A RESEARCH REPORT

PRESENTED TO THE

GRADUATE SCHOOL OF BUSINESS LEADERSHIP

UNIVERSITY OF SOUTH AFRICA

In Partial Fulfilment of the

Requirements for the

MASTERS DEGREE IN BUSINESS LEADERSHIP

UNIVERSITY OF SOUTH AFRICA

By

SHAILENDRA NATHOO

STUDENT NUMBER: 0699 33 11

6 December 2002

BEST PRACTICES

FOR

BUILDING BUSINESS

COMPONENTS

Acknowledgements

This research report is dedicated to the following companies and people:

- ***Gartner Group*** *for assistance in compiling the research*
- ***Software Futures*** *in giving me the insight to the research,*
- ***Hennie Visser*** who has assisted with the supervision; and
- ***Jagruti and Prajay Nathoo.***

I certify that, except as noted above, the report is my own work and all references used are accuretely reported.

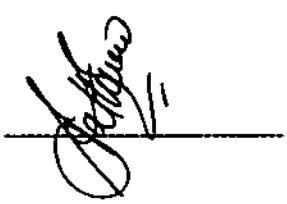

Table of Contents

1. INTRODUCTION 6

1.1. KEY DRIVERS FOR BUSINESS MODEL CHANGE 7
Globalisation 7
Virtualisation 9
Transparency 10
Time to Market 11
Increased Productivity 11
The 'Weightless' Economy 11
Consolidation 11
1.2. THE INTERNET: IMPACTING EVERY BUSINESS 12
1.3. SOFTWARE: THE KEY DIFFERENTIATOR FOR BUSINESS IN THE NEW ECONOMY 13
1.4. PROBLEM STATEMENT 14
1.5. IMPORTANCE OF THE PROBLEM 18
Examples from other Industries 20
A New Solution: Development based on Software Components 21
1.6. OBJECTIVES OF RESEARCH 25
1.7. LITERATURE REVIEW 26
Origins and Domain of CBD 26
An Architectural Approach 27
Benefits of CBD 28

2. BENEFITS OF DEVELOPING BUSINESS OBJECTS 30

2.1. OBJECT ORIENTATION – A PRIMER 30
2.2. "TRADITIONAL" VERSUS OBJECT-ORIENTATED SOFTWARE DEVELOPMENT 31
2.3. DEFINITIONS 33
What is an Object? 33
Instances and Classes 35
Methods 35
Encapsulation 35
Polymorphism 36
Class Hierarchies and Inheritance 37
2.4. ANALYSIS AND DESIGN WITH OBJECTS 38
2.5. OBJECT-ORIENTATED PROGRAMMING LANGUAGES 40
2.6. UNIFIED MODELLING LANGUAGE (UML) 41
UML – Its Business Benefit 43

UML – Basic Definitions 43
2.7. Benefits and Risk – Object Technology 51
2.8. Benefits and Risk – UML 53
3. COMPONENTISATION – A BEST PRACTICE 54
3.1. Introduction 54
3.2. What are Software Components? 56
3.3. The Reason for Software Components 59
3.4. Components as the Service Provider 63
3.5. Types of Components 64
4. GOING OVER THE WATERFALL 66
4.1. Introduction 68
4.2. Six Winning Strategic Imperatives 69
Develop Software Iteratively 69
Manage Requirements 69
Use Component-Based Architectures 70
Visually Model your Application 71
A Continuous Circle of Quality 71
Change Control 71
4.3. The Iterative Development Process 72
4.4. The Rational Unified Process® 75
4.5. RUP® A Process Perspective 78
4.6. Risk Mitigation 88
5. THE CHANGE CYCLE 89
5.1. Introduction 89
5.2. An Architectural Approach For Organisational Change 89
5.3. Best Practices For Implementing Organisational Change 92
5.4. A Team Perspective 95
5.5. A Skills Perspective 97
5.6. The Suitability of Migrating Legacy I.T. Staff 98
6. CASE STUDIES 102
6.1. Deloitte Consulting (2001) 102
Description 102
Business Problem 102
Business Solution 103
Key Benefits 103
6.2. ARINC Incorporated (2000) 104
Description 104
Business Problem 104
Business Solution 105
Key Benefits 106

6.3. AU-SYSTEM (1999) 106
Description 106
Business Problem 106
Business Solution 107
Key Benefits 108
6.4. VOLVO INFORMATION TECHNOLOGY (2000) 109
Description 109
Business Problem 109
Business Solution 110
Key Benefits 111
6.5. FORD FINANCIAL 112
Description 112
Business Problem 112
Business Solution 113
Key Benefits 114
7. CONCLUSION **116**
7.1. OVERCOMING RESISTANCE 117
7.2. BUSINESS AND I.T. ALIGNMENT 118
8. DEFINITIONS **120**
9. REFERENCES **122**

List of Figures

Figure 1: The Net-Liberated Organisation – A Philosophy for Post E-Business Survival [7] 6
Figure 2: Globalisation [7] 8
Figure 3: Virtualisation [7] 9
Figure 4: Transparency [7] 10
Figure 5: Growth of the Weightless Economy [US Bureau of Economic Analysis] 12
Figure 6: The Software Development Paradox [13] 17
Figure 7: A Schematic of Technology Development – 1950 to 2000+ [3] 18
Figure 8: Use of Components by the Automotive Industry 21
Figure 9: Software Components are Like High-Tech "Legos" 22
Figure 10: A Definition of Objects 30
Figure 11: The Difference in Object-Orientation 33
Figure 12: Encapsulation 36
Figure 13: Polymorphism 37
Figure 14: Inheritance 38
Figure 15: An Example of an Object-Orientated System 39
Figure 16: Use Case Diagram Example for Withdrawal of Funds 46
Figure 17: A Class of Objects 47
Figure 18: Two Classes with an Association 47
Figure 19: A Sequence Diagram Notation 48
Figure 20: State Chart showing State Transitions 49
Figure 21: Impact of Components [16] 54
Figure 22: Basic Component Concepts [12] 57
Figure 23: Applications in the New Millennium [16] 60
Figure 24: Components and Interfaces 63
Figure 25: Types of Components [16] 65
Figure 26: The Software Paradox [13] 68
Figure 27: An Iterative Approach to Software Development 69
Figure 28: The Waterfall versus the Iterative Approach 72
Figure 29: The Iterative RUP ® Model [13] 78
Figure 30: The Phases and Major Milestones in the Process [13] 79
Figure 31: Lifecycle Objective [13] 80
Figure 32: Lifecycle Architecture [13] 82
Figure 33: Initial Operational Capability [13] 83
Figure 34: Product Release [13] 85
Figure 35: Risk Reduction Profile for Waterfall and Iterative Development [16] 86

Figure 36: Volvo I.T. 's Implementation Approach [13] 110

List of Tables

Table 1: Software Project Performance Evolution 16
Table 2: Changes in Software Development Costs [3] 19
Table 3: The Relevance of CBD to the Agile Business [8] 23
Table 4: A Comparison of Major Object-Orientated Languages [7] 40
Table 5: Characteristics of Components 58
Table 6: Prime Business Objectives for Applications 61
Table 7: Prime I.T. Objectives for Applications 62
Table 8: An Example of Risk Mitigation 87
Table 9: Organisational Assessment of Readiness for a Change Initiative 91
Table 10: Migration Suitability Matrix [7] 100
Table 11: Acceptance Threshold Matrix [7] 101

1. INTRODUCTION

The 1990s were characterised by a corporate obsession with cost efficiency. This translated into an information technology (I.T.) mantra of standardisation, simplification and integration, and led to the rapid adoption of enterprise resource planning systems.

The Internet has brought about a more open, immediate and global market, leading to the emergence of a new economy. Organisations must free themselves from the constraints of yesterday's business models and transform themselves into more agile, efficient customer-focused enterprises, in order to uniquely differentiate themselves from their competition. Gartner defines this as the Net-Liberated Organisation. Leading enterprises are being forced to adopt an externally focused business strategy that centres on the customer to drive value generation.

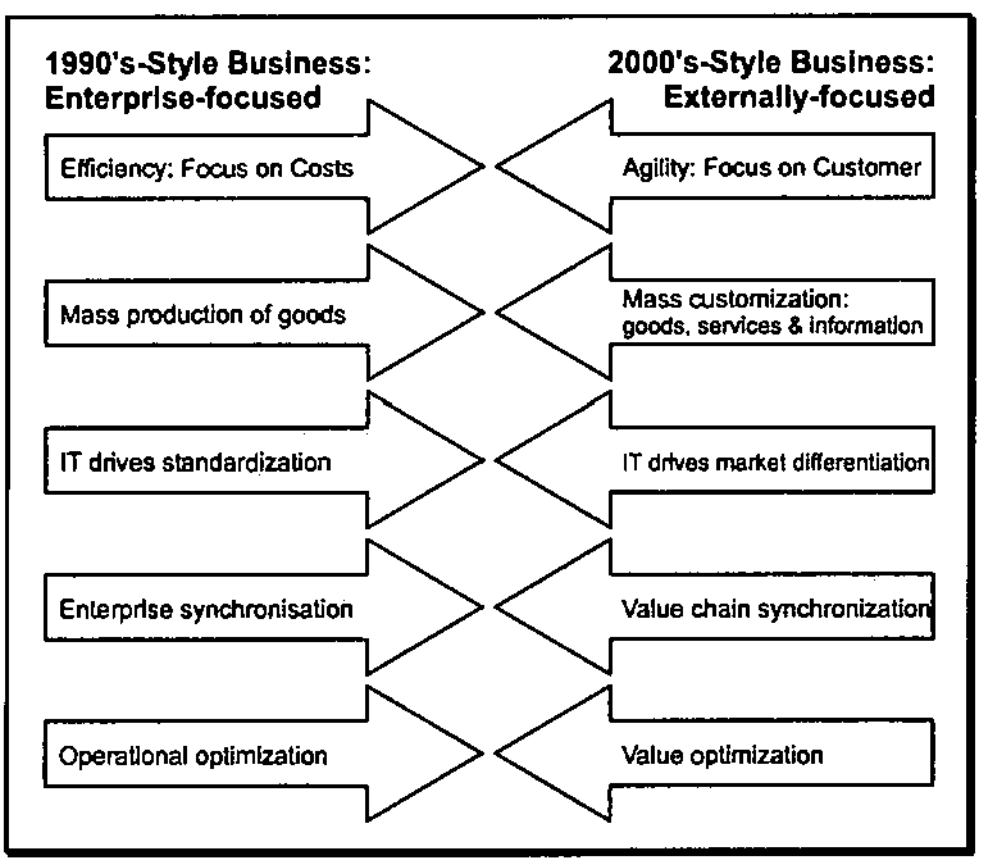

Figure 1: The Net-Liberated Organisation – A Philosophy for Post E-Business Survival [7]

To respond to the challenges for Post-E-Business Survival, organisations must:

- Allow the enterprise to decompose and recompose itself on a value-orientated basis, i.e. enhance their value chain;
- Make value creation the benchmark for all organizational decisions;
- Reduce the power of organizational hierarchies;
- Remove all source of friction to undertaking business;
- Ensure that a zero-latency information environment is available; and
- Outsource business activities, including critical elements that can be better provided by business partners that have superior economics of operation.

Such actions will enable enterprises to liberate themselves from the constraints of yesterday's business environment and its focus on local and physical infrastructures. The net liberated enterprise will transform itself to be more agile with the ability to meet customer demands more efficiently.

1.1. KEY DRIVERS FOR BUSINESS MODEL CHANGE

GLOBALISATION

At the beginning of the 20th century it took more than three months to ship goods from the USA to Europe. International telegram services were prohibitively expensive and the alternative was surface mail. The speed of communication of both goods and information has been revolutionised, and yet many business models still clearly exhibit signs of their 19th century origins. The shrinking of distances makes opportunities for the creation of global forces, which have never previously been possible.

Markets are global: competition is global: capital is global; and Information and Communication Technology (ICT) is a potent force in the development of globalisation.

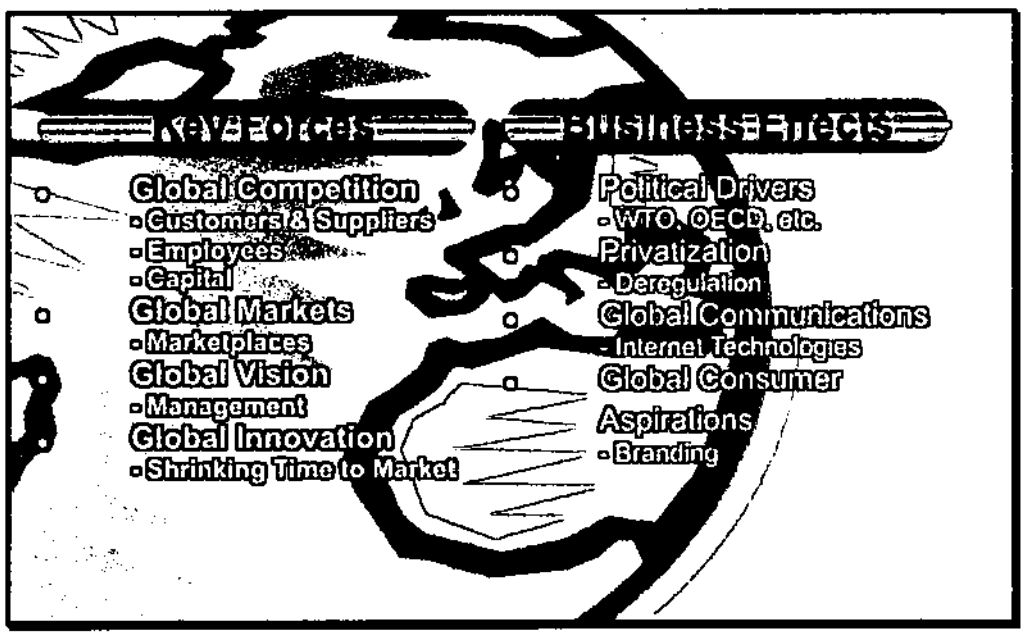

Figure 2: Globalisation [7]

Globalisation is not just working at the economic level, but at the social, political and cultural levels as well. The enterprise need to consider the way in which globalisation shapes their markets and stakeholders.

Now the opportunity is for organisations to become the biggest and the best, not only by expanding beyond the country of origin, but also by merging or partnering with companies overseas.

VIRTUALISATION

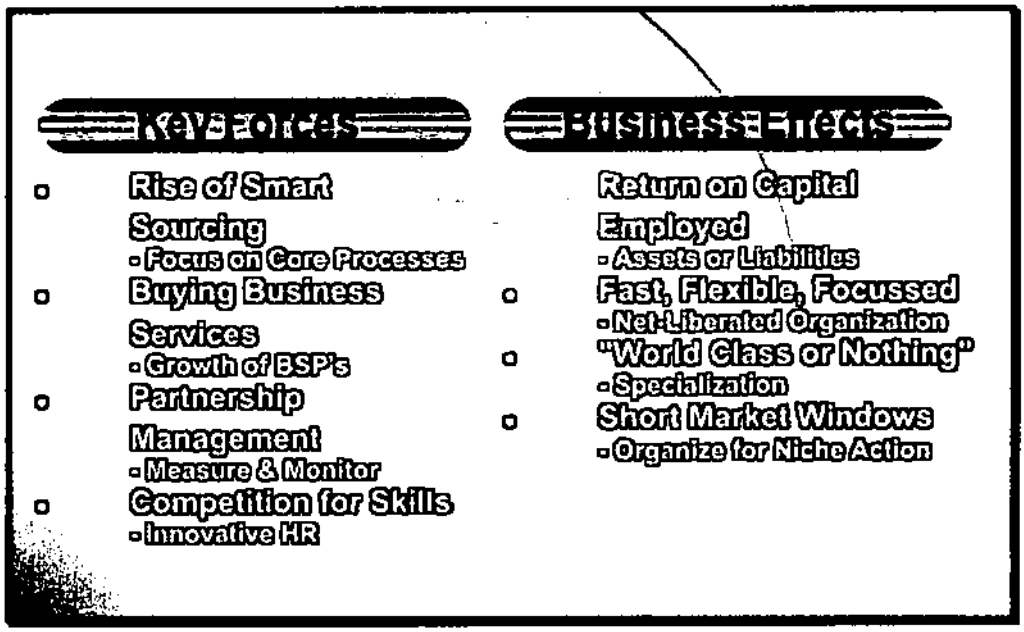

Figure 3: Virtualisation [7]

As global forces shape the macro-economic competitive environment, the shape of the organisation is also being radically changed. In order to be competitive in global markets, enterprises are obliged to strip out all those processes, which are not completely core activities. Organisations should seek world-class sourcing partners to deliver these non-core services. Whilst many of these Business Service Providers (BSPs) will make extensive use of ICT to deliver their services, the underlying trend to virtualization is driven as much by the need to improve fundamental business measures such as the Return on Capital Employed (ROCE), by utilising the capabilities of emerging technologies.

TRANSPARENCY

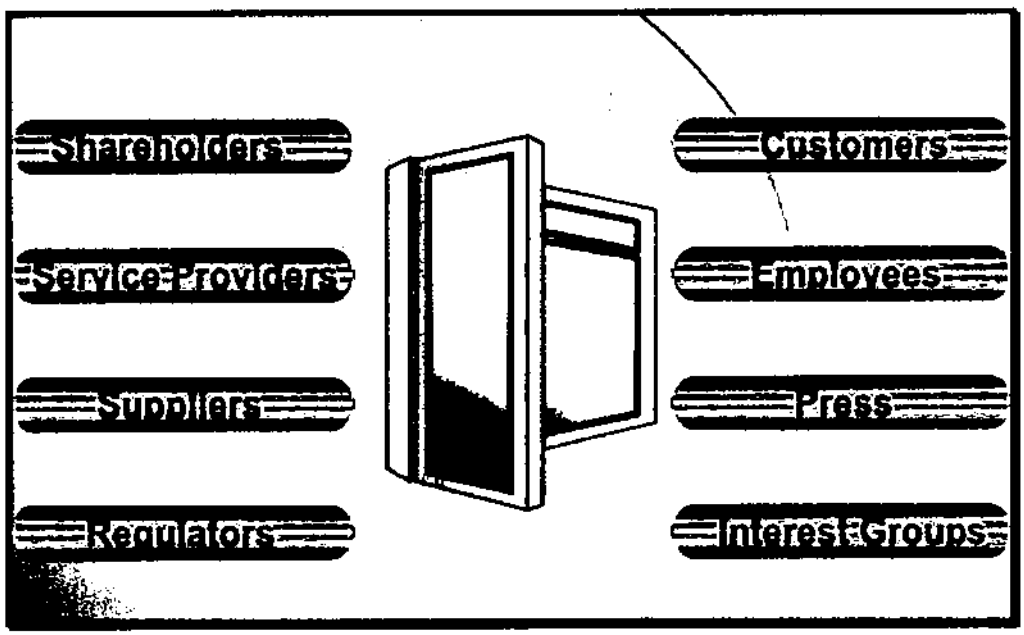

Figure 4: Transparency [7]

There was a time when, what happened inside an organisation was strictly private. The management team could run the concern as their personal fiefdom, and stakeholders were entitled to very little information. The very solid walls of the organisation have started to crumble, but there is still a long way to go. In the networked economy, enterprises will become increasingly transparent to all their stakeholders.

They will be transparent to their:

- **Shareholders**, driven by the twin forces of regulatory pressure for disclosure and competition for capital, where the capital markets will become increasingly intolerant of closed-door management;
- **Customers**, who will expect zero-latency access to current performance data and to planned performance;
- Strategic sourcing **partners**, encouraging them to identify new ways they can add value; and

- **Society** at large, as a representation of a multitude of special interest groups, all of who will wish to ascertain that the highest standards of behaviour are achieved.

TIME TO MARKET

Both new and existing companies are being pushed to develop new products more quickly than ever before or risk losing to a competitor that has taken the maximum advantage of the new environment.

INCREASED PRODUCTIVITY

Both the private and corporate sector has been one of the major factors contributing to growth in the new economy, and this growth has occurred without inflation. The US Federal Reserve has estimated that the use of information technology and the production of computers accounted for about two-thirds of the productivity growth between the first and second halves of the 1990s.

THE 'WEIGHTLESS' ECONOMY

We are witnessing the increased valuation of intellectual property, which is displacing oil, gas and other types of durable or physical goods from the old economy as the primary means of making money.

CONSOLIDATION

Global mergers and acquisitions continue to redefine virtually every industry, especially telecommunications.

Software has become the way we engage the world, and software is ubiquitous. It is everywhere, from the cars we drive, to the military equipment, to the cell phones that are allowing us more freedom and mobility. Software even exists in household appliances like refrigerators, and in tiny devices like the cochlear ear implant for the hearing impaired.

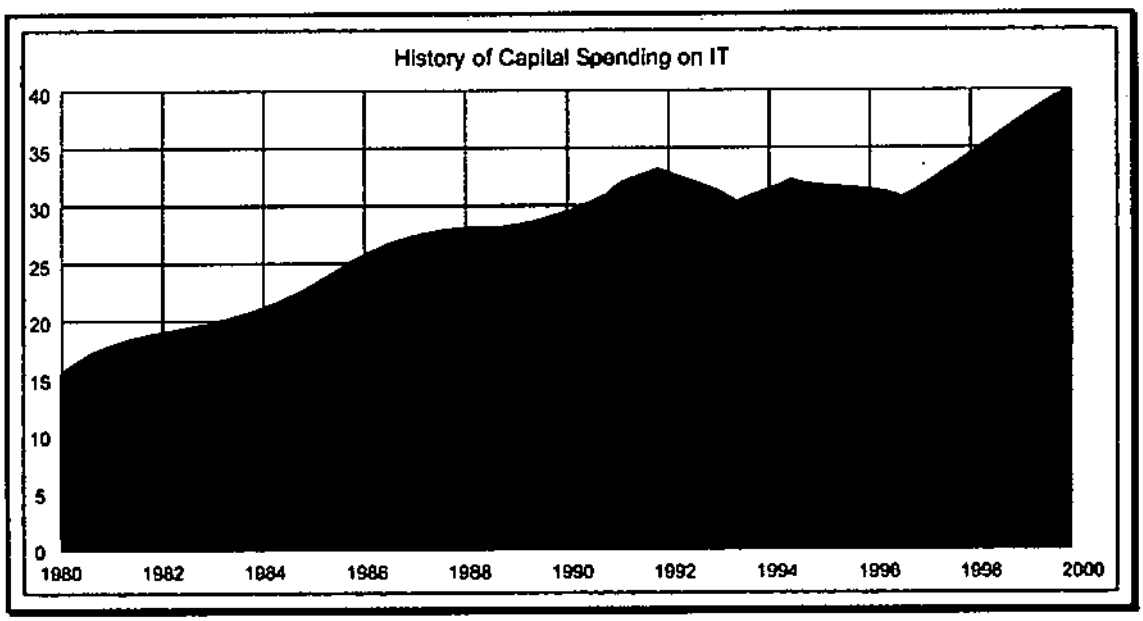

Figure 5: Growth of the Weightless Economy [US Bureau of Economic Analysis]

The graph shows corporate spending as a percentage of total capital spending growing from 18% in 1980 to 40% in 2000.

