DIRECT CAR INSURANCE Relational Database Conversion Project Implementation Guide

Jack L. Worsham

capjack@bellsouth.net
Table of contents

DIRECT CAR INSURANCE Project Executive Summary

4

Analysis of the implementation guide

Background 5
General development approach 6
Waterfall 8
Incremental 8
Spiral 9
Incremental vs. Spiral 9
Agile 10
Scrum 11
Extreme programming 11
Which is best for this project? 12

Requirement analysis and definition phase 13
System design phase 14
Program design phase 14
Writing the program phase 15
The unit testing phase 15
The integration testing phase 16
The system testing phase 17
The system delivery phase 17
The maintenance phase 18
Reference List 19

Appendix I Implementation Guide 20
DIRECT CAR INSURANCE Relational Database Conversion Project

Executive Summary

Identifying ITT’s Network Problems

Recent staff and customer surveys have indicated two serious network problems:

1. Slow Response Time
2. Inflexible Database Management

ITT’s backbone network comprises of one dedicated T3 line. When the bandwidth of one T3 line is divided among 85 plus schools this leaves a throughput of less than 3.1 kbps (kilo bits per second) for each network host. Most home dial-up lines are rated for 56 kbps and this is generally considered frustratingly slow. This has led to customer and staff frustration.

A second problem consists of ITT’s database management system, which is composed of several small flat databases. Each database is independent and contains redundant data making them extremely inefficient and costly. If multiple pieces of data is required different domains must be entered to collect the data, this requires access through different departments. Each database has its own security system with varying degrees of effectiveness; this gives hackers more chances to break into ITT’s confidential data. This degree of security presses federal security guidelines as well as adding to staff frustration.

Proposed Project’s Solutions:

One programmer and one network designer from each school can be used to convert ITT’s current flat databases to one relational database using Oracle and MySQL as well as replacing the current T3 with a T4 line, replacing the current headquarters router with a high-speed core router, and replacing selected switches with faster layer 4 switches.

Proposed Project’s Results

Using a single relational database will be much easier to secure, will eliminate duplicated data, and will increase data retrieval efficiency. Changing from a T3 to a T4 line will increase network bandwidth six fold. Changing the headquarters distribution router to a high-speed core router will increase the speed of the WAN. Changing the layer 2 switches to more efficient layer 4 switches will increase the speed of the LANs.

ITT’s new network will experience improved response time, increased efficiency, and improved database security.
DIRECT CAR INSURANCE Relational Database Conversion Project

Analysis of the Implementation Guide

Background

The DIRECT CAR INSURANCE relational database conversion project was created to solve problems produced by using multiple flat databases. DIRECT CAR INSURANCE is a school system with over 85 campuses nationwide. Each of these campuses is composed of several departments, for instance, education, finance, career services, recruiting, and administration. Each department maintains its own database. Information in these databases cannot be shared between departments, even though some of the information is duplicated. If a person from the finance department needed to know the attendance of a student they would have to get someone from the education department to look up the information for them. These multiple databases also present a network security problem; many separate databases raises the possibility that one of them is weakly secured. Thus, the use of multiple independent flat databases has created three problems: duplicate information storage, lack of intradepartmental communication, and database insecurity.

The first problem, duplicate information storage, wastes hardware and software resources. The total amount of storage needed for duplicate information is well over twelve petabytes. This much unnecessary storage costs the company millions annually. The second problem, lack of intradepartmental communication, leads to work inefficiency. The amount of time and effort wasted tracking down someone to access needed information could be best put to use doing more productive tasks. Improved intradepartmental interaction could also save the company millions of dollars annually. The third problem, database insecurity is a very serious problem from many viewpoints. A company’s data is its lifeblood. The integrity of that data must be guarded. Even the appearance of corrupted data can destroy a company. Federal mandated guidelines like Sarbanes-Oxley has made database security a more serious undertaking by imposing strict new rules and penalties. Multiple databases mean multiple points that must be secure. The more points that need to be secure the higher the likelihood that a point will be vulnerable. Hackers will exploit any network weakness to steal or corrupt data. Database security may not only cost the company millions of dollars but cause it to go out of business.

A properly designed relational database can add interdepartmental communication flexibility, reduce the amount of information storage cost, and improve database security. In short, converting to a relational database can save the company millions of dollars, improve job efficiency, and reduce the likelihood of database corruption.
The Relational Database Conversion project has been approved. The project will have a seven member oversight committee, a project sponsor, a project manager, two person quality assurance committee, a project accountant, 90 network design engineers including one lead network designer, and 90 programmers including one lead programmer.

