Research

B.Kendrick, A.Syed

Contents

[Introduction 4](#_Toc412625032)

[An Integrated Development Environment 4](#_Toc412625033)

[NetBeans 4](#_Toc412625034)

[NetBeans for SWING (Modules) 5](#_Toc412625035)

[Plugin Support 5](#_Toc412625036)

[NetBeans-BlueJ 5](#_Toc412625037)

[Advantages 6](#_Toc412625038)

[Limitations 6](#_Toc412625039)

[Eclipse 6](#_Toc412625040)

[Eclipse SDK/ Eclipse Project 6](#_Toc412625041)

[E4 7](#_Toc412625042)

[Advantages 8](#_Toc412625043)

[Limitations 8](#_Toc412625044)

[BlueJ 9](#_Toc412625045)

[Interaction 9](#_Toc412625046)

[Visualisation 10](#_Toc412625047)

[Simplicity 10](#_Toc412625048)

[Advantages 10](#_Toc412625049)

[Limitations 11](#_Toc412625050)

[The “Useful” Features of BlueJ 11](#_Toc412625051)

[Object Bench 11](#_Toc412625052)

[Class Diagram 12](#_Toc412625053)

[Object Inspector 12](#_Toc412625054)

[Scope Highlighting 13](#_Toc412625055)

[Code Templates 14](#_Toc412625056)

[Test Class Generation 14](#_Toc412625057)

[Other Notable IDEs 15](#_Toc412625058)

[JDeveloper 15](#_Toc412625059)

[IntelliJ IDEA 15](#_Toc412625060)

[Eclipse Plugins 15](#_Toc412625061)

[Plug-in Development Environment (PDE) 15](#_Toc412625062)

[Main Components 15](#_Toc412625063)

[Extra Components 16](#_Toc412625064)

[Extendability 16](#_Toc412625065)

[Marketplace 16](#_Toc412625066)

[Examples 17](#_Toc412625067)

[EBob 17](#_Toc412625068)

[Ambient 17](#_Toc412625069)

[Teaching Programming 18](#_Toc412625070)

[Language Selection 18](#_Toc412625071)

[A Basic Process To Learning Programming 19](#_Toc412625072)

[Syntax and Semantics 19](#_Toc412625073)

[Combinations 19](#_Toc412625074)

[Problem Solving Skills 19](#_Toc412625075)

[Procedural and Object-Oriented 19](#_Toc412625076)

[Other Considerations 20](#_Toc412625077)

[Original Projects and the Rise of the Internet^ 20](#_Toc412625078)

[Information Availability 21](#_Toc412625079)

[Teaching Object-Oriented Programming 22](#_Toc412625080)

[Shortcomings of Tradition Systems Environments 22](#_Toc412625081)

[BlueJ Approach 22](#_Toc412625082)

[Transition 23](#_Toc412625083)

[Designing Software For Teaching/Learning 25](#_Toc412625084)

[Ease of Use 25](#_Toc412625085)

[User Interface 25](#_Toc412625086)

[Error Handling 25](#_Toc412625087)

[Documentation 26](#_Toc412625088)

[Methodology 26](#_Toc412625089)

[Conclusions 31](#_Toc412625090)

[IDE Selection 31](#_Toc412625091)

[Modular Development 31](#_Toc412625092)

[Teaching Programming 31](#_Toc412625093)

[Our Methodology 32](#_Toc412625094)

[Final Thoughts 33](#_Toc412625095)

[Future Research 33](#_Toc412625096)

[References 34](#_Toc412625097)

# Introduction

Developing a plugin for the Eclipse IDE with the aim of being beneficial to learning presents a set of challenges that will need to be addressed. We will have to develop a product that is simple, intuitive and sensible. In this section we look at the framework in detail and attempt to identify the areas in which we need to be able to manipulate to our project goals. Plugins will be researched and compared in order to determine whether there is an existing product that could meet some or all of our requirements and compare their functions in regards to the suggested functions of our end product.

We will attempt to look at existing solutions for creating software for learning and attempt to determine the most common elements that assist a potential student in learning and how to make software approachable and suitable for a learning environment. This will also require a study in how programming is taught and how our plugin can assist future educational efforts when teaching programming, alongside determining effective teaching methods and whether to somehow integrate these into our project. We will look at the methodology and delivery of education and how specific software has been utilised in a pedagogical manner.

# An Integrated Development Environment

The definition of an Integrated Development Environment (**IDE**) is an application that facilitates and assists in application development. These tools are usually based around a GUI that allows a developer an integrated environment with all required tools to hand. (techopedia.com, 2010)

The most common features of an IDE are debugging, version control and data structure browsing. An IDE provides a developer the means to quickly perform tasks without relying on several different applications thus maximising developer productivity.

Since the introduction of Java in 1995 there has been a desire to streamline and increase productivity of development on the platform (Geer, 2005). Developing an IDE for Java could theoretically make Java a more competitive language in the wake of the Microsoft created .NET framework which utilised Visual Studio. From this notion came the development of several IDEs for the Java language, these include Borland JBuilder, Microsoft Visual J#, Oracle’s JDeveloper and Sun’s NetBeans (Geer, 2005).

It is important that we look at the impact of IDEs and their impact on developing Java applications so that we can better understand the needs of developers and in turn be able to assess requirements for improving upon the Eclipse IDE through the development of plugins.

The majority of advantages and disadvantages of an IDE are largely down to user preference. A developer will tend to gravitate towards the IDE they are initially shown when learning programming; this is in part due to familiarity but often because the IDEs used in the majority of courses are selected because of their high level of stability and ease of use.

# NetBeans

NetBeans is the official IDE of Java sponsored by Oracle (NetBeans, 2013), NetBeans started as a student project designed to mimic the features of Delphi for Java. Xelfi, as it was previously known, was the first IDE for the Java platform and was initially intended to be a commercial product. It wasn’t until 2000 when NetBeans became open-source software and fully adopted a plugin-based system.

NetBeans is probably, alongside Eclipse, the most well-known and used Java IDE.

## NetBeans for SWING (Modules)

NetBeans is the only known IDE with a modular Swing application framework (Petri, 2010). This means that desktop application development is simplified by the addition of re-usable components and techniques and patterns to fully make the most out of the Swing platform.

NetBeans promotes modular application development; a module is a self-contained software component which can communicate with other modules through well-defined interfaces whilst hiding the implementations of each individual module (Petri, 2010). This approach theoretically reduces chaos in code and minimises the responsibilities of each individual class. In modular applications the implementation of features can only be used through public interfaces which means that dependencies become more well-defined and aren’t accidental.

Using a modular approach benefits developers because of a lack of chaos but can also benefit the end user and any beneficiaries to a project. Performance can increase as modules are flexible and dynamic meaning they can be loaded and unloaded as required, all at runtime. Features can be contained to a single module per feature which means that support can always be provided for a specific feature and any issues that arise can be easily fixed and maintained. Each module must be unique and specify its version; this is handled by NetBeans and ensures that modules can be easily maintained and that nothing conflicts without developer intentions. This uniqueness is handled by a manifest file, created using XML, which maintains the details of each individual module and directs the compiler on the compile order of each module.

Swing provides a user with a means to quickly develop professional looking applications with features that resemble the operating system that an end user is utilising. This gives the advantage of being cross-platform and highly beneficial when creating desktop applications, it also can assist in rapid prototyping. This does however have the disadvantage of making an application less novel and interesting, as though no thought was placed into an applications aesthetic; it must be stressed that whilst aesthetic is not important in the grand scheme of things it can be a contributing factor in an end users comfort when working within an application.

## Plugin Support

Being open source, NetBeans has been extended to allow for the use of plugins. These plugins enable the use of any kind of tool required by a developer to be added to a streamlined workflow. The major advantage to being able to add plugins to an IDE is extensibility and an increase in productivity.

### NetBeans-BlueJ

A collaboration between NetBeans and BlueJ existed to try and bridge the gap between a basic IDE and a fully professional IDE without the need to install two separate software packages (bluej.org, 2006). This project aimed to allow students the tools required in learning whilst enabling them to become more familiar with the NetBeans environment. This project was discontinued and at the time of writing there exists no alternative project to continue this work. The goals for the project were similar to our own goals and the reasons for its failure lie in a divide in approaches between the two separate teams and their difference in opinions on the usefulness of a tool that can assist in learning.

## Advantages

There are several advantages to the NetBeans IDE:

* **Extensibility – NetBeans** provides a good set of tools to develop plugins to extend its use.
* **Community Support –** Being the official IDE of Java there is an active community of developers working to improve and extend NetBeans.
* **Java Support –** NetBeans is the official IDE of the Java platform, this means that new Java features are integrated quickly and bugs are often fixed quicker than other IDEs.
* **Flexibility –** The IDE is highly flexible and can be used for a variety of different projects with different goals, especially due to its modular nature.
* **Debug Tools –** NetBeans provides a highly sophisticated set of debugging tools which are incredibly useful for the majority of developers, and some that are useful to the minority.
* **Cross IDE Compatibility –** NetBeans offers support to open projects created by other IDEs such as Eclipse.
* **One Fits All –** NetBeans servers well as a good tool for all projects out of the box without the need for installing and configuring plugins.