1.2. The Internet: Impacting Every Business

As software is becoming more present in more places, so too is the Internet impacting every business, as it continues to drive more change and more opportunity. This includes all the software that runs the infrastructure of the modern economy, the computing and

telecommunications platforms, all the operating systems and middleware, and all the underlying switching and routing software. This will also include our electronic devices and embedded software systems, and e-business whether it is done by traditional brick-and-mortar firms that are embracing new ways of developing relationships with customers or by dot-coms that exist only online.

The role of ICT is not simply to support existing administrative processes but to create competitive advantage through the innovative use of new I.T. and related technology. Effective and efficient use of I.T. is a significant factor in the success or failure of businesses in all sectors and industries.

Object oriented development has been around for over 20 years and was among the first programming paradigms to acknowledge that software must continually change and evolve throughout its lifetime. The technology improves the development process by making applications more powerful and easier to use, and minimizes the impact of change by combining behaviour and its associated data into a single package – the object.

1.3. Software: The Key Differentiator for Business in the New Economy

Companies are changing the way they think about technology and how they use it, so too are organisations changing the way they buy the software they need. In the 1980s, software was the domain of a centralised technology group that typically made all the purchasing decisions and controlled the back-office systems used to automate business processes.

Through the 1990s, departments and systems were becoming more integrated through distributed architectures and corporate intranets. More corporate data became visible to more people, and this led to a more robust front-office.

Today, in the new economy, we are observing a much more collaborative environment for doing business. To stay competitive, every business must embrace the technologies of its customers, its supply chain, and its partners, and software is used as the 'glue' for these connections. Since software is becoming more of a strategic investment, senior management of the organisation makes more and more decisions regarding these investments.

If organisations intand to thrive over the next decade, they will have to utilise software in a creative way to deliver products and services or delight their customers in new ways. Mere software competence is not enough, as software excellence is becoming essential.

1.4. Problem Statement

An object can be defined as a piece of software that consists of data (attributes) and the operations (methods) that work with the data. Each object has a name, and each has a method. The attributes and methods define what it means to be a particular type of object and how those objects behave.

Component technology is a method of creating applications and is finally gaining momentum and a wide spectrum of acceptance. It promised to speed up the time it takes to create software and also to make software products more reliable.

"The PC market really exploded when components came in, when you could buy memory and motherboards and put together a computer in any configuration you wanted."

"Component technology is about to create the same type of explosion in software productivity" Joseph Damassa, Vice President of IBM Marketing, Software Division.

One area that has not been blessed with such alacrity is software development. Programs or any major internet application takes several months to develop, and is often prohibitively expensive.

The problem, component technology hopes to solve is a core software development issue:

- An application's logic, its business rules, the database, and presentation layer are all inextricably mixed up in a monolithic application. A company typically cannot make a change in any part of the program without affecting all others.

This approach worked a couple of years ago when change was predictable and slow.

It does not work today in the new-networked economy, as companies realize that speed is not a luxury, but the difference between survival and extinction.

	1960s-1970s	1980s-1990s	2000+
Complexity		30% Components 70% Custom	70% Components 30% Custom
Process	Ad-hoc	Repeatable	Managed and Measured
Team	Predominantly Untrained	A Mix of Trained and Untrained	Predominantly Trained
Tools	Proprietary Not Integrated	Mix of Proprietary and Commercial Not Integrated	Commercial Integrated
Project Performance	Predictable Always *Over budget* *Over schedule*	Unpredictable Infrequently *On budget* *On schedule*	Predictable Frequently *On budget* *On schedule*

Table 1: Software Project Performance Evolution

The problem is that speed and quality have typically been opposing forces in software development, and they still are. In the past decade, businesses could sacrifice software quality to make their shipping date, or compromise software features to meet time-to-market deadlines. In the new economy, we have no choice and we must produce higher quality software in Internet time.

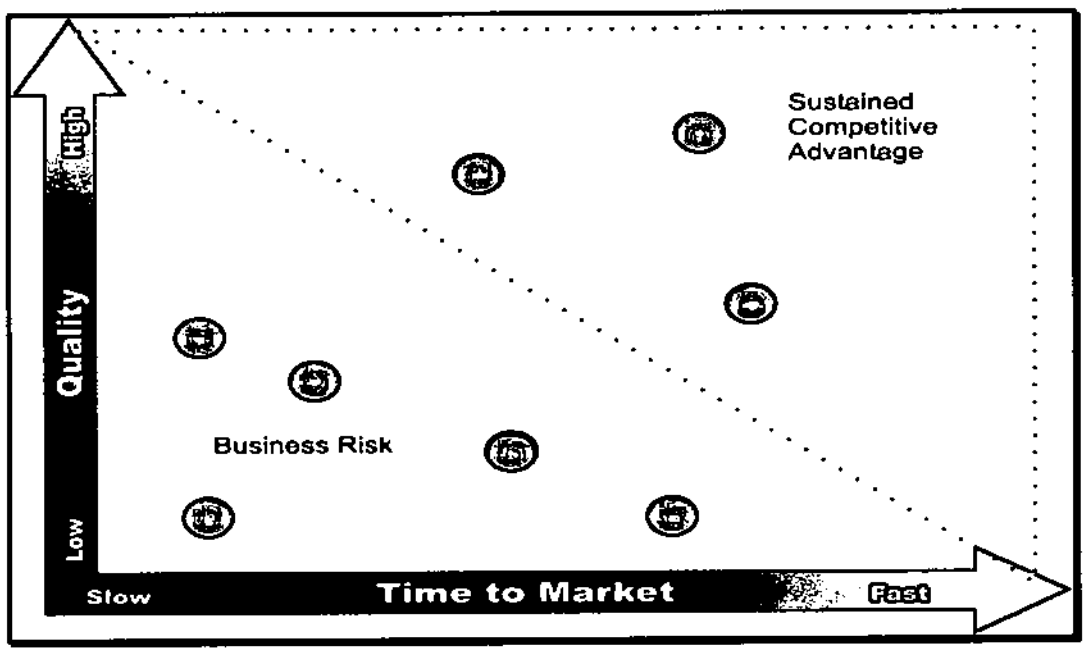

Figure 6: The Software Development Paradox [13]

This conundrum forms the heart of what we call the 'software development paradox'. The fast time-to-market is absolutely critical for business success. The competitive features that once took years for the market to adopt now have become absolute 'checkbox' requirements for consumers. Businesses respond with their own product plans, or risk losing market share. At the same time, a poorly engineered product that gains wide visibility simply because it captures 'first mover advantage' will ultimately fail, due to low customer satisfaction. Products rushed into market will typically not live up to the expectations in the vast Marketspace of the Internet.

On the other hand, if we can get business to a point where we are not obligated to trade off speed for quality, we can enjoy a sustained, competitive advantage in our respective markets. Business success will depend on how well we execute in the process of software development. We need speed **AND** quality in the software development process, in order to evade the 'software development paradox'.

An agile organisation may reduce complexity by:

- Managing the scope of the project,
- Reducing the size of human-generated code by utilising higher level software components to enable economically significant reuse, and
- Raising the level of abstraction by providing a visual model of the software design to promote unambiguous communication with the business level.

1.5. IMPORTANCE OF THE PROBLEM

Figure 7 illustrates the basic flow of the computer hardware and software technology evolution since the 1950s, when computers began to come into general use.

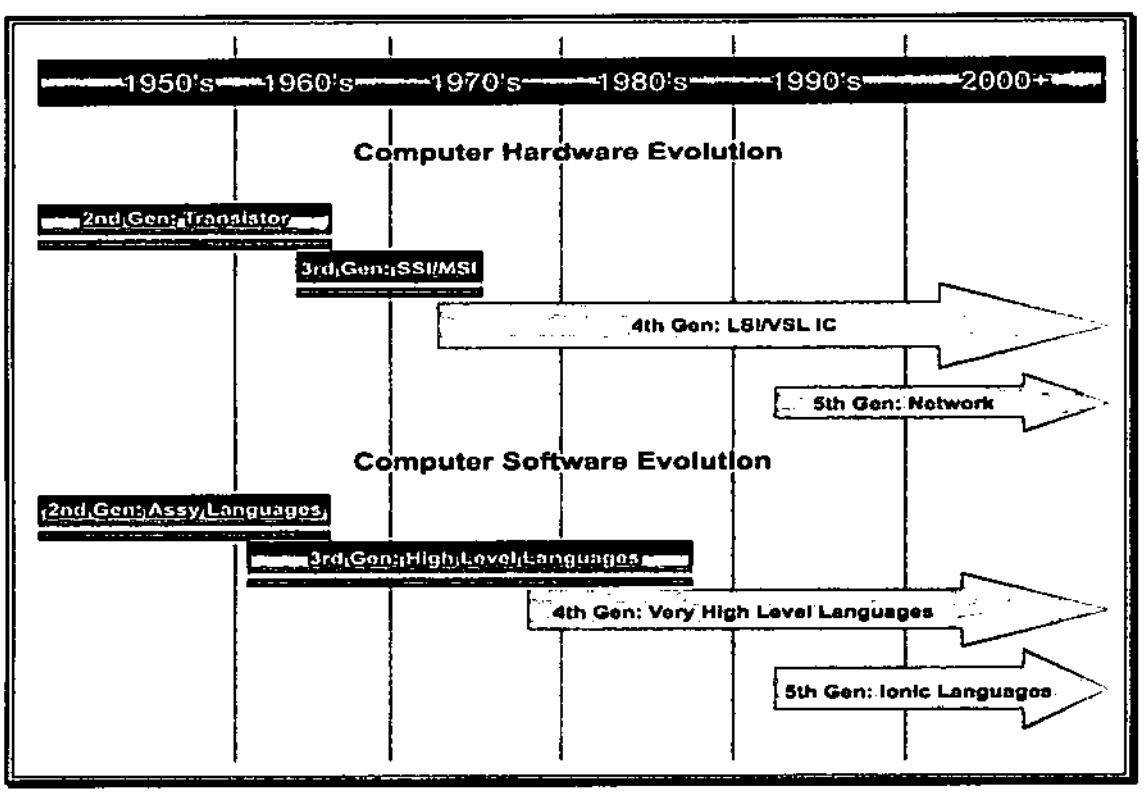

Figure 7: A Schematic of Technology Development – 1950 to 2000+ [3]

The modern age of computers started in the mid-1980's with the advent of the advanced mainframe computers and the emergence of the desktop/networked computing environments, coupled with the emergence of the premier Fourth Generation language, COBOL and its successor languages.

Since 1985, there has been a startling advance in both software development time to market and labour reductions, and these advances have been relatively modest.

Steve McConnell [1] used a metric called the Function Point to compare programmer productivity across dissimilar languages.

VERY APPROXIMATE TIME FRAMES	1985 COBOL	1990 C++	1995 VISUAL AGE, SMALL TALK	2000 JAVA
5000 Function Point (FP) Source lines of code	455k	265k	105k	90k
Number of software developers	34	32.2	11	9
Nominal schedule (calendar months)	28	22	15	15
Hourly cost of software developers	$20 to $75/hr	$40 to $100/hr	$40 to $150/hr	$45 to $165/hr
Typical LOW project estimated total cost ($ millions)	$3.0	$3.0	$1.0	$1.0
Typical HIGH project estimated total cost ($ millions)	$11.6	$7.7	$4.0	$3.7
LOW 1985 REAL $ Cost	**$3.0 (baseline)**	**$4.0**	**$1.7**	**$2.0**
HIGH 1985 REAL $ Cost	**$11.6 (baseline)**	**$10.3**	**$6.8**	**$7.3**

Table 2: Changes in Software Development Costs [3]

A five-year block of time was used to benchmark software technologies during the period 1985 to 2000, where a comparison of the development cycle times and costs were prepared. In 1985;

- A 455k source line program was reduced to 90k lines in year 2000,
- While the typical number of programmers required reduced from 34 to 9; and
- The typical schedule went from 28 weeks to 15 weeks.

This constituted a 4:1 reduction in staffing needs, but only a 2:1 reduction in development cycle time.

Since 1985 there have been waves of methods introduced to attack both development cycle time and development labour/costs. A few of these include:

- Structured programming
- Object-orientated programming
- Top-down design methodologies
- Improved project management methods
- Unified modelling language (UML)
- Requirements management.

EXAMPLES FROM OTHER INDUSTRIES

How do mature industries deal with the development of complex systems? The automotive industry develops very complex, precision devices called cars, and the part counts run into tens of thousands, with countless variations that seem to share only four wheels in common. The

problems include being in a high-competitive, somewhat regulated industry that needs to produce new products annually or more often.

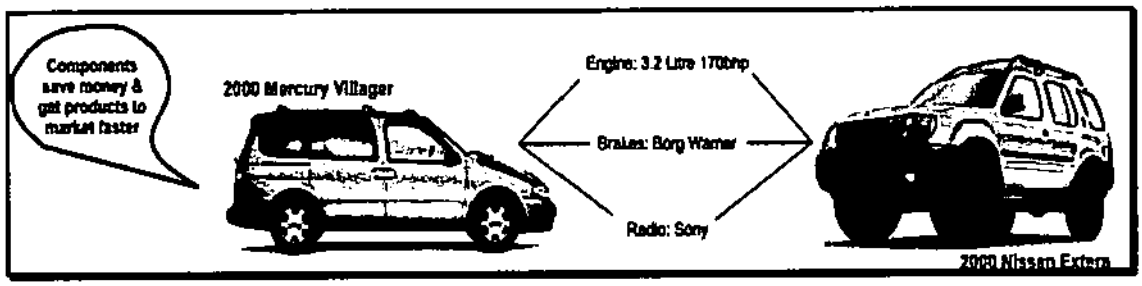

Figure 8: Use of Components by the Automotive Industry

The modern automotive manufacturer has become more of a system integrator than manufacturer as the automotive companies have learned the economic rule of Comparative Advantage and long ago quit making their radios. Figure 8 illustrates how Nissan produces two apparently different cars from virtually identical major subsystem components.

The logic of system component use by the automotive industry is germane to the high-tech industry, and serves in many ways as a straightforward example of how to use components in software. The Information Technology industry makes effective use of components by a rigorous set of standards that define interoperability.

A New Solution: Development based on Software Components

Component based development (CBD) is an inevitable next-wave solution that has the potential to improve time to market and resource/cost trends that have been ongoing.

Software components can be compared to Lego's as shown in Figure 9 below. The diagram depicts how components 'fit' onto an operating system or framework in two hypothetical applications, with native components (written to interface directly with the framework or other components) ' snapping' into place, while those components that are not native requiring an

adapter component to provide interoperability. Again, like the automotive car, interoperability is the key.

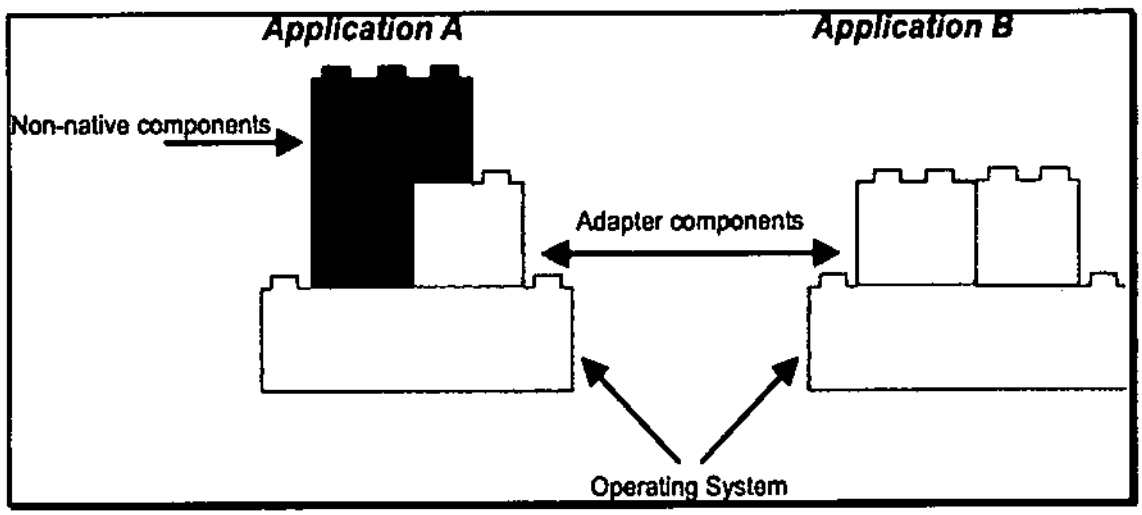

Figure 9: Software Components are Like High-Tech "Legos"

The underlying software is needed to realize effective e-business, which must transcend technology barriers and provide business capabilities that have greater flexibility in meeting changing business needs or requirements. In Table 3, we depict the business factors that will induce the technology challenges for the agile organisation.

FACTOR	AGILE BUSINESS CHALLENGE
Time to market	There are rapidly shrinking windows of opportunity.
Business fit	There is a need to reflect, integrate and rethink business processes.
Quality	Consequences of software errors are often catastrophic for e-business solutions.
Cost savings	There is a need to reduce costs, be competitive and exploit niche markets.
Adaptability	Business can change overnight.
Scalability	There are fluctuating and demanding response times across vast numbers of users.
Consistency	There is a need for a single organisational view presented to the global village.
Integration	Pressure increase to purchase treadmill software and levarage of existing I.T. investments.

Table 3: The Relevance of CBD to the Agile Business [8]

A software development process is no longer appropriate in the globalised economy, and it is now a process of evolution and integration. The new process must encompass e-business planning, component architecture and legacy integration.

In the new networked economy, it is clear that the e-business phenomenon is a powerful force emanating from the business side of the enterprise. Executives seem to understand that things will be a lot different in the future and that they had better be prepared. The I.T. function of the organisation is asking what changes they should make to their infrastructures and architectural directions.

The challenge is to include the need for greater speed in operations, more extensive business process outsourcing, and even greater sharing of enterprise knowledge. The resultant of this challenge is for the I.T. function to open up their business logic to external sources, give internal

and external users easy access to internal information, transactions and knowledge, and enable independent software vendors (ISV) to provide their application via portal access.

All of these changes share common characteristics or needs:

- High-speed (zero latency) immediate access to new information
- Quick conversion for ease of sharing information across multiple boundaries
- High-speed, high-capacity connectability
- Flexibility to set-up and change rapidly, and
- Adaptability to respond to new demands,

The issue is almost never portability and nearly always interoperability.

Software is the world's most critical industry and will be for years to come. Yet most companies are spectacularly unprepared to create the software that will redefine how they interact with customers or that will help deliver goods and services in new ways.

As we look at the drivers of the 'new economy', we see why software is so critical to the success of companies around the world, it is clear that organisations need to closely examine and most cases, dramatically change how they create the software that will serve the heart of their business. Many companies have already learnt the hard way that the Internet is not a quick fix, nor is it a trivial matter to build a Web site that works well for both the customer and the business.

Over the past year, the media have pointed to the demise of certain dot-com companies as if their disappearance indicates a trend regarding the importance of high-tech in business. In fact, many dot-coms have disappeared because bad business models were unable to sustain the promises made to their boards and shareholders. Does this mean that technology is no longer

important? Not at all, as technology still drives the new economy. The difference is the stakeholders in the new economy have remembered how important profitability is. Underneath all the former hype lies the enabling power of the Internet and the software and communications technologies that drive it, and all of which constitute a more dramatic and fundamental transformation of business than anything we've ever seen. The rate of change is just as rapid as it was last year, and the direction is unequivocal.

The cover story of *Fortune* magazine's dated October 9, 2000 issue notes that we're living through one of the most profound transformations in human society since the dawn of the Iron Age, and calls this the 'Age of the Network'.

1.6. OBJECTIVES OF RESEARCH

In the new-networked economy, it is clear that the e-business phenomenon is a powerful force emanating from the business side of the enterprise. Executives in an organisation seem to understand that things will be a lot different in the future, and that they better be prepared for it. The I.T. function of the organisation is asking what changes they should make to their infrastructures and architectural directions.

The challenge is to include the need for greater speed in operations, more business process outsourcing, and even greater sharing of organisation knowledge. The resultant of this challenge is for the I.T. function to open up their business logic to external sources, give internal and external users easy access to internal information, transactions and knowledge, and enable independent software vendors (ISV) to provide their application via portal access.

All of these changes share common characteristics or needs, which are the objectives of this research:

- High-speed (zero latency) immediate access to new information;
- Quick conversion for ease of sharing information across multiple boundaries;
- High-speed, high-capacity connectability;
- Flexibility to setup and change rapidly; and
- Adaptability to respond to new demands.

The issue is almost never portability and nearly always interoperability.

1.7. LITERATURE REVIEW

ORIGINS AND DOMAIN OF CBD

The business world is being profoundly transformed by the internet and the World Wide Web. The zeal with which customers have embraced the web and begun to buy products over the Internet completely alters the economics involved in selling and distributing goods and services. The web has also provided the foundation on which best breed retail and service companies are beginning to evolve. In a similar way, technology and the internet has allowed us to integrate both internal legacy applications and the applications of multiple companies, which promises us a new era of productivity that will reduce the costs of goods and services, and promote a worldwide swell in economic growth.

E-business applications require profound changes in the way companies are organised. They depend on the major changes in core company business processes and they require a whole new approach to software development (Allen, 2001). E-Business applications will integrate legacy applications to make their resources available to customers throughout the world, and this type of integration will require the use of software components.

These software components will not only aid the integration issues, but also guarantee that the company's resources will be available to the World Wide Web.

In Realizing e-Business with Components [12] the author agrees that the common approach to software development is a lack of planning and analysis, resulting in inflexible solutions that are unable to integrate or change at the speed of light. Most organisations jump on the e-business bandwagon, without fully understanding what they are getting into. Allen [12] provides a practical approach to planning, analysing and the delivery of e-business systems using component-based development (CBD). He defines CBD as the approach to problem solving using software building blocks.

