Collaboration Among Software Components

Dick Hamlet

Department of Computer Science
Portland State University
Portland, OR 97207
hamlet@cs.pdx.edu

Portland State
UNIVERSITY

Summary of the Talk

Questions

- What can unit testing do in principle?
- For what kind of code does unit testing work?

Background

- Unit testing here and there
- *Software components* are nice units
- Component-based software development (CBSD)
- Subdomain testing tools to synthesize CB systems

Results

- Pitfalls of component testing in CBSD
- Design rules in aid of component testing
- A new development/testing scheme

Summary of the Talk

Questions

- What can unit testing do in principle?
- For what kind of code does unit testing work?

Background

- Unit testing here and there
- *Software components* are nice units
- Component-based software development (CBSD)
- Subdomain testing tools to synthesize CB systems

Results

- Pitfalls of component testing in CBSD
- Design rules in aid of component testing
- A new development/testing scheme

Summary of the Talk

Questions

- What can unit testing do in principle?
- For what kind of code does unit testing work?

Background

- Unit testing here and there
- *Software components* are nice units
- Component-based software development (CBSD)
- Subdomain testing tools to synthesize CB systems

Results

- Pitfalls of component testing in CBSD
- Design rules in aid of component testing
- A new development/testing scheme

Summary of the Talk

Questions

- What can unit testing do in principle?
- For what kind of code does unit testing work?

Background

- Unit testing here and there
- *Software components* are nice units
- Component-based software development (CBSD)
- Subdomain testing tools to synthesize CB systems

Results

- Pitfalls of component testing in CBSD
- Design rules in aid of component testing
- A new development/testing scheme

Summary of the Talk

Questions

- What can unit testing do in principle?
- For what kind of code does unit testing work?

Background

- Unit testing here and there
- *Software components* are nice units
- Component-based software development (CBSD)
- Subdomain testing tools to synthesize CB systems

Results

- Pitfalls of component testing in CBSD
- Design rules in aid of component testing
- A new development/testing scheme

Unit Testing Today

A haphazard activity directed at finding failures

Unit Testing Today

A haphazard activity directed at finding failures

Some of its problems:

- Units seldom have good specifications
- 'Coverage' metrics are weak surrogates
- Developers make lousy testers (too close to code)
- Independent testers are lousy (don't understand code)
- Stubs are impossible to devise

Unit Testing Today

A haphazard activity directed at finding failures

Some of its problems:

- Units seldom have good specifications
- 'Coverage' metrics are weak surrogates
- Developers make lousy testers (too close to code)
- Independent testers are lousy (don't understand code)
- Stubs are impossible to devise

Results are discarded because

- Not quantitative
- No connection with subsequent development
- Useless even for regression testing

Unit Testing Today

A haphazard activity directed at finding failures

Some of its problems:

- Units seldom have good specifications
- ‘Coverage’ metrics are weak surrogates
- Developers make lousy testers (too close to code)
- Independent testers are lousy (don’t understand code)
- Stubs are impossible to devise

Results are discarded because

- Not quantitative
- No connection with subsequent development
- Useless even for regression testing

But everyone hopes it will help...

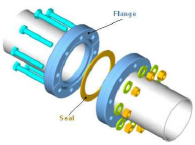
What Do the Other Engineers Do?

Designing a Vacuum System from Components

What Do the Other Engineers Do?

Designing a Vacuum System from Components

1. Choose components from catalog



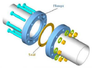
(http://us.trinos.com)

- Type CF flange, 304 stainless
- 50mm ID, 72.4mm bolt circle
- 8 M8 bolts
- Viton seal
- 10^{-8} Torr

What Do the Other Engineers Do?

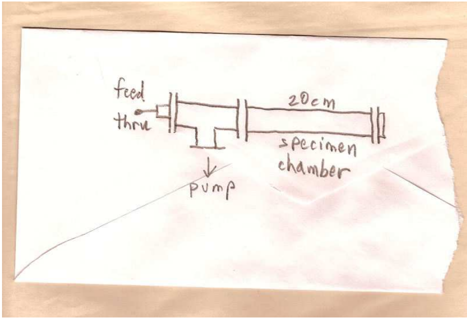
Designing a Vacuum System from Components

1. Choose components from catalog


(http://us.trinos.com)

- Type CF flange, 304 stainless
- 50mm ID, 72.4mm bolt circle
- 8 M8 bolts
- Viton seal
- 10^{-8} Torr

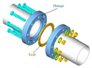
2. Sketch system using data 1



What Do the Other Engineers Do?

