Software:
Designing for Profitability

Required Design Concepts for IMPROVING BUSINESS PERFORMANCE

O:\PROMO\BUSINESS\DESIGN BOOK\000-DR-Design-Book - v7.doc
created : 12/12/2011 12:32:00 PM
printed : 12/12/2011 2:57:00 PM

Table of contents

Table of Contents

1 Profitability and Software

Profitability depends on generating revenue in the most effective operational fashion, whether you are manufacturing widgets, or carrying out complex discovery in a legal case.

Our most fundamental principle is that sound software design can help your company make money and improve profitability.

This is a complex subject that most software designers avoid tackling head on, preferring to promote (business-IT alignment).

We feel our framework can provide much of what is missing in the theory and practice of making software systems work well from a business perspective.

Too often, software deliverables "meet the spec" in the strictest definition of the term, but they yet do not fully support business needs. Sometimes software systems have designs flaws that render them only marginally effective, and occasionally impossible to use.

1.1 Shift the perspective from technical to strategic

The first area where we provide critical contribution is in shifting the focus from technical construction to a business and strategy driven focus. Pursuing this objective, we identify specific logical capabilities that can augment and extend the core software functions and make sure these software assets support more than just the narrow elements of a business function specification.

Systems do things that the business logic requires.

- Tracking physical material
- Understanding the status of customer accounts
- Providing new features for a healthcare product

That can help your business thrive. However, in times of increased stress in the economy, business activities require commensurate increases in power and flexibility of information systems they depend on.

In a contracting, recessionary economy, cost management and cost reduction take on priority importance. Most companies have to do much more than just implement a 5% or 10% staff reduction. Often fundamental improvements are required – just when there are NO budgetary resources to support such initiatives.

To reduce costs, businesses must make better use of computers – replacing administrative labor and overhead with faster, more accurate, and more responsive computer systems. However, that change places stress on the design of most software systems – they just are not that adaptable. We can show you how to make better business processes using better software designs.

Overhaul the way your business processes and information systems integrate.

Design strategies that take advantage of technology are completely different from designs based on manual methods, and they will continue to change along with technological advances.

Replacing labor with automated functions is not always easy. Human cognitive processing performs very well in environments where inputs are varied or where paper forms prevail and present (often) poorly formatted information. Computer systems require more structure than the customer service periods allow.

Even just supporting labor (to gain in accuracy, timeliness, etc) is not easy.

However, although the initial setup can be resource-intensive, automation *will* save money over time.

1.2 Plan for future demands

Another key area where we provide critical contribution is in guiding you as you plan for future demands. Extra time spent during the planning stage minimizes surprises down the road. Furthermore, it can reduce costs as well. Spending extra person-hours on getting the design right "from the start" is infinitely preferable to wasting development time on a design that incorporated flaws from the beginning.

Why do so many designs fail to implement the desired strategy? Is it because of poor communication – stretching, perhaps, across numerous organizational levels and stakeholders? Can it be that, despite decades of technological advancement, business systems still cannot achieve everything that corporate leadership envisions?

These may be factors, and yet in practice, the most obvious reason that crops up repeatedly is the sudden emergence of the unexpected. Sometimes this is due to poor planning – more often it is simply force of events.

2 Strategic Perspective

Senior executives and operating managers just do not have the frameworks in place that can help them control the technology tiger and apply software resources.

Almost all published application architectures and design principles relate to specific code (the technical perspective), rather than to business issues and functionality (the strategic perspective). Software design and strategic design are two different things. Sometimes they are in harmony, but often, unfortunately, they conflict.

Most books and papers on *application design* deal with technical design (Service Oriented Architecture, HTTP Handlers, ASP.NET request processing pipeline, and so on). While these are fascinating topics in their own right, they have little to do with the actual business concepts that lead to lower costs or increased revenue.

2.1 Application Concepts

For the most part, we can treat products and services on an equivalent level of abstraction. Of course, the service framework lacks Material Requirements Planning, Bill of Materials, and Master Production Schedule. Still, services can depend in large part on a product structure, particularly in insurance applications like health care insurance, property and casualty insurance, and life insurance. The concept of *product* is central to delivering services to customers.

Application software must have better facilities for defining, designing, and managing the key elements of the business environment. Further, to support profit-making initiatives, software systems should support certain business-level characteristics, for example:

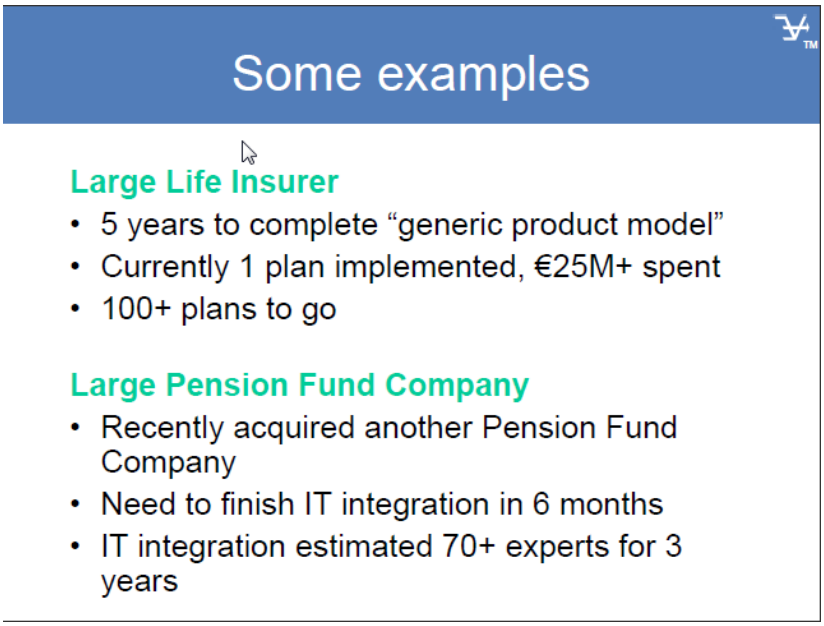
- Product Definition
- Accounts
- Transactions
- Customer Characterizations