A software development approach where all aspects and phases of the development lifecycle, including requirement analysis, architecture, design, construction, testing, deployment, the supporting technical infrastructure, and project management are based on components. (Herzum and Sims, 2000).

AN ARCHITECTURAL APPROACH

E-business fundamentally changes the traditional approach to business strategy, which rests on the assumption that IT exists to support business. E-Business improvement planning is an incremental and continuous process that recognises that I.T. is now an integral component to

the success of business strategy (Allen, 2001). Hence this implies that the business has to deliver a clear-cut business case that balances the strategy with the delivery.

"For those of us working in I.T. today, the key change is that CEOs will be driving this transition, CEOs don't know about technology, but they will be demanding results. If I.T. groups aren't careful, they will agree to undertake development efforts without first putting new architectures, new infrastructures and new component development processes in place. If they do, they will probably doom their companies to obsolescence during the course of the next decade." (Harmon, August 1999)

Business requires a way to balance strategic issues with fast results and timely software delivery, requiring I.T. to raise the bar and be more proactive. The generations of software technology to the 4th generation languages (4GLs), have fallen short of raising the bar and enabling the shift in mindset.

Component technology development (CBD) provides real opportunities to adapt to the agile organisation and its continuous improvement in business processes demanded by the client on the World Wide Web.

BENEFITS OF CBD

The component based development approach addresses three pervasive problems with traditional systems development:

- Quality,
- Productivity, and
- Flexibility.

Technology solutions (applications) developed with the traditional approach have been notoriously error-prone, expensive and inflexible. The component based approach has the potential to reduce errors, reduce costs, and increase flexibility because of its inherent features.

Firstly, each component in a system is relatively small, self-contained, and manageable – which reduces the complexity of systems development and can lead to higher quality systems that are less expensive to build and maintain. Secondly, once a component is defined, implemented and tested, it can be reused in other systems. The reusability of components will greatly increase productivity, resulting in improved quality as the reused components are proven products. Lastly, systems can be modified or enhanced very easily, by changing some of the components or by adding new types of components, because components are self-contained units that can be changed or replaced without interfering with the core of the application.

These potential benefits are the driving force behind CBD.

2. BENEFITS OF DEVELOPING BUSINESS OBJECTS

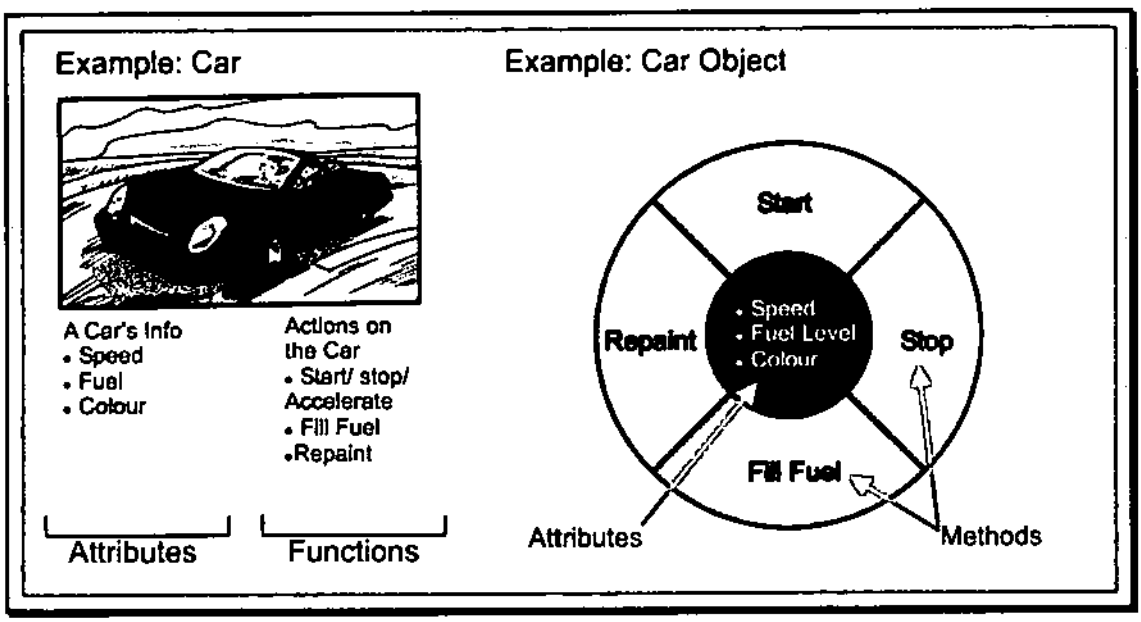

Figure 10: A Definition of Objects

2.1. OBJECT ORIENTATION – A PRIMER

Object orientation development has been around for over 20 years. It has been the first programming paradigm to acknowledge that software must continuously change and evolve throughout its lifetime.

In the early 1970s, software had advanced to the point where an ad hoc development process was no longer feasible. The field of software engineering emerged to bring advanced planning and external controls to the process and succeeded, to some extent, in organizing chaotic, time consuming endeavor. However, software engineering only provided an interim solution. By the 1980s, programs and applications had become so complex, that they required vast amounts of

resources and programmer time to during the development, debugging and implementation phases. The combination of an integrated enterprise-wide application system and the popularity of the Internet have introduced more complexity to the systems development lifecycle.

To facilitate the delivery of consistent, scalable software within budget and time constraints, the I.T. function of the enterprise have been turning to the object-orientated paradigm, in the hope of solving these problems.

This permanent change to developing software applications, utilizing object orientation will:

- Improve the development process by making applications more powerful and much easier to use;
- Minimize the impact of change by combining behaviour and its associated data into a single package – the object;
- Reduce the amount of time and effort necessary to produce an application;
- Encourage reuse of existing software code; and
- Make such reuse practical.

2.2. "TRADITIONAL" VERSUS OBJECT-ORIENTATED SOFTWARE DEVELOPMENT

Object technology has been publicized as a revolution in software development because it turns the conventional view of systems design upside down. In the "traditional "scenario, software

and systems development starts with a problem, which is then decomposed into a numerous sub-problems. Each of these sub-problems may be further broken down until a level is reached at which code can be developed.

An example would be a private client portfolio application which is broken down into smaller problems like developing a user interface, a command processor, a client loading interface, and so on. This approach is known as the 'top down' and is well suited to centralized, hierarchical computing environments. A traditional, structured design technique will include entity-relationship diagrams, where an application is seen as a collection of real-world entities that interact with each other, and the data-flow diagrams will treat the application as a way to move data through the system.

Object-orientated development is 'bottom up'. Since the world appears to be composed of objects, software systems would be closer to reality if they, too, were to be built up from objects. A system will solve particular problems or address particular tasks by creating logical objects corresponding to the elements of a given scenario. A private client portfolio application will comprise of the following few basic objects – clients, accounts, portfolio values and recurring raports, and the application is created by specifying communication paths and inheritance hierarchies.

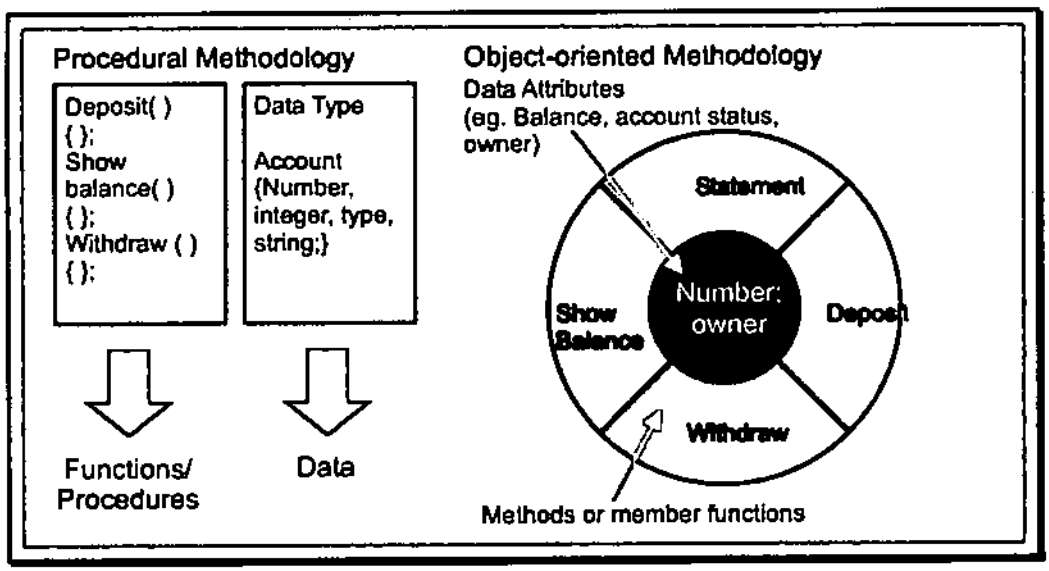

Figure 11: The Difference in Object-Orientation

2.3. DEFINITIONS

WHAT IS AN OBJECT?

An object is a piece of software that consists of data (attributes) and the operations (methods) that work with the data. Each object has a name, and each method has a name. The attributes and methods define what it means to be a particular type of object and how that object behaves.

James Martin's definition:

"From a very early age, we form concepts. Each concept is a particular idea or understanding we have about our world. These concepts allow us to make sense of and reason about the things in our world. These things to which our concepts apply are called objects." [14]

Coleman defines an object as:

"An object is a thing that can be distinctly identified. At the appropriate level of abstraction almost anything can be considered to be an object. Thus a specific person, organisation, machine or event can be regarded as an object." [15]

Software objects are actually modeled after real-world objects. An example would be a portfolio account object might contain a set of data (client details, cash balance, current holdings, capital gains tax on the current holdings) and the methods that operate on that data (calculate current holdings, subtract cash balance, deduct management fee, calculate brokerage and calculate net gain on current holdings).

The types of objects in a computer system may be classified as **user interface objects**, **operating environment objects** and **task-related objects**.

- **User Interface Objects**

  These are objects that physically appear on the computer screen, and end-users directly interact with them. All these objects have attributes, they exhibit behaviours, they interact with each other, and most importantly, we as users interact with them.

- **Operating Environment Objects**

  These are objects that exist somewhere in the computer network or is controlled by the operating system of the computer. In a client-server architecture, the client and the server are regarded as objects, as the server-object will provide services for other objects (print services and database services), and the client-object will request services from the server-object.

- **Task-Related Objects**

  These objects actually complete the work. They include document objects and multimedia objects, and sometimes referred to as problem domain objects. Their tasks, in most instances, are involved in information processing in the application.

INSTANCES AND CLASSES

These objects actually complete the work. They include document objects and multimedia objects, and sometimes referred to as problem domain objects. Their tasks, in most instances, are involved in information processing in the application.

METHODS

A method may be defined as the behaviour or the responsibility of the object, i.e. defined as a procedure. There are two typas of methods – standard methods and custom methods. Standard methods include knowing how to add a new object, knowing how to show information about an object, knowing how to delete object, and knowing how to change the values of the attributes of an object. Custom methods are custom designed for a specific class of objects.

ENCAPSULATION

Encapsulation is the process of combining data and functions. This process dictates that an object can identify its methods and data, and not allowing other objects to manipulate its state without its knowledge.

The benefit of encapsulation allows:

- Changes without causing a ripple effect, and
- Developers to 'divide and conquer', when coding.

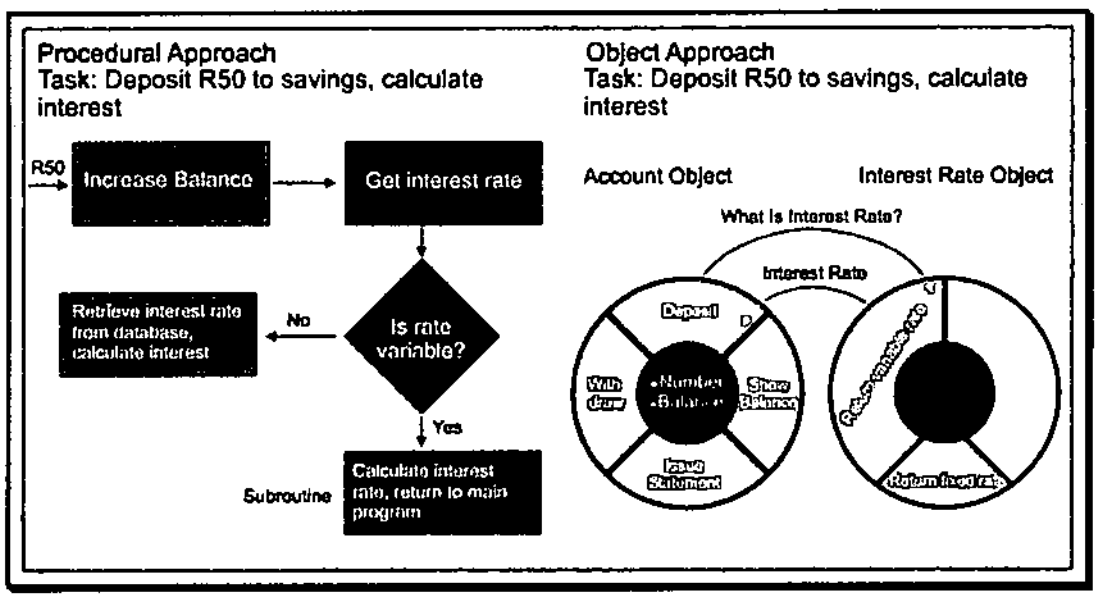

Figure 12: Encapsulation

POLYMORPHISM

It is possible to use the same name to denote the different methods for different objects, as they are defined independently from each other. The method *add* may have a different definition to a word processor object (bring new text), a spreadsheet object (aggregate the contents of specific cells), or a contact manager object (include a new name and address to a database of clients). Using the same name to denote different procedures is called *polymorphism*.

The benefit of polymorphism is that it separates the operations from method names, allowing 'divide and conquer'.

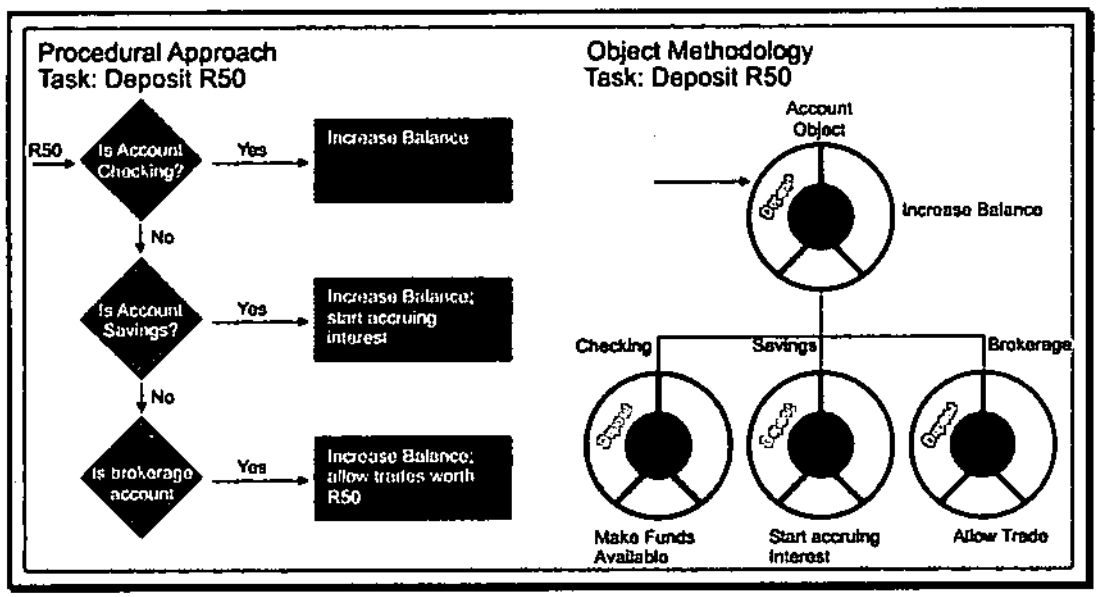

Figure 13: Polymorphism

CLASS HIERARCHIES AND INHERITANCE

Classes are related to one another in a hierarchy. An example would be a portfolio object as the single system object. The second level of the hierarchy may consist of bank account class, which contains information on the variety of accounts offered by the investment bank. The third level of would identify the specific types of accounts, such as savings accounts, mortgage loans, cheque account and credit card account. At a lower level is the class of the specific account – an example being savings account 1234 78927.

This hierarchy implements the concept of *inheritance*. In an object-orientated system, classes at the lower levels of the hierarchy will inherit attributes and methods from the parent classes above them.

The benefit of inheritance is that it:

- Reduces redundant coding, and
- Simplifies maintenance.

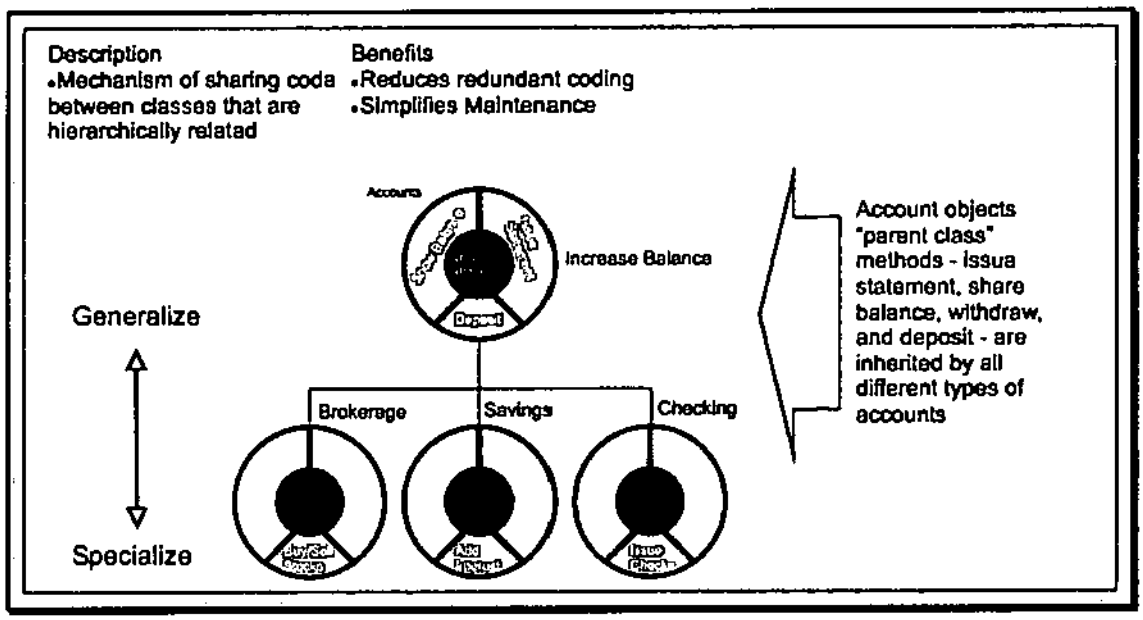

Figure 14: Inheritance

2.4. Analysis and Design with Objects

Structured and information engineering methodologies are still popular, but many organisations are turning towards object-oriented analysis and design where their business processes and core systems require frequent change. A core feature of object-oriented analysis and design is that it represents reality better than the traditional data flow, entity-relationship, or state transition design approaches.

Critical to object-oriented analysis and design is viewing applications as modelling an organisation's business processes. During the analysis phase, these business processes are identified and operationalised as objects and communication flows. The translation into an object-oriented design is straightforward, and creating the program consists of developing (coding) the objects and gluing them together.

As there are multiple methodologies available for traditional software system analysis and design, the same holds true for object-oriented analysis and design. The primary methodologies are:

- **OMT** from James Rumbaugh. This methodology is based primarily on structured analysis and design.
- **Booch Method**. An analysis and design technique developed by Grady Booch specifically for C++, and now being revised JAVA.
- **Objectory**. Developed by Ivar Jacobsen which models a system as a series of scenarios that shows all the avenues a user will interact with such a software system.
- **Rational Unified Process (RUP)**. A technique that combines Booch, OMT and Objectory and supports Unified Modelling Language (UML).

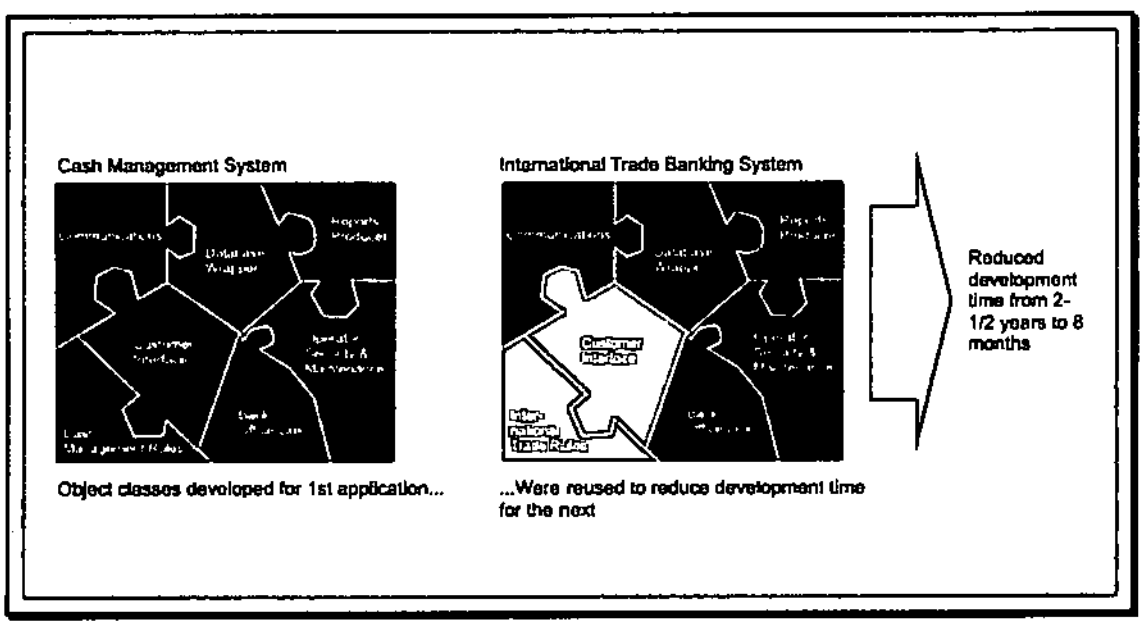

Figure 15: An Example of an Object-Orientated System

2.5. OBJECT-ORIENTATED PROGRAMMING LANGUAGES

	Java	C++	Smalltalk	Object Cobol	Eiffel
Object-Oriented Language	Pure	Hybrid, extension to C	Pure	Hybrid, extension to COBOL	Flexible
Binding Time (Early or Late)	Either	Either	Late	Either	Complies to C
Classes	Yes	Yes	Yes	Yes	Yes
Encapsulation	Yes	Yes	Yes	Yes	Yes
Component Model	EJB and JavaBeans, CORBA Components	Microsoft COM, Active X, CORBA Components	CORBA Components	No	No
Inheritance	Yes	Yes, except for Microsoft COM	Yes	Yes	Yes
Multiple Inheritance	No	Yes	No	No	Yes
Class Libraries	Yes	Yes	Yes	Yes	Yes
Type Checking	Both	Static	Dynamic	Dynamic	Static
Polymorphism	Yes	Virtual Function	Yes	Yes	Redefinition
Partially Defined Class	No	No	No	No	Deferred Class
Memory Management	Automatic	Programmer	Automatic	Automatic	Automatic
Garbage Collection	Yes	Beginning to appear	Yes	Yes	Yes
Commercial Availability	Yes	Yes	Yes	Yes	Yes

Table 4: A Comparison of Major Object-Orientated Languages [7]

The market leaders at this time are Java, which is a pure object-oriented language.

