will be able to determine the payback of adopters who entirely used JDK versions 1.1 and 1.2, but only JDK 1.1 projects launched the earliest and having the shortest duration have thus far reached the point where a payback analysis could yield tangible results.

IDC expected that the pre-JDK 1.1 adopters would struggle through their initial projects. Given the immaturity of JDK 1.0.2, IDC thought a common consolation to the early adopter might turn out something like "Well, it was good learning experience." Contrary to IDC's expectations, the vast majority of the primeval Java software development projects studied in this paper actually proved successful for their purveyors.

Often we find the most inventive software development organizations, namely independent software vendors (ISVs), willing to wade into the waters of emerging software technologies first. Though the success of the Java technology movement ultimately sits on the shoulders of corporate buyers, the product push that comes from ISVs creates awareness and a technical starting line for the corporate adopters. Therefore, at this early stage, IDC chose to split the payback studies into both corporate and ISV adopters.

IDC chose to use a payback metric for these studies, rather than, for example, return on investment. The early Java technology adopters that IDC studied adapted mainly existing resources, including programmers, hardware, networks, software, production support practices and other personnel, for Java projects. Therefore, the spending profiles of the projects reviewed herein do not fit the term "investment."

Indeed, one of most compelling benefits of Java technology adoption involves taking advantage of existing resources, and making them work better communally. The actual investment required to get a company's feet wet with the Java language is mild, yielding unusually low denominators for a return on investment metric. Relatively speaking, payback analysis -- how quickly (or not) a company recoups its cost of switching from C++ and proprietary deployment targets to the Java language and to the 100% Pure Java platform -- provides a more meaningful metric.

This document begins with an Executive Summary, which provides the high-level results of the study. The document then discusses a series of Challenges discovered during the study, and finishes with a Conclusion. IDC then supplies four detailed cases that document some of the more compelling Java software development projects encountered in the overall study.

IDC will follow up with additional payback and perhaps (if applicable) return on investment studies regarding Java software development projects in the future. With the recent release of several second-generation Java application development tools, and the initial wide-scale appearance of both cross-industry and vertically-oriented packaged Java applications, the next set of payback studies should provide a view into the beginning of mainstream Java technology adoption.

This early adopter perspective provides dependable insights and clues into how early majority adopters will fare.

Executive Summary

The payback for the nine companies involved in Java projects covered by this study varied by the complexity of application. Companies that used the Java language and 100% Pure Java platform to build and execute relatively small footprint applications, and did not require much variation from the basic Java APIs, realized either comfortable or even astounding payback. Companies that used JDK 1.0.2 as the basis for building large and sophisticated applications felt the bite of maturing, but not truly mature, technology. Some companies that did not possess a base of expertise in C++ found the training requirements for Java adoption daunting.

The length of time it took for the Java software development projects studied to break-even in terms of monetary benefit matching investment, also known as payback period, are listed in Table 1.

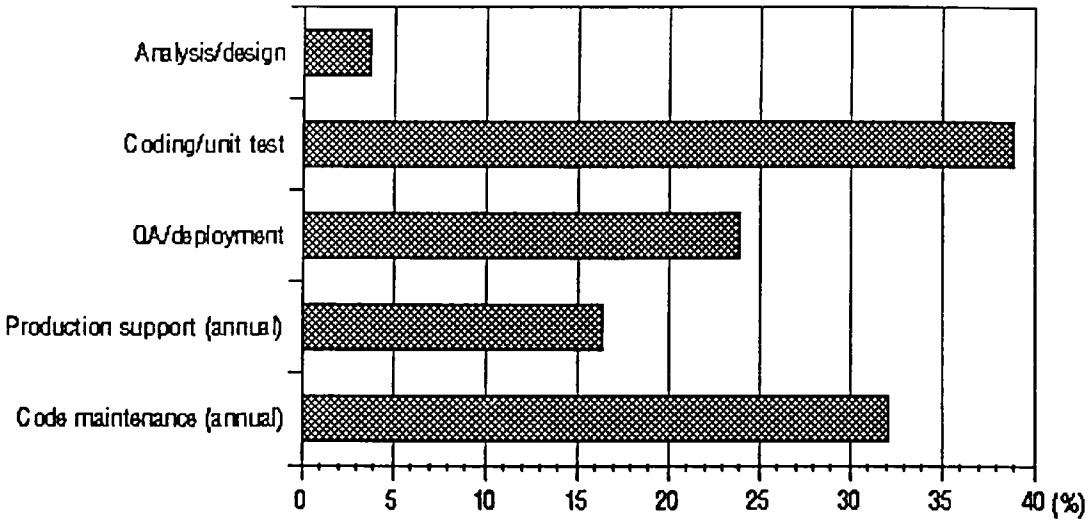
Table 1 - Summary of JavaTM Software Development Projects Length of Time to Reach Breakeven (Payback Period)

Size/Type of Adopter	Payback Period (Months)
Medium-size software vendor	3
Large-size software vendor	3
Medium-size software vendor	36
Small-size software vendor	24
Large-size government agency	10
Large-size photography corporate	7
Medium-size consultant to corporations	12
Large-size commercial bank	12
Large-size international news agency	24
Average of nine projects	14.6

Source: International Data Corporation, 1998

The average payback period was 14.6 months. The average savings associated with Java technology adoption compared to C++ for various phases of the software development life cycle for the nine companies studied, by phase, are exhibited in Figure 1.

Figure 1 - Average Project and Production Savings of Nine Java Projects by Phase (% of Savings in $)



Source: International Data Corporation, 1998

Training Expenses Not Expensive

On the cost side of Java technology adoption, few companies were required to invest heavily in new equipment, software, or expertise. In the majority of cases studied, the existing pool of C++ programmers was sufficient, after a short training period, to develop in the Java language. IDC was impressed that, after a one to two month exposure to writing Java code, C++ developers were as, or even more, effective programming in the Java language compared to C++. Despite the relatively short training phase, the vast majority of incremental expense, or investment, to the adopting companies occurred in lost productivity during the C++-to-Java transition period.

Even though training represented the highest incremental expense of the projects studied, even the training costs were low compared to other forms of technology adoption. On average it appears that an experienced C++ programmer can learn to code the Java language with little more than a development tool, a book, and a mentor, in less than two months -- and at full effectiveness. Java technology adoption does not typically require programmers to upgrade hardware or alter operating systems. Java software

development tools, especially at this early stage of market maturation, are typically well under $1,000 per seat -- or even free when the pure JDK was used.

IDC found, however, that the development features available in the JDK, which currently lacks visual programming tools, are not a viable alternative for long-term, serious, Java application development for most companies. Of course, the programming shops who spurn commercial visual development products sometimes add their own visual programming layers on top of JDK. IDC found only two of nine companies, however, who chose such a rogue option.

Savings During Development

The projects studied were classic software development projects, and the key metric in most cases for success or lack thereof was time and usefulness of the final application. Tangible savings, therefore, such as a reduction in office supplies and electricity, were not a factor -- the environment of project operation did not alter.