Designing a Vacuum System from Components

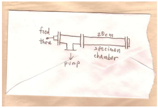
1. Choose components from catalog



(http://us.trinos.com)

- Type CF flange, 304 stainless
- 50mm ID, 72.4mm bolt circle
- 8 M8 bolts
- Viton seal
- 10^{-8} Torr

2. Sketch system using data 1

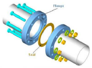


3. Calculate and check system properties
   - Volume (add component volumes)
   - Pressure loss (combine flange losses)
   - Required pump

What Do the Other Engineers Do?

Designing a Vacuum System from Components

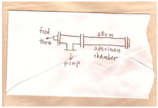
1. Choose components from catalog



(http://us.trinos.com)

- Type CF flange, 304 stainless
- 50mm ID, 72.4mm bolt circle
- 8 M8 bolts
- Viton seal
- 10^{-8} Torr

2. Sketch system using data ❶



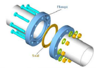
3. Calculate and check system properties
   - Volume (add component volumes)
   - Pressure loss (combine flange losses)
   - Required pump

4. Repeat ❶ – ❸ if needed
   - Oops! Need more volume, use 80mm ID
   - Oops again! Pressure loss too high, use OFHC copper seal
   - OK, system properties look good

What Do the Other Engineers Do?

Designing a Vacuum System from Components

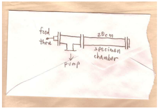
1. Choose components from catalog



(http://us.trinos.com)

- Type CF flange, 304 stainless
- 50mm ID, 72.4mm bolt circle
- 8 M8 bolts
- Viton seal
- 10^{-8} Torr

2. Sketch system using data ❶



3. Calculate and check system properties

- Volume (add component volumes)
- Pressure loss (combine flange losses)
- Required pump

4. Repeat ❶ – ❸ if needed

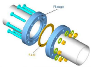
- Oops! Need more volume, use 80mm ID
- Oops again! Pressure loss too high, use OFHC copper seal
- OK, system properties look good

5. Assemble system (no surprises!)

What Do the Other Engineers Do?

Designing a Vacuum System from Components

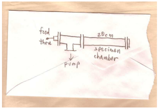
1. Choose components from catalog



(http://us.trinos.com)

- Type CF flange, 304 stainless
- 50mm ID, 72.4mm bolt circle
- 8 M8 bolts
- Viton seal
- 10^{-8} Torr

2. Sketch system using data ❶



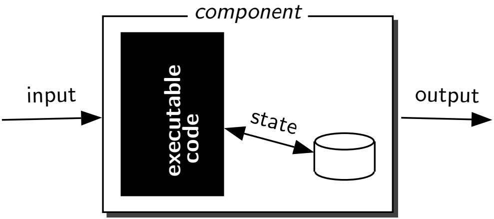
3. Calculate and check system properties
   - Volume (add component volumes)
   - Pressure loss (combine flange losses)
   - Required pump

4. Repeat ❶ – ❸ if needed
   - Oops! Need more volume, use 80mm ID
   - Oops again! Pressure loss too high, use OFHC copper seal
   - OK, system properties look good

5. Assemble system (no surprises!)

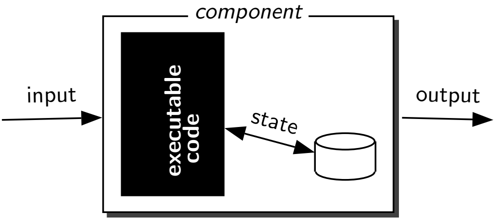
Would that it were so in software!

Software Components



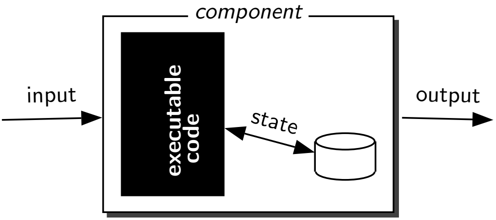
Software Components

- Executable code
- Interface
- Black-box behavior
- Local state



Software Components

- Executable code
- Interface
- Black-box behavior
- Local state



Why components?

