# Front Page

School of Computing and Mathematics

Teesside University

Middlesbrough

TS1 3BA

SCIPA

Software Controlled Industrial Process Automation

BSc (Hons) Computing

[Date of Completion]

William Whitehead

M2006199

Supervisor: Jim Longstaff

Second Reader: [Second Reader]

# Abstract

Process automation is a key system in most of today’s industry; it’s able to reduce costs, maximise efficiency, improve safety and, amongst many other benefits, is able to ultimately improve the profitability of a process.

This project follows the design and development of a supervisory control and data acquisition system (SCADA) that is able to monitor and self-control a set process loop via a user defined configuration. The project also contains an investigation into the implementation of a ‘big data’ database using MongoDB.

# Acknowledgements

## Teesside University

First and foremost, I’d like to thank Jim Longstaff and John Goodge whose support as my supervisors has been second to none, respectively this year and last. Without their help, final year would have been much more of a challenge.

I’d also like to thank Mansha Nawaz, whose obscure teaching and mentoring style has proved to be invaluable. His support through the Advanced Database Systems and Computing Project modules has been thorough, throughout and very much appreciated.

## DuPont Teijin Films

Many thanks to both Andrew Doonan and Andrew Taylor whose perfectionisms, intolerance to ignorance and intrigue to new ideas was passed down to me during my time on placement. Process automation is a field of computing I take great interest in, and without my time with the company, I wouldn’t be aware of any of the concepts of control theory, never mind be able to design and implement a ‘big-data ready’ SCADA system.

For their guidance, support and friendship, I thank them both.

# Contents

[Abstract i](#_Toc447544649)

[Acknowledgements ii](#_Toc447544650)

[Teesside University ii](#_Toc447544651)

[DuPont Teijin Films ii](#_Toc447544652)

[Contents iii](#_Toc447544653)

[Table of Figures v](#_Toc447544654)

[1. Introduction 1](#_Toc447544655)

[1.1. A Brief Overview of Process Control and Control Theory 1](#_Toc447544656)

[1.2. Process Control via Process Loops 1](#_Toc447544657)

[2. External Hardware and Software 2](#_Toc447544658)

[3. Project Methodology 3](#_Toc447544659)

[4. Problem Domain and Example Deliverable Solution 4](#_Toc447544660)

[5. Research Stage 5](#_Toc447544661)

[6. System Specification 6](#_Toc447544662)

[7. System Design 7](#_Toc447544663)

[8. Version Control 8](#_Toc447544664)

[9. Project Development 9](#_Toc447544665)

[9.1. Internal Projects and Solution Structure 9](#_Toc447544666)

[9.2. Inbound Data Handling and Reading 9](#_Toc447544667)

[9.3. Controlling the Process with Rules 9](#_Toc447544668)

[9.4. Outbound Data Handling and Writing 9](#_Toc447544669)

[9.5. Data Storage and Management 9](#_Toc447544670)

[9.5.1. Relational Databases within SQL Server 9](#_Toc447544671)

[9.5.2. NoSQL Databases within MongoDB 9](#_Toc447544672)

[9.6. Business Intelligence Tools 9](#_Toc447544673)

[9.7. Backend and Automated BI 9](#_Toc447544674)

[9.8. Frontend and User-Focussed BI 9](#_Toc447544675)

[10. Testing 10](#_Toc447544676)

[10.1. Acceptance Testing 10](#_Toc447544677)

[10.2. Unit Testing 10](#_Toc447544678)

[10.3. Component Testing 10](#_Toc447544679)

[10.4. Performance Testing 10](#_Toc447544680)

[11. Project Evaluation 11](#_Toc447544681)

[11.1. Project Goals 11](#_Toc447544682)

[11.2. Next Steps and Future Improvements 11](#_Toc447544683)

[11.3. Recommendations 11](#_Toc447544684)

[12. Critical Review 12](#_Toc447544685)

[13. Conclusion 13](#_Toc447544686)

[14. References 14](#_Toc447544687)

[15. Appendices 15](#_Toc447544688)

[15.1. Project Proposal 15](#_Toc447544689)

[15.2. Poster Presentation 24](#_Toc447544690)

[15.3. Arduino Sketch Source Code 26](#_Toc447544691)

[Trending Application 26](#_Toc447544692)

[Input Acceptance Application 26](#_Toc447544693)

[Basic IO Application 26](#_Toc447544694)

[Other 26](#_Toc447544695)

# Table of Figures

**No table of figures entries found.**

# Introduction

## A Brief Overview of Process Control and Control Theory

Process automation is a fundamental element in most of today’s industry. The ability to actuate hardware based on a user-defined configuration and real-world sensory data is a topic both businesses and mathematicians have invested significant amounts of time and money in, especially where such systems can be used to enhance safety, efficiency and profitability.

