

PART

I

Introduction to Software Factories

This is a continuation of what started well before the Industrial Revolution. Genghis Khan was one of the first to grasp the necessity of quick transmittal of information. Now we can do everything instantly, worldwide. It's frightening.

John C. Dvorak

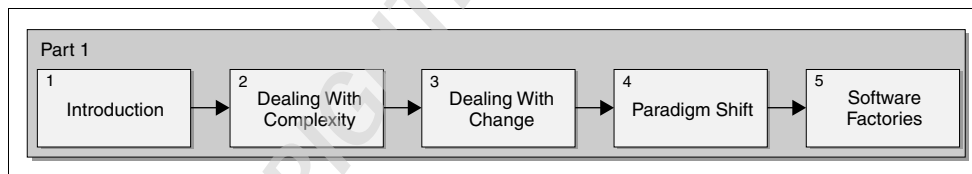


Figure P1.1 A road map to Part I

The road map shown in Figure P1.1 describes Part I. Part I introduces Software Factories. Chapter 1 presents an example used throughout the book and looks at challenges facing software developers. Chapter 2 describes the problem of complexity and looks at current methods and practices for dealing with it. Chapter 3 describes the problem of change and looks at current methods and practices for dealing with it. Chapter 4 describes chronic problems that object orientation has not been able to overcome, and introduces critical innovations that form the basis of Software Factories. Chapter 5 describes Software Factories in detail and presents a vision of industrialization.

CHAPTER

1

Introduction

If this industry doesn't deliver, it will lose the hearts and minds of business leaders, and they will look elsewhere to make their gains.

Michael Fleisher

According to the Standish Group, businesses in the United States spend about \$250 billion annually on software development, with the cost of the average project ranging from \$430,000 to \$2,300,000, depending on company size. Only 16% of these projects are completed on schedule and within budget. Another 31% are canceled, primarily because of quality problems, creating losses of about \$81 billion annually. Another 53% cost more than planned, exceeding their budgets by an average of 189%, creating losses of about \$59 billion annually. Projects that reach completion deliver an average of only 42% of the originally planned features.

In any other industry, results like these would generate a quick response aimed at restoring the bottom line. For at least the last 10 years, however, stakeholders have looked the other way, letting corporate earnings generated by the strong economy absorb the costs of these failures. In these less certain economic times, they are now calling the industry to account for underperforming technology.

Are these problems of interest only when the economy is weak? Will they recede into the background again, as the

Most software projects fail to deliver what they promised on time and within budget.

Stakeholders are focusing on underperforming technology.

Progress appears to have slowed significantly.

4 Chapter 1

market rebounds, or are there more pervasive and fundamental forces at work? For quite some time, they were cast against a backdrop of rapid innovation. No one seemed to mind if we struck out twice for every three times at bat, as long as we hit it out of the park the third time. Recently, however, progress seems to have slowed significantly. Is the engine of innovation running dry, or is something holding it back?

Tools Lag Platforms

Development methods and tools have not kept pace with rapid advances in platform technology...

We think the problem is that software development methods and tools have not kept pace with the rapid advances in platform technology of the last few years. Using the latest wave of Web service platform technology, for example, we can now integrate heterogeneous systems across multiple businesses located anywhere on the planet, but we still hand-stitch every application deployed on this technology, as if it were the first of its kind. We automate large abstract concepts like insurance claims and security trades using small concrete concepts like loops, strings, and integers. If the hardware industry took the same approach, they would build the massively complex processors that power these applications by hand soldering transistors. Instead, they assemble Application Specific Integrated Circuits (ASICs) using software and generate the implementations. Why can't we do something similar?

...and the ensuing raised expectations of stakeholders.

The problem is that stakeholder expectations tend to grow as platform technology advances. We have met increasing expectations for some time by finding ways to hone the skills of developers. Or, to put it differently, we have met the expectations of an increasingly computerized society in much the same way that artisans in the first few years of the industrial revolution met the expectations of an increasingly industrialized society by honing the skills of craftsmen. Up to a point, craftsmanship works well enough. Beyond that point, however, the means of production are quickly overwhelmed, since the capacity of an industry based on craftsmanship is limited by its methods and tools, and by the size of the skilled labor pool. We think the software industry will soon face massive demand on a scale beyond anything we have seen before, driven by a blistering pace of platform technology innovation, not only in the business application market, but also in the markets for mobile

Introduction 5

devices, embedded systems, digital media, smart appliances, and biomedical informatics, among others.

Currently, the most popular way to hone the skills of developers is to harvest and disseminate the practices of the most productive developers, as exemplified by the agile development movement [Coc01]. The value of this approach has been amply demonstrated. We question the assumption, however, that we can solve current problems, much less prepare to meet the demands of the next decade, by honing developer skills. Apprenticeship is a proven model for industries based on craftsmanship, but it does not scale up well. Observers have noted that the most productive developers produce as much as a thousand times the output of the least productive ones, while the least productive developers generally outnumber the most productive ones by a similar ratio [Boe81]. This means that there will never be more than a few extreme programmers, no matter how much we hone the skills of developers. Moreover, the practices of the most productive developers cannot be applied on an industrial scale because they are optimized for people with uncommon talents. We need a better way to leverage the talents of the best developers, one that can scale up beyond what that they can collectively deliver today.

Other industries solved problems like the ones we are now facing by learning how to rapidly customize and assemble standard components to produce similar but distinct products, by standardizing, integrating, and automating their production processes, by developing extensible tools and configuring them to automate repetitive tasks, and by managing customer and supplier relationships to reduce cost and risk. From there, they developed product lines to automate the production of product variants, and formed supply chains to distribute cost and risk across networks of specialized and interdependent suppliers, enabling the production of a wide variety of products to satisfy a broad range of customer demands. This is what we call *industrialization*. It is also the vision espoused by the pioneers of object orientation.

We are not suggesting, simplistically, that software development can be reduced to a highly mechanical process tended by unskilled workers. On the contrary, the key to meeting demand on an industrial scale is to stop wasting the talents of skilled developers on rote and menial tasks, so that they can spend more time thinking and less time doing housekeeping chores. Also,

Apprenticeship is a proven model for industries based on craftsmanship, but it does not scale up well.

Other industries embraced industrialization to solve problems like the ones we now face.

We propose to replace apprenticeship with automation that exploits best practices.

6 Chapter 1

we must make better use of these few but valuable resources than expending them on the construction of end products that will require maintenance or even replacement when the next major platform release appears, or when changing market conditions make business requirements change, which ever comes first. We need a way to leverage their skills and talents by asking them to encapsulate their knowledge as reusable assets that others can apply. Does this sound far fetched? Patterns have already demonstrated limited knowledge reuse in software design [GHJV94]. The next steps are to move from the manual application of patterns expressed as documentation to the codification of patterns in frameworks, then to the assisted application of patterns by developers using tools, then to the encapsulation of patterns by languages, and then finally to the fully automatic application of patterns by language compilers. By encapsulating knowledge in languages, patterns, frameworks, and tools, and by reusing these assets systematically, we can automate more of the software life cycle.

*Reuse has been
an elusive goal.*

Software components will not be as easy to snap together as ASICs soon, since ASICs are the highly evolved products of two decades of innovation and standardization in component packaging and interface technology. Software components are moving in the same direction, however. The Web service technology that we mentioned earlier represents significant progress in packaging and interface technology for software components. There is much more to be done, of course, as we will demonstrate in Chapter 13. Building reusable software components is challenging, and assembling them generally requires adaptation or changes in the components themselves.

*There are
innovations to
exploit.*

We think there is room for significant innovation in these areas, however, such as using models and patterns to automate many aspects of component assembly, and using aspect-oriented methods [KLM97] to weave contextual features into functional ones, instead of rewriting the functional features repeatedly by hand for every new context that we encounter. We think the software industry can climb the innovation curve faster than the hardware industry did by using specialized languages, patterns, frameworks, and tools in software product lines. Of course, the hardware industry has the added burden of engineering the physical materials used for component implementation, while we have the luxury of building with bits. At the same time, the ephemeral nature of bits creates challenges

like those faced by the music industry, such as the protection of digital property rights.