- Reuse is better, cheaper (?)
- Precise software “units”
- Sidestep programming-language and design issues

Component-based Software Development (CBSD)

Components

- Specified, designed, implemented, tested *in isolation*
- Later to be used in systems *without modification*
- *component catalog* records data for later use

Component-based Software Development (CBSD)

Components

- Specified, designed, implemented, tested *in isolation*
- Later to be used in systems *without modification*
- *component catalog* records data for later use

Systems

- Assembled by matching components' interfaces
- Combination scheme is the system *architecture*
- In principle, design is done from the component catalog
- Assembled system tested against its specification

Component-based Software Development (CBSD)

Components

- Specified, designed, implemented, tested *in isolation*
- Later to be used in systems *without modification*
- *component catalog* records data for later use

Systems

- Assembled by matching components' interfaces
- Combination scheme is the system *architecture*
- In principle, design is done from the component catalog
- Assembled system tested against its specification

Ideal context for studying unit vs. system testing

Subdomain Testing Tools

Describe components and systems with configuration files

- Component description
  - Executable code file (any source language)
  - Subdomain decomposition of the domain
- System architecture
  - Flowgraph of component connections

Subdomain Testing Tools

Simplifying restrictions

Floating-point values on each execution:

1. One input value (read `STDIN`)
2. One output value (write `STDOUT`)
3. One non-functional value (run time) reported (write `STDERR`)
4. One state value read/written (disk file)

Subdomain Testing Tools

Simplifying restrictions

Floating-point values on each execution:

1. One input value (read `STDIN`)
2. One output value (write `STDOUT`)
3. One non-functional value (run time) reported (write `STDERR`)
4. One state value read/written (disk file)

Why so restricted?

- Simplify a complex situation to study it
- A small research group can implement powerful tools

A Component Description

A sawtooth with three ‘teeth’ modulated by an inverted parabola

saw.ccf

```
saw.bin
0 0.416666666666625 5
0.416666666666625 0.83333333333325 5
0.83333333333325 1.24999999999987 5
1.24999999999987 1.6666666666665 5
1.6666666666665 2.08333333333313 5
2.08333333333313 2.49999999999975 5
2.49999999999975 2.91666666666637 5
2.91666666666637 3.333333333333 5
3.333333333333 3.74999999999975 5
3.74999999999975 4.1666666666665 5
4.1666666666665 4.58333333333325 5
4.58333333333325 5 5
5 5.41666666666675 5
5.41666666666675 5.8333333333335 5
5.8333333333335 6.25000000000025 5
6.25000000000025 6.666666666667 5
6.666666666667 7.08333333333363 5
7.08333333333363 7.50000000000025 5
7.50000000000025 7.91666666666687 5
7.91666666666687 8.3333333333335 5
8.3333333333335 8.75000000000012 5
8.75000000000012 9.16666666666675 5
9.16666666666675 9.58333333333337 5
9.58333333333337 10 5
```

A Component Description

A sawtooth with three ‘teeth’ modulated by an inverted parabola

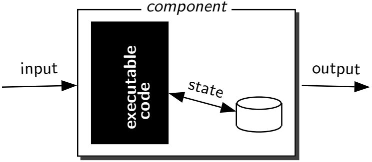
saw.bin

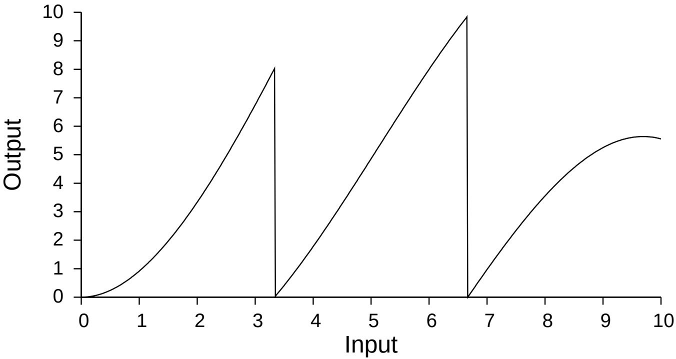
```
#!/usr/bin/perl -w
#
# executable  saw.bin
#
# sawtooth with parabolic envelope
$cycles = 3; #number of "teeth"
$interval = 10.0; #[0,10)
$env = 10.0;
$X = <STDIN>;  #read input
$Y = $X*$env*$cycles/$interval;
$Y -= int($env)*int($Y/$env);  #sawtooth
$Y *= 1.0 - (($X-6)**2)/(36);  #parabola
print "$Y\n";  #write output
print STDERR "1.0\n";  #constant 'run time'
```