Personnel cost savings, however, were substantial, though not consistent through the various phases of the development life cycle. IDC found one project that realized significant savings during the design phase where platform differences required a significant design workaround. In eight of the nine projects studied, however, IDC found no reduction in the systems analysis and design amount of work. IDC found that on average the analysis/design phase accounted for 35% of project time.

Coding in the Java language, however, yielded dramatic savings. In seven of the nine projects, the Java language produced a time savings, and thus personnel expense savings, of around 40-50% due to Java's clean implementation of C-based, object-oriented programming capabilities when compared to C++. IDC found that reuse, an important allure of OO programming, varied significantly by the types and number of projects a company undertakes. The applications with long feature lists fell back in reuse, and even ran into code bloat, à la C++. IDC believes that the lack of a component model, JavaBeans specifically, in JDK 1.0.2 projects, mollified the level of reuse. In fact, several adopters complained about a lack of commercial components.

The two main Java language factors which provided the greatest benefits during the coding phase include built-in memory management and less convoluted OO options. The Java language naturally instills a discipline that yields a tighter OO framework and fewer errors during unit testing. The coding/unit test phase accounted for 40% of project costs.

Returns on the quality assurance (QA) and deployment phase varied. Three companies found that they needed to test on every target platform, and in those cases all or nearly all of the savings garnered during coding were lost during QA. In cases where companies only targeted a few deployment environments, or kept their applications small in terms of size and complexity, QA savings were realized in the 20-60% range, with three companies saving 50% over equivalent C++ QA/deployment expense. QA/deployment accounted for 25% of the project cost on average.

In cases where companies had already adopted a Web-based deployment scheme (whether Internet or intranet), Java technology simply fit in, and yielded no additional savings. In some cases companies used the Java software development project to push themselves into Web-based deployment, and they realized marked savings. In all cases, IDC believes that new technologies will generally not initially provide dramatic improvements during QA. The lack of technical familiarity by the early adopters tends to result in elongated QA experiences. IDC asserts that companies will gain additional benefit during QA in a company's second and third Java software development project.

Production Reflects Development

Most companies either realized or anticipated realizing savings associated with Java's heralded ease-of-deployment when faced with releasing new code versions. Most companies also felt that Java software would be far easier to maintain, producing maintenance savings nearing 30% on average.

Given the typical software product life cycle of at least five years, and given that annual software maintenance costs (assuming a well-designed product) tend to run conservatively at 25% per year of the original product cost, maintenance strengths of the Java language should prove a boon to adopters.

There was some correlation of maintenance advantages to production support -- but not wholly. While most adopters anticipated that the number of Java software bugs over the long haul would be reduced compared to C++, many felt that it might take a few years to realize the reduction. In several cases the number of Java bugs from a version 1 product exceeded what was expected from a C++ implementation. Overall, IDC feels that companies can expect about a 10-20% savings per year in production support over a typical five year life cycle span.

Market Benefit

Market benefit, perhaps the most difficult metric to pin down accurately, also varied widely from project to project. In one ISV case, specifically OpenConnect, the vendor's choice to go with the 100% Pure Java platform provided an absolutely clear market benefit. OpenConnect's market share increased congruently with the time to market and technological competitive advantage associated with Java adoption.

On the corporate side, the United States Postal Service experienced a renaissance of market benefit directly related to its Java technology adoption. The Postal Service's Java software development efforts have led to new customers, improved customer satisfaction, and in some cases, improved internal morale.

Challenges

While a number of the challenges unveiled in the studies are ameliorated by JDK 1.1 based products, the challenges were consistent across virtually all of the interviewed companies. The most typical difficulties with Java technology adoption include:

Performance (that is, speed) and maturity challenges on certain platforms, notably Windows 3.1, the Mac, and non-major Unix versions

Lack of mature coding tools. Until the Java supply-side provides tools which can compile large code bases, there will be few Java enterprise class applications brought to market, and few large-scale, mission critical applications in the corporate section. A JavaBeans-based architecture, however, may significantly decrease the necessity for handling huge builds.

Lack of team programming features

Lack of testing and quality assurance tools

Lack of commercial components and class libraries. Again the promise of the just launched JavaBeans market and the maturation of Sun's Java Foundation Classes (JFC) may reduce such limitations in the future.

Immature server capabilities. Though JDK 1.1 certainly enhances core server services, it will take value-added server framework products and tools from ISVs to make server-side Java software development digestible by most corporations. IDC already sees more than 10 companies with applicable server products in the pipeline or recently released into the market. The recent surge of success stories of server-side Java software attests to the rapid maturation in this area.

Methodology

IDC's Java technology payback analysis project consisted of three major phases, including:

1. Determining viable candidates for the study

2. Interviewing the selected respondents

3. Analyzing and quantifying the results

IDC used a variety of sources to build a list of potential candidates. One source included publicly available information from trade magazines, the business press, etc. Given the recent, high intensity level of Java technology coverage, IDC found many early Java development projects were covered by the trade press. IDC also used candidate lists from previous end-user and ISV surveys, and from Sun Microsystems, Inc.

IDC developed a "screener" questionnaire to determine whether each candidate's Java technology adoption warranted participation in the payback study. IDC also authored an interviewer's guide which IDC used to conduct the final interviews.

IDC typically interviewed the technical project manager and product/marketing project manager. IDC endeavored to identify all of the quantifiable, from a financial perspective, aspects of each organization's Java technology undertaking. The interviewers made sure that the interviewees verified as much of the project financial information as possible, and where possible, provided exact project financial data (such as from a project accounting system).

Most of the interviewees had only limited production experience with the final product(s). Therefore, in most cases IDC requested that the interviewees estimate the financial impact during production phase. Interviewees were given the opportunity to review and correct the results of their particular case study before IDC published the results.

In order to create a framework for quantification, IDC assumed that each project consisted of four phases, Analysis/Design, Coding/Unit Test, Quality Assurance/Deployment, and Production/Maintenance. IDC also realizes that in certain projects there are up-front costs, such as purchasing hardware and training materials, hiring personnel, etc., and accounted for such investments where applicable.

IDC determined that there were several financial categories for investments/expenses and revenues/savings/benefits. IDC decided to use the following summary categories:

Initial expenditures (including but not limited to depreciable expenditures). IDC used a 5-year, straight-line depreciation schedule for hardware, and a 3-year straight-line for software.

Project expenses (by sub-phase, including Analysis/Design, Coding/Unit Test, and Quality Assurance/Deployment)

Production expenses (including code maintenance)

Project savings (loss) (again by sub-phase)

Production savings (loss)

Market benefit (loss)

IDC maintained a ledger of transactions associated with each project studied. The interviewees were given the opportunity to verify and change the ledger after discussion with IDC. Payback was calculated by subtracting (in terms of months) the project start time from the point in time where the amount invested into Java technology adoption was equaled by the savings and/or benefit value associated with the undertaking.

