CSE503: SOFTWARE ENGINEERING

COMPLEXITY, PROVING ADTS

David Notkin
Spring 2011

But first… from today's Seattle Times
2011.5.4

"Industry experts believed they knew where to look for crack-inducing metal fatigue on aging airplanes, but the in-flight r… …hwest Airlines Boeing 737 on Friday ho… …s about part of the fuselage they pre… …wasn't vulnerable.

"A similar hole op… …hwest 737 only 21 months ago, and … …rican Airlines 757 last year, raising awareness that metal fatigue can cause the aluminum skin to separate at the so-called lap joints, where panels are spliced together."

Software complexity

Complexity

"Software entities are more complex for their size than perhaps any other human construct, because no two parts are alike (at least above the statement level). If they are, we make the two similar parts into one… In this respect software systems differ profoundly from computers, buildings, or automobiles, where repeated elements abound."
—Brooks

- From last lecture – complexity in the "why is it hard?" sense
- Today: "how complex" is a piece of software?

Complexity and people – Dijkstra

- "The competent programmer is fully aware of the limited size of his own skull."
- "Software is so complex that our poor head cannot cope with it at all. Therefore, we have to use all possible means and methods to try to control this complexity."

- First, some common software complexity measures
- Then, why they are weak measures and (perhaps) a way forward

Lines of code (LOC, KLOC, MLOC)

- Count the lines, often omitting comments and/or omitting blank lines
- Lines vs. statements
- Delivered vs. total (including tests, etc.)
- Productivity: LOC/person/time
  - I've seen published numbers ranging from ~2K-8K LOC/person/year

"I have made this letter longer than usual, because I lack the time to make it short." –Blaise Pascal

Halstead software science metrics

5

n1 = #distinct operators	**n2 = #distinct operands**	*n = n1 + n2* *"vocabulary"*
N1 = total # of operators	**N2 = total # of operands**	*N = N1 + N2* *"length"*

- **V = N × $\log_2$(n)** • *Volume is intended to capture the size of the implementation*
  - Making N choices from the vocabulary – assuming that humans do logarithmic search – leads to the formula
  - "The volume of a function should be at least 20 and at most 1000. The volume of a parameterless one-line function that is not empty; is about 20. A volume greater than 1000 tells that the function probably does too many things." [verifysoft.com]
- **D = (n1 / 2) × (N2 / n2)** • *Difficulty is proportional to the unique operators and the ratio of total operands to the number of operands*
  - The intent of the second part is based on a belief that repeated use of operands is more error-prone
- **E = V × D** • *Effort to implement or understand a program*
- …

Cyclomatic complexity (McCabe)

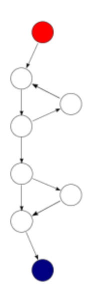
6

- Take the CFG and find the number of edges (**E**), number of nodes (**N**), and the number of connected components (**P**)
  - Connected components are subgraphs for which there is a path between any two vertices
- The cyclomatic complexity is **M = E − N + 2P** and is intended to measure the number of linearly independent paths through a program's source code
- #tests (branch coverage) ≤ **M** ≤ #tests (path coverage)
- Question: should the complexity include method dispatch in OOP?

Examples

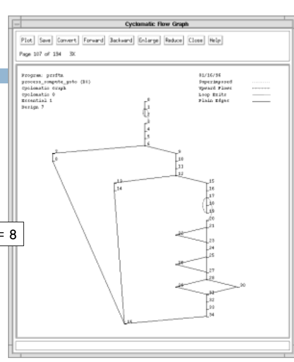
7

- E = 9
- N = 8
- P = 1
- M = 3



http://en.wikipedia.org/wiki/Cyclomatic_complexity

M = 8



http://hissa.nist.gov/HHRFdata/Artifacts/ITLdoc/235/chapter3.htm

Software structure metrics

Henry and Kafura

8

- Measures complexity in terms of fan-in and fan-out of procedures
  - *fan-in*: the number of local flows into a procedure plus the number of data structures accessed.
  - *fan-out*: the number of local flows out a procedure plus the number of data structures that the procedure modifies.
- Complexity is L^2 × FI × FO
  - Where L is the length of a procedure

And many more

- Variants of these
  - Some incremental improvements
  - Some extending to interprocedural complexity
- Others that measure
  - Coupling and cohesion
  - Data complexity
  - Data flow complexity
  - …
- Function points and feature points – intended to measure the function of a system as perceived by users, without reference to the implementation

So?

