Training software for a Concept II rowing machine

George R Palmer

October 2003 - April 2004

Word count: 17,000

This report is submitted as part of the degree of Software Engineering to the Board of Examiners in the Department of Computer Science, University of Durham. All material is original and from the above author unless explicitly referenced.

Abstract

Concept II indoor rowing machines are a standard used by both professional and amateur rowers. A PM2+ (the display presenting numerical data to the rower) enables retrieval of real time data through a serial port. This project developed a system that retrieves such data and presents it in a graphical form to enhance training. More specifically the software represents a river with moving boats, upon which the user can race other rowers. The system also supplies the functionality to store performances that can be raced at a later date.

An incremental model was chosen to guide development as it provides a high suitability for working with a client. This is achieved through implementation as a series of discrete levels of functionality. Such a strategy allows flexibility when the client changes a requirement which, in the instance of this project, occurred several times. The flexibility offered by the model meant it was possible to adapt to changing circumstances with little effect on the overall project schedule. This was considered one of the primary factors leading to the successful completion of the project.

Evaluation was conducted through a questionnaire given to Durham University Boat Club rowers after their March five kilometre ergo tests. It was found that rowers with more than one year's experience found the system more useful than rowers with less experience. Furthermore, rowers who had used the system before the ergo tests found it more useful than those who were using the system for the first time. Overall, the evaluation determined that half the rowers believed the system helped them to achieve a better time. It was therefore concluded that the original project objective – to help rowers enhance performance – had been achieved.

Acknowledgments

I would like to offer my sincere thanks to Dr. Elizabeth Burd and Owen Wade Hall-Craggs for their continued support throughout this project.

I would also like to extend my thanks to Simon Larkin at Concept UK for his help with the many hardware related problems encountered throughout this project.

Contents

List of Figures

Chapter 1

Background

Chapter 1 discusses the problem domain and objectives of the project. From these objectives, the deliverables are derived and a project plan formulated. The terminology used throughout the report is also defined and an overview of the rest of the report presented.

1.1 Problem Definition

This project will develop an application that retrieves real time data from an indoor rowing machine and presents it in a graphical form to enhance training. The most common and unofficial standard rowing machines are produced by a company called Concept. The current model is number two; hence the term ”Concept II” is often used to refer to a Concept indoor rowing machine [8]. A performance monitor (PM2) displays the 500 meter split, heart rate and power of the stroke as well as the stroke rate, time, average power and average 500 meter split for the workout. The advanced monitor (PM2+) offers in addition to the features provided by the standard performance monitor, a serial port on the rear enabling connection to a computer. This allows for real time retrieval of stroke and workout data. Concept II rowing machines are used by professionals and amateurs alike, as well as being very commonplace in most health clubs and gyms [8]. It is hoped this will make for a good project scope as the software will be of use to many rowers of ranging abilities.

To improve in any sport, it is important to train regularly and subsequently analyse the

results of training to make improvements. Many top athletes keep a training diary as former national indoor triple jump champion Jon Best [3] highlights, ”Keeping a training diary is the done thing among top athletes. However it is rare anyone ever performs improvement analysis based on the information contained within it”. A few programs such as NetAthlon (*http://www.fitcentric.com*) and UltraLog (*http://www.ultralog.com*) now provide a training log facility either as a web-based or standalone program. Unfortunately there are regular complaints that this kind of software is not sport specific; that is certain data cannot be entered in the required format for a given sport. It is hoped that this project will overcome such problems for rowing as it will be specific to the sport.

Concept currently provide a software package called e-row that allows connection to a Concept II and the storing of training results. This lacks the functionality required by many rowers, such as the ability to race a personal best performance or have a pace boat represented as a shadow. A visit to the Concept forums confirmed this with a number of frustrated e-row users and a list of requested features for the product [7]. Concept stopped the development of e-row in the year 2000 and as such none of these requests have been fulfilled [7]. Recently a company called Digital Rowing (*http://www.digitalrowing.com*) has released RowPro, a software package that covers a few of the requests made by e-row users. This software focuses on 3D graphics making it aesthetically pleasing, but functionally not much more advanced than e-row. This means many of the feature requests, most of which are valuable in increasing rower performance, are yet to be implemented.

Given the lack of high quality software for rowing training, the relative ease with which data may be obtained from a Concept II, and the amount of rowers who may benefit from rowing specific training software, it is proposed to create an application that improves upon the functionality of existing products.

1.2 Terminology

Terms and abbreviations specific to computer science and rowing are used regularly throughout the report. They are identified here for reader reference:

500m split	The time it will take to row 500 meters at the current stroke power.
Concept II	A Concept II indoor rowing machine (model C).
Ergo	The term used by professional rowers for indoor rowing machines.
Java	A platform independent object orientated programming language.
Jar	A Java archive file. Contains a set of classes and their associated resources.
JVM	*Java Virtual Machine* The software employed by Java as a bridge between compiled Java class files and the underlying operating system.
PM2+	*Performance Monitor 2+* The unit that displays the rowers stroke data and allows retrieval of this data via the serial port.
Stroke	The term used for a single complete movement when rowing from the back of the rowing machine slide to the front and back again.
SWT	*Standard Widget Toolkit* Native widget functionality that is provided in an operating system independent manner.
XML	*Extensible Markup Language* A common format used for the interchange of structured data.

1.3 Project Objectives

The project was guided by a series of objectives that can be used to measure the overall success of the project.

The software should:

1.3.1 Retrieve data from the PM2+ and present it in a form suitable for the rower.

1.3.2 Support the competitive racing of human and computer boats.

1.3.3 Allow the storing and retrieval of training data.

1.3.4 Assist rowers to increase performance capability over time.

The project should:

1.3.5 Make a strong case for an increasing use of computer technology in sport.

1.3.6 Follow good software engineering practices.

1.4 Project Deliverables

The project deliverables are split into three categories (Basic, Intermediate and Advanced) representing the approximate difficulty of implementing each deliverable. Each deliverable is additionally associated with a number for traceability.

Basic Deliverables are essential functionality. The software must fulfil all basic deliverables.

1.4.1 Capture data from the PM2+ and convert it into a usable form.

1.4.2 Provide a simple graphical user interface (GUI) with a representation of a boat moving over water.

1.4.3 Provide a distance sign post beside the water.

Intermediate Deliverables are highly desirable functionality. The software should fulfil at least three intermediate deliverables.

1.4.4 Allow the racing of a custom distance or time.

1.4.5 Support constant speed pace boats.

1.4.6 Support the storing of rower training data.

1.4.7 Allow a previous workout to be used as a pace boat for the current workout.

Advanced Deliverables are useful features that should be implemented if time remains after the intermediate deliverables. The software should fulfil two advanced deliverables.

1.4.8 Support up to four human boats with optional shadows as pace boats.

1.4.9 Ensure any visual representations of the data are presented back to the rower in real time.

1.4.10 Ensure that the software is platform independent.

1.5 Project Plan

It is important that the project is planned well, so that any shortcomings or contingencies are identified at an early stage and can be dealt with accordingly. Figure 1.1 illustrates the plan that was developed and used throughout the project.

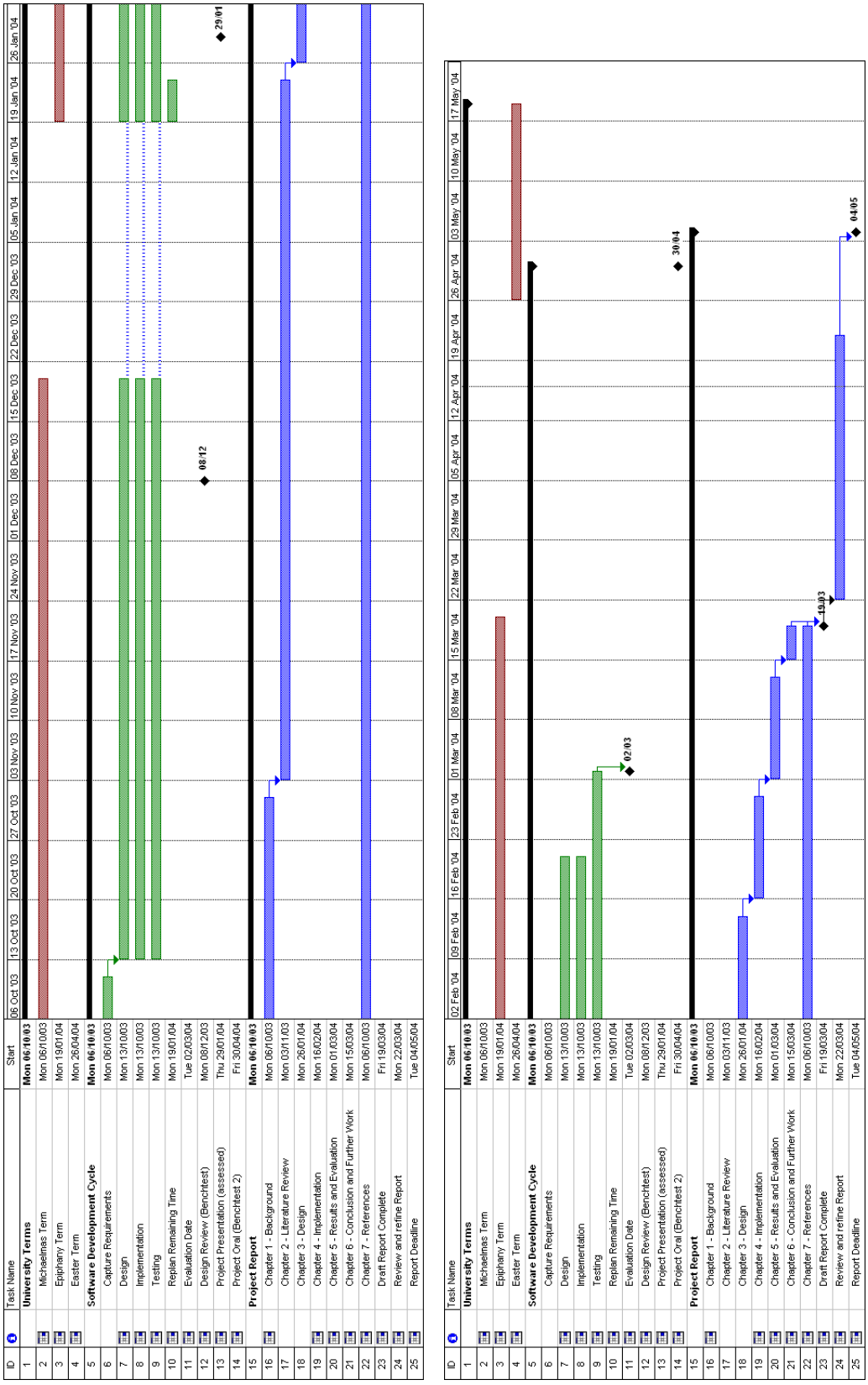

Figure 1.1: The Project Plan

1.6 Report Structure

The remainder of the report is structured as follows:

Chapter 2 – Literature Review
This chapter presents an overview of current research in the field of software engineering. The areas most relevant to the project are covered, namely computers in sport, the development process, requirements and user involvement and user interface considerations.

Chapter 3 – Design
This chapter details the architecture of the system and discusses major design decisions. The development model used is discussed and the design components traced back to the system deliverables.

Chapter 4 – Implementation
This chapter describes the key technical issues that were faced during implementation. Requirement and architecture realisation are demonstrated and the software development practices that were used discussed.

Chapter 5 – Results and Evaluation
This chapter presents the strategy and method used to perform the software evaluation as part of the Durham University Boat Club five kilometre ergo tests. The results are discussed and analysed with specific reference to the original project objectives.

Chapter 6 – Conclusion and Further Work
This chapter summaries the progress made throughout the project with specific reference to the project objectives and deliverables. The project management approach is reviewed and recommendations for further work are made.

Chapter 7 – References
This chapter contains a list of all references used throughout the report.

Chapter 8 – Appendix
This chapter contains additional information that did not fit naturally into the flow of the main text. It includes two UML class diagrams, a copy of the project brief, a copy of the evaluation questionnaire and the raw data from the user evaluation.

Chapter 2

Literature Review

Chapter 2 presents an overview of current research in the field of software engineering. The areas most relevant to the project are covered, namely computers in sport, the development process, requirements and user involvement and user interface considerations.

2.1 Computers in Sport

2.1.1 Obtaining the Data

The last several decades have seen a substantial drop in the price of computer memory and an equally significant increase in processor power. This has allowed sufficient data to be captured for the modelling of sports training and events [17]. In order to achieve the most accurate results, it is desirable that this data is obtained in the athlete's natural environment [27]. The Concept II was designed to simulate the rowing action as closely as possible (Hahn, Tumilty, Shakespear, Rowe and Telfored, cited in [27]), making it an ideal device from which to capture data. As the data is acquired electronically the likelihood of errors is reduced, time is saved and results can be stored in a form that can be easily assimilated [26]. Unfortunately "data acquisition ... is usually hand made, because its automatization is difficult" [19, p. 257]. This difficulty is clearly echoed by Nichols, Morgan, Chabot, Sallis and Calfas [30] who used specialist accelerometer recorders, Partridge and Franks [33] who customised a touchpad and Macfarlane et

al. [27] who developed specialist force transducers.

The reason acquiring data is so difficult is because it is ”a three dimensional problem not subject to constraints in many cases” [19, p. 257]. Just visualising a runner down a road makes one realise the sheer problems associated with such capture. Whilst suitable three dimensional methods have been proposed (Clayton, cited in [19]), they are expensive and as Hubbard [17, p. 55] warns ”researchers can fall into the trap of making such a complex mathematical model that its results are essentially uninterpretable, there being so many independent variables that their effects become inextricably intertwined”. Inesta et al. [19, p. 257] contest that data acquisition tools can be used effectively when the ”movements are constrained by means of their specific nature”. For example, when rowing on a Concept II the rower will not leave the machine, meaning movement is constrained and therefore measurement of power is feasible [27]. Furthermore, in sports in which it is the position of a ball that is more important than that of the athlete, a few systems have been developed that can systematically and objectively record the required data [33].

2.1.2 Benefits of use

There are many benefits of using computers in sport. They can act as a good motivation aid [10, 31], can substantially ease the burden associated with data collection [26, 33], can ensure that the athlete does not become too stressed as a result of exercise [26], can make recommendations from expert knowledge [31] and by the use of concurrent visual feedback can enhance performance [38]. The most common use, however, is that of performing analysis on large quantities of data. The Orlando Magic Basketball team made use of vast quantities of player data and data mining software to see which players should be selected based on past performances [35]. Whilst this can not improve players, it has enabled the team to pick the best performing players for each match, something that can be missed by ”subjective observations (eyeballing) of a performance” [33, p. 208]. In fact research suggests that subjective observations are both inconsistent and inaccurate [11]. Yet, there are still very few teams that employ any kind of systematic observation that can lead to increased accuracy in analysis [33].