2.6. UNIFIED MODELLING LANGUAGE (UML)

Unified modeling language (UML) is the latest trend in methodology and design circles. The idea of an oriented design and development standard began during the "method wars" of the late 1980s and early 1990s, when dozens of software development methodologies, each with its own set of artifacts, vied for attention. Since no method provided all of the features required for full-cycle development, a new group of 'second generation' methodologies appeared in the mid 1990s.

Three of this generation's methodologists, Grady, Booch, Rumbaugh and Jacobsen embarked on complementing each other's ideas as part of their own methods. In 1995, Rational Software hired Rumbaugh, Grady and Booch and they proposed a unified method that would combine the best features of their methodologies.

At that time, each methodology had its own modeling language, making it very difficult for users and organisations to transfer skills or reuse existing models. As work moved forward at Rational Software, the trio (Rumbaugh, Grady and Booch) developed a single modeling notation, and not a single methodology. UML 0.8 of 1995 became the common language for models and artifacts.

In July 1997, the Object Management Group (OMG) adopted UML 1.0 as the language standard. In June 1998, the standard was UML 1.3, and the OMG is now evaluating proposals for the long-awaited UML 2.0.

UML is a language for specifying, constructing, visualizing and documenting the artifacts of a software-intensive system. UML has significant potential to ease the application development, maintenance, and even integration in today's Web-enabled, composite software world.

UML provides a standard set of graphical and descriptive artifacts, which works with any design methodology or any software development technology, as it is an aid to collaborative development.

Since the adoption of UML as a standard, application development has undergone a paradigm shift, as much of today's software development does not focus on creating entirely new systems from scratch, but instead on developing composite systems consisting of both old and new code, sometimes in different languages, and bringing it out over the Web. These systems require effective integration of disparate applications and at least some degree of component-based development, thereby complicating development, deployment and maintenance.

Many systems must also run on different operating platforms and multiple tiers, so that user or platform interfaces, program or business logic, and back-end information must be separate subsystems. Some new systems consist entirely of reusable plug-and-play components interconnected via scripting or glue code (middleware), so that the interactions between these components must be well understood.

UML, as a modeling standard, can help alleviate at least some of this complexity. By creating accurate diagrams these, these can catch bugs or unwanted behaviour during the multiple stages of development. Today, most of the tools available for analysis and design offer reverse

engineering, so that developers can create diagrams from existing code, and synchronise between code and diagrams, so that changes made in either view are also made in the other.

One of the weak points with early UML tools was that they were not well integrated with development environments, creating a no-bridge effect between modelling and coding. As a result, organisations tended to overlook modelling in the rush to get applications, especially those used for e-business up and running as soon as possible. In today's integrated development environment (IDE), both modelling and development tool vendors are trying to eliminate this gap by partnering with one another. Vendors have created an organisation called Eclipse (www.eclipse.org) to benefit from such partnering or alliances.

UML – Its Business Benefit

UML is designed for visualising, specifying, constructing, and documenting object-oriented and component based software. The language provides consistent diagrams and specifications that can be applied to any methodology, computer language or technology. It can be used to design individual classes or components to entire systems and even databases, as many vendors offer products featuring round-trip engineering between UML diagrams and database schemas or application code. During the last couple of years, UML applications have broadened to include modelling new technologies like EJBs (Enterprise Java Beans), ActiveX, COM and XML document formats.

UML – Basic Definitions

The language consists of three main elements: the basic vocabulary, the rules for combining vocabulary elements, and several common mechanisms.

UML Vocabulary

The vocabulary provides a facility for treating things, describing relationships between these things, and constructing diagrams that group these things together.

Things	They can be defined as UML's basic building blocks and include structural things, behavioural things, grouping things and notational things.
Structural Things or Nouns	They are primarily static components and function as 'nouns' in the UML. They include the following: **Classes**, which are abstract representations of people, places, systems, etc., which have shared characteristics. Classes have attributes (values) and operations (behaviours that alter these values); **Interfaces**, which let objects pass information between one another independently from their inner workings; **Collaborations** between classes, where two or more classes work together; **User Cases**, which graphically describe the set of behaviours that produce a specific result; **Controller Classes** for managing and sequencing events; **Components**, which subsume one or more classes, interfaces, and/or collaborations; and **Nodes**, which are physical elements like computers or printers.
Behavioural Things	These are the 'verbs' of the UML. These include the following: **Interactions**, which is a set of messages between the objects that generate a specific result; and **State Machines**, which describe how an individual object

responds to events that may occur during its lifetime.

Grouping Things

They describe how an application is organised.

Package, is a conceptual tool for describing relationships between components. Since a package is logical, it may eventually end up as an object, a subsystem, or an entire system when the application is coded

Relationships

These describe how things interact with each other or combine to form largar things. There are four types of relationships in the UML:

Dependency, where a change to one element affects another element (the dependent);

Association, which is a link between unrelated classes. Aggregation is a special type of association, where one object may comprise of two or more objects;

Generalization/Specialisation, where a child object is a specialised form of (and substitute for) a more general, or parent object. This is the core concept behind class hiererchies and inheritance, where the most general class sits at the top and each lower class is a more specialised version of the class (es) above it; and

Realisation, when one element specifies something for another to carry out.

Diagrams

Diagrams provide tha syntax for the UML, as it helps users visualise a system from different perspectives. While all structured and object-oriented analysis and design methodologies use diagrams, the UML standardises them into nine distinct types. *Here we use a banking system as an example.*

Use Case Diagram

This shows a complete set of events initiated by a human actor or oomputer system. It represents a specific task and, at a high level, identifies the people, systems, and actions that are included as part of the task – as it concentrates on what is done and not on how it is done.

Use case diagrams can help teams gather requirements, communicate with a system's end users or generate test cases.

Use cases consist of one or more scenarios, each of which represents one step within a task. Withdrawing money from an account is an example of a use case. Scenarios for this use case might include working with an ATM, speaking to a teller, or transferring money from one account to another.

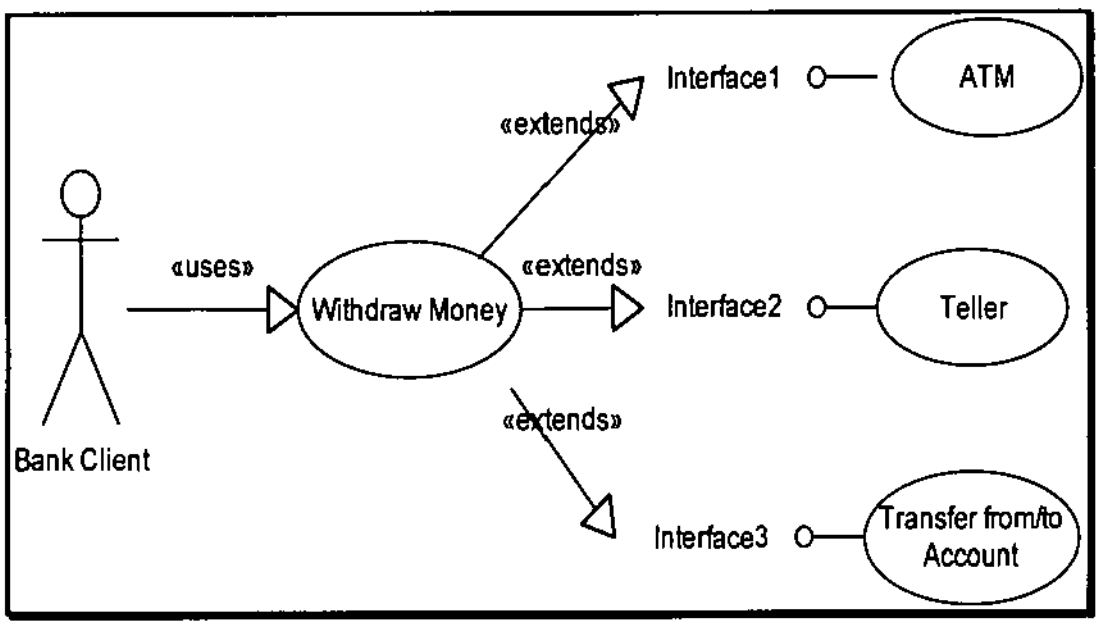

Figure 16: Use Case Diagram Example for Withdrawal of Funds

Class Diagram

These diagrams show all the classes, interfaces, controllers, collaborations and relationships between them. For each class, the diagram shows the class name, its attributes or characteristics and the names of its operations (tasks that it performs)

Cheque accounts, savings accounts, tellers, ATM machines and customers – as well as their attributes and behaviours are shown in class diagrams.

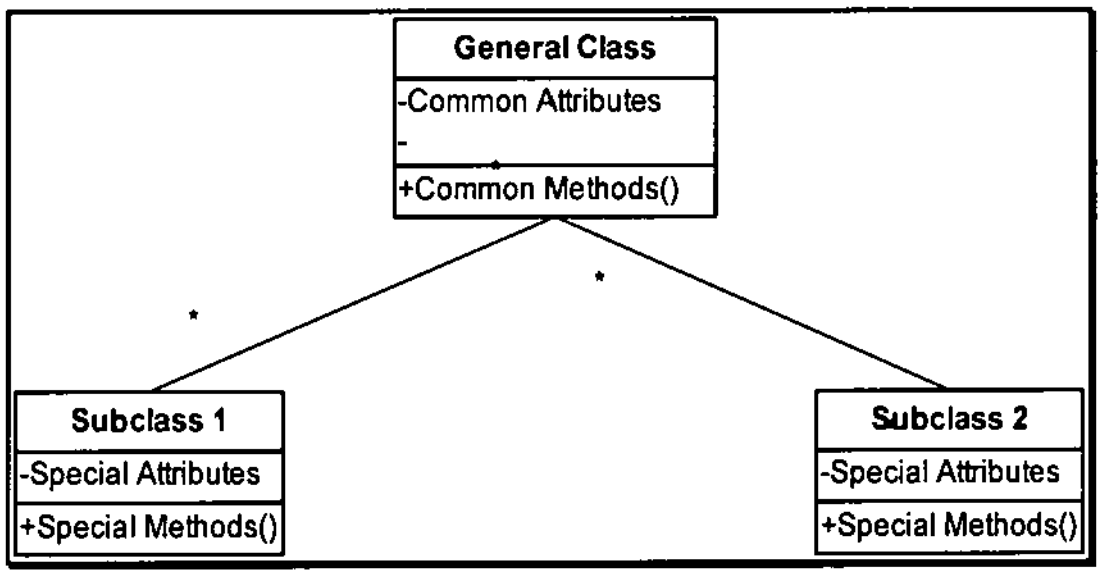

Figure 17: A Class of Objects

Object Diagram

These provide a real-world example of a class diagram, as it shows what happens to specific instances of classes. They can assist us to explain very complex relationships between classes.

An object diagram might be used to model the behaviours of interest-bearing versus non-interest bearing cheque accounts.

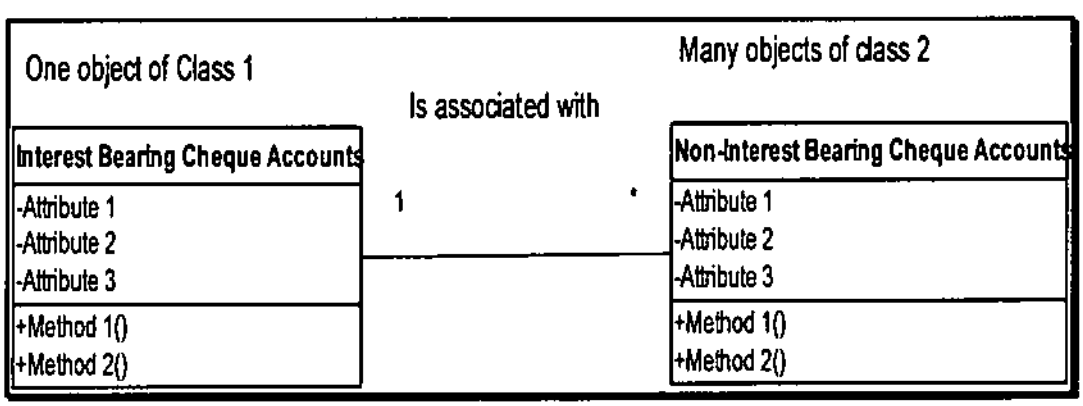

Figure 18: Two Classes with an Association

Interaction Diagrams – Sequence and Collaboration Diagrams

These diagrams show how objects collaborate to make an action. There are two types of interaction diagrams, which show different views of the same set of information.

A **sequence diagram** shows how operations are carried out over a period of time. It displays the objects involved in the interaction, the messages sent between them, and the order in which the messages are sent.

A **collaboration diagram** identifies which objects take part in the scenario. These emphasise object roles instead of the order of operations.

In the case of withdrawing money, sequence or collaboration diagrams might be used to describe all of the possible scenarios involved with withdrawing money. These might include trying to withdraw too much money from an account, successfully making a withdrawal, or withdrawing from multiple accounts.

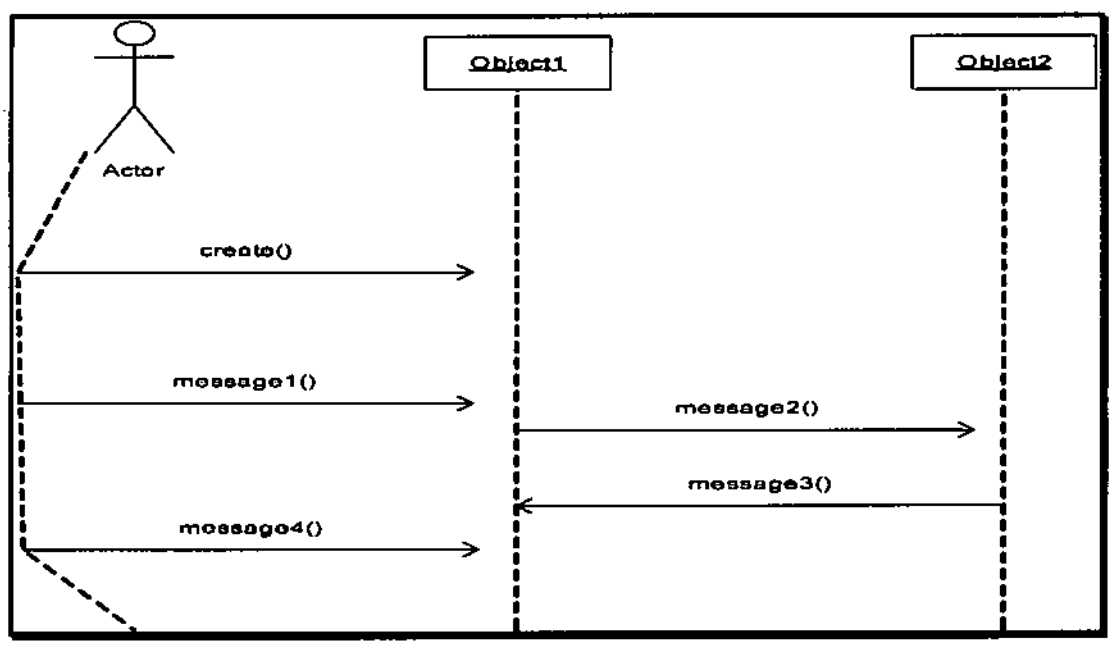

Figure 19: A Sequence Diagram Notation

State Chart Diagram

This diagram models the life cycle of a complex object, showing how it reacts to specific events within the life cycle. It identifies all of the object's possible states and all of the events, or transitions, that cause the object to go from one state to another.

In UML, a state is a situation where an object performs behaviour, waits for an event, or exists in a specific condition. State charts model the dynamics of a system, subsystem, or class.

A state chart might describe the entire life cycle of an account, showing its state before, during, or after deposits, withdrawals or other actions.

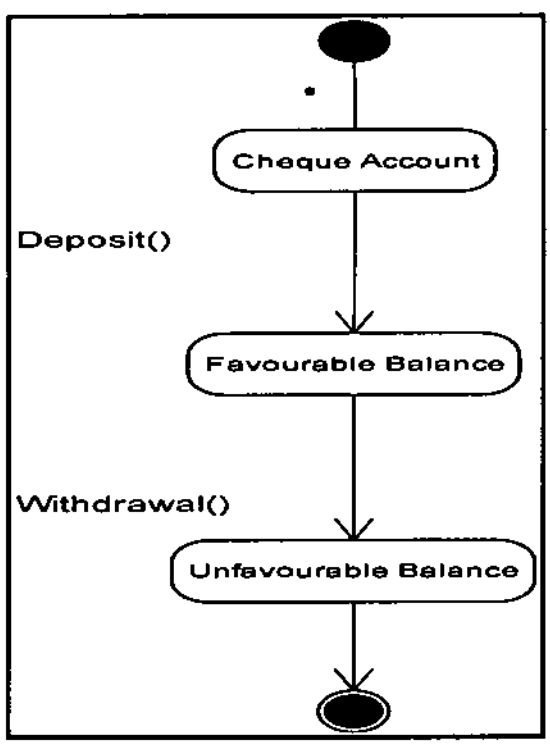

Figure 20: State Chart showing State Transitions

Activity Diagram

These are the 'flowcharts' of the UML, as they identify the sequence of activities in a process. Activity diagrams can model how objects move between states, the flow of control between activities, or the dynamics within a specific set of objects. Where state chart diagrams model how a system or class changes over its lifetime, activity diagrams model the flow of activities.

An activity diagram for an account would show all possible ways for transactions to take place over its lifetime.

Component or Package Diagram

These diagrams show how individual objects are combined in packages or components. These diagrams are abstractions that represent how objects, classes, or logical things are actually packaged together in the system.

A component diagram might represent all of the accounts owned by a particular bank branch.

Deployment Diagrams

These diagrams provide static view of how software will be implemented on specific pieces of hardware (nodes) when the system is built. These diagrams focus on hardware topology and provide the same functions as class diagrams, but applied to the physical hardware.

Deployment diagrams are useful for complex systems and can help mange round-trip engineering.

A deployment diagram may depict the type of computers (workstations, mid-range, and mainframes) and networks at each branch and at the main office.

Static and Dynamic Views

Together these nine diagrams provide both static and dynamic views of the system. Static views include the use case, class, object, component and deployment diagrams, which show the relationships between the elements that remain constant over time. Dynamic views include sequence, collaboration, state chart, and activity diagrams, which show how the state of a system and its components changes over time.

Rules

Like all languages, the UML includes a set of rules for constructing a well-formed model – semantically consistent model that works correctly by itself and in relationship to other models. The UML rules primarily relate to how language elements are named and the scope of the names within the system.

2.7. BENEFITS AND RISK – OBJECT TECHNOLOGY

Today, object technology is used to create enterprise-wide, heterogeneous software systems that integrate different types of applications and data and operate over the Web. Using the traditional, structured programming method would make it impossible for these applications to be developed, to be maintained or updated. Web access, is absolutely critical to the new generation of B2B and B2C applications, many of which requira communication between heterogeneous applications and systems from multiple organisations.

Object-oriented applications are more likely to appropriately model the world of the user than those developed by traditional methods. This is because of the almost one-to-one correspondence between objects in the user's world and program objects in the application – a relationship that makes the translation from user requirements to completed application easier

and more straightforward. From a developer's perspective, the main benefit of object-oriented analysis and development is that they are forced to think in terms of the organisation rather than the software. By using an object-oriented language, software developers are better able to model the essential features of the application quickly, since the 'meat ' of the application is largely contained within the objects themselves.

A second advantage is that objects are more likely to be reused on other software development projects, as object reuse has become one of the strongest-selling points for object-oriented programming. Reusability is especially helpful with graphical user interface (GUI) components, since using the same components in a number of different applications will generate a uniformed appearance, making the applications much easier to learn and use. Reusability results in decreased development costs and development time, as well as increased reliability and ease of maintenance. Traditional programming languages fail to provide the flexibility and component reusability needed for rapid prototyping and ongoing maintenance.

An object is an entirely self-contained package, and it can theoretically be used on any software system running on an appropriate hardware platform. Up until now, object interoperability was restricted, and most software vendors would limit the number of platforms on which their products could operate. In the past three years, efforts from a few vendors and standards organisations have made the idea of platform-independent, distributed objects possible. Such an organisation is www.eclipse.org, allowing the interoperability of objects across multiple hardware platforms.

2.8. BENEFITS AND RISK – UML

The UML's greatest strengths are its flexibility and extensibility. The notation can be used with any modern software development methodology and provides facilities for creating components as well as objects. Features like stereotypes, constraints, and tagged values make it possible to extend the language's capabilities or create custom syntax.

UML is also vendor neutral. Although originally developed by Rational Software and a variety of partners, the OMG has maintained the UML and associated standards since 1997. The technology is freely available to everyone so that nearly all object-oriented modelling tools support at least some form of UML.

The UML standard is continually evolving. Early versions suffered from incomplete semantics, including misaligned activity diagrams and state chart semantics. Over the past two years, the standard has been challenged by linguistic subsets, such as OCL and workflow. Most of these have been addressed in the latest version (version 1.3), and the next major release (version 2.0) will further refine the language.