Product Definitions

Although product definition is important in most businesses, the following financial products require particular attention:

- Life Insurance
- Annuities
- 401(k) Plans
- CDs

A recent presentation by Intentional Software underscores the magnitude and the expense of design problems in this context.



Accounts

Application software must allow the user to:

- Define accounts
- Establish account properties
- Relate a large set of properties to a single record (owner)

Transactions

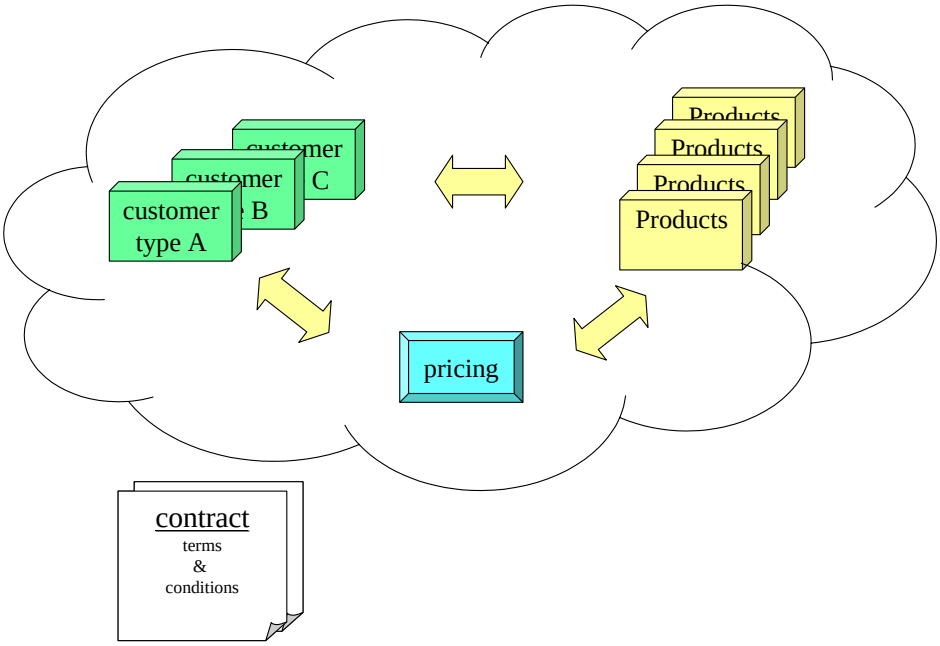
Application software must allow the user to:

- Perform calculations to arrive at tax amounts, discounts, etc.
- Track real world events
- Make the database conform to observed data

Customer Characteristics

In the basic-concept framework that includes customer, product, service, contract and pricing, we can also include other conceptual elements such as distribution channels, outsourcing, regulators, etc.

In this concept framework, we assume that customers fall naturally into *types*. We do the same for products and services, selling sets of products and services preferentially to specific customer types. Pricing establishes the financial outcomes. Servicing determines the cost stream of maintaining a customer in the portfolio over time. For example, for loan origination capabilities, this also includes the option of securitization: packaging up a collection of customers and "selling them off" to other servicing organizations.



Contracts define the terms of the economic exchange between customers and the company, and encapsulate the relationships, expectations, and obligations that Customers and the company have. Activity-based management provides the tools you need to determine which customers and products yield an acceptable contribution to the profitability of the company.

This is all very well – but it is merely a starting point. To enable the rapid, reliable process engineering that most firms desire, the tools and methodologies must "flesh out" this skeletal framework.

2.2 Limitations of the Functional Perspective

Most development processes rely on Subject Matter Experts (SMEs) to provide the structure and details of functional requirements. The traditional approach generally follows these steps:

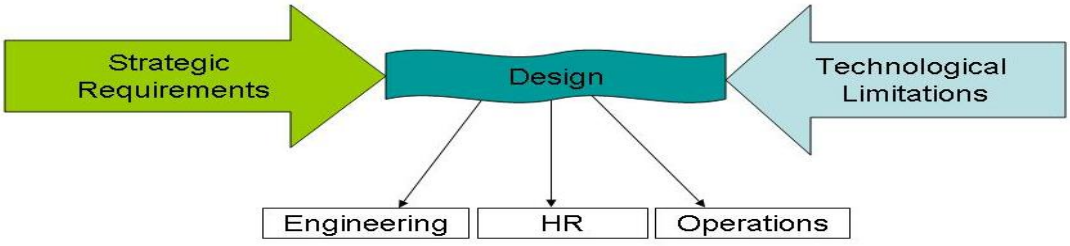
1. Business analysts and designers interview Sees and capture these functional requirements in charts, diagrams, and special languages like Unified Modeling Language (UML).
2. In conjunction with the Subject Matter Experts (SMEs), analysts and designers validate the contents "logically", resolving ambiguities and incomplete requirements through an issue resolution process.
3. Finally, the designers devise a design specification that programmers implement.