saw.ccf

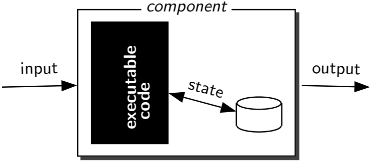
```
saw.bin
0 0.416666666666625 5
0.416666666666625 0.83333333333325 5
0.83333333333325 1.24999999999987 5
1.24999999999987 1.6666666666665 5
1.6666666666665 2.08333333333313 5
2.08333333333313 2.49999999999975 5
2.49999999999975 2.91666666666637 5
2.91666666666637 3.333333333333 5
3.333333333333 3.74999999999975 5
3.74999999999975 4.1666666666665 5
4.1666666666665 4.58333333333325 5
4.58333333333325 5 5
5 5.41666666666675 5
5.41666666666675 5.8333333333335 5
5.8333333333335 6.25000000000025 5
6.25000000000025 6.666666666667 5
6.666666666667 7.08333333333363 5
7.08333333333363 7.50000000000025 5
7.50000000000025 7.91666666666687 5
7.91666666666687 8.3333333333335 5
8.3333333333335 8.75000000000012 5
8.75000000000012 9.16666666666675 5
9.16666666666675 9.58333333333337 5
9.58333333333337 10 5
```

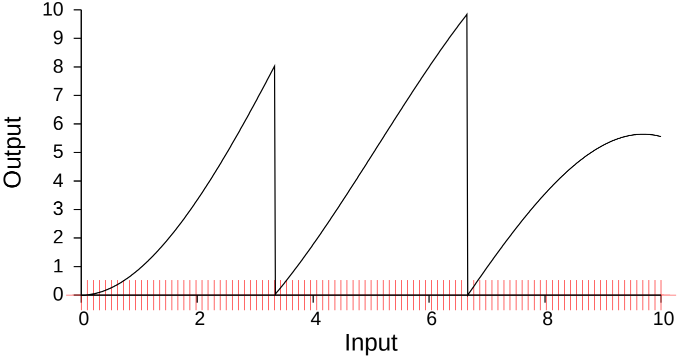
Subdomain Component Testing



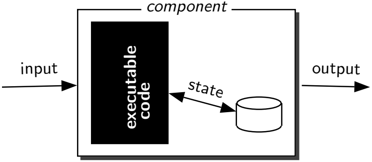


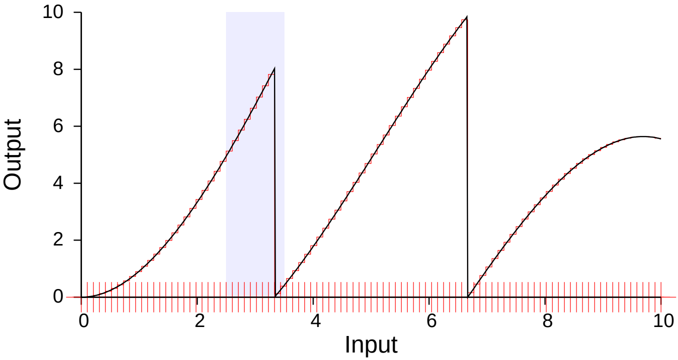
Subdomain Component Testing



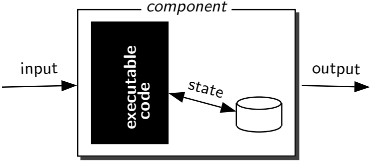


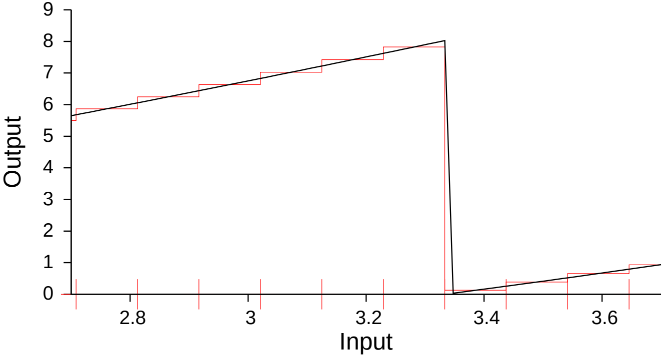
Subdomain Component Testing





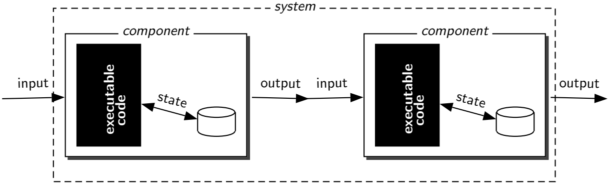
Subdomain Component Testing





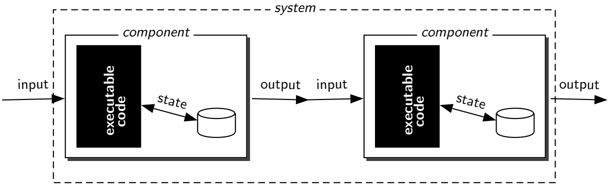
Predicting System Behavior

Two copies of the sawtooth in series



Predicting System Behavior

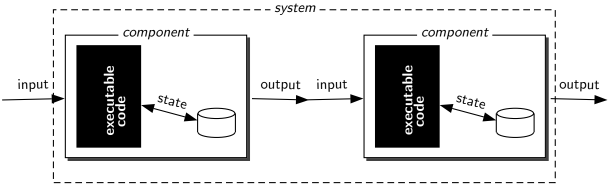
Two copies of the sawtooth in series




```
system.pscf
1 2 S
saw.ccf
saw.ccf
```