3. COMPONENTISATION – A BEST PRACTICE

3.1. INTRODUCTION

The emergence of component-based development (CBD) is one of the most important events in the evolution of information technology. Whilst there has been a significant development in companies increasing their capabilities in both the hardware and network arenas, the delivery of quality software applications at the right time, has been a critical inhibitor in supporting business enterprises. Software development has remained a 'craft' industry, plagued with problems of delayed and cancelled projects, inadequate quality, long cycle times, and associated high costs.

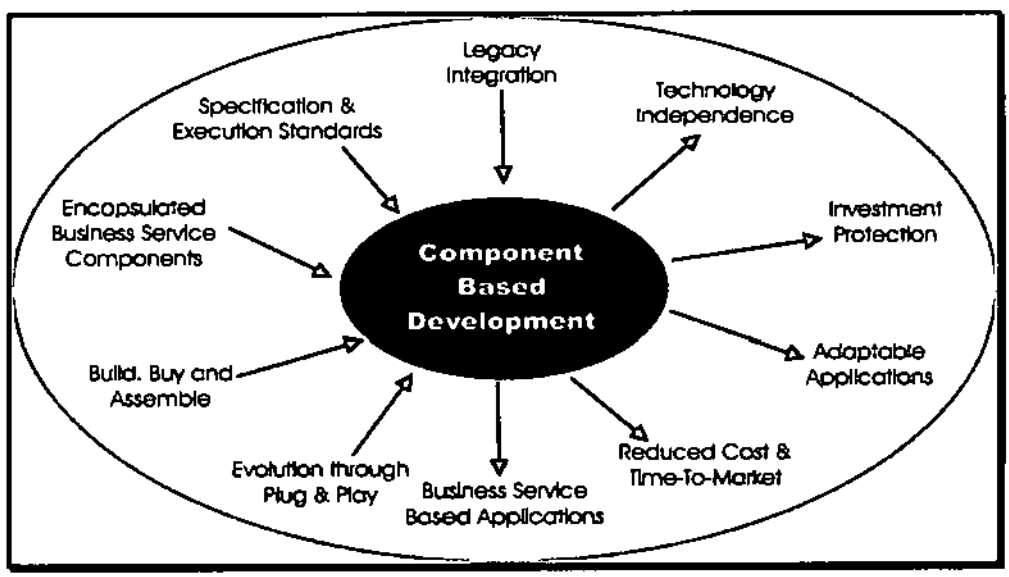

Figure 21: Impact of Components [16]

Software components address these core issues in four fundamental ways:

- They deliver applications that are designed to be adaptable to business and technology change;

- They increase productivity and speed of delivery, without a loss of quality;
- They provide a practical transition framework that combines legacy applications and legacy technologies, together with modern approaches. By providing a transition, these components protect the existing I.T. investment and enable a progressive evolution and rapid response to the changing needs of business; and
- Componentisation removes the need to make binary decisions between build and buy, enabling an integration approach.

Components offer the promise of a software marketplace where components may be built, bought or sold in a similar manner to components in other industries. Certified components can be acquired, implemented or updated alongside existing applications without disruption to the existing operational systems, which is the equivalent of hardware plug-and-play. If applications are constructed this way, they would no longer become out of date, or be unable to adapt to changes in the business processes, or adapt to new technologies. Any new component can be implemented to extend the functionality of the existing application in a controlled approach, and applications can therefore evolve rather than require periodical and costly replacement.

Component based development (CBD) is not a fad, but a set of concepts that are extremely simple to implement. The crux of CBD is that if business services/processes are designed as components, they are inherently reusable, as opposed to being designed for obsolescence.

Two years ago, the software industry was concentrated on supporting distributed objects, object-orientation and object request brokers (ORB). As technology progresses, there has been significant evolution and re-orientation of the underlying concepts. Today, there is widespread consensus that object-orientation is the best systems design approach, and software

components are the delivery mechanism and architectural foundation on which applications should be integrated and managed.

The industry leaders like IBM, Rational Rose, Sun Oracle and SAP and many others have embraced component models as the interoperability back-plane for their respective product lines. We can now observe that:

- Component technologies are becoming a fundamental part of the operating system environments;
- Middleware products now provide support for the de facto component models;
- New age development tools provide modelling support and component build/generation capabilities; and
- Enterprise package solutions all embrace componentisation as a means of solving some of the core issues relating to enhancement, maintenance and upgradeability.

The Internet has also been a major influence in raising the awareness that websites and their processes require constant business process change, and the best way to enable constant change is by componentisation.

3.2. WHAT ARE SOFTWARE COMPONENTS?

A definition:

"A component is a unit of composition with contractually specified interfaces and explicit context dependencies only. Context dependencies are specified by stating the required interfaces and the acceptable execution platform(s). A component is subject to

composition by third parties. For the purposes of independent deployment, a component needs to be a binary unit". Szyperski, 1998 [12]

Another definition:

"A component is an identifiable piece of software that describes and/or delivers a meaningful service that is only used via well defined interfaces". Butler Consulting Group.

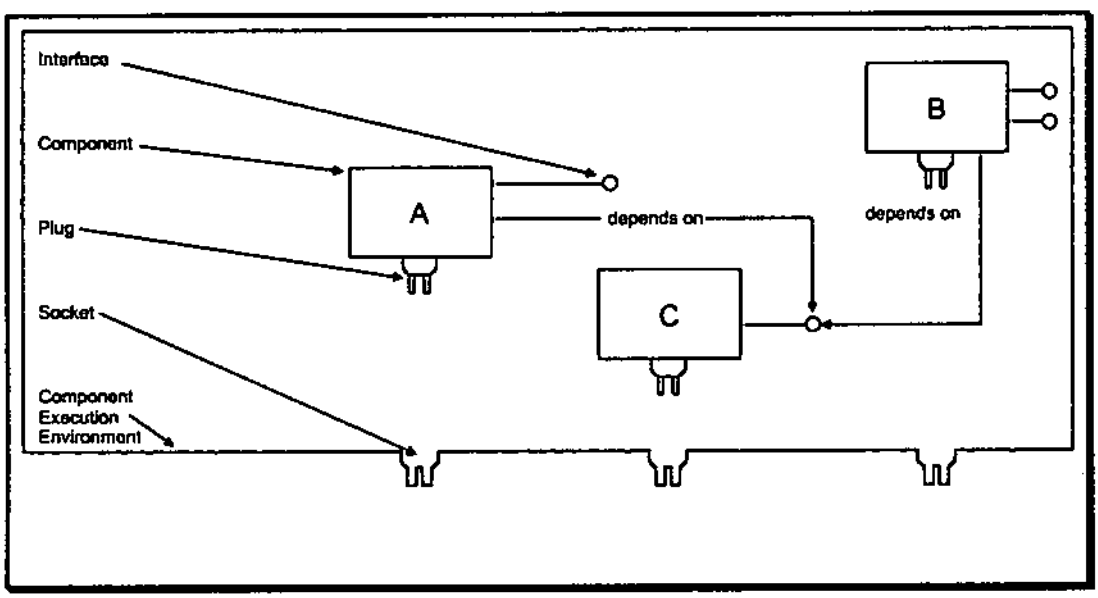

Figure 22: Basic Component Concepts [12]

A component is part of a larger structure, and it is an element that contributes to the composition of the whole, with stable, defined interfaces. We can draw analogies with manufacturing organisations or motor assembly organisations to illustrate how components work in a system.

Essential Characteristics
• Identifiable • Traceable through full development life cycle • Replaceable by components offering the same services • Accessed on via interfaces • Services offered through interfaces must not change • Accurately documented services
Desirable Characteristics
• Physical implementation is hidden • Independent of other components • Encapsulated • Reuse of services not constrained by physical implementation • Can be used dynamically • Offer generic services that can be adapted to specific needs

Table 5: Characteristics of Components

Components exist in a physical, logical or conceptual form, and they should all exhibit the following characteristics:

Identifiable – components should each have a clear identity;

Traceable – components should retain their identity and be traceable once they have been consumed into another component or application;

Replaceable – they could be replaced by a new version or another component offering the same service, with no impact on the consuming application;

Accessed via Interfaces – they are only accessed vie defined interfaces with no dependencies on the physical implementation of the component;

Services offered through interfaces should not change – although the physical implementation of a component might change, the service offered via the interface should not; and

Accurately documented service – in order to enable reusability, services offered via an interface must be accurately documented in such a way that it is possible to understand not just how to communicate with them, but what service (functionality) they provide.

These characteristics are desirable and optional:

Physical implementation is hidden – the actual service delivery of the component should be hidden from the consumer;

Independent – components should be independent of the implementation of other components, otherwise they will be termed as sub-components;

Encapsulated – components should be encapsulated and only expose their services they provide via the interfaces;

Can be used dynamically – components can be dynamically assembled into applications. Due to a growing need for applications to be available 24*7*365 (e-commerce applications), availability of these applications can be better met if these applications are dynamically upgraded; and

Offer a generic service – components may provide a generic service that can be used in unpredictable combinations, thereby increasing reusability.

3.3. THE REASON FOR SOFTWARE COMPONENTS

Moving to software components is not just a technology issue, but it is more about the application architecture, the delivery process and the adoption of standards. For many organisations, the delivery of quality software applications at the right time has been a critical inhibitor to business success.

Dissatisfaction with application development over the past decade has led many organisations to acquire packaged application software, and this switch to packaged solutions has not been a satisfactory alternative. Software packages have not been a satisfactory alternative, as it often involves long, costly and difficult implementations, integration and upgrading of these systems.

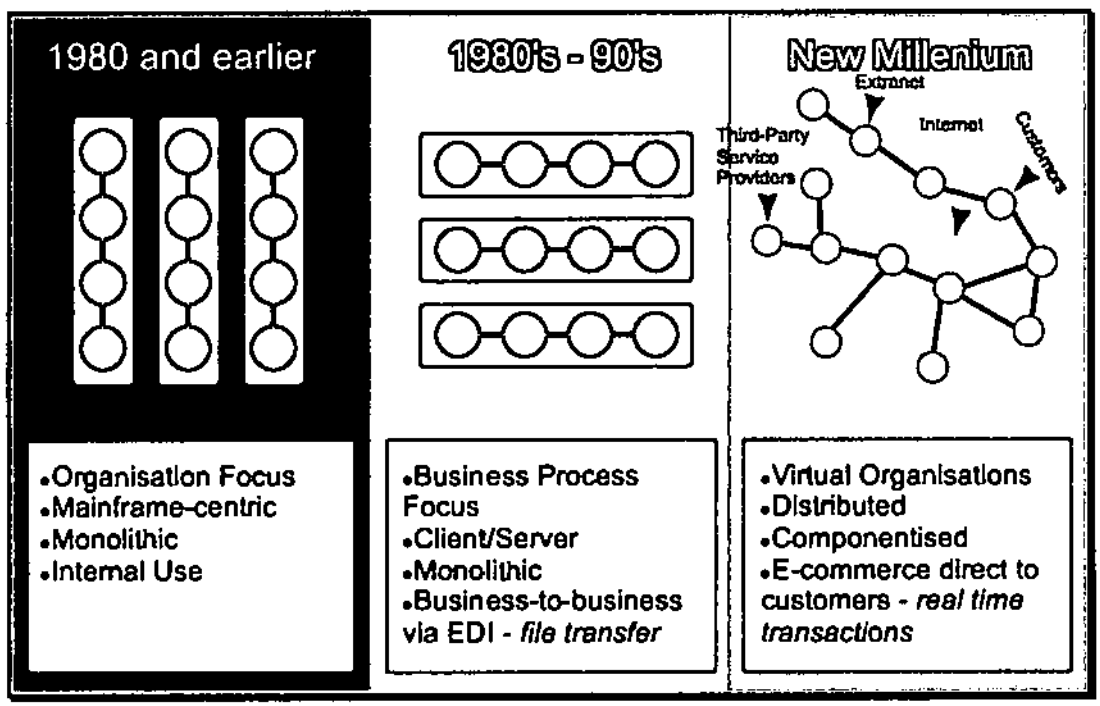

Figure 23: Applications in the New Millennium [16]

In the past, an application was something that was relatively contained. Each system was scoped within the responsible organisational unit or function. Figure 23 above illustrates that applications were built in silos or were functionally-driven. In the last decada, client/server based information systems were being designed to support cross-functional business processes, and they were still typically monolithic business process implementations.

The new age (Internet processes and e-commerce) requires a more adaptable application and information system environment. It has become apparent that it is no longer acceptable to specify requirements or select packaged solutions that fit solely your current business products

and processes. In all industry sectors, the impact of business and technology changes are such that applications constantly have to meet new requirements.

- **Time to Market**
  Applications must be delivered synchronously with the business change cycle time
- **Adaptable**
  Applications must be able to respond rapidly to change without forcing costly rewrites of the entire system
- **Supporting Integration**
  There is often no time or justification to replace legacy applications. New functionality must be integrated with other packages, existing applications and data sources
- **Applicable**
  Applications must align with the business. It is unacceptable to require the business to change to the application functionality.
- **Upgradeable**
  Improvements to an existing function must be possible without impact to other functions

Table 6: Prime Business Objectives for Applications

Componentisation addresses these core business issues directly, as they are inherently and explicitly designed to adapt to change. If applications were constructed in this manner, they would no longer become out-of-date, or be unable to adapt to changing business processes or new technologies.

The service approach is an ideal match to emerging business and technology requirements. A virtual organisation does not want to get involved in the implementation of the service, and only needs to know if the service has been performed or not.

- **Productivity**
  Reduce costs and time to market by improving productivity of developers
- **Quality**
  Improve the quality of deliverables in terms of reliability and business fit
- **Reduce Costs**
  Reduce cost of delivery and maintenance of applications
- **Manageable Process**
  More predictable delivery life-cycle
- **Management of Skills**
  Concentrate business domain and technical skills where most required
- **Control over Outsourcing and Acquisition**
  Improve specification and reduce binary decisions at application level

Table 7: Prime I.T. Objectives for Applications

I.T. departments also have a number of objectives, which enable them to deliver on the prime business objectives. Componentisation addresses and delivers on these issues in a number of ways.

- Component reuse provides radical improvement in productivity and quality. Black box components extend the option for reuse by removing dependencies on development tools and languages, as they are already certified and tested. The reuse of components may be inhibited by cultural and organisational issues;
- The component approach breaks down large applications into more manageable chunks that can each be micro-managed in terms of their delivery and resources, and which can be assigned on the basis of their appropriate skills and competence; and

- Outsourcing and package acquisition can take place at the component level, enabling organisations to retain control at the application level and more easily integrate them alongside legacy applications and in-house new build.

Based on the aggregation of the three delivery mechanisms, componentisation can reduce costs and improve manageability of the application delivery.

3.4. COMPONENTS AS THE SERVICE PROVIDER

Components provide a service, and applications will provide the solution. An application is delivered as a solution to a particular business requirement and provides computer support to a business process. We can also define the business requirement as an expression of a service that it needs at any point in the business process.

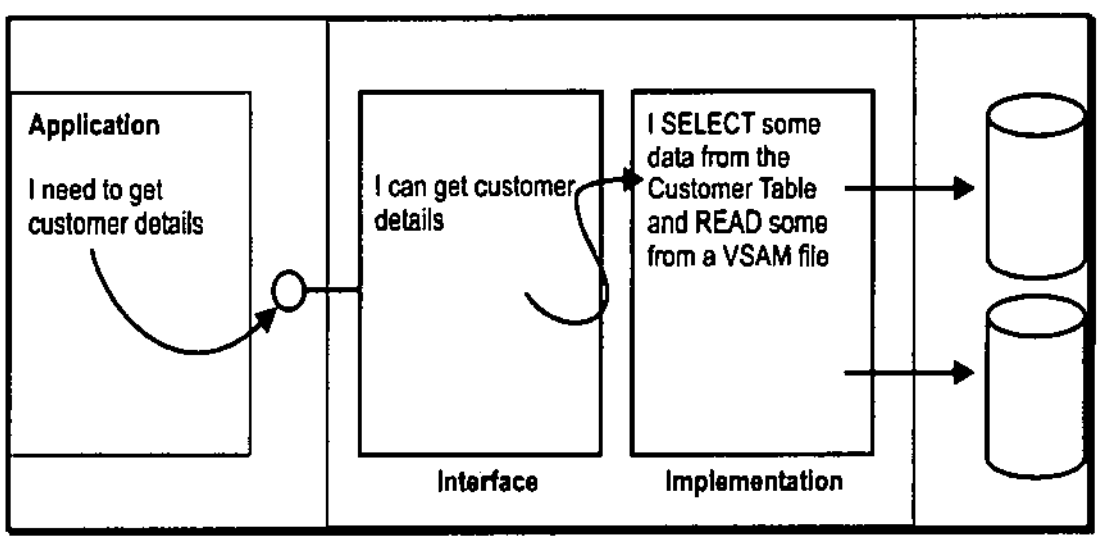

Figure 24: Components and Interfaces

In Figure 24, we explain what services the component logically provides to the application. Assume that the logical service of ***get customer details*** may require a physical implementation

of a SQL Query on the Customer Table in a relational database. Most developers would embed the SQL statement into the application. Also assume, to satisfy the service, it is necessary to retrieve part of the data from a legacy VSAM file. Assume, next year the VSAM file is replaced by another table in another relational database, and the following year the VSAM file is replaced by an object-oriented database. It would imply that the service remains logically the same, yet the application would be required to be changed twice.

Moving the code that deals with the physical implementation of ***get customer details*** from the application into a component removes the need to maintain the application as these changes occur. The component would still be required to be changed twice, but if no component was used in the five applications that use the ***get customer details***, we would have seved us eight maintenance tasks.

3.5. TYPES OF COMPONENTS

Implementation Specific Components – They are components at their lowest level of abstraction, e.g. class libraries, and they exhibit many of the characteristics of a component, though they do not provide a meaningful service on their own. Since they are language or tool specific, the reuse opportunities are more limited.

Encapsulated Components – They provide a set of meaningful services focused on a particular area, and are constructed in various ways. Due to the component only being accessible via the interfaces, the internals of the component are irrelevant. Encapsulated components form a boundary around all their contents facilitating replacement.

Component Frameworks – Is a collection of items that are reusable as a group, but which remain incomplete in terms of delivering an application to the business. A framework is a pre-built assembly of components, together with glue logic that binds them together and is designed

to be extended. A framework is also designed to offer a large number of related services, centred on a broad theme.

Componentised Applications – Is a complete working application. At this granular level, the application will still exhibit the characteristics of a component.

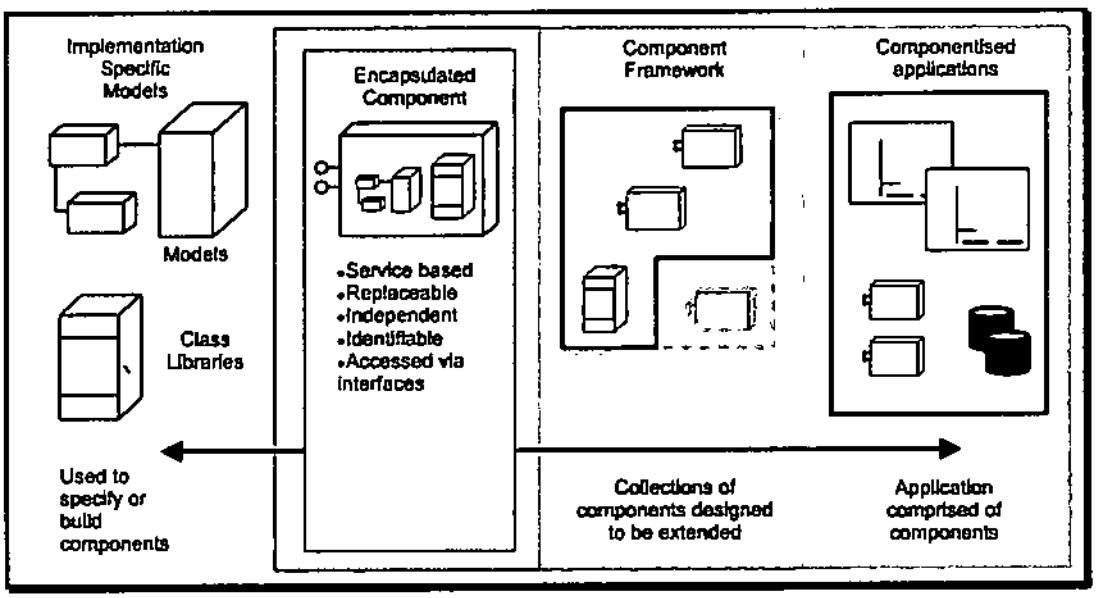

Figure 25: Types of Components [16]

4. GOING OVER THE WATERFALL

4.1. INTRODUCTION

Information technology (I.T.) has become the heart of every business. It is part of the vehicles we drive, the cell phones we use to communicate with, and the back-office transactions that allow us to purchase books or magazines without leaving our homes. In the networked economy, unfortunately time pressures, change of business processes, growing complexity, and a shortage of skilled developers have made software development more challenging than before.

In today's digital economy, effective software solutions have become the critical competitive differentiator for businesses in all markets. Industries in many sectors of the economy are incorporating more and more software, embedded devices, and computer infrastructure into their products and their business operations.

The traditional barriers to exploiting the powers of computing (processing power, network access and storage capacity) are no longer barriers for most companies. However the ability to consistently build truly excellent applications, and to complete it within the prescribed timeframe, often remains elusive. According to the Standish Group's widely published 1999 CHAOS study, only 26 percent of today's software development projects are completed on time and within budget.

Business leaders in every market segment including telecommunications, financial services, electronic product manufacturers and e-business infrastructure providers are looking for ways to

improve the economics of software development, but software development is getting tougher and more stylish, and sophisticated functionality is required. Software development teams are also requested to interface more diverse, complex hardware and software resources, and these requests are required within a restricted timeframe. To succeed in the new economy, software development teams must address two factors:

SPEED: No tolerance for delay

Time to market pressures on development teams have escalated radically, as market innovation and competitiveness depends so much on the roll-out of new software, and development life-cycles have become mission critical. Organisations can no longer wait nine to twelve months for a development team to design, write, test and refine an application. Those that cannot accelerate the development process will miss important windows of opportunity and lose out more to nimble competitors.

QUALITY: No tolerance for error

In the past, applications were strictly developed for internal consumption, and minor flaws in the software may have delayed a project. Today, software touches every relationship in the supply chain, or every user on the Internet, and errors or bugs in the software may result in a revenue loss and permanently alienate clients or prospective clients. In this highly exposed, high-stake environment, development teams have to consistently deliver software that is robust under the 24*7*365 demand of the marketplace.