The Software Development Landscape

In this chapter, we want to set the scene for the discussion of critical innovations in software development methods and practices in later chapters. To justify our call for a new approach to software development, we must demonstrate our understanding of the challenges that software developers face today. Their world, indeed our world, consists of rapidly evolving platform technologies, increasing customer expectations, a seemingly endless need to acquire new skills quickly, and intense pressure to extract value from older applications while realizing strong returns on investments in new technology. At the same time, few developers are fortunate enough to build green field applications. Most developers must also deal with challenges presented by existing application portfolios and platform technologies. To keep the story simple and easy to follow, we tackle these two sets of challenges separately.

First, we explore the changes in platform technology and stakeholder expectations that created new kinds of applications and produced the portfolios that businesses hold today. To support this study, we introduce a fictitious company, and show how it might have grown over the last 30 years, describing the evolution of its application portfolio and platform technologies. We will follow this fictitious company throughout the book as it adopts the Software Factory approach.

Second, we analyze new platform technologies that are now emerging, identifying the new challenges they create and the existing ones they exacerbate. Here, again, we use our fictitious company. We walk through a skeletal set of requirements for an application that must be developed to support its relationships with customers and suppliers. We then describe the architecture of this application in just enough detail to expose some of the challenges that the company encounters using the emerging platform technologies. At the end of this book, we will rebuild this application using the Software Factory approach to show how it compares with current methods and practices.

We explore changes in platform technology and stakeholder expectations.

We analyze new platform technologies and the challenges they create or exacerbate.

8 Chapter 1

Platform Technology Evolution

We introduce our fictitious companies.

Our fictitious company manufactures tools and equipment used in the construction industry. Its name is Construction Tools, Inc., or CTI. We will also need a fictitious software company to illustrate some of the ideas described in the book. It will be hired by CTI from time to time to help design and implement business applications. Its name is Greenfield & Short Software, Inc., or GSS. Neither the companies nor the applications intentionally resemble any actual companies or applications.

We will use the timeline shown in Figure 1.1 to help us keep our bearings as we tell this story.

Platform technology evolution has pushed the scope of automation outward.

As we shall see, earlier generations of platform technology supported applications that served users inside the business and the transaction volumes they generated, while the current generation, by contrast, supports distributed applications that serve large numbers of users outside the business, such as customers and partners, and the transaction volumes they generate. Each wave of platform technology pushed the scope of automation outward, from inward facing processes at the heart of the business, such as accounting and materials management, to outward facing ones on its periphery, such as supplier management and warranty maintenance.

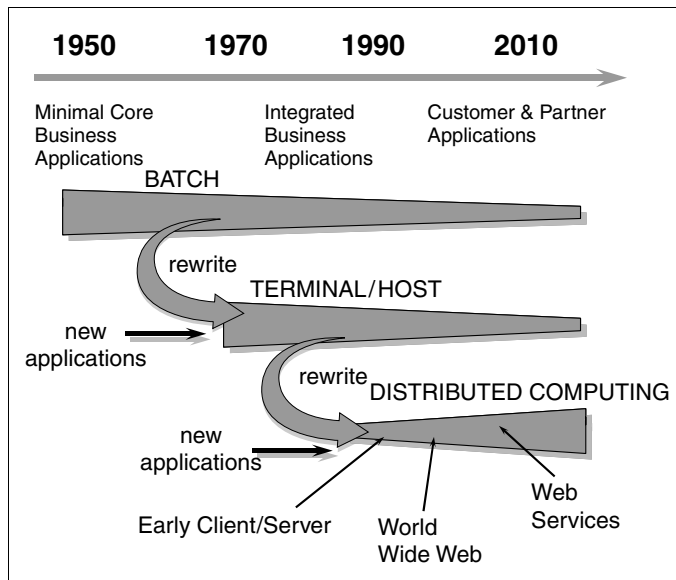


Figure 1.1 Application types and technology trends

The Batch Era

In 1968, CTI employed about 500 people, and had annual revenues of \$250m. It was a typical medium-sized manufacturing business. Structured along conventional lines for the period, the organization chart for CTI is shown in Figure 1.2. The company employed a full-time data processing staff of 30 people, including 10 in operations who managed equipment and job scheduling for their small IBM® mainframe running OS/360. Notice that Data Processing reported to Accounting. This was common in the Batch Era. Expenses for computer equipment, development, and operations did not fall into well-defined categories, and were often buried as accounting expenses.

Batch applications of the 1960s and early 1970s were used by only a few people within the business. Clerks captured data on paper forms that were sent to data entry staff who keyed the data onto machine-readable media, such as punched cards or tape. Most batch applications automated existing manual business procedures. The goal was to gain productivity by exploiting the calculating power of the computer to reduce processing time.

The Materials Management and Shop Floor Operations applications at CTI were typical Batch Era applications. As

CTI's organization structure.

Batch applications computerized existing practices.

Few people actually used computers.

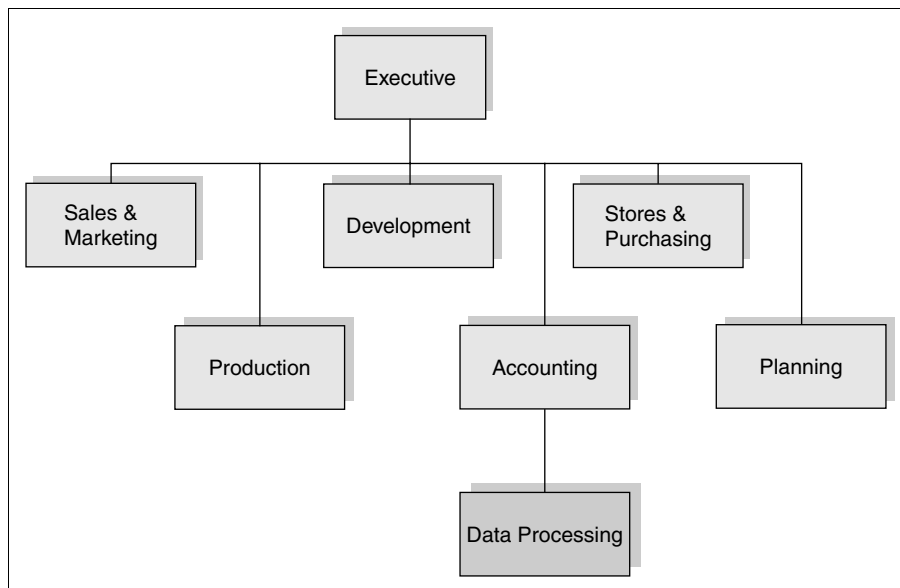


Figure 1.2 CTI's organization structure circa 1968

10 Chapter 1

machine operators milled parts for Hand Drills and Jack Hammers, they reported time and material usage to a clerk who compiled the information and sent daily summaries to the data entry staff. The data was fed into a daily run of each application. At the start of each day, the Shop Foreman received a report from which he could determine the status of each order and allocate work to machine operators. The Stores Manager also received a daily report identifying the materials to be trucked from the warehouse to the various shop floors around the factory.

Dependencies between applications made maintenance a nightmare.

Because they automated existing manual business procedures, most applications were associated with a single department. The resulting application portfolios looked like Figure 1.3. They contained multiple islands of data, each used by a small number of applications, or sometimes by just a single application. Data was often duplicated across applications and stored in a different file structure by each application, making it difficult to retrieve and process. In the early part of the era, data was stored mostly on tape, making it difficult to keep the islands consistent. For example, in addition to entering daily data