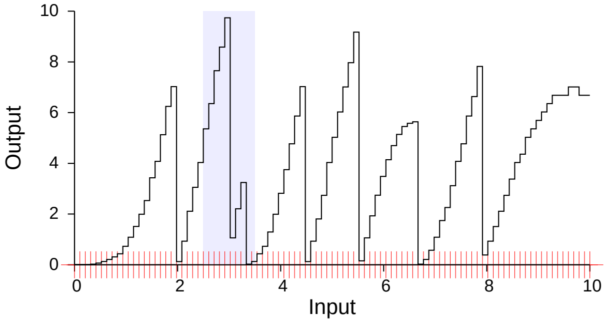
Predicting System Behavior

Two copies of the sawtooth in series



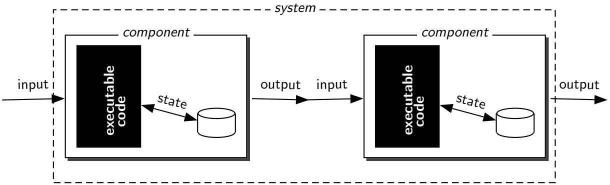
system.pscf

```
1 2 S
saw.ccf
saw.ccf
```



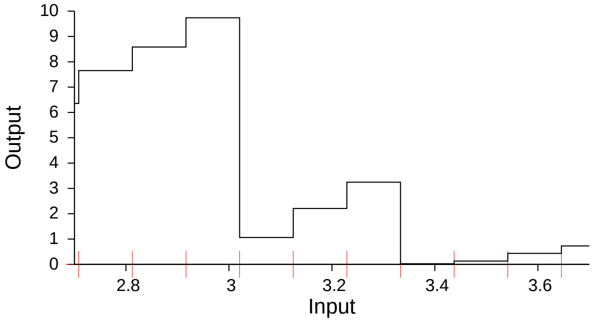
Predicting System Behavior

Two copies of the sawtooth in series



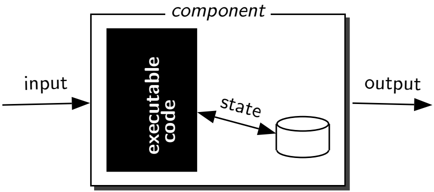
system.pscf

```
1 2 S
saw.ccf
saw.ccf
```



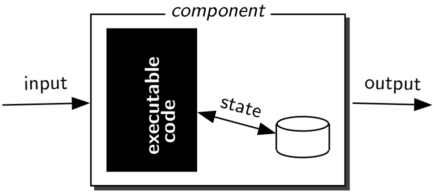
A Component with State

- On non-negative inputs 0..9, store count of longest sequence
- On negative inputs -10..-1, return stored sequence counts (for 9..0)



A Component with State

- On non-negative inputs 0..9, store count of longest sequence
- On negative inputs -10..-1, return stored sequence counts (for 9..0)

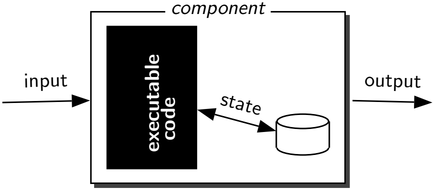


Example:

−1	1	0	−1	1	1	5	−2
0	0	0	1	0	0	0	2

A Component with State

- On non-negative inputs 0..9, store count of longest sequence
- On negative inputs -10..-1, return stored sequence counts (for 9..0)

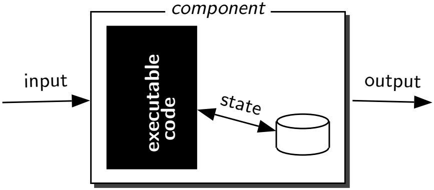


Example:	−1	1	0	−1	1	1	5	−2
	0	0	0	1	0	0	0	2

Implementation state: `5.11200010000`

A Component with State

- On non-negative inputs 0..9, store count of longest sequence
- On negative inputs -10..-1, return stored sequence counts (for 9..0)