Control theory is the principle of controlling a process based on external factors and ‘rules’. The purpose of such is to “achieve optimal process operation despite the presence of significant uncertainty about the plant behaviour and disturbances” (Engell, 2007, p.203). Control theory, therefore, requires any application of such to allow unexpected changes in the incoming data and be able to handle the change appropriately. An example may be the sudden drop in a temperature reading of a vessel, the control system must be able to interpret the unexpected change and automatically implement a remedy via any preconfigured routes, be that software based or mechanical.

This project focussed on the development of SCIPA, a Supervisory Control and Data Acquisition system, often abbreviated to ‘SCADA’. SCADA is a term that “generally refers to an industrial computer system that monitors and controls a process” (Subnet, 2015) where a process contains one or more process loops.

Industrial processes often implement specialist hardware and control devices in order to maintain a steady production environment which are, at both a manufacturer and a vendor level, often controlled via Programmable Logic Controllers, often referred to as PLCs.

## Process Control via Process Loops

A process loop is the term used to describe a single algorithm that runs repeatedly within a process. A basic thermometer, for example, has a single process loop that loops over the following algorithm repeatedly:

* Read ‘local temperature’;
* If ‘local temperature’ is less than the ‘set point’ send ‘ON’ command to boiler;
* If ‘local temperature’ is more than or equal to the ‘set point’ send ‘OFF’ command to boiler.

Industrial process often consist of many process loops, with the more complex systems housing collections of process loops within PLCs, which in turn are controlled within a parent process loop.

A process loop within this project is taken to mean a ‘closed loop’ in that “an operation, process, or mechanism is regulated by feedback” (Merriam-Webster, 2015). By allowing both data input and output from the application, SCADA systems like SCIPA are able to allow users to define ‘rules’ in order to manage and automate actions taken by a computer for process automation.

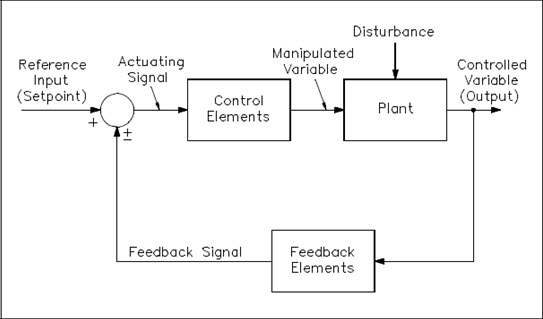


Figure 1.2.1 Closed Process Loop (TechTransfer, 2014).

Figure 1.2.1 shows the process in which a process loop is executed. A reference input, commonly referred to as a ‘set point’, is passed to the system alongside live process data, shown as ‘feedback signal’. The control element of the loop, which in terms of this report is a SCADA system, performs one or more actions in order to manipulate and control the variable which is then output back to the ‘plant’ or process environment. The loop will repeat forever, with the SCADA system controlling the values output to the plant in an organised manor.

## Project Outcome

The aim of this project is to design and build a fully working process control system that has the ability to read live data, output data or commands to both software and hardware based components based upon process rules and provide users with useful and meaningful ways to investigate and visualise process data.

The key targets are to:

* Read data via common industrial methods, such as Serial or Database;
* Allow users to create complex rule sets for data;
* Output either process data or commands based upon the rules;
* Store live data in a relational database for fast-access and reporting;
* Store historical data using a long-term data repository for data archiving;
* Grant access via an industrially-design HMI.

This report is split down to cover all of these aspects in detail, focussing on the real-world practises and the academic benefits of working with such.

# External Hardware and Software

Process control systems are often able to read data from and write data to a selection of platforms. In order to emulate a real-world and industrial environment, SCIPA has been designed to communicate with live data from databases that allow access via OLE, SQL or ODBC drivers, as well as flat files based in Unicode, ANSI and ATF-8, and serial data, including devices that require RTS (Request To Send) and DTR (Data Terminal Ready) signals.

In terms of hardware connections, the prototype application can successfully communicate with a multitude of hardware devices via RS32 and USB connections, with testing completed using a collection of Arduino Uno devices to simulate serial process data and recievers.

For acceptance and integration testing with external software, basic C# applications have been developed to simulate process data handlers using flat text files based on the local development computer.

Further implementation of SCIPA was conducted using the Microsoft Azure IaaS (Infrastructure as a Service) platform for hosting SQL Server databases, with NoSQL being handled by a local instance of MongoDB.

MongoDB has been chosen as the long-term data store for this project because it is the fourth most popular database engine in the world, ranking first in the world as a non-relational database engine (gmbh, 2016). By deciding to implement MongoDB, the project has used an industry-standard tool and acted as a strong academic introduction to such technology.