- Although there is somewhat mixed data, it appears that most of these measures are proportional to LOC
- "Les Hatton claimed recently (Keynote at TAIC-PART 2008, Windsor, UK, Sept 2008) that McCabe Cyclomatic Complexity has the same prediction ability as lines of code." –Wikipedia [cyclomatic complexity]
- Also, how "actionable" the information is has always confused me: if you are told your program is an "8" what are you supposed to do?

A hypothesis

- Every complexity measure I've seen is based entirely on the static program (except feature/function points, which don't consider a program directly)
- ***If*** complexity measures are to have any real utility, it seems that they must also consider the relationship between the program and its behaviors
  - That is, the way the developer associates behaviors with a program is material to complexity, but is ignored by the literature
- It is also imaginable that this measure would be "actionable" by identifying specific dependences that make this mapping complex – they could perhaps be addressed similarly to dependences that preclude parallelization

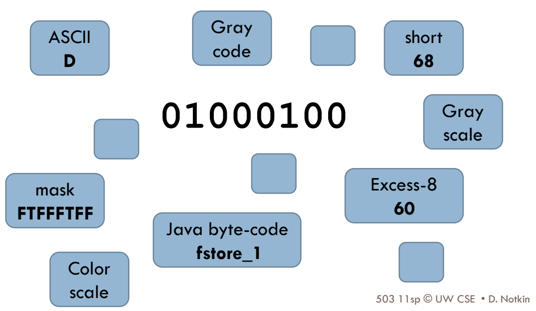
Project(s)?

- Any attempt at trying to make this notion more precise would be terrific
- Maybe a simple model and some empiric data
- Showing that a reasonable model is proportional to LOC would weaken my hypothesis

- Stop by and chat if you're interested
- Fits into NSF-funded work with Reid Holmes
  - ICSE 2011: "Identifying Program, Test, and Environmental Changes That Affect Behaviour"
  - Potential quals project

What is this?

13



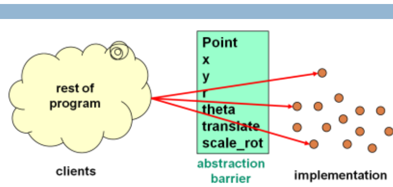
Types

14

- Without getting precise, types are used to interpret and manipulate the bit patterns – that is, they give them (some level of) meaning
- "Concrete" types manipulate the information in memory directly
- Abstract types define a protocol for manipulating instances of those types, but they do not define an implementation

Abstract data type = objects + operations

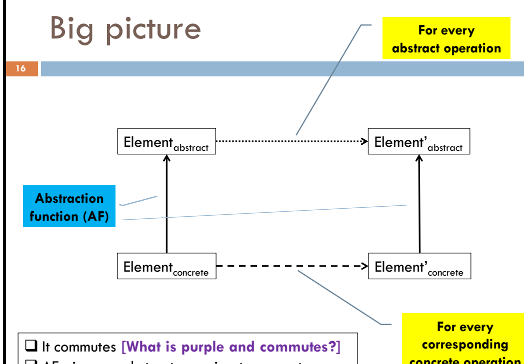
15



- The only operations on objects of the type are those provided by the abstraction
- The implementation is hidden
- We need to show that the abstraction and the implementation are each "correct" … and properly related

Big picture

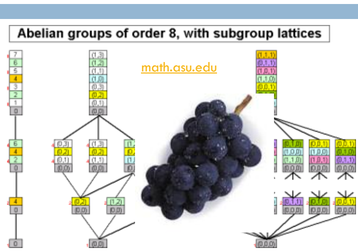
16



- It commutes [What is purple and commutes?]
- AF gives an abstract meaning to concrete representations – more soon

An Abelian grape (sorry)