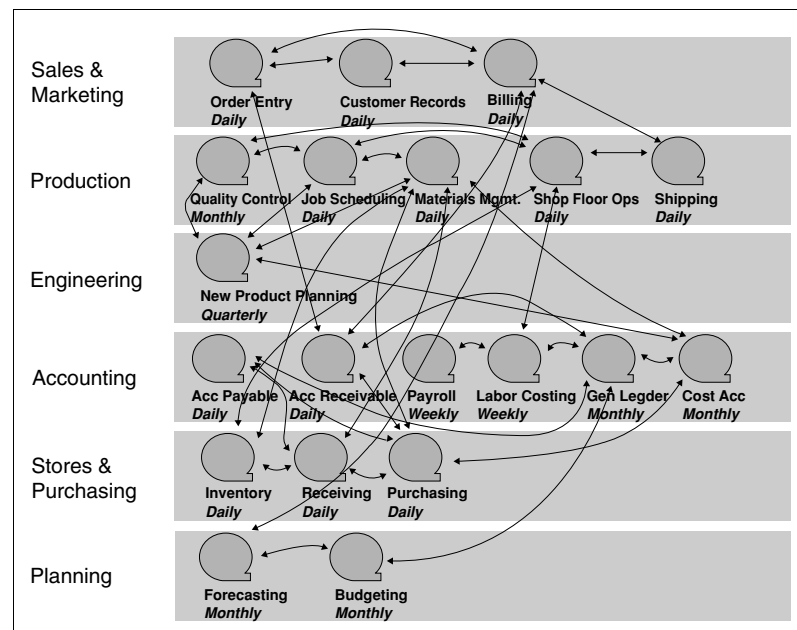


Figure 1.3 CTI's application portfolio in the batch era

for Materials Management, the clerks had to record operator job times on a separate form for the weekly run of the Labor Costing application. The overall effectiveness of Shop Floor operations could be reported only by sorting and merging the files maintained by both of these applications. Figure 1.3 illustrates the kinds of interdependencies between applications created by affinities between data items or by data duplication. For example, data describing parts and assemblies in inventory was spread across applications managed by Accounting, Production and Stores, since some of the items were purchased and some were produced in-house. Maintaining consistency became increasingly difficult as the business grew, requiring more complex sorts and updates over larger numbers of files.

This portfolio of complex interdependent applications eventually became fragile, as overnight processing approached the limits of the time window. During the sixties, CTI twice upgraded its IBM mainframe. By the end of the decade, however, the portfolio was so complex that new applications took unreasonable amounts of time to develop. When the Director of Sales and Marketing was told that his campaign effectiveness report would take two years to implement, it became clear that something would have to change, if CTI was to remain competitive. Obviously, hardware upgrades would not solve the problem. Application architectures and development methods would have to change.

Developers could not keep up with demands for new applications.

The Terminal/Host Era

The innovations that made the necessary changes possible were transaction processing systems, database management systems, terminals, fast random access disks, and networking hardware. Introduced in the early 1970s, they collectively created the Terminal/Host Era illustrated previously in Figure 1.1. *Data integration* was needed to solve the data duplication and dependency problems, and *data independence* was needed to solve the application fragility problems. These were the two siren calls of this era. Database management technology provided a way to integrate the islands of data and to isolate programs from physical data storage structures. Data communications and transaction processing technology provided the means to serve many types of clerical and managerial users across the business.

Database, transaction processing and network technology heralds a new era.

12 Chapter 1

The industry has a vision of widespread online access to information.

A vision of online access to enterprise-wide data was established. Everyone's job would be made easier by the new technology. Shop floor staff, order entry clerks, and storekeepers could directly enter operational data without going through data entry staff. Some of the data entry and clerical staff would be displaced, but they could be reassigned to do more productive work using the new green-screen 3270 terminals. Once the data had been integrated into the new database, managers would be able to use report generators to define new reports, eliminating the need to wait for the Data Processing staff to produce one-off programs.

The new technology gives rise to new application types that offer huge efficiencies.

New kinds of applications that exploited the new platform technology, inconceivable in the Batch Era, were now possible. CTI upgraded their mainframe and installed IMS DB/DC. A database was built to manage customer order information. Using a simple linking technique (starting a new transaction using the same customer key), sales staff using the new online order entry application could see while a customer was on the telephone whether there were any bills outstanding for the customer, and could take action depending on the severity of the problem, ranging from a reminder to order denial. Using this real time information saved CTI hundreds of thousands of dollars per year in bad receivables.

Batch applications were rewritten.

The batch applications that once supported Production were rewritten. Their data was extracted and placed into the new database, and their functionality was recast as the Materials Requirements Planning (MRP) application. The new programs could be used via online terminal by shop floor workers, storekeepers, and planners, enabling the real time entry of time and material data. Up-to-the-minute status reports allowed managers to respond to factory events, load balancing, and changing job schedules in real time. The MRP application saved millions of dollars by raising operator efficiency and reducing material waste, and enabled CTI to outstrip competitors slower to adopt these technologies. Data independence created huge savings during the maintenance phase of an application. New data items could be added to the database with minimal disturbance to running applications, and physical database designs could be changed to accommodate new usage patterns without breaking applications in production.

Information technology as competitive advantage.

Budget for data processing grew enormously in the mid-1970s, as online access to integrated data became a major

Introduction 13

competitive advantage. At CTI, it reached 10% of net revenue, a typical figure for the period. The Data Processing group, now called the Information Systems Department, became a top-level organization with a direct report to executive management.

Of course, there were downsides to the new technology. Developers had to learn new programming skills. Administrators had to learn new servers, operating systems, and communications infrastructure. Whole new disciplines appeared to support the new technology. Database administrators, network architects, and data modelers were in demand by the early 1980s.

Despite the hopes of the data modeling evangelists of the early 1980s, widespread data integration across multiple business departments proved a tantalizing vision that was extremely difficult to realize. Many ambitious attempts to produce enterprise-wide data models for enterprise-wide databases foundered due to *analysis-paralysis*, a term that summarized the difficulty of achieving enterprise-wide consensus on the design of shared resources, and the resulting delays in starting the actual coding of applications. Data integration was ultimately achieved, but it was often reduced to a more localized scope than originally envisioned, such as a department or perhaps a division, leading to the “application stovepipe” problem that we see today, where applications in different parts of the organization have different architectures, making them difficult to integrate. Integration across databases was often rudimentary, and most often implemented by “swivel-chair integration”¹ or redirection from common data elements on the screen. This can be seen in a snapshot of CTI’s application portfolio circa 1980, shown in Figure 1.4.

This era produced a net increase in skills required, since, of course, many batch applications did not disappear. Some online applications were older batch programs with CICS or IMS/DC facades, added to allow broader access. For example, it was common to build an online application to allow clerical staff to enter data for the original batch application. This was the case at CTI with respect to Payroll. The original batch application had become expensive to maintain, but was lower in priority for rewrite than the MRP application. Consequently, an online time-reporting application was created by batching data and sending it to the weekly run of the original Payroll application that was still maintained by IS staff.

The new technology created new disciplines with significant learning curves.

Enterprise Data Modeling was mostly a failure.

Some older applications remained in service, causing maintenance headaches.

14 Chapter 1

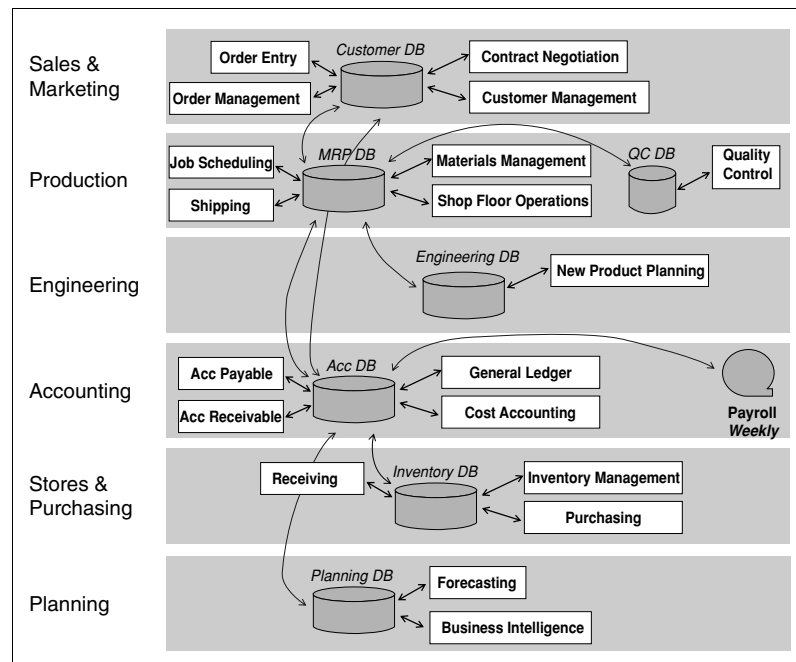


Figure 1.4 CTI's application portfolio circa 1982

And all the backlogs and frustrations reappeared.