With the development of drop down menus, multitasking operating systems and highly graphical displays, computers are becoming increasingly easy to use [17]. It is argued however that there are many unseen negatives associated with using computers in sport. Tisdall [42] highlights that users have high expectations of new technology

and quickly become frustrated when there are bugs in the software or the technology becomes unavailable for periods of time. Smith and Crome [37, p. 22] found the ”continual barrage” of new products and computer terminology very confusing, especially when working in non-technical industries. This is resounded by Tisdall [42] who highlights the confusion consumers face over different technology solutions with each vendor claiming to offer the fastest, most efficient solution. Furthermore it must be remembered that a computer system is only ever as good as the data entered, the user and his privileges on that system [35, 42]. Although these factors can be disconcerting for the some athletes and coaches it is generally accepted that, due to the dropping costs of computer hardware and capturing equipment [26], sports systems are now becoming inexpensive and easy to implement [19]. The benefits of such systems clearly outweigh the disadvantages as summarised by Smith and Crome [37, p. 21] ”the time saved and the mistakes avoided more than outweigh the negatives of ’computer gremlins’... It is up to us to embrace this technology and utilize its positive aspects”.

2.1.3 Using Technology in Sport

When the software has calculated the results of an evaluation, they should be made available ”as soon as possible” [17, p. 60] in order for the ”tool to be effective” [17, p. 60]. Feedback may then be given about a performance which is an important variable affecting the learning curve. Thus, it is important that the feedback provided is both accurate and reliable [33]. Smith and Loschner [38] state that, in the case of rowing, whilst any valid feedback will improve novice performance, the elite will require accurate information for the detection of errors in a performance. Partridge and Franks [33] add that the feedback should be put into context with performances from the current year or season. It is this area in which computer data and analysis can help to provide excellent quantitative feedback. Coaches should also be trained in how to provide feedback and players trained in how to use it [33]. That is, whilst it is useful to have accurate and reliable feedback, it must be utilised and interpreted correctly for maximum benefit.

Although some excellent computer technology has been developed, very little of it is benefiting sport itself [26]. This could be due to financial reasons, but is more realistically due to the lack of direct dependence that sport has on technology [26]. Whilst effective training that is aided by computers can be important, it cannot replace the natural ability of athletes. It can only help them to perform nearer their maximum [26]. This is reiterated by Restivo [35] who notices that despite modern training software

offered through a web interface, there still seems to be little take up. From personal experience this is both time consuming (as results must be recorded at a training session and transferred to a computer at a later date) and very limiting in the analysis that may be performed once the data is entered. Nigg [31] suggests that, although the technology may be attractive and motivational, it is questionable how many athletes it is reaching. He indicates this is probably because the research has been mainly grant funded, further limiting its exposure to the general population. In order to reach large portions of the public the technology must reach them in their homes. To achieve this, companies in the fitness market need to invest in making suitable products for home training, hoping that amateur athletes will spearhead their use [31].

2.2 The Development model

2.2.1 The difficulty of writing software

In order to develop high quality software, careful attention must be paid to a

> *precise specification of the requirements that the software must satisfy, decomposition of the software into components, design and documentation of module interfaces, precise documentation of internal module design decisions, writing well structured code and disciplined testing* [32, p. 54].

Even when following these careful steps, as Brooks [5, p. 10] famously wrote, there is no ”Silver Bullet”. The difficulty that arises in writing software is, according to Jackson [20], because of the differences between the formal world of computer programming languages and the informal world in which the problem is situated. Turski, cited in Jackson [20], proposes another difficulty – many problems are one-off problems, meaning there is little experience in solving problems within that domain. Although software development is clearly difficult, the process can be eased by carefully following a development process and using effective development tools.

2.2.2 Traditional models

The waterfall model was the first published occurrence of a software development process [39]. Within this model, one phase is completed before starting the next and, as such, there is a temptation to quickly progress through the earlier stages to reach the implementation. Parnas contests such practices by saying that time spent on the earlier stages, where he recommends use of mathematics in the requirements document, is ”handsomely rewarded in the later development stages” [32, p. 54]. However it was quickly realised that the waterfall model was very limiting, especially when dealing with clients who frequently change the requirements. Therefore a backtracking version of the model was proposed to allow the reassessment of previous phases. Tsai, Stobart, Parrington and Thompson [43] argue this does not solve the original problem as the backtracking will take time and can cause the product to be delivered late with increased costs. The problems associated with the waterfall model are repeated by Ainger and Schmid [1, p. 106] who found that ”failed projects followed more rigorously than others the waterfall project lifecycle”.

Given that the software engineering field is still less then four decades old [13] and that it has spent much of this time failing to deliver projects on time and on budget [1], it was clear that many more development processes would be adopted. The Ainger and Schmid helical model uses the principles of concurrent engineering, meaning certain phases of the waterfall lifecycle can be carried out concurrently [1]. In theory this could lead to a shortened development time and a reduction in cost, but in reality this time is lost when an earlier phase makes a substantial change [1]. A more popular development model is the spiral development process. It has been generally praised for its concentration on good software engineering practices as it considers the development cycle in a spiral fashion, where the four quadrants represent objective setting, risk assessment, development and planning [39]. Each development phase is evaluated in every quadrant and, unlike the previous models, the project may be terminated at any phase ensuring project costs do not run out of control [39]. It is, therefore, used regularly when organisations need to undertake high risk projects.

2.2.3 Incremental models

The commonality between the helical and spiral model revolves around the belief that software development should be a series of formal steps. The incremental development

process was developed to achieve higher flexibility and to better satisfy customers' actual requirements [14]. In this model the system is delivered in a series of stages, each building on the functionality of the last. The first advantage of such a model is that the requirements can be prioritised, meaning the client can choose the functionality that is most important to be delivered first [14]. Second, as "it is often difficult to obtain a complete set of requirements" [44, p. 3] and there will undoubtedly be changes within development that lead to new requirements [41], the ability to change requirements is understandably very beneficial to both developer and client. Third, if a project has a deadline that cannot be missed then functionality can be easily postponed to ensure the deadline is met [14]. Fourth, the first development increment can be used as a proof of concept to gain management and client confidence [41]. Fifth, plans can be adjusted to reflect user feedback at the end of each increment, ensuring the software is closer to what the client requires [14]. Last, the "client feels closer to the software development process because they have a powerful common communication medium (the prototype) that they can discuss with the developers" [44, p. 7]. Increased communication will bond the client and developer, which can result in increased client commitment to the project [44].

When developing software using the incremental process, Stephens [41] advises that time should be taken to architect and design the system. This does not need to be finalised but will avoid much redevelopment work later, saving both time and money. Stephens [41] highlights another good point by stating that care must be taken to avoid scope creep where the customer starts asking for functionality that has little to do with the original project objectives. This is clearly undesirable as the client may have lost touch with the original project objectives and the developer's morale will drop as they get a feeling the project will never be completed. Given the benefits over the negatives of such development, it is clear why many organisations (66% in 1997 [44]) are using an incremental process. However Verner and Cerpa [44] claim that the waterfall model can produce a more robust system and can be preferred by management as it offers a greater level of control, develops more robust systems and produces more maintainable systems. Therefore, the relative merits of the waterfall and incremental approach should be considered in the light of the client, developer and management, before the most appropriate model for the software development is chosen.

2.2.4 Test and reuse driven models

Extreme programming advocates the use of test-driven development [2]. This is a process in which the test case that the code must pass is written before the code itself. Whilst this development process does lead to an increase in development time, this does result in better quality code [12]. This is a direct result of the process encouraging constant testing throughout each phase. The V model of testing, conversely, only promotes testing at the end of each phase [43]. When the deadline is close and programmers think they have done a good job, the testing time is often compressed, ultimately leading to poorer quality code [12, 32]. Test-driven development has the additional benefit of increasing program comprehension by encouraging programmers to explain their code through test cases and through the code itself [12]. With test driven development the test cases must be updated to make changes to the system and therefore much of the documentation is, by the nature of the process, constantly updated. This is extremely useful for an organisation as software will deteriorate over time if changes are made and they are not documented [32].

None of the processes described so far have any formal focus on creating reusable assets. Sommerville [39] suggests component-based software engineering, in which the system is designed with reuse in mind, both from a consuming and producing point of view. Lee, Kang, Chae and Choi [25] argue that reuse is normally not the primary goal of development, only a by-product. To address such problems domain orientated methods have been developed in which common abstractions are identified across the applications of a domain, helping to identify reusable assets [25]. Implementing a long term reuse strategy has clear financial benefits because assets do not need to be redeveloped [39].

2.3 Requirements and user involvement

There is irrefutable evidence showing that software produced does not often meet the customer's requirements [6]. This, according to Saiedian and Dale [36], is due to the developer not understanding or addressing the users' real requirements. This is reiterated by Brooks [5, p. 17] who states "the truth is, the client does not know what he wants" and it is therefore "necessary to allow for an extensive iteration between the client and the designer" [5, p. 17]. Involving the client will inevitably increase the cost of the development but research shows the benefits gained far outweigh the costs

induced [34]. Furthermore, if the project is in danger of missing its deadline it should be the client who chooses the functionality that is to be compromised [6]. Critics argue that high user involvement can turn the development process from a creative discipline to a uninspiring exercise. Whilst this may create a better product for the client, it cuts down on innovation [34] and can be very tedious, as well as time consuming, for non-technical clients who may just require a quick solution [6].

Brooks [5] believes the most difficult and therefore important stage of software development is the gathering of requirements. They should be consistent, complete and correct [45] and as Jitnah, Han and Steele [21] argue, developed through an iterative process over time. Consequently prolonged developer-client communication is required and serious issues often arise due to the inadequate way in which the communication is conducted [36]. Given the views that ”product definition is unavoidably a battle of trade-off” [36, p. 421] and it is ”the customer that decides our fate whether we like it or not” [36, p. 427], the developer must be both careful what they say and how they say it. In particular, the use of domain terminology specific to computer science should be avoided [36].

Communication with the client is clearly very important to the project success and as such, it should continue throughout the development process. Under no circumstances should it stop once the requirements document is finished [36]. Given the complexities that can arise due to the number of people working on a project, software has been developed to aid the communication process. ClientPro is one such piece of software and it is built upon Microsoft Access to ensure at least some familiarity for novice computer users [22]. Requirements may be entered and then modified by any party who thinks they may be ambiguous or do not fully satisfy their requirements for the system. ClientPro also aids traceability, increasing the likelihood of a set of correct requirements [22]. However many clients prefer to interact face-to-face and will not enjoy the software abstraction. This can lead to poor relationships and could ultimately put the success of the project at risk.

2.4 User interface considerations

Very few organisations employ specialist user interface designers and as such the responsibility of interface design often falls to the software engineer [39]. This creates a problem because, although software engineers can construct an interface, they often

struggle to produce an appropriate interface for the end user [39]. The waterfall model of development does not help this problem, as users are ”unable to grasp the functions and support the software would provide until they view a finished or nearly finished product” [28, p. 428]. As a result of this issue prototype development was adopted by many organisations so that user feedback can be incorporated into the design process [28]. Such a development process ensures that the end user achieves a greater say in how the interface will function. This is most beneficial, because it is the end user that will ultimately be using the system. More recent developments have seen user-centred design practices that put greater emphasis on user capabilities of the product [23]. These improved development methods have lead to greater realisations in user interface design by considering a wider range of users and accommodating them in the interface design process.

When designing a user interface it is important to consider user familiarity, consistency, recoverability, user guidance and user diversity [39]. These are important factors in the success of an interface and, as such, should be tested through user evaluations. One problem is that user evaluations are often expensive and nearly always time consuming. It is therefore important that a strategic decision is made about the amount and type of user evaluations and thus the cost, compared with the potential market for a product [23]. A popular choice for evaluations is cognitive walkthroughs, the basics of which can be quickly learnt from an expert [24]. It is a quick evaluation process and produces constructive advice leading directly to suggestions for improving the design [24]. The results of such an evaluation can then be incorporated into the interface design ensuring a more usable product, with little expenditure in terms of cost and time.

2.5 Chapter Summary

This chapter has presented the key issues likely to impact a project in the sports domain of software engineering. The difficulties associated with capturing data were highlighted, the benefits of use demonstrated and the lack of computer technology in sport explained. The possible development methods were then discussed with particular attention paid to traditional and incremental models. User involvement was subsequently addressed and the considerations that must be made by both parties examined. The chapter concluded with an overview of user interface considerations with reference to the development model.

Chapter 3

Design

Chapter 3 details the architecture of the system and discusses major design decisions. The development model used is discussed and the design components traced back to the system deliverables.

3.1 Design Method

3.1.1 Development Model

When choosing a development model, it is important that the type of project is considered. In this case the project is working with a client to produce software with discrete levels of functionality.

The literature review identified three strong candidates for the development model; the waterfall model, the spiral model and the incremental model. The waterfall model guarantees the early phases of development are complete before implementation begins. Whilst this can create dependable software, it does not allow the flexibility that is required should the client change a requirement. The spiral model places strong emphasis on the risks associated with development, which can sometimes lead to project termination before costs spiral out of control. Whilst this model is appropriate for industrial environments, it is not suitable for academia where the project must be completed in a dictated time frame. The incremental model dictates the implementation of software

functionality in stages. This is most useful when working with a client, as they can determine the functionality that is most important and thus needs implementing first. Furthermore as the implementation of a requirement is not started until it is needed, requirement changes are easy to accommodate. This makes the incremental model of development suited to this project and it was therefore used to guide the development.

3.1.2 Software Reuse

Reuse within software occurs by exploiting similarities within requirements and/or architectures. Whilst this may lead to an increase in development time and decrease in asset performance, it beneficially leads to increased productivity, quality and business performance. It is therefore desirable to produce and consume reusable assets wherever possible in the project.

Given that the project does not form part of a formal reuse process and any assets produced remain the property of the University of Durham, it was deemed inappropriate to produce any reusable assets. That is, whilst assets could be produced, the extra time required to develop them would be poorly spent as they were unlikely to be reused. The project does, however, present several opportunities for consuming assets. For instance, the software must communicate with the PM2+ through the serial port and as serial communication is not part of the standard Java development kit, a third party solution must be used. Solutions were found to be available from Sun (*http://java.sun.com*) and SerialIO (*http://www.serialio.com*), both similar in the functionality they provide. Sun's communication solution was chosen as it is produced by a reputable organisation and is available free of charge.