17



Specifying ADTs

18

- A common way is to define the abstract effect of each operation (including constructors) using formal/informal pre- and post-conditions
- Might see this using an extended JavaDoc

Example

19

```
// Overview: An IntSet is a mutable, unbounded set of integers.
class IntSet {
  // effects: makes a new IntSet = {}
  public IntSet()

// returns: true if x ∈ this
//          else returns false
public boolean contains(int x)

// effects:  this_post = this_pre ∪ {x}
public void add(int x)

// effects:  this_post = this_pre - {x}
public void remove(int x)
…
```


Algebraic specifications

From Stotts (http://www.cs.unc.edu/~stotts/723/adt.html)

20

- Define a *sort* – give signatures of operations (you've seen this kind of thing before in typed OO and functional languages)

```
sort IntSet imports Int, Bool
signatures
    new : -> IntSet
    insert : IntSet × Int -> IntSet
    member : IntSet × Int -> Bool
    remove : IntSet × Int -> IntSet
```

Define axioms

- "Just" like high school algebra

```
variables i, j : Int; s : IntSet
axioms
    member(new(), i) = false
    member(insert(s, j), i) =
        if i = j then true else member(s, i)
    remove(new(), i) = new()
    remove(insert(s, j), i) =
        if i = j then remove(s, i)
                 else insert(remove(s, i), j)
```


Are these really sets?

- Posit stuff like…
  - `insert(insert(s, i), j) = insert(insert(s, j), i)`
  - `insert(insert(s, i), i) = insert(s, i)`
- Prove from axioms
- Tons of issues about completeness, consistency, equality (initial vs. final algebras), etc.
- But again, "just" like high school algebra

Proving specification properties

- Regardless of the style of specification, proofs are usually done inductively
- No information about the concrete representation and implementation – rather, showing the correctness of the protocol over the ADT's operations

LetterSet

case-insensitive character set [from Ernst]


```
// effects: creates an empty LetterSet
public LetterSet ( );

// effects: this_post =
//          if (∃ c_1 ∈ this_pre | toLowerCase(c_1) = toLowerCase(c)
//                    then this_pre else this_pre ∪ {c}
public void insert (char c);


// effects: this_post = this_pre – {c}
public void delete (char c);

// returns:  (c ∈ this)
public boolean member (char c);
```

Prove desirable property of `LetterSet`

Large enough `LetterSet` contains two distinct characters

25

Prove: $|S|>1 \Rightarrow (\exists c_1, c_2 \in S \mid [\text{toLowerCase}(c_1) \neq \text{toLowerCase}(c_2)])$

- Base case: S = ∅, vacuously true
- Inductive case: S was produced by a call of the form `T.insert(c)`
  - Assume: $|T|>1 \Rightarrow (\exists c_3, c_4 \in T\ [\text{toLowerCase}(c_3) \neq \text{toLowerCase}(c_4)])$
  - Show: $|S|>1 \Rightarrow (\exists c_1, c_2 \in S\ [\text{toLowerCase}(c_1) \neq \text{toLowerCase}(c_2)])$ where `S = T.insert(c)`
  - Remember `insert`'s post-condition:
    $\text{this}_{post} = \text{if } (\exists c_1 \in \text{this}_{pre} \mid \text{toLowerCase}(c_1) = \text{toLowerCase}(c) \text{ then } \text{this}_{pre} \text{ else } \text{this}_{pre} \cup \{c\}$
- For inductive case, consider the two possibilities for S
  - If S = T, the theorem holds by induction
  - If S = T ∪ {c}, there are three cases
    - $|T|=0$: Vacuously true
    - $|T|\geq 1$: T did not contain a char of toLowerCase(c), so the theorem holds by the meaning of union
    - $|T|>1$: By inductive assumption, T contains different letters, so by the meaning of union, T ∪ {c} also contains different letters

Now: Assume abstraction is correct

26

- Abstraction function **(AF):** $E_c \rightarrow E_a$
  - Maps a concrete object to an abstract value
  - Defines how the data structure is to be interpreted
  - Oh, that's a "D", that's an `fstore_1`, that's a 68, etc.
- Representation invariant **(RI):** a boolean predicate characterizing legal concrete representations
  - States data structure well-formedness
    - In essence, defines the domain of **AF**
  - Captures information that must be shared across implementations of multiple operations

