Software Component Synthesis: Theory and Supporting Tools

Dick Hamlet

Portland State University
E.T.S. Walton Fellow

Software Component Synthesis: Theory and Supporting Tools

Dick Hamlet

Portland State University
E.T.S. Walton Fellow

National University of Ireland, Galway
Ollscoil na hÉireann, Gaillimh

fondúireacht eolaíochta éireann

PLEAN FORBARTHA NÁISIÚNTA

Components in Engineering

Mechanical component:

- An independently specified and produced (small) part of a machine. Example: A rivet

Components in Engineering

Mechanical component:

- An independently specified and produced (small) part of a machine. Example: A rivet

Component specification:

Functional: rivet size, material, etc.

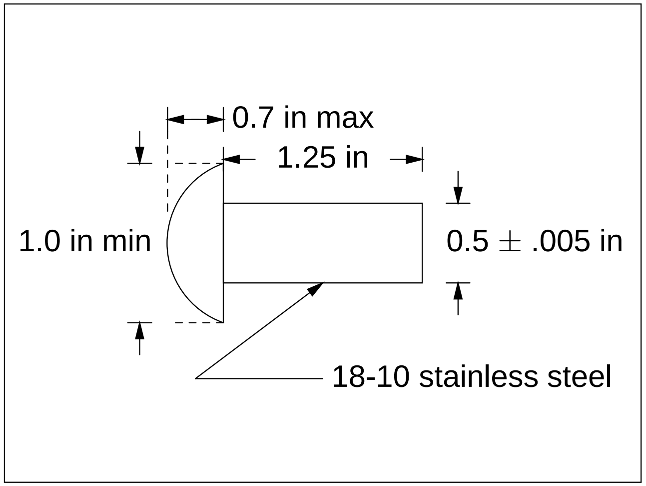
Quality: tolerance, hardness

Components in Engineering

Mechanical component:

- An independently specified and produced (small) part of a machine. Example: A rivet

Component specification:

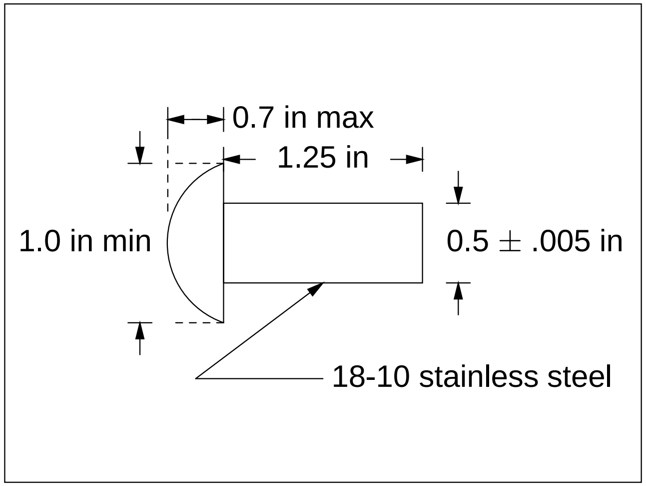


Components in Engineering

Mechanical component:

- An independently specified and produced (small) part of a machine. Example: A rivet

Component specification:



Engineering design is combining components and calculating properties of the resulting system

Component-based design spectacularly successful in electrical, mechanical engineering

Software Components? (Me too!)

- Definition (Szyperski): An executable, independently-deployable software unit with a black-box interface
- In practice: Anything from a small utility program (e.g., UNIX ‘cp’) to a large system (e.g., Adobe acrobat)
- In this research: An executable program with one floating-point input and output, no persistent internal state

Component-based Software

Component developer:

- Specify components: catalogue
- Complete access to all component details
- *No* knowledge of subsequent application

Component-based Software

Component developer:

- Specify components: catalogue
- Complete access to all component details
- *No* knowledge of subsequent application

System designer:

- Devise system structure (architecture)
- Select components from catalogue
- Calculate system properties
- *No* access to actual components

Theory from Program Proving

Specify function f and run time $T(x)$ on input x

Mathematical specifications:

- Program logic (Hoare)
- Algebraic equations (Gougen)
- Functional set theory (Mills)

Two components C_1 and C_2 in series:

$$f(x) = f_2(f_1(x))$$

$$T(x) = T_1(x) + T_2(f_1(x))$$

Practicality: Deriving T and f is difficult

Theory from Program Testing

Sample the input space to estimate f and T.