Conclusion

IDC found Java technology offered significantly quick payback benefit to adopters, even to this set of lead users who largely labored through pre-JDK 1.1. Given the breadth of some of the projects, the fact that only one of the nine organizations must wait beyond 24 months for payback offers clear testament to the advantages of both the Java language and 100% Pure Java platform. The somewhat extended paybacks, those beyond 18 months, occurred to two ISVs and one corporate adopter. In all three cases of extended payback, the difficulty seemed to stem from the relatively gigantic project undertaking of these organizations, rather than any glaring weakness of Java technology. IDC is convinced that the payback periods for these largest projects would be far quicker now with JDK 1.1.

The following summarizes the benefits IDC found that the Java technology adopters realized:

Adopters were impressed by the speed which C++ developers were able to gain proficiency in the Java language -- it seldom took more than two months to gain proficiency.

In the majority of cases, the Java language's "clean" C implementation sped the coding process. In addition, the lack of memory manipulation and straightforward OO implementation improved the unit testing process as well. Most adopters felt that reuse and natural enforcement of good OO design technique would yield savings during the code maintenance phase as well.

The greatest inconsistency took place during quality assurance. Depending on the level of maturity of Java Virtual Machines, and the number of platform variations, adopters experienced mixed results. IDC believes that the current JDK 1.1, plus the recent advent of more sophisticated commercial testing capabilities, will improve the QA process.

Only one of the nine companies studied backed off from a commitment to using Java technology going forward. Despite the bumps in the road, the vast majority of adopters studied continue to believe in the primary value propositions of Java technology.

Case 1: OpenConnect

Company Profile

OpenConnect, a Dallas, Texas-based software vendor, specializes in mainframe connectivity products. The privately held OpenConnect has become a leading provider of terminal emulation and related screen and datastream-scraping products. As evidenced by this analysis, OpenConnect's decision to try Java technology for Web-enabled terminal emulation provided a major boost to OpenConnect's competitive position.

Business Value

OpenConnect, like so many other companies, wrestled with how to migrate its products to the Web. They wanted to bring OpenConnect's terminal emulation and datastream-scraping capabilities to the explosion of the Internet. Only one month into OpenConnect's HTML-based project in October of 1995, they discovered that the lack of HTML's programmability seemed too high a barrier to climb. Since OpenConnect currently supports over 50 combinations of client hardware/software, ActiveX offered a far too limited platform reach. The Java platform offered the only legitimate alternative.

The Java technology version of the project commenced in December of 1995, and OpenConnect reached first customer ship for the Java-based WebConnect product by June of 1996. Development on OpenVista, OpenConnect's Java-based GUI builder using data-stream scraping technology, commenced in February of 1996 and reached production in December of 1996.

Payback Results

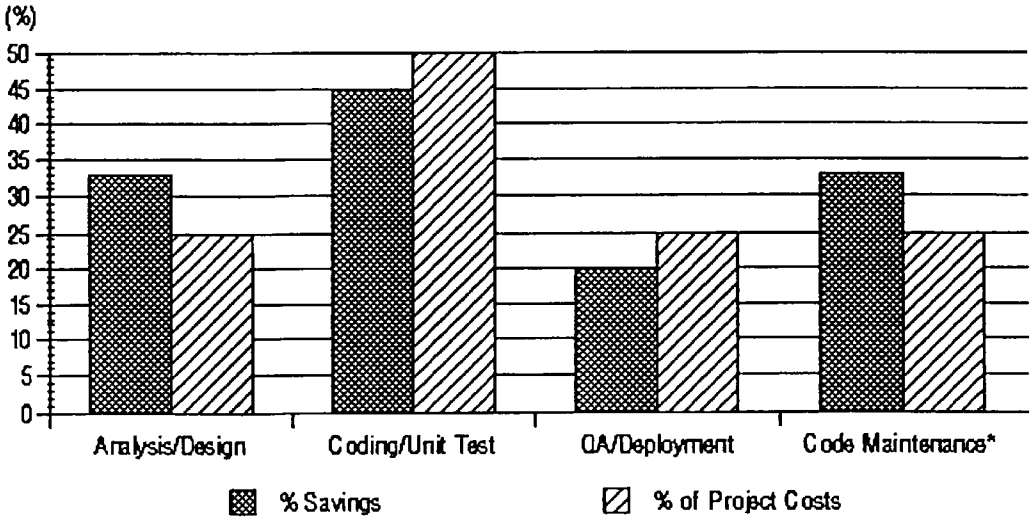
OpenConnect realized a startling payback in three months after release of its initial Java-based product. The key factors contributing to the dramatic payback came from the savings throughout the development life cycle, decreased ongoing production costs, and a key contribution in terms of time to market and Java-based product advantages. The ledger for OpenConnect's Java technology adoption is exhibited in Table 2. Figure 2 displays the savings OpenConnect realized during the development life cycle phases resulting from using Java technology versus C++.

Table 2 - Ledger for OpenConnect's JavaTM Technology Adoption

1.	Personnel costs remained consistent as a sunk cost from previous C++ staffing (10 programmers).
2.	No additional hardware expenditures were required -- all were sunk costs.
3.	Software purchases at $100 per programmer costs $1,000.
4.	Employees were non-productive for 1.5 months on average each due to Java training period.
5.	A consultant provided mentoring during the early phases of the project for six months.
6.	Fifteen Java training books were purchased at $50 each (10 Java in a Nutshell, 5 Sun Java books).
7.	Analysis/design phase took 33% less time than a C++-equivalent project and 25% of the total project cost.
8.	Coding took 45% less time than for an equivalent C++ project and 40% of the total project cost.
9.	Quality assurance took 20% less time than for an equivalent C++ project and 35% of the total project cost (savings on QA were negligible, but savings on deployment were extensive).
10.	Code maintenance will require 33% less expenditure post deployment (at 25% per year of original development costs).
11.	Production support costs will decrease 33% per year compared with an equivalent C++ product.
12.	The resulting Java product provided time to market and recognition benefit to OpenConnect. IDC conservatively estimates the market benefit at $2 million in the first 1.5 years of production, pre-tax.

Source: International Data Corporation, © 1998

Figure 2 - Life Cycle Savings/Costs of JavaTM Technology: OpenConnect's WebConnect



*Code maintenance costs estimated at 25% per year of initial development costs.
Source: International Data Corporation, 1998

Staffing and Training

The WebConnect project required four full-time Java software developers. OpenVista required six. OpenConnect started with C++ developers, and provided them with Java language training books from

Sun and Java in a Nutshell. OpenConnect also hired consultants at the project's outset, more as mentors and experts than for code development.

After only two weeks of informal inculcation into the Java language, all the C++ developers were already productively writing Java code. It took the neophyte Java software developers only one to two months to reach the same level of programming effectiveness as C++. The developers who were responsible for the GUI aspects of the project needed a little extra time to grasp the then rather immature nature of AWT (the Java APIs GUI and windowing classes) and made up the "two-month" contingent. IDC allocated 20 person-months in training costs for the total of the WebConnect and the OpenVista projects, which included the funding for consultant support during the beginning of the WebConnect project.