CharSet Abstraction

A finite mutable set of Characters[From Ernst]

27

```
// Overview: A CharSet is a finite mutable set of Characters

// effects: creates a fresh, empty CharSet
public CharSet ( )

// effects: this_post = this_pre ∪ {c}
public void insert (Character c);

// effects: this_post = this_pre - {c}
public void delete (Character c);

// returns: (c ∈ this)
public boolean member (Character c);

// returns: cardinality of this
public int size ( );
```


A `CharSet` implementation

28

```
class CharSet {
  private List<Character> elts
       = new ArrayList<Character>();
  public void insert(Character c)   {
       elts.add(c);
  }
  public void delete(Character c)   {
       elts.remove(c);
  }
  public boolean member(Character c) {
       return elts.contains(c);
  }
  public int size() {
       return elts.size();
  }
}
```

```
CharSet s = new CharSet();
Character a
    = new Character('a');
s.insert(a);
s.insert(a);
s.delete(a);
if (s.member(a))
    // print "wrong";
else
    // print "right";
```

The RI can help identify an error

29

- Perhaps **`delete`** is wrong
  - It should remove all occurrences
- Perhaps **`insert`** is wrong
  - It should not insert a character that is already there

```
class CharSet {
  // Rep invariant: elts has no nulls and no
duplicates
  private List<Character> elts;
  …
```

- Or…
  - ∀ `indices i of elts . elts.elementAt(i) ≠ null`
  - ∀ `indices i, j of elts . i ≠ j ⇒ ¬elts.elementAt(i).equals(elts.elementAt(j))`

Where's the error?

30

```
// Rep invariant: elts has no nulls and no duplicates
public void insert(Character c) {
       elts.add(c);
}
public void delete(Character c) {
       elts.remove(c);
}
```


The RI constrains structure, not meaning

31

- Another implementation of `insert` that *preserves the RI*

```
public void insert(Character c) {
  Character cc = new Character(encrypt(c));
  if (!elts.contains(cc))
    elts.addElement(cc);
}
public boolean member(Character c) {
  return elts.contains(c);
}
```

- The program is wrong … call on the AF!

Abstraction function

concrete to abstract value mapping

32

- `AF(CharSet this) = { c | c is contained in this.elts }`
  - set of `Character`s represented by elements contained in **`this.elts`**
  - Typically not executable, but useful to reason about client behavior
- Helps reason about the semantics of `insert`
  `// effects:` $this_{post} = this_{pre} \cup \{c\}$
  `public void insert (Character c);`
- Helps identify a problem
  - Applying the AF to the result of the call to insert yields `AF(elts) ∪ {encrypt('a')}`
  - Consider the following reasonable AF
    - `AF(this) = { c | encrypt(c) is contained in this.elts }`
    - `AF(this) = { decrypt(c) | c is contained in this.elts }`

"Placing blame" using AF

33

- `AF(CharSet this) = { c | c is contained in this.elts }`
- Consider a call to insert:
  - On entry, the meaning is $\texttt{AF(this}_{pre}\texttt{)} \approx \texttt{elts}_{pre}$
  - On exit, the meaning is $\texttt{AF(this}_{post}\texttt{)} = \texttt{AF(this}_{pre}\texttt{)} \cup$ `{encrypt('a')}`
- Does this AF fix things?

AF(this) = { c | encrypt(c) is contained in this.elts }
= { decrypt(c) | c is contained in this.elts }

503 11sp © UW CSE • D. Notkin

Some final odds and ends

34

- Looking at these examples using the commutative diagram may help clarify any confusions
  - Or ask!
- AF's can be maintained across fairly complicated implementations that (for example) reorganize dynamically for performance
  - Multiple concrete values still map to the same abstract value
- Why map concrete to abstract?
  - It's not a function in the other direction
    - Ex: lists [a,b] and [b,a] each represent the set {a, b}
  - It's not as useful in the other direction

503 11sp © UW CSE • D. Notkin