

**SE 2AA4, CS 2ME3 (Introduction to Software  
Development)**

**Winter 2017**

# **29 Introduction to Verification (Ch. 6)**

Dr. Spencer Smith

Faculty of Engineering, McMaster University

March 19, 2017



# Introduction to Verification

- Partially based on slides by Dr. Wassying, Ghezzi et al
- Administrative details
- `pointInRegion(p)`
- Outline of verification topics
- Testing so far in SFWR ENG 2AA4
- Need for verification
- Properties and approaches to verification
- Goals of testing
- Test plan
- Types of test - white box, versus black box, manual versus automated, etc.

# Administrative Details

- Investigating 9 academic integrity cases for A2
- A3 deadlines
  - ▶ Part 2 - Code: due 11:59 pm Mar 20
  - ▶ Part 1 spec available in repo
  - ▶ Change of  $<$  to  $\leq$  in natural language and spec
- A4
  - ▶ Your own design and specification
  - ▶ Due April 3 at 11:59 pm

# A Table for pointInRegion(p)

- Consider all of the cases
- Draw a picture
- Short form notation
  - ▶  $px = p.xcoord()$
  - ▶  $py = p.ycoord()$
  - ▶  $llx = lower\_left.xcoord()$
  - ▶  $lly = lower\_left.ycoord()$
  - ▶  $llxw = lower\_left.xcoord() + width$
  - ▶  $llyh = lower\_left.ycoord() + height$
  - ▶  $T = Constants.TOLERANCE$

# Nine Cases

		out
$px < llx$	$py < lly$	$p.\text{dist}(\text{PointT}(llx, lly)) \leq T$
	$lly \leq py \leq llyh$	$(llx - px) \leq T$
	$py > llyh$	$p.\text{dist}(\text{PointT}(llx, llyh)) \leq T$
$llx \leq px \leq llxw$	$py < lly$	$(lly - py) \leq T$
	$lly \leq py \leq llyh$	True
	$py > llyh$	$(py - llyh) \leq T$
$px > llxw$	$py < lly$	$p.\text{dist}(\text{PointT}(llxw, lly)) \leq T$
	$lly \leq py \leq llyh$	$(px - llxw) \leq T$
	$py > llyh$	$p.\text{dist}(\text{PointT}(llxw, llyh)) \leq T$

# Seven Cases

		out
$px < llx$	$py < lly$	$p.\text{dist}(\text{PointT}(llx, lly)) \leq T$
	$lly \leq py \leq llyh$	$(llx - px) \leq T$
	$py > llyh$	$p.\text{dist}(\text{PointT}(llx, llyh)) \leq T$
$llx \leq px \leq llxw$		$(lly - T) \leq py \leq (llyh + T)$
$px > llxw$	$py < lly$	$p.\text{dist}(\text{PointT}(llxw, lly)) \leq T$
	$lly \leq py \leq llyh$	$(px - llxw) \leq T$
	$py > llyh$	$p.\text{dist}(\text{PointT}(llxw, llyh)) \leq T$

# Six Cases

		out
$px < llx$	$py < lly$	$p.\text{dist}(\text{PointT}(llx, lly)) \leq T$
	$py > llyh$	$p.\text{dist}(\text{PointT}(llx, llyh)) \leq T$
$llx \leq px \leq llxw$		$(lly - T) \leq py \leq (llyh + T)$
$px > llxw$	$py < lly$	$p.\text{dist}(\text{PointT}(llxw, lly)) \leq T$
	$py > llyh$	$p.\text{dist}(\text{PointT}(llxw, llyh)) \leq T$
$lly \leq py \leq llyh$		$(llx - T) \leq px \leq (llxw + T)$

# Three Cases

	out
$llx \leq px \leq llxw$	$(lly - T) \leq py \leq (llyh + T)$
$lly \leq py \leq llyh$	$(llx - T) \leq px \leq (llxw + T)$
$\neg(llx \leq px \leq llxw) \wedge \neg(lly \leq py \leq llyh)$	$\min[p.\text{dist}(\text{PointT}(llx, lly)),$ $p.\text{dist}(\text{PointT}(llxw, lly)),$ $p.\text{dist}(\text{PointT}(llx, llyh)),$ $p.\text{dist}(\text{PointT}(llxw, llyh))] \leq$ $T$



# Nine Cases, but 2D

- How would you write all 9 cases, but with a tabular form that closely matches the original 2D problem description?