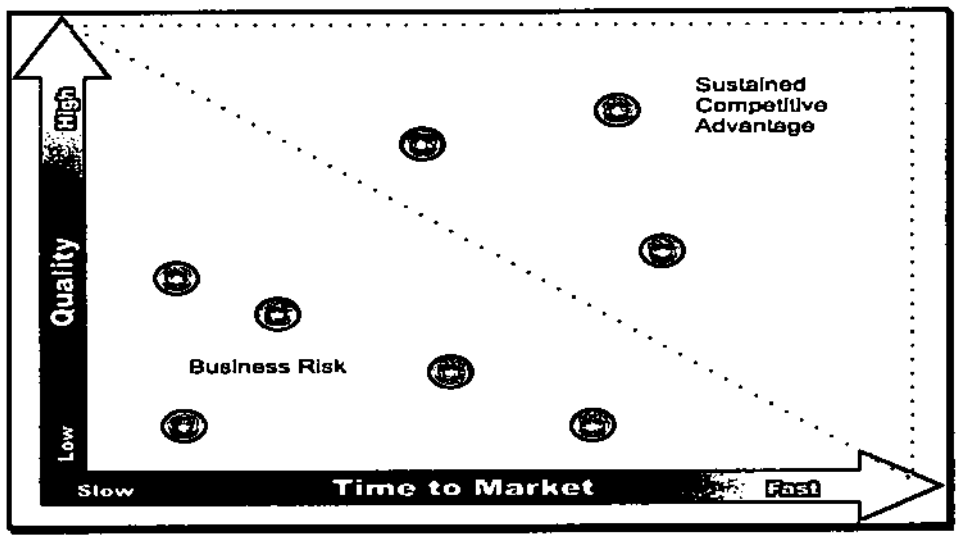

Figure 26: The Software Paradox [13]

The two requirements; a faster time-to-market and extremely high quality form the heart of the software development paradox. In the new economy, the challenges of software development cannot be solved simply by hiring more development staff, or adopting a new programming language. In fact, new languages and technologies to I.T. and technology skills portfolio tend to intensify these problems.

New aged winning companies embrace the reality of software as a competitive advantage, by improving the economics of software development with better tools, processes and enhanced skills of existing team members, which is the key to producing better bottom line results.

Employing best practices that optimise both speed and quality of software development has exhausted the software development paradox. Six winning strategies have been created for time compression and risk reduction.

4.2. Six Winning Strategic Imperatives

Develop Software Iteratively

One cannot completely eliminate problems across the development lifecycle, but it is possible to identify risks earlier during the analysis, design, coding, testing or managing phases of the project. New age organisations use an iterative development approach that allows for continuous refinement of project deliverables.

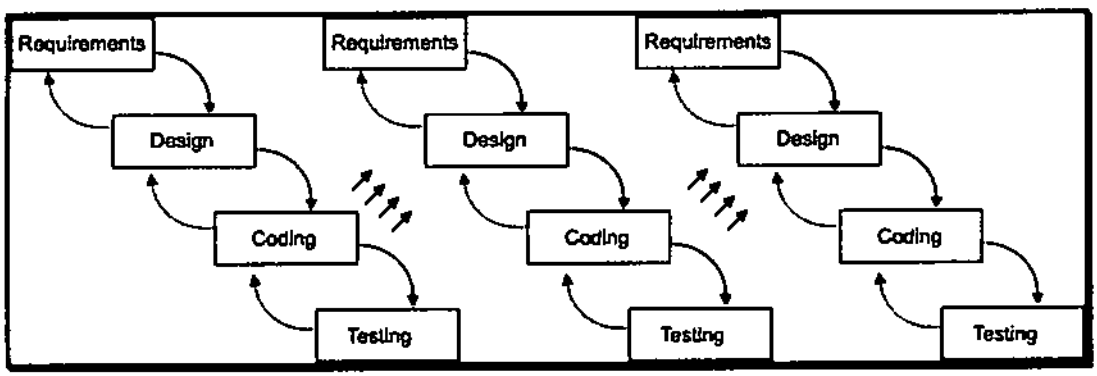

Figure 27: An Iterative Approach to Software Development

An iterative approach to software development confirms that specific functionalities have been effectively developed at agreed milestones, and successful teams manage and assure quality throughout the project.

Manage Requirements

Requirements management is a systematic approach to eliciting, organising, communicating end managing the changing requirements of a software intensive system or application. Too little time is often spent understanding the real business problems, requirements and the needs of the users and other stakeholders of the system. Where there is no clear understanding of these issues, organisations can develop off-target solutions, miss critical windows of opportunity or be overrun by their competitors. By following a process that keeps projects on track from the

beginning, organisations will avoid the costs of repairing or re-releasing software later in the cycle, and by managing requirements, teams are able to address changes, avoid 'feature creep' and keep projects on budget and on schedule.

Hence, winners clearly keep their projects focused on core business value and consistently apply this type of financial discipline to their development efforts without compromising speed or quality.

USE COMPONENT-BASED ARCHITECTURES

In complex and large applications, changes in one area of an application will have a 'domino effect' and thus impacting many other areas of the application and any dependent applications. Successful development teams address these potential dependencies and perils through the use of component-based architectures.

Assembly of software in manageable modules or chunks create resilient and flexible architectures that prevent the ripple effect of change, and it also enables developers to build applications that can be easily modified without massive effort.

The iterative approach allows early iterations to produce and validate a software architecture that will gradually evolve, through subsequent iterations, into a final system.

The re-use of components also reduces the total amount of code re-organisations needed to write or develop further systems. This further enhances the speed of development and reduces the opportunities for errors to be introduced into the process.

VISUALLY MODEL YOUR APPLICATION

In the new economy, most software applications execute highly complex operations and processes. In order to understand the underlying functionality, development teams require an accurate way of modelling this functional complexity, in order to comprehend, specify or document the systems they have been asked to build.

A visual modelling aid should be used to achieve this critical level of abstraction, which is important to clarify the design decisions and to manage system complexity. Through the use of a modelling language, such as the industry standard Unified Modelling Language (UML), development teams can achieve a common ground that allows them to communicate technical complexities into business abstractions or business processes.

A CONTINUOUS CIRCLE OF QUALITY

It is no secret that it is much more expensive to find and fix problems after an application is deployed, than it is to catch them early on in the process. The winning process is that development teams continuously verify the quality at each step of the development process.

Most organisations accomplish this continuous testing through the assistance of I.T. automated testing tools.

CHANGE CONTROL

The development of applications is about change, as the application changes, so does the underlying business processes. Thus development of software requires an effective change

management programme. With an effective probe programme in place, development teams can produce releases on time, on budget and with predictable quality.

New age organisations understand that change management is the key to prioritising team activities and priorities. Controlling the change throughout the development lifecycle and managing its inherent complexities can achieve this. Development teams know that if they do not control change, change will surely control them.

4.3. The Iterative Development Process

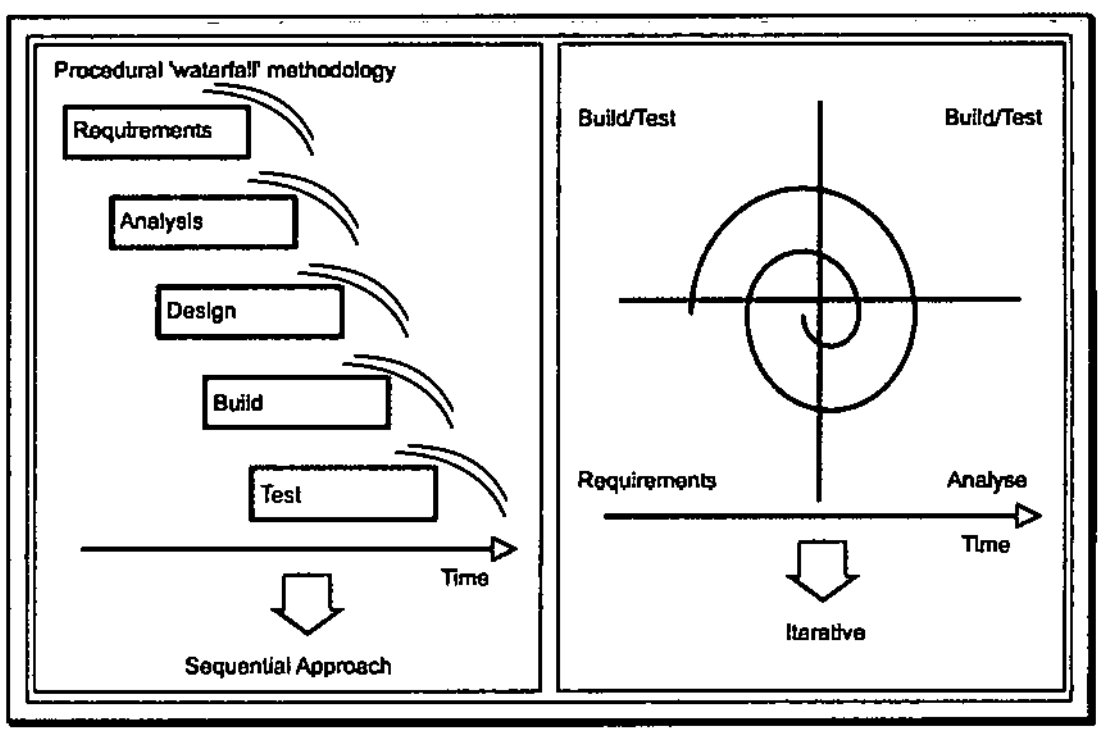

Figure 28: The Waterfall versus the Iterative Approach

Most development teams still use a waterfall process for development projects, completing in strict sequence the phases of requirement analysis, design, implementation, integration and testing. In this inefficient approach, key team members remain idle for extended periods and the approach also defers testing until the end of the project lifecycle, when problems tend to be

tough and expensive to resolve. This poses a serious threat to release deadlines or to the quality product.

Iterative development is a technique for developing software that reduces risk by breaking a large body of work down into smaller, more manageable units of work. It embraces the notion that software development tends to be an organic process, subject to constant change and evolution throughout the duration of the project lifecycle.

As it is not possible to understand everything about a software system in the early stages of a development project, the process acknowledges that change will be a constant, and that change to the software system design, if effectively managed, will be positive. As change is imposed by many factors, it should often be initiated as a result of a better understanding of the requirements. Under these circumstances, embracing change is essential because satisfying the requirements is always the fundamental goal of the project.

The iterative process provides the opportunity for full stakeholder involvement and continuous feedback with the development team, as they are producing tangible and visible results in short intervals of time, the stakeholders have opportunity to provide feedback and input into the process based on their exposure to the emerging software system.

The iterative approach to systems development is superior for a number of reasons:

- It takes into account the changing requirements, as the truth is that requirements usually change. The change of requirements, or 'scope creep' that are unnecessary and are not customer-driven as the project progresses, are always primary sources of project trouble, leading to late delivery, missed schedules, dissatisfied customers and a frustrated development team;

- With the waterfall approach, integration is one 'big bang' at the end of the process. With the iterative approach, integration is approached progressively and broken down into six to nine smaller integration levels, involving fewer elements of integration, thereby saving at least 40% of the total effort at the end of the project;
- Risks are usually discovered or addressed during the integration phase. The iterative approach mitigates risks earlier in the cycle, as earlier iterations allow one to quickly see whether perceived risks prove to be real, or new risks are uncovered in each iteration;
- Iterative development provides management with a means of making tactical changes to the product or to compete with existing products. It also allows the development team to release a product early with reduced functionality to counter a move by a competitor;
- Iteration facilitates reuse, as it is easier to identify common components as they are partially designed or implemented, than to recognise them during the planning phase. Design reviews in early iterations allow software architects to spot potential opportunities for reuse, and then develop and mature common components for these opportunities in subsequent iterations;
- If errors are corrected over several iterations, the result is a more robust architecture. As the product moves beyond inception phase into elaboration phase, flaws are detected even in early iterations rather than during a massive testing phase at the end of the project.
- Performance bottlenecks are discovered at a time when they can still be addressed, instead of creating a panic on the eve of delivery;
- Development staff can learn along the way, and their various competencies are more fully employed during the entire lifecycle;

- The development process can be improved and refined along the many iterations of the project. The assessment at the end of iteration not only looks at the status of the project from a product or schedule perspective, but also measures the change required from an organisation perspective, in order for the next iteration to perform better;
- The software development team is able to focus on a smaller collection of product deliverables at a given time, thereby increasing the quality of the results; and
- Iteration milestones keep the development team continuously focused on results, which encourages peak productivity.

Many organisations have slowly become aware of how significant a well-defined and well-documented software development process is, to the success of their projects and as well as to the organisation. The Capability Maturity Model (CMM), which was developed by the Software Engineering Institute (SEI), has become a beacon for many organisations, as they aim to attain a level 2, 3, or higher.

4.4. THE RATIONAL UNIFIED PROCESS®

RUP is a software engineering best practice that provides an organisation with guidance for streamlining their teams' development activities. It is an industry-wide process platform that has a span of collaboration with key industry leaders, including IBM, Microsoft and Sun Microsystems.

Several industry trends have driven the development of Rational's industry wide platform for best practices. Some of the most influential are:

Rapid Evolution of Technology. The constant evolution of technology and the lack of knowledge is becoming a major impediment for the adoption of new technology. To address this issua, software vendors have increased their focus in guidance and best practices in the marketplace, with the goal of effectively distributing their advanced tools and technologies onto a developers' desktop;

Standardisation of Tools, Methods and Processes. Over the last 30 years, the software engineering industry has continuously moved towards standardisation and commercialisation of technology and knowledge. The Unified Modelling Language (UML) was originally developed by Rational and its partners, and later adopted and managed by OMG as an industry standard. By the late 1990s, companies were ready for the standardisation of processes, and many companies abandoned their homegrown processes, and began looking for commercially available ones. This allowed the large investments necessary to build an enterprise-wide process, and RUP was borne; and

Consulting Companies became less Reliant on Process as a Competitive Differentiator. The competitive differentiation based on proprietary processes that consulting companies once relied on, has become less significant as a business driver. Process used to be a selling factor for any consulting firm, but customers got exhausted by project overruns and poor quality of delivered applications, and they required more assurance that the practices used were proven. Since commercially available processes with a proven track record were creating a demand among clients, system integrators tended to move away from home grown processes to commercially driven processes, e.g. RUP; and

Industry Scepticism regarding Existing Processes. The software development industry became sceptical towards traditional and heavyweight processes that were prescriptive, e.g. the waterfall approach. These processes typically promoted a waterfall sequence of development, functional decomposition and a document-centric approach. They proved to be ineffective for several reasons:

- Processes were of little value to developers as the guidelines primarily focused on describing *what* should be done, and not *how* it should be done;
- The waterfall approach did not allow effective risk management, as it was ineffective in addressing key risks early in the project cycle;
- The process was difficult to access, as it was often published in thick binders, and the content was not integrated with the development tools; and
- The process did not focus on delivering value to the customer and other stakeholders, as its activities did not focus on delivering real business value to end users.

RUP proposes an iterative approach to software development, which implies that a project is divided into several small projects that run sequentially or directly after another. Every one of the iterations has a well-defined set of objectives, and concludes by delivering an executable that is a step closer to the final product than the last one. A typical iteration will contain elements of requirements management, analysis, design, integration, testing and implementation.

This structured approach to iterative development divides a project into four phases:

Inception – understand what to build;

Elaboration – understand how to build it;

Construction – build a beta version of the product; and

Transition – build the final version of the product.

Building the software product incrementally allows not only for early risk mitigation, making tactical changes, and managing scope, but also for fine-tuning the process itself.

4.5. RUP® A PROCESS PERSPECTIVE

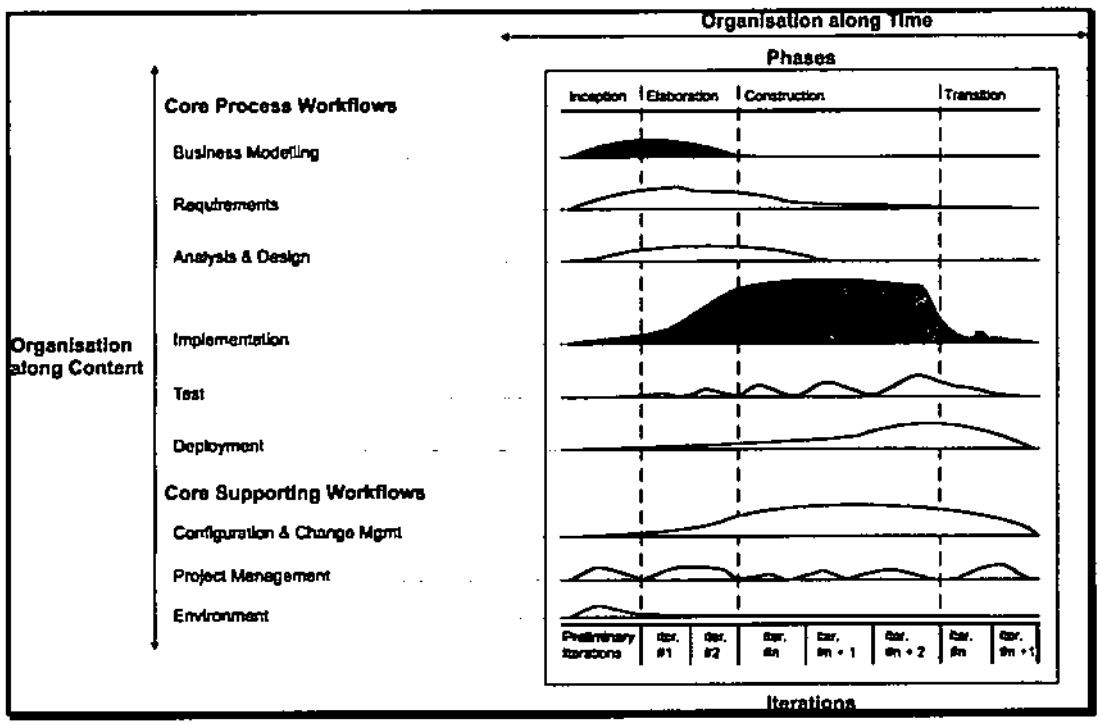

Figure 29: The Iterative RUP ® Model [13]

The iterative process can be defined along two dimensions:

The **horizontal axis** representing time, as it illustrates the dynamic aspect of the process as it is endorsed by the stakeholders. It is expressed in terms of cycles, phases, iterations and milestones; and

The **vertical axis** representing the static aspect of the process. It is expressed in terms of activities, artefacts, workers and workflows.

The software lifecycle is divided into *cycles*. Each cycle works on a new generation of the product. The Rational Unified Process divides one development cycle into four consecutive *phases*:

Inception phase, Elaboration phase, Construction phase and Transition phase.

Each phase is concluded with a well-defined *milestone*. A milestone point in time where certain critical decisions must be made or key goals be achieved.

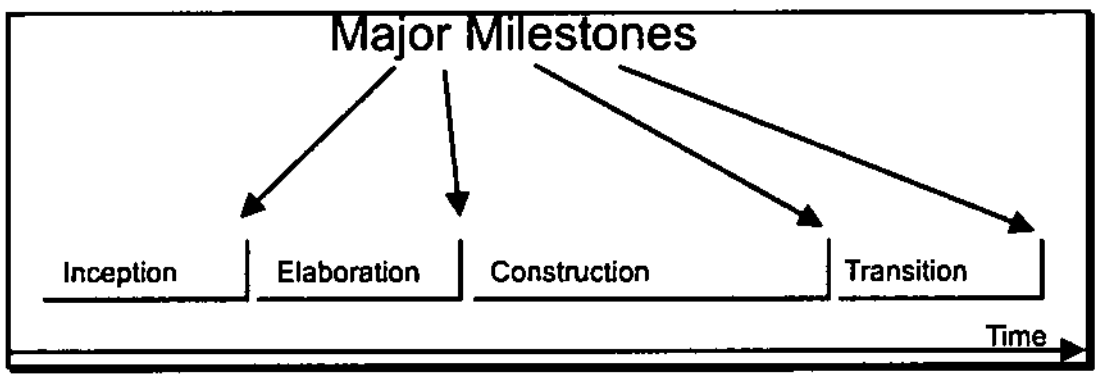

Figure 30: The Phases and Major Milestones in the Process [13]

The Inception Phase. The team would establish the business case for the application, and delimit the scope of the project. This can be accomplished by identifying all external entities with which the system will interact and a definition of this interaction at a high-level. Use-cases are utilised to describe these external entities. The business case should also include critical success factors (CSFs), a risk assessment, an estimate of the resources required, and a plan of all the phases showing the dates of major milestones.

A brief description of the outcome of this phase would be:

- A vision document describing a general vision of the core project requirements, key features, and major constraints;

- An initial use-case model (10%-20% complete);
- An initial project glossary defining all terminology used in the project;
- A business case, including the business context, critical success factors such as revenue projection, market recognition and a financial forecast;
- An initial risk assessment;
- A project plan, describing the various phases and iterations; and
- One or several prototypes.

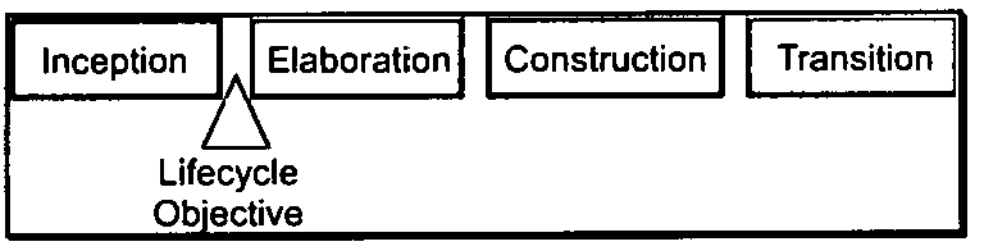

Figure 31: Lifecycle Objective [13]

At the end of the Inception phase, we have the first major project milestone, namely the Lifecycle Objective, which needs to be evaluated on the following criteria:

- Agreement and consensus from all stakeholders on scope definition and cost schedule estimates;
- An understanding of the requirements as depicted by the reliability of the primary use-cases;
- Creditability of the cost schedule estimates, priorities, risks and development process;
- An in-depth understanding of any architectural prototype that was developed; and
- Actual expenditures versus planned expenditures.

The project may be canned or considered for re-thought if it fails to pass this milestone.

