Debugging an Operation
A Formal Methods Approach

Part 2 – A Standalone Operation
Recursive

```
void appendRV1 (QueueOfT& r, QueueOfT& g) {
//! updates r
//! clears g
//! ensures r = #r * #g
//! decreases |g|
   if (g.length() > 0) {
      T y;
      g.dequeue(y);
      r.enqueue(y);
      appendRV1(r, g);
   } // end if
} // appendRV1
```

X Places to Hunt for Defects

Example Operation: *appendRV1*

- Standalone operation, i.e., it is not a member of a class
- Uses recursion
- Makes calls to other operations

- Take a few moments to convince yourself this implementation meets its spec, i.e., is correct

X Places to Hunt for Defects

Assume:

- The operation's specs are correct
- But the operation fails under test

Claim about the debugging process:

- There is a systematic approach (based on design-by-contract ideas) that can be taken when searching for a defect

- This approach provides at least X locations to inspect when hunting for a defect

```
void appendRV1 (QueueOfT& r, QueueOfT& g) {
//! updates r
//! clears g
//! ensures r = #r * #g
//! decreases |g|
   if (g.length() > 0) {
      T y;
      g.dequeue(y);
      r.enqueue(y);
      appendRV1(r, g);
   } // end if
} // appendRV1
```

X Places to Hunt for Defects

- Work with your neighbor(s)
- Try to identify the *X* locations where defects can pop up
- *Important*: Each location should somehow be related to how the code is tied to the spec (or at least *supposed* to be tied to the spec)
- *Remember*: The specs of called operations are also involved

- Again, there are no defects in this implementation
- So don’t look for actual defects

```
void appendRV1 (QueueOfT& r, QueueOfT& g) {
//! updates r
//! clears g
//! ensures r = #r * #g
//! decreases |g|
   if (g.length() > 0) {
      T y;
      g.dequeue(y);
      r.enqueue(y);
      appendRV1(r, g);
   } // end if
} // appendRV1
```

4 Places to Hunt for Defects

Summary:

1. Blows a Precondition
2. Developer Misunderstands a Postcondition
3. Fails to Satisfy Own *ensures*
4. Fails to Make Progress to Base Case