If all goes well, the resultant software conforms to the stated business needs and complies with a technical architecture that provides requisite performance.

The process based on the functional perspective is inefficient. First, it takes quite a while to traverse the sequence of events from initial requirement to delivered, tested code. However, let us leave aside for the moment the long "waterfall" time delays from the initial interviews with SMEs to the final production (or in the case of a more iterative process, the several time-consuming cycles required to achieve convergence among the developers).

Secondarily and possibly even more basic and important is the inability of the SME to incorporate vital design features that protect the application from the usual pattern of short-shelf-life.

In short, SMEs have little understanding of good software design, and programmers have little understanding of business requirements. Design, in practice, is a compromise between these two opposing forces, which sadly, are often lack of synchronization.



Furthermore, creating a design invariably demands coordination between different departments. In this example, the design for a new HR application affects HR itself (the users of the system), the engineers who build the system and the Operations team who are responsible for ongoing support.

2.3 Software Capabilities from a Strategic Perspective

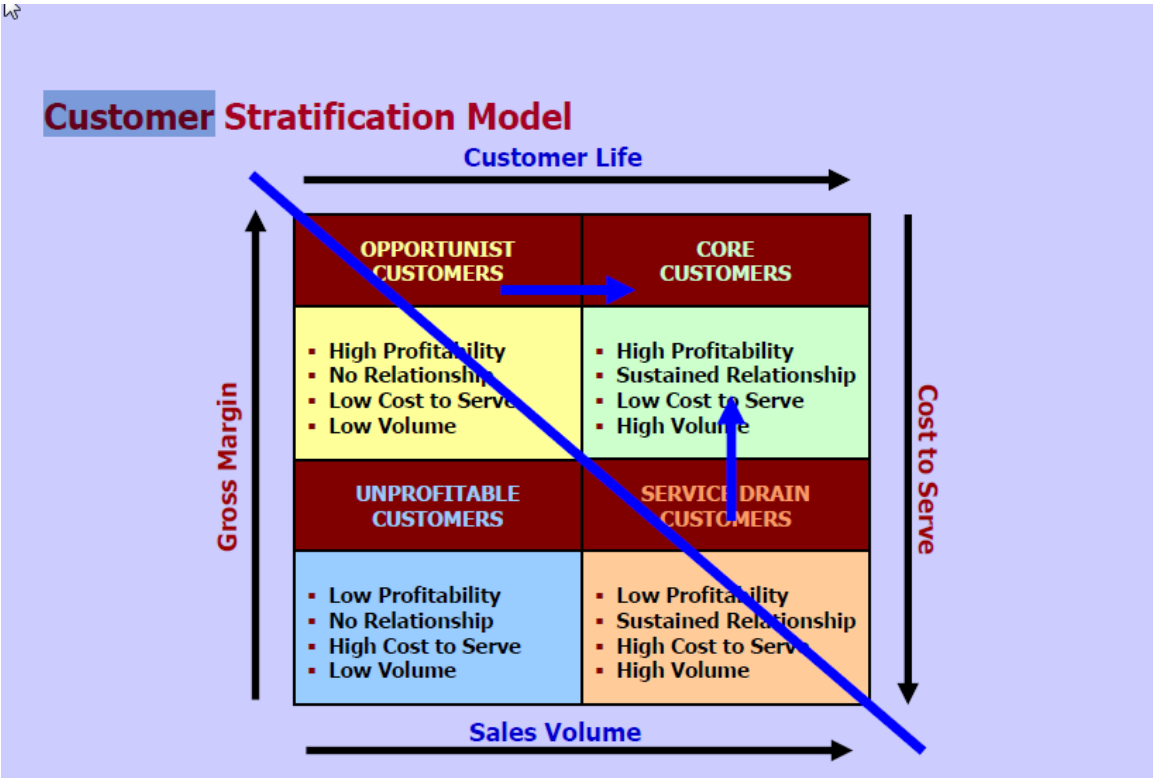
When designing software from a strategic perspective, your application suite should have the capabilities discussed in this section. This list is not comprehensive, but it does indicate the general range of capabilities that you should be able to implement.

2.3.1 Customer Stratification

"All customers are created unequal." Each customer is a unique enterprise. Few systems, however, are able to capture, manage and apply the numerous distinguishing characteristics. To compromise, managers must identify customers using a much smaller set of characteristics, and then adapt the business capabilities to those parameters. This is what we call "customer stratification".

- Identify a set of customer based on sales history, volume, past products, etc.
- Identify customers with a high number of calls to customer service.
- Apply a specific business function to a class of customers.

From the Supply Chain Systems Laboratory at Texas A&M University, separating customers according to observed business impact. Then, one needs to design applications that capture the characteristics that lead to profit outcomes.


Customer Stratification Model
Customer Life
OPPORTUNIST CUSTOMERS
CORE CUSTOMERS
High Profitability
No Relationship
Low Cost to Serve
Low Volume
High Profitability
Sustained Relationship
Low Cost to Serve
High Volume
UNPROFITABLE CUSTOMERS
SERVICE DRAIN CUSTOMERS
Low Profitability
No Relationship
High Cost to Serve
Low Volume
Low Profitability
Sustained Relationship
High Cost to Serve
High Volume
Gross Margin
Cost to Serve
Sales Volume

2.3.2 Product Extensibility

Customers demand increasing "perceived value" (convenience, transparency, and so on). They want the ability to monitor their accounts from web browsers, and they demand a wider range of product options instead of one-size-fits all, or "Bronze, Silver, Gold" limited approach.