It became apparent that before the project team could begin converting the existing database an implementation guide was needed to ensure consistent quality and compatibility. The way DIRECT CAR INSURANCE is functional structured, the extreme distance of the campuses, the complexity of project team communications, and the types of problems to be solved made it impossible to find an off-the-shelf implementation guide. Therefore a customized guide was developed in-house.

The following sections contain an analysis of the choices made for each of ten different sections of the implementation guide. The ten sections of the implementation guide are:

1. General development approach
2. Requirements analysis and definition
3. System design
4. Program design
5. Writing the programs
6. Unit testing
7. Integration testing
8. System testing
9. System delivery
10. Maintenance

**Analysis of the General Development Approach**

Many different development strategies were measured against the unique needs of the DIRECT CAR INSURANCE relational database project. The following software development methodologies (models and techniques) where compared and evaluated:

- Development Models
  - Pure Waterfall (non-iterative)
  - Incremental
  - Spiral
  - Agile
    - Scrum
    - Extreme Programming
  - Development Techniques
    - Prototype
    - Cleanroom
    - Object-oriented

A methodology is composed of one of the software development models used in conjunction with one or more techniques. Any mixture of techniques may be used whole or in part with any model. Each model has its own advantages and as Michelle Davison in her article,
“Addressing software quality issues with development models” states, “It is up to software makers and their development teams to determine which model best fits their needs.” (Davidson, 2007)

Prototyping can help the development models by communicating the current state of the product. It is the process of creating an incomplete model which can be used to let users have an idea of the completed program. The designer can use the prototype to obtain feedback from the users early in a project.

While the need for feedback is vital, the ability to comply with that need is limited by costs. Sometimes the costs of producing prototypes are prohibitive. The goal is to get the best feedback that the project can obtain. Darryl Green in his article, “A survey of system development process models,” stated that prototyping consists of the following steps:

- **Requirements Definition/Collection.** Similar to the Conceptualization phase of the Waterfall Model, but not as comprehensive. The information collected is usually limited to a subset of the complete system requirements.
- **Design.** Once the initial layer of requirements information is collected, or new information is gathered, it is rapidly integrated into a new or existing design so that it may be folded into the prototype.
- **Prototype Creation/Modification.** The information from the design is rapidly rolled into a prototype. This may mean the creation/modification of paper information, new coding, or modifications to existing coding.
- **Assessment.** The prototype is presented to the customer for review. Comments and suggestions are collected from the customer.
- **Prototype Refinement.** Information collected from the customer is digested and the prototype is refined. The developer revises the prototype to make it more effective and efficient.
- **System Implementation.** In most cases, the system is rewritten once requirements are understood. Sometimes, the Iterative process eventually produces a working system that can be the cornerstone for the fully functional system. (Green & DiCaterino, 1998)

Prototyping, however, can lead to false exaggerated expectations that the project is finished. Developers can use prototyping to become lazy. Both of these conditions need to be avoided.

Cleanroom emphasizes defect prevention rather than defect removal. Iterations are tested until quality standards are not reached which signals the need to return to the design phase.

Object-oriented techniques emphasize modularity and re-usability of code. Each object is independent and capable of receiving messages, processing data, and sending messages. Object-oriented techniques develop module libraries.

The development techniques of prototyping, cleanroom, and object-oriented design will be incorporated with one of the following development models.
Waterfall

Sandip Jorwekar, in his article, “Waterfall software development model,” outlines the different steps involved in the waterfall development model as:

1. System/Information engineering and modeling
2. Software requirements analysis
3. Systems analysis and design
4. Code generation
5. Testing
6. Maintenance (Jorwekar, 2005)

Waterfall places the emphasis on completing a phase of development before proceeding to the next phase. When a phase completes, the development is halted and only a formal project change procedure can inject new ideas into the product. Formal reviews are attended by all stakeholders at the end of each phase. Adam Kolawa highlights the time to use waterfall in his article, “Which development method is right for your project”, when he states, “If you’re working on a somewhat traditional project where the features are defined and it’s possible to freeze the specification, the waterfall process is your best bet.” (Kolawa, 2007) Once the design is set, waterfall projects progress from implementation, design, integration, and testing. The waterfall method has merit if the project’s scope is well defined and the project does not vary.