## Limitations

NetBeans does come with some limitations:

* **Platform Support –** NetBeans is not as widely supported across other platforms such as Eclipse and BlueJ.
* **Performance –** NetBeans is often considered to be slower and less efficient than the Eclipse IDE which can impact on user preference. One of the biggest hits to performance is that NetBeans often takes time to scan projects on initial start-up.

# Eclipse

Eclipse started life as an IBM developed tool based on technology from tools maker Object Technologies International (OTI). The Eclipse platform is seen as more than just an open source Java IDE but also a tool integration platform and a community driven platform of projects and ideas (Sam-Bodden, 2006). Eclipse was handed over to the Eclipse foundation in 2001; this meant a switch to an open-source distribution and development to become community based instead of proprietary.

According to Goth (2005) a number of surveys place Eclipse as the most widely used Java IDE, this is likely due to its highly extensible and customisable nature.

## Eclipse SDK/ Eclipse Project

At the core of Eclipse is the SDK which can be used to build a variety of products and tools around this primary source (eclipsepluginsite.com, 2010). All of these created products and tools can then be further extended in a potentially never ending cycle. The Eclipse architecture is easy to extent through the creation of plug-ins; everything within Eclipse is located within a plugin, with the exception of the kernel of the software.

Everything created with Eclipse will make use of the core Eclipse SDK, each product can extend tools and vice versa, Figure 1 - The Eclipse Core highlights this point in a simple to understand manner.

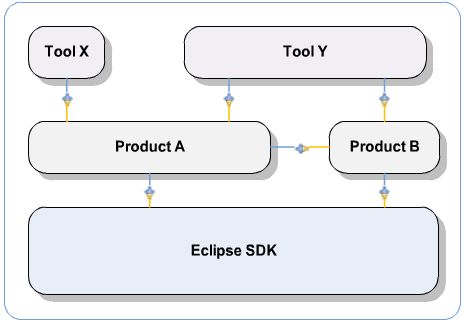


Figure 1 - The Eclipse Core

The Eclipse SDK is comprised of three main parts, each used to develop the Java IDE of Eclipse:

* **Plug-in Development Environment (PDE) -** The PDE provides the tools to create Eclipse plug-ins and extend existing functionality across the whole of the Eclipse IDE. The PDE tools are broken down three main areas: Build, UI and API, each are large topic on their own. These tools provide developers the means to extend the Eclipse framework as they see fit.
* **Java Development Tools (JDT) –** A sub-project of Eclipse that is used to develop tools for programming in Java. In the Eclipse SDK the base tools that are a part of the JDT are a Java compiler, a builder, basic editors wizards and content assistants plus all other features required for a good working IDE. The core of the JDT is the infrastructure of the IDE and should have no dependencies and run headless.
* **Rich Client Platform –** The Eclipse platform is designed as an open tools platform, however it is architected so that it can be used to build any client application. The minimum set of plug-ins required to build a rich client application are known as the Rich Client Platform, in Eclipse these are the UI and the core runtimes (and their dependencies). These are the tools from which the Eclipse IDE are built upon, the value of utilising the RCP in a project is that it allows a developer to quickly build a professional looking application. All aspects of the RCP are deemed to be of high quality and are actively maintained.

All of the aspects of the Eclipse SDK are open source and actively maintained by a dedicated community. The developers make liberal use of git repositories to share code and exchange ideas on the future of Eclipse.

## E4

E4 is incubator for community exploration of future technologies within the Eclipse project. This means that the community can use this area of the Eclipse IDE to attempt to develop new features that will ultimately become a part of the core Eclipse IDE.

The core principles of E4 are:

* **Explore technologies to include in the SDK.**
* **Simplify the Eclipse programming model.**
* **Enable the platform for use on web based runtimes.**
* **Broaden participation in the development of Eclipse.**

E4 is highly beneficial to the Eclipse Project as it ensures that there is always a forward goal, a drive to improve both the software and the capacity of the community it enables. Eclipse is fairly unique in having such a community driven development process, many new ideas for increasing productivity have come out of this initiative and will hopefully continue to do so for the foreseeable future.

## Advantages

Eclipse is the leading Java IDE for a reason; there are several key advantages to using Eclipse for Java projects:

* **Extensibility –** Eclipse can very easily be extended through the use of plug-ins. Plug-ins enable new functionality within the Eclipse IDE that allow developers freedom to use tools they are comfortable with and also highly increase productivity.
* **Community Support –** Eclipse has one of the largest active communities for an IDE, this means that support will always be available to developers who are using it.
* **Flexibility –** The IDE is highly flexible and can be used for a variety of different projects with different goals.
* **Platform Support –** Eclipse is supported on most major platforms meaning it can be portable as well as deployed efficiently in any working environment.
* **Task Management System**- A system that helps keep track of all of the views and files opened in a highly context sensitive manner.
* **RCP –** The rich client platform enables rapid and intuitive development within the Java environment.
* **Debug Tools –** Eclipse provides a highly sophisticated set of debugging tools which are incredibly useful for the majority of developers, and some that are useful to the minority. But also in Eclipse the debugger can be swapped with a plugin that enables deeper profiling alongside debugging.
* **Core Performance –** Eclipse is very quick to load, even on older systems.

## Limitations

There are a couple of limitations to the Eclipse platform but even these could be argued against as they are largely based on opinion:

* **Complexity –** Eclipse is not easy to use or learn, many new comers could be put off by the complexity of the software and the steep learning curve required to make full use of the IDE.
* **Not a miracle worker –** Like all IDE’s Eclipse will not fix or often even point out critical mistakes in a developers logic, a lot of what can be produced with Eclipse will ultimately always depend on a developers level of skill and proficiency with the target language (Java).
* **Bloated –** When using a lot of plugins the performance and storage overheads of Eclipse can start to take their toll, especially on older systems.
* **Missing Components –** Eclipse does not contain certain components “out of the box” and additional components have to be downloaded and installed, this can be time consuming and frustrating, especially when working on large team projects.

# BlueJ

BlueJ is an integrated development environment (IDE) designed specifically to introduce students to programming in an object oriented manner (Kölling, 2003). BlueJ is an extension of the Java SDK and as such makes use of the standard interpreter and compiler; this is advantageous because it means that the code written in BlueJ should be transferable to any other IDE. Contrary to most IDEs BlueJ presents a unique environment that is conducive to learning, BlueJ is built on simple concepts.

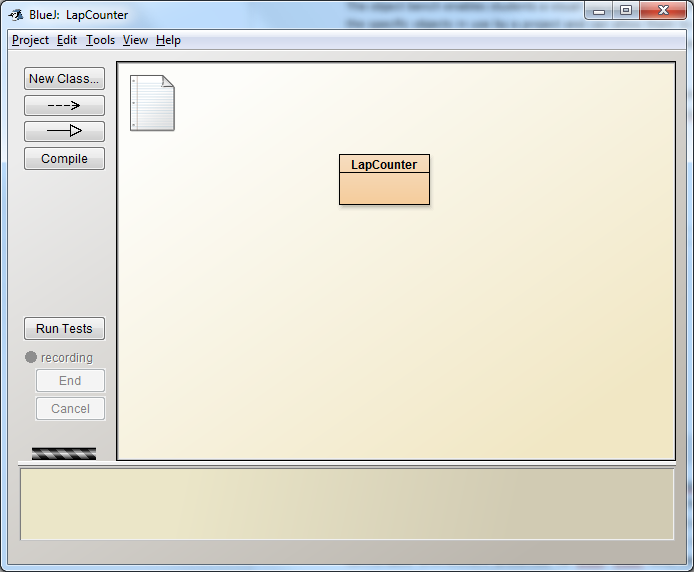


Figure 2 -The BlueJ interface

## Interaction

BlueJ provides the means to create objects of designed classes as required. This allows a user to see how an object is created, BlueJ provides the means to pass parameters to the objects constructor, this allows a user to be able to better understand the concepts of instantiation and allow them to understand how the *new* keyword works. Objects can also be created based on the classes provided in the standard Java library; this can prove highly beneficial to learn some of the standard Java classes and the effects of their methods.

Once the object is created it is placed on an object bench and can be manipulated as the user wishes. This manipulation often comes from calling the methods associated with an object; this can be achieved through a simple context menu accessed via a right click. Parameters can be passed to objects and the user can see the results of the method in relation to the selected object almost instantly. This is highly beneficial as it allows a user to better understand how methods directly affect the object they are associated with.

