CIS 771: Software Specifications

Lecture 1: Course Overview

Copyright 2001, Matt Dwyer, John Hatcliff, and Rod Howell. The syllabus and all lectures for this course are copyrighted materials and may not be used in other course settings outside of Kansas State University in their current form or modified form without the express written permission of one of the copyright holders. During this course, students are prohibited from selling notes to or being paid for taking notes by any person or commercial firm without the express written permission of one of the copyright holders.

Software is ...

...one of the most complex man made artifacts

"I believe the [spreadsheet product] I'm working on now is far more complex than a 747 (jumbo jet airliner)"

-- Chris Peters (Microsoft, 1992)

"It's different [from other engineering disciplines] in that we take on novel tasks every time. The number of times [civil engineers] make mistakes is very small. And at first you think, what's wrong with us? It's because it's like we're building the first skyscraper every time."

-- Bill Gates (Microsoft, 1992)
Software is ...

...one of the most complex man made artifacts

Microsoft Word is ...1 million lines of code
Microsoft NT ...16 million lines of code

Even pacemakers have 100 thousand lines of code ...

...but perhaps software complexity shouldn't even be measured in terms of lines of code, but instead, in terms of number of states

States >> SLOC

- The size of a system is sometimes more accurately expressed using a semantic point of view
  - the number of different states a system can reach
  - ... an integer has 4.2 billion possible values
  - ... an object with 2 ints and a boolean field has 40 thousand quadrillion values
- How about Windows NT?
Software is…

...critical to the conduct of modern life.

- Process Control (oil, gas, water, ...)
- Transportation (air traffic control, ...)
- Health Care (patient monitoring, device control ...)
- Finance (automatic trading, bank security ...)
- Defense (intelligence, weapons control, ...)
- Manufacturing (precision milling, assembly, ...)

Failing software costs money and lives!

Failing Software Costs Money

- Thousands of dollars for each minute of factory down-time
- Huge losses of monetary and intellectual investment
  - Rocket boost failure (e.g., Arianne 5)
- Business failures associated with buggy software (Ashton-Tate dBase)
Failing Software Costs Lives

- Potential problems are obvious:
  - Software used to control nuclear power plants
  - Air-traffic control systems
  - Spacecraft launch vehicle control
  - ...
- A well-known and tragic example
  - Therac-25 radiation machine failures

Software is ...

... becoming the dominant component of society's infrastructure.

In the future...

- Everything will be monitored/controlled
  - networked watches, clothes, ...
  - autonomous vehicles, intelligent highways, ...
  - virtual X rather than physical X
- These systems may not have manual backup
  - no workarounds for y2k-like problems
- Failures will be very costly and dangerous
Priorities are changing ...

From: Bill Gates
Sent: Tuesday, January 15, 2002 2:22 PM
To: Microsoft and Subsidiaries: All FTE
Subject: Trustworthy computing

... Trustworthy Computing is computing that is as available, reliable and secure as electricity, water services and telephony.

... In the past, we’ve made our software and services more compelling for users by adding new features and functionality

... We’ve done a terrific job at that, but all those great features won’t matter unless customers trust our software. So now, when we face a choice between adding features and resolving security issues, we need to choose security.

... These principles should apply at every stage of the development cycle of every kind of software we create

... Bill

Software is ...

...what you'll be building after graduation.

- You’ll be developing systems in 2020+
  - in the context we just mentioned

- Given the importance of software
  - you may be regulated, licensed
  - you may be liable for errors
  - your job may depend on your ability to produce reliable systems
Software Development Cycle

- Requirements Analysis
- Design
- Code and Unit Test
- Subsystem Test
- System Test

Rigorous/Iterative Development

- Requirements
- Planning
- Design
- Implementation
- Evaluation
- Testing
- Deployment
Current Software Development Methods Are Insufficient

- Testing
  - samples execution behavior, misses some
- Systematic Inspections
  - don’t scale very well, although they are thorough
- Rigorous development processes
  - helping but most organizations don’t apply them

Formal methods are becoming more popular

A Formal Specification is...

The expression in some formal language and at some level of abstraction of a collection of properties that some system should satisfy [van Lamsweerde]

- Formal language
  - syntax can be mechanically checked for correctness
- Abstraction
  - above the level of source code (e.g., C is a formal language)
  - many useful levels of abstraction
A Formal Specification is...

The expression in some **formal language** and at some level of **abstraction** of a collection of **properties** that some system should **satisfy** [van Lamsweerde]

- Properties
  - often expressed in some logic, but not necessarily
  - must have a well-defined semantics to be useful
- Satisfaction
  - generally, should be able to tell if system satisfies the specification
  - ideally (but not usually), satisfaction is decided mechanically

Beware!