Operational-profile problem:

- Input domain is on the order of size $2^{64} \approx 10^{19}$ but only 10^5 samples are practical
- Valid sampling requires a distribution
- The distribution a component sees in a system depends on the *system* input distribution and the *system* structure, not available to the component developer

Solving the Operational-profile Problem

- Partition component input space into subdomains $D = \cup_{i=1}^{n} S_i$
- Approximate component behavior on each subdomain: $<v_1, v_2, ..., v_n> \equiv <v_i>_{i=1}^{n}$
- Profile $<w_1, w_2, ..., w_n>$ as a weighting over subdomains
  - Component developer measures $<v_i>_{i=1}^{n}$
  - System designer later applies $<w_i>_{i=1}^{n}$

    to get a system average: $\frac{1}{n}\sum_{i=1}^{n} w_i v_i$

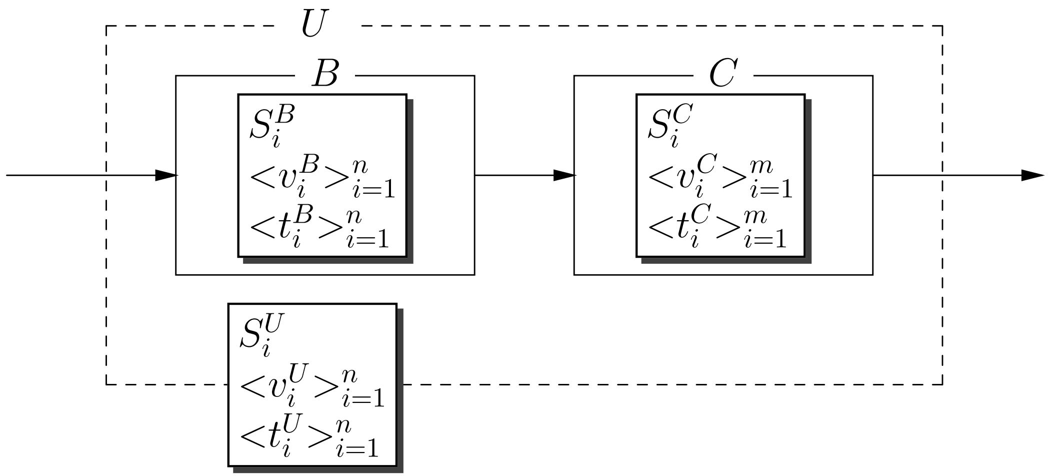
Algebra of Equivalent Components

- ‘Equivalent component’ construction rules:
  - sequence
  - conditional
  - iteration

$$\{\text{component specs}\} \xrightarrow{rule} \text{equiv-component spec}$$

- Repeatedly apply the rules to analyze a system of arbitrary complexity

Subdomain, Vector Notation



Component subdomains: $\{S_1^B, S_2^B, ..., S_n^B\}$
$\{S_1^C, S_2^C, ..., S_m^C\}$

Functional vectors: $<v_1^B, v_2^B, ..., v_n^B>$
$<v_1^C, v_2^C, ..., v_m^C>$ $\Rightarrow <v_1^U, v_2^U, ...v_n^U>$

Run-time vectors: $<t_1^B, t_2^B, ..., t_n^B>$
$<t_1^C, t_2^C, ..., t_m^C>$ $\Rightarrow <t_1^U, t_2^U, ...t_n^U>$

Sequence rule: $C_1; C_2$

Subdomains: $S_i^U = S_i^{C_1}$

Suppose $v_i^{C_1} \in S_j^{C_2}$. Then:

Functional values: $v_i^U = v_j^{C_2}$

Run-time values: $t_i^U = t_i^{C_1} + t_j^{C_2}$

Conditional: `if` B `then` C_1 `else` C_2 `fi`

Let: $D_T = \{x | B(x)\}$
$D_F = \{x | \neg B(x)\}$

On subdomains $D_T \cap S_i^{C_1}$:

$$v_k^U = v_i^{C_1}$$
$$t_k^U = t_p^B + t_i^{C_1}$$

On subdomains $D_F \cap S_j^{C_2}$:

$$v_k^U = v_j^{C_2}$$
$$t_k^U = t_p^B + t_j^{C_2}$$

Iteration rule: `while` B `do` C `od`

- Unwind to:

  $$\texttt{if}\ B\ \texttt{then}\ C\ \texttt{else}\ I\ \texttt{fi};\ \texttt{while}\ B\ \texttt{do}\ C\ \texttt{od}$$

  where I is the component computing identity with zero run time
- Repeat until residual loop disappears in each subdomain
- Use conditional and sequence rules

Piecewise-linear Approximation

Replace the vector constants v_i, t_i with pairs

(*slope, intercept*)$_i$

of the best-fit line over the subdomain

Piecewise-linear Approximation

Replace the vector constants v_i, t_i with pairs

(*slope, intercept*)$_i$

of the best-fit line over the subdomain

- Advantages:
  - Better subdomains for sequences
  - Exact representation of the identity component I

Piecewise-linear Approximation

Replace the vector constants v_i, t_i with pairs

(*slope, intercept*)$_i$

of the best-fit line over the subdomain

- Advantages:
  - Better subdomains for sequences
  - Exact representation of the identity component I
- Disadvantage: Can't be used with non-numeric data types