# Project Methodology

The development of this project’s artefact was conducted, for the most part, using the Agile methodology. The principle of the Agile development method is that projects are completed by using an iterative cycle of production. Each element of the developed system will be designed, built and tested in its own right.

“Individuals and interactions over processes and tools  
Working software over comprehensive documentation  
Customer collaboration over contract negotiation  
Responding to change over following a plan

That is, while there is value in the items on  
the right, we value the items on the left more.”

(Agilemanifesto.org, 2015)

There are several implementations of the Agile methodology, all of which must follow the above quote which lays out the requirements for what Agile should mean to developers. For this project, the Scrum technique has been used as far as was reasonably possible. Scrum is usually implemented for projects where developers are able to work within teams of six or seven, however, the ‘sprint’ technique was used as a strong foundation on the iterative development of the system. Each of the required elements of the system, as laid out in the project proposal formed the product backlog, with each ‘product’ housing a set of sprints. Sprints were completed within the allotted time and committed to the repository.

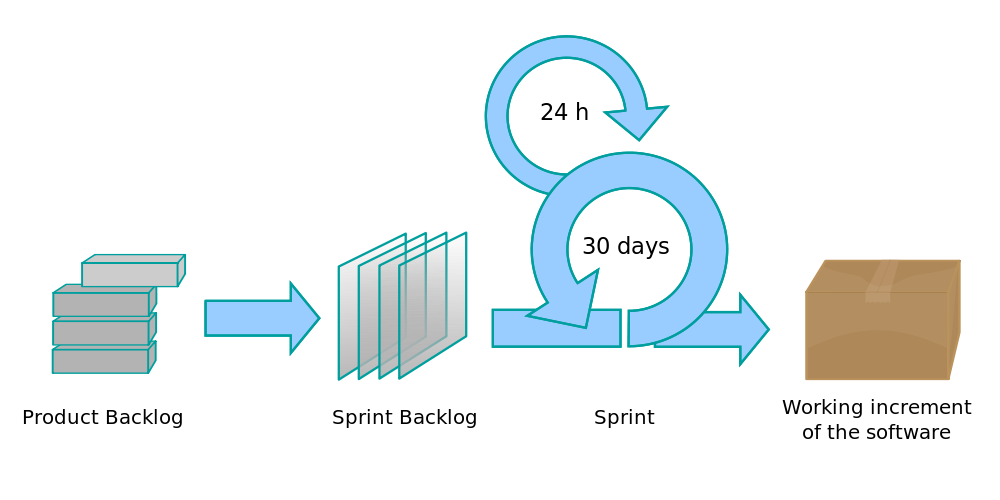


Figure 3.0.1 Scrum development methodology (Lakeworks, 2009).

The key idea of products and sprints within Scrum is that each product is a self-contained, fully tested and working element of the application, made up over one or more sprints.

## Analysis

As process control is often considered a topic more suited to students of Masters or Doctorate courses focussed on mathematics or chemical engineering, the analysis stage of this project focussed on reading materials surrounding basic mathematical principles of control theory as well as engineering practises in terms of process development in the real world.

Steve Mackay, the foundation Dean of Engineering for the Engineering Institute for Technology, gives complimentary online seminars which formed a more visual and interactive basis for researching the requirements of a SCADA system. These seminars formed the foundations for the project’s research, giving insight into the necessary engineering principles and industrial-expectations for Human Machine Interfaces (HMIs).

Further research was conducted to fully investigate best-practises for multithreaded applications, especially when working with several data sources and backend services within a time critical environment.

Alongside the research with regards to multithreading, several applications were developed both for the development computer to simulate database and flat file communication on both the inbound and outbound channels, as well as Arduino sketches, small looping applications designed to output trending values within a given range. The sketches designed for the Arduinos submitted serial data every second with values trending either positively or negatively and were used to research and test both Windows’ and C#’s ability to accept data from several sources simultaneously.

The more technical programming challenge of working across threads was combatted via the further research into method invocation which was required to ensure that the User Interfaces could show live data without becoming overwhelmed with data processing, rendering them unfit for purpose.

## Design

As the design of the application architecture directly impacts the efficiency of the program, the layout of the project played an important and integral part of the design phase. The methodologies used to fully plan, articulate and progress the design of the system

Discussing the elements of such, Learn.org states that “Design methodology stresses the use of brainstorming to encourage innovative ideas and collaborative thinking to work through each proposed idea and arrive at the best solution” (2003). Brainstorming, basic sketches and technical drawings were all used to design the architecture of the application as a whole, the functionality for each respective user interface.