The software must also read and write XML files. As the data will be stored in Java objects and then written to a XML file, it would make sense to leverage one of the existing frameworks that convert Java objects to XML structures. Due to the amount of possible solutions found, an IBM technical article [40] was consulted. After reading the article, dom4j (*http://www.dom4j.org/*) was chosen as it offered the best performance for the required functionality.

3.1.3 Test-driven development

Test-driven development results in higher quality code but at the cost of increased development time. Such a development strategy can help to increase the reliability of the software by ensuring that the code is performing as expected. In the case of this project, test-driven development will help to ensure the analogical representations are accurate. Should this not be the case, the software will fail to fulfil its purpose as the rowers will be looking at incorrect data and any according performance adjustments may be wrong. It is therefore imperative that a comprehensive testing strategy is used throughout development.

3.1.4 Vision

Currently, rowers using a Concept II use the PM2 to see progress over a time or distance workout. Whilst this provides the necessary information, background research in Chapter 1 suggests that rowers can find it hard to perform calculations whilst rowing. Therefore, the aim of the system is to provide a visual representation of the data that will instantly inform the rower how well they are performing. It is envisaged that over time, as rowers adapt to the visual representation, the software will become the performance feedback device of choice. It is further hoped that the software will result in increased motivation to perform well when close to a target. This vision was used to guide the design and implementation of the system.

3.2 Architecture

3.2.1 Implementation Language

In order for the design to be optimised, it was decided to make a decision on the implementation language as early as possible. Three main candidates were identified; C++, Microsoft's .NET and Java. C++ has proven itself well over many years and offers the distinct advantage that it can be compiled to native code. Unfortunately the language is not platform independent and thus achieving deliverable 1.4.10 would be infeasible. Microsoft's .NET is a new language and is yet to prove itself. It is not

platform independent but does offer a Linux port (this is still in development though – *http://www.go-mono.com/*). As such, it is too immature and it would be likely that a series of in-depth technical issues would be faced. Java has proved itself well in recent years and is platform independent. Furthermore it is well documented, well supported through forums and employs managed memory allocation. This makes it a ideal choice for the project.

3.2.2 Architectural Overview

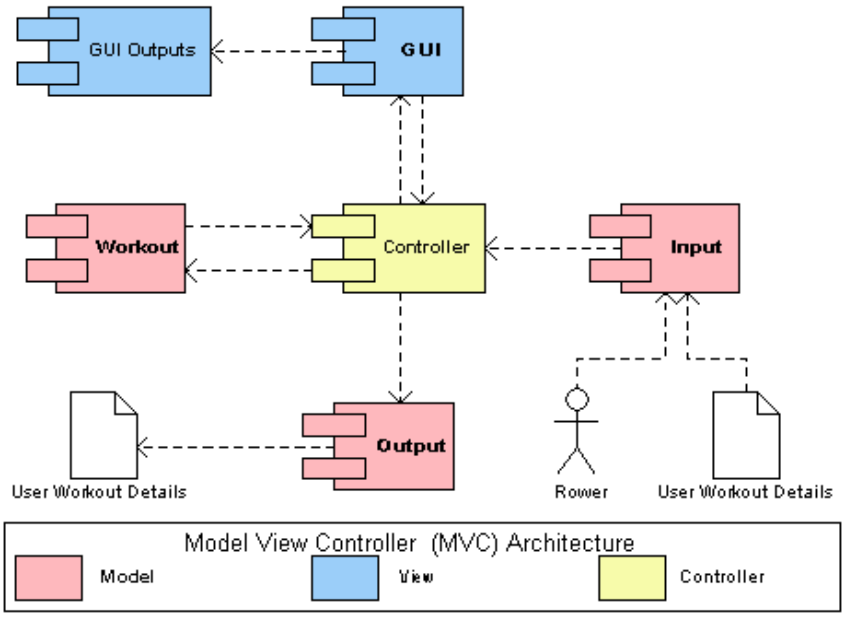

Figure 3.1: PC-Rower Architecture

Figure 3.1 shows the centralised (or event-loop) architecture that is very similar to the model used in soft real time systems [39]. It has a controller that manages the execution of the program, retrieving the latest data from the input module and updating the GUI and any output devices accordingly. The architecture utilises the Model-View-Controller design pattern, keeping the program data, the GUI and the controller distinct. This offers among other things, clarity of design and ease of growth. Together the architecture patterns create an efficient way to implement the system that aids comprehensibility and expandability, whilst ensuring real time operability.

Each of the modules has a distinct purpose:

Controller	The controller manages the execution sequence of the program.
Input	The input interface handles all sources of input to the system. For example, data from the PM2+ or data from an XML workout file.
Output	The output interface handles all outputs other than the GUI. For example the saving of a workout in XML format.
Workout	Workout encapsulates information about the workout to be performed.
GUI	The GUI handles the menu system and windowing.
GUI Outputs	GUI Outputs present the input data in analogical form . For example, a boat moving down the river.

Modules with notable design features are discussed in Section 3.3.

3.3 Module Design

3.3.1 Controller Module

The controller module manages the execution of the program by handling input devices, output devices and GUI interactions. When the user is rowing, the module retrieves the latest input data and passes this on to the GUI and any output devices. These should then delegate the responsibility (for example the GUI delegates all updates to registered GUI outputs) or use the data to update themselves. This retrieval and update loop will continue until the rower finishes or abandons the workout. Although this is processor intensive, it will provide the best performance in terms of frame rate, which is an important factor in ensuring the boat movements are both smooth and in real time. This approach is similar in style to that which is used by computer games where the maximum frame rate is more desirable than spare processing capacity [16]. In the case of this project the user of the software will be rowing and it is, therefore, unlikely they will want to perform additional processor intensive tasks at the same time.

The performance of the system could be improved if the workout and controller modules were combined. However it was decided this would represent bad design practice as it would have two derogatory effects elsewhere within the development process. First, due to the poor abstraction within the module, it would have been more complex to write the code for the system. This would increase the development time and

likelihood of bugs within the module. Second, the system would be more complex to maintain due to poor modularity. As this is an academic project with limited scope, maintenance of the system would seem unlikely. However project objective 1.3.6 states good software engineering principles should be followed and thus the system should be implemented with a view of it being maintained. For these two reasons, it was decided that keeping the system conceptually simpler was more important than small performance gains.

It was considered whether the IO device handling part of the controller should be abstracted out into a separate module. It was decided however that this would be a poor design decision due to two main reasons. First, the level of coupling that would be required would considerably increase. This would lead to increased complexity in maintaining the two modules. Second, the amount of functionality that the IO device handler provides is low and thus does not warrant an individual module. It was therefore decided that the IO device handler should be part of the controller module in order to keep the system architecture as simple as possible.

The controller module class diagram is included in the appendix, Figure A.1 for reader reference.

3.3.2 Input Module

The input module manages the input devices that provide the data used to drive the system and update the outputs, such as the GUI. As all the input devices are contained within the input module they will be programmed to a common interface and produce a common object encapsulating the stroke data. This promotes low module coupling, as the other modules within the system are not concerned with the actual input device type, just the data that they are producing. Low module coupling along with simple conceptual encapsulation were considered the two most important factors when abstracting the modules. It was therefore important that this module was designed to fit consistently with the rest of the architecture.

The most common source of input within the system is from the PM2+. This handles the connection to the serial device, the retrieval of data and the conversion to a standard form. The second input device, a fixed speed pace boat, will produce the required data from a chosen input time or 500m split. The third input device provides the ability to use a previous workout as a pace boat. In order to have sufficient information to create

a pace boat from a previous workout, a suitable file format and structure had to be decided upon. XML was chosen as the file format as it provides a platform independent way to store structured information in a standard form. Furthermore there are many XML parsers freely available for Java which can be leveraged to significantly reduce development time. The XML workout format, which is shown in Figure 3.2, contains the basic rower and workout information along with details of each stroke taken. This provides sufficient information to construct a pace boat that accurately represents the stored workout.

```
<Session>
   <Rower>
      <Name>George Palmer</Name>
   </Rower>
   <Details>
      <Date>2004-2-9-20:9:42</Date>
      <Type>1</Type>
      <Distance>20</Distance>
   </Details>
   <Stroke>
      <Time>0.0</Time>
      <Distance>0.0</Distance>
      <Split>0.0</Split>
      <Power>0.0</Power>
      <SPM>0</SPM>
      <CaloriesPerHour>0.0</CaloriesPerHour>
      <HR>0</HR>
   </Stroke>
   <Stroke>
      <Time>0.952</Time>
      <Distance>1.0234</Distance>
      <Split>160.32</Split>
      <Power>134</Power>
      <SPM>22</SPM>
      <CaloriesPerHour>256.3</CaloriesPerHour>
      <HR>0</HR>
   </Stroke>
    ...
</Session>
```

Figure 3.2: A sample workout XML file

The input module class diagram is included in the appendix, Figure A.2 for reader reference.

3.3.3 GUI Module

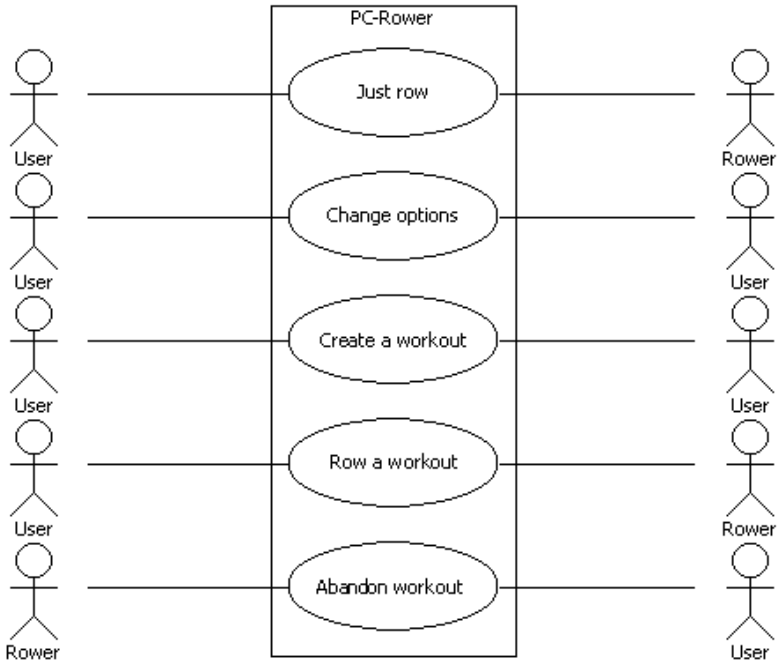

Figure 3.3: PC-Rower use case diagram

Figure 3.3 guided the development of the GUI by establishing the functionality that the user should be able to access. Whilst this provided a useful framework for the GUI development, it must be remembered that a user of the system could potentially be a rower in the workout. It was therefore decided that any human-computer interaction by a party other than those rowing in the workout, would be seen as a failing in the GUI design. To avoid such problems a series of preventative features were adopted.

A workout can start either when a rower takes a stroke or after a countdown period elapses. The first stroke approach offers the rower a significant advantage in allowing them an infinite amount of time in which to start rowing. This is very useful if the rower is unsure of the duration required to prepare for rowing after using the computer. Whilst suitable for one rower, the first stroke approach can be problematic for several rowers. Should one rower start before the others are ready, the timer for all rowers will start and consequently the end results be inaccurate. The countdown approach overcomes such problems, as any rower starting prematurely will not have their efforts processed by the PM2+. The countdown approach is therefore much more suitable when there are several rowers, although the rower setting up the workout must allow ample time to prepare for rowing. Given the relative advantages and disadvantages of each start method, it was decided that the user should be able to make the choice each time a

workout is created.

When a workout is created the countdown could begin immediately. It was foreseen though that this could take some users by surprise, meaning that they would not be ready to row in time. It was therefore decided that a dialog box should be used to inform the user that a countdown is about to begin. It would be logical as well, to let the user decide how long the countdown lasts. This gives the user the opportunity to allow ample preparation time before starting to row, and thus cannot be put at disadvantage by using the computer. It was decided that this option should be implemented as a program option as it is likely that a user will want to change this value for all workouts created. It was further decided that the countdown should be displayed on the PM2+ so that all rowers in the workout can see it. This will be useful if, for example, the rower setting up the workout blocks the computer screen when moving back to their ergo.

When a workout finishes the results window is displayed. It was decided that this should be a modal window so the user cannot close the program without closing the results window first. This not only ensures that the user will see the results window, but that the result window will remain on screen until the user has recovered and is ready to continue using the computer. The results window will display the results in a graphical fashion, using the same boat representation as used on the river. This will allow the rowers to quickly see how they performed relative to the other boats, without having to read any text. A prototype of the window is shown in Figure 3.4.

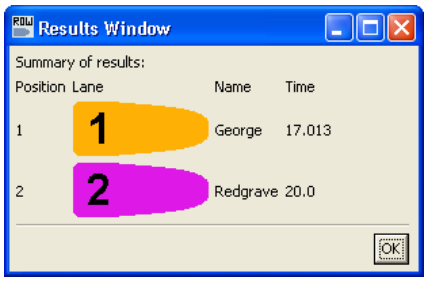

Figure 3.4: Prototype of the results window

3.3.4 GUI Output Module

It was decided that the GUI output module should be distinct from the GUI. Whereas the GUI module will handle the menu and windowing system, the GUI output module will handle the visual representation of the data. This is a crucial conceptual difference as there is only one GUI, but many GUI outputs. Separating the GUI and GUI outputs

into modules also means that any additional GUI outputs can be implemented without knowledge of the GUI. It was decided that this separation would be a good architectural device for aiding any further developments of the system.

The primary example of a GUI output to be included in the system is the river that shows boats moving over water. A prototype of this is shown in Figure 3.5. A representation of the PM2 will also be provided, should the user wish to have this information on screen as well. A user of the system will primarily be concerned with the usefulness of the GUI outputs and it is, therefore, vital that the outputs are simple, clear and motivational. As such, it was decided that the user should have the ability to determine the distance of the river shown within the window. It was further decided that the GUI should follow the leading human rower and not the leading computer boat. Both of these will allow the human boats to be kept on one screen, keeping motivation high as rowers can see the relative distances between boats. Keeping the rowers motivated and performing well is central to the success of the project in meeting its objectives.