Knowing that a project will be the same tomorrow as it is today has some merit. It should streamline communications and it does help guard against scope creep. Making sure that each phase is 100% complete before proceeding to the next phase will help coordinate a projects efforts. Waterfall’s emphasis on documentation ensures continuity if the project members change. Many hardware projects use the Waterfall model; because it is extremely difficult to change hardware components once the ‘box’ is built. Unfortunately, Waterfall does have weaknesses; this approach however has a tendency to kill innovation. It is nearly impossible to predict ahead of time exactly what the customer will want; even in traditional projects the implementation phase can bring new feature ideas. One serious drawback to waterfall is the fact that there is little to no feedback until the end of a project phase.

Incremental

Reed Sorensen in his article, “A comparison of software development methodologies,” defined the incremental development model as, “Waterfall in overlapping sections.” (Sorenson, 1994). Incremental development has the stability of waterfall with the added advantages of early functionality. A project may start with general objectives, in other words, the project does not have to be as completely defined as Waterfall before the team can start coding. However, general objectives can be uncomfortable for management. Also, some modules will be
completed long before others pointing out the need for well-defined interfaces between modules and the greater need for module coordination.

One of the incremental method’s strongest advantages is that it is an iterative process. Bill Walton made a strong case for iterative approaches when, in the article “Iterative vs. waterfall software development: Why don’t companies get it?” he stated, “Innovation drives the need for feedback because the development team’s job is not just to produce a product. It’s to produce a product that the customer likes, that satisfies a need or desire.” (Walton, 2004)

Spiral

Spiral is an iterative development model that develops the final product in increments with the added component of risk assessment. Nilesh Parekh, in his article, “Spiral model - a new approach towards software development,” lists the steps involved in the spiral modeling process as:

1. Plan
2. Risk Analysis
3. Engineering
4. Customer Evaluation

The article, “The spiral model as a tool for evolutionary acquisition,” defines the spiral development model as, “A risk-driven process model that is used to guide multi-stakeholder engineering of software-intensive systems. It has two main distinguishing features; a cyclic approach and anchor point milestones.” (Boehm & Hansen, 2001) The spiral model solves the most significant risks during each cycle of development. The milestones offer a means of judging progress. Typically the spiral process is intended for large complicated projects.

Incremental vs. Spiral

Mr. Kolawa went on to describe when to use the spiral development model and when to choose the incremental approaches:

If you’re working on a project whose nature and scope are vague or unpredictable, the spiral process produces many short development cycle iterations. These iterations are used to refine the projects features. In contrast, the incremental process has short development cycle iterations but implements different features during each cycle. (Kolawa, 2007)

The spiral process works best when the project’s scope is expected to constantly change. Incremental approaches work best when the scope is defined but the product is going to be changed by continually adding features. Both types have short iterations which allow the customers the opportunity to observe their latest requests. This feedback can be used to help
guide the project along. These approaches have the customer in mind during every phase of project development.

**Agile**

Agile is a set of processes for software development, that use both iterative and incremental techniques and relies on self organizing, self managing, cross functional teams. Michelle Davidson defined agile software development as, “a methodology for the creative process that anticipates the need for flexibility and applies a level of pragmatism into the delivery of the finished product. It focuses on keeping code simple, testing often and delivering functional bits of the application as soon as they’re ready.” (Davidson, Addressing Software Quality Issues With Development Models, Methods, 2007) The emphasis is to build upon small customer approved parts instead of presenting one large product at the end of the project. The Agile development model is driven by the Manifesto for agile software development which emphasizes customer satisfaction and face-to-face conversation.

Michelle Davidson, in her article, “Agile development best for delivering products on target,” states, “software makers must get away from waterfall development, because a significant amount of time is spent gathering user requirements and creating a requirements document. It is a pointless exercise to try to gather all these detailed requirements because they’re going to change.” (Davidson, Agile development best for delivering products on target, 2007) Agile does not spend time with user requirements but instead gathers user stories then goes directly to coding. Agile developers have smaller tasks in shorter amounts of time, obtaining frequent feedback. The article continues to state, “The point is to create the simplest possible thing now that meets the requirement and move on.” According to Colleen Frye in her article, “Agile methods bring improved software quality, but challenges remain” adds, “Much more proactive attention to requirements in the form of user stories is emphasized, there’s a very active collaboration between people who are saying what they want and communicating with developers who are building what they’re being asked to build.” (Frye, Agile methods bring improved software quality, but challenges remain, 2007) This co-location results in immediate requirements and design reviewing. Using the Agile development model may cause projects to take longer. The emphasis is on ultimate customer satisfaction and not rapid delivery. The biggest problem usually encountered was pointed out by Colleen Frye in her article “the state of software quality, part 2: the challenge of building quality into the development life cycle,” “agile methods in and of itself does not improve quality, but rather exposes poor quality and shows the consequences of it more quickly. If relationships are wrong, agile will just produce bad quality results faster” (Frye, The state of software quality, part 2: the challenge of building quality into the development life cycle, 2007)

