Arquitecturas de Software

(Software Architectures)

Master of Engineering

Néstor Cataño

ncatano@puj.edu.co

Faculty of Engineering

Pontificia Universidad Javeriana

About the Lecturer

- Engineering degree in Comp.Sc., 1994-1999, Universidad del Valle, Cali.
- Software engineer, Open Systems, Cali, Colombia, 1999-2000.
- Master in Comp. Sc., University Paris 7, France, 2000-2001. Master thesis The JML's **`assignable`** Clause: Semantics, Verification and Application
- PhD. in Comp. Sc., University Paris 7 and INRIA, France, 2001-2004. PhD thesis Formal Methods for **Java** Programs.
- Research Associate, University of York, United Kingdom, 2004-2006. Working on Model-checking of distributed asynchronous systems.
- From 2007, Associate Professor, Pontificia Universidad Javeriana, Cali, Colombia.

Lecturer's Research Interests

- Formal specification and checking of **Java** applications
- Design and implementation of formal methods tools
- Mechanised correctness proofs of state-space reduction algorithms

This Course

- does cover the use of Automated Verification in Software Engineering
  - Verify means to check whether a program satisfies a specification
  - Programs and specifications are modelled separately
  - Models for representing programs and specifications depend on the particular formal technique used for verification

This Course

- does cover refinement calculus
  - From a formal specification, a correct program is found, while refinement calculus techniques are applied
  - Programs are correct by construction!!

Grading

1. Exam 1 (30/100)
2. Exam 2 (30/100)
3. Exam 3 (30/100)
4. Homework (10/100)

Course Plan

1. Introduction to Formal Methods
2. First Order Logic and Proof Systems
3. The PVS Specification and Verification System
4. The Java Modeling Language (JML)
5. The Refinement Calculus and Z

Course Plan

1. Introduction to Formal Methods
2. First Order Logic and Proof Systems
3. The PVS Specification and Verification System
4. The Java Modeling Language (JML)
5. The Refinement Calculus and Z

Introduction

- Software systems have become a key part of all our lives
  - Bank transaction systems
  - Web-based commerce applications
  - Smart Cards
- Software systems have become more important as they become more pervasive,
- and suppliers and users are both starting to worry about their correctness

Introduction

- Failures are unacceptable in today's software systems
- It is not feasible to shutdown a malfunctioning system in order to restore to a safe state
- Think of critical systems where human lives are involved

Introduction

Errors in hardware and software systems are often very costly

- Intel Pentium floating-point arithmetic bug costed $500M (1994)
- Ariane 5 rocket loss $7B (1996)
- Mars Polar Lander crash costed $120M (2000)

Ariane 5 Rocket

- It was launched on June the 4th 1996 and exploded 40 seconds after launched
- During the launch an exception occurred when a 64-bit floating point number was converted into a 16-bit signed integer
- This conversion was not protected by handling exception and caused the computer to fail
- The same error caused the backup computer to fail
- As a result incorrect data was transmitted which caused the destruction of the rocket

Software Engineering

- Software Engineering aims at being a disciplined approach to software development, but
- Software systems fail very often
- Possible causes
  - Methodology
  - Specification Language
  - Testing

Formal Methods

- Formal methods largely overcome this problem
- Formal methods are mathematically-based notations and techniques for specifying and analysing software and hardware systems
- Formal methods are based on
  - mathematics
  - logic, predicate calculus, set theory,
  - automata, graph theory

Formal Methods

- Traditional formal methods are model-checking and theorem-proving

Formal Methods

- Traditional formal methods are model-checking and theorem-proving
- Temporal-logic model-checking
  - Programs are modelled as transition systems
  - Properties are expressed in temporal logic
  - A model-checker conducts an exhaustive exploration of all states

Formal Methods

- Traditional formal methods are model-checking and theorem-proving
- Theorem proving
  - Programs and specifications are modelled in logic, i.e., first-order logic, higher-order logic
  - The underlying deduction system is used to prove whether the specification holds

Formal Methods Terminology

- Verification is used in different contexts
  - Process of obtaining the formal correctness proof of a system by using deduction (theorem proving)
  - Any action taken to find errors in a program by using automatic verification (model-checking)
- Software Testing
  - It's not a verification technique
  - It's much closer to sampling than to an exhaustive correctness proof

Formal Methods Limitations

- Verification methods do not guarantee the correctness of the actual code, but rather they verify some abstract model
- The correctness proof can itself be incorrect
- The verification process only captures a small part of the functionality of a system