The Elaboration Phase. We analyse the business requirements to establish a sound architectural foundation, develop the project plan, and eliminate the highest risk elements of the project. In order to realise these objectives, we must have an in-depth understanding of the system, as architectural decisions will have to be made with an understanding of the whole system – its scope, major functionality and non-functional requirements, such as performance of the system.

The elaboration phase is said to be the most critical phase of the lifecycle, as the engineering is said to be complete, and a decision has to be made on whether or not to commit to the construction and transition phases. The activities within the elaboration phase will ensure that the architecture, requirements and plans are stable enough, and the risks are sufficiently mitigated, in order to predict the cost and schedule for the completion of the development. The organisation would commit to a fixed-price construction phase with this level of reliability of the output of this phase.

- A brief description of the outcome of this phase would be:
- A use-case model (at least 80% complete);
- A description of the non-functional requirements and any requirements that are not associated with a specific use-case;
- A software architecture model;
- An executable architectural prototype;
- A revised risk list and revised business case;
- A project plan defining the overall project, including a granular state of the iterations, and the evaluation criteria for each iteration; and

- A preliminary user manual.

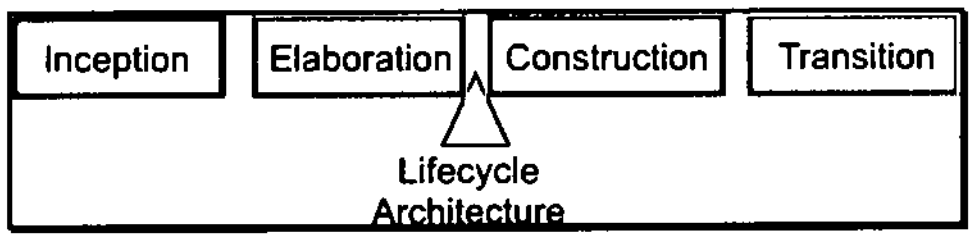

Figure 32: Lifecycle Architecture [13]

At the end of the elaboration phase, we reach our second important project milestone, called the Lifecycle Architecture Milestone. We examine the detailed system objectives and scope, the choice of architecture and technologies, and the resolution of the major risks.

The evaluation of the milestone is based on answering the following questions:

- Is the vision of the product stable?
- Is the architecture stable?
- Does the executable prototype show that major risks have been addressed and resolved?
- Is the plan for the construction phase sufficiently detailed and accurate?
- Do we have consensus from the various stakeholders for the current vision, and do they agree with the current plan for development of the system? and
- Is the actual resource expenditure versus planned expenditure acceptable?

The project may be canned or considered for re-thought if it fails to pass this milestone.

The Construction Phase. This phase consists of developing and integrating all components and application features into a product, and ensuring that all features are thoroughly tested. An

analogy of this phase would be a manufacturing process where emphasis is stressed on managing resources and controlling operations to optimise costs, durations and quality. The development team undergoes a mindset transition from the development of intellectual property during tha inception and elaboration phases, to the development of deployable products during this and the subsequent phase.

The outcome of the construction phase is a product that is ready for the end-users. It consists of:

- The software product integrated into the organisation's architecture;
- The user manuals; and
- A description of the current release.

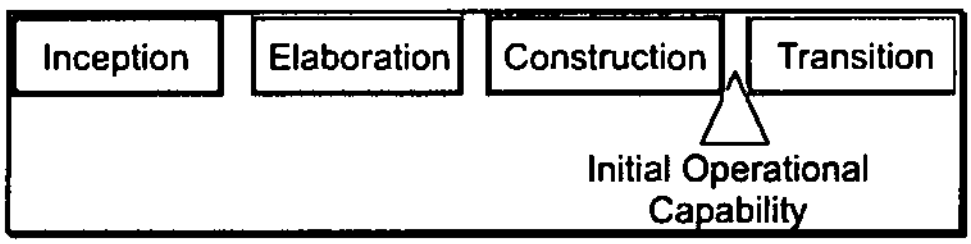

Figure 33: Initial Operational Capability [13]

At the end of the construction phase, we reach our third major project milestone, called the Initial Operational Capability Milestone. The team would decide if the software and the users are ready to go operational, without exposing the project and to high risks. This release if often called a 'beta' release.

The evaluation of this milestone is based on answering the following questions:

- Is the product released stable and mature enough to be deployed in the user community?

- Are the stakeholders ready for the transition? and
- Are the actual resource expenditures versus planned expenditures still acceptable?

The transition phase may be postponed by one release if the project fails to reach this milestone.

The Transition Phase. The core purpose of this phase is to transition the software product into the user community. Once the product is deployed, the user community would usually require the team to develop new releases, correct some problems, or finish the features that have been postponed.

This phase is usually entered into when the baseline is mature enough to be deployed in the user community, and quality has reached an acceptable level of assurance.

The transition phase would typically include several iterations, including beta releases, general availability releases, as well as bug-fix and enhancement releases. Effort is also spent in developing user-oriented documentation, training users and supporting users.

The outcome of this phase will include:

- Achieving user self-supportability;
- Achieving stakeholder acceptance of the complete product, which should be consistent with the criteria set out in the vision; and
- Achieving final product baseline as rapidly and cost effectively as possible.

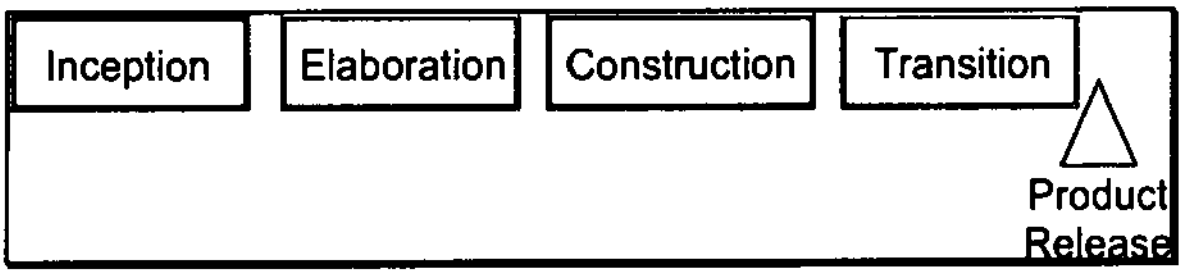

Figure 34: Product Release [13]

At the end of the transition phase, we reach our fourth major project milestone, called the Product Release Milestone. At this point the team decides if the objectives were met, and if they should start another development lifecycle.

The evaluation of this milestone is based on answering the following questions:

- Is the user satisfied? and
- Are the actual resource expenditures versus planned expenditures still acceptable?

Iterations. Each phase in the cycle can be further broken down into iterations. An iteration is a complete development loop resulting in a release, which may be internal or external, of an executable product, or a subset of the final product under development. Each release will grow incrementally to become the final application.

4.6. Risk Mitigation

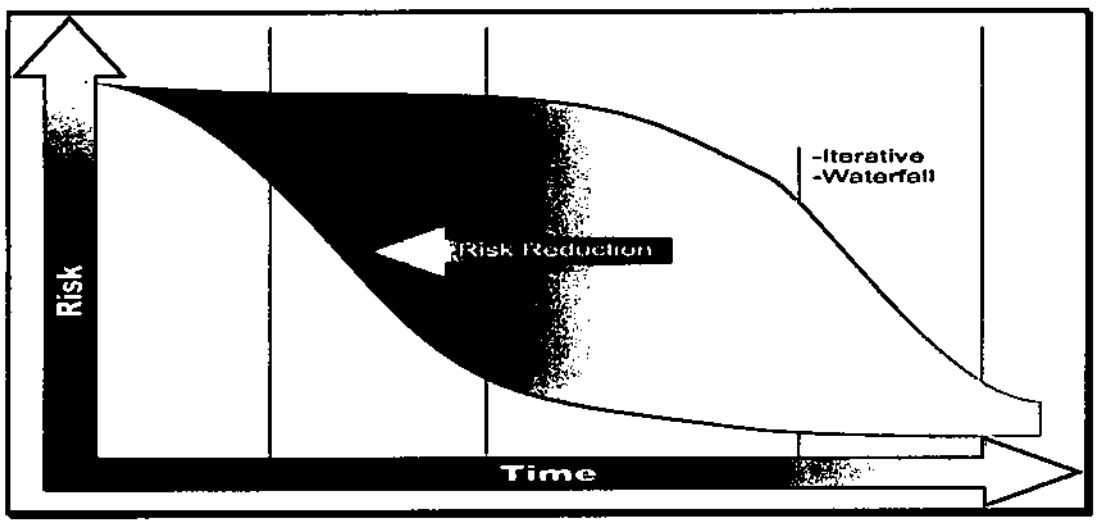

Figure 35: Risk Reduction Profile for Waterfall and Iterative Development [16]

- "If you don't actively attack the risks, they will actively attack you". One of the prime benefits of the iterative approach is that it allows the team to identify and address risks early in the project.

  If risks were unaddressed, it would imply that the organisation would be investing in a faulty architecture and a non-optimal set of requirements, which would result in a bad set of software economics. In order to accurately estimate the duration of the cycles or phases, it would be an imperative to address the risks upfront.

  At the beginning of the iteration, the iterative process requires the team to make, or revise a list of top risks. Each risk then requires a prioritisation, before the team mitigates the risks. Usually the top three risks of the phase are addressed. An example of the top three risks would be:

RISK	MITIGATION OF RISK
Based on past experience, we are afraid that the user community will not understand what requirements we plan to provide, and as a result they will require changes after the beta release of the software	As the use-cases are developed, complement them with user-interface prototypes. Ensure that we setup a 'walk through' to explain each use-case – i.e. create a storyboard. Once complete, ensure that we obtain user-signoff. Keep the user base informed during the progress, and provide them with early prototypes or alpha releases.
We do not understand how to integrate with legacy system X.	Build an actual prototype that shows us how to integrate with the legacy system, and ensure that the prototype actually integrates. Throughout the project, ensure appropriate testing of the integration. An alternative would be to eliminate the integration from the scope of the project – called 'risk avoidance'.
We have no experience in developing with Microsoft.NET technologies.	Plan for the team to be skilled with the relevant technologies, and budget for a consultant with the relevant skills to assist with the subsequent phases, e.g. 3 days consulting of e MS.NET consultant.

Table 8: An Example of Risk Mitigation

Once the risks have been listed, the team should address them as a priority. The plan should be modified to deal with the risks and they need to be addressed from multiple perspectives, such as requirements, design and testing. The best approach is to start from a coarse solution, and successively diminish them at a granular level.

Many projects risks are associated with the architecture, and in the iterative approach, the primary objective in the Elaboration phase is to ensure that the team constructs the architecture in a proper fashion.

The list of risks will continuously change as we move across the different phases. Attacking risk is always going to be a constant battle, and in each iteration the team will be able to reduce or remove some risk, while others will grow and new ones will appear.

5. THE CHANGE CYCLE

5.1. INTRODUCTION

The only constant in the new economy is change. A continual advance in software development and Internet and communications technology, coupled with an ever increase in demand for information; have created a new, evolving business environment. This evolving business environment has been driven by requirements to deliver quality products and services faster, and the challenge is to exploit new technologies, the prospects of discovering and conquering new markets, and a growing need for a managing better customer relationships.

For the software development team, these new demands mean that we will never again be finished with a plan or product. In the past, these teams delivered new software releases on an annual basis, and in the Internet Age, it is the norm to deliver these releases several times a year, and even weekly or daily for some E-Commerce based companies.

Today, the world is characterized by a heightened sense of urgency, and organisations must understand to embrace and drive change if they hope to survive.

5.2. AN ARCHITECTURAL APPROACH FOR ORGANISATIONAL CHANGE

When it comes to change, most organisations focus exclusively on technology (new tools, processes, equipment), and ignore the modification necessary in culture and behaviour. Change by nature intrudes on people's 'comfort zones' and many associate change with pain,

whether or not they think it will result in business improvements. In majority of change initiatives, team members meet change with resistance, so it is critical for organisations to address this key issue up front in the process.

It is important to recognise that cultural and behavioural factors are pivotal for organisational change, as they are the basic building blocks that lay the foundation for it. If these blocks are not in place, any technology introductions will fail to satisfy expectations and may even produce adverse results.

Below is an organisational assessment to ensure readiness for a change initiative:

DESCRIPTION
LEADERSHIP Is there adequete leadership at all levels? Leadership is necessary for driving and sustaining change, with the ability to guide the organisation through unknown territory.
VISION & PLANNING The organisation is required to formulate a flexible plan so that everyone understands its goals and objectives. The vision end planning should identify key dependencies (skills, resources, etc) and risk factors.
REASONABLE ATTRACTIVENESS The organisation should consider and study alternative paths within a 'trade space' bounded by fixed constraints such as budgets, timeline, needs versus wants, etc.
CULTURE & BEHAVIOUR The organisation should change or be willing to change the way it is structured and in the way it thinks, distributes incentives and rewards, and approach naw challenges. All stekeholders should be communicated the urgency of implementing the prescribed plan, and the bar for performance and quality should be raised.
SKILLS, RESOURCES & PERSONNEL New talent, equipment and experience will be required to drive and sustain this change. In addition to the latest technology skills, project management, leadership, vision interpersonal skills and commitment are key. Treining programs play a key role in the overall solution.
TECHNOLOGY New technologies are always required when a change initiative is implemented in an organisation. New technology implemented should be able to scale and adapt to the changing environment, and yield a return on investment to make the effort cost effective.

Table 9: Organisational Assessment of Readiness for a Change Initiative

The requirements above will be dictated by the scope and nature of the change and the organisation's complexity.

5.3. BEST PRACTICES FOR IMPLEMENTING ORGANISATIONAL CHANGE

When adopting new technologies and processes to accelerate an organization's software development capabilities, it is critical that the proper building blocks are in place to ensure that the change initiative would be successful.

Best practices are consistently successful in reducing risk and driving change.

In the new economy, changes affect not only the organisation, but also their customers, suppliers and partners. This can be true if the change is a pervasive one that affects all components of the business.

Identify and Agree on Key Change Drivers. Change should be driven by a key business concern that has significance on the market it operates in quality or efficiency in its production process. Change should be articulated clearly in order to eliminate perceptions and negative effects. All stakeholders should support the change initiative.

Create Demand for Change; don't Mandate or Force it. If everyone in the organisation adopts to change, then a demand for the change is created at every level. This 'buy-in' is required if we expect stakeholders to drive change and withstand setbacks and hurdles. When a change initiative is mandated in the organisation, they are met with resistance, half-hearted attempts to adapt to change, false starts and enormous wastage of time, money and sagging morale.

Exercise Consistent Leadership and Communication Practices. Lack of leadership and vision will cause the change effort to stall and lose credibility. Changes to the plan are

inevitable, and it is critical to keep all stakeholders informed about when and why they are made.

A consistent message should be communicated in all directions of the organisation. In order to ensure cohesiveness across the organisation, open communication, sharing of information and assisting those to understand the common vision is vital. Communication should be kept simple so that it remains consistent over time and not promote confusion and misinterpretation.

Continually Update and Fine Tune the Vision and Project Plan. When the change initiative is planned, the vision and project plans assist management in identifying key issues and risks, establishes a communication plan, and fixes a starting point that everyone can agree and begin at. As the vision and project plan are living artifacts, it is important to remember that the plan is not fixed in stone. As they provide a roadmap for change, it is necessary to correct the course and validate its direction throughout the implementation process.

Achieve Incremental, Demonstrable Success. The best strategy is to strive for incremental, demonstrable success as the plan is implemented in stages or phases. This mode will assist in stabilizing the environment and building confidence, credibility and experience required at each phase to manage chaos and sustain performance. Involving key top management will not only increase the change initiative's chances for success, but also ensure more management attention and support.

Find Champions for Solutions. It is an imperative to identify champions to support the proposed solutions and propel the change initiative forward on the right track, as they can provide valuable insight on the working environment and team morale.

Acquire and Develop New Employee Skill Sets. The change initiative will create a demand for modern techniques, skills and mindsets. As the vision and plan is formulated, it is important to proactively identify the new skill requirements that will be required to facilitate change and operate in the new environment. A training program should be executed to assist the current skill set. Software development teams are usually highly skilled in technical skills but lack depth in the non-technical areas, such as soft skills.

Establish a Collaborative Environment. It is critical to establish a collaborative working environment among all stakeholders and partners at the outset of any change initiative. Successful organisational change requires risk sharing, a collective sense of ownership, and the ability to leverege knowledge and skills from each other. Throughout the initiative, the development project team should strive to maintain a high-trust environment that fosters collaboration.

Collect and use Metrics to Monitor Progress. It is important to capture the correct metrics on a regular basis and use this data to perform course correction as needed.

An initial metrics assessment can assist team members identify which metrics add value, learn how they are typically collected and analysed, and understand the organisation's current state. This assessment can be utilised as a benchmark for moving forward and to reach for members to assist in achieving this effort.

Overhaul Incentives. To adopt change, organisations will require modifications in behaviour, skill sets and responsibilities. Team members will require to be incentivised to make this change a success. By making the change to the incentive programs before the change process begins, will generate new energy and behaviours, which are critical to success. Incentives can

be structured around achieving milestones on or before schedule, a certain level of quality or accomplishing specific goals.

As change is a learning experience, the organisation will expect to encounter setbacks and hurdles. These should be viewed as opportunities to gain knowledge, which the organisation can build on, and it should be allowed that the team be discouraged by these or they lose focus.

When change within an organisation ceases, so does improvement and growth.

5.4. A TEAM PERSPECTIVE

In order to realize successful building of software components, an increased diversity of specialist skills both in the technical and business arena is required. In the new economy, business people are taking a more central role in software development, and I.T. resources have to apply a much greater level of business understanding. New breeds of individuals are emerging as the dividing line between business and I.T. grow blurred.

In the new world, teams are not only geographically separated, but also work for different employers with different cultures. This new emergence, presents new and interesting challenges for management. These e-business solutions that virtual teams create span easily across organisational boundaries, different geographical locations and different hardware and software platforms.

These factors add new dimensions of scale and complexity to the organisation and management of teams.

The significance of teams and team roles bring objectivity to the process in that one person may fulfill more than one role and many individuals may supply one role. Different roles, played by various individuals at appropriate times, are applied through the stages of a project. The more virtual the project, the more that the lines between business and I.T. start to blur, and the lack of role definition starts to hamper the project.

"A major reason that many of today's teams are ineffective is that they overlook the implications of the obvious. People do not make accommodations for how different reality is when they and their colleagues no longer work face-to-face. Teams fail when they do not adjust to this new reality." Lipnack and Stamps (1997)

Traditionally software project teams members worked in close proximity for the same employer within a common corporate culture that separated the I.T. and business worlds. In the mid 1980s, businesses started to downsize and streamline their operations, and so did I.T. start to downsize, as software teams started to lose their jobs or moved out of their corporate silos into much leaner and meaner, empowered teams with a focus on meeting defined business objectives.

Teams are growing increasingly to be virtual, resulting in greater individual autonomy and with great potential for communication problems. E-business makes technology a central business enabler, which implies that I.T. is no longer merely a support function. E-business and component based development involves some fundamental shifts and an increasingly blur between business and I.T. A clear focus is required to eliminate corporate software anarchy.

5.5. A SKILLS PERSPECTIVE

According to Gartner, Object-Oriented skills will account for more than 40% of new development software projects by 2005. From a South African perspective, another skill shortage stampede is on its way. Organisations will be required to take steps now to pre-empt this shortage and obtain a return on their current investments.

Object-oriented technologies moves us away from building straightforward and fairly specialized technical applications into the more challenging world of enterprise systems - grown up critical applications that will not only run the enterprise, but talk to existing applications, databases and other resources.

An object-oriented technology makes it difficult for software developers who have long practiced non-OO and procedural paradigms to make a transition, as OO requires developers to think about application logic in very abstract ways. Objects are created to accept messages and to exist in en event-driven environment, in contrast to the procedural method of making specific procedural call in a hardwired, directed fashion. Therefore, it will take time to teach this new paradigm to existing programmers. Gartner predicts that 60% of COBOL mainframe programmers will fail to switch to OO, yet it is often these people who understand the business best and have the most in-depth experience, which is en essential attribute when designing enterprise-wide applications.

Recruiting full-fledged OO skills is not going to be an option for the foreseeable future. The available OO resources currently constitute a fairly exclusive club, and most of them work for IBM, SUN Microsystems and Rational Rose, and they are not easy to lure away. The only option is for organisations to grow their current skill-set into OO skill-sets.

The grow-your-own approach constitutes extensive and expensive training for team members. The downside of training your own skill-set in a marketable skill like OO is that they tend to go off and market it to someone else. The answer is not to avoid training, but to provide an environment in which team members get the challenge and freedom that they need to stay in high spirits. By making your team members productive as possible, the challenge is to ensure that the organisation takes advantage of OO capabilities such as the re-use of components.

5.6. The Suitability of Migrating Legacy I.T. Staff

Gartner evaluated the options of transforming the various skills sets to the new OO environment. The cost estimates make it clear, that this will be an expensive exercise for organisations. However, as stressed before, organisations may want to leverage their valuable human assets by migrating some of their legacy I.T. staff to OO, as the decision to recruit cannot be fulfilled, given the lack of qualified OO software developers.

Three factors were considered in the analysis, namely:

- The *cost* of the migration;
- The *time* required for the migration to be complete; and
- The *risk* of a software development staff member failing to make the transition.

The cost factor reflects expenses and losses that an organisation will incur when migrating their current I.T. staff. It consists of training expenses, loss of productivity during training, loss of productivity during OO technology adoption, and the compensation increase required for retaining the converted staff.

The time factor reflects the duration of the entire process, including training and the maturation time.

The risk factor measures the probability of failure of achieving full productivity as a OO team member by the end of the migration time frame. It also estimates the amount of money the organisation will put at risk when migrating these I.T. staff members.

The analysis does not consider factors such as hiring qualified trainers to conduct on-site training, or qualitative factors such as unique and business experience possessed by these legacy I.T. employees, or the organisation's desire to retain such employees, if migration is the only means of doing so. External sourcing costs such as recruiting costs are also excluded.

MIGRATION TO OO SKILLS	COBOL	4TH GENERATION	C++
Transition cost (training & lost productivity)	$41 000	$38 000	$19 000
Total cost (including compensation increase)	**$57 000**	**$52 000**	**$23 000**
Original salary	$65 000	$67 000	$75 000
Total cost/original salary	90%	75%	30%
Total costs/COBOL developer's cost	100%	90%	40%
Time (duration of migration)	**12 months**	**10 months**	**5 months**
Time/COBOL developer's time	100%	85%	40%
Risk (Probability of failure)	**60%**	**40%**	**5%**
Cost of failure (transition cost x probability of failure)	$25 000	$15 000	$1 000
Cost of failure/COBOL's cost of failure	100%	60%	4%

Table 10: Migration Suitability Matrix [7]

The analysis above concludes that migrating legacy I.T. staff members will be a costly, lengthy and risky affair. Gartner further recommends that organisations assign an 'acceptance threshold' to cost, time and risk factors in the decision-making process.

