Engineering Design w/Embedded Systems

Lecture 34—Verification & Validation; Formal Methods

Patrick Lam
University of Waterloo

April 5, 2013

Part I

Verification and Validation

Verification vs validation

Two similar terms. Not the same.

- **Verification**: “Is the project being built correctly?”
- **Validation**: “Will the project meet users’ needs?”

Need both!

Projects can fail because:

- meet users’ needs, but don’t work right; or
- bug-free but solve the wrong problem.

A successful project must pass both verification and validation.

More on Verification

Verification:

- assume a set of requirements; and
- establish that the product satisfies the requirements.

The requirements might be wrong. Not our problem!

How to verify: two options—

- testing (as seen previously);
- static analysis
  (computers exhaustively check code/design).

"Building the thing right."

More on Validation

Validation:
make sure that the requirements are the right ones.

(How does XP incorporate validation?)

Go beyond checking that code meets specifications;
work with customer to ensure specifications are correct.

"Building the right thing."

One way to validate code: *beta testing*.

- customers try the software and see if it's good.

Which first?

Verification, then validation.

Validating buggy software is frustrating.
Too much verification can be wasted work.

Part II

Formal Methods

About Formal Methods

Formal methods: techniques for **verification**:
make sure that code/designs conform to a specification.

Some formal methods techniques:

- static analysis;
- model checking.

Key Idea

To use formal methods, you need:

- a **model** of the artifact in question; and,
- a **property** that you would like to verify.

Often, the model is an abstract graph (ECE103!) representing system behaviours.
The property is usually a temporal logic formula.

Verification exhaustively searches model for violations.

Result of Verification

After the exhaustive search, either:

- the property definitely holds (on the model); or
- you get a counterexample.

With cleverness, we can search huge state spaces (10^{100} states).

Main insight: leveraging symmetries.

Case Study: Microsoft Static Driver Verifier

Why does Windows crash?

Usually, not the Windows kernel—fairly bombproof now. Instead: Windows drivers; run at same protection level as the kernel.

Scientists at Microsoft Research integrated existing and new techniques to verify drivers.

Windows Driver Kit includes the Static Driver Verifier; any “Certified for Windows” product must pass the SDV.

Formal Methods in Action

Recall: formal methods tools *exhaustively* explore all possible states of the system and driver.

SDV knows about all of the ways that the operating system can call the driver, and (symbolically) tries all possible combinations of calls.

Formal Methods: Discussion

In general, formal methods tools are still for experts (i.e. not you, at least today), who must give hints to tools.

Problems:

- need to wait too long for a result;
- can't verify something that is correct;
- false positives: warnings about problems that can never happen.

False positives occur when the model is too coarse and includes cases that can never happen in practice.

Formal methods particularly useful when the problem domain is too hard to reason about manually,
e.g. concurrency.