Principles of Software Construction: Objects, Design, and Concurrency

Specifications

Christian Kästner Charlie Garrod

School of Computer Science
institute for SOFTWARE RESEARCH

Learning Goals

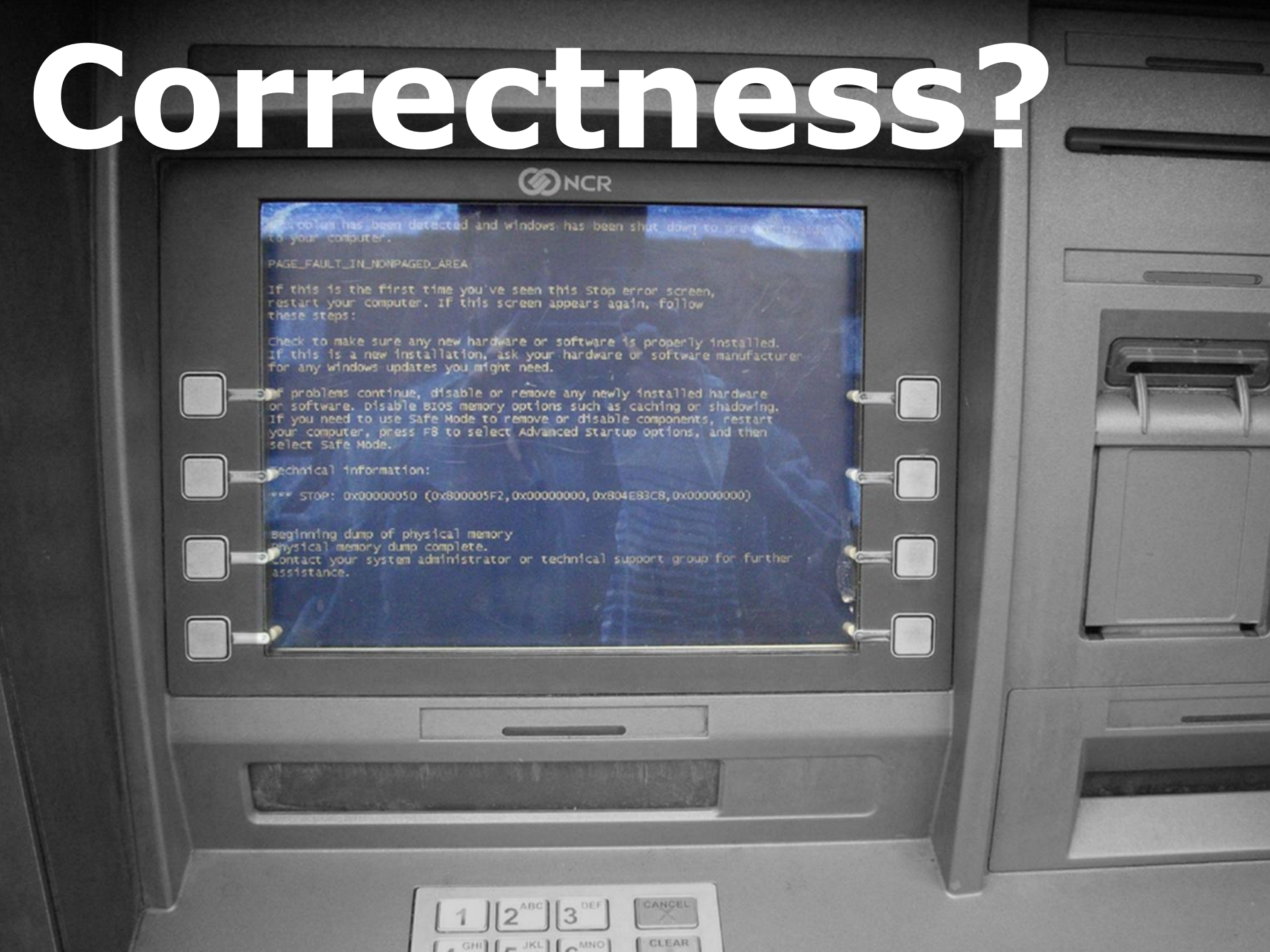
Understand the different forms of specifications

