Chapter 1

Introduction

This thesis identifies and develops specific ways of improving Empirical Modelling tools and concepts as a *plastic software environment* to better support *plastic applications*. The terms *plastic software environment* (PSE) and *plastic applications* (plastic apps) have been introduced by this thesis to classify a certain kind of software and software environment, the definitions of which are given in this chapter in order to place the rest of the thesis in context. Empirical Modelling is an existing area of research that provides a good foundation for the support of plastic apps and will be briefly introduced here with more detail given in chapter 2. The remainder of this chapter will then discuss more specific thesis aims and questions and give an outline of each chapter.

1.1 Plastic Applications

The notion of *plastic applications* is for the most part an extrapolation from what is currently possible, and relates to the vision of making *software soft* [Fischer, 2009]. It is a vision where an artefact can be sculpted in a creative fashion from the ground up by end-users, where the artefact gradually evolves and solidifies into a useful, non-trivial, sophisticated application. In the beginning a plastic app could be some informal experimental model developed by a user who can then use this to learn about their specific

problem through continued experimentation. At any point, even after it has solidified to a usable application, the application can be adapted and moulded into something new. This subsequent adaptation is not so plausible in traditional programs that have been developed and optimised specifically for a machine. The result of enabling plasticity is a concrete contextualised application as opposed to a generic off-the-shelf product. An additional part of this vision would be the ability to link many different plastic apps together in unforeseen ways to produce an entirely plastic computing environment, perhaps even a plastic operating system[1].

The concept of plastic apps comes from a collection of research areas, including end-user application development (EUD) [Lieberman et al., 2006], formerly end-user programming (EUP). Nardi gives a clear introduction in her book entitled "A Small Matter of Programming" [Nardi, 1993] as to why end-user programming is vital. More recent work by others discusses how, in specific cases such as with spreadsheets, it has proven remarkably successful [Burnett, 2009]. One such argument given by Nardi is that end users "have the detailed task knowledge necessary for creating the knowledge-rich applications they want" (p.xi) and that "the process of transferring domain knowledge to a programmer... is inefficient" (p.123). Perhaps, however, we are dealing with end-users who do not have a detailed task knowledge but who might be attempting to understand a particular solution to a problem whilst they are developing the application? In which case EUD is also a learning experience [Repenning and Ioannidou, 2006]. A claim made by Nardi that I strongly agree with, and which is still valid today, is that "we have only scratched the surface of what would be possible if end users could freely program their own applications" [Nardi, 1993, 3].

EUD has been defined as "*a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify, or extend a software artefact*" [Lieberman et al., 2006]. It has been claimed by some that EUP and perhaps EUD has succeeded [Burnett, 2009]

[1]Operating systems may already be considered as plastic, but not to the degree being proposed in this thesis.

as we now have spreadsheets, web authoring tools, graphical languages and various educational tools. To a degree this is true - there are now over 50 million end-user programmers who adapt or create software in some form. However, is the software they are creating or adapting really that plastic? Certainly as an everyday user I am not often able to adapt software to my personal needs except in the most specific and pre-defined ways such as with e-mail filters or user interface customisation, or by learning how to program and write custom extensions. The problem seems to be that EUD systems in practice are typically domain restricted or only support a tiny amount of end-user adaptation of existing environments [Repenning and Ioannidou, 2006]. What is needed then is an environment which supports the extreme notion of *plastic applications* which is not, as a system, domain specific and not restricted to specific developer chosen adaptations. EUD needs to be broadened beyond adaptation of existing environments. There have been attempts to do this in the EUD community [Burnett et al., 2001][Repenning et al., 2000] but such research has not gained widespread recognition in the software industry as a whole [Fischer, 2009]. More will be said on these tools and others in chapter 2.

Other researchers in the Human Computer Interaction (HCI) and software development communities have adopted terms such as "software plasticity" to describe the degree to which an interactive piece of software can adapt to changing contexts of use [Calvary et al., 2004][Morris, 2005][Sendn et al., 2005], although this is often in a developer focused context rather than that of an end-user. There is also now widespread use of technologies whose purpose is to increase flexibility to better deal with change, including Service-Oriented Architectures [Erl, 2005], Aspect-Oriented Programming [Kiczales et al., 1997] and Agile methodologies [Dingsyr et al., 2010], all of which have the potential to radically improve end-user application development but currently, for the most part, remain in the domain of professional software developers. There are a lot of technologies out there which could help enable plastic applications but that have yet to be used in this way.