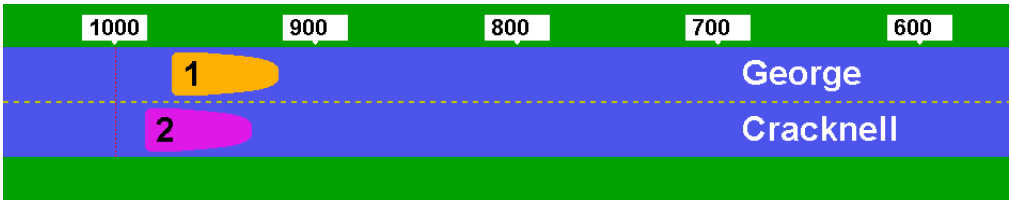

Figure 3.5: Prototype of the river

3.4 Chapter Summary

This chapter has outlined the development strategy that will be used throughout the remainder of the project. An incremental model was chosen as it both flexible and dictates the development of requirements in stages. This is valuable as the project involves working with a client, who may change the deliverables, to produce discrete levels of functionality.

The deliverables were used to produce an architecture for the system and the architecture was designed to permit implementation in an incremental fashion without distorting the design. This ensured any work in fulfilling early deliverables was not made obsolete by the requirements of more complex ones. Figure 3.6 shows the link between the architectural components and the project deliverables.

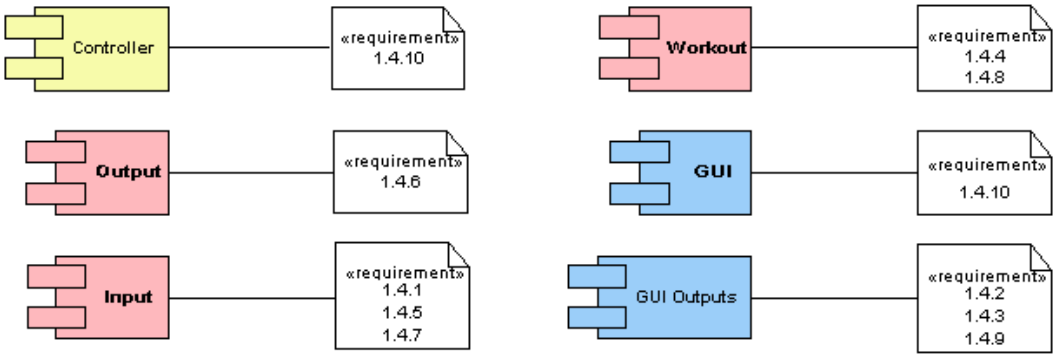

Figure 3.6: Architectural requirement trace

The chapter finished by looking at some of the more interesting design decisions, which were discussed with reference to design simplicity, system performance and usability.

Chapter 4

Implementation

Chapter 4 describes the key technical issues that were faced during implementation. Requirement and architecture realisation are demonstrated and the software development practices that were used discussed.

4.1 Technical Issues

4.1.1 PM2+ data retrieval

A complex issue was encountered in the retrieval of the PM2+ data. Given that Concept publish an outline of the protocol used [9] and further work on reverse engineering the protocol has been conducted [4], it was assumed that retrieving the data from the PM2+ would be trivial. This was not found to be the case, however, as the data received from the PM2+ arrives as a series of four bytes in reverse order. Furthermore the data needs to be converted from hexadecimal and the Java method `Float.intBitsToFloat()` called to return the floating point value of the underlying binary bits. Sending data to the PM2+ requires the reverse of this sequence.

The reason for such complexities is Java's big-endian architecture (this is where the most significant byte has the lowest address). At the time of the PM2+ protocol design the most popular programming language would have been C++ and so a little-endian based protocol would have been a logical choice. As the difference in language archi-

tecture was not known at development time, considerable effort was taken attempting to solve the issue. It was only through a search on the Internet that a similar problem was found and the necessary changes could be made.

4.1.2 GUI Outputs

Creating the river GUI output was more difficult than envisaged in the design. This was due to the sequence of moves that must be performed by the river, signs and boats in order to create a correct representation of the boat positions on the river. When a workout starts the boats must move from the start line to the centre of the screen, at which point the boats stop but the river carries on moving until the start line moves off the left of the screen. The river then stops but the signs must continue to move to give the impression that the boats are moving along the river. It was decided in Section 3.3.4 that the GUI should follow the leading human boat and thus all boat positions relative to the leader must be updated (indeed the leading boat itself can change). When the boats approach the finish line the river must once again start moving so that the finish line comes into view. The boats should then stop once they have passed the finish line.

Boat ”jerking” occurs when a boat movement is over a too great distance for a given time period. Rather than create an illusion of a smooth boat motion moving over a river, it resembles that of a slow updating program. To avoid such problems it was decided to move a boat to all intermediate positions in any movement, thus creating a smooth transition. The intermediate positions can be calculated by using the rower 500m split, which is updated once per stroke by the PM2+. This method will work well unless the rower slows down, in which case the boat will move backwards. This is because the 500m split is not updated until the end of a stroke and therefore the program will calculate the boat positions based on a faster 500m split than the rower is actually achieving. Backward moving boats are clearly not desirable as it may cause the rower to question the validity of the program and stop the workout. Given that in any workout a rower must slow down at some point and therefore will experience backward moving boats, it was decided that boat smoothing should only be used on computer boats where the stroke data is available for all strokes and the boat positioning can therefore be accurate.

Given the complexity of the issues discussed, much time was spent developing the river output and a lot more time was spent testing that it was correct. This was felt to be critical to the success of the project as the system would fail to fulfil its purpose if

the river representation was incorrect.

4.1.3 Performance

Initial performance tests showed that the data retrieval from the PM2+ and subsequent processing took around 40ms, implying a GUI update rate of up to 25 frames per second was possible. This was deemed to be adequate as the lag between a rower reaching a distance and seeing it on the computer would be sufficiently low that it would be hard to notice. However when proof of concept tests were performed with Java's Swing, it was found to be performing very poorly to the extent that the lag was noticeable. It was therefore decided that an alternative GUI solution must be used. SWT was chosen as it offers a low level, platform independent framework that utilises the native widgets of the underlying operating system. Due to the low level at which it operates the widgets are extremely fast. When used with the test system it was found to allow an update rate of over 20 frames per second, and it was therefore an appropriate choice for the GUI framework.

As the software will be used for several rowers the cycle time could increase, for instance, to 160ms for four rowers. This is clearly too slow for updating the GUI and it was consequently decided to add an option that allows partial retrieval of the PM2+ data. That is, only the time and distance values are retrieved from the PM2+. This requires two PM2+ queries and results in a more acceptable cycle time of 80ms for four rowers. It was decided that the user should decide if they wish to make this trade-off and the option was therefore added to the program option dialog. In doing so they will get a greater frame rate, but will sacrifice any information on pace or heart rate.

When the basic deliverables (1.4.1–1.4.3) were complete the performance of the system was analysed with respect to memory usage, which is a valuable factor in identifying memory leaks within a system. The tests found that the memory requirements for the system were increasing by 1MB every 10–15 seconds, which was considered very large and not acceptable for the final performance of the system. Upon further investigation the memory increments were found to be required by the GUI when updating itself. As SWT is a new technology and there is little in the way of performance documentation, a newsgroup post was made to the IBM SWT forum [18]. It was found that in SWT, any resources allocated to an object must be released by using a `dispose()` method. This was not called within the code as Java does not usually require the programmer to be concerned with memory management. However as SWT accesses an underlying

native library `dispose()` should be called or else the widgets will re-access the raw data every time an update is called. This was the case with the boat and sign images being reloaded 20 times per second. Once the necessary alterations were made memory requirements for the system remained static throughout the runtime of the program, ensuring the program was suitable for computers with smaller amounts of RAM.

In order to maximise the general performance of the system an article on Java performance by McCluskey was consulted [29]. This laid down several guidelines that were followed throughout implementation to ensure an efficient system was developed. It was essential that this happened, as any performance gains would help to increase the GUI frame rate. This in turn, is a key factor in ensuring the GUI Outputs looks real time and smooth. To further increase the speed of the system, a Java compiler (*http://www.excelsior-usa.com*) was used to compile the code into a native executable. Whilst the executable would only work on a Windows platform, it significantly increased the performance of the system as it did not require a Java virtual machine (JVM). It should be noted that compiling the code does not inhibit the fulfilment of deliverable 1.4.10 as the original Java files can still be used.

4.1.4 Usability

An additional benefit of SWT is that the native widgets provide a familiar C++ style GUI for the user. This improves the usability of the system as the user will not have to contend with Swing, which has been criticised for being both too slow and confusing to novice users of computers. For reader reference an example of a typical SWT (left) and Swing (right) widget are shown in Figure 4.1.

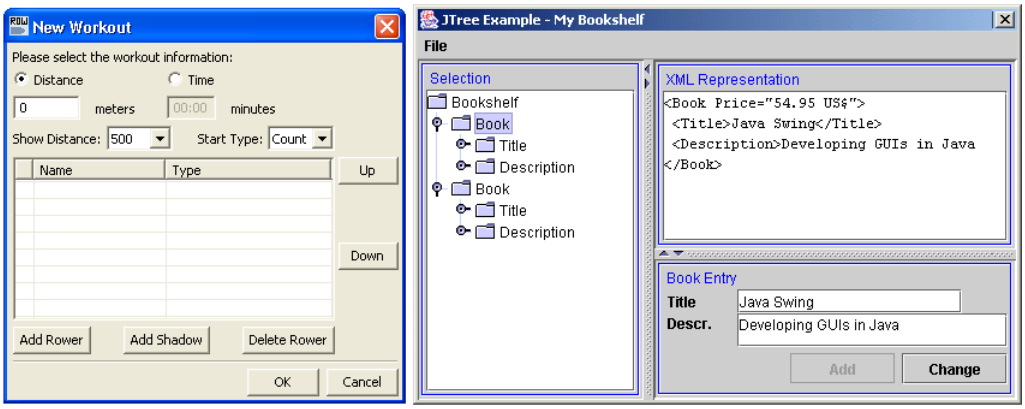

Figure 4.1: An example of a SWT and Swing GUI

As a result of SWT utilising native widgets, the GUI will look slightly different on each

platform. However when the final system was tested the only differences were found in box and button positioning to the extent of a few pixels, which will be unnoticeable to the average user. Therefore although SWT has a few aesthetic issues it is a much better choice both in terms of performance and usability.

It was realised at the time of implementation that should the program crash due to an uncaught exception, then the PM2+ units would reset. This would be disastrous for a rower in the middle of a workout, as they would lose any workout data both on the computer and on the PM2+. To avoid such situations a fail safe system was implemented that meant the PM2+ would carry on logging if the program crashed. However, if the user exits the program, then the units should be reset as they are no longer required. To solve this conflict of interests, a confirm dialog was added to the exit function of the program that informs the user of the consequences of their actions. This should prevent any accidental exiting of the program and loss of data whilst completing a workout.

4.2 Requirement and Design Fulfilment

4.2.1 Basic Deliverables

The basic deliverables were the first to be delivered. Deliverable 1.4.1 formed part of the input module as it was concerned with retrieving PM2+ data, whereas deliverables 1.4.2 and 1.4.3 form part of the GUI Output module as they were concerned with displaying the data visually. A skeleton GUI and controller module were created to allow the system to function in the architecture described in Section 3.2. It was crucial that the architecture was used at the earliest possible stage as the addition of further functionality can then be achieved with just a few changes to the existing code. A screenshot of the basic system is shown in Figure 4.2.

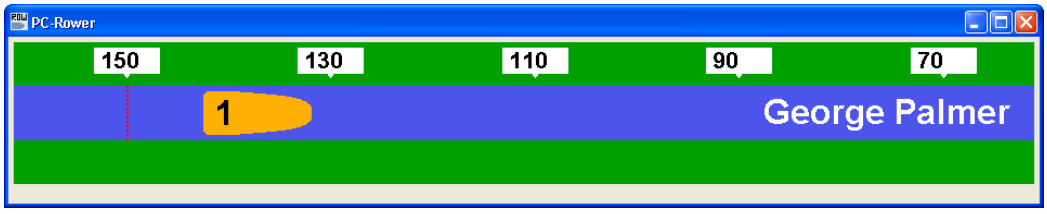

Figure 4.2: Basic Deliverable Screenshot

4.2.2 Intermediate Deliverables

The intermediate deliverables made the system more complete by adding a series of core features. Deliverable 1.4.4 brought the workout module to the architecture as well as extending the GUI module to support the addition of workout information. Deliverable 1.4.6 added the output module to the architecture and once more extended the GUI module to support the functionality. Deliverables 1.4.5 and 1.4.7 extended both the input module by providing alternative input devices, and the GUI module by adding the required input facilities. A full implementation of the controller was carried out to complete the architecture to the design described in Section 3.2. Screenshots of the intermediate system (Figure 4.3) and the new workout window (Figure 4.4) are included.

