Lecture 18 — Design Validation

David J. Pearce

School of Engineering and Computer Science
Victoria University of Wellington

Coding Guidelines for Safety Critical Software

"The FDA has recognized that if product developers had tools that enable them to **examine and evaluate software earlier** in the development cycle, then there would be a greater likelihood that the resulting software would be more robust. Just as architects now have 3D modeling tools that allow them to take their clients on virtual tours of a new building before ground has been broken, the software engineering community has been developing tools for **modeling software** and its interactions with the system it controls. The **safety properties** of the model can be systematically examined, and once the model has been verified, the software derived from it can be proven to conform to the model. The result is software designs that are **far more robust** than those developed using traditional methods."

–FDA, Software Safety Research

What is Design Validation?

Formal Methods

"In computer science, specifically software engineering and hardware engineering, formal methods are a particular kind of **mathematically based** techniques for the **specification**, development and verification of **software and hardware systems**. The use of formal methods for software and hardware design is motivated by the expectation that, as in other engineering disciplines, performing appropriate mathematical analysis can contribute to the **reliability and robustness** of a design."

–Wikipedia

What is Program Specification?

"In computer science, a **program specification** *is the definition of what a computer program is expected to do. It can be* **informal***, in which case it can be considered as a blueprint or user manual from a developer point of view, or* **formal***, in which case it has a definite meaning defined in mathematical or programmatic terms."*

–Wikipedia

Why Specify Programs?

```
/**
 * Returns the maximum of the two input parameters
 */
int max(int x, int y) {
    if(x > y) {
        return x;
    } else {
        return y;
    }
}
```

- Specifications are helpful to **users**
- Specifications are helpful to **implementors**
- Can check implementations **meet their specifications**

Specification with Preconditions and Postconditions

```
// REQUIRES: items is not null
// REQUIRES: length of items greater than zero
//
// ENSURES: item returned greater-or-equal than all in i
// ENSURES: item returned matches one in items
int max(int[] items) {
   ...
}
```

- A **precondition** identifies valid input values
- A **postcondition** identifies valid output values

Exercise Time!

```
int indexOf(int[] items, int item) {
    int i = 0;
    //
    while(i < items.length) {
        if(items[i] == item) { return i; }
        i = i + 1;
    }
    //
    return -1;
}
```

- Give suitable **precondition(s)** and **postcondition(s)** for this function
- **HINT:** try to break pre/postconditions down into small pieces which address specific points, rather than just giving e.g. one giant postcondition

Specifications as Contracts

"A specification is a contract between two parties: the **client** *and* **supplier***. The client represents the person(s)* **using** *the given program, whilst the supplier is the person(s) who* **implemented** *it."*

- Supplier **must ensure** valid outputs, **assuming** valid inputs given
- Client **must ensure** valid inputs given to function
- Client can then **assume** valid outputs are returned

Meeting the Contract (Supplier)

```
// REQUIRES: items is not null
// REQUIRES: length of items greater than zero
// ENSURES: item returned greater-or-equal to all in items
// ENSURES: item returned matches one in items
int max(int[] items) {
    int r = items[0];
    for(int i = 1; i!=items.length; ++i) {
        if(items[i] > r) { r = items[i]; }
    }
    return r;
}
```

- This function **meets its specification**
- Given invalid inputs, it behaves **unexpectedly** (e.g. crashes)
- Q) Is this the **only** valid implementation of this function?

Meeting the Contract (Client)

```
// ENSURES: largest parameter value returned
int max(int x, int y) {
  int[] xy = new int[]{x,y};
  return max(xy);
}
```

- This function meets its specification. **Why?**
- **Hint:** if precondition for `max(int[])` holds ...

  ... then can assume its postcondition holds

Formal Specification

"In computer science, **formal specifications** *are mathematically based techniques whose purpose are to help with the implementation of systems and software."*

–Wikipedia

- Formal specifications are written in a **well-defined language**
- Therefore, can **precisely determine** if specification is met or not
- In contrast, English is **inherently ambiguous**, meaning ...

... we can **never** be sure if specification in English is met

Example of Formal Specification: @NonNull

```
class BoundedList {
  private @NonNull Object[] items;
  private int length;

  public BoundedList(int size) {
    items = new Object[size];
  }

  public void add(@NonNull Object x) {
    if(length < items.length;) {
      items[length++] = x;
  } }

  public @NonNull Object get(int i) {
    if(i < length) { return items[i]; }
    throw new ArrayIndexOutOfBoundsException();
} }
```

- Q) What are the **preconditions**? What are the **postconditions**?

Example of Formal Specification: Whiley

```
function max(int[] items) -> (int r)
// length of items greater than zero
requires |items| > 0
// item returned larger than any in items
ensures all { i in 0..|items| | items[i] <= r }
// item returned matches one in items
ensures some { i in 0..|items| | items[i] == r }:
    //
    int r = items[0]
    int i = 1
    while i < |items|:
        if items[i] < r:
            r = items[i]
        i = i + 1
    return r
```

- Q) how can we be **certain** implementation meets specification?

Over / Under Specification

```
// Check whether or not x contained in xs
function contains(int[] xs, int x) -> (bool r)
// If true returned, then x is contained in xs
ensures r ==> some { i in 0 .. |xs| | xs[i] == x }:
    //
    return false
```

- Q) does this function meet its specification?
- Under specification means specification **not strong enough**
- Over specification means specification is **too strong**

Non-Functional Specification

Specifications examined in this course are **functional**, meaning they relate a function's inputs to its outputs. Other aspects of program correctness we do not consider include:

- **Termination**. *Is the function guaranteed to terminate?* Without knowing this, unsure whether function can fail in infinite loop

- **Memory usage**. *How much memory does the function require?* Without knowing this, unsure whether function can fail with out-of-memory error

- **Worst-Case Execution Time (WCET)**. *How long will the function take to execute in the worst case?* Without knowing this, unsure whether timing constraints are met or not (if operating in **hard real-time environment**)