A more key element of the design process for this project was the algorithm to be used in order to effect a process loop. An “algorithm is a statement about how a problem will be solved” (Beech, no date), with the problem in this instance being the understanding of the live process value. The required algorithm must implement many of the algorithmic design concepts, including sequence, decision and repetition constructs in order to prove effective and fit-for-purpose. Given that an algorithm is the process of solving a problem in a finite number of steps, the design of the algorithm is key in removing unnecessary required processing of data.

The product is designed to be distributed over several logical tiers so as to allow cleaner separation of concerns and reduce unnecessary dependencies. This approach, known as N-Tier, is designed to “separate processing into discrete tiers that are distributed between the client and the server” (Microsoft, 2016) and is a modern approach to application development.

## Implementation

Implementation of this followed the project’s methodology as a whole, using Scrum. Each incremental build of the product was a further step towards the completion of one or more of the products from the product backlog, and each step was in accordance with the relevant product’s design.

As the project was designed to use a layered architecture, the individual projects that make up SCIPA as a whole were initially implemented in isolation of all other layers so as to remove any unrequired dependencies being formed. As the system grew ‘tighter’ and projects required access to one another in order to be of intended use, projects were conglomerated into a single solution.

During both the isolation and ‘packaged’ phase for each project within the solution, the system was stored on a secure, cloud-based backup platform with automated version control so as to ensure that any changes made could be easily ‘rolled back’ within a matter of seconds. Formal versioning and version control was also part of the implementation phase, in that each implementation iteration was committed to a Git repository to provide a further layer of developmental security.

The Milestone project management methodology was used throughout the project to provide a solid basis and backup ‘guide’ during the implementation. The milestone approach ensures that every milestone that is met is committed to a form of permanent backup storage.



Figure 3.3.1 Milestone implementation methodology.

For this project, a slight modification was made to the chosen methodology so as to better incorporate the other backup and versioning systems in place.



Figure 3.3.2 Modified Milestone implementation methodology.

The revised implementation of the milestone methodology allows for committals to the repository, which is a much more frequent exercise than permanent backups of all of the stable versions.

## Testing

Testing has been a continuous and rigorous element within this project, with unit and observation testing being conducted at regular intervals for each product, and often sprint, within the relevant Scrum backlog.

Unit Testing forms an entire project file within the overall solution, and is used to ensure that individual components of the application return values as expected. Expected values are prepared prior to the execution of the application via a manual run through of that element’s algorithm. The unit tests will pass should the system return the same result as the expected value, otherwise these will fail.

Observational testing is the second form of testing that was used during development. This method requires visible input data, often by hand, being entered into the application with any associated debugging messages or printed statements being checked to ensure the values are consistent. The project has been implemented with a built-in logging service which assists in this testing type.

Integration testing has formed a significant proportion of the testing conducted as part of this project. Such testing involves ensuring the desired interoperability of each tier within the solution, especially those that have a direct reference, and thus, dependence on another tier.

Acceptance testing is an expansive term used to describe a multitude of testing types. The definition that acceptance testing has been taken to mean as part of this project is as follows:

“A technique performed to determine whether or not the software system has met the requirement specifications. The main purpose of this test is to evaluate the system's compliance with the business requirements and verify if it is has met the required criteria for delivery to end users” (Acceptance testing, 2016).

In this respect, SCIPA’s acceptance testing focussed primarily on the functionality of each individual component and was conducted alongside integration testing to ensure such functionality worked appropriately across the application’s tiers.

# Problem Domain and Example Deliverable Solution

The domain of this project is vast, with implementations being able to range from between basic process data monitoring through to the complete control of a process. Given the application is designed to be a generic controller as opposed to being designed for a specific process or PLC type, the examples used within this report can be considered to be a minor selection of the available illustrations of the functionality the end product is able to fulfil.

There following are implementation examples used within this report:

* Simple thermostat system;
* Example two.

The delivered solution file will contain no pre-populated data because it is not possible to assume inbound or outbound data sources will be available at the required locations. Some SQL Server database regeneration scripts will be included to allow example investigatory work to commence with regards to Device, Communicator, Rule and Action examples so end users are able to develop their own example and test processes within the application.

# Research Stage

This project has posed several significant areas for research both in terms of academic development and in the Chemical Engineering field of Process Control.

Process Control, while a technical subject, is often overlooked by journals, publications and publically available papers as most solutions in this area are considered intellectual property. Providers such as ABB, Idhammer and Rockwell Automation all develop and supply their own Process Control software but keep any internal architecture, developmental knowledge and “developer-focussed” information internal to their organisation. For this reason, the research conducted as part of this project has been focussed towards the implementation of formal software architecture principles with regards to:

* Application design and structure;
* Multi-threading and system process management;
* User interface layout and handling (with a focus on industrial environments).