One additional area of research which provides a background for the plastic ap-

plications concept is that of Empirical Modelling (EM) at the University of Warwick [EM Website][2]. The Empirical Modelling group have been exploring informal modelling tools and principles which enable users to develop models without needing to be expert programmers, or experts in the model's domain. A key aim of Empirical Modelling is to allow users to develop models informally without specification or design so that they may gain understanding of a problem they know little about prior to the modelling activity [Milner, 1986][Beynon and Russ, 2006]. EM is a conceptual framework for end-user development but rather than looking at ways of adapting traditional applications it has taken the more radical step of rethinking programming [Beynon et al., 2006a] and software from the ground up. The ideas and tools behind EM provide a great deal of insight into possible ways of implementing the EUD vision and the vision of supporting plastic applications. There are already several example EM models [EM Website], developed using existing tools [Ward, 2004, p.194], which illustrate the plastic app concept, including one model for timetabling which has been used as an application by staff at the University of Warwick [Beynon et al., 2000b]. The EM concepts and tools, however, have not yet been able to support the development of fully-fledged applications from models, in part because of the EM research objectives not being aligned with those of EUD but focused more on personal and individual modelling activities, and in part due to inadequacies of the tools.

1.2 Plastic Software Environments

A *plastic software environment* (PSE) is defined here to be, as the name suggests, an environment that hosts and enables the continuous development of one or more plastic applications. Such an environment may just be a form of modelling tool or virtual machine but could be an entire operating system hosting many different but connected plastic applications. The principles and concepts behind a PSE will be discussed further in section 2.1.1 which identifies key EUD principles and also towards the end of this

[2] An excellent introduction to EM can also be found in [Harfield, 2008, p.31]

work in 6.1. Some PSE characteristics, however, include:

- supporting the reversable refinement of an application.
- using a single conceptual model allowing gentle-slope development.
- not being brittle in the presence of the unexpected.
- being live and interactive for immediate feedback.
- not requiring initially undesirable precision and formality.

Whilst there are many examples of environments and applications that support end-user development to some degree, for example GIMP, Kate and Google-Mail (with their supporting technologies), there are few that would be considered a PSE. Examples of a PSE are hard to find but include spreadsheet environments, Forms/3 [Burnett et al., 2001] and AgentSheets [Repenning et al., 2000] along with the EM tool Eden [Ward, 2004, p.194]. With regards to spreadsheets, plastic applications would be models that have been sufficiently developed to become useful applications, perhaps for financial modelling or a teacher developed model for a school sports day. In these examples the model may well be used as an application but is always open to modification to adapt, extend or fix at any time.

Apart from perhaps spreadsheets, plastic software environments are far from ubiquitous. With the enormous success of the spreadsheet it seems most surprising that EUD environments have not become increasingly powerful and available even though, in general, software is now moving towards EUD. Perhaps the reason is the lack of research into the principles and conceptual frameworks for them?[3] The key players should be the operating systems developers and communities since a plastic operating system[4] would be a "dream come true".

[3] Of course it is also necessary for such software to be marketed.

[4] A plastic OS would be one with extreme end-user flexibility, beyond configuration files, scripts and utilities to a comprehensive modelling environment. There is some connection to SmallTalk and similar all inclusive environments.

1.3 A Lack of Plasticity

In order to achieve the level of flexibility (or plasticity) desired for plastic apps there is a need to look more closely at why existing systems are not capable of providing it, ignoring for the moment the environments developed by the EUD community (discussed in chapter 2). To aid this discussion let us use an analogy of developing a recipe for making a cup of tea. It is relatively easy to provide a generic recipe for making tea that is along the lines of: fill kettle with water, turn kettle on, get tea bags and mugs, pour boiling water from kettle into mugs, add tea bag to mug, remove tea bag, add milk and so on. Such a *program* could be parameterised by the developers to allow the user to control, for example, how long to leave a tea bag in the water, how much milk to use, how many mugs, how much water and so on. What if, however, the person making the tea was in an unfamiliar house and did not know where to find the tea or there is no kettle and so a sauce pan is needed instead?