A user can also inspect an object using a built in inspector, this is useful as it allows a user to see the state of variables not accessible through a getter. Used in conjunction with the dynamic calling of methods an object can be tested fully to ensure complete functionality and the direct effect of a method call can be observed as required.

The interaction of an object can enable a very simple method of testing that eliminates the need for traditional case based testing. A user can test the functionality of an object instantly and can determine the usefulness of a developed class at the touch of a button.

## Visualisation

BlueJ is designed to be a highly visual IDE, the first thing a user sees when they launch the software is a UML based class diagram, this can be used to see the organisation of a project as well as understand the links between classes, this is useful in teaching topics such as inheritance, polymorphism and further a student’s understanding of class structures.

The object bench enables students a visual representation of instantiated classes, this provides an insight into the specific objects in use by a project and can allow them to see how they change and the effects method calls have on them. The object bench is useful in managing objects within a system and serves to illustrate the basics of creating objects and manipulating them.

By a combination of interaction and visualisation OO concepts can be presented in an easy to understand manner without the need for long and often dull explanations (Kölling, 2003).

## Simplicity

The entire BlueJ system is built upon the idea of simplicity (Kölling, 2003, Kölling et al.1999). Beginners require a different set of tools to professionals and will often benefit from having tools specialised to assisting their education in object oriented concepts. This simplicity doesn’t make BlueJ suitable for anything other than a learning environment; however there are many students who have issues ‘breaking away’ from the comfortable environment presented by BlueJ. BlueJ lacks many advanced features that can be found in traditional IDEs such as Eclipse but makes up for this, in an educational sense, by proving ease of use and good visualisation of concepts.

## Advantages

BlueJ has several advantages/ benefits that make it an effective tool for learning object-oriented programming:

* **Simplicity** – BlueJ is easy to use, new students can make use of its features with little tuition.
* **Visualisation –** The ability to see classes and their respective objects is highly beneficial to understanding some of the concepts behind OOP.
* **Interactivity –** BlueJ is highly interactive which provides its users with a ‘hands-on’ approach to learning the concepts as required.
* **Lightweight –** The IDE is highly lightweight and requires a minimum of resources to be utilised.
* **Easy to install –** BlueJ installs all of the required libraries and SDKs so the user can get straight into programming.

## Limitations

Whilst BlueJ is highly efficient as a learning tool it does face some issues and limitations:

* **Simplicity –** Whilst simplicity makes the tool easy to learn it also inhibits the education of some advanced topics and obscures the need for manual organisation of projects.
* **Power –** BlueJ lacks many of the features that other IDEs have that make BlueJ lacking in a professional environment.
* **Transition –** Many students have issues breaking away from the BlueJ environment and ‘graduating’ to a more fully fledged IDE.

## The “Useful” Features of BlueJ

BlueJ has a number of useful features that are key to the usefulness of the tool. These have been touched on above in the interactive area of BlueJ topic, here we will explore the useful features in more detail and look at the potential benefits of each feature for a variety of end users.

### Object Bench

The object bench allows for interaction with instances of user classes. These instances can be used to test methods and inspect the fields within. Instances on the object bench can reference each other and be utilised to perform a variety of local tasks, all without the need to build a full program with a “main” method.

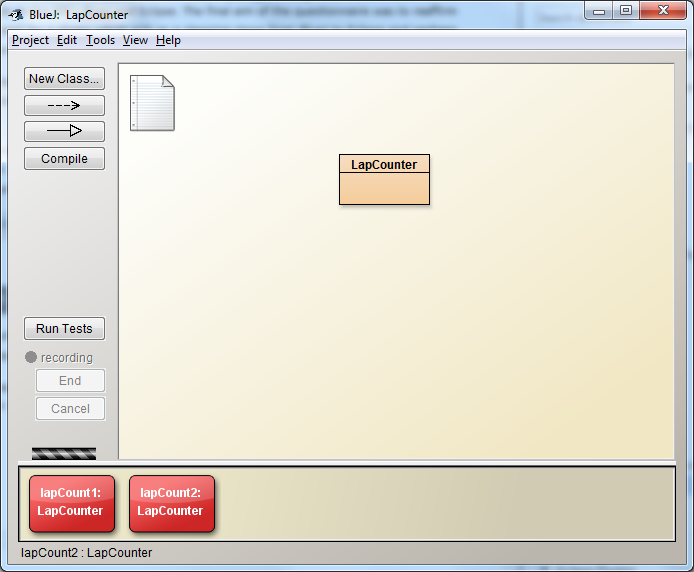


Figure 3 - The Object Bench

The object bench is highly useful as it allows for instant testing of methods and allows us to fully see the effects of code on an object. This is useful for helping students understand how object instantiation works and give an idea on how their code works. This tools is also useful for professionals, the means to test methods as we write them is highly beneficial and could theoretically lead to less mistakes when writing software.

### Class Diagram

The class diagram view within BlueJ allows the user to see the classes they write and how they are connected to each other. The class diagram helps to visualise concepts such as dependence and inheritance which are fundamental parts of object-oriented design.

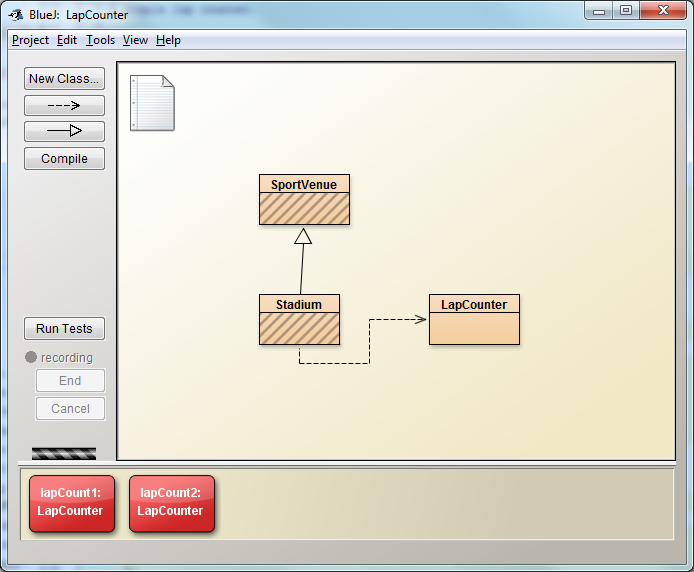


Figure 4 - A Class Diagram

A class diagram is useful to both students and professionals as it allows us to see the how a piece of software is structured. It allows us to make decisions on relationships between classes and assists us in creating logical structures that are easy for non-programmers to understand when discussing how a piece of software is created. The class diagram also provides an example of UML to students which can assist in learning how we model software in a visual manner.

### Object Inspector

BlueJ contains an inspector that can be used to view the state of an instance. This tool allows a user to see the various fields within a class and the values held by the instance. This is highly useful as it allows the user to see the results of variable declarations and can assist in the evaluation of the effects of their methods.

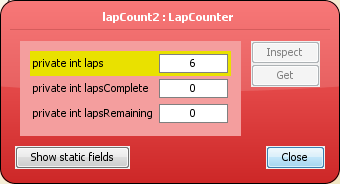


Figure 5 - The Inspector

The inspector is highly valuable in BlueJ because it allows for interaction and visualisation of instances that are not usually provided by debugging tools. This makes the inspector highly valuable for debugging and testing code and assessing the effects of the code being developed.

### Scope Highlighting

The BlueJ IDE code editor provides scope highlighting when the user is writing their code. Scope highlighting allows a user to visualise a complete “block” of code. BlueJ’s scope highlight places each section placed in curly braces into a colour coded box dependent on the depth of the code being written.