Resources were also invested into the surface-level understanding of process control, process loops and process control systems in general, as discussed previously in sections 1.1 and 1.2.

## Application Design and Structure

Research into the benefits, development techniques and required standards of N-Tier-based architectures formed

The research paper, ‘A data-centric design for *n*-tier architecture’, focusses on the implementation of n-tier based systems within retail, though the principles discussed remain entirely relevant for this project.

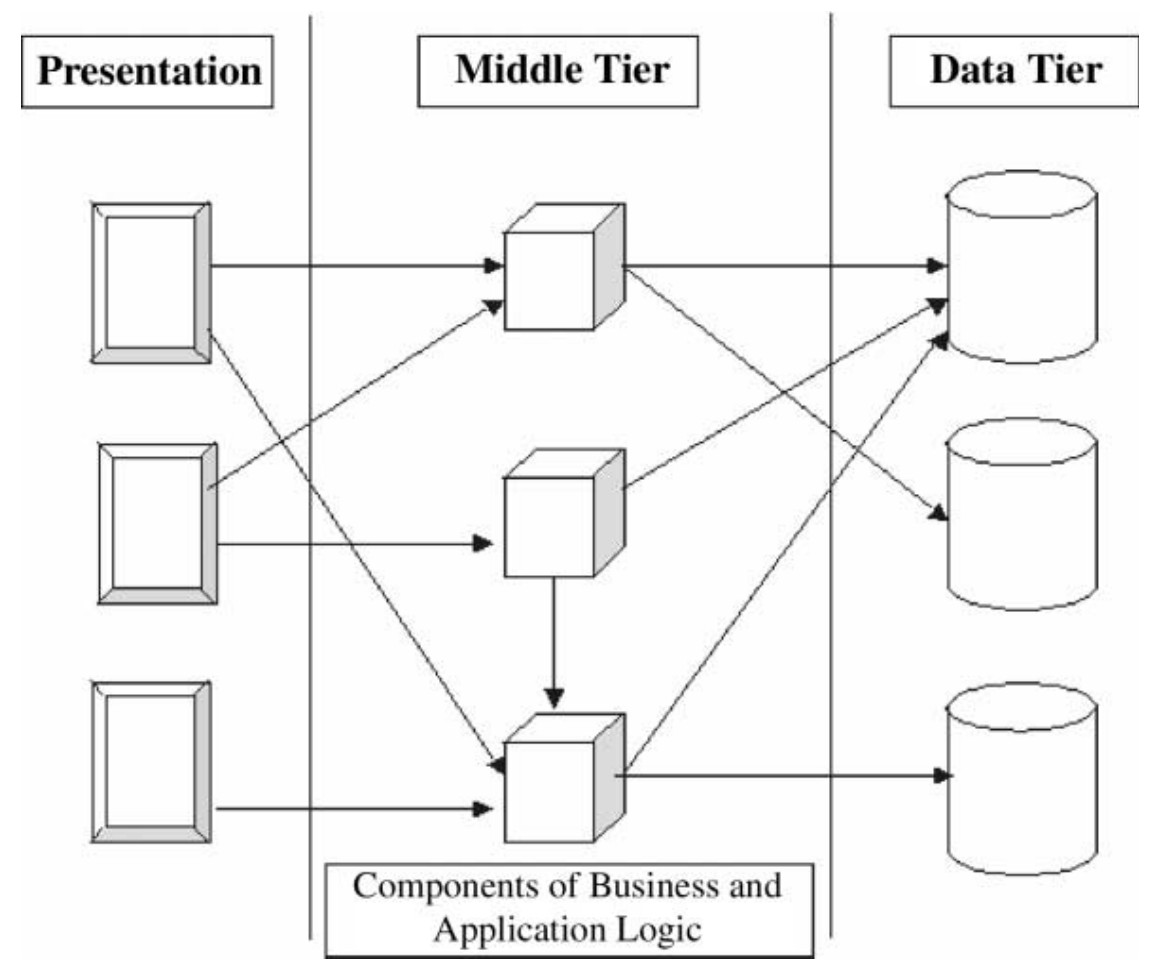


Figure 5.1.1 n-tier architecture example (Manuel and AlGhamdi, 2003).

Figure 5.1.1. shows how the presentation layer is maintained in a separate package to the domain/business logic, which in turn is abstracted from the data tier. The benefits of such a design allows changes to any of the software elements and, so long as the contracts imposed on each section are maintained, none of the dependant functionality will be aware of the change. This provides software developers with a more efficient way to update and upgrade applications. It also allows easier scalability.

As applications grow, the individual elements can be scaled independently to all other elements, both horizontally and vertically. Horizontal scaling is a reference to increasing the number of nodes or processors that compute the specified element whereas vertical scaling is the increasing of compute power within a single node.