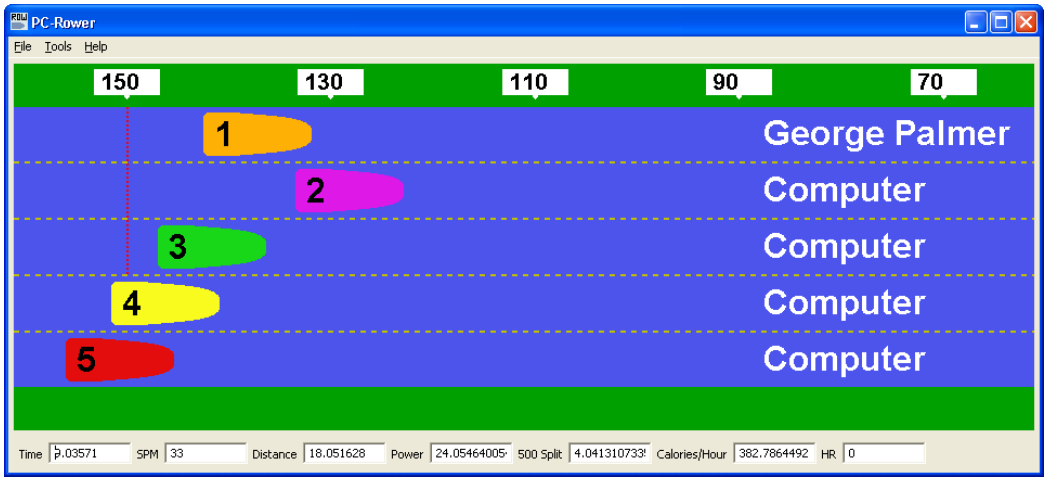

Figure 4.3: Intermediate Deliverable Screenshot

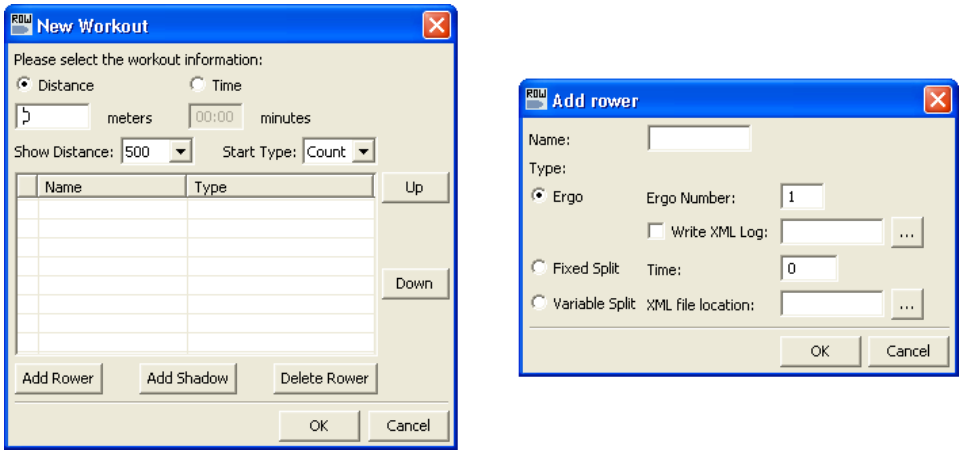

Figure 4.4: New Workout and Add Rower Window

4.2.3 Advanced Deliverables

The advanced deliverables added code to some modules but did not alter the architecture in any form. Deliverable 1.4.8 involved extra work in the GUI Output module and a few additions in the workout module. Deliverable 1.4.9 was considered throughout development and formal tests were conducted to ensure the system was operating in real time. An account of the problems faced and the solutions used can be found in Section 4.1.3. By careful development using Java's platform independent features (for example the class `File` provides a platform independent way to access files) it was hoped that deliverable 1.4.10 would be achieved. When tested under Linux, it was found a few minor alterations needed to be made to fulfil the deliverable (these are detailed in Section 5.3.3). A screenshot of the system with shadow pace boats is shown in Figure 4.5.

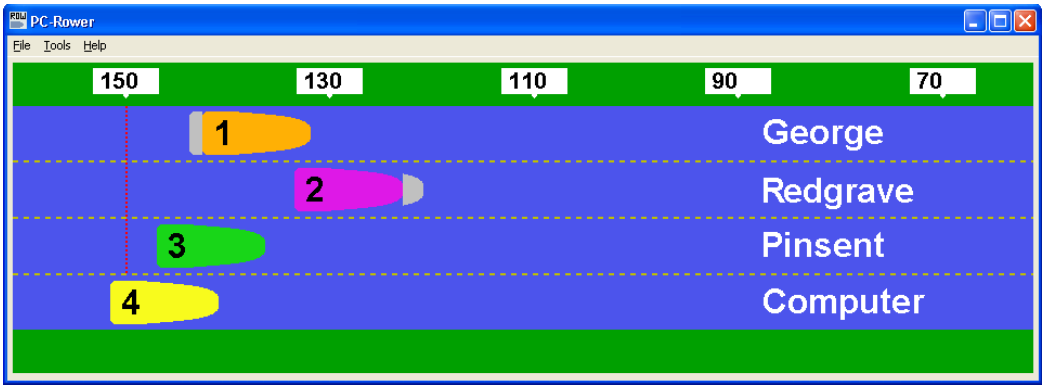

Figure 4.5: Advanced Deliverable Screenshot

4.3 Software Development Practices

4.3.1 Development Model

It was stated in Section 3.1.1 that an incremental model would be used in the development of the system. The primary reason for choosing such a model was its high suitability for working with clients by advocating development as a series of discrete levels of functionality. In the implementation phase this theory was found to hold true on several occasions when the client changed their requirements. The first such instance was in a software demonstration where the basic deliverables were presented. The client thought a substantial level of functionality had been achieved and, having

realised the potential of the system, decided on a few deliverable changes. Most notably, rather than have a graph system showing heart rate and training zones for the advanced deliverables, the client decided it would be more beneficial to provide pace boats as shadows and allow several rowers to race each other. Due to the flexibility of the incremental model, this change could be easily adopted with little disruption to the implementation. This further iterated the strength of the incremental model as a waterfall model would have necessitated the need to go back to the requirements stage and follow the process through.

The second instance where the incremental model showed its strength was when Durham University Boat Club changed their five kilometre ergo test date. The tests were originally planned for the end of Epiphany term and it was therefore logical to complete the implementation by this point. The ergo tests could then be used as a basis for the software evaluation. However when it was decided by the boat club to move the ergo tests to half way through the Epiphany term, it meant the likelihood of the software being finished on time was uncertain. It was because of the incremental approach that certain deliverables, for example 1.4.10, could be delayed as they would not be required for the tests. This meant more effort could be directed to the deliverables that would be required and thus they were achievable by the test date. This helped to ensure a good working relationship with the client, as a version of the software with the required functionality was available for the tests.

4.3.2 Development Tools

Eclipse (*http://www.eclipse.org*) is an open source, extensible, integrated development environment produced by IBM. It was chosen for development above other environments as it offers full support for SWT development (indeed is built using SWT, see Figure 4.6), is produced by a reputable organisation, has good forum support and has an excellent Java development kit offering among other features automatic syntactic correction. Furthermore the extensible environment means that plugins can be installed encouraging all areas of development to be handled under one environment. This increases the efficiency of development through less program switching and subsequent refreshing and, as all plugins utilise the same GUI model, it decreases the learning curve associated with a new product. Plugins used within this project include UML, XML, CVS, GUI development, JUnit and code auditing and metrics. As the eclipse project is open source the vast majority of plugins are available free of charge, meaning commercial standalone products can be avoided if required (although it should be noted

that most of these now offer an eclipse plugin version).

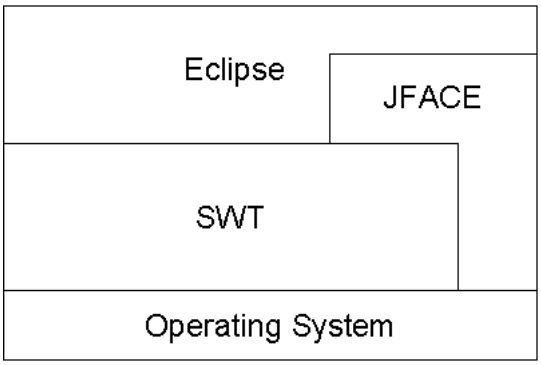

Figure 4.6: The eclipse framework

Overall, it was felt that the use of eclipse significantly reduced development time and it was therefore a sound choice for the integrated development environment due to efficiency gains.

4.3.3 Configuration Management

It is important to adopt a configuration management policy when developing code so that any changes can be reverted if necessary. Eclipse has an inbuilt history that records all changes and allows comparison with, as well as retrieval of, previous code. Given that the history only lasts for a limited period and that there is a risk of a hardware failure, an external configuration management system must still be adopted. Therefore a free CVS host (*http://cvsdude.kicks-ass.org/*) was used to upload code to on a daily basis. CVS, like all other tools used in this project, is available as an eclipse plugin ensuring that the tool can be used efficiently within a development environment.

Throughout the project there were several occasions where code changes needed to be made beyond the scope of eclipse's history and CVS was used in these circumstances. It was therefore extremely valuable to use a configuration management policy within the development of this system.

4.3.4 Testing

Unit Testing

Unit tests were used throughout development under the extreme programming ethos of ”write the test before the code” [2]. Under most circumstances JUnit was used to perform the tests, but in the case of the GUI Output module, the tests had to be performed manually. This involved taking a screenshot of the system and checking that both the boats and distance signs were in the correct position.

When refactoring the system, unit testing was used extensively to ensure that code functionality remained constant and the quality of code remained high. The complete set of tests were also run at regular intervals to ensure any modifications to code within development did not lead to unforeseen consequences. Extensive unit testing was therefore extremely useful in improving code quality and the ultimate reliability of the final system.

Integration Testing

A bottom-up incremental test approach was used as it advocates an increasing amount of modules to be tested, with the GUI added last. Stub classes were created to provide the inputs for the non-integrated modules and these were referred to if a bug was traced to an interface issue. Overall the incremental approach was very effective at identifying weaknesses between the module interfaces and consequently a few changes were made to increase the system reliability.

System Testing

System testing for the module focused on the performance of the system. A full account of the performance factors considered can be found in Section 4.1.3.

Acceptance Testing

A series of acceptance tests were performed after each requirement had been delivered in order to validate that the requirement had been completely implemented. Requirements 1.4.1–1.4.9 were required for the Durham University Boat Club five kilometre tests and therefore the use of the software at this event demonstrated the completeness of these deliverables (see Section 5.3.1). Additional tests were performed on deliverable 1.4.10 to show its completeness (see Section 5.3.3).

4.4 Chapter Summary

This chapter has identified the technical issues that were faced in the implementation. Broadly speaking these fell into four categories; PM2+ data retrieval, the river output, performance and usability. Requirement and architecture realisation was then shown by splitting the system down into basic, intermediate and advanced levels of functionality. The chapter finished with a view of the software development practices used. In particular the development model, development tools, configuration management and testing were discussed.

Chapter 5

Results and Evaluation

Chapter 5 presents the strategy and method used to perform the software evaluation as part of the Durham University Boat Club five kilometre ergo tests. The results are discussed and analysed with specific reference to the original project objectives.

5.1 Strategy

Durham University Boat Club conducted a mens squad five kilometre ergo test on the 2nd March 2004. The ergo tests are used to measure the progress made by the rowers over the previous two months of training and, in this instance, to aid the selection of the boat crews before the summer regattas. It is therefore critical that squad members perform to their maximum potential. By using the software it was desired to help the rowers perform strongly and, additionally, to carry out evaluation based on their feedback.

Before the ergo test date the system was set up and some final tests performed. It was during this period that two previously unconsidered factors became apparent. First, the screen saver and power saving functions of the computer should be disabled. Otherwise it is likely they will activate whilst the rowers are completing a workout, meaning the software can no longer fulfil its purpose. Second, the rowing machines location had to be decided upon, so that the projection could be clearly seen without dimming the lighting (which will restrict the rowers view of the PM2+). A room off the side of the main gym was chosen, as it offered controllable lighting and had little natural light. A

diagram of this setup is shown in Figure 5.1.

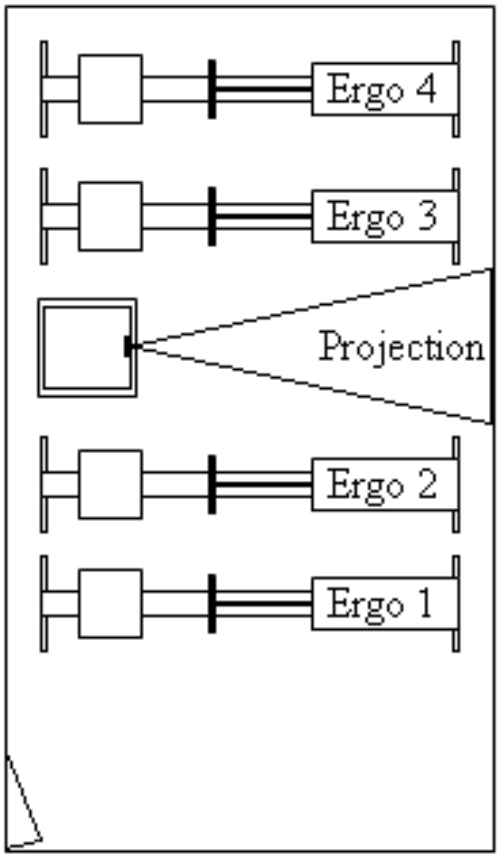

Figure 5.1: Ergo test setup

On the day of the tests the software worked without fault. This success can be attributed to the extensive testing policy used throughout implementation, that ensured any bugs were detected at development time and not in the ergo tests. Several weeks after the tests however, an issue was found where one of the batteries in the PM2+ units was low. It was discovered that should the voltage drop below a certain amount, then the PM2+ will continue to display its data to the rower, but cease to communicate with the computer. This happens in order to preserve battery life. The PM2+ does not display a low battery warning until the battery voltage is too low to power its own screen. This two-tiered approach to the battery life was not known at development time. A solution was subsequently found, however, as the PM2+ can be queried before starting a workout, to see if the battery is too low to communicate with the computer.

5.2 Method

The evaluation of the software fell into three distinct categories; software effectiveness, performance and platform independence. The first, software effectiveness, was measured by a questionnaire given to the rowers after performing their ergo test. The functionality of the software used within the ergo tests and the subsequent questionnaire covered deliverables 1.4.1–1.4.8. The performance and platform independence tests were conducted at a later date and covered deliverables 1.4.9 and 1.4.10.

The evaluation questionnaire, which is included in the appendix Figure A.4, was created as a means of evaluating the software. Questionnaires offer the clear benefits of evaluator anonymity, the ability to put quantitative figures on a subjective view and are cheap and easy to conduct. As it was the effectiveness of the software that needed to be evaluated and little experience in formal evaluations was required, using a questionnaire was an ideal evaluation technique.

The questionnaire begins with a description of the software and assures the evaluator that their responses will remain anonymous. The rationale of such a statement is to remind the evaluator of the program functionality and to encourage them to communicate their genuine opinions about the software. Where possible questions have a scaled answer from one to five, ensuring evaluators have a suitable range over which they can rate the software. In circumstances where it was not possible to provide a scale, a qualification question was added so the evaluator could reason their choice. The end of the questionnaire provides a space for any other comments or suggestions the evaluator may wish to add. The rationale for this statement is to provide a method that collects both areas missed by the questionnaire that the evaluator thought were important, and also to find possible ideas for further work.

The performance tests considered the performance of the software with respect to validating deliverable 1.4.9. They consisted of testing a Java version of the software on both a SUN JVM and IBM JVM under Windows XP, testing a Java version on a SUN JVM under Linux, testing a native version under Windows XP and testing e-row. The tests were run on the same computer each time and the software was configured to retrieve all PM2+ data. That is, the quick retrieve method where only the time and distance are queried was not used. Performance analysis was achieved by adding code that writes a log of system times at strategic points in execution. A serial port analyser (*http://www.hhdsoftware.com/sermon.html*) was used to analyse the times of serial port activity, and therefore calculate the cycle time for e-row. It was hoped the results

would provide ample data to compare the performance of the software under different environments and thus provide the information required to choose the optimum way in which to distribute the program.