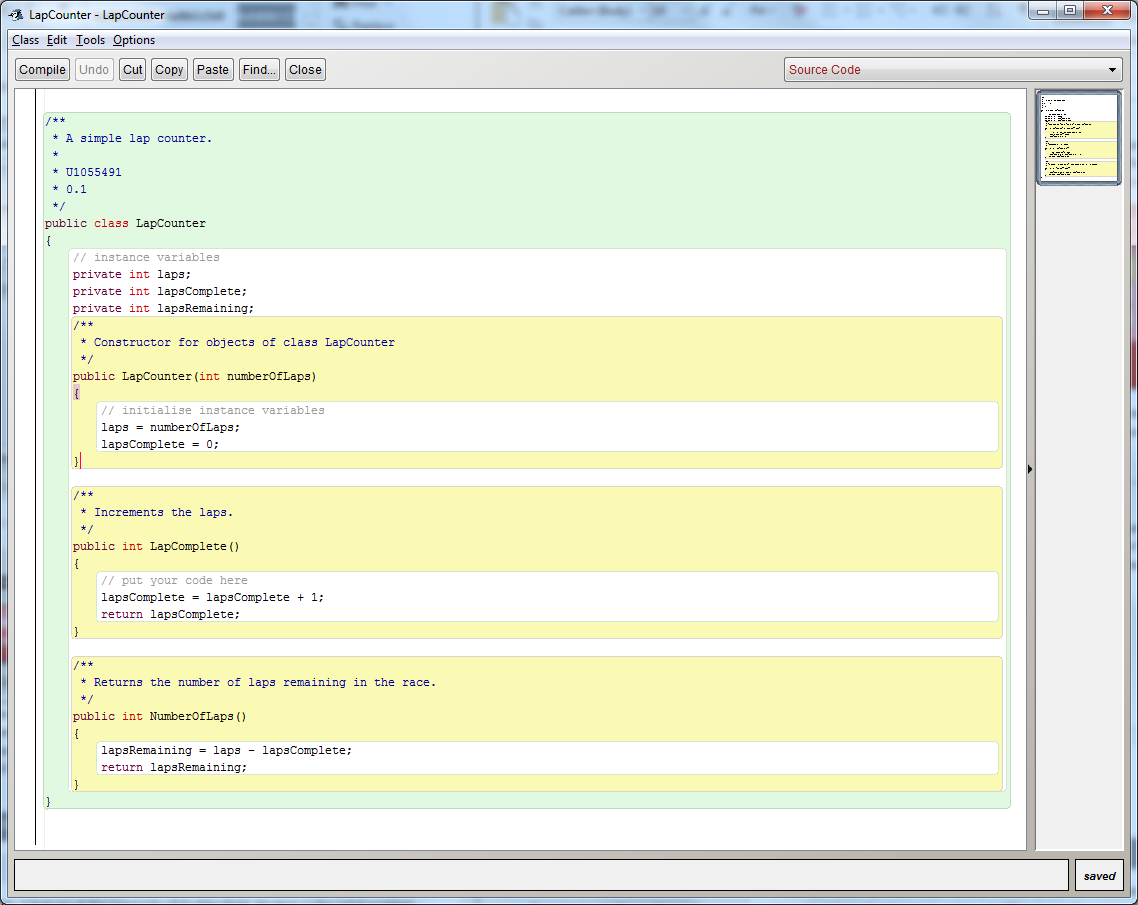


Figure 6 - Example of Scope Highlighting

As can be seen in the figure above, the class is highlighted in green, each method and the describing comments are highlighted in yellow and all code within each section is placed within a white box.

Scope highlighting makes it easier to understand the code being written and also provides another means to debug software, if a brace is missing then the highlighting will be broken. This will often help users see where they have missed a brace and allows a better visualisation of how classes and methods are structured.

This is highly useful in terms of education because it allows a student to visualise the code they are writing which should help bolster their understanding of the system being written.

### Code Templates

When creating a new class in BlueJ a user is offered a variety of options with regards to the template or type to be used in the initial setup of the class.

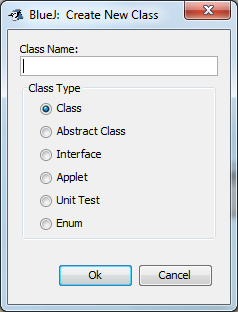


Figure 7 - Selecting a code template

A newly created class within BlueJ will utilise a template with a field, constructor and method and sample comments. This gives the basic structure of a class to a user, theoretically this should assist in preventing minor mistakes in class structure.

This allows students to re-enforce their knowledge on class structure but also provides a starting point to work from, by modifying existing structures students can gain a better level of understanding on how a specific type of class works.

### Test Class Generation

BlueJ provides the means to generate basic test classes. This allows the user to set up a set of unit tests related to a specific class.

This is useful because it could allow students a better understanding in how to write and design unit tests. Unit testing is quickly becoming a standard in testing software and as such any practice gained when learning programming is a bonus.

# Other Notable IDEs

These are IDEs that have been briefly looked at but are outside of selection due to a lack of information regarding plugin development and a vast difference in the community size when compared to NetBeans, BlueJ and Eclipse.

## JDeveloper

JDeveloper is an IDE developed by the maintainers of Java, Oracle. JDeveloper is simple and easy to use but lacks in many of the features required by many Java developers.

## IntelliJ IDEA

IntelliJ IDEA is another large IDE and has a strong community, it has a variety of tools that are useful in modern development, however to get the most out of it a commercial license is required. A free version does exist but has limited tools and features.

It is worth noting that IntelliJ has a good plug-in framework and has excellent platform support, especially for developing mobile applications within a team.

# Eclipse Plugins

Eclipse is designed to be plug-in friendly. The majority of components within Eclipse are built from plug-ins which inherits from the core kernel. By enabling developers the tools required to develop plug-ins the Eclipse IDE can be re-purposed and made to fit each individual project by enabling developers to condense their software pipeline into one inter-connected environment.

## Plug-in Development Environment (PDE)

The Plug-in Development Environment (**PDE**) provides the toolset required by developers to develop plug-ins for Eclipse. The PDE is a community driven framework that aims to improve the quality of Eclipse, built upon the Rich Client Platform (**RCP**) and Java Development Tools (**JDT**).

The PDE is broken down into three main components and two additional non-critical components which can be seen as plug-ins on their own merit.

### Main Components

**Build –** The build component of the PDE is designed for the automation when building plug-ins. The Build produces Ant scripts based on the developer designed parameters. These generated scripts fetch the relevant components for specified sources and produce a working build ready for distribution.

The Build component also allows for integration with JUnit testing so that plug-ins can be tested extensively for any issues before release. This allows a developer peace of mind in knowing the plug-in will work as expected if sufficient test cases are satisfied.

**UI –** The UI component contains all of the tools required for designing a workable user interface. The UI component is fully modular and allows for easy addition and removal of plug-in modules as required. This component contains tools for form building, wizard creation, view management and a whole host of other tools that allow a developer to create an easy to use and intuitive User Interface.

**API Tools –** The API tools are designed to assist developers in maintaining the links to the Eclipse framework. Issues such as binary incompatibilities and incorrect version numbers are controlled by these tools.

The tool is designed to:

* Identify compatibility issues between two versions fo a software component.
* Update version numbers for plug-ins.
* Update @since tags for all newly implemented methods, a @since tag is a tag added to a method/class/interface etc. when it is first added to a deployed build of a plug-in, useful for identifying bugs and issues with a specific version.
* Provide tags for javadocs to assist in code completion.
* Define the visibility of packages between different plug-ins.
* Identify use of non-API code between plug-ins.
* Identify leakage of non-API types into API.

These tools are essential in ensuring that a developed plug-in is well maintained and up to date.

### Extra Components

**Incubator – The** incubator is a component that contains new tools that are deemed not ready to be added to the official Eclipse SDK but are in need of testing.

The incubator has been the birthing ground of many key Eclipse features, through use of the Incubator new Eclipse components can be thoroughly tested and their usefulness fully evaluated.

**Doc –** The help documentation for the framework. This will be highly beneficial in learning the process is creating a plug-in for eclipse and will provide a good point of reference when difficulties are encountered.

### Extendability

All aspects of the PDE are open source and fully extendable. New features can be implemented in a local scope without the need to commit to the main Eclipse repository. The ability to customise and develop extensions enables us to create an ideal environment for our goals; we can build upon the existing foundations provided by the PDE. This allows us to save time ‘re-inventing the wheel’ and focusing on the core development of features and functionality.

## Marketplace

Eclipse host a marketplace of plugins that are maintained and available for commercial purchase and (license permitting) download. The marketplace makes it easy to find a specific type of plugin. Visibility within any online market is always a challenge, one aspect that should be considered when developing a plug-in is clarity, and a plug-in should dictate what it does in its name or as a minimum the tagline.

## Examples

There is a large variety of plug-ins available for the Eclipse IDE, the majority of which are developed with the PDE. Whilst an in-depth study on the usefulness of specific types of plug-in could be an interesting venture we only took the time to look at a few plug-ins to look at how they perform and make use of the available tools.

### EBob

Eclipse-Based Object Bench is an open-source eclipse plugin that aims to emulate the object bench functionality of BlueJ. The project is quite explicit in its aims in mimicking a key feature of the BlueJ IDE, the description of the tool (Sourceforge, 2013) states:

‘E-BOB is an Eclipse-based plug-in that allows for graphical instantiation of objects, and models to a limited extent the Object Bench of BlueJ’

This plugin contains an element that we wish to mimic from the BlueJ implementation and their implementation may be of value in studying techniques in developing the tools required for our plugin. However our client (Allen, 2014) feels that the tool is buggy and lacks polish, this is probably due to the open-source implementation of the project still being in an early stage.

### Ambient

Ambient is a simplified view for the Eclipse workbench, it provides a set of tools that load as a plug-in and is intended for use by novice computer science students. Developed as a product of the CURIOUS program hosted by Duke University, Ambient is an open source tool designed for ease of use and increased student productivity.