In some ways, IS was too successful for its own good. The terminal/host era brought unprecedented growth in business efficiencies, and propelled IS executives to key positions in major organizations, creating the role of the Chief Information Officer (CIO). As IS budgets for new hardware and software grew, however, dissatisfaction set in, again. Application backlogs increased, and eventually became worse than they had been in the early 1970s. By the 1980s something had to change to allow IS to meet the increasing stakeholder expectations. The stage was now set for another major change in platform technology.

The Personal Computing Era

The PC had a profound effect on business applications and IS staff.

That change was provided initially by innovations that led to widespread availability of the Personal Computer and local area networks. In 1985, CTI employed more than 1,000 people and had annual revenues of \$700 million, pushing it into the lower end of the large corporations category in the USA. The IS department now employed more than 150 people. By the end of 1986, the CIO estimated that there were more than 250 PCs

Introduction 15

in use throughout the company. Most of them had been bought on departmental budgets and most of the traditional users of green-screen terminals had either moved to PCs for their connection to the mainframe, or were hankering to do so. What caused such a significant change in user practices?

Much has been written about the effects of the PC revolution on businesses and information technology. For example, Vaskevitch [Vas93] presents the perspective of a Microsoft® executive on this phenomenon in his book *Client/Server Strategies*. He identifies two key issues that led to a dramatic change in platform technology and the application portfolio, creating serious challenges for IS developers:

- Proliferation of departmental applications written by individuals or small teams of people outside IS.
- Changes in common perceptions about user interfaces to computers, creating demands for PC capabilities in mainframe and server-based enterprise applications.

With the availability of PC applications like MultiMate, Lotus 1-2-3, dBASE and WordStar, staff outside the shop floor, who became known as knowledge workers, had enjoyed the ease with which they could describe and manage budgets, design and run multiformula forecasts, write memos, manage project staff, and organize databases relevant to their work. They effectively became *custom application developers*. As Vaskevitch asks,

Why wait for IS to develop a sales forecasting system when your local hacker can build the same system in three days?

Why wait, particularly when the system developed in three days will be easier to use, more flexible, and run on cheaper hardware?

And most of all, why wait when the IS [solution] will take two years, cost tens of thousands of dollars, and then be too expensive to run anyway?

Of course, these applications were simple. They rarely had to deal with shared data, multiple users, or backup and restore. They added to the concerns of IS staff, however, since their users required helpdesk support and assistance with upgrading to new versions of PC applications and operating systems. By the end of the 1980s, these concerns had become serious. Information technology bifurcated into two streams, one for

The PC revolution created serious challenges for IS developers.

Knowledge workers became application developers...

... But still required support from IS.

16 Chapter 1

personal computing, and one for “serious” enterprise applications. The software industry bifurcated too, along similar lines. Despite these changes, most IS departments still struggled to keep up with the demand for new applications and for the maintenance of older ones. The only difference was that now they also had to satisfy the demands of a growing population of PC users. This illustrates the phenomenon described by Christensen, who notes that newer, cheaper products are constantly cutting into the low end of the market. While they may not initially pose serious threats to established products, they often improve over time and ultimately displace the market leaders [Chr97].

Users demanded integration between their enterprise applications and their PCs.

Added to this problem was another consequence of the proliferation of business PCs. Attracted to the PC by the rich user experience, graphical user interfaces and local personal storage, knowledge workers demanded that corporate applications be updated to exploit these benefits. They demanded not just connectivity to other PCs via local area networks, but also connectivity to enterprise applications above and beyond running IMS DC transactions in a window. This is not to belittle the use of PCs as replacements for 3270 terminals. Many knowledge workers wanted better integration of data across separate departmentally oriented host applications. Copy-paste integration was far better than swivel-chair integration or transaction linking. Beyond these simple, but real, productivity gains was the desire to make the PC part of the enterprise application. For example, at CTI, managers in the Accounting department not only wanted to use their PCs as replacements for their 3270 terminals, they wanted to make their Lotus 1-2-3 forecasting spreadsheet part of the Accounting application suite. They wanted to extract information from the database, perform local analysis, and then write the results back to the database.

The Client/Server Era

The PC raised the expectations of business users and heralded the start of the distributed computing era.

With the main focus on data integration, and each database mostly managed by a large server or mainframe, there was little real demand for developers to build features that required connectivity. Connectivity protocols were arcane and were generally used only in system code. But the widespread use of the business PC again raised stakeholder expectations. To achieve the level of integration mentioned earlier, developers had to

view applications not as a monoliths, but as distributed entities composed of connected parts. In the early days of the client/server era, part of the application, called the client, ran on the PC, and part, called the server, remained on the mainframe, or ran on a departmental computer. For client/server architecture to become mainstream, connectivity protocols had to become simpler, so that developers with ordinary skills could use them. This was achieved by standard and proprietary remote procedure calls (RPC).

The first client/server application at CTI was an adjunct to the Order Entry application. In 1988, the Sales Manager defined a set of discounting rules and built them into a dBASE application. The dBASE application extracted information from the Customer Database and applied the discounting rules to support the completion of order forms when customers telephoned or faxed orders. Completed orders were posted as IMS DC messages using terminal emulation software installed on the PC. This example reveals the features of early client/server applications, which used the local processing power of the PC to create a “thick client” for mainframe-based applications, as illustrated in Figure 1.5.

Early client/server systems matched the power of the PC to 3270 terminal emulation.

More complex client/server applications that had to be written by the professional developers in the IS group included quality control applications, which used the PC to perform statistical analysis on data extracted from the MRP database, and business analysis applications, which used spreadsheet programs like Excel to drill down into sales data sliced by region, by customer type, and so on, and to display the results

New higher level programming languages made complex applications much easier to write.

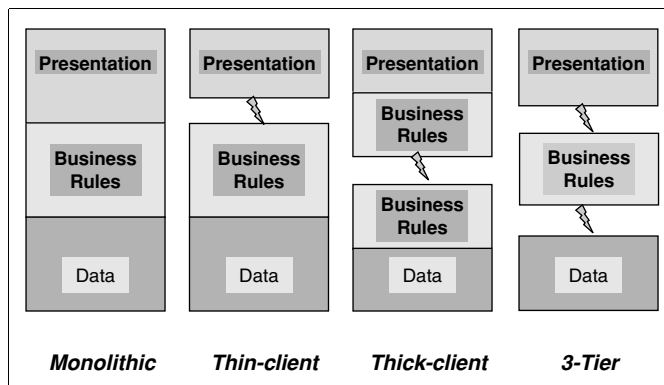


Figure 1.5 Common application architectures

18 Chapter 1

graphically. Until the early 1990s, these were challenging applications to build, since RPC programming was too difficult for all but the top 10% of developers. New higher level programming languages like dBase from Ashton and Visual Basic from Microsoft made client/server programming much easier by introducing event-based user interface programming abstractions like Form and Control, letting almost any developer write efficient client/server applications.

Client/Server had downsides too—like deployment problems...

While the first generation of client/server applications brought improvements in usability, local processing and the disconnected use of mainframe-based data, it also compounded software deployment problems. Many client/server applications were no more flexible than the mainframe applications they replaced. While they were physically partitioned into client and server components, they were logically monolithic, hard wiring every client to a specific server in a fixed location. This lack of flexibility played havoc when programs moved from development to test, and from test to production. Also, when the client had to be upgraded, tracking which PCs had which version of the client software was a major headache. What would happen if a key PC was turned off when a major upgrade occurred?

...data duplication and data staleness.

Problems that were solved in the terminal/host era began to resurface. If data was moved from mainframe systems to PCs, how could IS avoid the problems of data duplication and data disintegration that were overcome by moving from batch to terminal/host? Making data read-only at the PC was one answer, but then how could it be as fresh as the business demanded? This problem surfaced in the order entry application at CTI. The customer data snapshot was downloaded to the order entry clerk PCs only once a day. Bad discounting decisions could be made if a receivables problem was found after the snapshot was taken.

The Growth of Packaged Applications

The 1990s was an era of rapid sales growth for packaged application vendors.