The platform independence test was concerned with validating deliverable 1.4.10. There have been many requests in the Concept forums for rowing software that can work on Linux and Apple Macs platforms [7] and it is desirable to fulfil such requests. The system utilises the SWT GUI framework and as such depends on a SWT jar that varies between platforms. It is therefore necessary to download the correct SWT jar from IBM. Once this was obtained, a copy of the software was tried under a Linux operating system and the acceptance tests run to prove its functionality.

5.3 Results

5.3.1 Software effectiveness

The software effectiveness was assessed by a questionnaire. The results in raw data form are shown in the appendix, Figures A.5–A.13.

The first question was used to find the experience of the evaluator. Rowers of less than a year were considered a beginner and those of more than a year considered experienced. This data was then used in the subsequent questions to analyse the differences between evaluator experience.

Software Usefulness

The second question was used to evaluate the general usefulness of the software. Prior to the evaluation it was thought that the experienced rowers would not want to deviate from the PM2 due to the familiarity built over a period of years, whereas the beginners would be more flexible due to inexperience. However as Figure 5.2 establishes the opposite was the case with experienced rowers preferring the software. It is most likely that this difference is due to beginners lack of experience in not wanting to deviate from a recently familiar PM2. The experienced rowers will be less concerned with this due to previous ergo test experience and as their feedback suggests, they find the software

a useful mechanism to deviate from the boredom of using the PM2 all the time.

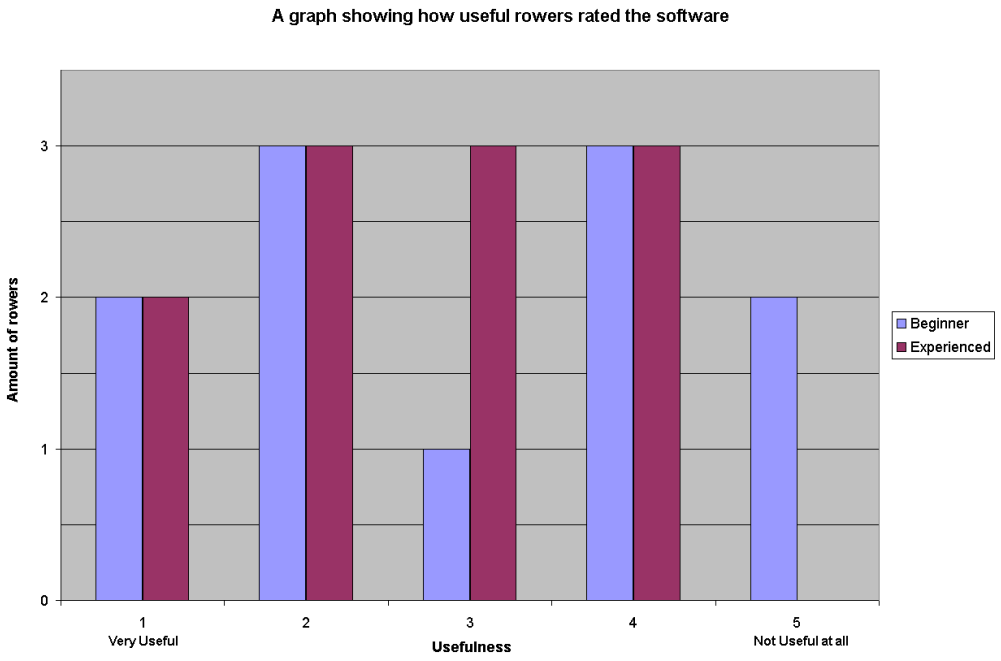

Figure 5.2: A graph showing usefulness of the software

The average usefulness of the software was calculated by averaging the sum of the value rated by each evaluator for the software's usefulness. In the case of the experienced rowers the value was 2.3 whilst for beginners the value was just 3. The overall average for all rowers was 2.8, making the software nearer neutral than useful. It was discovered in informal conversations after the ergo tests however, that the rowers who had previously used the software preferred it as they could prepare mentally for its use. Feedback received by rowers who had not used the system reinforced this need for mental preparation by indicating they would have liked the opportunity to evaluate the software in training first. Ergo tests are physically strenuous and mental preparation is therefore an important factor in performing well. Last minute alterations are not recommended [15]. Such an ethos justifies the feedback received and care should therefore be taken to ensure that the rowers are familiar with the software before being subjected to its use.

Rowers use the 500m split to monitor their speed throughout the workout and it is vital, therefore, that this value is visible at all times. Rower feedback highlighted that this was not available on the software and it was suggested that adding the value to the end of the rowers lane would increase the usefulness of the software. It is envisaged that if the workout time is also added then the software would be even more useful as there would be little need to refer to the PM2+.

Performance Enhancement

The third question focused on whether the rowers thought the software was helping them to achieve a better performance. The response to the first part of the question was split with $\frac{2}{3}$ of experienced rowers believing it helped them achieve a better time, but only $\frac{1}{3}$ of beginners believing so. This not only supports the results from the first part of the questionnaire, but extends the gap between the two types of rowers. It is clear, therefore, that experience is a significant factor in how useful a rower will find the software.

The evaluators response to the first part of the question was qualified in the second part, the results of which are shown in Figure 5.3. It can be seen that both beginner and experienced rowers prefer the graphical representation to the PM2 display. This would be logical as the software is a more natural way to visualise the workout, especially as rowers spend more time in rowing boats than on ergos.

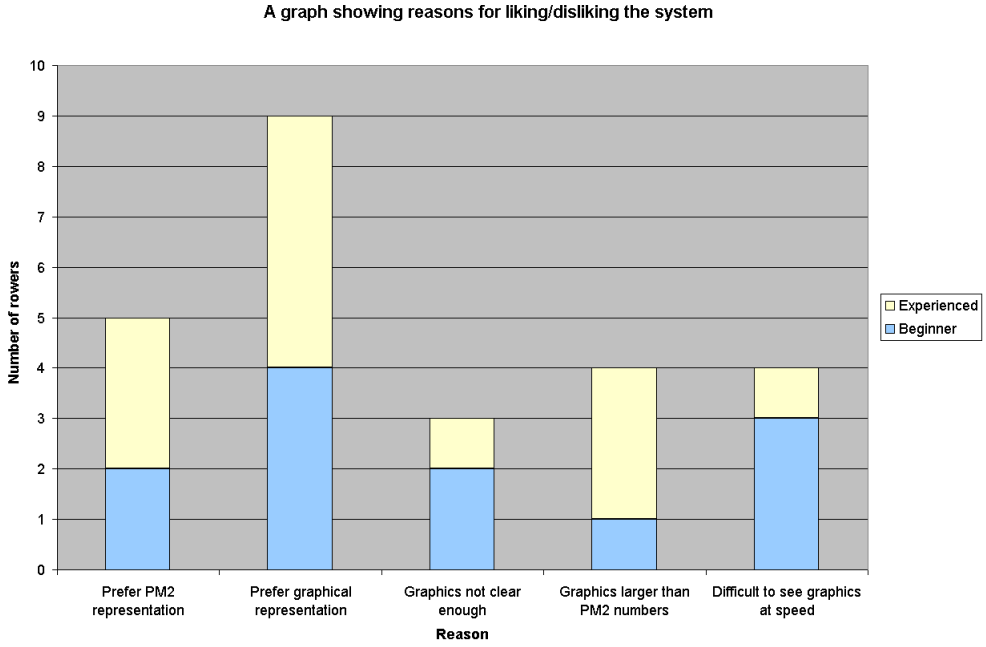

Figure 5.3: A graph showing reasons for liking/disliking the software

The graph indicates that some evaluators had difficulty seeing the projection. This was reinforced through feedback received by rowers from lanes one and four. These were the outside lanes and there was therefore, a greater distance to the projection. To resolve such issues one feedback suggestion involved using multiples screens and another, in an informal conversation after a test, suggested lining up the ergos in a two by two grid. It was further noticed that the colours were very different on the computer screen to the projection. On the projection the finish line was hardly visible and the yellow and

orange boats looked very similar. It is therefore reasoned that care should be taken over the setup of the ergos and projector to ensure all rowers have an equally good view.

One of the main purposes of the software was to provide direct motivation for the rowers, but it was also observed that the client was using the software when encouraging the rowers to perform well. Comparisons were made with the other boats in the workout, which is very similar to a real race where knowing the distance over a competitor offers a crucial mental advantage. In the ergo tests it was clear the client was trying to recreate such an effect. This further reiterates the previously stated point that the software provides a more relevant representation of the data for the rower.

Software use

The next question on the questionnaire focused on whether the rowers thought the software would be more useful in ergo training or tests. The first part identified that training was foreseen as the most beneficial use of the software, with little difference between experienced and beginner rowers. The second part of question focused on the rowers justification, the results of which are shown in Figure 5.4.

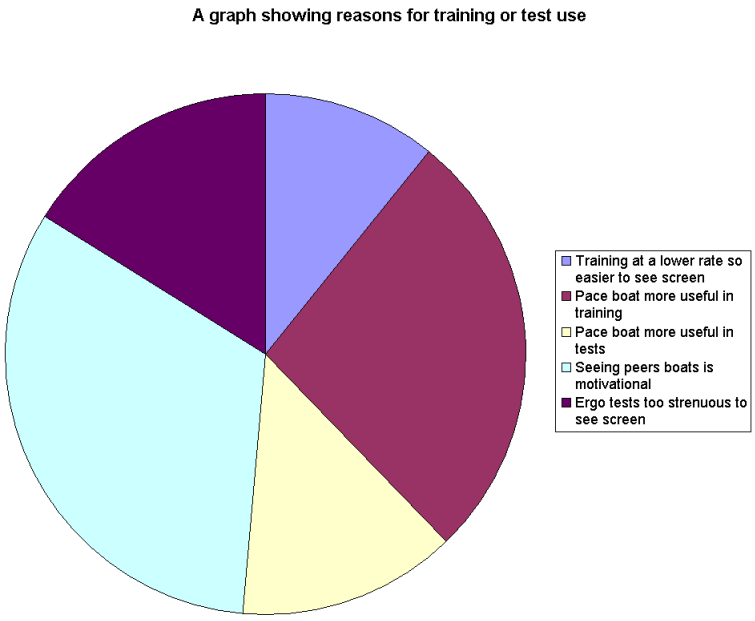

Figure 5.4: A pie chart showing reasons for training or test use

The chart shows that rowers believe seeing their peers boats is motivational. This relates well with the theory behind the development of the deliverable, which stated allowing several rowers to compete in the workout will be motivational to all involved. The chart

also indicates that rowers will find the pace boat more useful in training. This would be a logical result as ergo training involves rowing at a fixed pace and, as such, it would be beneficial to have a pace boat to follow. In ergo tests rowers are trying to perform at a maximum and so pace boats would be less useful as the rower is not following a fixed pace.

The fifth question was designed to see how useful the rowers would find the software in ergo tests if used regularly in training. By comparing the results of this question in Figure 5.5 with those in Figure 5.2 where the usefulness of the software in the ergo tests was analysed, it can be seen that the rowers clearly think the software will become more useful in tests when used regularly in training. This is most likely because the rower will become familiar with the software's representation of the data and more comfortable using it for important ergo tests. The average usefulness of the software in the ergo tests was found to be 2.8, whereas the perceived usefulness of the software when used regularly in training is 2.4. This reinforces the belief that the software can become more useful if used regularly in training.

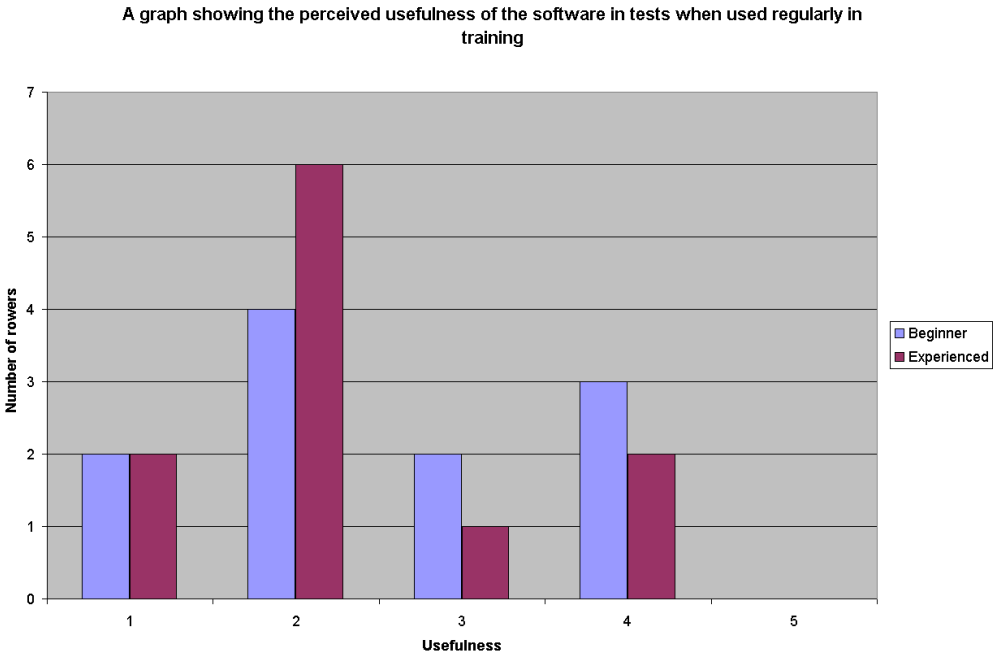

Figure 5.5: A graph showing the perceived usefulness of the software in ergo tests when used regularly in training

Spectator use

The final question focused on the use of the software for spectators. All but one of the evaluators believed it made the tests more exciting for those watching and it was further

noticed that the software induced more people into watching and shouting encouragement. It is believed this is because the software presents the workout visually, whereas it is hard to visualise the workout from the PM2's numerical data. More practically, it was remarked by one member of the coaching staff that the software can help to visualise how close the rowers are to each other, a figure that is hard to quickly calculate from the PM2's display. Therefore the software has been found to be both useful for spectators and coaching staff alike.

5.3.2 Performance

A monitor refresh rate of less than 60Hz results in an unsmooth movement that can be noticed by the human eye [16]. This is equivalent to a cycle time of 16.6ms which, on initial consideration, would be a good limit to determine whether the system is real time. However it must be remembered that the user of the software will be sat on a rowing machine, not directly in front of a monitor. Furthermore it is the time between the rower reaching a distance on the ergo and this distance being reflected on the GUI, that is most important. It was therefore decided that a time of 100ms or less would provide a sufficiently fast implementation for the software to be considered real time. Tests were performed to validate this and it was found that, even when the rowing machine was next to the computer, it was not possible to notice a time lag of 100ms.