**Scrum**
Scrum is an agile process or framework for managing agile like projects. It is a project management process based on multiple small specialized teams. According to Ken Schwaber, in the article, “Agile software development with scrum:”

Scrum is based around the idea that people can manage themselves and that the future is unpredictable and the best we can do is to make the most intelligent adaptations to it that are possible to deliver something of value. Shifting the way an organization works based on these different assumptions is massive and requires a significant change process. This is the difficulty of implementing Scrum.” (Schwaber, 2006)

Scrum requires well-trained personnel capable of self-management. The teams work together to solve common interests. The teams work in a highly interactive way, producing the product through continuous dialog, exploration, and iteration.

Ken Schwaber listed the steps involved in Scrum as:

- Initial appointment of a project manager called the “scrum master.”
- Definition and prioritization of tasks to be done.
- Planning sessions for each task.
- Daily meetings among teams.
- Identification and evaluation of potential project risks and process pitfalls.
- Execution of projects in brief, high-intensity, frequent work sessions.
- Reviews of progress and evaluations of completed projects.
- Openness to constructive criticism and ideas for improvement. (Schwaber, 2006)

The development teams tend to have a more intimate relationship with the customers. The benefits are the ability to develop and implement high value, high priority software quickly to increase your return on investment, to reduce your staff turnover, to increase your staff satisfaction and to increase the quality of your product.

Unfortunately, Scrum is a foreign concept to most American companies where a high degree of nano-management coupled with the concept of management through fear tends to eliminate Scrum as a possible development model.

**Extreme Programming**

Extreme Programming (XP), one of the agile development models, is an iterative process which emphasizes small simple steps. Tests are written before coding, which provide goals for development. Pair programming is used and coding is complete when all tests pass. These steps are repeated and the design emerges from refactoring.

Extreme programming advocates believe that it is more adaptable to changing requirements and that it is impossible to know all the requirements at the beginning of a project. Its adaptability comes from the close association of programmers with the customers. Extreme
programming increases productivity with pair programming and simple coding. It stresses simple designs, common metaphors, collaboration of users and programmers, frequent face-to-face communication, and feedback.

Software architects are still faced with the challenge of creating a reliable starting point from which to develop. XP often requires a fair amount of upfront analysis and prototyping. The iterative nature of XP can be costly. It also relies on the customer knowing exactly what they need. Therefore, in order to write the coding tests the project has to be well defined which is counter to the agile philosophy.

**Which Model Best Suits the DIRECT CAR INSURANCE Relational Database Conversion Project**

DIRECT CAR INSURANCE’s database development project would be considered a traditional type of project. This would make the waterfall model an appealing choice. It is simple and easy to manage. However, even though this project is expected to take less than three months to implement, the scope of the project cannot be fully defined. A working version appears much too late to produce useful prototypes and the risk of requirement change is so high that the waterfall model must be ruled out.

The incremental model breaks up the development cycles into smaller iterations. Working versions of the software are produced during the first iteration. This gives the stakeholders at DIRECT CAR INSURANCE the opportunity to refine their requirements early in the project. It also allows for easier testing and debugging. These advantages made the incremental model very appealing. However more attention to risk management was needed.

The agile models are small and fleet of foot. Their adaptability and simple rule structure are major advantages. However, because DIRECT CAR INSURANCE is made up of over 85 schools scattered all over the country, the agile models do not provide enough structure to ensure smooth communications and documentation.

The spiral model has the advantages of the incremental model with a greater degree of emphasis on risk management. While requirements are gathered risk is assessed alternate solutions are considered and a prototype is produced. This model gives DIRECT CAR INSURANCE the advantage of risk management, gives the customer early software to evaluate, and the project team small manageable iterations. It is an internal project and therefore time spent with the customer is perceived as valuable and not excessive or costly. With the size and complexity of DIRECT CAR INSURANCE’s database conversion project it was determined that the spiral development model was the best option.

