# Design Document template

Title of doc: Design\_teamName\_V1.?.doc

# title page

Team 3

Team 3

|  |  |
| --- | --- |
| Team Member name |  |
| Conor Brennan | Person A |
| Adam Guerin | Person B |
| Ryan Kelly | Person C |
| Will Mannix | Person D |
| Mark Mukiiza | Person E |

**Table of Contents**

page numbers

|  |  |
| --- | --- |
| **page** | Contents |
| 1 | Assignment of use cases |
| 2 | System overview/architecture |
| 3 | Description /Goals/architecture goals/ |
| 4 | System architecture |
| 5 | Component diagrams |
| 6 | Component diagrams |
| 7 | Hardware / Software Mapping |
| 8 | Tracking Entity Objects Team member initials |
| 9 | Client Server systems Team member initials /conclusion |

### Introduction

### Assignment of use cases (all)

|  |  |
| --- | --- |
| **Table of Use Case Assignment** | |
| Log In | Adam |
| Log Out | Adam |
| Check Stock | Ryan |
| Add Stock | Conor |
| Approve Stock | Conor |
| Add Work Order | Ryan |
| Assign Work Order | Mark |
| View Employees | Mark |
| QC Check | Will |
| Daily Statistics | Will |
| Sign Out Operator | Conor |
| Request Stock | Ryan |
| Check Work Order | Ryan |
| Request QC Check | Adam |
| Mark as Finished | Adam |
| Mark as Broken Stock | Conor |
| Add User | Adam |
| Edit User | Adam |
| Delete User | Adam |
| View Signed In Employees | Ryan |
| View Work Order Teams | Ryan |

### System Overview (Conor)

The Software we have developed is for a medical company called ‘Medex’ that manufacturers and produces medical equipment for hospitals. The software gives several different functions to 3 employees of the Medex Company; Floor Manager, Stock Manager and Operator. The software we have created makes it easier for each of these employees to carry out daily time-consuming tasks quicker more efficiently.

Some of the tasks that our software does include:

Creating work orders, signing in and out operators from the work order. Showing statistics of most products being made, most components used and most productive employee and much more.

The development process we used was a custom version of Unified process and Agile.

We developed the software in such a way that the more we met client and got an understanding of what they wanted, the more functionality we could put into the software.

This made things very easy for tailoring the software to whatever the client insisted on having in the program. The extensible architecture is utilised and will be discussed in later sections.

The current version of this project is version 1 and in the future, it will be easy to add further functionality if necessary. The system architecture utilises loose coupling between subsystems which will facilitate future cloud-based deployment of the BusinessRules subsystem, DAL subsystem and SQL server instance. Similarly, as a result of the extensible architecture, there is flexibility to build a range of future clients such as Web clients and smartphone applications

SOLID design principles are utilised which stand for (Single responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversion). The principles, when applied together, make it more likely that a [programmer](http://en.wikipedia.org/wiki/Computer_programmer) will create a system that is easy to [maintain](http://en.wikipedia.org/wiki/Software_maintenance) and extend over time which is very desirable for this project.

Instead of using a cloud-based configuration management system integrated with Visual Studio, we used SQL server management studio instead because we found it easier to use and felt it intergrade better than a visual studio management system.

### System architecture (Conor)

The system architecture is a loosely coupled multi-layered extensible architecture and use of SOLID principles.

We use type safe multi objects i.e. type safe lists

We also use implementation of Factory pattern to separate object creation from business logic.

The application is a Windows Desktop application written in C# and be decoupled from Business Layer and DAL. The Business Layer is written in C# and DAL is written in C# as well.

The program utilises Windows Form and Windows Presentation Foundation and it is a windows desktop application. As a result of the decoupled, extensible architecture it would not be technically difficult to add other kinds of clients at some future date.