By the end of the 1980s, demand for both new applications and updates to existing ones far outstripped the capability of most corporate IS groups to supply them. And the bifurcation into Personal Computing and Enterprise Computing pressured IS budgets. Together these effects created a huge growth in the packaged application industry, especially in the early 1990s, as IS groups turned to packaged software vendors, who

Introduction 19

offered to meet their needs with ready-built software. Packages did a good job at providing well-structured databases that served the business needs of functional business areas like payroll and financial control, and packages such as R/3 from SAP dominated the market for the broader Enterprise Resource Planning (ERP) applications. No matter that often it was easier to change business practices than to change the package, important functionality and high degrees of data integration could be had without waiting for customized applications to be developed internally. Yet despite providing broader data integration, packaged applications used limited and proprietary mechanisms for accessing data, making it very difficult to integrate them to support cross-function business processes.

CTI succumbed to this temptation in 1991. They started with the old payroll application, which was still essentially a much-modified variant of the original batch application from 1968. It had become brittle, making it hard to change without breaking it, and only a few developers who could maintain it were still at the company (it was written in COBOL and RPG). Faced with yet another change in employment law, the CIO declared that the existing application could no longer be altered and a search was made to purchase a ready-built package. PeopleSoft's package was selected and installed in a six-month project, and the old payroll application was taken out of service at the end of 1991. The other employee applications were converted to work with the PeopleSoft application since they provided functionality not present in the package.

After 10 years of service, the CTI home-built MRP system was also showing signs of age. Faced with the task of adding client/server functionality to the old application, the CIO elected to replace it, along with parts of the accounting application, with an integrated package that was already based on client/server architecture. The main Accounting Database remained, since many of the business rules for accounting at CTI resided there. In 1994, after a lengthy installation project, CTI joined thousands of other US companies switching to the R/3 ERP system from SAP, as illustrated in Figure 1.6. In the process of selecting R/3, CTI also took a major decision to move away from IBM mainframes. Along with the installation of PeopleSoft and R/3, the company purchased several large server machines running UNIX from Hewlett Packard, became the proud owner of relational database technology, and entered forever the wonderful world of heterogeneous platform technologies.

CTI buys a package to replace its ancient payroll application ...

... and then another to replace and upgrade its old MRP application.

20 Chapter 1

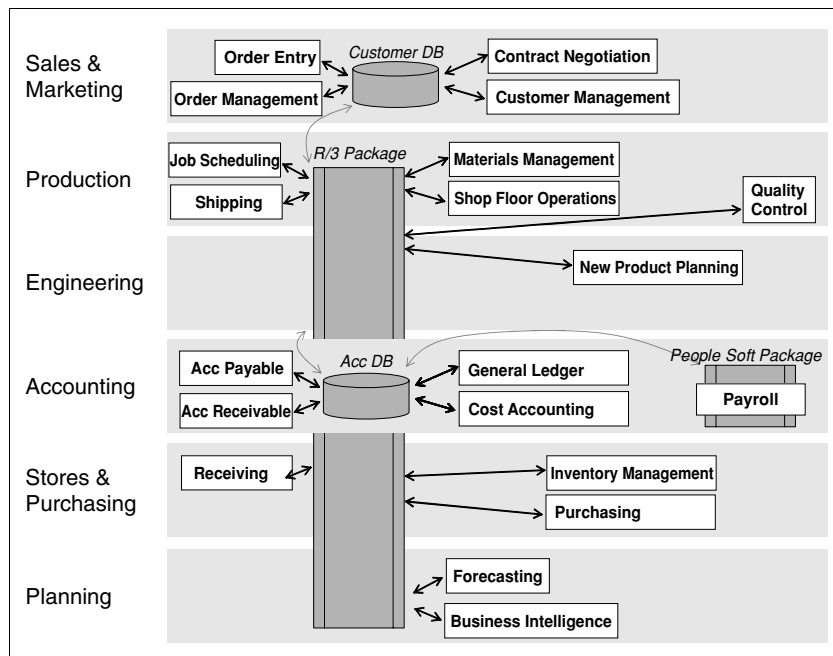


Figure 1.6 CTI's application portfolio circa 1994

The Rise of Outsourcing

Outsourcing is a reaction to rising data center costs and application backlogs.

Another phenomenon that arose in the 1990s in response to the application backlog at in-house IS groups, and the rising cost of datacenter operations, was outsourcing. In its extreme form, outsourcing can mean handing over all responsibility for datacenter management to a third party—eliminating a large part of the in-house IS staff, and most of its hardware. Another trend that continues to grow in popularity is outsourcing application or component development to offshore programming teams, who command lower salaries than their peers in the U.S. do. There is much political controversy about this activity, and what it means for the home market in Information Technology (IT) jobs. Such issues aside, there is still some doubt about the effectiveness of outsourcing, due to mixed results.

It can be successful if managed well, but requires a disciplined approach to development.

Having heard about outsourcing in 1997 from a CIO at another manufacturing company, CTI's CIO thought he would try outsourcing a project to a company based in the Philippines that specialized in extending R/3 for manufacturing business intelligence reports. The results were mixed. While the code had few defects, and was delivered on time and within budget,

deployment in the production data center did not go well. This was primarily the fault of the CTI business users and IT staff who wrote the requirements for the project given to the off-shore company. They were not used to writing precise specifications with well thought out scenarios, exception handling, and acceptance criteria. They had also failed to communicate information about CTI's custom extensions to R/3. The result was several costly debugging exercises, and protracted negotiation with the offshore company over the changes required to correct the software. The project was finally completed, but at three times the original cost, eradicating all of the anticipated savings. The CIO decided not to risk another outsourced project until his own organization had much better development discipline.

Stitching Applications Together

By the late 1990s and beyond, businesses were realizing that data integration, though important, was not enough, and that it was also necessary to identify, manage, and integrate business processes. Business processes usually span several organization units, and therefore had to be automated by stitching together multiple applications. Since these applications and packages often ran on different servers, simplifying server-to-server connectivity was an essential prerequisite. But viewing them as components from which cross-function business process applications could be assembled raised a number of issues. Restricted access to data within the packages was a key challenge, as was data transformation between them, which was usually unable to preserve all of the information. A category of software known as Enterprise Application Integration (EAI) software was created to solve these problems and to manage the flow of data between packages, creating the illusion of an integrated process-oriented application.

At CTI, as we've seen, heterogeneous platforms are the order of the day. Writing cross-functional code therefore forces developers to also cope with server-to-server code that jumps between different platform technologies, requiring bridges and special adapters. Figure 1.7 illustrates this with some common cross-functional business processes that became the subject of industry discussions and *Harvard Business Review* articles. Brokering technologies that addressed a few of these connectivity

Distributed computing is driven by the need for business process integration.

DCOM and CORBA were partially successful but were too closely tied to component implementation technologies.

22 Chapter 1

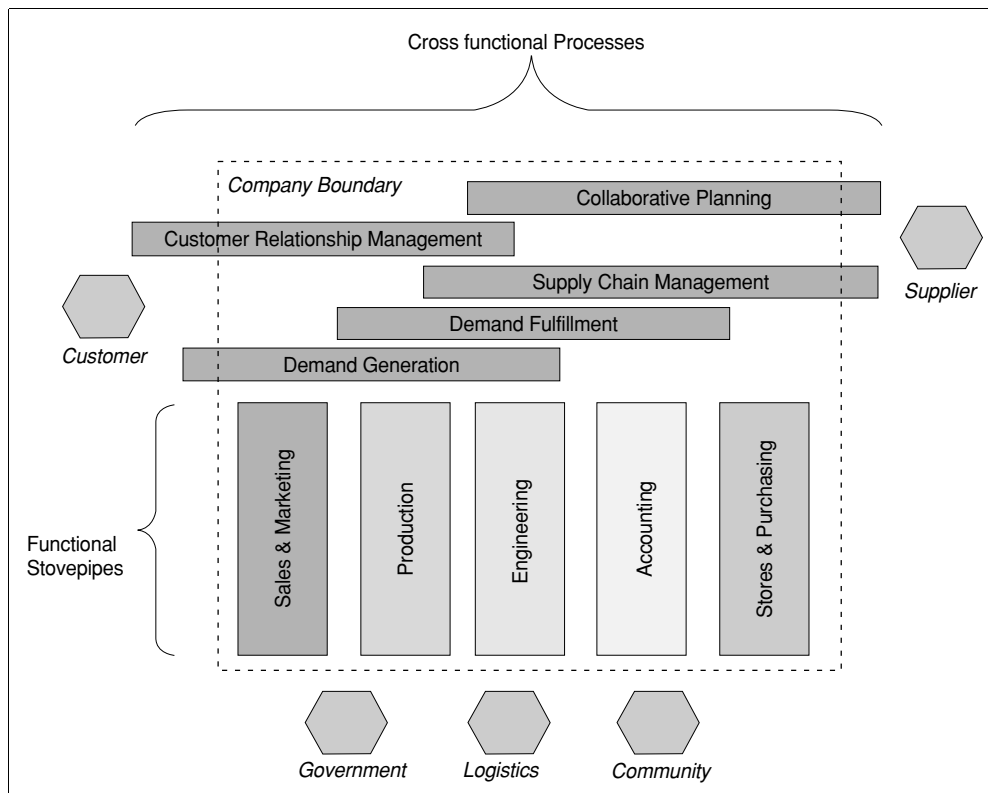


Figure 1.7 Application stovepipes and business processes

issues were available from several vendors as implementations of the OMGTM CORBA[®] specification, and as a technology called Microsoft DCOM[®] for the Microsoft Windows[®] operating system. While these technologies were successful within their domains, problems with cross-platform scenarios, and their close coupling to specific component implementation technologies kept them from becoming more widely adopted.