The Project

Analysis and Design

OpenConnect realized significant savings during the early phases of the life cycle directly related to its choice to program with the Java language. While logical analysis was not directly impacted, the detailed design for OpenConnect's Java software product was treated as a single platform. It would have required eight distinct flavors of detailed design for the target platforms had the project been conducted in C++. The savings in project management time alone, only needing to plan a single line of effort versus eight platform branches, provided clear benefit to OpenConnect. IDC counted a 33% savings during the design phase of the project versus a C++ equivalent. The pre-coding phases of the development life cycle consumed about 25% of the total pre-production project expense budget.

Coding

OpenConnect used a unique set of criteria when choosing which Java software development tools to use for WebConnect and OpenVista programming; they left the decision completely up to the developers. As long as the final code was 100% Pure, OpenConnect's development management reasoned that it shouldn't make any difference which visual development environment each programmer used. As a result, about 80% of the OpenConnect programmers used Symantec's Café or recently released Visual Café. The other 20% used Microsoft's Visual J++. Café, Visual Café and Visual J++ supplanted Microsoft's Visual C++ and Visix's Galaxy GUI builder as the primary development tools. The team used Solaris as the unit testing platform.

The single source to multiplatform characteristic of Java technology design provided significant payback to OpenConnect during the coding phase. Quite simply, the Java development team dealt with a single code base, rather than eight which they would have managed in an equivalent C++ effort. Though certainly a C++ equivalent project would involve a single main source base, with porting to the other seven platforms, the lack of even needing to consider the seven ports yielded dramatic savings.

Since the terminals that WebConnect emulates do not typically present a particularly complex user interface, OpenConnect did not experience the difficulties with the Java APIs GUI classes that we found slowing some other early adopter Java software development projects. OpenConnect possesses vast experience with the types of datastreams that OpenVista needs to use. They found no problem stepping up to the mainly logic-driven and simple I-O demands of the parsing and datastream management. Even the OpenVista GUI building features provided a competitive advantage, since most of OpenConnect's competitors had gone with an HTML solution and therefore could not approach the GUI sophistication offered by a Java product.

Overall IDC found that OpenConnect realized 45% savings related to the choice to go with Java technology versus sticking with C++ during the coding phase. Coding accounted for 40% of the total project expenditures.

Quality Assurance (QA) and Development

Though OpenConnect only used a single base of source code for WebConnect and OpenVista, they still worked their way through an entire quality assurance pass on all of the target platforms. The platforms included key versions of the Netscape, Microsoft and Hot Java browsers on a variety of operating systems, including AIX, HP-UX, IRIX, OS/2, Solaris, Windows 3.x, Windows 95 and Windows NT. OpenConnect also tested the Java-based software products on four types of NCs, including JavaStation.

Often terminal emulation products link end users to mission-critical mainframe systems. Therefore, OpenConnect left no stone unturned to ensure that WebConnect was utterly solid. Essentially, OpenConnect's philosophy on and commitment to QA is language-independent. Therefore the costs of QA for the Java software development projects ran essentially identical to what they would have been in a C++ project.

OpenConnect chose to use a combination of typical methods and Web-based methods for deployment of WebConnect to customers. Today OpenConnect takes advantage of the Web for WebConnect upgrades as well. Overall, OpenConnect realized a 25% savings during deployment compared to a C++ project.

Project Summary

OpenConnect drew advantage from its choice of Java technology over C++ throughout the software development project. With palpable savings in detailed design, coding, and deployment, a quick payback on the project was assured. The relatively limited scope of the project, plus OpenConnect's expertise in the WebConnect feature set, added up to an eminently successful initial Java software development undertaking, and has led to additional projects based on Java technology.

Production

Soon after release of WebConnect, a CASE statement processing bug in a JIT (Just-In-Time) compiler emerged in the user community. Though the bug clearly was not OpenConnect's mistake, detecting and fixing the problem was ungainly and embarrassing. OpenConnect felt that a similar bug occurring in a C++ implementation was highly unlikely given the maturity of C++. So the initial support costs of WebConnect ran somewhat higher than the equivalent costs of a C++ product. Fortunately, this bit of bad news during production was easily offset by great news on the competitive advantage front and even surprisingly on the performance front.

Probably due to the network I-O bound nature of its Java-based software products, OpenConnect experienced little or none of the feared performance-related challenges of Java applications. In fact, OpenConnect found its Java-based/Web-based WebConnect actually outperformed some of the C++ native implementations of OpenConnect's competitors. OpenConnect also believes that ongoing maintenance of the Java software products will result in additional savings, to the tune of 40-50%, during the product life cycle.

Without the hindrance of performance problems, OpenConnect was free to attack its competition with the first Java-based terminal emulation and datastream-scraping products in the market. The decision to go forward with Java technology, combined with bringing in the project about 40% faster than an equivalent C++ project, gave OpenConnect a clear time to market and technical (Java technology versus HTML) advantage. OpenConnect won virtually every bidding situation it entered. Before its Java-based software product came to market, OpenConnect won perhaps one-quarter of the direct bid situations. IDC conservatively estimates that the Java technology adoption returned approximately $2 million in pre-tax profit to OpenConnect that they would not have otherwise seen. IDC calculated the $2 million by multiplying the typical margin of OpenConnect's products by the increased revenue associated with WebConnect's time-to-market advantage for the first year of WebConnect's life.

Summary and Future

OpenConnect's Java technology experience was the most positive of any of the ISVs interviewed in this study. The resulting quick payback underscores the success. But OpenConnect will not stop with

WebConnect. Already they are looking to extend their tool set, and considering adding third-party class libraries from companies like KL Group and RogueWave. They already use a Java software product from RSA for the security aspects of WebConnect.

The development manager and marketing manager were both extremely pleased with the Java software performance, which worked out far better than anticipated. The quick progress of the project also surprised project management. Ironically, OpenConnect was disappointed with the level of reuse. Apparently the tendency of even the Java language to bloat code combined with the quite specific needs of WebConnect and OpenVista prevented OpenConnect from realizing the vaunted reuse levels.

Perhaps most important of all, OpenConnect is pleased with its customers' reaction to WebConnect and OpenVista. The decision to go with Java technology clearly resulted in a win-win for OpenConnect and its customers. At the time of the interview, OpenConnect had 140 corporate customers, at an average of about 2,500 end users per customer, using WebConnect.

Case 2: IBM

Analyst: Evan Quinn

Company Profile

IBM's international foundation classes development unit, historically mainly a C/C++ shop, began working with the Java language in May of 1996. The unit produces two Java software products, the IBM International Foundation Classes and WebRunner. The former product provides internationalization and graphics localization Java software libraries as part of JDK 1.1. WebRunner offers a suite of servlet JavaBeans. Though we will anecdotally mention some of the WebRunner development experience, we will focus on the internationalization libraries project for the payback analysis. IBM previously developed similar libraries using C++, so a basis for comparison, required for this payback analysis, exists.

Business Value

IBM, always looking for cutting edge development technologies, considered using Java technology almost as soon as the first JDK hit the Web. Ultimately, however, IBM's Java technology adoption was driven by customer demand: Several of IBM's existing C++ internationalization library customers requested a Java language version.