* 1. Description – Conor

The product tracker program is like our program as you can both login with different users and manipulate data in the database. A Single-Document Interface (SDI) is a type of GUI. Any program not capable of displaying more than one document per window is considered to use an SDI interface. Examples of this include Windows Notepad, IE 6 or earlier, and the windows command prompt.

The advantages of an SDI interface over a Multiple-Document interface (MDI) or a Tabbed document interface revolve mostly around simplicity. SDIs are far easier to program, and if implemented appropriately, easier to use as well. However, using an SDI limits a program's multitasking capability.

An ADO is a set of classes, that can be used to interact with data sources like Databases and XML files.

* 1. Design Goals – Adam

The quality’s of our system we would like to optimise is the look and feel of the program, initially the program was written in windows forms which for some reason when used in conjunction with images there tends to be a lot of flickering, maybe writing the program in Windows Presentation Foundation (WPF) would solve this issue but is yet to be seen. There are also some features we would like to add to the program, when a work order is created we would like to be able to generate a barcode that when you scan would actually be the work order number, I feel with more time we could have completed this as we have researched some ways to do it. Another feature we would like to have added is notifications for the users, for example when an operator is assigned to a work order a notification will pop up in the top right of the screen. Overall, we are happy with how our program turned out we hope will aid the customer with their work.

* 1. System Architecture Goals – Conor

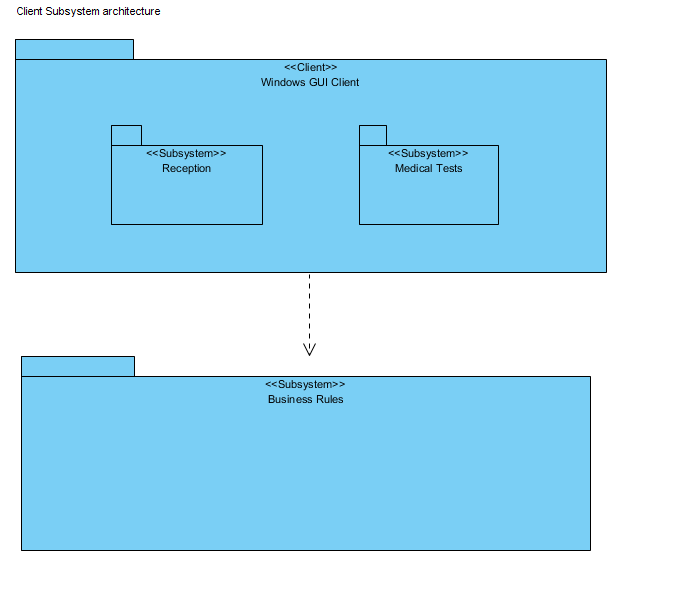
System Architecture Overview diagram. Few sentences describing assignment of functionality to each sub-system

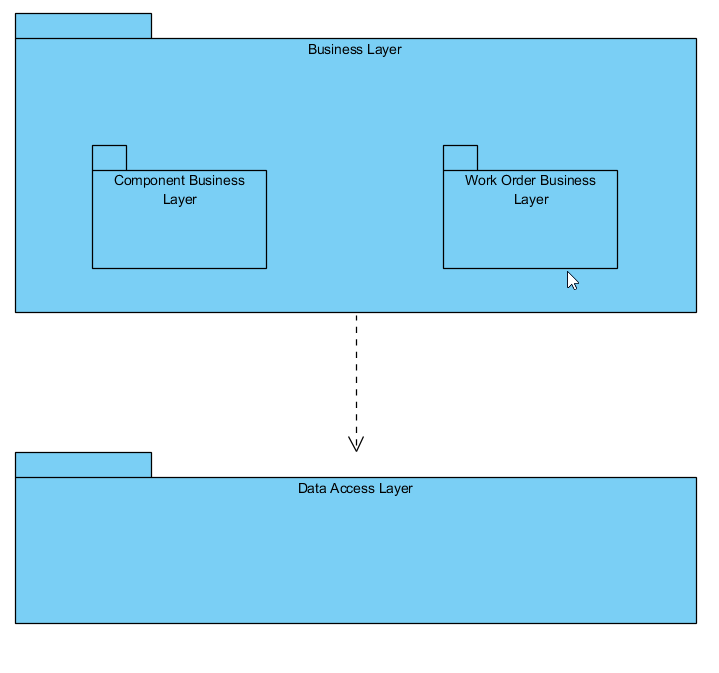
A system is a procedure, process, method, or course of action designed to achieve a specific result. Its component parts and interrelated steps work together for the good of the whole. Creating effective business systems is the only way to attain results that are consistent, measurable, and ultimately benefit customers.