Select a suitable formality for a specification

Write specifications for functions and classes

The four course themes

- Threads and Concurrency
  - Concurrency is a crucial system abstraction
  - E.g., background computing while responding to users
  - Concurrency is necessary for performance
  - Multicore processors and distributed computing
  - Our focus: application-level concurrency
  - Cf. functional parallelism (150, 210) and systems concurrency (213)
- Object-oriented programming
  - For flexible designs and reusable code
  - A primary paradigm in industry – basis for modern frameworks
  - Focus on Java – used in industry, some upper-division courses
- Analysis and Modeling
  - Practical specification techniques and verification tools
  - Address challenges of threading, correct library usage, etc.
- Design
  - Proposing and evaluating alternatives
  - Modularity, information hiding, and planning for change
  - Patterns: well-known solutions to design problems


Correctness?
NCR
PAGE_FAULT_IN_NONPAGED_AREA
If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:
Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any windows updates you might need.
If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.
Technical information:
*** STOP: 0x00000050 (0x800005F2,0x00000000,0x804E83C8,0x00000000)
Beginning dump of physical memory
Physical memory dump complete.
Contact your system administrator or technical support group for further
assistance.
1
2 ABC
3 DEF
CANCEL
CLEAR

Software Errors

- Functional errors
- Performance errors
- Deadlock
- Race conditions
- Boundary errors
- Buffer overflow
- Integration errors
- Usability errors
- Robustness errors
- Load errors
- Design defects
- Versioning and configuration errors
- Hardware errors
- State management errors
- Metadata errors
- Error-handling errors
- User interface errors
- API usage errors
- …

Software Quality

Source: Braude, Bernstein, Software Engineering. Wiley 2011

- Sufficiency / Functional Correctness
  - Fails to implement the specifications … Satisfies all of the specifications
- Robustness
  - Will crash on any anomalous even … Recovers from all anomalous events
- Flexibility
  - Will have to be replaced entirely if specification changes … Easily adaptable to reasonable changes
- Reusability
  - Cannot be used in another application … Usable in all reasonably related applications without modification
- Efficiency
  - Fails to satisfy speed or data storage requirement … satisfies speed or data storage requirement with reasonable margin
- Scalability
  - Cannot be used as the basis of a larger version … is an outstanding basis…
- Security
  - Security not accounted for at all … No manner of breaching security is known

Software Quality

Source: Braude, Bernstein, Software Engineering. Wiley 2011

- Sufficiency / Functional Correctn
  - Fails to implement the specifications … Satisfies all o
- Robustness
  - Will crash on any anomalous even … Recovers from a
- Flexibility
  - Will have to be replaced entirely if specification chan reasonable changes
- Reusability
  - Cannot be used in another application … Usable in al applications without modification
- Efficiency
  - Fails to satisfy speed or data storage requirement … requirement with reasonable margin
- Scalability
  - Cannot be used as the basis of a larger version … is
- Security
  - Security not accounted for at all … No manner of bre

Our focus this and next week: Functional Correctness

Object-Design Challenges

(later in this course, and in 15-313)

Architectural Design Challenges

Additional Analyses (manual and automated)
=> 15-313 topics

Who's to blame?

```
Algorithms.shortestDistance(g,
    "Tom", "Anne");
```

```
> ArrayOutOfBoundsException
```

Who's to blame?

```
Algorithms.shortestDistance(g,
    "Tom", "Anne");

> -1
```

Who's to blame?

```
Algorithms.shortestDistance(g,
    "Tom", "Anne");

> 0
```

Who's to blame?

```
class Algorithms {
    /**
     * This method finds the
     * shortest distance between to
     * verticies. It returns -1 if
     * the two nodes are not
     * connected. */
    int shortestDistance(…) {…}
}
```

Who's to blame?

```
Math.sqrt(-5);
```

```
> 0
```

Who's to blame?

```
/**
  * Returns the correctly rounded positive square root of a
  * {@code double} value.
  * Special cases:
  * <ul><li>If the argument is NaN or less than zero, then the
  * result is NaN.
  * <li>If the argument is positive infinity, then the result
  * is positive infinity.
  * <li>If the argument is positive zero or negative zero, then
  * the result is the same as the argument.</ul>
  * Otherwise, the result is the {@code double} value closest to
  * the true mathematical square root of the argument value.
  *
  * @param a a value.
  * @return the positive square root of {@code a}.
  * If the argument is NaN or less than zero, the result is NaN.
  */

public static double sqrt(double a) { …}
```