Formal Methods Limitations

- The verification is done with respect to a given specification, which is formed manually, and is sometimes incomplete
- Automatic verification techniques (model-checking) are restricted to finite-state systems
- Automatic verification may fail to handle full fledged programs with real and integer variables and with pointer references

Formal Methods Limitations

- The verification is done with respect to a given specification, which is formed manually, and is sometimes incomplete
- Automatic verification techniques (model-checking) are restricted to finite-state systems
- Automatic verification may fail to handle full fledged programs with real and integer variables and with pointer references

Unlike model-checking, testing is not limited to finite-state systems. However, testing is not as comprehensive technique as model-checking

Formal Methods Successes

- Recently new tools have introduced that mechanise important parts of the proofs
- New automatic and semi-automatic verification techniques have demonstrated the scalability of automatic verification tools
  - Binary Decision Diagrams (BDDs)
  - Partial order reduction
  - Abstraction

Formal Methods Successes

- Formal specification techniques introduce an unambiguous description of the properties of a system. This can be used to discover errors in early stages of software development
- They reduce the risk of damage incurred by not finding an error in a system before deploying it

Formal Methods Prejudices

1. Formal methods can only be used by mathematicians

Formal Methods Prejudices

1. Formal methods can only be used by mathematicians
2. Using formal methods will slow down projects

Formal Methods Prejudices

1. Formal methods can only be used by mathematicians
2. Using formal methods will slow down projects
3. The verification process itself is prone to errors, so why bother at all?

Applying Formal Methods

- Formal methods can be used in various stages of the software development process, from early design to the acceptance tests of the completed product
- Ideally, the use of such methods is integrated into the development process through a methodology
- Certifying that the software has passed certain validity checks can be used as milestones for the development process

Applying Formal Methods

- It is difficult but not impossible to apply formal methods to a complete system. The tendance is to look for compositional methods
- The use of formal methods is not restricted to the code of software system. One may start by checking the requirements for possible contradictions

Formal Methods in Software Engineering: Requirements

- Mathematical models capture requirements in a unambiguous way
  - Easier to validate client's intentions
  - Sound basis for undertaking future development are provided
  - Feasible to use mathematical proof to show whether the intended system behaviour follows from specifications

Formal Methods in Software Engineering: Design

- Formal methods provide basis for validating design against requirements, e.g., in the form of a mathematical proof
- This increases engineers' confidence on that the software meets client's intention

Formal Methods in Software Engineering: Implementation

- Specifications can directly be ported into code (**B**, **Z**, **Circus**, **Alloy**)
- Tools that rely on refinement and correctness by construction principles exist

Formal Methods in Software Engineering

- A wide range of formal methods tools available exist
- Each tool uses its own specification language in general
  - one has to understand the theory behind the tool and the specification language used
- This makes industry reluctant to apply formal methods

Formal Methods in Software Engineering

- A wide range of formal methods tools available exist
- Each tool uses its own specification language in general
  - one has to understand the theory behind the tool and the specification language used
- This makes industry reluctant to apply formal methods

An interesting work on having a single specification language for a programming language is JML

The Java Modeling Language (JML)

- Formal specification language for **Java** programs
  - record design and implementation decisions
  - specify behaviour of classes
- JML specifications describe
  - interfaces (names and static information)
  - behaviour (how classes and interfaces act when used)

The Java Modeling Language

- JML specifications
  - Predicates in First-Order Logic
  - **Java** boolean expressions
  - Specification-purpose boolean JML constructs
- Specifications are added between `/*@ ... @*/` or after `//@` in **Java** code
  - preconditions
  - postconditions
  - invariants

Conclusion

- It is necessary to formally reason about the programs and tools that are constructed
- Formal methods provide support to traditional Software Engineering methods
- Formal methods are aimed at enhancing the quality of systems
- The benefits of using Formal methods pay off their limitations

Literature and Resources

- Z. Manna and A. Pnueli. The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer-Verlag, New York, 1991.
- Software Reliability Methods. Doron A. Peled. Springer.
- Model-Checking. Edmund M. Clarke, Jr., Orna Grumberg and Doron Peled.
- System and Software Verification: Model-Checking Techniques and Tools, Springer, M. Bidoit et al.
- Professor Edmund M. Clarke's web-page at Carnegie Mellon University `http://www.cs.cmu.edu/~emc/`