Microsoft SQL Server Management Studio (SSMS) is an integrated environment to manage a SQL Server infrastructure. It provides a user interface and a group of tools with rich script editors that interact with SQL Server.

2.4 Sub System Architecture – Team member name(Adam)

Description of layering and partitioning into subsystems and responsibilities of each Subsystem

The way our system works is by utilising layers for each sub task, our layers are The Business Entities Layer, The Business Layer, The Data Access Layer, and our main program, any objects that need to be created will be taken from the Business Entities Layer and will use a factory to be created for example if I need a component object I will use the componentFactory class to get a component object. The business layer holds the model which contains for example on object of the current user that’s logged in, a list of work orders and many more things which are crucial to the program, this model will also hold functions to interact with the DAL (Data Access Layer) or the Business Entities layer, the model is almost like a middle man interacting with all of the layers in the system. Finally, the DAL is what interacts with the database, this is used for getting a list of the users or work orders and many more functions for our program. Overall, when the user is interacting with the main program is will use all 3 layers to divide up the work and get the job done.

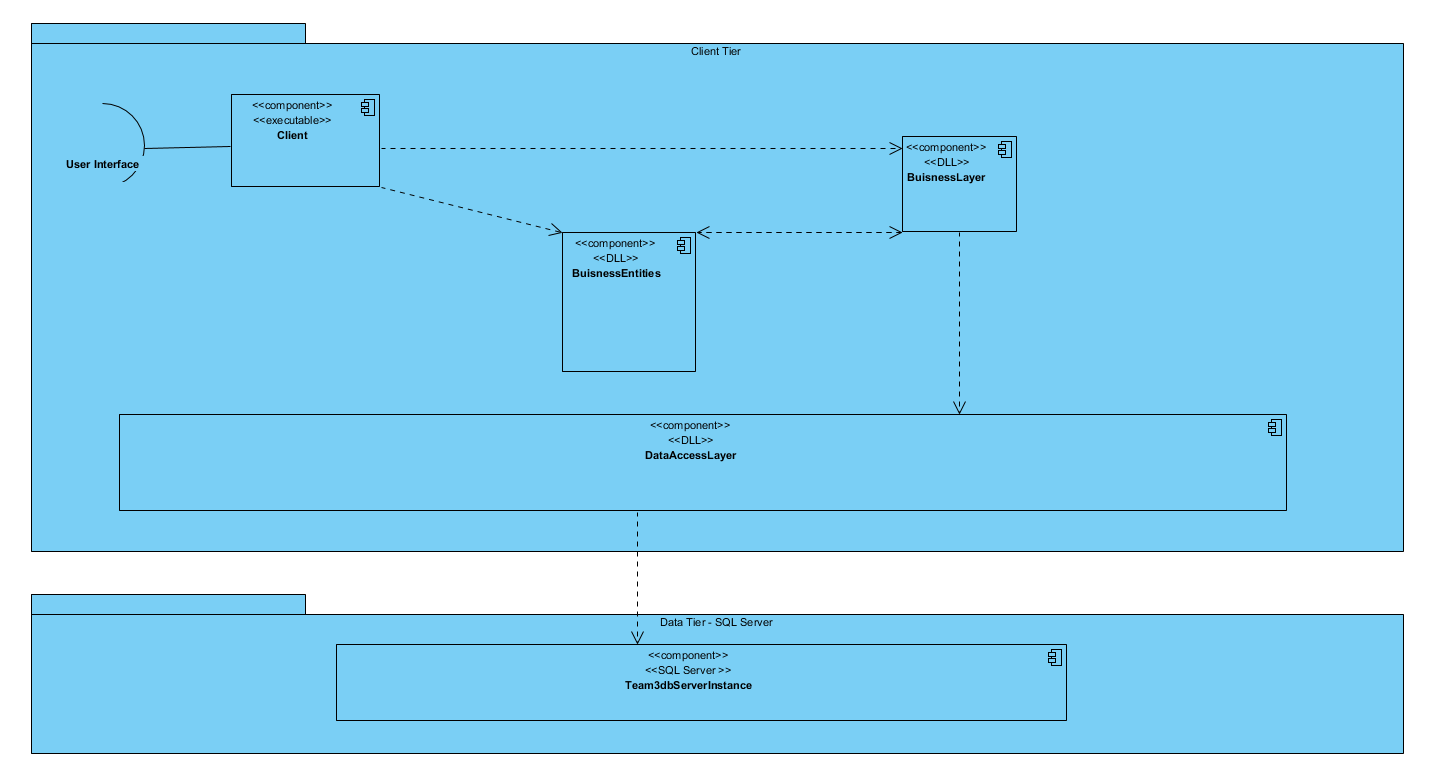


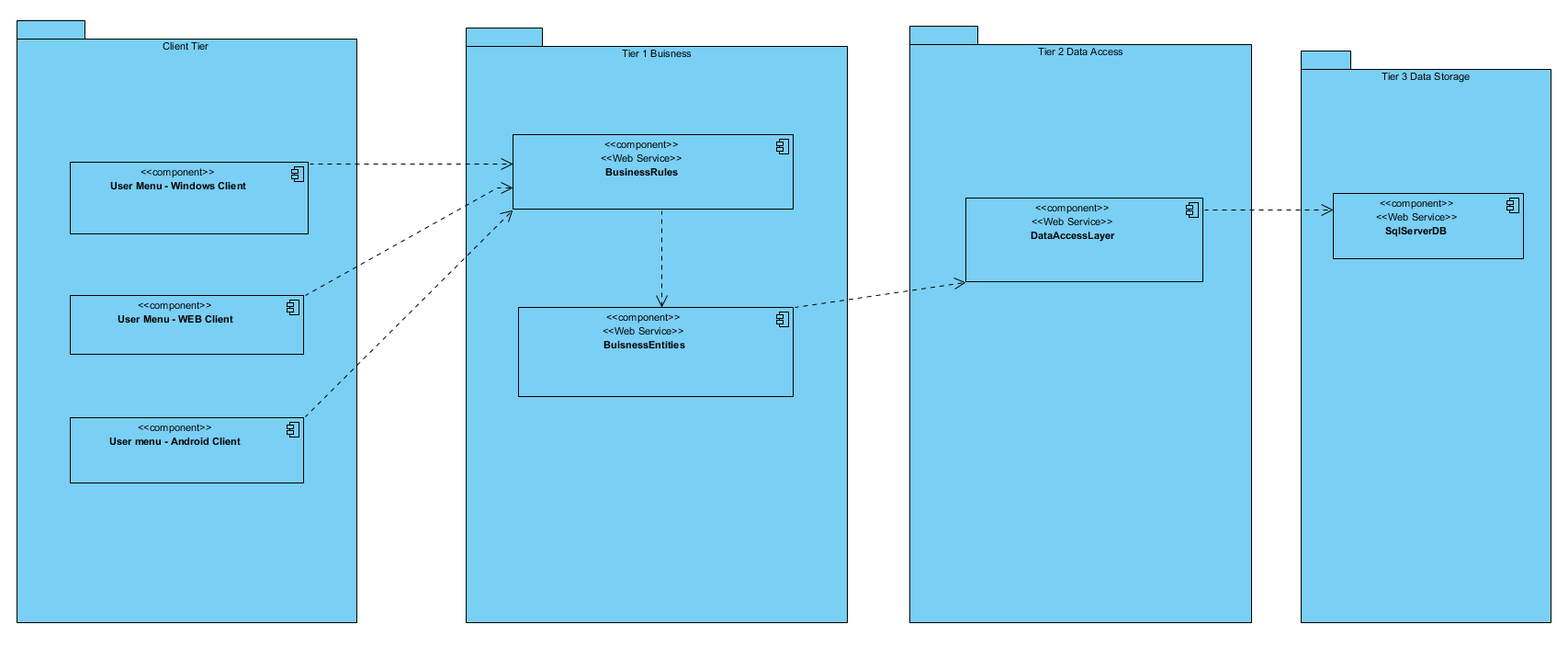
* 1. Component Diagrams – Team member name(Person C to complete)