Figure 5.6 shows the cycle time figures for different runtime setups of the software, as outlined in Section 5.2. The native version of the software not only outperformed the versions running on a JVM, but also outperformed e-row. In fact, e-row was the slowest of all configurations tested. This was very surprising as e-row was implemented in C++ and it was envisaged that this would be faster than a Java implementation. Although no details on the inner workings of e-row are available, it is used in national indoor rowing competitions and so presumably is considered sufficiently fast by Concept. It was decided that a cycle time of nearly a quarter of a second was too slow though and the original limit of 100ms was used to evaluate the system.

Time (in ms)	PC-Rower				e-row
	Win Sun JVM	Linux Sun JVM	Win IBM JVM	Native	Native
PM2+ query	60	60	40	40	100
GUI update	90	60	30	5	125
Total	150	120	70	45	225

Figure 5.6: Runtime Performance Comparison Table

It can be seen that the total PM2+ query time does not drop below 40ms, regardless of the runtime setup. In development it was originally thought this was a consequence of a Java threading issue but was later realised, with the help of the serial analyser, that this is the total time taken for the PM2+ to respond. That is, there are four queries with the PM2+ taking around 10ms to respond to each. This 10ms is the time associated with the data travelling down the wire, the PM2+ responding to the query, and the data travelling back. As such there is nothing that can be done to speed up the query process. It is therefore, as Figure 5.6 records, in the updating of the GUI where significant performance gains are realised.

From these results it can be stated that the system is real time when running on an IBM JVM or when compiled to native code. This is because the cycle time is less than the required 100ms. Although not tested on an IBM JVM running under Linux, it is envisaged that this would also be sufficiently fast. As these three setups are real time, they are all suitable options for deployment should this be required.

5.3.3 Platform Independence

When the system was run in a Linux environment a few problems were encountered. First, the location of the image files had been stored in a Windows format. Whilst the software used ”images\boat.gif”, Linux required ”images/boat.gif”. This was an oversight in development and the Java class `File` was used to encapsulate the image locations in a platform independent form. Once this issue was resolved, no further instances of windows specific file locations were found within the code.

Second, the reference used to access a serial port in Windows is different to that in Linux. Whereas Windows uses ”COM1” and ”COM2”, Linux uses ”/dev/ttyS0” and ”/dev/ttyS1”. This was overlooked in the software development and an operating system detect command was therefore used to populate the correct serial port references in the options dialog. This is a program option where the user chooses the serial port that the PM2+ is connected to. The alteration corrected the issue so that the correct port could be selected regardless of the operating system.

Once these two corrections were made the program was successfully loaded. The acceptance tests were then run on the software to prove its functionality. No further issues were found and it can therefore be concluded that the software is platform independent.

5.3.4 Relation to project objectives

The deliverables 1.4.1–1.4.8 were shown to be complete by the successful use of the software in the ergo tests. This can be stated as the functionality provided by these deliverables was required for the tests. Had the functionality not been present, the software could not have been used. The fulfilment of these deliverables further shows that the project objectives 1.3.1–1.3.3 have been achieved. The software was shown to operate in real time in Section 5.3.2 and to be platform independent in Section 5.3.3. As such it can be said that deliverables 1.4.9 and 1.4.10 have also been fulfilled.

The results from the questionnaire and the rowers feedback establish that project objective 1.3.4 has been fulfilled. That is, rowers have found the software helps increase performance capability. Additionally project objective 1.3.5 stated the project should make a strong case for an increasing use of computer technology in sport. As the evaluation of this project has shown, rowers find the software helps to increase performance. It is logical therefore to say that computer technology should be used more in sport, if the same effects can be realised. Objective 1.3.6 was concerned with the good use of software engineering principles. These were followed throughout the project and a thorough account can be found in Sections 3.1 and 4.3.

5.4 Chapter Summary

This chapter has outlined the strategy and method used to evaluate the system. It was found that in order to maximise the potential of the software, the ergos must be setup carefully so that all rowers have a good view of the projection. Rowers who have seen the software before the tests preferred it and those rowers who have been rowing for a year or more, find the system of more use than those who have not. More than half the rowers felt it would be of more use in a training environment than that of ergo testing and the software was further found to be useful for coaching staff who can better visualise the distances between rowers. Overall it can be said that the software met the original project objectives by, most notably, helping rowers to achieve a better performance.

Chapter 6

Conclusion and Further Work

Chapter 6 summaries the progress made throughout the project with specific reference to the project objectives and deliverables. The project management approach is reviewed and recommendations for further work are made.

6.1 Summary of work

6.1.1 The software

Through the project brief (appendix Figure A.3) and background research in the rowing domain, a comprehensive understanding of the project scope was developed. This knowledge was employed to construct a series of project objectives and deliverables, against which the success of the project can then be measured. A thorough literature review was subsequently conducted in which the current developments in the fields of software engineering and computers in sport were presented. The research highlighted the successes and difficulties faced by past use of technology in sport. The need for a successful client relationship was demonstrated and an overview of potential development models presented. An incremental development model was chosen as it offered the ability to implement the functionality in stages, whilst still allowing the required flexibility for working with a client. This proved to be most useful, as the client changed their requirements on several occasions.

A centralised architecture was chosen for the system as it dictates a real time structure. The composing modules were designed to be both conceptually simple, and to permit implementation in an incremental fashion without distorting the architecture. This ensured that any work in the early deliverables was not made obsolete by the requirements of more advanced ones. After a series of significant design and subsequent implementation decisions, the system was completed within the planned time period. The software was then tested in the Durham University Boat Club five kilometre ergo tests and its usefulness evaluated by the participants.

6.1.2 Results of evaluation

The usefulness of the system was measured through a questionnaire given to rowers after the ergo tests. It was found that, in all areas of evaluation, the rowers with over a years experience preferred the software to the traditional PM2 display. As an explanation for such a result, it was proposed that experienced rowers do not mind deviating from the PM2 display, whereas beginner rowers are more concerned with monitoring their 500m split. This was reiterated through rower feedback that requested the 500m split at the end of each rower's lane. It was further found that rowers who had experienced the software prior to the ergo tests found it more useful. This was a logical conclusion as they could mentally prepare for its use. The rowers who were not aware of the software, and therefore had not had chance to prepare for its use, would clearly not want to alter their race plan at the last minute.

Although the rowers found the software to be useful in the ergo tests, over half believed it would be even more use in training. This result was rationalised as training often involves rowing at a fixed 500m split for which pace boats are much more useful. The software was additionally found to be useful for coaching staff who can better visualise the distances between rowers. This can help them in monitoring progress and calling relevant instructions or increasing motivation. On the whole, the use of the software was very successful in helping the rowers to increase their performance. However it was found that the setup of the ergos inhibited the rowers furthest from the screen from using the software to its maximum effect. This was due to difficulties in seeing the projection clearly. Several solutions were suggested through rower feedback including individual screens, a bigger projection and setting the ergos up in a two by two grid.

After the ergo tests a performance and platform independent test were conducted. The first test looked at the performance of the software over several different deployment

options. It was found that the software performed best when compiled to a native executable, which resulted in a cycle time of 40ms. This was much faster than the required 100ms to be considered real time. The code was further found to be sufficiently fast when running on an IBM JVM, which resulted in a cycle time of 70ms. The platform independence test was run under a Linux operating system and after a few initial issues were resolved, the software passed all acceptance tests. It was therefore concluded that the software was both real time and platform independent.

6.1.3 Progress against project objectives and deliverables

The project deliverables concern the functionality that should be provided by the system. These were split into three levels representing the approximate difficulty of implementing the deliverable. Once the software was complete, acceptance tests were run to show the deliverables fulfilment and the results showed that all the deliverables were fully implemented. This was further iterated by the ergo tests that required deliverables 1.4.1–1.4.8, the performance test that required deliverable 1.4.9 and the platform independent test that required deliverable 1.4.10.

The project objectives are a higher-level achievement than the project deliverables. By fulfilling all the deliverables, project objectives 1.3.1–1.3.3 were consequently achieved. Project objective 1.3.4 focuses on rowers achieving a better performance as a result of the software. As argued in Section 5.3.4, this has been achieved, as half the rowers believed the software helped them to achieve a better performance. This result also implies that project objective 1.3.5 has also been fulfilled. That is, this project has shown that computer technology in sport can aid performance, and thus its use should be promoted if such results can be realised in other areas. The final objective, 1.3.6, concerns the use of good software engineering practices and as such these were followed throughout the project to ensure its success. A full account can be found in Sections 3.1 and 4.3.

6.2 Project Management

In Section 1.5 a project plan was formulated that was used regularly throughout the project to assess progress. In situations where progress was behind target, effort was

made to get back on schedule. This was important or the ultimate project deadline could have been missed. At the start of the Epiphany term, a revision was made to the project plan to allow for progress and problems encountered in the previous term. Once this replanning had occurred, the project was completed successfully according to the revised plan. Overall, the project plan provided a means to asses progress and was therefore very useful in the planning of the project.

The success of the project can be partly attributed to the incremental development model. Some deliverables were extremely difficult to implement and it was only because of the flexibility offered by the model, that less important areas could be deferred until the difficulties were sorted. Furthermore, when the date of the ergo tests was changed, a version of the software with the required functionality could be provided. This success was attributed to the strength of the incremental model in allowing some requirements to be postponed.

The following of sound software engineering principles also helped to ensure the success of the project. The use of eclipse as an integrated development environment helped to decrease development time by use of its automatic syntactic correction. Test-driven development helped to increase code quality, whilst the use of configuration management ensured previous code versions were accessible throughout development. The reuse of serial and XML libraries further decreased development time and upheld code quality.

6.3 Recommendations for further work

From the user evaluation, a few areas of improvement were identified. First, the rower pace should be displayed alongside the rower name at the end of each lane. This will enable the rowers to see the 500m split of others participating in the workout. Second, the time elapsed for the workout should be displayed. Then, if both the pace and time are shown on the GUI, there is little need for the rower to look at the PM2+. Finally, a high contrast option should be provided that can be toggled for use with a projector. This will make the display much clearer and, hopefully therefore, increase the usability of the system when using a projector. These three improvements were the most requested in the rower feedback and it is hoped that implementing them will encourage the use of the software within training.

Additional functionality for the project was similarly identified from rower feedback. The use of graphs to show a performance characteristic, such as power, over a workout would be extremely useful. Such graphs could then be compared with previous performances by loading in an XML workout file. This would not only identify weaknesses at certain points within the workout, but also show progress over time. Furthermore it would be helpful to add training zones that, should the rower go above or below, will result in an audio-visual alert. This would be useful as much of rowing training is based around keeping a fixed 500m split or staying within a heart rate zone.

Several rowers requested intelligent pace boats that can alter their speed over the course of a workout, typically every 500m. As the rowers themselves often alter their pace at strategic points in the workout, it would be very beneficial to have a pace boat that can do the same. Should a fixed pace boat be used, then it is much harder for the rower to judge whether they are actually ahead or behind of their target if they are varying their pace. It is not desirable for the rower to be making such calculations whilst rowing and it can therefore be stated that the implementation of intelligent pace boats would increase the usefulness of the system.

At the end of each ergo test the rowers record their average pace for every 500m rowed (the PM2 stores this). It would be a great time saving mechanism to automate this process by retrieving the data and exporting it to Microsoft Excel. A more advanced version would offer user profiles within the system that record every workout conducted. This could then be used to compare workouts over time. Although more complex to implement, the user profiles would be more useful as it combines the entire rower's data within one program without the restrictions of a spreadsheet.

Chapter 7

References

[1] A.W. Ainger and F. Schmid, *The helical approach to software design*, Computer Integrated Manufacturing Systems **8** (1995), 105–115.

[2] K. Beck, *Extreme programming explained: Embracing change*, Addison-Wesley, 1999, ISBN: 0201616416.

[3] J. Best, *In a conversation about analysis of sport performance over time*, 2003.

[4] Terry Braun, *Concept PM2+ ergometer interface daemon*, 2002, *http://pm2erglib.sourceforge.net/pm2d.pdf* Last accessed: 16th April 2004.

[5] F.P. Brooks, *No silver bullet - essence and accidents of software engineering*, IEEE Computer **20** (1986), 10–19.

[6] C. Clavadetscher and B. Lawrence, *Head to head: User involvement key to success?*, IEEE Software **15** (1998), 30–33.

[7] Concept, *e-row forum*, 2004, *http://www.concept2.com/forums/erow_forum_frames.htm* Last accessed: 6th March 2004.

[8] Concept, *Indoor rower information*, 2004, *http://www.concept2.co.uk/rower* Last accessed: 16th April 2004.

[9] Concept, *PM2+ specification*, 2004, *http://www.concept2.com/products/erow/pm2plusspec.asp* Last accessed: 16th April 2004.

[10] M.F. Cunningham, F.H. Kent, and D. Muir, *The cyber-olympics - schools, sports and the superhighway*, Computer Education **30** (1998), 61–65.

[11] I.M. Franks and G. Miller, *Training coaches to observe and remember*, Journal of Sports Sciences **9** (1991), 285–297.

[12] B. George and L.A. Williams, *A structured experiment of test-driven development*, Information and Software Technology (2003), To appear.

[13] R.L. Glass, I. Vessey, and V. Ramesh, *Research in software engineering: an analysis of the literature*, Information and Software Technology **44** (2002), 491–506.

[14] D. Greer and G. Ruhe, *Software release planning: an evolutionary and iterative approach*, Information and Software Technology (2003), To appear.

[15] O.W. Hall-Craggs, *In a conversation about race preparation for rowing performance*, 2004.

[16] N. Holliman, *In a conversation about the relevance of refresh rates to graphics animation*, 2003.

[17] M. Hubbard, *Computer simulation in sport and industry*, Journal of Biomechanics **26** (1993), 53–61.

[18] IBM, *eclipse forums*, 2004, *http://www.eclipse.org/newsgroups/index.html* Last accessed: 16th April 2004.

[19] J.M. Inesta, E. Izquierdo, and M.A. Sarti, *Software tools for using a personal computer as a timer device to assess human kinematic performance: a case study*, Computer Methods and Programs in Biomedicine **47** (1995), 257–265.

[20] M. Jackson, *Why software writing is difficult and will remain so*, Information Processing Letters **88** (2003), 13–25.

[21] D. Jitnah, J. Han, and P. Steele, *Software requirements engineering: An overview*, Peninsula Press (1995).