For this project, n-tier ensures “the clients and components of the architecture should be able to efficiently execute across multiple hardware platforms of a network” and “the applications within the architecture should be able to work together in a consistent manner to perform tasks for the users of an information system” (Manuel and AlGhamdi, 2003) which will allow the end product to be suitably scaled if housed in an environment that requires more processing power.

One example on this could be the industrial implementation of the project where physical Devices are connected to more than one computer. The presentation layer provides each computer with access to the domain logic, though this does not necessarily have to be on the same system. Assuming the multiple computers are connected via a network, the domain logic and data access layers (tiers) could be housed on a central server. This would enable SCIPA to run over several nodes (horizontal scaling) on the internal network. This would not be possible if the project design had been developed using legacy design techniques.

SOA(D) was an alternative architecture that was investigated, which, whilst continuing to differentiate between layers and tiers within the solution, acts as a much more stringent and formal architecture with each application element given a set and prescribed role.

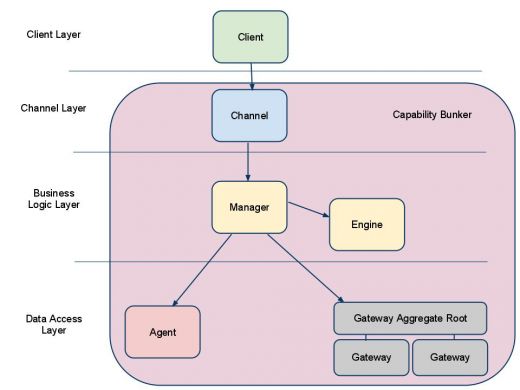


Figure 5.1.2 SOA Design architecture diagram MetalLemon (2015).

The SOA design pattern, whilst just as appropriate for this project as N-Tier, would have imposed a greater requirement in terms of time. As a time-critical project, this option was therefore ignored in favour of the similar, more industry-used n-tier methodology.

It is important to note that multi-tier and multi-layered architecture types are different things. N-Tier is the description of multiple process boundaries, whereas N-Layer is the description of multiple logical-layers within a single tier. In terms of this project, the descriptions for both N-Tier and N-Layer have been amalgamated as the system has been built using both approaches.

## Multi-Threading and System Process Management

As research suggested that the production of the project should be conducted within an N-Tier style, the system, by design, operates over one or more processes, with each process operating with several internal layers, n-layer within n-tier.

Due to the requirement of constant updates on the User Interfaces in order for live data and alarms to be visible to users, the design and implementation of threading was required as a necessity. The C# Thread Pooling service allows dynamic creation of threads from computable methods which, ultimately, allows new threads to be created during normal application run time. The largest consideration of developing with a multi-threaded application is the requirement of showing computed material within or as part of the user interface. As all UIs run on their own thread, abstracting processing of data to alternative threads reduces the chance of application ‘hanging’ or ‘freezing’ mid-operation, however, it also requires update methods on the UI to be invoked by the runtime environment and underlying operating system as and when possible.

Research into the invocation service within C# found that whilst such a technique is more user-friendly, dynamically created threads poses as a single point of failure against attackers. Should a malicious user find an opening, a denial of service attack could be launched via the management of incoming data, and “if an outsider can control your incoming work, she can control your thread creation” (Kumar, M., no date). Generation of too many threads can cause symptoms that similarly mimic the creation of too few threads – slow operation times and the appearance of an ‘overwhelmed’ application or system. When working within an industrial environment, such symptoms are unacceptable to users as the occasional missing data value can be just as problematic the complete halting of a system, if not worse due to the lack of warning or errors.

Research into the management of threads has shown that all implementations of such should be done so in a controlled, event-driven manor. Event-driven thread creation ensures that developer-based domain/business logic can have command and control over the generation of threads, which in turn, mitigates against denial of service attacks on all layers/tiers of the application.

## User Interface Layouts and Handling

User interfaces are the ‘window’ into the application for almost all users and use-cases, with API level and machine-based uses being two of the most probable exceptions.

With process control being a distinctly industrial topic, the research for user interfaces has focussed primarily on the design and implementation of HMIs – Human Machine Interfaces. Such interfaces are sometimes referred to as MMIs (Man-Machine Interfaces) or HCIs (Human-Computer Interfaces), for reference.

According to Gruhn, P. and Triplex, P.I.E. (2010) “Poor HMI designs have been identified as factors contributing to abnormal situations, billions of dollars of lost production, accidents, and fatalities”. Their paper, ‘Human Machine Interface (HMI) Design: The Good, The Bad, and The Ugly (and what makes them so)’, goes on to explain the key requirements of HMIs:

* Contrast – different things should be displayed as clearly different entities;
* Repetition – visual elements and controls should be repeated where possible;
* Alignment – all visual elements must ‘visually connect’ with neighbouring elements;
* Proximity – similar or associated controls should be grouped together.