The key features are:

* **Assignment Integration –** Download, submit and receive feedback for assignments all in the same viewpoint.
* **CVS Management-** Simple CVS management tools intended to help new programmers ease into using CVS repositories both individually and as part of a team.

This plug-in provides a very user-friendly interface designed to ease a new developer into programming with a fully featured IDE. By being easy to use the software becomes less intimidating to new users and can instantly improve motivation and productivity.

# Teaching Programming

Programming is a difficult subject to teach as it covers a broad range of specific skills and knowledge. Programming will always provide a challenge to a student and to be apt in software development both strong problem solving and linguistic skills need to be nurtured and developed (Hartwig, 2004).

The ever evolving nature of computer science means that teaching methods have to be constantly updated to reflect new technologies and advances in the field. A key point to consider here is the rise in scripting languages such as Python; these languages are typically freer in expression of statements meaning that there is less of a need to focus on specific rules and grammar within code, which can be a major hurdle in new programmer projects.

There is an anxiety amongst University teachers that students being taught programming within their degree are given an immense amount of information and that many important aspects of programming and software development aren’t absorbed (Pyshkin, 2011). Courses involving programming often have high dropout rates and becoming classified as an expert programmer requires experience far beyond a taught course.

## Language Selection

Choosing a programming language for teaching programming is an incredibly difficult task. There are a huge variety of languages designed for generic and specific purposes. There is no language that is the definitive ‘best choice’ or ‘one-size-fits-all’ as every language brings with it a set of strengths and weaknesses.

Reghunadh and Jain of IBM (2011) suggest a number of key factors to consider when selecting a programming language. In regards to education the most relevant of these are:

* **Elasticity –** How well can the language adapt to specific needs and can code be shortened with less imports. The key point here is simply how easy it is to add new features to an existing program and how much will we need to depend on new libraries.
* **Performance –** Does the programming language compile quickly and more importantly are applications efficient when compiled? Performance is important when selecting a programming language because it is important to teach the importance of saving time and ensuring a high degree of efficiency.
* **Support and Community –** The more people that are using the language the more likely it is that help and support can be found. This is crucial when selecting a programming language for learning as it enables student to independently find information about a language. This information is often provided from experts with deep knowledge of how a language works, ultimately the more information that can be learned about a language the more likely a student is to fully engage and utilise the features of said language.

Whilst the authors are inclined to agree with the points outlined it is felt that there should also be some considerations to the following areas:

* **Industry Trends –** It is important to look at the languages being used in industry and how prominent and employable a specific language will be. For example, it is useless teaching a scripting language like JavaScript on its own as the majority of jobs in web will demand knowledge of several languages.
* **Grammar Complexity –** If the grammar is overly complex valuable time could be lost on performing basic tasks within a programming language. An ideal solution would be to select a ‘high-level’ language that allows for a loose and free grammar and then move into a more formal language once a base understanding of programming concepts has been obtained.

It is impossible to satisfy every individual with the choice of programming language, one language may be ideal for the majority but suffer from a vocal minority that prefer another language. This can also occur in reverse situations where the majority prefer another language over the selected language.

## A Basic Process To Learning Programming

It is important that a teacher keeps initial information simple and expand and refine once a student has had the opportunity to gain experience and develop their skills. Gaining understanding is crucial so all topics discussed must be scaled relative to known experience levels of students, this ensures that students can be eased into the complexities of programming when they are at an appropriate knowledge level(Winslow 1996).

Researchers Linn and Dalbey(1989) suggest an ideal methodology for learning programming, they suggest an approach that ultimately will allow a student to develop the necessary skills.

### Syntax and Semantics

The first stage in learning programming should be to get familiar with one language and one language feature at a time. Students should spend some time becoming familiar with the syntax in a language and then learn each feature in detail, possibly with small projects to encourage use of specific features.

Students should be exposed to code frequently and be encouraged to both read and write small programs that help build on fundamental knowledge.

### Combinations

The features learnt during the initial phase of education can be combined with known design skills to begin to develop programs that solve problems. An emphasis should be placed on improving design skills through use of models. Models are crucial to building an understanding of the concepts in programming, models can be used for control, and data structures, data-representation, software design and domain are all important topics to be developed once a core understanding of the language is developed.

We have to be careful to encourage correct combinations of design and implementation to ensure that a student maintains a high standard of quality and combines the correct features into meaningful application code (Winslow 1996).

### Problem Solving Skills

There should be a continual drive to improve students’ problem solving skills, both specific to programming and in a generic sense. A good programmer will almost always have strong problem solving skills and have a large degree of enthusiasm towards solving a given problem. It is important that a student gains practice in problem solving. Teachers should continually pose problems and engage in discussions as to the optimal solution to a problem.

## Procedural and Object-Oriented

There is often debate in teaching programming whether to adopt a procedural or object oriented approach to teaching. Both methods offer opportunities for problem solving and are viable options of building software but there are of course caveats of either method.

Procedural programming utilises top down design, in this style of programming we create procedures to perform specific functions of the program that can be called from anywhere within code. This is functional decomposition; in this we split a problem into multiple procedures until it can be solved. The main issue with this method of programming is that it is incredibly difficult to maintain and can be time consuming and expensive to develop. If a developer changes something in one small procedure it may affect the entire program leading to new bugs and issues.

The upside to utilising procedural programming techniques are that algorithms are easier to explain and implement for a new programmer as the core overhead is small and there is little need to worry about complex topics such as instantiation and handling objects. Procedural programming is also arguably quicker in performance but admittedly the point becomes moot where the difference is minimal.

Object-oriented programming addresses many of the difficulties in procedural programming by allowing the creation of classes that are used to model real-world objects. In object oriented programming data is supposed to be locked inside an object and accessed through functions, “good” object oriented practice means that classes never share global data and each object only makes use of the data they have access to or data that is passed in through a function. The key issue with an object oriented approach is that it is complex and could take a substantial amount of time to learn to its full potential. Object oriented programming requires a different set of skills and needs to be more thoroughly planned and approached with a specific mind-set. Often developers using object-oriented programming will question whether they are writing code correctly and may sacrifice functionality over following conventions. This can lead to a lack of productivity (Stannard, 2009).

However, when object-oriented systems work well they are invaluable, code can be re-used and classes can be instantiated for manipulation, this makes handling large volumes of data much more manageable. Code becomes more readable and easier for new developers to understand. The majority of programming done in industry today utilises an object-oriented approach when developing software.

In teaching programming both approaches should be considered and possibly an attempt to create a curriculum that incorporates both styles could be most beneficial for new programmers and students. By approaching both methods both algorithmic problem solving and proper object oriented programming concepts such as code re-use can be put to use.

## Other Considerations

### Original Projects and the Rise of the Internet

Students should have experience in both reading and writing code. One of the best approaches towards this is to utilise existing good solutions. Students can make use of the existing solution and use it to write a similar approach but acknowledge and extend upon knowledge gained with additional tasks and challenges.

Existing code can be used to highlight good common practice and highlight code formatting and standards. However it is important that a student then utilises this knowledge and writes original code that mimics the functions of the specified system, by allowing a student to write code to mimic that of an existing source we can begin to gauge the understanding and insight gained from a particular project.

With one of the key resources for students being the internet, we have to be careful of plagiarism in any work undertaken when learning programming. The internet has a wealth of information on programming and solutions to common programming problems. It is increasingly easy for a student to take an existing solution and blatantly plagiarise it, thus the student doesn’t learn anything from the solution to the problem. However it is worthwhile encouraging students to look at other code, but not to use it for their own needs, looking at code can teach a student about common practices such as coding style and layout as well as basic conventions for naming and structure.

## Information Availability

When posing a problem to be solved we have to be cautious in changing it enough that a direct solution cannot be found with ease. Today it is very easy to search for a specific solution to a problem and copy it without any ethical consideration; this poses the problem that a student may not learn from this practice as they have simply copied a solution instead of solving it for themselves.

This problem can be solved by ensuring that large projects are used from an early stage and a variety of techniques are taught to create a final product, thus encouraging more software design and development and theoretically less plagiarism.

# Teaching Object-Oriented Programming

When teaching programming it has become standard to teach object-oriented languages such as Java, C# or even C++ over procedural languages such as cobalt (Kölling, 2003), this is due to the flexibility offered by object oriented programming languages however the entirety of the pros and cons of object-oriented programming fall out of the current scope it is worth noting that the authors agree with this approach and understand the benefits that can be obtained from teaching this discipline in programming. Kölling et al. (2003) suggests that the difficulty of teaching object-oriented programming lies in the lack of available tools and resources specifically for the purpose; this can be interpreted as a need for more tools with the specific purpose of teaching object-oriented programming.