To support business process integration we need a process perspective and multiplatform technology.

The application stovepipes that most companies found in their data centers by the late 1990s created inertia against the construction of cross-functional applications to support reengineered business processes. Technology resisted the early adopters of process-oriented applications, and while that is changing with modern Web service technology, this is more than just a technology issue. In some respects, Hammer's classic "stop paving the cow paths" paper on business process reengineering was ahead of its time [Ham90]. To avoid integration problems, applications should be built from a process perspective, using widely accepted technologies that work on multiple platforms

and that adhere to industry standards. We address these two issues in the next section.

Opening Up the Enterprise

The arrival of the World Wide Web technologies in the early 1990s had a profound effect on us all. We do not need to go into that effect here. But keeping our focus on technology change and its effect on stakeholder expectations and corporate application portfolios, we can observe how the Internet technologies were adopted by businesses. Early use of Internet technologies was for the most part the creation of static content at a web site with public access. Soon, however, it was recognized that the real power of the web for business was in driving electronic commerce—interactions between organizations participating in potentially complex *value chains* offering rich, rewarding business relationships to their participants. Using the Internet, even small companies now handle millions of transactions per business day from thousands, tens of thousands or even millions of concurrent users. The ability to include customers and partners in the scope of an information system makes reengineered business solutions possible.

By promoting near-universal adoption of location-independent protocols, especially *http* and *https*, the web allows distributed applications to include data and services outside the businesses that host them. At the same time, however, these protocols do not yet provide some of the features required to distribute business and mission critical applications, such as transactions, security, routing and referral, and reliable asynchronous messaging. Distributed middleware platforms, such as DCOM and CORBA, provided these features. Because they were tightly coupled to specific component implementation technologies, however, they were difficult to use across business boundaries.

As a standard format for data exchange, eXtensible Markup Language (XML) has been one of the primary catalysts of electronic commerce. XML technologies at either end of an interaction can eliminate many problems related to data transformation between applications. XML documents adhering to well-formed and discoverable rules, described in XML using XML Schema Definitions (XSD), are becoming the standard medium of interaction between connected applications wherever they are located.

The World Wide Web for business is about electronic commerce.

The web extends applications to include data and services outside the business.

XML enable a standard format for data exchange.

24 Chapter 1

Internet technology is also used within companies.

As the nascent Distributed Computing Era progressed, a less visible but equally important development was increasing server-to-server connectivity. While PC-to-PC connectivity had been around since the early days of the local area networks, and PC-to-server connectivity was commonplace, by the early 1990s server-to-server connectivity was still unusual in applications. In this guise, the web has done more to push application boundaries than anyone could have predicted. Many companies have found it easier to set up internal Internet servers using the same protocols and standards—the so-called Intranets. This is the basis for a new approach to process implementation that can supersede early EAI technology.

Web service technology enables application connectivity on an unprecedented scale.

Building Applications with Services

New Web service technologies, such as those being standardized by the World Wide Web Consortium (W3C) and the Web Services Interoperability Organization (WS-I), illustrated in Figure 1.8, promise to provide the necessary features by supporting service-oriented architectures (SOAs), where loosely coupled, coarse grained components interact by exchanging messages. These technologies are also breathing new life into component-based development (CBD) methods, as we shall see later in this book.

It crosses platform and business boundaries.

For most organizations, heterogeneous technology platforms are the norm, so the role of Web service technology as a platform for distributed applications is a lifeline. With Web services, distributed business processes have become a reality, since a software component residing within a partner's business data center can legitimately be regarded as part of a larger application that spans the boundaries of both enterprises. Similarly, software components locked within proprietary packages or

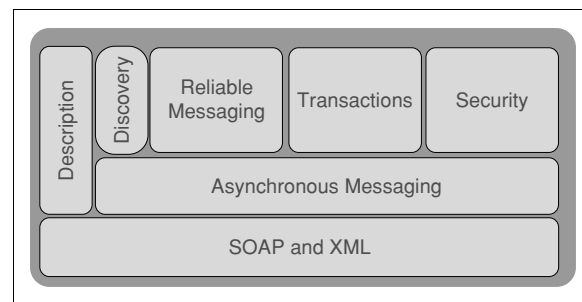


Figure 1.8 Layered web service protocols

Introduction 25

buried in older, more monolithic applications can be wrapped with Web services and easily integrated to better support business processes. As Web service technology matures to better support security, cross service transactions, and other key features, so does its suitability as a platform for connected applications and distributed computing, both between businesses and within them.

The application of Web service technology creates service-oriented architectures, where loosely coupled coarse grained components interact by exchanging messages containing documents according to sequencing protocols defined by well-formed contracts subject to negotiated constraints called service level agreements. We will revisit this definition in Chapter 13. Examples of service-oriented architecture can be seen today in the edge systems that provide facades for back end systems to expose business processes to the Internet. Of course, this is just an early example of service-oriented architecture. As Web service technology matures, we expect to see service-oriented architectures become much more pervasive and eventually ubiquitous.

Inevitably, platform technology evolution has once again come with a price. Building distributed applications using principles of service-oriented architecture is a challenge. Technical issues such as security, versioning, caching, deployment and management, which were tough enough in the client-server era, become even harder with service oriented architectures. Problems created or exacerbated by service orientation are described in the following sections of this chapter.

Many observers believe that the next step in platform technology evolution will be the emergence of Business Process Management Systems (BPMS). These are logical analogs of Database Management Systems (DBMS). DBMS applied structure to corporate data, offered data independence to programs that used the data, and provided a wide range of support utilities, such as common backup and restore, transactions, ad hoc query and business intelligence. BPMS, on the other hand, will apply structure to corporate business processes, and allow them to change independently of functional programs, while providing useful features like process state management, process composition, process tracking and process monitoring. Strong theoretical underpinnings exist for both technologies—relational calculus in one case, pi-calculus (and other formal algebras of cooperating processes) in the other. As BPMS technology matures, process-oriented applications will form the

Web services have popularized service-oriented architectures.

But service orientation does not solve all problems.

Business Process Management Systems are the next step in platform technology.

26 Chapter 1

backbone of application portfolios over the next decade. Let's look at how SOA and BPMS would be used by CTI.

An eCommerce Application

CTI builds an eCommerce app.

CTI has once again found its stakeholders raising pressing requirements for new kinds of applications already in use by their competitors and partners based on the new Web service technology. Three main business drivers are the following:

- CTI has produced a new line of power tools and accessories specifically oriented towards the consumer. Their existing channels have mostly been dealers who sell to large contracting firms, with large volume orders at predictable intervals. The CTI Marketing Director decides that the consumer products will be sold through two new channels. First, via an online web-based catalog that would potentially reach millions of consumers. Second, via a small number of tight collaborative deals with major out-of-town discount retail outlets like Target and Wal-Mart.
- CTI needs to improve its relationships with both its customers and its partners. It has determined that its customer loyalty is lower than the industry average, and that its purchasing activities could be optimized by reducing the number of suppliers and sharing demand forecasts with those that remain, resulting in better terms and faster delivery of goods.
- CTI needs to better track its existing sales force and combine their leads with those generated from online marketing campaigns.