Example:

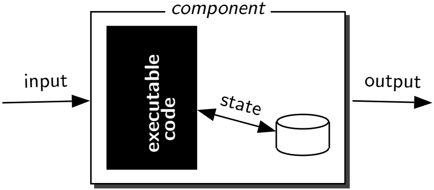
−1	1	0	−1	1	1	5	−2
0	0	0	1	0	0	0	2

Implementation state: `5.11200010000`

current digit *current count*

A Component with State

- On non-negative inputs 0..9, store count of longest sequence
- On negative inputs -10..-1, return stored sequence counts (for 9..0)



Example:

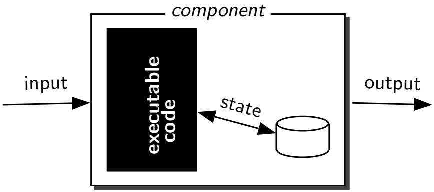
−1	1	0	−1	1	1	5	−2
0	0	0	1	0	0	0	2

Implementation state: `5.11200010000`

maximum zero sequence *maximum one sequence*

A Component with State

- On non-negative inputs 0..9, store count of longest sequence
- On negative inputs -10..-1, return stored sequence counts (for 9..0)



Example:

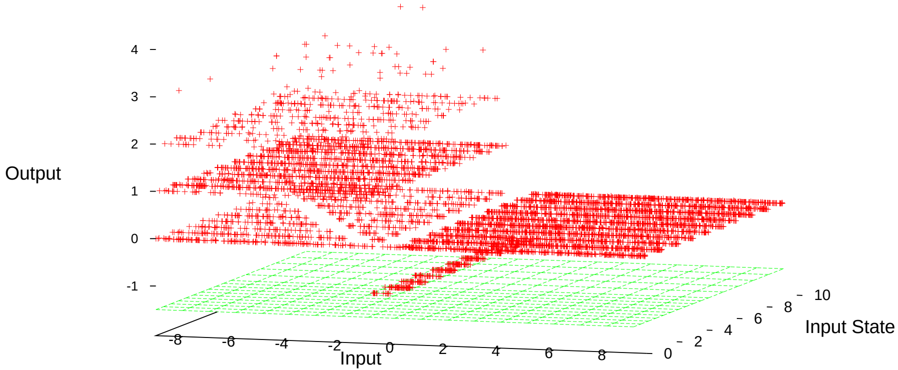
−1	1	0	−1	1	1	5	−2
0	0	0	1	0	0	0	2

Implementation state: `5.11200010000`

maximum five sequence

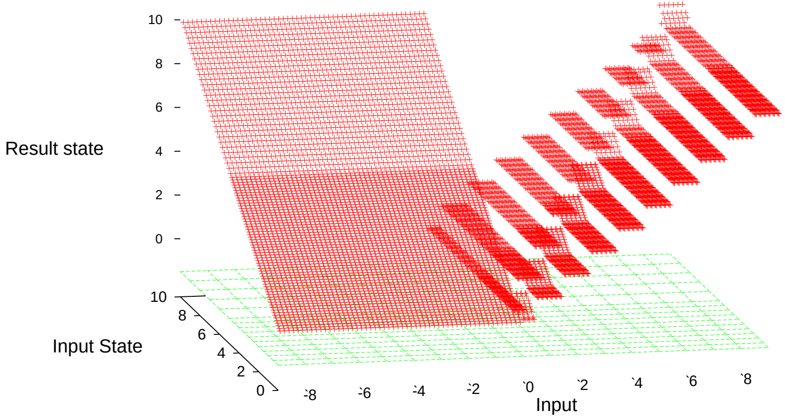
Component Output Behavior

Random-length sequences of random test points:



Component Result-state Behavior

Systematic coverage of (input $\times$ state) pairs:



Summary of the Talk

Questions

- What can unit testing do in principle?
- For what kind of code does unit testing work?

Background

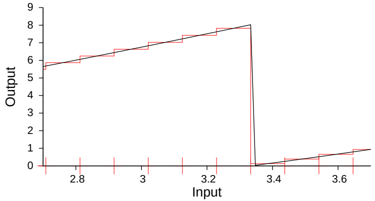
- Unit testing here and there
- *Software components* are nice units
- Component-based software development (CBSD)
- Subdomain testing tools to synthesize CB systems

Results

- Pitfalls of component testing in CBSD
- Design rules in aid of component testing
- A new development/testing scheme