**Analysis of the Requirement Analysis and Definition Phase**
Requirement analysis matches the project with DIRECT CAR INSURANCE’s problems and goals. If the project’s tasks are not in line with DIRECT CAR INSURANCE’s goals there would be no reason to complete the project. Furthermore, if the problems are not thoroughly understood, writing effective code will be impossible. Requirement analysis includes meetings with managers, stakeholders, and users. Raymond Lewallen in his article, “Software development Life cycle models,” highlighted the questions that need to be answered during this phase when he wrote, “Who is going to use the system? How will they use the system? What data should be input into the system? What data should be output by the system?” (Lewallen, 2005)

There are three stages to requirement analysis, these are:

1. Requirement gathering (specification)
2. Requirement documentation
3. Requirement analysis

Requirement gathering can be done many ways; holding interviews, surveys or conducting workshops. Functional and quality requirements will be specified. The objective is to determine what requirements will be fulfilled by the software. Keeping in mind the inherent weaknesses of communications all efforts must be made to design the clearest unambiguous requirements as possible. DIRECT CAR INSURANCE stakeholders will know their jobs, not the associated jargon used while programming. Also, make sure that a complete sample of stakeholders is taken, for example, management, users from different departments, students, etc...

Acceptable requirements will be:

- Verified
- Consistent
- Unambiguous
- Complete
- Feasible
- Relevant
- Testable
- Traceable

Requirement documentation will take the form of use cases (See appendix II). These use cases provide scenarios to describe how the system will interact with the user. A use case should match a task to a business goal, be detailed, and be able to be developed within one software development cycle.

Requirements analysis is used to devise a plan for implementing our software to solving the requirements. Use cases from the stakeholders will then be placed against business interests and those found to be in accord with business interest will be used to produce prototypes.
These prototypes will then be used to determine if the requirements are being fulfilled, or are the requirements that should be pursued.

The DIRECT CAR INSURANCE database conversion project will conduct interviews and surveys to produce requirements. The goals of the company will be weighed against the requirements to determine which requirements will be written into use cases. These use cases will be used to develop prototypes. These prototypes will be used to generate feedback.

**Analysis of the System Design Phase (Software architecture)**

System design is the natural result of the requirements phase. The hardware, software, architecture design is produced during the system design phase. This project’s purpose is to convert many independent flat databases to one central relational database. The hardware design will be client-server architecture with a dedicated database server. The interaction of the client-server will by described using the standardized Unified Modeling Language. The relational database management system will define the projects system design. A set of tables will be designed with relational operators that can be shared across many departments.

The hardware will be designed in a multi-tiered architecture using clients, application servers, and database servers. The operating systems used will be windows XP and Windows 2000 on the clients and Windows Server 2003 on the servers. A multi-tiered architecture will result in an output that is highly scalable. Having a common set of database servers for the network will make the data easier to secure and easier to update. All servers will have their hard drives configured for RAID level 5 to maximize access speed and data safety.

The software will be designed using a database-centric architecture consisting of a query language and a RDBMS. Three different RDBMS systems were compared as to size of network they could handle, ease of installation, modularity, and scalability. The three RDBMS systems that were compared were: MySQL, Microsoft SQL, and Oracle. Oracle was found to be easier to install and upgrade and could handle large network databases as well as MySQL. Therefore, Oracle was chosen for the DIRECT CAR INSURANCE database conversion project. SQL will be used to generate the data tables. This project will be part data-oriented and part object-oriented. Databases naturally evolve around repository designs.

**Analysis of the Program Design Phase (software design)**

Program design entails the low-level component and algorithm implementation. The result of a software requirements analysis determined that the project would be too complicated for simple flow charts. The use cases generated in the requirements analysis phase will be generated into the Unified Modeling Language which will be used to generate abstract models. UML has evolved over the years and the tools have been kept up to date. UML will be used to generate the functional, static, and dynamic views.
There will be three types of diagrams:

1. Structure diagrams that describe the objects in the system
2. Behavior diagrams that describe the actions that occur in the system
3. Interaction diagrams that describe the relationships between the different parts of the system

Examples of UML notations that will be used are:

- Use-case diagrams
- Class diagrams
- Sequence diagrams
- Collaboration diagrams
- State-chart diagrams
- OCL properties

**Analysis of the Writing the Program Phase (Computer programming)**

All current programming standards and procedures will be adhered to. All current documentation will be studied by the program manager and lead programmer to be sure that all current guidelines are not violated and that all current databases can be translated to the new format. All coding will use proper documentation, comments, and structure. If any data needs to be reformatted before using SQL, due to its portability, Java will be used.