CTI gets help from GSS.

Clearly over its head in facing these challenges, CTI has decided to contract out to Greenfield and Short Software, Inc. (GSS), a systems integrator with a track record of helping its clients integrate their value chains. GSS works with CTI to describe its business processes and the flows of information among them. The result of this effort is a model of the CTI value chain, illustrated in Figure 1.9. The key to helping CTI will be to facilitate the exchange of information among the processes that comprise the value chain. Figure 1.9 shows groups of business processes that GSS has identified for CTI. While

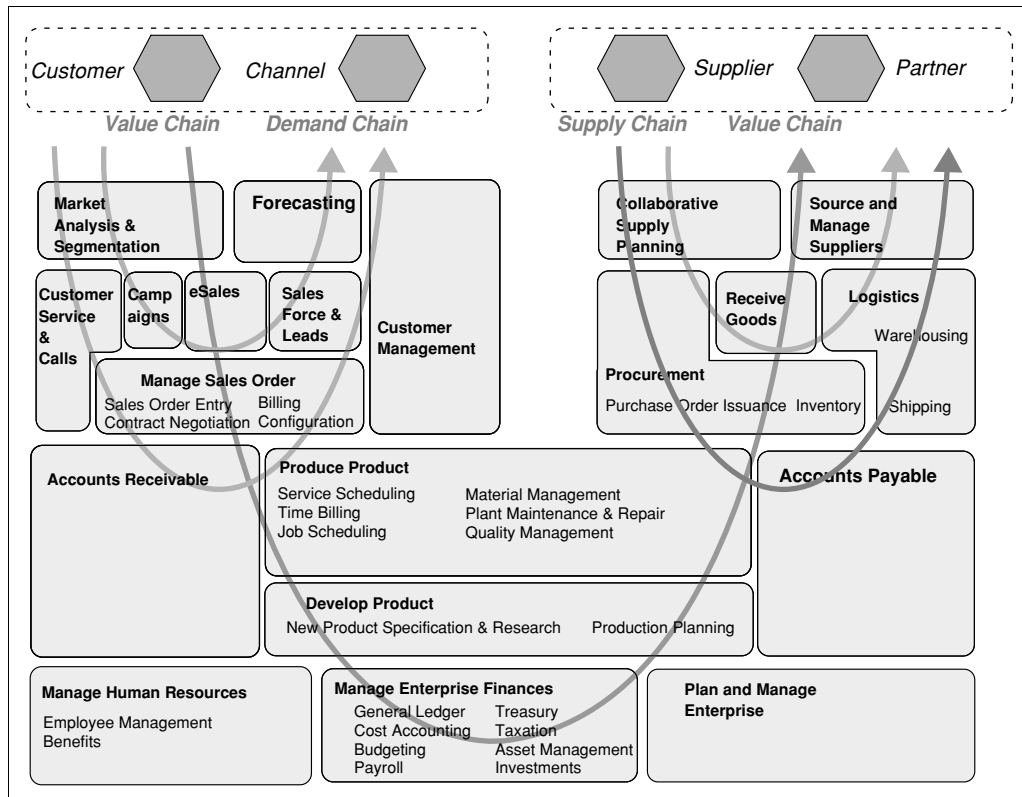


Figure 1.9 Business process map showing value chain

this kind of chart is useful as a summary for executive feedback sessions, more detailed business process definitions are drawn up by GSS consultants working with key CTI people. This analysis of the way CTI does business, combined with the process models to meet the new requirements, is a critical prerequisite used by the GSS systems staff in designing software components that will wrap existing CTI applications, or implement new functionality. The new business processes are summarized next.

Business Processes

The Web Site Management process will provide a business web presence and the basic web-based storefront for CTI, as illustrated in Figure 1.10. It defines how CTI will setup e-mail accounts, and how it will design and manage the layout and content of its web site. Other activities supported by the process

Web Site Management Process

28 Chapter 1

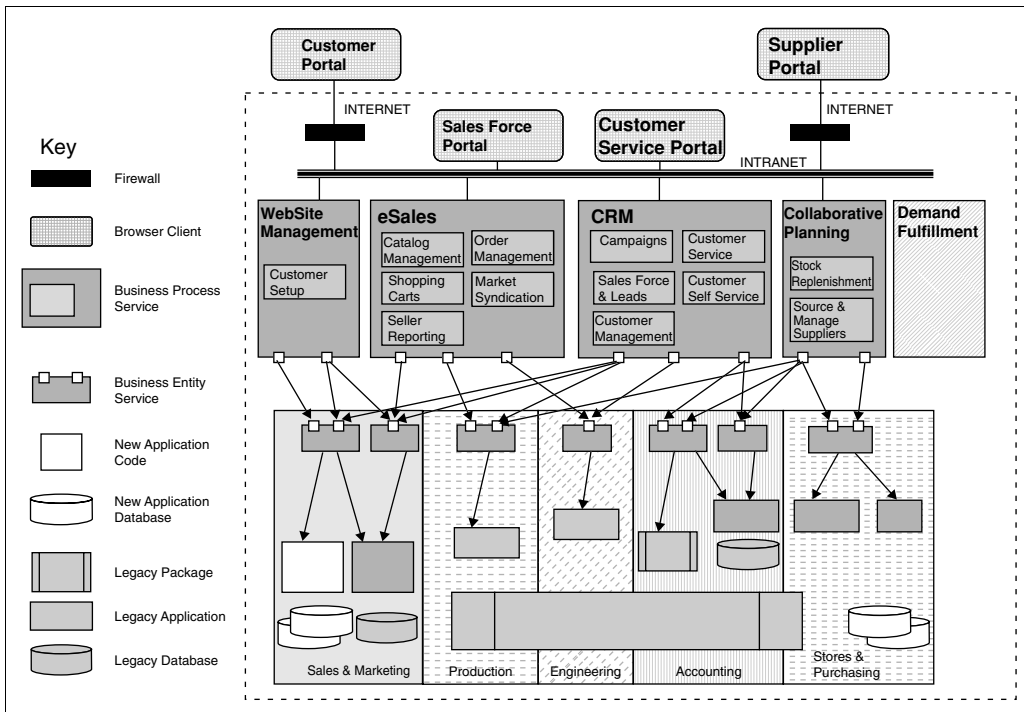


Figure 1.10 Service-oriented architecture for the new business processes

include registration of the site with search engines and submission to banner exchange networks.

Customer Relationship Management Process

The Customer Relationship Management (CRM) process will provide a complete suite of tools to automate and improve customer acquisition, conversion, service, retention and feedback gathering. It contains the following sub processes:

- Campaigns: the process used to create, execute, and track direct marketing campaigns, initially focusing on e-mail direct marketing.
- Sales Force and Leads: the process that defines how sales people and sales managers track and manage their leads, prospects and current customers, including automated prospecting, sales processes, task management, customer acquisition, team selling, goal setting, and commission tracking.
- Customer Management: the process that defines how customer information is recorded and used, what

happens when a new customer is acquired, and what happens when customers ask to be unregistered.

- **Customer Service:** the process that defines how customers are provided with self-help (knowledge base access, natural language search, solution rating), how incidents are tracked and escalated, how service representative effectiveness is tracked, and how chat/e-mail/phone routing and queue management, and Service Level Agreements (SLA) are managed.
- **Customer Self-Service:** the process that defines how a customer “logged in” on the web site is managed, and how they perform self service, how they check order status, and how they obtain incident or opportunity/sales status, or customized quotes and configurations.

The eSales process contains five primary sub processes:

*eSales—the
eCommerce
Selling Processes*

- *Catalog Management:* the process that enables CTI to enter products and services into their catalog, manage merchandising rules, configuration rules, and create custom catalogs and pricing.
- *Order Management:* the process that defines how CTI manages orders received from any marketplace or partner. It’s also the process by which CTI will configure rules (e.g., discounts) and identify service providers (e.g., credit card processors) required in processing orders
- *Shopping Cart Processing:* the process that defines how orders from customers are handled, and that defines order self-service activities allowing customers to check on the status of their orders and retrieve their order history.
- *Market Syndication:* the process that defines how CTI submits catalog data to marketplaces, such as eBay.
- *Seller Reporting:* the process that gives Sales Management a snapshot of sales activity across all their partners and marketplaces.