Textual Specification

public int read(**byte**[] b, **int** off, **int** len) **throws** IOException

- Reads up to len bytes of data from the input stream into an array of bytes. An attempt is made to read as many as len bytes, but a smaller number may be read. The number of bytes actually read is returned as an integer. This method blocks until input data is available, end of file is detected, or an exception is thrown.
- If len is zero, then no bytes are read and 0 is returned; otherwise, there is an attempt to read at least one byte. If no byte is available because the stream is at end of file, the value -1 is returned; otherwise, at least one byte is read and stored into b.
- The first byte read is stored into element b[off], the next one into b[off+1], and so on. The number of bytes read is, at most, equal to len. Let *k* be the number of bytes actually read; these bytes will be stored in elements b[off] throughb[off+*k*-1], leaving elements b[off+*k*] through b[off+len-1] unaffected.
- In every case, elements b[0] through b[off] and elements b[off+len] through b[b.length-1] are unaffected.

- **Throws:**
  - IOException - If the first byte cannot be read for any reason other than end of file, or if the input stream has been closed, or if some other I/O error occurs.
  - NullPointerException - If b is null.
  - IndexOutOfBoundsException - If off is negative, len is negative, or len is greater than b.length - off

Specifications

- Contains
  - Functional behavior
  - Erroneous behavior
  - Quality attributes
- Desirable attributes
  - Complete
    - Does not leave out any desired behavior
  - Minimal
    - Does not require anything that the user does not care about
  - Unambiguous
    - Fully specifies what the system should do in every case the user cares about
  - Consistent
    - Does not have internal contradictions
  - Testable
    - Feasible to objectively evaluate
  - Correct
    - Represents what the end-user(s) need

Function Specifications

- A function's contract is a statement of the responsibilities of that function, and the responsibilities of the code that calls it.
  - Analogy: legal contracts
    - If you pay me $30,000
    - I will build a new room on your house
  - Helps to pinpoint responsibility
- Contract structure
  - Precondition: the condition the function relies on for correct operation
  - Postcondition: the condition the function establishes after correctly running
- (Functional) correctness with respect to the specification
  - If the client of a function fulfills the function's precondition, the function will execute to completion and when it terminates, the postcondition will be fulfilled
- What does the implementation have to fulfill if the client violates the precondition?

Formal Specifications

```
/*@ requires len >= 0 && array != null && array.length == len;
  @
  @ ensures \result ==
  @             (\sum int j;  0 <= j && j < len;  array[j]);
  @*/
int total(int array[], int len);
```

Advantage of formal specifications:

* runtime checks (almost) for free
* basis for formal verification
* assisting automatic analysis tools

JML (Java Modelling Language) as specifications language in Java (inside comments)

Runtime Checking of Specifications with Assertions

```
/*@ requires len >= 0 && array.length == len
   @ ensures \result ==
   @              (\sum int j;  0 <= j && j < len;  array[j])
   @*/
float sum(int array[], int len) {
      assert len >= 0;
      assert array.length == len;
      float sum = 0.0;
      int i = 0;
      while (i < len) {
            sum = sum + array[i]; i = i + 1;
      }
      return sum;
      assert ...;
}
```

java -ea Main

Runtime Checking with Exceptions

```
/*@ requires len >= 0 && array.length == len
   @ ensures \result ==
   @              (\sum int j;  0 <= j && j < len;  array[j])
   @*/
float sum(int array[], int len) {
     if (len < 0 || array.length != len)
          throw IllegalArgumentException(...);
     float sum = 0.0;
     int i = 0;
     while (i < len) {
          sum = sum + array[i]; i = i + 1;
     }
     return sum;
     assert ...;
}
```

Check arguments even when assertions are disabled.
Good for robust libraries!

Example Java I/O Library Specification (abridged)

public int **read**(byte[] b, int off, int len) throws IOException

- Reads up to len bytes of data from the input stream into an array of bytes. An attempt is made to read as many as len bytes, but a smaller number may be read. The number of bytes actually read is returned as an integer. This method blocks until input data is available, end of file is detected, or an exception is thrown.
- If len is zero, then no bytes are read and 0 is returned; otherwise, there is an attempt to read at least one byte. If no byte is available because the stream is at end of file, the value -1 is returned; otherwise, at least one byte is read and stored into b.
- The first byte read is stored into element b[off], the next one into b[off+1], and so on. The number of bytes read is, at most, equal to len. Let *k* be the number of bytes actually read; these bytes will be stored in elements b[off] throughb[off+*k*-1], leaving elements b[off+*k*] through b[off+len-1] unaffected.
- In every case, elements b[0] through b[off] and elements b[off+len] through b[b.length-1] are unaffected.