Ironically, even though the original idea and execution was to port the C++ international libraries to the Java language, the Java software development experience proved so successful and relatively clean that IBM then ported the Java software version of the libraries back to C++. In essence, the Java code base became the primary code base for IBM, and C++ the port.

Payback Results

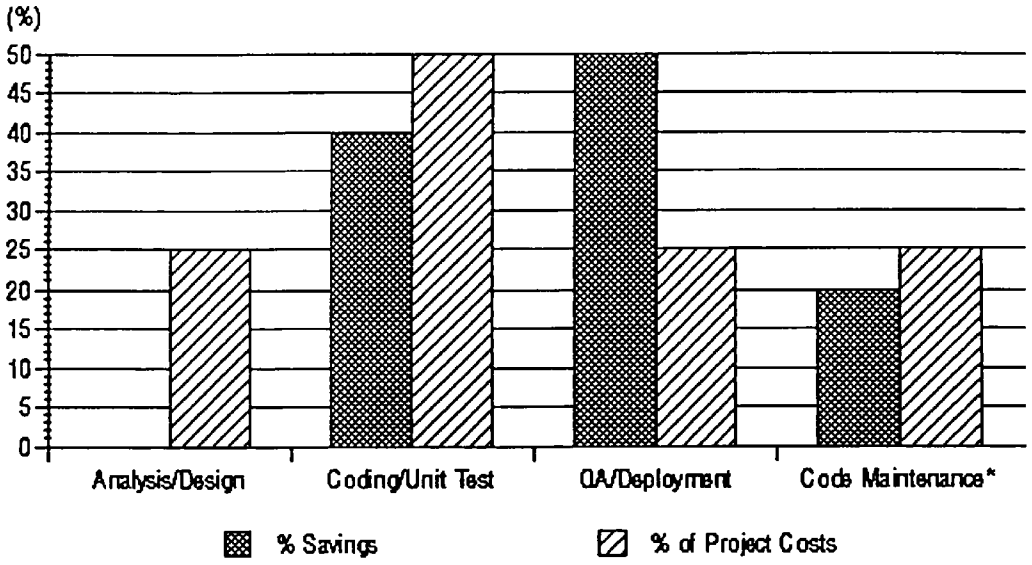
IBM realized a payback only three months after production. The key contributors to the rapid payback came from impressive savings during the coding and quality assurance phases of the project, and ongoing savings related to the reduced cost of coding maintenance and production support. The analysis ledger is exhibited in Table 3. Figure 3 compares the savings and costs of IBM's Java technology adoption experience by development life cycle phase.

Table 3 - Ledger for IBM's JavaTM Technology Adoption

1.	Personnel costs remained consistent as a sunk cost from previous C++ staffing (seven programmers).
2.	No additional hardware expenditures were required -- all were sunk costs.
3.	Software purchases at $100 per programmer cost $700.
4.	Employees were non-productive for three weeks on average due to a Java training period.
5.	No consultants were used for the project.
6.	Java training books were purchased at $50 each per developer.
7.	Analysis/design phase took the same amount of time as a C++-equivalent project and 25% of the total project cost.
8.	Coding took 40% less time than for an equivalent C++ project and 50% of the total project cost.
9.	Quality assurance took 50% less time than for an equivalent C++ project and 25% of the total project cost.
10.	Code maintenance will require 20% less expenditure post deployment (at 25% per year of original development costs).
11.	Production support costs will decrease by 20% per year compared with an equivalent C++ product.
12.	The resulting Java product provided time to market and recognition benefit to IBM. However, the market benefit in this case cannot be directly quantified.

Source: International Data Corporation, 1998

Figure 3 - Life Cycle Savings/Costs of JavaTM Technology: IBM's Internationalization Libraries



*Code maintenance costs estimated at 25% per year of initial development costs.
Source: International Data Corporation, 1998

Staffing and Training

The Java software development team for internationalization libraries consisted of seven C++ programmers and less than one full time employee (FTE) of contract/consulting Java language mentoring help. For training the developers were given a book (Java in a Nutshell), the Java API descriptions from Sun, a few informal training sessions, and a development tool.

IBM chose Symantec's Café for code development, feeling that Café offered the most robust and proven tools option in the market. They also liked Café's interactive debugger. IBM considered a few other products as the products came to market, but stuck with Café. At this point IBM continues to use Café, but is piloting some development with Visual Age for Java, a recently released IBM development tool.

The team previously used products such as Microsoft's Visual C++, IBM's Visual Age C++, IBM's CSet for AIX development, Apple's MPW and IBM's own internal C++ compiler.

Despite the lack of formal training, IBM found most of the developers effectively using Java technology in about two weeks, at about 85% the effectiveness of C++. By the end of the first month, the Java internationalization library developers were as or more effective at coding with Java technology than C++, though apparently a few of the developers still long for C++. IDC estimates that the training cost associated with the switch to the Java language equaled only four additional person-months.

The Project

Analysis and Design

IBM spent approximately 25% of the development project time and cost on the analysis and design phase. IBM did not recognize any improvements or slowdowns during the analysis and design phase related to the decision to use the Java language or 100% Pure platform. The detailed algorithms underlying the internationalization APIs were designed independent of programming language.

Coding

The IBM team was impressed by the Java language during the coding phase. The language's built-in prevention of several typical C++ programming traps, such as array bounds checking, and mismanaged pointers, provided a much safer environment for programmers. IBM really missed the C++ feature of being able to declare an object as a constant. They also missed some of the team programming features found in more mature C++ products. They were concerned about the Java language's inability to control access to class changes mutability. The team's development manager actually wrote a white paper about porting C++ to the Java language based on the IBM experience.

The benefits of writing in the Java language returned about 40% savings during the coding phase of the development project and the coding phase accounted for 50% of the overall project development costs. The elimination of memory management concerns, positive experiences with reuse, and early error detection which held down unit testing time, all contributed to the pool of savings.

IBM also makes extensive use of code reviews -- every person's code and documentation gets looked at by several other developers. The Java language provided an advantage over C++ during code review due to the Java language's more straightforward nature. The code reviews also proved a fertile ongoing training ground for the IBM programmers, improving their overall effectiveness writing in the Java language and ultimately adding to the savings realized by IBM during coding.

Quality Assurance (QA) and Development

IBM limited its testing of the internationalization libraries to the Sun and Symantec Java Virtual Machines (JVMs), and the Netscape and Microsoft browsers, though they did work with Sun's HotJava browser due to its early JDK 1.1 compliance. Due to the tactic of limiting the deployment base to more mature platform targets, and due in part to the algorithm-based nature of the libraries, IBM realized close to 50% savings during the QA phase associated with the project.

Despite the sterling improvement in QA efficiency, IBM felt that the relative lack of GUI testing associated with its product helped push the QA savings up and that a more user interface-centric product would probably have reduced or eliminated the savings. Nonetheless, in this particular project's case, IBM cleared the testing hurdles far more rapidly than they had in previous C++ implementations. The addition of a formal test harness, not available commercially during the peak effort period of IBM's project, might have optimized the QA process even further.