Three different commercial RDBMSs we compared; IBM, Microsoft SQL, and Oracle. All judged to be able to handle the size database that DIRECT CAR INSURANCE maintains. Many comparison tests were studied were factors like access times and column lengths were presented and Oracle proved most favorable. The only serious problem was the prices. Here, all three present sticker shock. Once the management realized that the new system will save millions of dollars annually, the prices were easier to swallow. The RDBMS itself will be Oracle with SQL being used to generate the database tables. The UML will be used to diagram the progression of database relationships. SQL allows relatively unsophisticated users the ability to write useful programs. During the develop iterations mock data will be used to test the tables. After the format is accepted, the current data at each individual site will be converted to the Oracle RDBMS.

**Analysis of the Unit Testing Phase**

Individual installations are tested against the use case requirements. Each individual unit will be independent and offline. Mock data will be generated and accessed. The outputs generated by the inputs will be compared with the use case to validate the code. This type of testing is commonly referred to as grey box testing. It is very important to document all test performed,
the results, and any changes made to the code. Tim Burns in his article “Effective unit testing,” outlined some critical unit testing issues when he stated:

The crucial issue in constructing a unit test is scope. If the scope is too narrow, then the tests will be trivial and the objects might pass the tests, but there will be no design of their interactions. Likewise, if the scope is too broad, then there is a high chance that not every component of the new code will get tested. The programmer is then reduced to testing-by-poking-around, which is not an effective test strategy. (Burns, 2001)

A practical approach to unit testing is summarized in Charles Miller’s “six rules of unit testing.” These rules were listed in his article, “The desktop fishbowl:”

1. Write the test first
2. Never write a test that succeeds the first time
3. Start with the null case, or something that doesn't work
4. Don't be afraid of doing something trivial to make the test work
5. Loose coupling and testability go hand in hand
6. Use mock objects (Miller, 2002)

Syntax, timing, performance, and documentation faults will be the main focus of the unit fault tests. Tests will be written before coding from the use cases. The PM and the lead programmer will form an inspection team that will perform periodic code walkthroughs. This will also be used to check for code uniformity.

**Analysis of the Integration Testing Phase**

Individual installations are networked and the communication between installations is tested. Progressively larger parts of the network are connected and the database interfaces are tested. Connection to the local LAN will be used to test the interface of the new database server to the LAN. All tests performed, outputs received and expected, and any code changes will be documented. These documents will be the responsibility of the PM. The Microsoft developer network outlined integration testing in their article, “Integration Testing,” when it stated:

In a realistic scenario, many units are combined into components, which are in turn aggregated into even larger parts of the program. The idea is to test combinations of pieces and eventually expand the process to test your modules with those of other groups. Eventually all the modules making up a process are tested together. (Microsoft Developer Network, 2007)

This suggests a bottom-up testing methodology. It was compared with top-down, sandwich, and big-bang integration testing methods and was found to fit better with database testing. Oracle has some automated code analysis built-in and its use is encouraged.
Analysis of the System Testing Phase

The complete integrated system is tested. The test database will be integrated into DIRECT CAR INSURANCE’s network and tested. The results are compared with the use case to determine validity. Test cases are generated which document the tests, inputs, outputs, and expected results.

System testing consists of the following test types:

- Function
- Performance
- Acceptance
- Installation

The system function tests will be used to compare the final integrated output with the expected outputs described in the use cases.

The system performance tests will be used to check things like user response time and reliability.

The system acceptance tests will be used to verify to the system users that the new database is operating in accordance with their original goals.

The system installation tests will be used to allow the final users a chance to check the system’s operations.

Analysis of the System Delivery Phase

At the end of the testing phase of the last iteration, the completed database shall be released to DIRECT CAR INSURANCE. Each site will have at minimum one RAID 5 server 2003 with an Oracle database installed. The clients will have either Windows 2000 or XP installed. Headquarters will have, in addition, a master database that will be used to backup the individual databases. Once all the use cases have been exhausted, a period of training has completed, all documents will be turned over to DIRECT CAR INSURANCE. The PM will be in charge of coordinating the project turnover activities.

Two main functions occur during delivery:

- Training
- Documentation

User training will consist of training documents and hands on training during DIRECT CAR INSURANCE in-service. One week will be set aside to make sure everyone has a four hour block of time set aside for database training.