The Collaborative Supply Planning process contains two primary subprocesses:

*Supply Chain
Management
Process*

30 Chapter 1

- *Stock Replenishment*: the process that defines how stock from inventory is replenished by placing restocking orders to suppliers, and forecasts are shared with suppliers to enable automatic restocking and goods receipt.
- *Source and Manage Suppliers*: the process that defines how suppliers are vetted and selected for close collaboration.

An overview of the implementation of the new processes.

Implementation Strategy

CTI and GSS decide to implement the new set of business processes using a service-oriented architecture. The user interaction scenarios will be encapsulated in portals—easy to use and personalized web pages that provide access to business processes. Each business process will be implemented as a Web service, with its lower level activities performed by a new type of service—a business entity service—which takes responsibility for reusable business logic and access to underlying data stores. Since these data stores and their accompanying business logic still reside in older databases and packaged applications, the architecture must define the new services as facades that wrap those assets. These services will manage implementation of the newly defined service by working with older databases and packages. This is a complex undertaking for CTI and GSS. We'll return to see how this was finally accomplished using a software factory in Chapter 17.

We summarize the landscape as a list of development challenges.

Software Development Challenges

This concludes our overview of the landscape that provides the backdrop in which the development of new applications must take place. We have highlighted many serious challenges. To summarize, software developers must:

Some are created by new business requirements.

- Support reengineered business processes and an increasing focus on process-oriented applications
- Expose existing systems to massive user loads created by web-based front ends
- Design protocols (valid message sequences) and enforceable service level agreements to support processes that span multiple enterprises

Introduction 31

- Determine strategies for versioned data and snapshots, such as price lists, that are widely distributed yet have limited lifetimes
- Cope with the complexity created by transformed business models from business leaders such as Wal-Mart, who insist on deep integration with their partners
- Integrate heterogeneous application stovepipes and avoiding lossy transformations between them
- Avoid reintroducing the batch era problem of multiple file layouts and lack of data format consistency in the rush to describe XML schemas for every software service
- Determine strategies for wrapping older applications executing on heterogeneous platforms
- Customize packaged software to satisfy proprietary requirements
- Address the special needs of data warehousing and business intelligence
- Demonstrate return on investment in custom software in the face of rising software development costs
- Protect corporate data from hackers and viruses, while giving customers and partners direct access to the same resources
- Mitigate the increasing risk of legal liability from the improper use of corporate data
- Satisfy operational requirements, such as security, reliability, performance and supportability, in rapidly changing applications
- Integrate new and existing systems using a wide range of architectures and implementation technologies
- Understand the effects of partitioning and distribution on aggregate qualities of service
- Design multitiered applications that deploy correctly to server farms on segmented networks partitioned by firewalls, with each server running a mixture of widely varying host software configurations
- Support applications developed by end users (for example, spreadsheets to 4GLs) while enforcing corporate policy and maintaining the integrity of corporate data

Some arise from an increasing focus on security.

Some are caused by increasing deployment complexity.

Some are created by decentralized software development.

32 Chapter 1

- Integrate personal productivity applications, such as word processors and spreadsheets, with back end systems
- Work with applications or components that are increasingly outsourced to development centers in remote locations forcing a discipline on requirements designs and acceptance tests that was often neglected in the past
- Make departmental applications integrate effectively and scale to satisfy enterprise requirements

Discontinuous Innovation

The industry will not solve all of these problems through business as usual.

Software development technology evolves primarily through disruptive changes called paradigm shifts.

Now that we know the challenges we face, we can ask what methods and practices are at our disposal, how effective they are, and whether or not they can be improved upon. The remainder of this book seeks to answer these questions. Solving software development challenges is a major focus of the industry, and tremendous progress has been made in a mere 50 years. Historically, however, this progress has not been steady. Instead, it has followed the pattern of innovation curves, illustrated in Figure 1.11.

A discontinuous innovation establishes a foundation for a new generation of technologies. Progress is initially rapid, but gradually slows down, as the foundation becomes stable and mature. Eventually, the foundation loses the ability to sustain innovation, and a plateau is reached. At that point, another discontinuous innovation establishes another foundation for another generation of new technologies, and the pattern repeats. Kuhn calls the foundations paradigms, and the transitions between them paradigm shifts [Kuh70]. According to

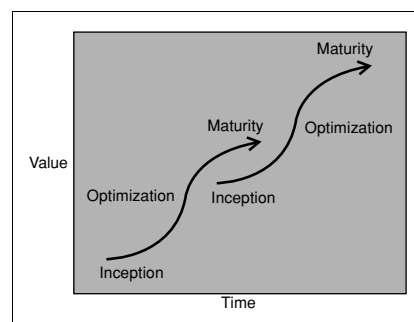


Figure 1.11 Innovation curves

Cox, each new paradigm is launched to correct the failures of its predecessors, and to explore new thinking that breaks away from established practice [Cox90]. Because they are produced by discontinuous innovation, paradigm shifts are by nature disruptive, and often interfere with existing social and economic structures. According to Shaw, engineering-based industries, like software development, are prone to paradigm shifts because they are driven by underlying currents in science and mathematics [Sha97].

Object orientation has changed much since its inception, but remains the current paradigm. It improved on structured programming by providing a way to encapsulate state in behavioral wrappers. A good overview of the paradigm is provided by Taylor [Tay97]. Like most new paradigms, it initially had many problems could only be corrected by gaining experience. One problem was the assumption that object-oriented solutions would be resemble the problems they solved, allowing functional requirements to map rather directly onto classes. Another problem was that inheritance, as initially implemented, violated encapsulation by allowing subclasses to depend on private features of their base classes. A third problem was that during early adoption, many object-oriented programming practices resembled practices established by the structured programming paradigm. These and many other problems were solved, some completely, some partially, by incremental but significant adaptations, such as interfaces, patterns, and agile methods.

Sadly, despite this tremendous progress, the vision espoused by the pioneers of object orientation, with its emphasis on industrialization, has not been realized. Many problems remain unsolved. Some have resisted solution since the inception of the paradigm, while others have simply been overlooked. According to Kuhn, this should be expected. A paradigm provides focus that accelerates the solution of problems at the center of its concerns, while distracting attention from those on its periphery. As the problems at the center are gradually solved, attention begins to move to the ones on the periphery, which then begin to loom large as impediments to progress. The focus that enabled the paradigm to accelerate the solution of the problems it chose to emphasize now prevents it from offering assistance in solving the problems it chose to deemphasize.

We think innovation based on object orientation has reached a plateau. Indeed, with a few notable exceptions, such as

Object orientation is the current paradigm.

Many parts of its original vision have been realized, but many have not.

Object orientation has reached a plateau—a new paradigm is needed.

34 Chapter 1

patterns and byte code languages, most of the innovations of recent memory represent only incremental progress. We still build software in basically the same way we did when early object-oriented languages like Smalltalk arrived on the scene. Perhaps the biggest change is that now these practices have been adopted much more widely, though imperfectly. In other words, there is evidence that we are near the end of the current paradigm, and that a new paradigm is needed to support the next leap forward in software development technology. If this is true, then we should be able to identify chronic problems with the current paradigm, problems that have stubbornly refused solution for a long time despite the best efforts of the industry to solve them. We should also be able to identify critical innovations that solve those problems at the cost of changes in accepted methods and practices. According to Kuhn, the new paradigms will build on the strengths of their predecessors while addressing some of the weaknesses that gave rise to their chronic problems. They will also create technical, social, and financial dislocations by disenfranchising entrenched constituencies.

On to the rest of Part 1.

In Chapters 2 and 3, we study complexity and change, the two fundamental forces responsible for the failures of object orientation and every other paradigm invented in pursuit of developer productivity. We also look at methods and practices that have been added to the foundation of object orientation, since they show us where to look to find the chronic problems that it cannot solve, and the critical innovations that will launch the next offensive in the assault on complexity and change. In Chapter 4, we describe those chronic problems and critical innovations. In Chapter 5, we explain how those critical innovations could be applied to form a new software development paradigm called Software Factories.

Notes

1. It was not uncommon for clerks to have more than one green screen terminal on their desk for this purpose—hence swivel-chair integration. Thanks to Pat Helland for this insightful terminology.