To provide this increased perceived value, the product suite should accommodate the following features:

- Add/retract options and features without programming interventions
- Service legacy customers without great cost inconvenience.
- Bundle a specific set of product features into a package with an incentive to the customer to buy without further customization.

3 The Strategic Perspective and Design

In this section, we review the traditional design principles associated with the functional perspective. We then discuss how the strategic perspective can build on and integrate with these traditional design principles.

3.1 Tier Design Model

Designers have long adopted a basic and simplistic three-tier model to describe the organization of business applications. These tiers, or layers, are:

- Data Presentation Layer
- Business Logic Layer
- Data Access Layer

Each of these layers has a well-defined purpose:

- DATA PRESENTATION normally refers to the user interface (the forms that users interact with in their ongoing work). Although the standard principles for good user interface design are well known, designers do not always follow them.
- BUSINESS LOGIC represents the code that provides computational and operational services that the business needs to do its work: e.g. for a bank, handling deposits and withdrawals, for a retailer making inventory adjustments, calculating sales tax, etc.
- DATA ACCESS is the layer that stores and retrieves information in one or more databases. These technical details need careful attention, because they can cause problems if not properly implemented (slow processing time, etc.) but from a strategic standpoint, they are almost independent of business design. Or rather, they have their own design principles that focus on performance optimization and should not cause strategic problems.

Many applications in operation today conform to this basic layer model. As more applications began to take advantage of networks, this layer approach morphed into an *n-tier* model, in which n implies an number of distinct tiers (2-tier, 4-tier, etc.) used in the architecture

N-tier application architecture provides a model for developers to create a flexible and reusable application. By breaking up an application into tiers, developers

need only modify or add a specific layer, avoiding the need to rewrite the entire application, if they decide to change technologies or scale up.

3.2 Integrating the N-Tier Model with the Strategic Perspective

The basic n-tier model is useful technically, and in later chapters, we will offer some analysis of how to optimize it. However, this model does little to provide business executives and managers with the capabilities and functionality they need – both for short-haul operational excellence and for long-term strategic advantage.

By designing software from a strategic perspective, and considering application design principles that may not be apparent in the n-tier approach, you can accomplish both goals.

Stated simply, we adopt a two-part structure:

- Business Design
- Systems Design

BUSINESS DESIGN defines business capability in a competitive context. Moreover, it encompasses aspects of what and how the business carries out its activities. In simple terms, business design is a solution to a strategic imperative. Strategy drives design and determines:

- What the business does.
- How the business does it (including, for example, workflow)

SYSTEMS DESIGN covers similar aspects of the structure and function of information systems. It also defines policies that should set usability, ergonomics of those information systems. Within systems design we find solutions for application capabilities, as well as the technical policies that govern the "look-and-feel" of the code that implements the applications.

4 Design from the Strategic Perspective

In an ideal world, strategy drives the design of information systems. In practice, unfortunately, it often seems as if the actual functionality of the system bears little resemblance to the desired strategy.

The most important reason technology often falls short of fulfilling business needs is the existence of a “built-in” disconnect between business-level and technical-level environments with regard to concepts, languages, time frames and sometimes radically divergent operational modes. Often what seems intuitively straightforward, reasonable, and doable at a business level is remarkably difficult or even impossible to achieve in a smooth and orderly way on the technical level.

This gap between perceived feasibility at the business level and the granite-rock lack of feasibility (or even controllability) at the technical level is a major element that all software platforms must address.

4.1 When Design Fails to Support Strategy

The problems caused by inadequate software design are familiar to everyone: applications are difficult to change, break often, and in many cases do not perform as desired. If deficiencies do not exist right now, then they will exist very soon in the future because environmental changes always outpace the ability of the software to adapt to change.

Traditional design techniques fail to take into account the requirements of today's business needs. Organizations often construct the designs for use with outdated technology as well. For example, engineers base traditional software design on use cases – but use cases can only describe "known" situations. What happens when the playing field changes?

Theoretically, designers develop information systems to meet business needs. Nevertheless, it often happens that software imperfections constrain business processes – as incredible as it sounds. Technological weaknesses should not compromise strategy. We can reverse that trend.

The following are danger signs that may reveal flawed design:

- "That's the way we've always done it"
- "That button is supposed to do X but it's never really worked"
- "I know it doesn't make sense but you have to do it that way"
- "This part of the process is still done manually"

4.2 Recognizing the Need for Change

Software is inherently inflexible; once you have completed an application, it is often as difficult to change as if it were carved in stone. Only human intelligence, based on experience, can recognize when change is necessary. When outdated systems are standing in the way of progress, it is up to enterprise leaders to set a new direction and do the required work.

4.3 Leveraging Performance Capabilities

In difficult times, it is important to keep in mind the company's strengths as well as its weaknesses. These strengths invariably come down to one thing: good people. Information systems exist – or should exist – to make it easier for people to do their jobs and to facilitate efficiency. Good people come up with good ideas, and the role of technology is to implement those ideas.

In our experience, there are always opportunities for improvement – usually in the form of greater efficiency, technological advancements, or improved metrics.

5 Data Presentation Layer

In this section, we present the strategic perspective on three data presentation components:

- User Interface
- Interaction Design
- Reporting

5.1 User Interface

The user interface encompasses everything that goes on at the workstation at the screen level, among other tasks, the following two major functions:

- Validation
- Lookup

I'm assuming there is more to say here at some point

5.2 Interaction Design

Interaction design governs the way people interact with machines. It is a critical part of the strategic perspective -- good interaction design supports the business function by providing tools that employees can use efficiently and simply. As computers replace increasing numbers of tasks that people used to perform manually or with mechanical machines, the need for good user interaction design becomes ever more important. It is an essential part ensuring that designs meet the business needs any software applications.