The International Society for Automation (ISA) maintain industry standards by providing certifications and training, advice and technical resources to all businesses interested in automation. Their advice for HMI design covers the following topics (Automation IT: HMI design (2015)):

* Selectively use colour and animation;
* Use graphics where possible;
* Use images where relevant;
* Ensure important items are always available;
* Provide situational awareness;
* Limit required access;
* Provide feedback to users;
* Provide visible time stamps and logging;
* Maintain style throughout.

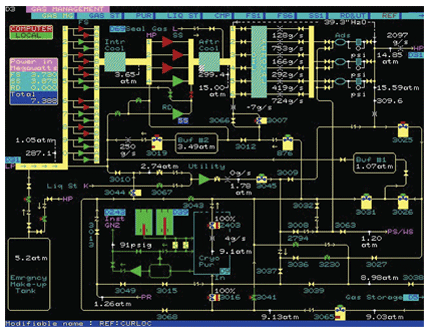


Figure 5.3.1 Example of bad HMI design – too much information (Automation IT: HMI design (2015).

The ISA show that, in figure 5.3.1., although operators will approve of basic or implicit access to information, the usefulness of the dashboard is hindered because there is a greater chance of causing confusion or complacency by overwhelming users.

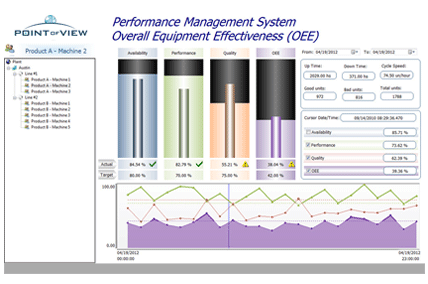


Figure 5.3.2 Example of good HMI design – inclusion of relevant graphics (Automation IT: HMI design (2015).

Figure 5.3.2. shows a “good example” of an HMI dashboard design as its use of graphics is clear, separated by colour and is easily readable.

# System Specification

Text

# System Design

Text

C:\Users\Will Whitehead\Downloads\SCIPA Loop Diagram (1).png

# Version Control

text

# Project Development

Text.

## 9.1. Internal Projects and Solution Structure

Text

## Inbound Data Handling and Reading

Text

## Controlling the Process with Rules

Text.

## Outbound Data Handling and Writing

Text.

## Data Storage and Management

Text.

### Relational Databases within SQL Server

Text.

### NoSQL Databases within MongoDB

Text.

## Business Intelligence Tools

Text.

### Backend and Automated BI

Text.

### Frontend and User-Focussed BI

Text.

# Testing

## 10.1. Acceptance Testing

## 10.2. Unit Testing

## 10.3. Component Testing

## 10.4. Performance Testing

# Project Evaluation

## 11.1. Project Goals

## 11.2. Next Steps and Future Improvements

## 11.3. Recommendations

# Critical Review

# Conclusion

# References

Paste the relevant references here.

