Scenario-Driven Review of Designs

Greg Butler

Computer Science and Software Engineering
Concordia University, Montreal, Canada

Email: gregb@cs.concordia.ca

Recall Design Cycle

Design Cycle

Loop:

- Draft design and describe clearly
- Review/Assess/Evaluate design — discover issues
- Resolve design issue = make decision

Review Designs by "Executing" Scenarios

Provide a Concrete Situation

Attribute to Evaluate: Correct Working Design
Context: Pick an example system input
Context: Pick an example system state
Context: Pick an example scenario

Hand "Execute" the Designed System

"Execute" scenario
    trace behaviour of the system of the design
        eg, use sequence diagram for messages to objects
        and object interface definitions

Scenario-Driven Review of Correctness

Ideal Review

Trace all scenarios on all possible inputs!

Practical Review

For each scenario, trace one relevant input.

Remember — No amount of testing can demonstrate correctness!

There are too many combinations of valid input to test for a practical software system.
There are simply too many tests to run.

Design Assignment 1 — Count Substrings

Small Example

Alphabet $= \{$ a, c, g, t $\}$ String $S=$ ataaaa size $n = 6$

Substrings of size $k = 1$

Substrings

a

t

Counts

a : 5

t : 1

Total $= n$

Positions

a : 5 : 0,2,3,4,5

t : 1 : 1

Substrings of size $k = 2$

Substrings

aa

at

ta

Counts

aa : 3

at : 1

ta : 1

Total $= n - 1$

Positions

aa : 3 : 2,3,4

at : 1 : 0

ta : 1 : 1

Decide which "statistics" your system should compute!

Scenario-Driven Review of Correctness

Provide a Concrete Situation

Attribute to Evaluate: ...
Context: Pick an example system input
Context: Pick an example system state
Context: Pick an example scenario

Attribute

Correctness

Input

Alphabet = { a, c, g, t }
String S= ataaaa
size $n = 6$
size $k = 2$

State

$i = 3$
ss = aa
Counts
aa : 1
at : 1
ta : 1

Scenario

Count substrings

Hand "Execute" the Designed System

"Execute" scenario
trace behaviour of the system of the design
eg, use sequence diagram for messages to objects and object interface definitions

Scenario-Driven Review of Correctness

Hand "Execute" the Designed System

"Execute" scenario

trace behaviour of the system of the design

eg, use sequence diagram for messages to objects
and object interface definitions

Hand "Execute": How to use the design

Objects (O): What do responsibilities tell you?
(O) + Attributes (A): Can you update state of objects?
(O) + (A) + Method Signature (S): Can you update state?
(O)+(A)+(S) + Contract for methods: Can you update state?
(O)+(A)+(S) + pseudocode for methods: Can you update state?

So ... How detailed must your design be for a review?

Scenario-Driven Review of Correctness

State: Initial state

Alphabet = { a, c, g, t } String S= ataaaa $n = 6$ $k = 2$
i is undefined; ss is undefined; $Count$ = Bag{}

State: End of loop iteration 1

Alphabet = { a, c, g, t } String S= ataaaa $n = 6$ $k = 2$
$i = 0$; ss = at; $Count$ = Bag{ at:1 }

State: End of loop iteration 2

Alphabet = { a, c, g, t } String S= ataaaa $n = 6$ $k = 2$
$i = 1$; ss = ta; $Count$ = Bag{ at:1, ta:1 }

State: End of loop iteration 3

Alphabet = { a, c, g, t } String S= ataaaa $n = 6$ $k = 2$
$i = 2$; ss = aa; $Count$ = Bag{ aa:1, at:1, ta:1 }

Scenario-Driven Review of Correctness

State: End of loop iteration 4

Alphabet = { a, c, g, t } String S= ataaaa $n = 6$ $k = 2$
$i = 3$; ss = aa; $Count$ = Bag{ aa:2, at:1, ta:1 }

State: End of loop iteration 5

Alphabet = { a, c, g, t } String S= ataaaa $n = 6$ $k = 2$
$i = 4$; ss = aa; $Count$ = Bag{ aa:3, at:1, ta:1 }

State: Final state

Alphabet = { a, c, g, t } String S= ataaaa $n = 6$ $k = 2$
i is undefined; ss is undefined; $Count$ = Bag{ aa:3, at:1, ta:1 }

Proving Correctness

Review versus Proof

Review

Review aims to discover issues.
Review can use one scenario at a time.
Review can never consider all possible scenarios and inputs.

Proof

Proof must show design works in all possible cases.
Proof cannot do this using a small number of scenarios and inputs.
Proof must reason about all possible inputs and scenarios.

Proving Correctness

What changes in system state?

i loop index; *ss* the substring; *Count* the bag of substrings

What property of the state is invariant?

Inv: *Count* is the bag of substrings in the first i positions of the string S.

Prove that Inv is invariant

1. Is Inv true initially?

2. If Inv is true at the start of a loop iteration.
is Inv true at the end of the loop iteration?

3. Does the loop terminate in a finite number of steps?

4. If Inv is true at the end of the loop,
is the algorithm/design/system correct?

Proving Correctness

Check that all your assumptions are correct

Assumption: the design of *Count* is correct.

So you have shown that:
"Your design is correct, provided Count is a bag."

What property of a bag must you prove is correct
in your design of the class for *Count*?

Proving Correctness — Recap

Design is White-Box of Black-Boxes

Your algorithm is written to use data structures as black-boxes:
String, Substring, Collection, Integer

You assume black-boxes work correctly

... and show your design is correct

but you need to be clear on required properties of each black-box

which are specified in contract for the interface of the black-box

(and clear from the responsibility of the black-box)

...then design the black-boxes

and show they are correct, ie, have the required properties

Scenario-Driven Review of Resource Usage

Practical Review

Pick relevant system input, state, and scenario
relevant to usage of selected resource

Trace scenario on input, and estimate resource usage
ie, count cpu cyles or bytes used

Computation Time

Develop formula that counts the number of cpu cycles
in terms of size of input

Remember: Count time for data movement too

Memory Usage

For each data structure,
develop formula that counts the number of bytes
used by the data structure
in terms of size of input

Scenario-Driven Review of Scaleability

Scaleability

What happens to resource usage as input gets very large?

Definition of Scaleable

Growth in resource usage should be linear, or less than linear, than growth in size of input.

Definition of Scaleable

and ..... system should safeguard against overflow of usage of available resources

Practical Review

1. Review resource usage
2. Check formulas are linear or sublinear in size of input
3. Review all safeguards for overflow of usage of resources