

# Secret Ninja Formal Methods

Joe Kiniry

School of Computer Science  
University College Dublin  
Dublin, Ireland  
[kiniry@ucd.ie](mailto:kiniry@ucd.ie)

Dan Zimmerman

Institute of Technology  
U. of Washington, Tacoma  
Tacoma, Washington, USA  
[\*dmzimm@u.washington.edu\*](mailto:dmzimm@u.washington.edu)

Formal Methods 2008, Turku, Finland, 29 May 2008

# Prelude

- ★ a ninja is silent and deadly, but we want you to hear us, so we will reveal our identities
- ★ many who have read the paper suggested that we give this talk in our *shinobi-shokozu* (ninja uniforms), so we are doing so
- ★ while we do want you to be entertained, we have some important points to convey

# Primary Lessons

- ★ do not give up on formal methods in your teaching: *they are not too difficult, impractical, or esoteric*
- ★ *rigorous process* coupled with *quality tools* help students learn subtle formal concepts
- ★ existence of *powerful, high-quality, automatic tools* is *mandatory* for adoption
- ★ *you must eat your own mochi*

# Reflecting on Arvind-san's Lessons

- ★ Arvind's main message was that FMs need to seem like they are not FMs, but we'd rather say that they should be *invisible*
- ★ ...and that they must be incorporated into *process*, with *no new notation*
- ★ our students know *English* and *Java*, so we use them for *informal* and *formal* specifications (all in a formal framework)

# Context

- ★ we have used variants of these tools and techniques at five universities, across three countries, and with freshmen to postgrads
- ★ communication and coordination is facilitated by Virtual Learning Environments (Moodle) and Collaborative Development Environments (GForge and Trac)
- ★ today we will focus on *strategy* and *weapons*

# Our Strategy

- ★ we use a “standard” process that is formal methods-rich, but does not appear formal
- ★ i.e., we use *stealth mathematics*
- ★ “standard” in the sense that we do what many textbooks *describe*, but very few developers (or professors) *actually do*
- ★ we *align learning with engineering* by coupling assessment with tool feedback

# Our Weapons

- ★ BONc, the BON compiler
- ★ CheckStyle and Metrics
- ★ FindBugs and PMD
- ★ the Common JML Tools
- ★ JUnit with JML
- ★ ESC/Java2
- ★ all integrated into Eclipse

# A Small Example

- ★ we will model the classic ninja weapon:  
the shuriken
- ★ of course, our courses focus on much  
larger, less dangerous, systems like  
simulators, smart card systems, and games

# Concept Analysis

★ agree upon the (domain) concepts

★ Weapon, Shuriken, Point, Velocity, Enemy

★ define each with a simple English statement

★ “a weapon in the form of a star with projecting points”

★ identify all *is-a* and *has-a* relations

★ Shuriken *is-a* Weapon

★ Shuriken *has-a* Point

# Describe Concepts

★ identify queries, commands, and constraints

★ Shuriken...

★ How many points do you have?

★ Let fly toward that enemy!

★ You must have at least three points.

# Capture Specs in BON

```
class_chart SHURIKEN
  inherit WEAPON
  indexing
    author: "Joe Kiniry and Dan Zimmerman"
  description
    "a weapon in the form of a star with \
    \projecting points"
  query
    "How many points do you have?"
  command
    "Let fly toward that enemy!"
  constraint
    "You must have at least three points."
end
```

# Refine Informal BON into Documented Types

```
/**
 * A weapon in the form of a star with
 * projecting points.
 * @author Joe Kiniry and Dan Zimmerman
 */
class Shuriken extends Weapon {
    /** How many points do you have? */
    /** Let fly toward that enemy! */
    /** You must have at least three points. */
}
```

# Introduce Signatures

```
/**
 * A weapon in the form of a star with
 * projecting points.
 * @author Joe Kiniry and Dan Zimmerman
 */
class Shuriken extends Weapon {
    /** How many points do you have? */
    byte points();

    /** Let fly toward that enemy! */
    void attack(Enemy the_enemy);

    /** You must have at least three points. */
}
```

# Specs

```
/**
 * A weapon in the form of a star with
 * projecting points.
 * @author Joe Kiniry and Dan Zimmerman
 */
class Shuriken extends Weapon {
    /** How many points do you have? */
    /**@ pure */ byte points();

    /** Let fly toward that enemy! */
    //@ ensures the_enemy.slain();
    void attack(/**@ non_null */ Enemy the_enemy);

    /** You must have at least three points. */
    //@ invariant 3 <= points();
}
```

# Implementation, Testing, and Verification

- ★ students implement their systems according to these specifications
- ★ static checkers provide feedback about code style, quality, and correctness
- ★ runtime checking and unit testing demonstrate system and unit correctness
- ★ a reminder: assessment is coupled to the results of the above analysis and execution

# Challenges

- ★ the weapons we *must use* are not always those we would *choose to use*, as there are limitations imposed by the local *daimyo* (e.g., our department head)
- ★ some of our samurai colleagues (i.e., fellow academics) do not understand or appreciate our techniques
- ★ it is not unusual to see a whole course in programming that never mentions documentation or assertions

# Epilogue

- ★ you should not fear us, please do ask questions and pose challenges
- ★ Ninja Dan has to disappear at the end of our conversation because his ninja skills are needed elsewhere
- ★ *all* the tools, pedagogical materials, student work over several years is freely available

# For More Information

★ Moodle — <http://csimoodle.ucd.ie/>

★ GForge — <http://sort.ucd.ie/>

★ Trac — <http://csi-trac.ucd.ie/>

★ Mobius PVE — <http://mobius.ucd.ie/>

Domo Arigato