

JamesKjx/ResearchProjects

Looking for ideas? I do my research as part of the Elvis software design research group. The best idea is to look at what we have been doing recently. In particular, look at our website, called `ElvisBrain` or read the papers and tech reports we have written recently.

Anyone planning to do honours research within Elvis should also plan on taking COMP462 Object Oriented Paradigms if it is being offered. Here is some old(ish) text about projects you could consider:

* * *

Safety Critical Java

Real-time systems are notoriously difficult to design and implement, and, as many real-time problems are safety-critical, their solutions must be reliable as well as efficient and correct. While higher-level programming models (such as the Real-Time Specification for Java) permit real-time programmers to use language features that most programmers take for granted (objects, type checking, dynamic dispatch, and memory safety) the compromises required for real-time execution, especially concerning memory allocation, can create as many problems as they solve. This project will look at new programming models for real-time systems. We have developed Scoped Types give programmers a clear model of their programs memory use, and, being statically checkable, prevent the run-time memory errors that bedevil models such as RTSJ. We use Aspects to build on Scoped Types guarantees so that Real-Time concerns can be completely separated from applications base code. Adopting the integrated Scoped Types and Aspects approach can significantly improve both the quality and performance of a real-time Java systems, resulting in simpler systems that are reliable, efficient, and correct.

We have funding from IBM to work on the next-generation Safety Critical Java systems.

* * *

Ownership Types for Aliasing in Object-Oriented Systems

Aliasing is endemic in object oriented programs. Because objects have identity, and can be uniformly referred to by essentially any other object, the dependencies between objects in a program can be arbitrary.

Interobject references are particularly important because they can undermine programmer's attempts to build encapsulated aggregate objects — objects outside an aggregate object can hold references — aliases — to objects inside the aggregate, breaking the encapsulation of the aggregate object. Although many object-oriented languages include some kind of protection for the names of object's slots, so an internal sub-object can be stored in a private slot, name protection is insufficient to protect objects — by error or by oversight, a programmer can write a public method which returns a value from a private slot.

Although they have long been ignored, the problems posed by aliasing are again being addressed within the computer science community, as illustrated

by papers, workshops and journal special issues dedicated to the topic.

Along with other researchers in America and Europe, I am developing a model for Ownership Types to address many of the problems caused by aliasing. Ownership Types separates the objects inside an aggregate into two categories — the aggregate’s representation, which should not be accessible outside the aggregate, and the aggregate’s arguments, which may be accessible outside, provided they are treated as immutable — similar to the distinction between aggregation and association in object-oriented design methodologies.

There are a number of possible projects within this topic, including implementing and evaluating new approaches to ownership types (either statically, using interpreters, or dynamically, using preprocessors and compilers), visualisation of aliasing structures within programs, mining design patterns for managing aliasing, and empirical studies of the amount and importance of aliasing with object-oriented programs. There are also a number of interesting theoretical problems in this area that I would be able to supervise in conjunction with more mathematically inclined members of staff.

A good place to begin is the 1998 paper “Flexible Alias Protection” that Google can find you.

There is also some software visualisation work on a similar theme.

* * *

Dynamic Software Updating

One of the problems facing developers is the constant evolution of components that are used to build applications. This evolution is typical of any multi-person or multi-site software project. How can we program in this environment? More precisely, how can language design address such evolution? In this project we will attack two significant issues that arise from constant component evolution. We have designed language-level extensions that permit multiple, co-existing versions of classes and the ability to dynamically upgrade from one version of a class to another, whilst still maintaining type safety guarantees and requiring only lightweight extensions to the runtime infrastructure. We show how our extensions, whilst intuitive, provide a great deal of power by giving a number of examples. Given the subtlety of the problem, we formalize a core fragment of our language and prove a number of important safety properties.

We have a design for a programming language, UpgradeJ?, that supports dynamic updating. This language has not been implemented. This project will complete detailed language design, implement the language, and then experiment and evaluate with the resulting implementation.

* * *

Agile Methods

Agile methods are the latest craze in software engineering. *Why write documentation when you could write software?* More seriously, agile methods have some clear advantages of structured methods — but most likely some serious disadvantages also. We’re studying actual Agile projects to find out how well they

work in practice: their strengths, their weaknesses, and to develop additional Agile practices to address gaps in existing Agile methodologies.

So far, we've worked on the Customer Role (with AngelaMartin), UI design (with JenniferFerreira) and Agile Project Management (with RashinaHoda), but we're open to investigating other aspects of Agile development in practice.

* * *

Software Corpus Analysis Methods (SCAM)

The LegoHypothesis is that software can be put together like Lego, out of lots of small interchangeable components.

We are attempting to conclusively disprove this hypothesis, by showing that component sizes follows a PowerLaw, and thus that there is no natural scale to software componentry. Bigger systems will have larger components than smaller ones.

See JavaCorpus or LegoHypothesis.

* * *

SoftwareSemiotics

Semiotics is the study of signs. Software Semiotics is the study of signs of software.

SoftwareSemiotics

* * *

Usage Centered Design for Usage Centered Design

Usage Centered Design is a new methodology for designing user interfaces in concert with more classical software development activities. Usage Centered Design constructs a number of interlinked models of users and their work, including user role models, role model maps, task models (essential use cases), use case maps, interaction context models, visual designs, as well as the standard models of object-oriented analysis and design. Managing the consistency and traceability of these models consumes a substantial amount of development effort which may be susceptible to automation. This project would attempt to apply usage centered design techniques to its own processes, resulting in a better description of the usage centered design processes and a prototype tool to support those processes.

Possible project outline

- survey of usage centered design practices
- usage centered design analysis of usage centered design
- tool design and implementation
- evaluation

Background

- An appreciation of issues and techniques in user interface design.

- Skill in rapid prototyping

Contacts:

- Larry Constantine, Constantine & Lockwood Ltd, and University of Technology Sydney (UTS).

Reading

- Software for Use, Larry Constantine & Lucy Lockwood. <http://ForUse.Com>

* * *

Survey of Mud programming languages

Since their first development by Richard Bartle (MUDDL), Lebling, Blank, and Anderson (ZORK) and so on, there have been a wide number of languages for programming (multi-user) adventure games. This project will survey a number of these languages, attempting to characterise their main features, object models, and computational power.

* * *

* * *

Other Areas

The projects above are the ones that are most worked out, and most suitable for Honours or Masters level work. I have been active in a number of other areas over the past five years or so, and would be able to supervise students in these areas also:

- object-oriented design patterns
- software visualisation, including visualisation for user interface design
- prototype-based object-oriented programming
- design and programming techniques for memory-limited systems
- component migration on the global scale
- concurrent object-oriented programming

* * *