One of the problems is that programs are based upon brittle assumptions. David King, in his thesis on "Parting Software and Program Design" [King, 2005], states that "the assumptions underlying programs are always brittle" (p.10) because "program descriptions must ... be both closed and complete" (p.14). Whilst "brittle assumptions [are] strong when true, but useless when false" (p.10) it is unfortunate that 'almost every 'interesting' system is incomplete" (p.14) meaning that these assumptions are likely to fail to hold. From the analogy, the *program* for making tea makes assumptions about knowing where to find tea bags and about the existence of a kettle. It is conceivable that the developers of the program foresaw this and provided the option of using a saucepan, but what if they did not? At this point, in a traditional application, the developers would need to alter the requirements specification, design and implementation of the program to allow for saucepans.

The intertwining of implementation and specification has been recognised for decades [Swartout and Balzer, 1982]. Lehman proposed the concept of software evolution back in 1980 [Lehman, 1980] (although the term was used prior to this), along with

a set of associated laws of software evolution which state that change is inevitable in certain kinds of applications and results in the need for an iterative approach to software development. One of the more significant laws, in my opinion, is that of feedback where the introduction of a piece of software will, by its very existence, change the nature of the problem [Lehman, 1996]. The consequence of this feedback is that it becomes impossible to predict in advance the requirements and design of an application [Fischer, 2009]. Since Lehman first studied the concept of software evolution there has been a great deal of research into software evolution, including empirical studies and the development of tools and techniques for dealing with evolving programs. The software industry has also recognised the problem of changing requirements, which has led to the development of technologies such as the recent Service-Oriented Architectures approach, Aspect-Oriented Programming and other kinds of Object-Oriented Programming, along with Agile methodologies to rapidly implement these changes. All of these new technologies and methodologies are there to reduce the difficulty and increase the speed with which changes can be implemented by developers. All of these technologies and methodologies are there to alleviate the problems associated with having brittle assumptions, the result of needing to use traditional programs.

This is where end-user programming comes in. Allowing end-users to make these kinds of change instead of having to go back to the developers is an efficient way of dealing with change. However, all this does is shift the burden from the developers to the end-users so there needs to be a fairly radical change to the way software is developed if this is to work or for there to be any real improvement. Shifting the burden to end-users has led to considerable research into ways of making programming easier for these non-professional developers, such as programming-by-example [Lieberman, 2001], visual programming [Resnick et al., 2009] and domain-specific languages [Mernik et al., 2005]. The problems of brittle assumptions and lack of flexibility still remain with EUP, even if the tools are now easier to use. Getting to the heart of the problem with programming involves more than simply getting the end-user involved and requires a more radical

rethink about the nature of programming itself. The EUP community have realised this and it is partly why EUP became End-User Development. The notion of evolution and experimentation came in and the whole development process, not just the programming, came under scrutiny.

The notion of software as being engineered is often misconceived by traditional computer science [Jackson, 2005]. Software engineering focuses on theory and formality through specifications of requirements using mathematical abstractions. Whilst for some applications this is an appropriate strategy, for many existing applications and for many that have yet to be conceived it is wholly inappropriate to think that they could be engineered in this way. Engineering practice is about the design and construction of an artefact [Rogers, 1983] which transforms the physical world, rather than focussing on the machine world. As Jackson puts it, requirements and specifications "are all to be understood in terms of physical phenomena rather than in terms of purely mathematical abstractions ... [any] abstractions must be firmly grounded in observable physical reality" [Jackson, 2005]. As Lehman's laws indicate, change is inevitable and potentially rapid in the software world. With software we are also dealing with the unknown where there is sometimes no pre-computer precedent and where the consequences cannot be foreseen. Such applications cannot be formally engineered but must be explored and evolved by experiment to first gain understanding, more akin to the real practice of engineering. Despite this realisation by many (especially in the EUD community) the idea of formal design and specification remains entrenched.

David King provides some insight into this when he proposes to abandon the relationship between software design and program design in order to gain a fuller appreciation of the problem [King, 2005, p.85]. King's very next statement, however, is that "for nearly all design methods, this is a step too far" (p.85) because "for software design to have any value, we must have some ability to translate between software and program design" (p.60). Perhaps it is this idea that we need traditional programs and that they need to be designed which is the problem? If the design and development

stages were fully intertwined so that the idea of needing to develop programs separately as an additional step were to be removed then the problem of brittle assumptions would also be removed. Applications evolve out of models without any separate translation into a program, this is what the concept of plastic applications is about. A suggestion made by King is that instead of attempting to reduce complexity there need to be tools which better enable us to deal with that complexity. Perhaps we need plastic software environments which appropriately manage complexity whilst being end-user friendly and supporting the experimental evolution of a model from the ground up until it becomes a plastic application, an application that is not made brittle by its translation into a traditional optimised program.