## Shortcomings of Tradition Systems Environments

There are three fundamental flaws in the environments used in traditional teaching systems (Kölling, 2003):

* **Not Object Oriented –** Using an Object Oriented language does not necessarily mean that areas taught will be object-oriented. It is important that lessons reflect the features of a language, a focus should be placed more on classes and objects over traditional input and output solutions. Far too much emphasis is placed on utilising file structures and working with source code over creating objects and their manipulation.
* **Too Complex –** Many teachers do not make use of an IDE (Kölling, 2003) as they cannot find one that works well for all, however this means that students have to waste time learning a command line instead of focusing on programming. Of course there could be issues with an IDE being too complex as well, it is important to find a suitable middle-ground that is conducive to the learning process.
* **Focus on UI –** Many IDEs place a focus on GUI when developing software. Often this means that a student will lose focus on the actual programming and not get to actually see how the code should be written by hand. NetBeans (used to be) a particularly bad IDE for this as it did not allow the user to edit generated GUI code meaning that a user couldn’t tweak things for experimentation and learning. We should be careful that an IDE does not get seen as a GUI builder and try to highlight features that can ease the creation of object-oriented systems.

These flaws aren’t often a major hurdle in teaching programming but modern approaches should take these points into consideration. It is highly beneficial to create a working environment that students are comfortable working inside.

## BlueJ Approach

BlueJ provides functionality that is beneficial to the learning of programming in an object oriented manner, the IDE can be utilised with several traditional textbooks and methods for teaching concepts of OOP and implementation of various algorithms. It is suggested by Kölling et al (2003) that when teaching OOP the content of teaching should, at least initially, focus on BlueJ and utilising the environment which is well suited for learning programming.

Kölling et al (2003) suggest the following guidelines when creating course content for teaching OOP:

* **Objects First –** When teaching object oriented programming it is important to put forward the concepts of objects as soon as possible, perhaps making use of real world examples to better illustrate the point.
* **No Blank Screen –** It is worthwhile always starting with a project that does something instead of a blank canvas. This will allow students to see how things work and also tie in with the next guideline of reading code to see how things work.
* **Read Code –** Students should be encouraged to read code to better their understanding of how classes are structured and how to instantiate classes to objects and make use of the methods within.
* **Use Large Projects –** Using large projects allows a student to apply problem solving techniques and also push their understanding of OOP concepts in a real world sense. Utilising large projects will enable teaching complex concepts early and practically.
* **Don’t Start with Main –** Understanding the main method is often awkward and can confuse new students, BlueJ provides means to ignore the main method until a student is prepared to venture into more complex projects. However the main method should be tackled in an early session so that parameter passing and a base understanding of how programs are run within a Java environment can be obtained.
* **Don’t Use Hello World –** Whilst ‘hello world’ is a traditional first program it does not highlight any concept of object oriented programming and is therefore a wasted venture in this area.
* **Show Program Structure –** With any project or program it is essential that students be shown the entire structure of the code to gain a better understanding on how to structure their own projects.
* **Be Careful with User Interface (UI) –** UI coding is a complex and often misleading subject, often utilising UI code (especially those with non-standard libraries) can obfuscate the object oriented nature of the core source code.

By following these guidelines it is possible to teach object oriented programming to an individual with minimal programming experience. This pathway ensures that the core concepts of OOP are made important early on and a focus on objects and instantiation allows a student to develop a core understanding from which to build onto.

This method is quite effective and provides a good baseline to learn OOP, however Xinogalos et al. (2007) finds that designing a course around these guidelines can often create a series of problems. For example students would often become confused around public and private fields and struggled to comprehend the main method. Xinogalos et al. (2007) agree with the majority of the guidelines but suggest that the main method and starting with a blank canvas should not be avoided for too long as these things may confuse students led into a false sense of security. It is also suggested that students should also write more code alongside reading it to put the skills being taught into practice.

## Transition

One of the most difficult aspects of teaching someone programming is the transition from utilising learning tools such as BlueJ to a fully-fledged IDE such as Eclipse (Kölling, 2003). It could prove to be highly beneficial for a student to be able to use tools they are familiar with in a new and more complicated environment. Often concepts that are introduced within an environment created specifically for learning are difficult to transfer into a more traditional programming environment. Tools like BlueJ are intended for beginners and to assist in teaching and understanding the concepts behind object oriented programming in an environment that can visualise the concepts as they are taught. The creation of a tool that can bridge the gap between a learning tool such as BlueJ and a complex environment such as Eclipse could assist future programmers in making the leap from the simple to the more complex and powerful. Kölling et al.et al (2010) suggests that tools like BlueJ should only be used for the first year in learning programming and students should be encouraged to make use of more traditional IDEs. It should be emphasised that this transition should be made as painless as possible and the argument could be made that some of the tools provided by learning environments could be useful in a more traditional environment.

The argument could be made that providing a means to transition between platforms is akin to ‘hand-holding’ and that providing students with the tools they are used to is potentially going to stifle curiosity and stop prospective programmers from exploring more complex tools, instead favouring a familiar and ‘easy’ environment.

Features of the proposed tool such as class visualisation could help with project organisation, the object bench could theoretically prove to be invaluable to the testing of classes, and a study could be conducted to see the benefits of combining an object bench with unit testing to determine if there are any clear advantages to this approach.

# Designing Software For Teaching/Learning

When designing software that is intended to support a user in learning and education we have to consider a multitude of elements. It is crucial that the software is designed in a way that supports the knowledge being taught.

### Ease of Use

Any software designed to assist in learning should be easy to use. The application should feature an intuitive workflow and enable a student to perform actions with little guidance. This is important because if a piece of software is difficult to then it is less likely that a student will gain any educational value from it and may possibly forgo using the software altogether. With difficult software a user will make more mistakes, whilst there is some benefit to learning from mistakes it is more critical that a student gain a sense of accomplishment from their actions.

The software should also be engaging and interesting to ensure that a student remains invested in their education within a specific area.

### User Interface

The user interface (**UI**) is the communication between the user and the intricacies of a developed program (Faghih et al, 2013). The UI should be simple, clear and easy to use. We should look at the intended user base and assess their age, abilities and previous experience and ensure that we design a UI that can be used by anyone with a base line level of knowledge (Fagih et al, 2013).

We should consider three key points when designing the interface:

* **User in Control –** The user should be made to feel in control of the software and have the freedom to make changes and arrange elements in a manner that suits them.
* **Reduce Visual Information** – We need to ensure that we are only displaying relevant information and provide options to expand make the information available but ensure that it is placed in relevant sub menus.
* **Consistency** – It is important the UI has consistency throughout, dialogs should follow set layouts, fonts should always be the same and the application should maintain a consistent look and feel. This will ensure a user remains comfortable with the software at all times.

A good UI design will always keep a user interested and enable them to make use of the application no matter their skill level.

We could perform user and task analysis and ensure that we model the interface in a manner that is suitable for all users and then validate this against our client and potential users.

### Error Handling

In an ideal world software would have no errors and everything would work as intended, however this is an unrealistic vision and there will always be issues that arise when developing any software.

In developing software for learning it is critical that all errors are handled in a meaningful and discrete manner. Poor error handling could lead to an inexperienced user panicking and ultimately causing more issues in the end. Errors should always be handled with a clear and concise explanation and provide steps to fixing an issue.

When developing the software we will have to be wary of all inputs and outputs and ensure that we place code safety as a high priority to ensure minimal bugs and errors.

### Documentation

Documentation is a key part of educational software, everything should be well documented and any samples should be reproducible from it. Good documentation ensures that if a user is unsure on an aspect of the software they have a point of reference to support their existing knowledge and allow them to troubleshoot any issues they may have.

It is important to maintain both technical and user documentation to ensure that the system can be both used to its full potential and maintained by possible third parties. Teachers can utilise documentation as a basis for some lessons, without good documentation it means that more time has to be spent working out a program instead of using it, thus slowing productivity drastically.

Bad documentation may lead to a user being confused about how to use certain aspects of a system and may lead to misuse or mistakes being made.

# Methodology

When a software development project is under taken one needs to decide on which methodology will be chosen to help provide structure and guidance during the entire software life cycle. These methodologies aim to divide the development work into distinct stages within the software life cycle, guidance to better planning and management is also provided in some methodologies.

The software lifecycle which many methodologies reference to are usually split up into the following generic headings:

* Analysis
* Design
* Implementation
* Testing
* Maintenance

However these stages are often broken up into smaller stages.

It is interesting to note that the software lifecycle is a concept which was first was first recognized in the 1960’s, possibly considered to be the oldest methodology.

Currently the common methodologies that are used are the following:

* Waterfall
* Prototyping
* Iterative and incremental development
* Spiral development
* Rapid application development
* Extreme programming
* Agile

Note that Agile is a generic term to describe methodologies which follow the manifesto for agile software development (Beck, Kent et al. 2001).

## Waterfall

The waterfall model is based on a sequential design where progress is flowing downwards through each phase of the software lifecycle (like a waterfall flows downwards).