5.2.1 Mechanical vs. Computer Interfaces

Consider the difference between the way we interact with mechanical machines and the way we use today's computers. Interaction with machines takes place physically and directly. In contrast, interaction with computers relies on a level of abstraction that takes place in the user interface. For example, in most homes, you control the room temperature by changing the setting on a mechanical thermostat dial. When you perform the same function using a temperature-control computer console, you must interact with the abstraction of the user interface menu, which may or may not mimic the original dial.

This evolution is taking place in the products we *and* in the services we consume. Contrast the experience of listening to your car radio (manually changing dials and volume settings) vs. listening to music through your cable TV menu system, which requires that you interact with the abstraction of the user interface menu.

To interact with the user interface, you must learn a set of steps to follow and understand the function of various options on the menu. The inputs and results are less obvious, making mastery of the task more complex than the original mechanical or manual system.

5.2.2 Addressing User Needs

To address this added complexity, designers must create user interfaces that are obvious and clear. It is no longer sufficient to focus exclusively on the product's function. Applications must also adhere to the principles of interaction design, emphasizing the quality of interaction between people and computers to improve the user experience.

The principles of interaction design arise from the application of cognitive psychology (including the ideas of mental models, interface metaphors and affordances) to the techniques of traditional design. This unique project-based approach to development applies several methods for describing and testing the usability of interface interaction, through iterative usability testing and evaluation.

5.2.3 The Benefit

Designs that emphasize the quality of the user experience embody the principles of the strategic perspective at the most visible level of the design – the user interface. Applications that incorporate good interaction design connect users, products, services, and functions in ways that meet the business need, while at the same time respecting user needs and providing a positive experience to the people who interact with the applications on a daily basis.

5.3 Reporting

We need something here that ties the strategic perspective to reporting, or perhaps just saying how it differs from traditional approach.

Simply put, reporting describes the outputs that the application provides. Much of the value of computer support comes from the user's options for viewing, sorting, or printing data. Reporting provides the mechanism for applying an application system's data resources to a given business environment.

In most organizations, paper reports continue to be highly useful and critical to business functions. Moving completely to paperless operations is not feasible. Organizations often find that they can reduce paper utilization, but not eliminate it.

Data in concrete form has very useful properties – you can move it around without elaborate electronic capabilities. Users can physically file it and organize it. They can scribble notes on the paper and refer back to them later. Of course, you could build similar facilities in software, but in terms of immediate applicability, you cannot beat paper.

However, the software can do things that are cost-prohibitive or essentially impossible with paper/manual systems, such as the following tasks:

- User selection of data to display
- Show data in any table
- Show data according to relationships

A report writer is an essential requirement of the software design that should support selection, grouping, and extraction functions. In addition, it must allow you to extract data subsets into tables that you can save.

5.4 Presentation Ordering

Users of an information system can rarely forecast which sort orders are important for viewing information. Keeping in mind the principles of strategic design, it is important to build in flexibility that can accommodate these changing requirements.

For example, managers may what to see data in count order to determine which customer has the greatest number (or the least number) of counts.

fname	lname	counts
John	Zeeman	2
Louise	Martin	5
Mary	Smith	11
Jerry	Lee	15

However, the next day however, managers might need to see customers ordered by last name in order to determine which match customers to salespeople.

fname	lname	counts
Jerry	Lee	15
Louise	Martin	5
Mary	Smith	11
John	Zeeman	2

6 Business Logic Layer

The business logic layer encompasses the business rules that determine what transactions the system handles and how that processing takes place. This layer of the architecture defines the following application characteristics:

- Portability
- Rules of behavior
- Workflow

It is in this area where most design problems arise, because this is where you implement your strategy. The business logic layer is also the starting point when converting business processes from manual to automated systems. Typically, designers create the business logic first, then the data access layer, and finally the data presentation layer. Because of its key role in the application design and its role in implementing strategy, it is in the business logic layer that the strategic perspective is most critical.

6.1 Automating a manual process

Business logic for automated processes often differs slightly from manual logic. In most cases, software makes the process easier, but badly designed software can create difficulties, and in some cases, it can introduce "bugs" into a process that worked well before automation.

In a pre-automated context, most companies have a good understanding of the manual process they seek to automate. Translating the process into software terms, however, takes time and planning. Often, designers are bogged down in working out the steps for individual tasks while they lose sight of the bigger picture.

Limited assumptions can cause many design flaws. For example, assumptions such as "This data will only need to be viewed one way" or "We don't need to collect that information." can cause problems. The resulting design is inflexible.

6.2 Improving an automated process

The process of modifying an existing software application is slightly different from automating a manual process from the beginning. Users can clearly see the steps involved in a manual process. In contrast, software code obscures the steps

involved in an automated process. Sometimes the resulting software is more inflexible than the manual process it automates.

6.3 Transactions

Software systems generally contain some sort of persistent collection of data, for example, in an order entry system, the database would maintain information about orders placed and the customers who placed them. The application's business logic must allow you to intereact with the database to enter transactions. In an order entry system, these transactions would include be a purchases or returns of an items.

Although different transaction types can occur on the same day, transactions are almost invariably "point-in-time" events. If these transactions represent money, they may represent debits or credits In these cases, the application must allow you to develop both sums and other calculated results from the transaction.