Project Summary

Overall IBM realized 33% savings during the development phases attributable to Java technology

adoption. IBM actually produced a demonstrable set of libraries less than two months after project start, and reached beta within four months -- clearly an improvement over similar C++ based projects. The actual deployment to production was delayed for several months since it was tied directly to JavaSoft's JDK 1.1 release. Testing with 1.1 compliant commercial browsers is still in the works, but the actual libraries were fully available upon JDK 1.1's release.

Production

During the Java software internationalization libraries' short life in production, IBM noticed some decrease in the amount of maintenance work required to fix problems in the Java code base in comparison to the equivalent C++ code base. Since most bugs are algorithm-related, the majority of bugs take the same amount of time to fix in the Java language and C++. IBM must ensure that the algorithm fixes get reflected in parallel in both code bases. Nonetheless, code base bugs in C++ take noticeably longer to track down and diagnose, and appear to occur more frequently.

IDC estimates that IBM realizes about 20% savings associated with Java technology adoption during the maintenance phase of coding. Because of the highly technical nature of the IBM libraries, production support costs likewise decrease in alignment with the reduction in code maintenance costs.

In terms of customer reach, the inclusion of IBM's internationaliza-tion library in JDK 1.1 provided a windfall. With the number of Java software developers worldwide approaching the half million mark, all leaning on JDK in one way or another, and many concerned with internationalization and localization, IBM's efforts have reached far more developers than ever reached by IBM's C++ efforts. IBM also feels that the association with JDK will reflect positively on IBM's position with more directly commercial products. Though IDC could not directly assign positive time to market or market consideration advantages associated with the internationalization libraries, IBM feels that an unquestionable indirect positive market spin resulted from its Java software development effort.

Summary and Future

The key to the quick payback and overall benefit that IBM derived from its Java software development project might have been even more dramatic with a test harness and a debugger API. A performance suite would also help Java technology's cause according to IBM -- they were forced to map C++ performance tests to Java software tests, and were only successful in this vein due to the heavy algorithm focus of the libraries. The lack of a revenue-based market benefit associated with the project also prevented higher returns.

Overall, the IBM team found Java applications performed as they hoped on a variety of fronts. The development manager was pleasantly surprised at the positive levels of customer satisfaction and the speed of development that IBM realized initially during the project. Despite extreme savings realized during the QA phase, the lack of a JDK 1.1 based tool threw curves into bringing application quality up to snuff -- it was tricky coding in pre-1.1 JDK for the product part of the JDK 1.1 release!

The early positive experience with the internationalization libraries provoked IBM to begin the more commercial WebRunner project. WebRunner actually boasts more Java software developers than the internationalization libraries project. The internationalization and WebRunner efforts are early successes of an overall highly strategic commitment that IBM has made to the Java language and the 100% Pure Java platform.

Case 3: United States Postal Service

About the Company and Project

United States Postal Service (USPS) is an independent government agency headquartered in

Washington, D.C. In 1997, it handled 190 billion pieces of mail. On a daily basis, it delivers 603 million pieces of mail to 128 million addresses. The USPS accounts for 43% of the world's mail. To handle this massive task, USPS employs 750,000 career employees in the U.S.

The USPS's goals include, "Making sure your business is more profitable through the USPS." Its mission statement goes even further by stating, "USPS's goal is to evolve into a premier provider of 21st century postal communications by providing postal products and services of such quality that they will be recognized as the best value in America." An ambitious goal to say the least, but each journey starts with one step. USPS's Java applications represent a giant step in helping USPS reduce its customers' costs while increasing their satisfaction.

USPS is in the customer service business to an even greater extent that just delivering the mail. Therefore, USPS created a Java application to automate the creation of postage statements for bulk mailings. According to USPS, "The Java Postage Statement software provides functionality that goes beyond the easy-to-follow, fill-in format. Additional features include arithmetic functions, data carry-forward from page to page, and automatic advancement to specific areas of the form based on mail piece weight and/or rate category." In addition to guiding the customer in filling out the form properly, it skips over irrelevant sections and shortens the time needed to complete the form. This postage form also reduces errors by calculating the correct postal rates. The customer benefits are not the only advantages. The postage form allows the USPS to easily change postal rates without reprinting the forms. Moreover, this Java-based technology is transferable to other postage forms.

Business Value

USPS improved customer satisfaction by using a Java application to simplify the complex process of buying postage for bulk mailing. In December of 1996, USPS finished translating the first of 13 postal forms into a Java-based application. Called Java Postage Statement, this Java application is targeted at the 500,000 smaller businesses engaged in bulk mailing. It is available on the USPS Web site at **http://www.usps.gov /formmgmt/Webforms/**.

Customer Pressures

USPS needed to simplify and automate the complex form for bulk mailings. They also wanted to improve customer satisfaction and thereby improve their business by servicing their customers better. To grow the bulk mailing business, USPS needed to help their customers eliminate mistakes and thereby reduce the tedious process of filling out postage statements. Their goal was to make mailings easier for the customer.

A permit account is a trust account. Used to pay for mailing, these accounts are held by over 500,000 USPS customers. Bulk mailings are withdrawn from these trust accounts. Each mailing must be accompanied by one of 13 commonly used forms (postage statements). Small to medium customers bring bulk mail and postage statements to the post office.

Before the Java Postage Statement, customers spent about 30 minutes filling out each form for each mailing. Now, the same process takes only five minutes. With customers filling out 100 or more forms per year, IDC estimates that the time savings can be over 42 hours per year. This rough calculation does not include the considerable time spent in correcting erroneous forms, wasted trips to the post office, and employee frustration with an annoying process.

After the permit holder brings in a mailing, a postal clerk checks the accuracy. Even customers who used Java Postage Statement must still go through this same process, but Zuckerberg's Customer Service Support Analysts report, "Fewer errors are coming in the door." Once again, the inconvenience to customers is minimized by the Java application. Overall, Zuckerberg says that, "The real customer value is in the application's ability to do rate calculations based on weight and follow that calculation throughout the postage statement."

Business Pressures

USPS's competition would seem an unlikely factor in motivating this government agency. Despite the fact that USPS delivers more mail in one week than Federal Express and United Parcel Service combined in one year, USPS is not complacent. Direct competition for bulk mailings is largely non-existent, but USPS does compete with other outlets for advertising dollars. These include newspaper inserts and local advertising circulars. Beyond print advertising, USPS also sees competition from radio and television.

Benefits

USPS is extremely pleased with the short amount of time (four months) that the applications development team took for the first application. They like the fact that a single application runs on a wide variety of customers' PC operating systems. Most importantly, USPS's customers are pleased with the application's ability to automate a tedious process and reduce errors.

A major USPS goal was "Platform neutrality because we are trying to reach a potential customer base of 500,000 and we didn't know what kind of PCs [our customers] had," according to Zuckerberg. Therefore, heterogeneous support was an absolute requirement.