The following documents will be generated and stored by the PM:
Analysis of the Maintenance Phase

The maintenance phase is used to update the software for environmental and user requirement changes. Maintenance can only be done efficiently if all the other phases have had maintenance in mind. Kagan Erdil in the article “Software Maintenance as part of the software life cycle,” identified four parts to software maintenance:

1. Corrective maintenance deals with fixing bugs in the code.
2. Adaptive maintenance deals with adapting the software to new environments.
3. Perfective maintenance deals with updating the software according to changes in user requirements.
4. Preventive maintenance deals with updating documentation and making the software more maintainable. (Erdil, Keating, & Yoon, 2003)

Corrective maintenance can be used to correct design, logic, or coding errors. Adaptive maintenance will be needed whenever the hardware or operating system is changed. Perfective maintenance is needed when the requirements for the database change. Preventive maintenance can be used to improve the modular structure of the system.

The ongoing maintenance phase will ensure that the customers and users are satisfied with our final product, and help us anticipate problems that may occur, analyze changing functional, hardware, software, interfaces, and business needs, as well as implement changes.
Reference List


Appendix I
Implementation Guide

Introduction
This software development guide explains the approach and processes to be taken throughout the development of the relational database conversion project. To ensure compatibility and consistent quality all project team members will follow the procedures and methods described in this development guide.

Each section of the guide will describe the actions to be performed in one of ten different project components. These project parts will include:

1. General development approach
2. Requirements analysis and definition
3. System design
4. Program design
5. Writing the programs
6. Unit testing
7. Integration testing
8. System testing
9. System delivery
10. Maintenance

General Development Approach
Considering the size and complexity of the DIRECT CAR INSURANCE database conversion project a developmental model will be used. Many different models were tested and the spiral development model was judged to provide the best fit with business and customer needs. Therefore the spiral development model will be used as the overall guide to the DIRECT CAR INSURANCE database conversions project.

Requirements Analysis and Definition
Extensive communication between the customer and the project team is essential. With the spiral development model, the requirements analysis and definition stage gathers customer requirements and analyzes risk. Risk analysis will be used to identify alternate solutions. From these alternate solutions a prototype is produced.

Requirement analysis will be conducted in three stages:

1. Requirement gathering
2. Requirement documentation
3. Requirement analysis

DIRECT CAR INSURANCE’s requirements shall be gathered through face-to-face communications and surveys with the stakeholders. Be very careful to identify all stakeholders and all the needs of the stakeholders. Whenever possible, groups of stakeholders should hold joint meetings to uncover contradictory requirements. The most important requirements gathered will deal with the needed fields to create the database. Agreement on primary keys should also be a major priority. The PM will be responsible for coordinating the field names and primary fields to all team members.

Requirement documentation will be through use case (See appendix II). They will be used to describe how the user interacts with the database. The PM will be responsible for communicating realistic expectations and guarding against excessive scope creep.

Doug Rosenberg in his article “Top ten use case mistakes,” outlines the task of generating use cases when he states:

The task of building use cases for your new system is based on immediately identifying as many as you can, and then establishing a continuous loop of writing and refining the text that describes them. Along the way, you will discover new use cases, and also factor out commonality in usage. (Rosenberg & Scott, 2001)

The requirements generated shall be measured against business interests. Those requirements found to be in-line with business interest will be used to generate use cases, which in turn will generate prototypes. These prototypes will be tested to see if they satisfy the requirements and will be used to gather customer feedback.

**System Design**

The project installs a Relational database management system. Due to the size of the database, the Oracle RDBMS was chosen. The Oracle RDBMS will use SQL as the query language. The hardware will consist of a local installation of Windows 2003 server, and clients running XP and Windows 2000, headquarters will, in addition, install a Windows 2003 server as a company-wide backup database server. All database servers will be configured to run RAID level 5. The local servers will be used to generate crystal reports, the headquarters server will be used to increase reliability and provide database backup support. The database interface will use the existing network.

The interaction of the client-server will be described using the standardized Unified Modeling Language. The relational database management system will define the projects system design. A set of tables will be designed with relational operators that can be shared across many departments.
Program Design

The use cases generated during the requirements analysis state will be used to generate UML structure diagrams, behavior diagrams, and interaction diagrams. To assist the programmers in communications standard UML design modeling notations will be used to diagram the different software functions and interfaces.