Originally this model was used in manufacturing / construction which later in the 70’s was adapted for software development (Herbert et al 1983). The waterfall methodology was first formally described in an article by Winston W.Royce (1970) in 1970.

The model consists of the following 7 phases (in order):

* Requirements specification
* Design
* Construction (implementation)
* Integration
* Testing
* Installation
* Maintenance

This methodology has received several positive comments throughout the years. First of all the waterfall methodology has been around for many years and therefore has an extensive knowledge base within the software development community. It also provides disciple as each stage has a start and end date and therefore progress and be easily identified. Another strong positive is that documentation is heavily encouraged as each stage has a formal specification, documentation always helps in knowledge transfer within an organisation.  
However more recently this methodology has been criticized for its flaws. Having a requirements stage at the beginning where the customer has their input on what they want allows for risk of their requirements not being met at the end of the waterfall process. Often customers do not know what they exactly want, requirements also change over time and the waterfall methodology provides no facility for customer input past the requirements specification phase. The model also assumes that each phase reliably moves onto the next phase, for example Design to Implementation. Usually once implementation begins often the design will have to get adjusted, mainly as some designs may not be feasible as once thought.

## Prototyping

Prototyping is a methodology which aims to create incomplete versions of the software product at certain stages of the development process. A prototype will usually simulate certain aspects of the finished product, for example the way the GUI works.

There are two main types of software prototyping, Throwaway and Evolutionary (Smith 1991).

Throwaway prototyping refers to the creation of a prototype that will be discarded when its use has become obsolete, rather than being integrated into the final product. These prototypes can be used to show a customer partially accomplished requirements thus allowing to gather useful feedback, this can be achieved relatively quick as these prototypes can be created in a short time.

Evolutionary prototyping is quite different to throwaway prototyping, in the sense that a robust prototype will be built instead which will then be refined as time goes on. This allows for making additions and changes to the product as time goes on, this is quite helpful when the customers’ requirements have changed. Another advantage to evolutionary prototyping is that it allows for the programmers to tackle the easier aspects of the product first and then work up to the more complex aspects.

## Iterative and incremental development

This methodology is a combination of both iterative design / method and incremental building / implementation (Larman 2003). Here to mitigate risk the project is broken into smaller segments and then worked on, this results in an incremental development approach. Usually a mini-waterfall model is used within each broken up segment, where each requirement is then accomplished. The advantage of this is that it allows for developers to learn from each previous segment and thus allowing for improved future work.

Each segment can be iterated over, the number is dependent on the project type and other variables such as the customer’s needs.

## Spiral development

The spiral model is a model which is driven by risk, based on the projects risk the model givers guidelines on adopting other models (incremental, waterfall etc.).

The spiral model was first mentioned by Barry Boehm in 1986 in the paper “A Spiral Model of Software Development and Enhancement” (Boehm 1986).

The spiral model is broken into 4 basic sections, determine objectives, identify and resolve risks, development & test and plan the next iteration.

The basic principles of the spiral model are:

* Focus is on risk and minimising project risk
* Each cycle has the same progression of steps
* Each trip around the spiral involves going through 4 basic sections.
* Each cycle begins with identifying the stakeholders and their requirements.

Due to the nature of the spiral model it has been adopted in many different ways and therefore lots of development teams have different views on how to use this model.

## Rapid application development

Rapid Application Development or RAD is a methodology which favours an iterative approach to development with rapid construction of prototypes. This approach therefore usually ignores a lot of the planning and designing of the final product, rather the planning/design of the software is integrated within the implementation of the software.

The RAD process usually starts off with the creation of data and business process models, after this the requirements are accomplished via prototyping. This stage is then iterated until all the requirements have been met resulting in a finishing product.

The basic idea of RAD was finalised by James Martin in his book Rapid Application Development in 1991 (Martin 1991. His approach divides the model into 4 phases:

* Requirements planning phase
* User design phase
* Construction phase
* Cutover phase

## Extreme programming

Extreme programming (also known as XP) is a methodology which aims to improve the quality of the software and the flexibility to the customers changing requirements. XP is similar to agile software development (extremeprogramming.org 2014) in the sense that it promotes frequent releases during each development cycle.

XP provides a lot guidance to programmers especially within the implementation phase of any product. Elements such as pair programming, extensive code review, avoiding programming of features before they are needed and so on.

XP has four basic activities that much be performed within the development process, these are:

* Coding
* Testing
* Listening
* Designing

With this XP’s goal is to reduce the impact of changes via change of requirements by having a multiple of development cycles. XP teaches that changes are always welcome and therefore this should be planned for rather than planning a set of stable requirements from the beginning.

## Agile

Agile development is a group of methodologies which are based on iterative development, they all follow concepts and guidelines of the Agile Manifesto. These methodologies have an iterative approach in their fundamental structure but add more guidelines to business processes, management and social side of things.

### DSDM

Dynamic systems development method or DSDM is an agile project delivery methodology but is primarily used as a software methodology (dsdm.org 2014).

It was first released in 1994 with the intention to provide more disciple to RAD. It is an incremental and iterative approach which uses principles from agile development. It uses the MoSCoW prioritisation of requirements, from there it then performs a feasibility assessment. Due to DSDM being so vague it isn’t just used for software projects but for non-IT projects as well.

### Scrum

Scrum is an agile development methodology which uses an iterative and incremental approach. The methodology emphasises that a customer needs will change and therefore this must be incorporated when working on the product.

Scrum provides guidelines on how to manage a software project by defining certain rules and roles. The three main roles are (Schwaber 2013):

* Product owner
* Development team
* Scrum master

The product owner is usually the customer who wants a certain software product building, he provides the requirements and will be available throughout the entire software process.

The development team will be the team responsible for working on the product, from beginning to end (Requirements to maintenance).

The scrum master is one member of the development team who manages the team and project. He ensures that the scrum methodology proceeds as intended, he provides guidance to team members, and holds meetings and events.

Events are within scrum are described as meetings and activities, the events consist of:

* Sprint
* Sprint Planning
* Daily Scrum
* Sprint Review
* Sprint retrospective

A sprint can be seen as an iteration which is limited by a time, they usually last for a month. Each sprint starts with a sprint planning event. Here what needs to be done and how is identified.

Daily scrums involve the team meeting up at the beginning of each day, here they tell each other their progress, what work they will be doing and if there are any obstacles preventing them from doing their work.

At the end of the sprint a sprint review and retrospective is held. The review involves looking back at what has been achieved and what hasn’t been achieved. The finished work is also presented to the stakeholders (customer for example) where feedback can be gathered. The sprint retrospective involves reflecting on the past sprint and identifying any improvements for the next sprint.

Lastly the Scrum methodology has artefacts; these are the documents that are produced during the software development process. These are broken down into the following:

* Product backlog
* Sprint backlog
* Burn down chart

# Conclusions

## IDE Selection

Selecting an IDE is a difficult task that requires an analysis of the strengths and weaknesses of all IDEs designed for use with a selected language. It is important the IDE matches the developer and their needs for a comfortable programming environment. The key factor in selecting an IDE is, as a matter of opinion, the extensibility of an IDE, being able to add new features through plug-ins is an invaluable tool in any developer’s tool-box.

Both IDEs for beginners and IDEs for professionals play an important part in both education and software development. There needs to be tools that help to learn as well as tools that enable development. There is a distinct lack of the former type of software at the moment, developers often lack the tools required to assist their learning, and ultimately this means more time is spent learning which could be spent developing the next innovative piece of software.

There are currently no existing solutions ‘built into’ existing IDEs that aid users with a transition from a beginner tool such as BlueJ into a full professional environment. This suggests that there is an opening for innovation and a unique project that solves a real problem. A lack of tools to help in transitioning means that individuals learning programming are having to spend more time independently learning how to use features of a fully-fledged IDE whilst ‘letting go’ of features in a novice IDE such as BlueJ.

A factor that should be considered throughout the development of this project is portability, it would be valuable to be able to utilise the code base across multiple IDEs thus providing a solid solution for transition that can be used by everyone with an interest in taking their programming from beginner to professional.

After extensive research into the various IDEs available Eclipse is the one that will be used to develop the transitioning plug-in. Eclipse has the best supported plug-in framework and seems best suited to meet the requirements of the project. The community support that is available for the Eclipse PDT will be incredibly useful at points where issues may develop due to a lack of core knowledge and experience in developing plug-ins for Eclipse.

## Modular Development

In studying the NetBeans IDE one aspect that was of particular interest was its modularity. Whilst we are making use of Eclipse for this project this is something that we can learn from and should strive to incorporate into the end product. Having a modular design enables us to better develop single modules independently, this will better suit our small team size, it will also benefit the end users as they will be able to utilise the things they need and disable anything they don’t.