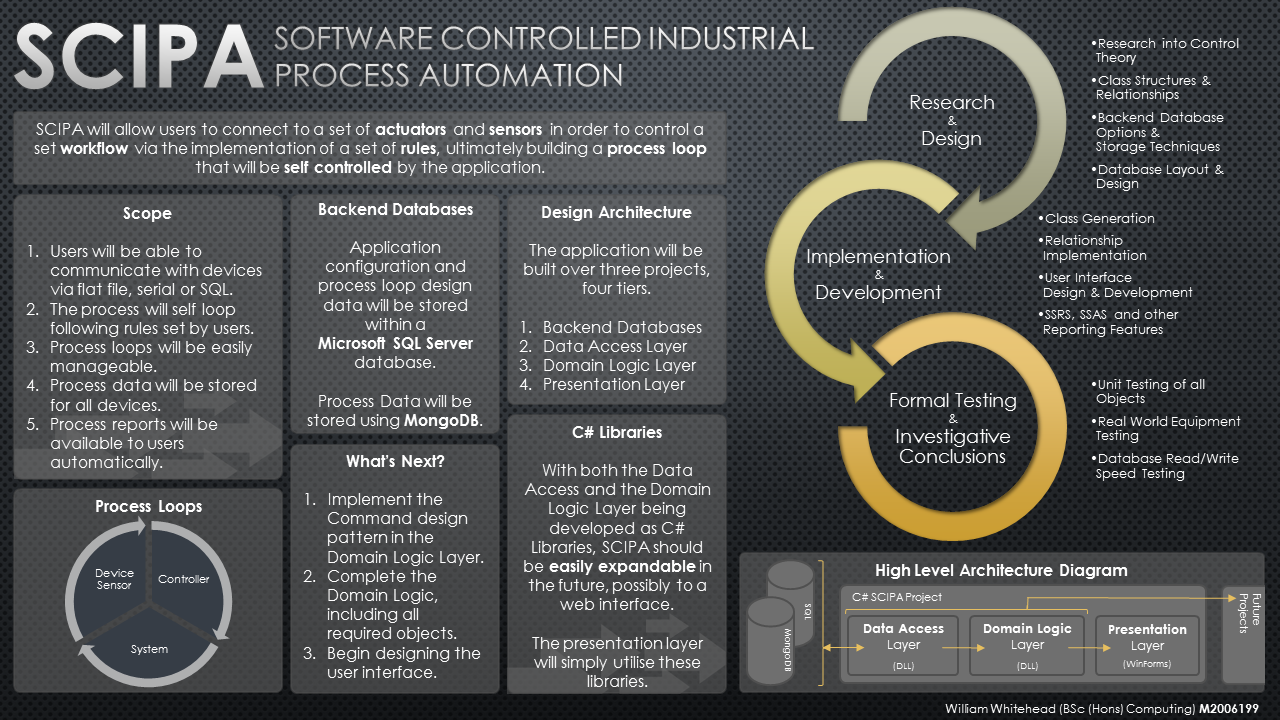
# Appendices

## 15.1. Project Proposal

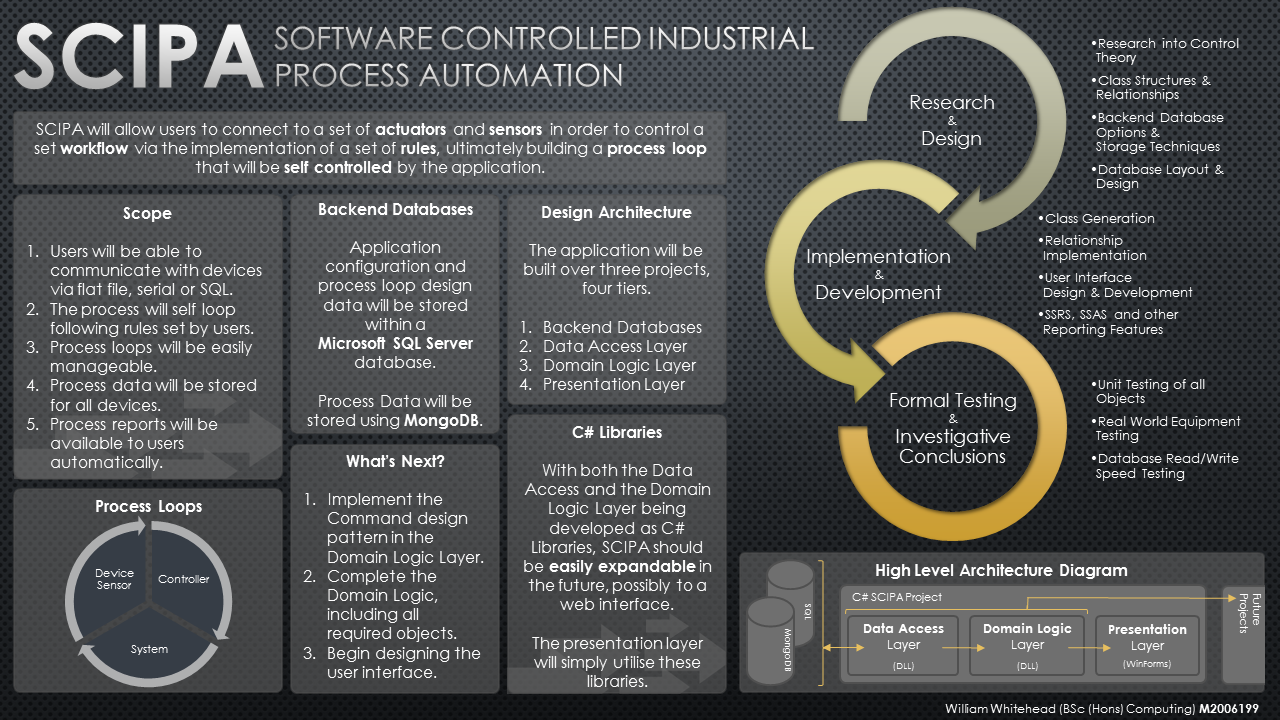
Proposal should be moved here from the “proposal\_appendix” file.

## Poster Presentation

Side A



Side B



## Arduino Sketch Source Code

### Trending Application

### Input Acceptance Application

### Basic IO Application

### Other