According to Zuckerberg, "The browser doesn't make any difference because our application is not a Java applet." This is a significant advantage because USPS customers run a wide range of browsers. By developing a full-fledged Java application (instead of an applet) USPS ensured browser independence, and the ability to run on any customer's desktop configuration. "This way we know the code is correct," according to Zuckerberg. "The other Java benefit is that USPS only has to QA once for all platforms," says Zuckerberg. "On old platforms, we had to develop and test four different versions of the code," explains Zuckerberg.

Implementation Details

Developed with Sun's JDK 1.1, the application is 100% Pure Java technology, but the installation shield and print sub-system are platform-sensitive, as well the interpreters. The application excluding the installation shield is 1.4 MB. Each form is 70K. The utilities file is 1.7MB. The application is strictly client-based and runs on a local PC after downloading.

There is no database, but the rate tables are hard coded into each form. Zuckerberg recognizes the weakness of this arrangement and comments, "In the future, USPS hopes to have a second tier of rates only so we can easily make rate changes."

The Web server is an existing Sun UltraSparc running Solaris 2.5 that had excess capacity, so no new hardware was purchased. As for hardware and software costs, Zuckerberg simply said that the hardware was already bought and paid for, so there were no expenses in this area. He also noted that software fell into the same category because it was all available as shareware.

Staffing was a sunk cost, but the project costs came in far lower when compared to non-Java technology development efforts. Zuckerberg led this project team. The project included three full-time programmers from a systems integrator.

On staffing, Zuckerberg comments, "If I had to do this application without Java technology, it would have taken at least twice as many people from the systems integrator." He adds, "In the old days, it would have taken 100 to 150 hours for this application. This figure would have been multiplied by three platforms." Allowing for some reuse of code, Zuckerberg believes that Java language reduced applications development costs by a factor of three.

Java technology furthered savings by reducing documentation costs. Because only one documentation set is needed for all platforms, an additional cost reduction of three or four times is possible. When all of

the costs are added together, applications development costs may be only 10 to 20% of other applications development environments.

As for maintenance, Zuckerberg estimates that this cost is also 30% to 50% less expensive. Because approximately 60-70% of software budgets are spent on applications maintenance, IDC believes that ongoing savings will result in a considerable benefit over time. As Java applications become a greater percentage of corporate software, IDC expects that Java applications' reduced need for program maintenance in heterogeneous environments will shift budgets from maintenance to new applications development.

Other benefits are more difficult to quantify, but still important. USPS has experienced no downtime due to the application's failure. With customers expecting 24-hour access, uptime is another measurable contributor to customer satisfaction. Moreover, the application is self-supporting and there are no training costs because USPS's Java application works with customers' existing PC operating systems, applications, and browsers. Finally, USPS incurs no media or software distribution costs because customers simply download the application from the USPS Web site.

Customers are very happy with the application. During the first eight months of production, more than 15,000 customers downloaded the applications. The Java Postage Statement has received more than 50,000 hits in its first 8 months. Zuckerberg stresses, "The key saving is for our customers. When they make calculation errors, it may result in the customer returning to the office for additional funds." According to customer feedback, time spent preparing the postage statement may be reduced by 80%.

Drawbacks

The Java Postage Statement is a relatively simple client application. As a result, it suffered from few problems. However, Zuckerberg is frustrated with the issue of interpreters. While Unix, Windows 95, and NT interpreters were no problem, finding Macintosh support was difficult. More importantly, Zuckerberg says, "Windows 3.x is a real problem, but USPS has found a solution for these customers."

Summary and Future

As for the future, USPS is making several changes that include:

Building a new GUI (graphical user interface).

Bundling all 13 postage statements and interpreters into one application that can be accessed with a single mouse click. Right now, there are 13 downloads plus the interpreter.

Building a Wizard for early 1998 that will help customers select and fill in the appropriate postage statement. This wizard will be coded in Java software.

After this task is finished, Zuckerberg says, "Then all I have to worry about is the interpreter, print sub-system, and Install Shield, but we only code the application once."

Further in the future, USPS plans to use Java technology for more than postage statements. Turning its attention to other business, the USPS is working on an application called "Direct Link" that will enable electronic transfers for USPS's larger customers. Who would have ever imagined USPS as a software developer of freeware?

The USPS is extremely happy with its customers' reactions to its Java applications. This warm reception for Web-based applications is not only allowing the USPS to increase the profitability of its customers, but it keeps the USPS competitive. Long term, the reusability of Java code and experience will translate into more applications without corresponding increases in application development costs.

Overall, USPS said they would gladly do it all over again. Zuckerberg was pleased with the initial

project and excited by the prospect of using Java software to move the USPS into the future while benefiting its customers.

Case 4: Kodak

About the Company and Project

Eastman Kodak Company (Kodak), is a $15 billion Rochester, NY progenitor and giant of the photography industry. Though Kodak already possessed an extensive Web site, the company wanted to offer online shopping to take advantage of the burgeoning commerce opportunity on the Web. Kodak wanted a highly scalable and flexible shopping site; one that could start out as a basic storefront but evolve to full personalization -- giving each site visitor a unique Kodak experience.

When Kodak examined the technology choices for its online shopping site, it chose the Java technology-based Web application server and E-Commerce suite called Dynamo from Art Technology Group (ATG) as the backbone for the site. Then Kodak performed custom development in the Java language for certain portions of the site's processing, such as back office interfaces, resulting in the online shopping site located at **http://www.kodak.com/go/shop/**.

Business Value

Kodak felt that its shopping site could potentially yield startling results. Like so many initial ventures into E-Commerce, the company found it initially difficult to predict usage levels. Kodak wanted to ensure scalability and high reliability. Kodak wanted the project to deliver the shopping site quickly, but the site also had to be able to evolve into more advanced E-Commerce functions, such as personalization and user-specific ad serving.

ATG's Dynamo, a high-end, Java-based Web server, plus Dynamo's cousin, E-Commerce applications, seemed like the only combination on the market that could step up to Kodak's tactical and strategic needs. The fact that Dynamo was multiplatform due to its Java technology underpinnings helped convince the Kodak decision-makers. Kodak already sensed that the industry was moving towards Java technology. Dynamo's perfect fit with Kodak's needs motivated Kodak to actually adopt Java-based software.