From a system that tracks property values, you might want to be able to calcuate the following:

- Days, dollars, visits, counts
- Property ranges
- Quantitative vs. discrete property value

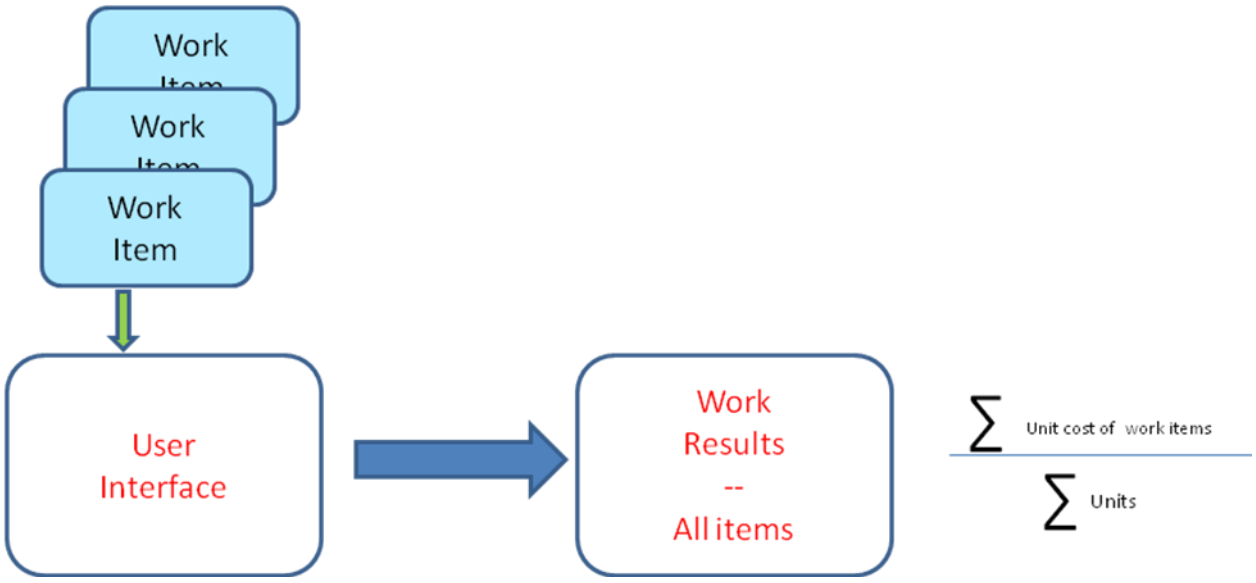
From a system that tracks work done for clients, you might want to be able to calculate various sums based on the following:

- Product
- Customer
- Customer type
- Work item
- Department

6.4 WorkFlow

There are many competent workflow management systems available. It makes sense to use them when there are complex work flow decisions, multiple rules, and a large number of participants. Economics (system life cycle costs versus investment) will determine when it makes most sense to use these vendor-supplied products.

Almost every application has some form of workflow however, and it doesn't always make sense to utilize available commercial products (either because of the magnitude of the investment, or ongoing support costs). It still is important though to have the basic workflow separate from the application program logic.



7 Data Access Layer

The Data Access Layer provides access to data stored in a repository of some kind, such as a relational database. It provides access to other program modules that can manipulate the data without the need to deal with the underlying complexity of the storage device.

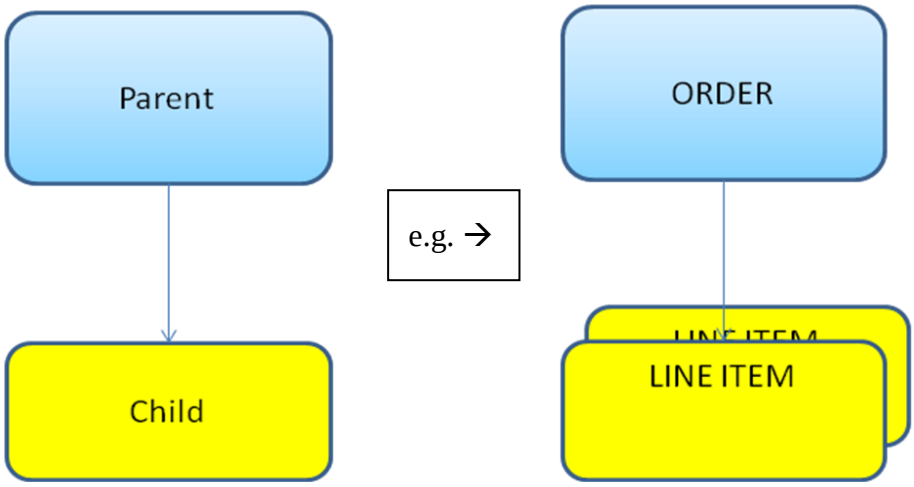
7.1 Design Patterns

Application systems models often have design elements that occur repeatedly, even in vastly dissimilar applications. Understanding these design patterns can lead to speedier, more effective development processes. By capitalizing on previous designs and relationships, it is possible to create working applications with much less effort than starting from scratch every time.

You can find the following design elements throughout many applications:

- Parent-child
- Parent selection based on child characteristics
- Tree
- Synchronized datasets across several forms
- Validation
- Lookup

Consider the parent-child design pattern in an order entry application. In this example, every order is an instance of the parent, each of which can have one or more children (associated line items).