Approximation Errors

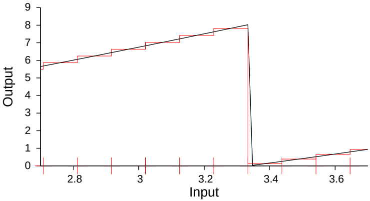
- Component measurement:



subdomain	*r-m-s error*
$[2.81, 2.92)$	2.93
$[2.92, 3.02)$	2.97
$[3.02, 3.12)$	3.01
$[3.12, 3.23)$	3.05
$[3.23, 3.33)$	3.08
$[3.33, 3.44)$	1.95
$[3.44, 3.54)$	2.02

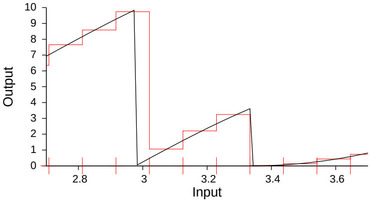
Approximation Errors

- Component measurement:



subdomain	*r-m-s error*
$[2.81, 2.92)$	2.93
$[2.92, 3.02)$	2.97
$[3.02, 3.12)$	3.01
$[3.12, 3.23)$	3.05
$[3.23, 3.33)$	3.08
$[3.33, 3.44)$	1.95
$[3.44, 3.54)$	2.02

- System prediction:



subdomain	*r-m-s error*
$[2.81, 2.92)$	7.00
$[2.92, 3.02)$	125.17
$[3.02, 3.12)$	7.30
$[3.12, 3.23)$	7.30
$[3.23, 3.33)$	6.97
$[3.33, 3.44)$	0.40
$[3.44, 3.54)$	1.19

Spurious-state Sampling

How to test components/systems with state?

Spurious-state Sampling

How to test components/systems with state?

- For $i = 1, 2, 3, ..., N$:
  - Externally set state S_i from specification
  - Choose input X_i
  - Execute on point (X_i, S_i)
  - Compare resulting state and output with specification

Spurious-state Sampling

How to test components/systems with state?

- For $i = 1, 2, 3, ..., N$:
  - Externally set state S_i from specification
  - Choose input X_i
  - Execute on point (X_i, S_i)
  - Compare resulting state and output with specification

Spurious-state Sampling

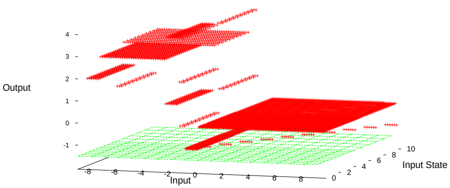
How to test components/systems with state?

- For $i = 1, 2, 3, ..., N$:
  - Externally set state S_i from specification
  - Choose input X_i
  - Execute on point (X_i, S_i)
  - Compare resulting state and output with specification

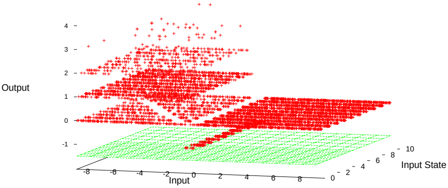
- Execute on input X_0 to initialize ('reset') state to S_0
- Check S_0 and output against specification
- For $i = 1, 2, 3, ..., N$:
  - Choose input X_i
  - Execute on input X_i (state is S_{i-1})
  - Compare result state S_i and output with specification

State Sampling vs. Input Sequences

Systematic state sampling

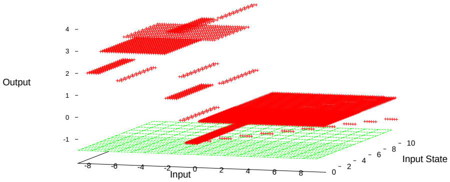


Random input sequences

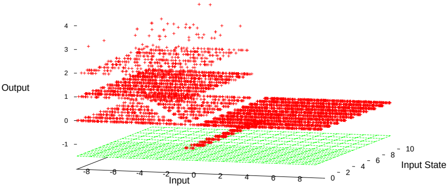


State Sampling vs. Input Sequences

Systematic state sampling



Random input sequences



Sampling infeasible states:

- Wastes scarce testing time
- Distorts the real behavior
- Hides unexpected real states
- Worst case: specified states are infeasible