## Teaching Programming

Teaching programming is incredibly difficult because of the highly wide variety of programming languages, techniques and methodologies. It is highly likely that there will always be a need for programmers as new languages are constantly emerging. An argument can be made that an adaptable programmer who can make an acceptable use of a variety of different languages could prove to be a more valuable asset than a specialist in a specific language for the majority of software development opportunities.

An approach that covers both the procedural and algorithmic side of programming and an object-oriented approach needs to be researched in more depth and whether there is one “correct” approach to teaching programming is a highly subjective view. The creation of our plugin will enable further research into the an object oriented approach that will

There is currently a severe lack of tools that can help educate students in programming and good software development ideals. It is important that software is designed that makes it easier for students to learn in an environment that allows them to make mistakes and learn to utilise all the features of a given program language easily and intuitively.

Both an increase in the amount of learning and the quality of learning are important going forward and any contribution that this project can make could have a positive impact on anyone wanting to learn programming. To this end it is important that we evaluate the pedagogical benefits of our proposed system.

## Our Methodology

When we were deciding on which methodology to adopt for our project we had to perform some research first. We also had to point out some unique variables / characteristics that were involved in creating a plugin project for eclipse. The main variables / characteristics to note were:

* 2 members in the group
* No previous experience in the project type
* Limited time
* Other unrelated responsibilities (other modules in our case)

These characteristics may also be seen as risks, therefore the methodology we adopt must incorporate this in some way.

Aspects of waterfall, incremental and iterative, prototype and scrum were investigated as these were believed to be the most applicable to our project.

As our projects characteristics are rather unique we had to cherry pick all the useful aspects of each methodology and make it work for us. As there is only 2 of us it means that there will be no need for a scrum leader, due to the unknown nature of the final product due to lack of previous experience performing a full waterfall will be very time consuming and impractical, time is another variable which is limited to us.   
However scrum does provide us guidelines on how to keep our product satisfactory to our client by having frequent meet-ups and demo sessions. Waterfall gives us strong direction towards keeping documentation of every phase within the software development process. Incremental and iterative / prototype methodology supports us in creating a product in small but iterative steps. This will be necessary when working on a project with no prior knowledge as research will have to be alongside working on the project through each software lifecycle phase.

We decided to go for a mix of scrum and incremental & iterative and prototype approach for the group project. Weekly meetings with the client (our group tutor) of current progress and any demonstration along with opportunity for feedback resembles a sprint in scrum. The incremental and iterative approach allows us to research and develop at the same time and allow for continuous improvements as time goes on along with evolutionary prototyping. This type of prototyping allows us to have a working product which we can demo to the client and any other stakeholders. This process will be used by both members of the team as it provides moderate direction and flexibility in tackling the group project of creating a plugin bench for Eclipse.

## Final Thoughts

Programming is a highly difficult area to teach effectively and the methods currently in use will not take any programmer to an ‘expert’ level. Most programmers will leave a programming course with a level that can be described as competent; it is only through years of experience that becoming an expert will be viable.

There is a severe lack of tools that enable teachers/lecturers the ability to teach certain concepts to a full and critical level of understanding.

Our project will potentially fill a gap in transitioning from a basic IDE like BlueJ into a fully professional environment such as BlueJ. As a result of this we have to attempt to create an end product that is both competent and fully featured so that it helps ease the difficulties usually found during transition? It is also crucial that we develop a tool that can be utilised by both competent programmers and novices, a tool that is both educational and useful.

# Future Research

There is room for looking into a merger of teaching techniques that utilise a variety of techniques to attempt to build a new curriculum that enables for better programmers.

We could also expand this research into being more focused in the variety of nuances within the Java language and attempting to see the manner in which our tool can be used in larger projects using a variety of complex libraries. There is a potential for seeing how this tool could be used for testing in professional programming environments and its usefulness as a general purpose tool over its usefulness as a learning tool.

# References

(E-BOB),. (2013). *Eclipse-Based Object Bench (E-BOB)*. *SourceForge*. Retrieved 25 November 2014, from <http://sourceforge.net/projects/ebob/>

Barnes, D., & Kölling, M. (2005). *Objects first with Java*. Harlow, England: Pearson/Prentice Hall.

Beck, Kent et al. (2001). "Manifesto for Agile Software Development". Agile Alliance. Retrieved 14 November 2014.

Benington, Herbert D. (1 October 1983). "Production of Large Computer Programs". IEEE Annals of the History of Computing (IEEE Educational Activities Department) 5 (4): 350–361. doi:10.1109/MAHC.1983.10102.

Boehm B, (August 1986). "A Spiral Model of Software Development and Enhancement", ACM SIGSOFT Software Engineering Notes, ACM, 11(4):14-24.

Cs.duke.edu,. (2014). *Ambient*. Retrieved 25 November 2014, from <http://www.cs.duke.edu/csed/ambient/index.htm>

Dsdm.org,. (2014). Front Page | DSDM CONSORTiUM. Retrieved 24 November 2014, from http://www.dsdm.org/

Extremeprogramming.org,. (2014). Extreme Programming: A Gentle Introduction.. Retrieved 8 November 2014, from <http://www.extremeprogramming.org/>

Faghih, B., Azadehfar, M., & Katebi, S. (2015). User Interface Design for E-Learning Software. *The International Journal Of Soft Computing And Software Engineering [JSCSE]*, *3*(3). doi:10.7321/jscse.v3.n3.119

Geer, D. (2005). Eclipse becomes the dominant Java IDE. *Computer*, *38*(7), 16-18. doi:10.1109/mc.2005.228

Goth, G. (2005). Beware the March of this IDE: Eclipse is overshadowing other tool technologies. *IEEE Softw.*, *22*(4), 108-111. doi:10.1109/ms.2005.96

Kölling, M., Quig, B., Patterson, A., & Rosenberg, J. (2003). The BlueJ System and its Pedagogy. *Computer Science Education*, *13*(4), 249-268. doi:10.1076/csed.13.4.249.17496

Larman, Craig (June 2003). "Iterative and Incremental Development: A Brief History". Computer 36 (6): 47–56. doi:10.1109/MC.2003.1204375. ISSN 0018-9162.

Linn, M., & Dalbey, J. (1985). Cognitive consequences of Programming Instruction: Instruction, Access, and Ability. *Educational Psychologist*, *20*(4), 191-206. doi:10.1207/s15326985ep2004\_4

Martin, James (1991). Rapid Application Development. Macmillan. ISBN 0-02-376775-8.

Michael, H. (2004). Challenges in teaching programming. *New Straits Times*, p. 23.

Murphy, G., Kersten, M., & Findlater, L. (2006). How are Java software developers using the Elipse IDE?. *IEEE Softw.*, *23*(4), 76-83. doi:10.1109/ms.2006.105

Netbeans.org,. (2014). *A Brief History of NetBeans*. Retrieved 25 November 2014, from https://netbeans.org/about/history.html

Objectorientedcoldfusion.org,. (2014). *Procedural vs Object Oriented Programming*. Retrieved 25 November 2014, from http://objectorientedcoldfusion.org/procedural-vs-object-oriented.html

Petri, J. (2010). *NetBeans platform 6.9 developer's guide*. Birmingham, UK: Packt Pub.

Projects.eclipse.org,. (2014). *PDE - Plugin Development Environment | projects.eclipse.org*. Retrieved 25 November 2014, from http://projects.eclipse.org/projects/eclipse.pde

Reghunadh, J., & Jain, N. (2011). Selecting the optimal programming language. *IBM Developer Works*.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, *13*(2), 137-172. doi:10.1076/csed.13.2.137.14200

Royce, W. (1970). Managing the development of large software systems. IEEE WESCON, 1-9.

Sam-Bodden, B. (2006). *POJOs*. New York: Apress L.P.

Smith MF, (1991) Software Prototyping: Adoption, Practice and Management. McGraw-Hill, London.

Schwaber K.,Sutherland J. “The Scrum Guide". Scrum.org. Retrieved November 28, 2014.

Techopedia.com,. (2014). *What is an Integrated Development Environment (IDE)? - Definition from Techopedia*. Retrieved 25 November 2014, from <http://www.techopedia.com/definition/26860/integrated-development-environment-ide>

Wells, D. (1999). *Pair Programming*. *Extremeprogramming.org*. Retrieved 15 April 2015, from http://www.extremeprogramming.org/rules/pair.html

Wiki.eclipse.org,. (2014). *Eclipsepedia*. Retrieved 25 November 2014, from https://wiki.eclipse.org/

Winslow, L. (1996). Programming pedagogy---a psychological overview. *ACM SIGCSE Bulletin*, *28*(3), 17-22. doi:10.1145/234867.234872

Xinogalos, S., Satratzemi, M., & Dagdilelis, V. (2007). Teaching java with BlueJ. *ACM SIGCSE Bulletin*, *39*(3), 345. doi:10.1145/1269900.1268914