[22] J.M. Kamara and C.J.Anumba, *Clientpro: a prototype software for client requirements processing in construction*, Advances in Engineering Software **32** (2001), 141–158.

[23] S. Keates, P. Clarkson, and P. Robinson, *Developing a practical inclusive interface design approach*, Interacting with Computers **14** (2002), 271–299.

[24] A. Ko, M. Burnett, T. Green, K. Rothermel, and C. Cook, *Improving the design of visual programming language experiments using cognitive walkthroughs*, Journal of Visual Languages and Computing **13** (2002), 517–544.

[25] K. Lee, K.C. Kang, W. Chae, and B.W. Choi, *Feature-based approach to object-orientated engineering of applications for reuse,* Software - Practice and Experience **30** (2000), 1025–1046.

[26] A. Lees, *Computers in sport,* Applied Ergonomics **16** (1985), 3–10.

[27] D.J. MacFarlane, I.M. Edmond, and A. Walmsley, *Instrumentation of an ergometer to monitor the reliability of rowing performance,* Journal of Sports Sciences **15** (1997), 167–173.

[28] M. Mantei and T. Teorey, *Cost/benefit analysis for incorporating human factors in the software lifecycle,* Communications of the ACM **31** (1988), 428–439.

[29] G. McCluskey, *Thirty ways to improve the performance of your java programs,* Glen McCluskey & Associates LLC Article, 1999, *http://www.glenmccl.com/jperf/index.htm* Last accessed: 16th April 2004.

[30] J.F. Nichols, C.G. Morgan, L.E. Chabot, J.F. Sallis, and K.J. Calfas, *Assesment of physical activity with the computer science and applications, inc., acelerometer: Laboratory versus field validation,* Research Quarterly for Exercise and Sport **71** (2000), 36–43.

[31] C.R. Nigg, *Technology's influence on physical activity and exercise science: the present and the future,* Psychology of Sport and Exercise **4** (2003), 57–65.

[32] D.L. Parnas, *Structured programming: A minor part of software engineering,* Information Processing Letters **88** (2003), 53–58.

[33] D. Partridge and I.M. Franks, *Computer-aided analysis of sport performance: An example from soccer,* The Physical Educator **50** (1993), 208–215.

[34] J. Preece, Y. Rogers, and H. Sharp, *Interactive design - beyond human computer interaction,* John Wiley & Sons ltd., 2002, ISBN: 0471492787.

[35] K. Restivo, *Professional sports teams turn to resellers for an edge,* Computer Dealer News **15** (1999), 33.

[36] H. Saiedian and R. Dale, *Requirements engineering: making the connection between the software developer and customer,* Information and System Technology **42** (2000), 419–428.

[37] J. Smith and P. Crome, *Computer technology: friend or foe,* Sport Health **14** (1996), 21–22.

[38] R.M. Smith and C. Loschner, *Biomechanics feedback for rowing*, Journal of Sports Sciences **20** (2002), 783–791.

[39] I. Sommerville, *Software engineering (6th edition)*, Addison-Wesley, 2001, ISBN: 0749437391.

[40] D. Sosnoski, *Xml and java technologies: Data binding performance*, 2003, *http://www-106.ibm.com/developerworks/xml/library/x-databdopt2/* Last accessed: 16th April 2004.

[41] M. Stephens, *Emergent design vs. early prototyping*, 2003, *http://www.softwarereality.com/design/early_prototyping.jsp* Last accessed: 16th April 2004.

[42] B. Tisdall, *A sporting chance*, Personal Computer World **October** (1996), 116–120.

[43] B.Y. Tsai, S. Stobart, N. Parrington, and B. Thompson, *Iterative design and testing within the software development life cycle*, Software Quality Journal **6** (1997), 295–309.

[44] J.M. Verner and N. Cerpa, *Prototyping: Does your view of its advantages depend on your job?*, Journal of Systems and Software **36** (1997), 3–16.

[45] D. Zowghi and V. Gervasi, *On the interplay between consistency, completeness and correctness in requirements evolution*, Information and Software Technology **45** (2003), 993–1009.

Appendix A

The appendix contains additional information that did not fit naturally into the flow of the main text. It includes two UML class diagrams, a copy of the project brief, a copy of the evaluation questionnaire and the raw data from the user evaluation.

UML Class Diagrams

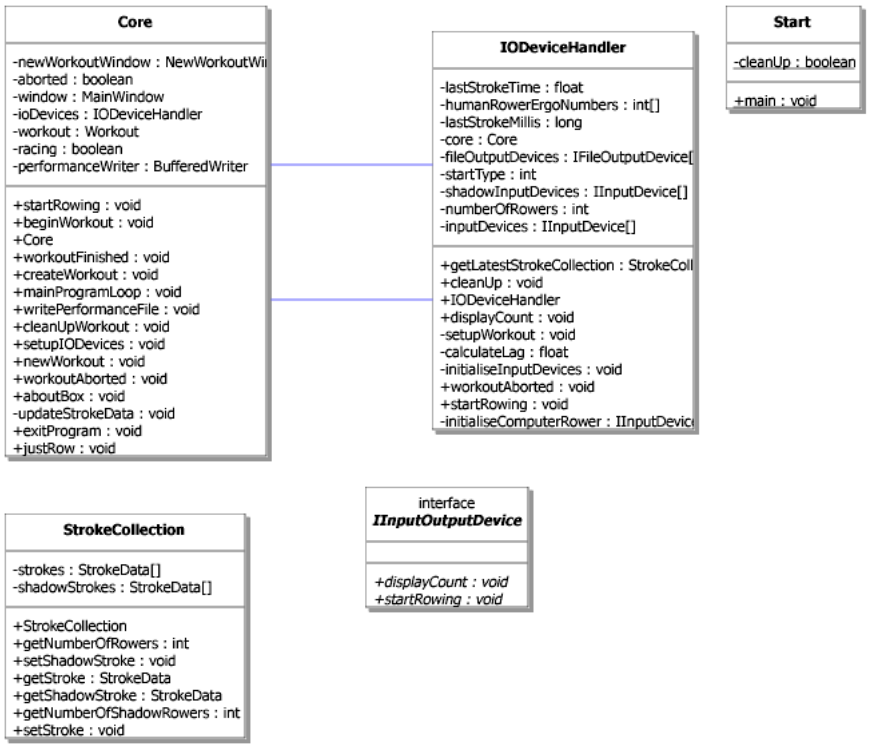

Figure A.1: Controller module class diagram

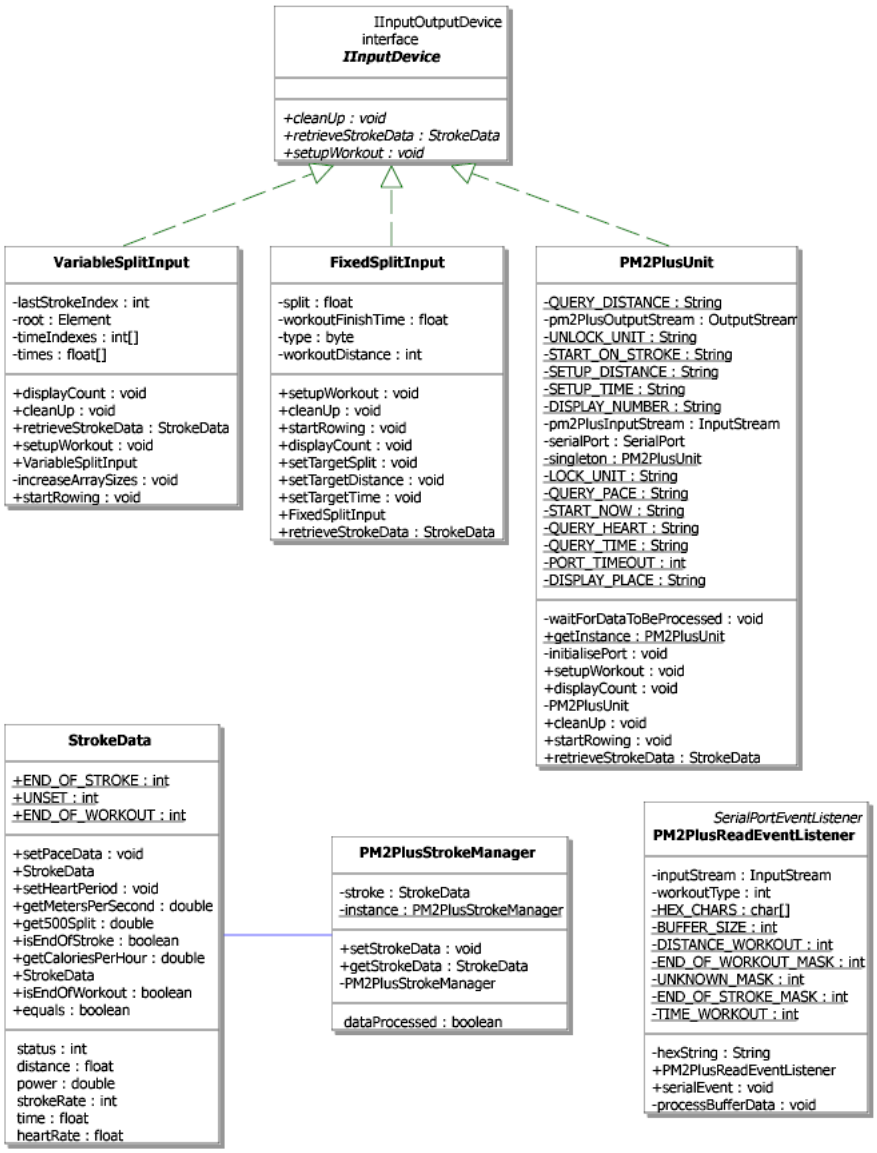

Figure A.2: Input module class diagram

Project Brief

The project brief is shown in Figure A.3.

Questionnaire

The evaluation questionnaire is shown in Figure A.4.

Training software for a Concept II rowing machine

Research Area:
Software Engineering

Hardware and Software Prerequisites:
A concept II rowing machine equipped with a PM2+ monitor and serial port cables. A computer with a serial port.

Description:
A concept II monitor displays data such as stroke power, distance travelled, average split and heart rate (assuming rower is wearing a heart rate chest strap). When using the machine for a workout only the averages of these values are saved for recall every 500m. It would be of great benefit see the rower's progress graphically and to store every stroke.

This project is concerned with capturing the data from the PM2+ monitor and displaying it into a form that would be more useful for training analysis.

Preliminary Preparation:
Familiarisation with published PM2+ interface.
Research into serial communications in the windows operating environment.
Research into graphics API's for the visual element.

Minimum Objectives:
Create a piece of software that captures the data from the PM2+ monitor and provides a simple GUI with a boat moving down a river.

Intermediate Objectives:
Support many users each having their workouts stored under a profile, with the option to race either a predefined or custom distance. Store the data captured as a workout and allow the retrieval of this as a pace boat to race against.

Advanced Objectives:
Support up to four rowers on one screen with an option to use a shadow as a pace boat. Ensure the system operates in real time to be of maximum benefit to the users.

Figure A.3: Project Brief

PC-Rower Evaluation

PC-Rower has been developed as part of a dissertation into computer technology within sport. The software displays boats moving down a river with distance signposts. It supports time or distance workouts with up to four human rowers and as many computer boats as required. Computer boats may be a fixed pace or use a previously saved user workout as the pace.

To evaluate the potential success of such software it would be appreciated if you could answer the following questions. Please ask if you're unsure about a question and use the following scale when required:

1 = very useful, 3 = neutral, 5 = not useful at all

How many months have you been rowing?

0-3 4-6 6-12 12-24 24-36 36+

How useful did you find the software when rowing in the ergo tests?

1 2 3 4 5

Do you think it helped you to achieve a better time?

Yes No

Why (underline all that apply)?

Prefer PM2 representation Prefer graphical representation Graphics not clear enough

Graphics larger than PM2 numbers Difficult to see the graphics at speed Other…

Do you think the software will be more useful in ergo training or tests?

Training Tests

Why (underline all that apply)?

Training can be at a lower rate so screen easier to see Pace boat more useful for training

Pace boat useful when performing at maximum Seeing peers boats is motivational

Ergo tests are too strenuous to see the screen Other…

How useful do you think the software would be in tests if used regularly in training?

1 2 3 4 5

As a spectator do you think the software made the tests more exciting?

Yes No

Any further comments, suggestions or ideas?

Thank-you for your time. All responses will be kept anonymous.

Figure A.4: Evaluation Questionnaire

Raw data

Experience (in months)	0-3	4-6	6-12	12-24	24-36	36+
Number of rowers	2	7	2	2	1	8

Figure A.5: Question 1 Result Table

Rower Experience	1	2	3	4	5
Beginner	2	3	1	3	2
Experienced	2	3	3	3	0

Figure A.6: Question 2 Result Table

Rower Experience	Yes	No
Beginner	4	7
Experienced	7	4

Figure A.7: Question 3a) Result Table

Rower Experience	PPR	PGR	GNC	GLN	DSS
Beginner	2	4	2	1	3
Experienced	3	5	1	3	1

Figure A.8: Question 3b) Result Table[1]

Rower Experience	Training	Tests
Beginner	6	5
Experienced	7	4

Figure A.9: Question 4a) Result Table

[1]PPR=Prefer PM2 representation, PGR=Prefer graphical representation, GNC=Graphics not clear enough, GLN=Graphics larger than PM2 numbers, DSS=Difficult to see the graphics at speed

Rower Experience	TLR	PBT	PBM	SPM	ETS
Beginner	3	5	4	5	5
Experienced	1	5	1	7	1

Figure A.10: Question 4b) Result Table[2]

Rower Experience	1	2	3	4	5
Beginner	2	4	2	3	0
Experienced	2	6	1	2	0

Figure A.11: Question 5 Result Table

Rower Experience	Yes	No
Beginner	10	1
Experienced	11	0

Figure A.12: Question 6 Result Table

Comment	Occurences
You need to have seen the system before using it in the tests	2
Would like to see a pace value at the end of each lane	2
Would like to see the time elapsed on the screen	1
It helps to keep concentration in the middle stages of the race by providing an alternative view of the PM2 data	1
I think it helped me to improve by 2 seconds	1
Thank you	1
Pace boats are useful but need to be clearer	1
Good for seeing progress so far	1
Hard to see screen from my ergo (lane 1)	1
Need a bigger or individual screen (lane 4)	1

Figure A.13: Question 7 Result Table

[2]TLR=Training can at a lower rate so screen easier to see, PBT=Pace boat more useful for training, PBM=Pace boat useful when performing at maximum, SPM=Seeing peers boats is motivational, ETS=Ergo tests are too strenuous to see the screen