**Fat Clients**

Fat clients are computers which are heavily dependent on the client side of the client-server architecture. This means that most of the actions that the user does is majority done on the local client however, this most also be communication to the server as well. An example of this is Filling out a form, if it is a Fat Client the form will be sent by the server, the client will fill out the form and then once its completed it will sent to the server. The opposite of this would be the form would update and send the data back to the server each line that the client would fill out. There are many advantages of using Fat clients rather then thin clients.

1. Because the Fat Clients being less dependant on the servers, The servers do not need high level of performance, this can be very beneficial for company’s Financially.
2. Fat clients are more flexible in the way that they do not need to have a constant connection to the server and could pull down information from the server, go offline and work on it and then re-connect to the server and send the information back. This allows Fat clients to be more flexible.
3. Pc’s are more high performance today so it would suit the client-server architecture to be more client based too use the infrastructure available to them on their pc rather then put most the traffic to the server.

Component Diagram for Version 1

Component diagram for Version Next

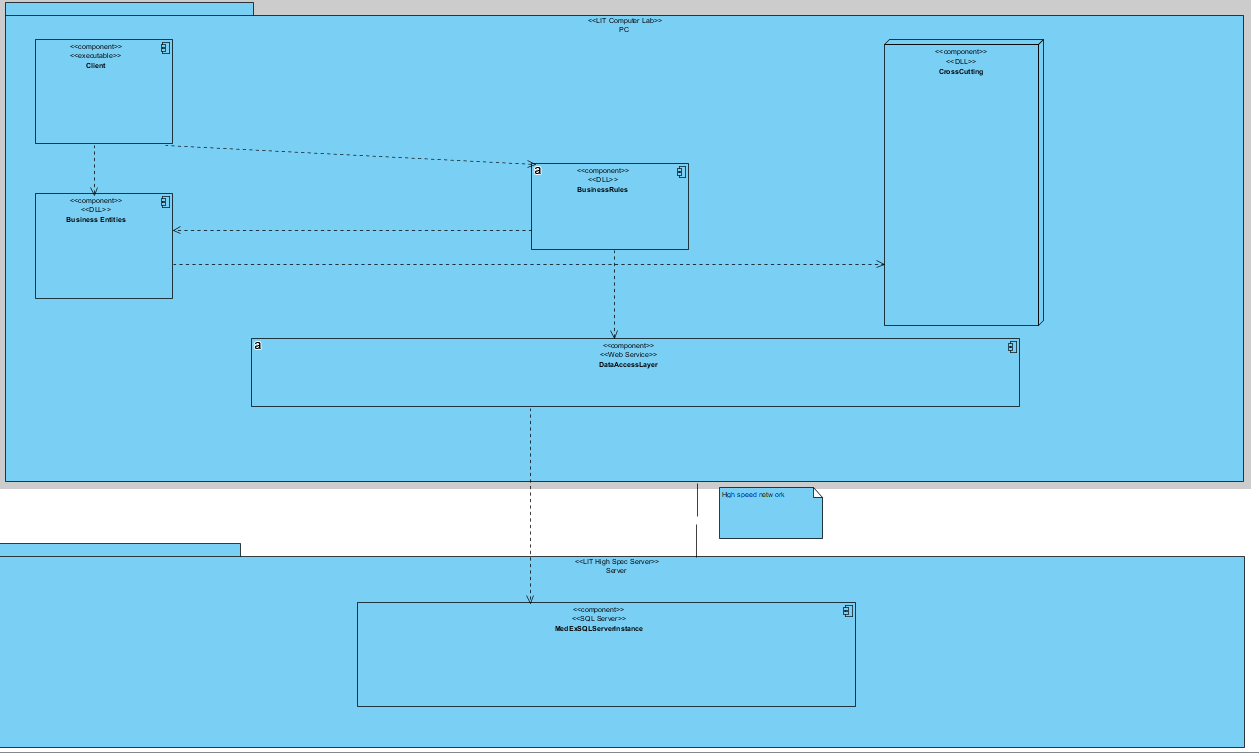
* 1. Hardware / Software Mapping – Team member name(will)