An example would be:

FACTORS	FOR MIGRATION TO 'PROFESSIONAL' OO DEVELOPMENT
Cost	$60 000 or less
Time	9 months or less
Risk	30% or less

Table 11: Acceptance Threshold Matrix [7]

The OO skills shortages should be attacked from every available angle. Lateral thinking in recruitment and retention plus up-to-date training methods are necessary, but not sufficient conditions for success.

6. CASE STUDIES

6.1. DELOITTE CONSULTING (2001)

DESCRIPTION

The Western Region Solution Centre of Deloitte Consulting is an organisation focused on providing clients in many industries with effective e-commerce and e-business solutions. In this case study, they were contracted to The California HealthCare Foundation to streamline their business practices and develop an e-business solution to improve the enrollment process.

BUSINESS PROBLEM

To enroll children in the new health insurance program, parents were requested to fill out a complex 28-page booklet of forms and worksheets. Therefore completing the application alone was a barrier to coverage. A better solution to public health insurance enrollment for families had to be provided.

Deloitte Consulting was contracted to design, test and implement an on-line enrollment system for the California Healthcare Foundation, and the application had to be easy to use, highly accurate and implemented and tested within a short time frame. The software interface and back-end database had to seamlessly integrate with a legacy mainframe application. The firm had to also meet the client's budget parameters while satisfying its low risk threshold.

BUSINESS SOLUTION

To develop better e-business solutions faster and with reduced risk, Deloitte Consulting had standardized on an Iterative Development approach using RUP®. The software development approach minimized risks, increased quality through proven best practices, and provided an enhanced time to market the solution.

The following challenges were presented to the consulting firm:

- The existing paper forms were complex and difficult to use, and they did not accurately capture some key information. This necessitated redesigning the application to create an on-line version that would be highly user-friendly, while capturing a wide spectrum of complex data. The application also required error checking processes to increase the quality and validity of the data as well as security and confidentiality of the data that was collected.
- The user community consisted of two groups, namely institutional users who assisted and enrolled people on a regular basis, and beneficiary users that self-enrolled themselves directly on the Internet. The forms and procedures had to be effectively developed for both groups.
- The new e-business application would have to be seamlessly integrated into the State of California's legacy mainframe system.

KEY BENEFITS

The application was developed using the iterative approach in a three to four month development window, meeting the fast time to market. Risk was reduced utilizing the proven process based on expert knowledge and industry acknowledged best practices. This was the first time that Deloitte Consulting developed an e-business application for such broad use, which

was based on developing components. The iterative development process offered the client both high quality and fast development, eliminating the software paradox.

6.2. ARINC INCORPORATED (2000)

DESCRIPTION

ARINC Incorporated develops and operates communications and information processing systems for the aviation and transportation industries and provides the systems engineering and integration solutions to the government and industry. They have their headquarters in Annapolis, Maryland, with over 2800 employees worldwide.

BUSINESS PROBLEM

ARINC embarked on a project that was critical to maintaining their leadership position, of the development of a state of the art digital communication network that allowed commercial aircraft around the world to send and receive messages concerning flight information, scheduling and operations, thereby providing increased safety and efficiency for passengers. The systems development component had an expected duration of 5 years and this effort required more than 100 software development resources.

A year down this critical project, the product development director realised that he would need to make some changes. The project was using the waterfall approach for software development, and a large amount of time was consumed to capture, refine and document their requirements. This very large and complex software system did not have a scalable process to attack the issues, as the team was not effective at formulating requirements or at developing a

design and architecture to satisfy these requirements. Therefore project management recognized that the existing software development process was not scaled to meet project costs and scheduled demands.

BUSINESS SOLUTION

The team was introduced to iterative development to solve the basic problem of attacking issues at hand by breaking down into manageable pieces that could be handled one at a time. The team was able to put a viable working process in place, and demonstrate it, in just three months. As RUP® provided the team members with a common vocabulary of a well defined and proven set of terminology, communication problems between individuals and groups were quickly and greatly reduced. Abstract modeling forced the team to get out of the quagmire of technology and detail, and think about the solution from a higher level, allowing the team to develop a pure, more robust and simpler business solution.

The team initially started with a three-week iteration to teach a small group of key project members about the activities involved in iterative development. Subsequently, the iterations consumed seven-weeks addressing relatively, simple use-cases.

Another key benefit was the ability to identify and define reusable components, which significantly reduced the duplication of effort and provided the team with a much more streamlined architecture. Furthermore, with each iteration, the architecture became clearer, and the system evolved into a complete system.

The project remained on track with full deployment in 2002.

KEY BENEFITS

A summary of the key benefits attained:

- Improved communication between the business and technical team members;
- Tangible results, in terms of working software, that was demonstrated early in the process;
- The reduction of key project risk areas;
- Process improvement with each iteration; and
- Identification and definition of reusable components, eliminating coding duplication.

6.3. AU-SYSTEM (1999)

DESCRIPTION

AU-System is Sweden's largest independent software development and consulting firm for wireless software solutions, systems integration and intranet and Internet solutions.

BUSINESS PROBLEM

Consumers in Sweden were uncomfortable using personal credit cards for Internet purchases. Their reluctance, shared by people in other Scandinavian countries, had hindered the growth of e-commerce in these regions when compared to other parts of Europe and the United States. Credit cards have become the most widely used payment method for Internet transactions between merchants and consumers.

MeritaNordanken, a leading Scandinavian bank with 6.5 million private customers and 400 000 commercial customers, sought to remedy this situation by establishing an e-commerce payment system based on secure account transfers instead of credit cards. AU-System was requested to design and implement the software system.

The approach that the client had was quite simple. Both merchants and consumers would have access to their respective MeritaNordbanken accounts over the Internet. To make a purchase, buyers would enter their password-protected account and transfer the appropriate amount of money into the merchant's account via a user friendly and familiar web-interface. The buyer could thus make payment without placing any sensitive information onto the Internet and the Bank would guarantee payment to the merchant.

BUSINESS SOLUTION

AU-System was handed a formidable list of project specs, which included a very simple user interface for buyers and sellers that would work with all common Internet browsers, a requirement that the application would integrate into the Bank's legacy mainframe system, and robust security and encryption was required throughout the entire process. A further requirement was that the client called for development, testing and roll-out to be complete within a three-month time frame. Another restriction was the limited access to the back-end integration, as development took place during the same period the bank was securing their data systems for Y2K.

A further problem that the development team had to deal with was the Swedish summer holiday season falling right in the middle of the software development schedule, as team members were entitled to their four week vacation. In order to meet the project delivery date new team members were introduced and brought up to speed in a very short order.

The iterative development process allowed flexibility in providing design changes right in the middle of the project and introduced new people into the team. RUP® made it easy for people to be introduced to the code, obtain a logical view of the development, and then quickly continue with the software development.

The overall problem was divided into a number of well-defined smaller problems, and in each iteration the top priority problems were defined, analysed, implemented and tested. Problems were thus identified and addressed early on the in the project, shortening time-to-market and greatly reducing the risk and expense of fixing major bugs after the release.

The use-case-driven approach described the software functionality from the end user's point of view, and ensured that they were properly understood, documented and implemented in the software.

KEY BENEFITS

A summary of the key benefits attained:

- The iterative development process provided for flexibility of the total design, as design changes were made in the middle of the project;
- New people could easily be integrated into the development team;
- Components were defined and re-used, eliminating duplication; and
- The iterative approach minimized risks from project inception.

6.4. VOLVO INFORMATION TECHNOLOGY (2000)

DESCRIPTION

Volvo Information Technology is a wholly owned subsidiary of AB Volvo, one of the largest industrial groups in the Nordic Region. Volvo I.T. provides all types of I.T. solutions in a variety of technical environments. The company was formed in 1998, through a merger of all I.T. resources from the different Volvo Group companies. Volvo I.T. provides the Volvo Group and other selected customers with cost-effective I.T. solutions resulting in long-term business value.

BUSINESS PROBLEM

Like most other companies in the I.T. business, Volvo I.T. faced new challenges that required changes in the way they developed, delivered and maintained software applications. Some of their challenges were:

- The applications were becoming more and more integrated with the business, therefore the application development process had to be integrated with the business engineering process;
- The pace of business changing is ever increasing, therefore Volvo I.T. had to improve productivity in order to respond to new and changing requirements; and
- Customers required global solutions, and this has lead to an increase in number of projects that are distributed over several countries and continents.

They defined their long term goals and required a new framework that should encompass the following (Method Strategy level):

- Well-defined business requirements as input to the application development environment;
- Better product fit to actual needs at time of delivery;
- Shorter lead time until delivery of first version of the application;
- More projects on budget and within the prescribed time;
- Reduced rework costs – reusability of code, if defined as components;
- Better product maintainability; and
- One common process for development of software applications.

BUSINESS SOLUTION

In order to implement RUP®, their efforts worked on three layers:

- The Method Strategy level;
- The Process/Method Development level; and
- The Application Development level.

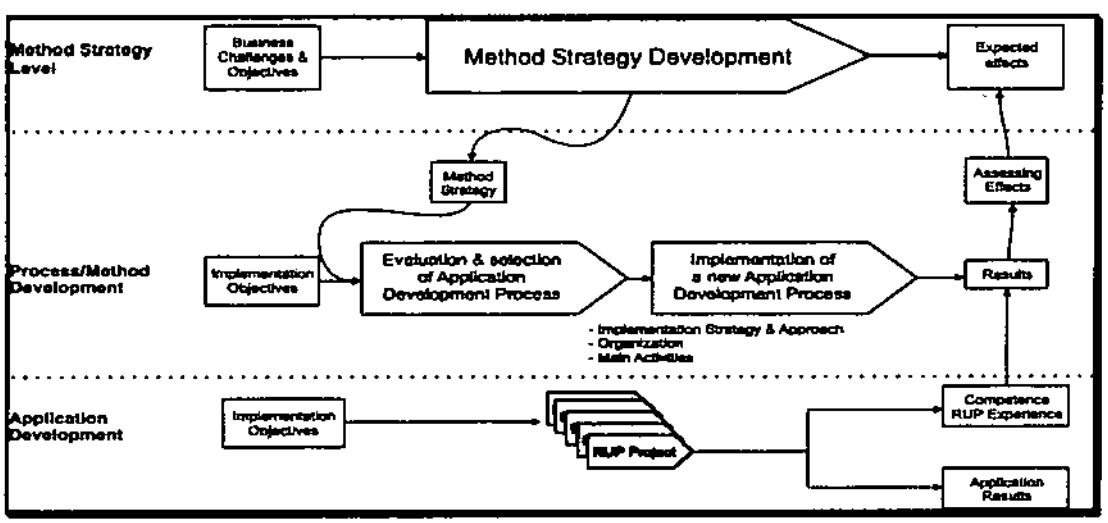

Figure 35: Volvo I.T. 's Implementation Approach [13]

At the Method Strategy level, they focused on the business challenges and objectives. They realized that implementing or changing the application development process in a business was a task with lots of traps. They defined the following critical success factors (CSFs) for the migration to the new process:

- Commitment from management was absolutely critical and that they understood the extent of change the company was to go through;
- In-house process engineering skills was a central part of Volvo I.T.'s business, and therefore it was important to have a centralized controlled process;
- There should be integration between the application development process, its methods and its tools; and
- The process had to be easy to distribute as it continued to evolve.

The Implementation Strategy component involved the company to setup a project for implementation of RUP®. They realized that it would be impossible for the company to start using the development process in all new projects, and decided to stage the implementation. In order to facilitate the cultural shift and gain experience in the new process, a few pilot projects were selected, and based on that experience; more projects were implemented with the new development approach.

KEY BENEFITS

A total of approximately 10 million US dollars was invested in the 1000 people that work for Volvo I.T. A questionnaire was developed to evaluate the iterative process, and an evaluation was done with customer representatives and the project team, at the end of each project.

After 9 months into the process, the evaluation was consolidated and the following savings were presented:

- Project schedules were reduced by 17%;
- Project effort was reduced by 46%; and
- Defects were reduced by 51%.

The Volvo I.T. approach predicted that they will reach Capability Level 3 in CMM within an 18 month window.

6.5. FORD FINANCIAL

DESCRIPTION

Ford Financial is a wholly owned subsidiary of Ford Motor Company, with 200 branch offices servicing 12 500 dealers. They are a global leader in meeting the financial needs of customers through product and service offerings. The company is also the largest non-government debt issuer, providing high-quality bond offerings to investors. They currently employ 20 000 people and service more than 10 million customers worldwide.

BUSINESS PROBLEM

Over the past decade, Ford Financial had seen how the Internet has affected the automotive industry and automotive financing houses. Internet technology has lowered the entry barriers for small services companies and for companies born on the Internet, resulting in increased competition and lower margins. Vast numbers of clients look to the Internet for comparative financial research, thereby expecting an easier and faster loan processing and customer

service. The company was determined to exceed the pace of technology and industry change and revolutionise its business via e-business initiatives, and developed a cohesive e-business strategy to leverage its market share, distribution systems, and breadth of product offerings. A proviso was that its existing legacy I.T. infrastructure and back-end systems would be leveraged to provide CRM and e-commerce services to its customers.

The company identified several key business drivers for its e-business strategy:

- Retain platform independence. The company did not want to be locked in to a single vendor for hardware or software, and required the flexibility to develop best-of-breed solutions based on a variety of products;
- Integrate new services with legacy back-end systems. They did not want to spend time and money on migrating the data to a new environment and instead write code to integrate Internet-based applications with this data;
- Lower distribution costs, as they had high overhead costs for distribution of software updates to its dealers via CD-ROM.
- Increase margins, improve employee productivity, and strengthen customer relationships. The architecture needed to support new and future applications that would e-enable paper based, manual processes. This would enhance the efficiency of loans, customer inquiries and credit application processing; and
- Shorten development lifecycles for new features and functionality. Their current development lifecycle was between 8 and 12 months, and they required it to reduce to 4 months for product and feature roll-outs.

BUSINESS SOLUTION

The business problem was divided into 4 large-scale initiatives:

- Workflow – a mission-critical, enterprise, business-to-employee (B2E) Intranet service that delivered enterprise wide information such as customer account payments, delinquencies and repossessions.
- Used Vehicle Information System (UVIS) – a business-to-business (B2B) system that interfaced with external auctions and shipping companies to facilitate the pickup, sale and transportation of used vehicles in which Ford Financial had financial interest.
- CreditWeb, a B2B service that enabled Ford Financial and auto dealers to enter and transmit EFT transactions to their branch offices; and
- International Account access – designed to enable customers from all over the world to access their account information online.

KEY BENEFITS

Some of the business benefits that have been documented:

- Faster time to market. The iterative development approach and reuse of components helped the company meet aggressive and compressed implementation schedules, reducing application development cycles from an average of 8 to 12 months to 3 to 4 months. The development methodology also assisted in streamlining and boosting development efficiency while concurrently reducing risks. Ford Financial has been able to reduce development costs by reusing approximately 70% of components as well as speed time to market by reusing approximately 95% of its architecture design;
- Reduced costs. On-line self-service allows customers to perform activities quickly, off-loading calls to the Ford servicing centres. Ford Financial has reduced costs by as much as 30%;
- Integration with legacy systems. By transitioning to an open-standards-based platform, Ford was able to leverage the data stored in legacy back-end systems

without expending time and money migrating the data to a new environment and writing code to integrate Internet-based applications with this data; and

- Increased productivity, through a combination of best practices and process learning.

7. CONCLUSION

A future analogy:

The cars rolling off production lines today are the result of some four years design and development work. As recently as the early 1990s, however, getting new models from the drawing board to volume production took twice as long.

The trend towards ever shorter development cycles is set to continue, and by around 2006, leading automobile manufacturers aim to have their latest models on the road in less than two years.

It is an ambitious goal, and one for those I.T. departments to provide effective support. At CeBIT 2002, DaimlerChrysler showcased its E-Factory solution, which integrated development work, production planning and actual production processes. As the name suggests, E-Factory enables manufacturers to model resources at production plants, simulate and review the entire product development lifecycle from end to end, helps improve the quality of production planning, cuts costs for design changes and reduce production lead times.

The key objective is to synchronize planning activities, ensuring that all participants in the manufacturing process can digitally exchange production and capacity data, whenever it is required. Today companies such as DaimlerChrysler increasingly rely on powerful information technology and telecommunications solutions for the entire lifecycle of their vehicle.

We can summarize that CBD has the following benefits for an agile organisation in the new economy:

7.1. OVERCOMING RESISTANCE

Project Management. It is surprisingly easy to get agreement based on the logical argument that CBD is beneficial to I.T. and the organisation, as it enables reuse, brings flexibility to the development project, can reduce development time and eventually costs to the project. The difficulty lies in turning the agreement into tangible action, or to sustain the action once it is initiated. This is especially true when using CBD means migrating people, processes and tools to an unfamiliar approach. Building a strong business case based on cost and time savings will have a positive effect on most project managers and most of all the business sponsors.

Business Sponsorship. Resistance to implementing component architecture can sometimes come from project managers under pressure to deliver. If the organisation is serious about balancing project and strategic aims, then CBD must have the sponsorship of senior executives who can provide this balanced perspective.

Persuasion. The ideal scenario is to have persuaded business and I.T. members of both the benefits and practicality of CBD. An approach would be to be persistent and consistent in communicating the benefits, and complemented with proof that this iterative development process works and brings positive benefits. Examples of recommendation would be technical references and pilot projects.

Industry Awareness. The use of component architecture frameworks (J2EE and COM+) is now standard in the industry. These frameworks are based on OO, and these component-based systems represent industry-wide best practice.

7.2. BUSINESS AND I.T. ALIGNMENT

A common understanding of components is vital for both business and I.T. Reusable components cut across business areas and organisational boundaries, and technology cannot alone solve these issues. Alignment of team members should be around architecture, rather than just projects, which will demonstrate the commitment to the understanding of CBD.

One way of reducing the risk inherent in moving towards a component-based approach, is to share the risk by partnering with vendors. These vendors can provide particular expertise, may have done similar development work before, and view success to be in their best interest.

Iterative development is not a magical wand that when waved solves all possible problems or difficulties in software development. Projects are not easier to set up, to plan or to control just because they are iterative. The development team will actually have a more challenging task, especially during the first iterative project, and when risks are high and early failure possible.

Eclipse (www.eclipse.org) is a project providing open-source development of robust and industry wide components. The organisation consists of a consortium of major I.T. vendors who formed to develop a better development environment and share their knowledge and practices among them. The end-result would be a library of common components that will be open-source and platform independent.

This report has highlighted the benefits of object orientation as a software development approach, and the practical examples have demonstrated tha following:

- Shorter software development cycles;
- A more maintainable product;

- Reduced development costs per project, which has a relative economic advantage;
- Reduced lifecycle costs;
- The change implication, competence and team perspective that is essential;
- More flexibility; and
- Interoperability between applications, hardware and business frameworks.

Lastly, the next evolution of I.T. architecture will still be based on component architecture, with a stronger emphasis on integrating legacy systems and on a common services layer to support application delivery for a wide variety of users, channels and devices. The demands and visions of industries such as the automobile industry can be enabled by CBD.

8. DEFINITIONS

- I.T. – Information Technology
- ICT – Information and Communication Technology
- WTO – World Trade Organisation
- OECD – Organisation for Economic Cooperation and Development
- BSP – Business Service Providers
- ROCE – Return on Capital Employed
- CBD – Component Based Development
- ISV – Independent Service Vendors
- UML – Unified modelling language
- OMG – Object Management Group
- www – World Wide Web
- IDE – Integrated Development Environment
- EJB – Enterprise Java Beans
- ActiveX – A type of component that uses COM technologies to provide interoperability with other types of COM components and services.
- COM – The Component Object Model is a software architecture that allows applications to be built from binary software components. COM is the underlying architecture that forms the foundation for higher-level software services.
- XML – Extensible Markup Language, designed to improve the functionality of the Web by providing more flexible and adaptable information identification.
- B2C – Business to Consumer

- B2B – Business to Business
- GUI – Graphical User Interface
- OCL – Object Constraint Language is part of UML, and is applied during the analysis and design phases to identify the conditions under which an object will provide a service (operation) and the result that occurs if these conditions are fulfilled.
- ORB – Object Request Broker
- 24*7*365 – a service or application that is available 24 hours a day, 7 days a week for 365 days a year.
- VSAM – Virtual Storage Access Method, a mainframe storage method.
- CMM – Capability Maturity Model
- SEI – Software Engineering Institute
- RUP – Rational Unified Process
- CSF – Critical Success Factor
- OO – Object Orientation
- CeBIT – The world's largest annual trade show for information and telecommunications technology.
- J2EE – Java 2 Enterprise Edition, an IBM/SUN component programming model implemented in application server products.
- COM+ - An extension to Component Object Model (COM) that allows developers to create and use software components in any software language, using any tool.

9. REFERENCES

1. McConnell, S., 1996, Rapid Development: Taming Wild Software Schedules, Microsoft Press

2. Yourdon et al, 1995, Mainstream Objects – An analysis and design approach for business, Prentice Hall

3. Jacobson et al, 1995, The Object Advantage – Business Process Reengineering with Object Technology, Addison-Wesley

4. Satzinger et al, 2001, The Object-Orientated Approach – Concepts, system Development, & Modeling with UML, Course Technology

5. Gates Bill, 1999. Business @ the Speed of Thought, Warner Books

6. Grulke et al, 2000. Ten lessons from the Future, @One Communications

7. Gartner Group – www.gartner.com

8. Metagroup – www.metagroup.com

9. The Source for Java(TM) Technology - http://java.sun.com/

10. Javaworld.com - www.javaworld.com

11. Bitpipe – www.bitpipe.com

12. Allen Paul 2001, Realizing E-Business with Components, Addison-Wesley

13. Rational Rose – www.rational.com

14. Martin, J. Principles of Object-Orientated Analysis and Design. Englewood Cliffs, New Jersey: Prentice Hall, 1993

15. Coleman, D. et al. Object-Orientated Development: The Fusion Method. Englewood Cliffs, New Jersey: Prentice Hall 1994

16. The Butler Group – www.butlergroup.com