Examples of UML notations that will be used are:

- Use-case diagrams
- Class diagrams
- Sequence diagrams
- Collaboration diagrams
- Statechart diagrams
- OCL properties

Writing the Programs

All current programming standards and procedures will be adhered to. All current documentation will be studied by the program manager and lead programmer to be sure that all current guidelines are not violated and that all current databases can be translated to the new format. All coding will use proper documentation, comments, and structure. The lead programmer will design a method of parameter naming that will be uniform throughout the project. If any data needs to be reformatted before using SQL, due to its portability, Java will be used. For the most part, all the coding will be new only the data will be reused. Due to the similarities of the schools, the coding will be uniform throughout all units of DIRECT CAR INSURANCE.

Oracle RDBMS with SQL will be used to generate the database tables using the UML as a guide to database relationships. The PM and lead programmer will assure that all individual units will generate the same data tables. All program, internal, and external documentation will be distributed to the lead programmer and PM. The PM will be responsible for keeping a master file of all documentation.

Unit Testing

A database on a computer using Windows Server 2003 will be programmed with mock data. Access of this data will be tested the results will then be compared to the use case to determine the validation of the code. Tests performed, results, and any code changes will be documented.
Syntax, timing, performance, and documentation faults will be the main focus of the unit fault tests. Tests will be written before coding from the use cases. The PM and the lead programmer will form an inspection team that will perform periodic code walkthroughs. This will also be used to check for code uniformity.

Integration Testing

The test database server will be connected to the local LAN and the interface will tested. The interface tests will be conducted first by running continuity tests then access to the database from a remote machine. The bottom-up approach will be used. This requires that the lowest level units are tested and integrated first. Then higher level units are brought into the testing. All tests and test results will be documented.

System Testing

The test database server will be connected to the DIRECT CAR INSURANCE network and tested. The test results will be compared with the use case to determine validity. All test results will be documented in a test case. The PM is responsible for maintaining all documentation.

System testing consists of the following test types:

- Function
- Performance
- Acceptance
- Installation

The system function tests will be used to compare the final integrated output with the expected outputs described in the use cases. The system performance tests will be used to check things like user response time and reliability. The system acceptance tests will be used to verify to the system users that the new database is operating in accordance with their original goals. The system installation tests will be used to allow the final users a chance to check the system’s operations.

System Delivery

After the use cases have been programmed and a suitable period of training have finished, the PM will start the process of turning the project over. The completed database system and all documents will be collected and formally turned over to DIRECT CAR INSURANCE.
User training will consist of training documents and hands on training during DIRECT CAR INSURANCE in-service. One week will be set aside to make sure everyone has a four hour block of time set aside for database training.

The following documents will be generated and stored by the PM:

- Problem request documentation
- Statement of work
- Project plan
- Software quality assurance plan
- Risk management mitigation and monitoring plan
- Software configuration management plan
- Test plan
- Requirements definition document
- Software requirements specification
- Architectural design document
- Low level design document
- Test scripts
- Test result forms
- Release notes
- User manuals
- Operator’s manual
- General system guide
- Tutorials and automated overview
- Failure message reference guide
- Programmer’s guide

**Maintenance**

The DIRECT CAR INSURANCE database conversion project is an internal project and the maintenance will conform to the four parts identified by Kagan Erdil as:

1. Corrective maintenance deals with fixing bugs in the code.
2. Adaptive maintenance deals with adapting the software to new environments.
3. Perfective maintenance deals with updating the software according to changes in user requirements.
4. Preventive maintenance deals with updating documentation and making the software more maintainable. (Erdil, Keating, & Yoon, 2003)

The project maintenance program will be guided by the definition of those four maintenance parts.
The ongoing maintenance phase will ensure that the customers and users are satisfied with our final product, and help us anticipate problems that may occur, analyze changing functional, hardware, software, interfaces, and business needs, as well as implement changes.
Appendix II
Implementation Guide

Use Case Template

| Use Case ID: | |
| Use Case Name: | |
| Created By: | Last Updated By: |
| Date Created: | Date Last Updated: |

| Actors: | |
| Description: | |
| Trigger: | |
| Preconditions: | 1 |
| Postconditions: | 1 |
| Normal Flow: | 1 |
| Alternative Flows: | |
| Exceptions: | |
| Includes: | |
| Priority: | |
| Frequency of Use: | |
| Business Rules: | |
| Special Requirements: | |
| Assumptions: | |
| Notes and Issues: | |