Some designs are extremely useful, and can help guide workflow in an almost effortless way. For example, in the parent-child structure, it is very useful to have functional capabilities that allow you to isolate certain parents with children exhibiting particular characteristics.

In such an application, you might want to find all of the orders with associated "back order" line items or select out line items that reference a particular part that may have a problem.

7.2 Search Capability

Searching the database to isolate files and records of special interest is one of the most important operations for the vast majority of business applications

Unfortunately, most of these searches are “hardwired”. They apply only to a small number of cases and only a few specific data fields. When the needs of the business change, requiring a different set of fields, it could take quite some time to implement the required changes.

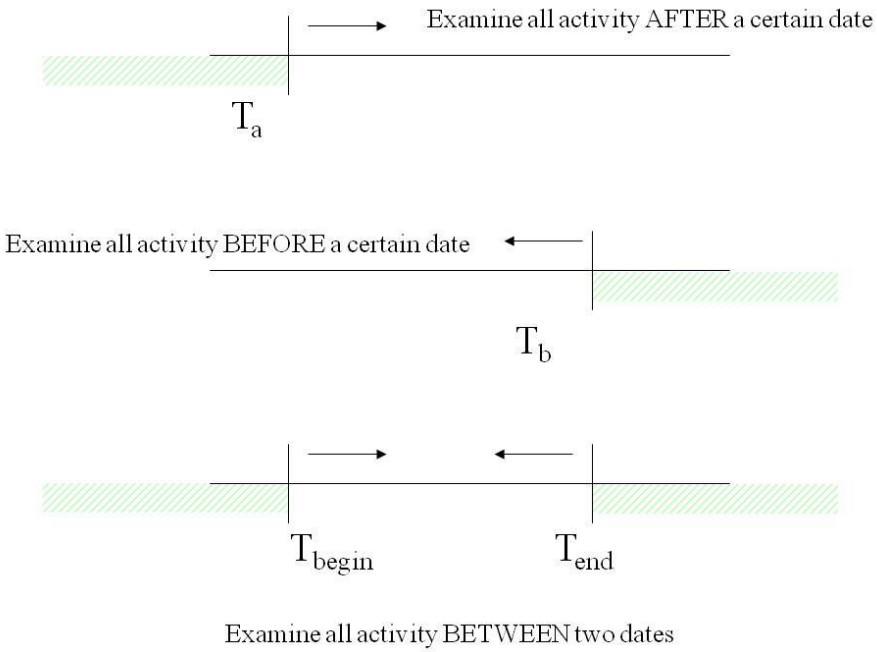
A design that is independent of the fields (including the number of fields) can provide lightening quick responsiveness when search criteria must change.

7.3 Database Integration

Modern businesses rely on multiple applications, and those application systems may or may not have straightforward mechanisms for integrating their operation.

Integrating these separate applications requires design methods that can span multiple databases, and multiple requirements. These techniques are not simple, but well worth the effort to produce a data environment where independent applications begin to work together.

For example, the systems must be able to manage one-time occurrences *and* continuing trends. To accomplish these goals, application software must be able to manage timeframes including discrete dates, intervals, and ranges, as shown in the following diagram.



Therefore, the software must allow you to perform the following type of date selections:

- before a date
- after a date
- between a date
- last week
- last month
- last year

8 Managing for Profitability – Case Studies

As we have discussed throughout this document, profitability results from good design guided by the principles of the strategic perspective. In other words, designers must emphasize the overall business need (profitability) while addressing the functional requirements of each system (payroll, inventory, client management, etc.).

"Managing for profitability" means *identifying and controlling performance shortfalls that reduce sales revenue or drive up costs*. Just as you must automate manual processes as the number of transactions, clients, employees, etc. grows, you must also automate the process of recognizing and managing unfavorable events and conditions that impinge on profitability as the complexity of your organization grows.

You can capitalize on opportunities to manage for profitability by incorporating techniques for handling revenue impairment, both on the cost side and the revenue side, into every design. The strategic goal of profitability must take its place among the functional design goals as you create and maintain systems. It becomes part of the functional specification for all applications throughout the organization.

In this section, we provide practical real-world examples of how companies can manage for profitability.

8.1 Case Study 1: Big Eastern Telecom

Big Eastern Telecom is a regional telecommunications and internet services provider in the metropolitan New York area. Management had determined that labor costs were out of line with revenue and expected industry-performance levels. They hired us to determine where and how they could trim back these costs.

The Challenge

Big Eastern Telecom was experiencing the following problems:

- difficulty completing customer orders in a timely fashion
- a high incidence of rework *(what does that mean?)*

They asked Management Strategies to examine basic fulfillment processes and measure how well they were meeting their goals. Armed with this analysis, we were then able to make adjustments to improve quality and performance.

The Process

Much of the required information resided in our client's TBS *(what does this stand for?)* sales order system. It also resided in the technical systems that recorded, tracked, and implemented changes to their network infrastructure and installed equipment.