Describe how subsystems are assigned to hardware and off the shelf components.

Hardware/software mapping describes how subsystems are assigned to hardware and off-the-shelf components. It lists the issues introduced by multiple nodes and software reuse.

A large system is usually decomposed into subsystems using both, layers and partitions. Partitions vertically divide a system into several independent (or weakly-coupled) subsystems that provide services on the same level of abstraction.

The subsystems are assigned to hardware and off shelf components based on the layer’s that interact with each other in order to achieve abstraction.

Deployment diagram Version 1

* 1. Database design – Team member names of participants(All team to complete)

Class diagram for project showing persistent (entity) classes. Include suitable

types for all attributes. Do not show associations but instead show key

and foreign key fields

* 1. References(Person D to complete)

RAD Doc Ver ?

* 1. Tracking Entity Objects Team member initials(Person E to complete)

DbContext in Entity Framework is responsible for tracking the changes made on an entity or object, so that the correct update is done to the database when the SaveChange() method of context is called. When we retrieve entities using an object query, the Entity Framework puts these entities in a cache and tracks whatever changes are made on these entities until the savechange method is called. Entity Framework tracks the query results that return entity types. The Change Tracking tracks changes while adding new record(s) to the entity collection, modifying or removing existing entities and all the changes are kept by the DbContext level. These track changes are lost if they are not saved before the DbContext object is destroyed. DbChangeTracker class gives you all the information about current entities being tracked by the context also to track any entity by the context, it must have the primary key property.

In Entity Framework, change tracking is enabled by default.

* 1. Client Server systems Team member initials(mark)

Client-server systems are a software architecture model consisting of two parts. Client systems and server systems, both communicating over a computer network or on the same computer. A client-server application is a spread system made up of both client and server software. Client server applications provide an improved way to share the resources. The client process is always the one that initiates a connection to the server, while the server process is always waiting for requests from any client.

The client-server relationship describes the relation between the client and how it makes a service request to the server, and how the server can accept these requests, process them, and return the requested information to the client. The interaction between client and server is often described using sequence diagrams. Sequence diagrams are standardized in the Unified Modelling Language.

When both the client process and server process are running on the same computer, this is called a single seat setup. Most servers have a one-to-many relationship with clients, meaning a single server can provide resources to multiple clients at one time when the server is running on a different computer than the client.

* 1. **Conclusions** (All team to complete **Team member names of participants**)

Describe why systems architectures are sometimes moved from a two tier to an N tier architecture.

Develop Component and Deployment diagrams for Version Next of your application and show the credit card validation on the Bank system.