# Outline of Verification Topics

- What are the goals of verification?
- What are the main approaches to verification?
  - ▶ What kind of assurance do we get through testing?
  - ▶ Can testing prove correctness?
  - ▶ How can testing be done systematically?
  - ▶ How can we remove defects (debugging)?
- What are the main approaches to software analysis?
- Informal versus formal analysis

# Testing on Assignment 1 to 3

- Limited guidance on test case selection
- Maybe improved test cases would improve the results?
- Consider the method for deleting from a sequence of T (next slide)
- We have been using automated testing
- We have seen the advantages of regression testing
- Some have adopted the excellent strategy of testing while developing
  - ▶ Helps isolate errors
  - ▶ Does not leave testing to the end when there is no time to do it properly
  - ▶ Helps improve the understanding of the problem and the program
- Hopefully the experience on the assignments has motivated you to think more about testing

# Incorrect Version of Delete

Using `s = new T[MAX_SIZE]`, for some type `T`

```
public static void del(int i)
{
    int j;

    for (j = i; j <= (length - 1); j++)
    {
        s[j] = s[j+1];
    }

    length = length - 1;
}
```

- What is the error?
- What test case would highlight the error?

## Correct Version of Delete

```
public static void del(int i)
{
    int j;

    for (j = i; j < (length - 1); j++)
    {
        s[j] = s[j+1];
    }

    length = length - 1;
}
```

Avoids potential `ArrayIndexOutOfBoundsException` Exception

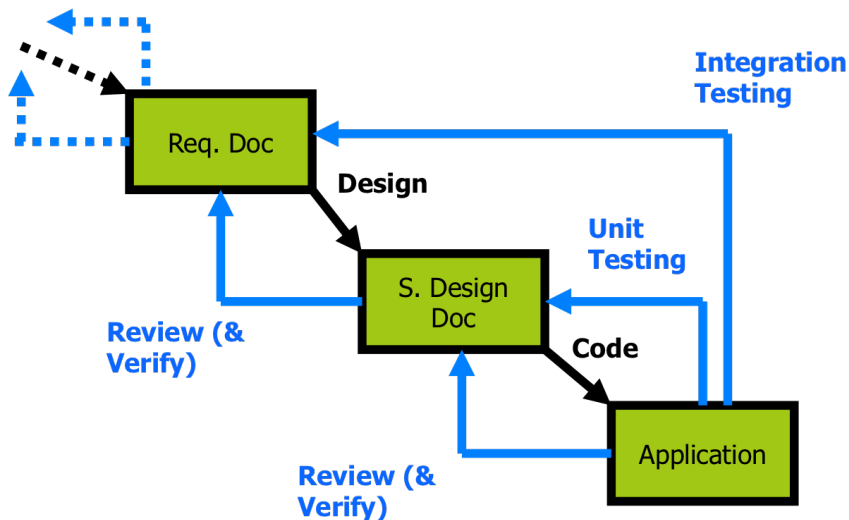
# Verification Versus Validation

- What is the difference between verification and validation?

# Verification Versus Validation

- Verification - Are we building the product right? Are we implementing the requirements correctly (internal)
- Validation - Are we building the right product? Are we getting the right requirements (external)
- According to **Capability Maturity Model (CMM)**
  - ▶ Software Verification: The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. [IEEE-STD-610]
  - ▶ Software Validation: The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements. [IEEE-STD-610]
- We will focus on verification

# Verification Activities





# Need for Verification

- Designers are fallible even if they are skilled and follow sound principles
- We need to build confidence in the software
- Everything must be verified, every required functionality, every required quality, every process, every product, every document
- For every work product covered in this class we have discussed its verification
- Even verification itself must be verified

# Properties of Verification

- May not be binary (OK, not OK)
  - ▶ Severity of defect is important
  - ▶ Some defects may be tolerated
  - ▶ Our goal is typically acceptable reliability, not correctness
- May be subjective or objective - for instance, usability, generic level of maintainability or portability
  - ▶ How might we make usability objective?
- Even implicit qualities should be verified
  - ▶ Because requirements are often incomplete
  - ▶ For instance robustness, maintainability
- What is better than implicitly specified qualities?

# Approaches to Verification

- What are some approaches to verification?
- How can we categorize these approaches?

# Approaches to Verification

- Experiment with behaviour of product
  - ▶ Sample behaviours via testing
  - ▶ Goal is to find “counter examples”
  - ▶ **Dynamic** technique
  - ▶ Examples: unit testing, integration testing, acceptance testing, white box testing, stress testing, etc.
- Analyze product to deduce its adequacy
  - ▶ Analytic study of properties
  - ▶ **Static** technique
  - ▶ Examples: Code walk-throughs, code inspections, correctness proof, etc.