- **Throws:**
  - IOException - If the first byte cannot be read for any reason other than end of file, or if the input stream has been closed, or if some other I/O error occurs.
  - NullPointerException - If b is null.
  - IndexOutOfBoundsException - If off is negative, len is negative, or len is greater than b.length - off

Example Java I/O Library Specification (abridged)

public int **read**(byte[] b, int off, int len) throws IOException

- Reads up to len bytes of data f
  attempt is made to read as ma
  The number of bytes actually r
  until input data is available, er
- If len is zero, then no bytes ar
  attempt to read at least one by
  end of file, the value -1 is retu
  into b.
- The first byte read is stored int
  on. The number of bytes read
  bytes actually read; these byte
  1], leaving elements b[off+*k*]
- In every case, elements b[0] through b[off] and elements b[off+len] through b[b.length-1] are unaffected.

- **Specification of return**
- **Timing behavior (blocks)**
- **Case-by-case spec**
  - **len=0 → return 0**
  - **len>0 && eof → return -1**
  - **len>0 && !eof →return >0**
- **Exactly where the data is stored**
- **What parts of the array are not affected**

- **Throws:**
  - IOException - If the first byte
    or if the input stream has beer
  - NullPointerException - If b is n
  - IndexOutOfBoundsException -
    than b.length - off

- **Multiple error cases, each with a precondition**
- **Includes "runtime exceptions" not in throws clause**

Data Structure Invariants (*cf.* 122)

```
struct list {
        elem data;
        struct list* next;
};
struct queue {
        list front;
        list back;
};
```

Data Structure Invariants (*cf.* 122)

```
struct list {
        elem data;
        struct list* next;
};
struct queue {
        list front;
        list back;
};

bool is_queue(queue Q) {
        if (Q == NULL) return false;
        if (Q->front == NULL || Q->back == NULL) return false;
        return is_segment(Q->front, Q->back);
}
```

Data Structure Invariants (*cf.* 122)

```
struct list {
        elem data;
        struct list* next;
};
struct queue {
        list front;
        list back;
};

bool is_queue(queue Q) {
        if (Q == NULL) return false;
        if (Q->front == NULL || Q->back == NULL) return false;
        return is_segment(Q->front, Q->back);
}

void enq(queue Q, elem s)
//@requires is_queue(Q);
//@ensures is_queue(Q);
{
        list l = alloc(struct list);
        Q->back->data = s;
        Q->back->next = l;
        Q->back = l; }
```

Data Structure Invariants (*cf.* 122)

- Properties of the Data Structure
- Should always hold before and after method execution
- May be invalidated temporarily during method execution

```
void enq(queue Q, elem s)
//@requires is_queue(Q);
//@ensures is_queue(Q);
{ … }
```

Class Invariants

- Properties about the fields of an object
- Established by the constructor
- Should always hold before and after execution of public methods
- May be invalidated temporarily during method execution

Class Invariants

- Properties about the fields of an object
- Established by the constructor
- Should always hold before and after execution of public methods
- May be invalidated temporarily during method execution

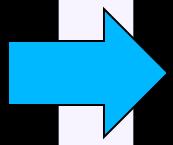
```
public class SimpleSet {

    int contents[];
    int size;

    //@ ensures sorted(contents);
    SimpleSet(int capacity) { … }

    //@ requires sorted(contents);
    //@ ensures sorted(contents);
    boolean add(int i) { … }

    //@ requires sorted(contents);
    //@ ensures sorted(contents);
    boolean contains(int i) { … }
}
```



```
public class SimpleSet {

    int contents[];
    int size;

    //@invariant sorted(contents);

    SimpleSet(int capacity) { … }

    boolean add(int i) { … }

    boolean contains(int i) { … }
}
```

Quality Attribute Specifications: Discussion

- How would you specify…
  - Availability?
  - Modifiability?
  - Performance?
  - Security?
  - Usability?

Much more on this in 15-313

Specifications in Practice

- Ideally formal pre- and post-conditions
- Textual specifications in practice
  - Best effort approach
- If any specification at all
- Specification especially necessary when reusing code and integrating code

- Writing specifications is good practice
- Writing fully formal specifications is often unrealistic