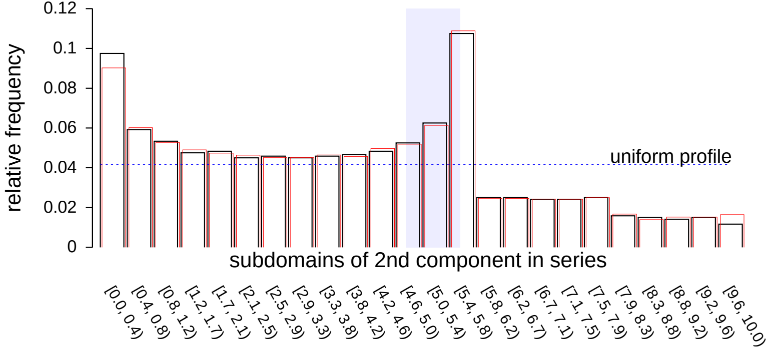
The Internal Profile Problem

Each component distorts the profile it receives

The Internal Profile Problem

Each component distorts the profile it receives

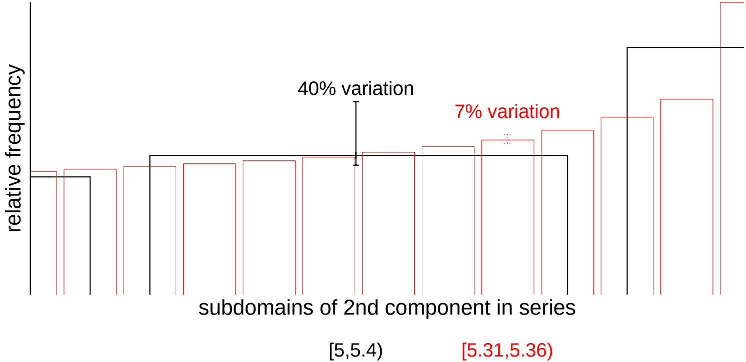
For the two sawtooth components in series:



The Internal Profile Problem

Each component distorts the profile it receives

For the two sawtooth components in series:



The same thing happens within a subdomain

Design Rules

Design Rule 1

Check calculated system profiles against component test profiles

Derived Rule 1-1

Don't use a general-purpose component for a few specific values

Design Rule 4

Group state values within as few components as possible – don't create cross-product states

Derived Rule 4-1

Group all 'modes' (preferences) in a control component; test all combinations

A New Component-based Development Scheme

1. Develop and test components $\rightarrow$ quantitative descriptions (or get quantitative descriptions from component catalog)

A New Component-based Development Scheme

1. Develop and test components → quantitative descriptions (or get quantitative descriptions from component catalog)
2. Design system using component descriptions from ❶

A New Component-based Development Scheme

1. Develop and test components → quantitative descriptions (or get quantitative descriptions from component catalog)
2. Design system using component descriptions from 1
3. Synthesize system using CAD tools

A New Component-based Development Scheme

1. Develop and test components → quantitative descriptions (or get quantitative descriptions from component catalog)
2. Design system using component descriptions from ❶
3. Synthesize system using CAD tools
4. Compare internal profiles predicted in ❸ with those used in ❶

A New Component-based Development Scheme

1. Develop and test components → quantitative descriptions (or get quantitative descriptions from component catalog)
2. Design system using component descriptions from ❶
3. Synthesize system using CAD tools
4. Compare internal profiles predicted in ❸ with those used in ❶
5. Repeat ❶ – ❹ until profiles are similar

A New Component-based Development Scheme

1. Develop and test components → quantitative descriptions (or get quantitative descriptions from component catalog)
2. Design system using component descriptions from ❶
3. Synthesize system using CAD tools
4. Compare internal profiles predicted in ❸ with those used in ❶
5. Repeat ❶ – ❹ until profiles are similar
6. Verify system against specification using predictions of ❸

A New Component-based Development Scheme

1. Develop and test components → quantitative descriptions (or get quantitative descriptions from component catalog)
2. Design system using component descriptions from ❶
3. Synthesize system using CAD tools
4. Compare internal profiles predicted in ❸ with those used in ❶
5. Repeat ❶ – ❹ until profiles are similar
6. Verify system against specification using predictions of ❸

That's what the other engineers do...

A New Component-based Development Scheme

1. Develop and test components → quantitative descriptions (or get quantitative descriptions from component catalog)
2. Design system using component descriptions from ❶
3. Synthesize system using CAD tools
4. Compare internal profiles predicted in ❸ with those used in ❶
5. Repeat ❶ – ❹ until profiles are similar
6. Verify system against specification using predictions of ❸

CAD Calculation ❸ is *much* faster and easier than system testing

Summary of the Talk

Questions

- What can unit testing do in principle?
- For what kind of code does unit testing work?

Background

- Unit testing here and there
- *Software components* are nice units
- Component-based software development (CBSD)
- Subdomain testing tools to synthesize CB systems

Results

- Pitfalls of component testing in CBSD
- Design rules in aid of component testing
- A new development/testing scheme