- The notion of “formality” is often misunderstood (formal vs. rigorous)
- The definition of “formal method” varies (see Jacky p. 5)
- The effectiveness of “formal methods” is highly debated
- The application of formal methods in industry is spotty
Why Use Formal Methods?

A number of reasons (see [van Lamsweerde])

- Forces you to think about issues in a systematic way
  - Leads to better design
  - Earlier detection of inconsistencies and flaws
- A precise reference to see if requirements are satisfied
- Gives direction to latter development phases (leading to coding)
- Provide a basis for reuse via specification matching
- Precise documentation within a team of developers
- ...

Formal Methods: Our Perspective

- Specifications should not be write-only
  - Too often, specs are not consulted after initial writing
- Specifications should be leveragable in multiple ways
  - At the very least, checked for syntactic well-formedness
  - Used to generate test cases
  - Used as input for specs coming later in the development
  - Semantically analyzable
    - If my system satisfies a spec, can a tool automatically infer that it satisfies some additional properties or invariants

We are mainly interested in “strong formal methods” -- formal methods with tool-supported semantic analysis
Iterative Artifact Development

Specification

Artifact/ Information

Evaluation

Possible Development Cycle

Requirements

Design

Code
In this course ...

- You will study 3 specification languages
  - **Alloy**: high-level semantic design
  - **UML/Statecharts/OCL**: code structure and simple behavioral properties
  - **ESC/Java**: a code-level spec language for specifying
    - Pre/post conditions, invariants, simple data constraints
- You will use tools for each language
  - to check syntax, simulate, and verify semantic properties
- You will apply each one of these language/tool approaches to a realistic software problem

Formal Methods in Cycle

Requirements

Alloy

Design

Alloy, UML/OCL

Code

ESC/Java
Alloy

- object model
  - describes set of configurations
  - each has a value for each set relation
- an odd instance
  - Daniel Jackson’s family

An instance of the above Alloy model

Alloy Textual Constraints

Syntax
- formulas
  - e1 in e2 subset
  - e1 = e2 equality
  - all v: S | F true when any atom in S substituted for v
- expressions
  - all exprs denote sets
  - e1 + e2 union
  - e1 & e2 intersection
  - e . r navigation
  - e . +r transitive closure

Example Constraints
- no incest
  - all p: Person | p.wife.mum != p.mum
- nobody’s her own (grand)mother
  - no p: Woman | p in p.+mum
- our family
  - some daniel, tim: Person |
    - daniel.mum = tim.mum
    - daniel.wife.mum = tim.wife.mum
UML, OCL, and USE

- Unified Modeling Language
  - Class diagrams – describe structure of object model
- Object Constraint Language
  - Enrich class diagrams – constrain field values and associations
- UML-based Specification Environment
  - Define object models
  - Generate model instances
  - Check for constraint violations

Class Diagram with Constraints

```
Student
public int ID;
public int gpa;

Faculty
public int ID;

0..* teaches 0..*

Undergraduate

Graduate

advisor/advisee
0..*

committee
0..*

1
3..5
```
**USE Specification**

```plaintext
model Academia

class Student attributes
  ID: Integer
  gpa: Integer
End

class Undergraduate < Student
class Graduate < Student

class Faculty attributes
  ID: Integer
End

association teaches between
  Student[*] Faculty[*]
End

association advisor between
  Graduate[*] Faculty[1]
End

association committee between
  Graduate[*] Faculty[3-5]
End

constraints

context Student
  inv gpaBound: self.gpa >= 0 & self.gpa <= 4

  inv uniqueID:
    Student.allInstances->forAll(
      s1, s2 | s1 != s1 implies s1.ID != s2.ID)

context Graduate
  inv advOnCommittee:
    self.committee->includesAll(
      self.advisor)
```

---

**Extended Static Checking**

- Specifications of behavior in terms of code
  - Java fields, methods, parameters
- Trades off expressiveness for automation
  - Halting is undecideable in general
  - Lots of properties can be checked in a reasonable amount of time
- Introduce specifications as ESC annotations into the code base
Summary

- Software is becoming pervasive and very complex
- Current development techniques are inadequate
- Formal methods are not a panacea, but we believe they will be necessary
- We will learn to use several different formal methods - each for different development stages
- We will emphasize strong formal methods because they are leveragable in multiple ways, and thus more likely to be used in practice.
Acknowledgements

• The quotes on slide 2 are from The Way of Z, by Jonathan Jacky, Cambridge University Press, 1997.
• Slides 17 and 18 showing Alloy examples are adapted from Daniel Jackson’s slides used in a talk given at Kansas State in Spring 2000.
• The JML examples are taken from the examples distributed with ESC/Java.