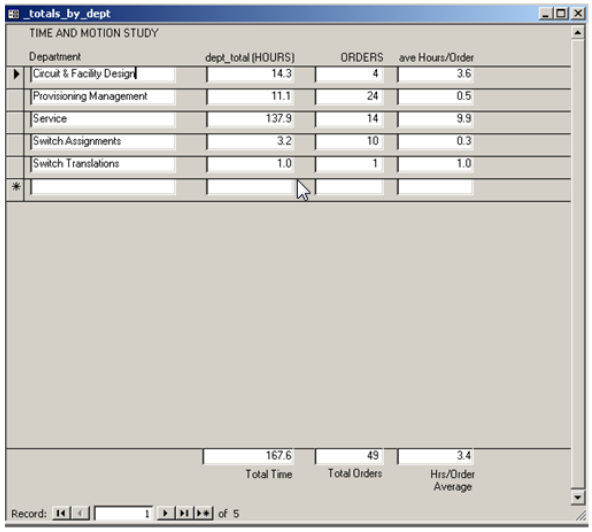
Because of the complexity of the organization, it was difficult to derive the performance metrics required to uncover system problems. More importantly, the complex fulfillment process comprised many interdependent steps. Thus, a delay in one process step (say, in network engineering, or network turn up) could have cascading and non-linearly increasing effects on downstream steps.

To get the information we needed, we harvested data from automated systems and augmented that information with actual "stop-watch" observations. An analyst monitored every order-processing step for a specific order end-to-end through the fulfillment process, recording and tracking what happened to it along the way.

We further spent time to "sample" continuous stretches of activity throughout every part of the process, generating enough information to form statistically valid data, This data allowed us to draw conclusions about exactly where, how, and why the process consumed resources.

The results, shown in the following figure pointed the way toward remarkable improvements.

Preliminary Rework Analysis



- 49 orders out of 472 were flagged as REWORK items.
- 168 hours out of a total of 972 hours were associated to rework items
- The majority of the rework hours measured were in the Service Department -- usually with equivalent time in Engineering for re-design.
- We may not be capturing all elements of rework in the other departments. The actual amount of rework is at least 17%.
- Eliminating rework would boost productivity by at least 17%.

We uncovered far more rework related activity (what constitutes a rework?) than met the eye in a first cursory examination. The analysis of the automated system provided the information we needed to make very specific recommendations for reallocating workers and talent and allowing Big Eastern Telecom to meet its profitability goals.

The Solution

By implementing our recommendations to incorporate simple guards against just one of their problem areas (rework), Big Eastern Telecom realized productivity gains of 17%, all of which flowed directly to the bottom line. By applying the principles of the strategic perspective to the functional design, we were able to address the business need of improved profitability through design changes.

8.2 Case Study 2 – Managing Customers for Profit

In "Managing Customers for Profit", V. Kumar studies the effect of retaining unprofitable customers on a firm. [1] This section explores how the results of such a study can provide management with the information it needs to make decisions

[1] V. Kumar, *Managing Customers for Profit: Strategies to Increase Profits and Build Loyalty*(Wharton School Publishing, 2008).

about customer management, which designers can then integrate into the functional design to achieve business goals.

The Challenge

A general merchandise catalog retailer wanted to determine the effect of retaining unprofitable customers on the firm. For these purposes, a customer is considered unprofitable when the when the customer's consumption of company resources exceeds its contribution to revenue.

The Process

To study the effect of retaining unprofitable customers, researchers segmented customers into two groups based on their level of loyalty and profitability. As you can clearly see in Figure 2.5, both segments of customers start out to be profitable, and they generate comparable profits until month 20. After that, customer 1 maintains its current profitability level, whereas customer 2 becomes less and less profitable.

By employing a forward-looking metric, the company can assess the behavior of its customers future profitability. The company may base its metrics on the assumption that customers will continue their past purchasing behavior and that employees will continue to allocate an equal amount of resources to both customers. Armed with this information, the company can decide when a customer is no longer worth pursuing.

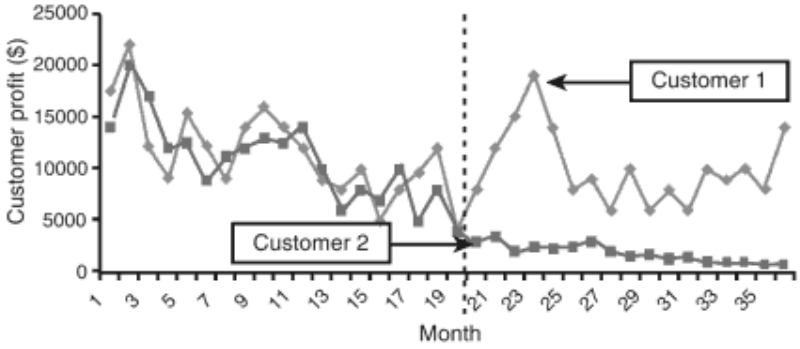


Figure 2.5 When to lose a customer: a case study

The Solution

By adding the business logic that evaluates customer profitability to the functional design, management has the information it needs to make decisions about customer management. For example, one response might be the classical activity-based cost-management approach that advocates firing customers who are absorbing more resources than others are without producing corresponding revenue.

However, with metrics in place to evaluate customer profitability, management now has the opportunity to consider the complexity of customer management and devise ways to use customer information to meet other business needs. For example, management may want to take into account the following customer-specific issues that might affect decisions pertaining to customer retention:

- Customers have different quality requirements, and different tolerances for errors and issues.
- One customer may complain about receiving marginal results (because they are incorporating your deliverable in what they are selling to their own meticulous customers).
- Still, another customer may take the same deliverable and be perfectly happy with what you have given them.

If you simply let customers go when they tie up too much engineering, sales, and maintenance resources, you may miss valuable feedback that you need to improve your business functions. In fact, you may want to pay special attention to those more costly customers because some of them may be identifying particular weakness in your delivery process.

If you fire unprofitable customers too quickly, you may never address fundamental problems in your profitability because you eliminate the inputs that drive the error processes. The analysis that discovers profitable and unprofitable customers provides the basis for the application of the company's customer-management strategy.