1.4 Thesis Aims

The higher aim behind this work is to answer the question of *how to bring extreme plasticity to software in a way amenable to everyday users*. One of the difficulties in bringing this about is the lack of suitable plastic software environments in which such an activity of moulding software can take place. There are many applications that provide possible examples of a plastic or semi-plastic environment but most are domain-restricted. One of the most promising but underdeveloped approaches is that of Empirical Modelling (EM). In addition to a lack of tool support for plastic applications there is also a lack of principles and of a conceptual framework for such an activity. Empirical Modelling may also provide a basis for such a framework. So my research question is:

> How to adapt, in specific and selective ways, Empirical Modelling concepts and tools to enable a migration from informal models to programs by end-users as an attempt at supporting the creation of plastic applications?

There are numerous problems with existing Empirical Modelling tools as well as some issues with its principles which prevent it from scaling up to be considered

a fully-fledged plastic software environment to be used in real situations. This work explores these issues and proposes possible solutions in the form of a new tool and re-conceptualisation of the principles. An additional aim that is being kept in mind (but not a key focus) throughout the work is the possibility of developing a plastic operating system, which will influence the design of the new tool to some degree. Specific objectives include:

- Critiquing of Empirical Modelling (and to a lesser degree EUD) to identify specific areas of improvement and ways that these improvements can be achieved by the development of a new tool.
- Developing a new prototype tool for Empirical Modelling and EUD which attempts to implement solutions to the specific identified problems.
- Creating models and examples within the new prototype environment to demonstrate it as an EM EUD tool and to demonstrate the new features it provides.
- Analysing the work to identify key principles and concepts behind the new environment to develop a framework which can help support the notion of plastic applications.

1.5 Thesis Outline

The thesis has been organised into 8 chapters which are as follows:

Chapter 1 is this introductory chapter where the motivations and aims are discussed along with definitions for *plastic applications* and *plastic software environments*.

Chapter 2 goes through recent work of the EUD community by looking at principles, guidelines and tools which they have developed and which provide useful material for directing this work. Empirical Modelling is also introduced in more detail to explain its key principles and current tools. Other relevant technologies such as programming languages and other software industry techniques are introduced throughout the thesis.

Chapter 3 explores specific problems with existing approaches in achieving the kind of plasticity being sought after. Primarily the focus is on Empirical Modelling (EM) tools and concepts, where a critique is given, as these are already a close match to a plastic software environment. The problems with EM are then critiqued with reference to the solutions from industry and EUD that appeared in the background material of chapter 2. At the end of the chapter specific research questions are posed and a form of specification is given for a new prototype environment called Cadence.

Chapter 4 forms a central part of this thesis. It documents the development of Cadence, proposed in chapter 3. The architecture and interfaces for Cadence are given, along with some examples of how Cadence is to be used.

Chapter 5 discusses various example models that were developed within Cadence in detail. These models include games, presentation environments and biological models. The examples show the flexibility and other characteristics of the tool which are relevant to both the problems identified in chapter 3 as well as the broader objectives of the thesis. Each model has been used to illustrate how the Cadence tool has resolved specific issues.

Chapter 6 identifies key principles and concepts from the prototype to develop an idealisation of Cadence. These principles have come from the work in chapters 3,4 and 5 and provide a possible framework for plastic software environments generally. It is this chapter that contains the second major contribution of this thesis.

Chapter 7 looks at the impact of the Cadence tool and the principles of chapter 6 upon Empirical Modelling. Hybrid tools are discussed which attempt to resolve limitations in both Cadence and existing EM tools, as well as provide a clear means of comparison. Example models are also included that were developed by students of Empirical Modelling who used either Cadence or a hybrid for their projects and coursework.

Chapter 8 summarises the work of all previous chapters and evaluates it with respect to the broader objectives identified in this chapter and the specific problems and questions posed in chapter 3. Further work is identified here, along with the contributions and limitations of the work contained within this thesis. A brief statement on applications

and future directions is also given.