Payback Results

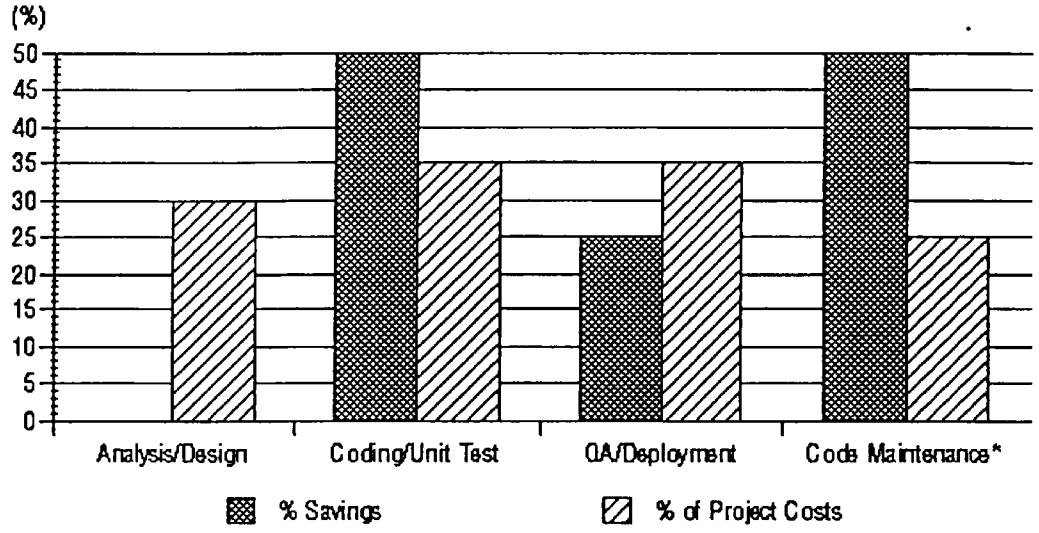
Kodak will realize payback on its Java-based investments after seven months of production. The key contributors to the rapid payback came from dramatic savings during the coding and deployment phases of the project. Kodak will also garner savings on an ongoing basis due to efficiency in code maintenance based on the Java language. The payback analysis ledger is exhibited in Table 4. Figure 4 displays the savings of phases of the development life cycle resulting from Kodak's use of Java technology versus C++.

Table 4 - Ledger for Kodak's JavaTM Adoption

1.	Personnel costs remained consistent as a sunk cost from previous staffing (five project head count).
2.	No additional hardware expenditures were required -- hardware costs would have been equivalent no matter the underlying software technology.
3.	Software development tool purchases cost $1,800.
4.	Three employees were non-productive for one month on average due to a Java training period.
5.	No consultants were used for this project.
6.	Two Java training books cost $100.
7.	Analysis/design phase took the same amount of time as an equivalent C++ project and 30% of the total project cost.
8.	Coding took 50% less time than for an equivalent C++ project and 35% of the total project cost.
9.	Quality assurance took 25% less time than for an equivalent C++ project and 35% of the total project cost (savings on QA were negligible, but savings on deployment were extensive).
10.	Code maintenance will require 50% less expenditure (at 25% per year of original development costs); there are no traceable production support savings at this time.

Source: International Data Corporation, 1998

Figure 4 - Life Cycle Savings/Costs of JavaTM Technology: Kodak's Shopping Site



*Code maintenance costs estimated at 25% per year of initial development costs.
Source: International Data Corporation, 1998

Staffing and Training

Kodak's development team consisted of two programmers, an analyst, a tester, and a project manager. No contracting or consulting help was used for the custom Java programming portion of the project, though ATG helped by training and working with Kodak to install Dynamo. The lead programmer possessed extensive object-oriented programming experience and acted as a mentor.

Kodak chose Symantec's Visual Café and Sun's Java WorkShop for coding. Kodak began with Café, which was the market leading and most robust tool on the market at the time, but switched to Visual Café when Symantec switched emphasis from Café to Visual Café, and empowered Visual Café with improved database access capabilities. Kodak added Sun's Java WorkShop due to the direct association to Solaris that the Sun programming tool offered. Kodak chose a Sparc/Solaris configuration for the servers. Kodak found the developers effectively using Java on average in about one month.

The Project

Analysis and Design

Kodak spent approximately 30% of the development project time and cost on the analysis and design phase. Kodak did not recognize any change to the costs of analysis and design phase related to the decision to use Java technology. The analysis was primarily business, and not technology, driven. Kodak would have reached the same analysis/design results in the same period of time with the same resources regardless of the software technology employed.

Coding

The Kodak programming team sped through the coding phase, taking advantage of the Java language's built-in advantages over C++. High levels of reuse were one of the keys to coding efficiency, and reuse could lead to even more efficient coding in future projects. Built-in memory management also helped increase the speed of coding, particularly during unit test. The benefits of programming in the Java language returned about 50% savings during the coding phase of the development project when compared to an equivalent C++ undertaking. The coding phase accounted for 35% of the overall project development costs.

Quality Assurance (QA) and Development

Kodak tested on Versions 3 and 4 of Microsoft's and Netscape's browsers, as well as the AOL browser. The server-side nature of Dynamo limited the amount of client testing required. Nonetheless, it's difficult to determine whether an equivalent C++ server, with the same browser testing configuration, would have led to any savings. Therefore IDC does not attribute any savings during QA to Kodak's use of Java technology. QA used about 17.5% of the resources of the overall project.

However, the server-side Java technology design of Kodak's site led to obvious savings in deployment. The wholly-contained Dynamo E-Commerce solution, plus customization by Kodak, made deployment a breeze. Kodak gained a 50% savings during deployment due to their Java-based architectural decisions when compared to an equivalent C++ deployment. Deployment absorbed 17.5% of the total project costs.

Project Summary

Overall Kodak realized about 26% savings during the development life cycle when compared to an equivalent C++ project. The total project, from installation of Dynamo to the custom Java programming, lasted only 3 months. The Sparc/Solaris hardware/operating system com-bination, combined with ATG's Dynamo, plus Kodak project management and custom coding, resulted in a Web site that is currently handling about 1.2 million hits per day. The six server site hosts an average of 25,000 users per day, with about 230,000 page hits. Kodak spreads the E-Commerce activity over 3 physical servers, one for the Web, one for Dynamo, and one for a 3.5 gigabyte Oracle database.

Production

An initial JDBC bug caused some production problems soon after deployment. However, once fixed, the Kodak shopping site operated without failure for the six months leading up to the case study -- in the face of rapidly climbing, and perhaps unexpected, levels of activity!

Since this is the first Kodak E-Commerce site, IDC cannot estimate whether Java technology produced an effect on production costs. However, IDC estimates that Kodak will realize about 50% savings per year in terms of code maintenance associated with programming in the Java language versus C++. Kodak began working on additional projects using Java technology soon after completion of the shopping site, including an application for image processing. IDC believes Kodak will discover reuse benefits in the succeeding projects based on Java technology.

Summary and Future

Kodak expresses no doubt that, given the chance to do it all over again, they would definitely choose ATG's Dynamo and Java technology. Kodak's development manager for the project, Alan Dray, rated the overall project as "better than expected." Alan cited speed of deployment and customer/user satisfaction as outperforming the expectations of the Kodak team in regards to Kodak's first E-Commerce foray. Kodak was a little disappointed in the Java programming tools at the beginning of the project, but Kodak upgraded to tools based on JDK 1.1, and found the development tools much improved.

IDC believes that the virtual preprocessing of Dynamo, in terms of development and runtime, provided Kodak with an advantage over Java technology adopters who went at projects completely from scratch. Kodak's willingness to launch even more complex Java application development projects, and to extend the scope of its shopping site into the sophisticated aspects of relationship commerce (that is, catalog, personalization, and advertising management) clearly attests to the success of the project studied herein. Kodak looks to be well on its way to enjoying the benefits of Java technology for several years to come.

Table of Contents - **Abstract** - Summary- **Full Document** - **Print Format** - Order - **Research Feedback**