Ideal Supporting Tools

Components are measured to drive a computer-aided design (CAD) tool that does system-design calculations

Systems are designed using a components catalogue, without any access to component code and without constructing or executing any system code

Tools for Component Developers

Create catalog entry (specification) from:

- Subdomain list
- Component code

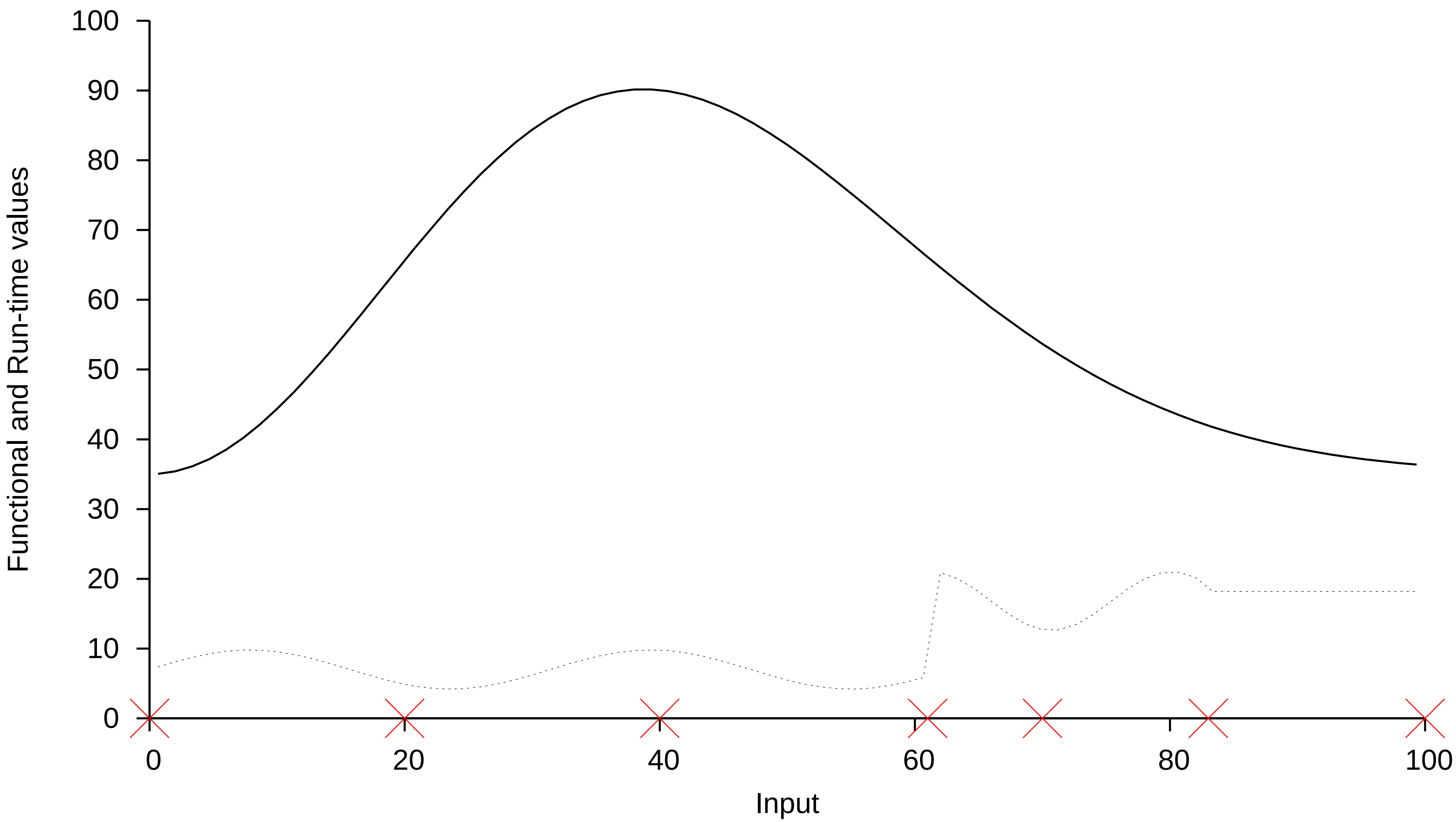
```
1.bin
0 20 13
20 40 13
40 61 13
61 70 13
70 83 13
83 100 13
```

```
#! /usr/bin/perl
$x = <STDIN>;
$y = (($x**2)/10)*exp(-($x**2)/1500) + 35;
if ($x < 61) {
  $t = 5 + 2*sin($x/5);
}
elsif ($x >= 61 && $x < 83) {
  $t = 12 + 3*sin($x/3);
}
else {
  $t = 13;
}
$t = $t*.2;
print STDERR $t,"\n";
print $y,"\n";
```

Tools for Component Developers

Create catalog entry (specification) from:

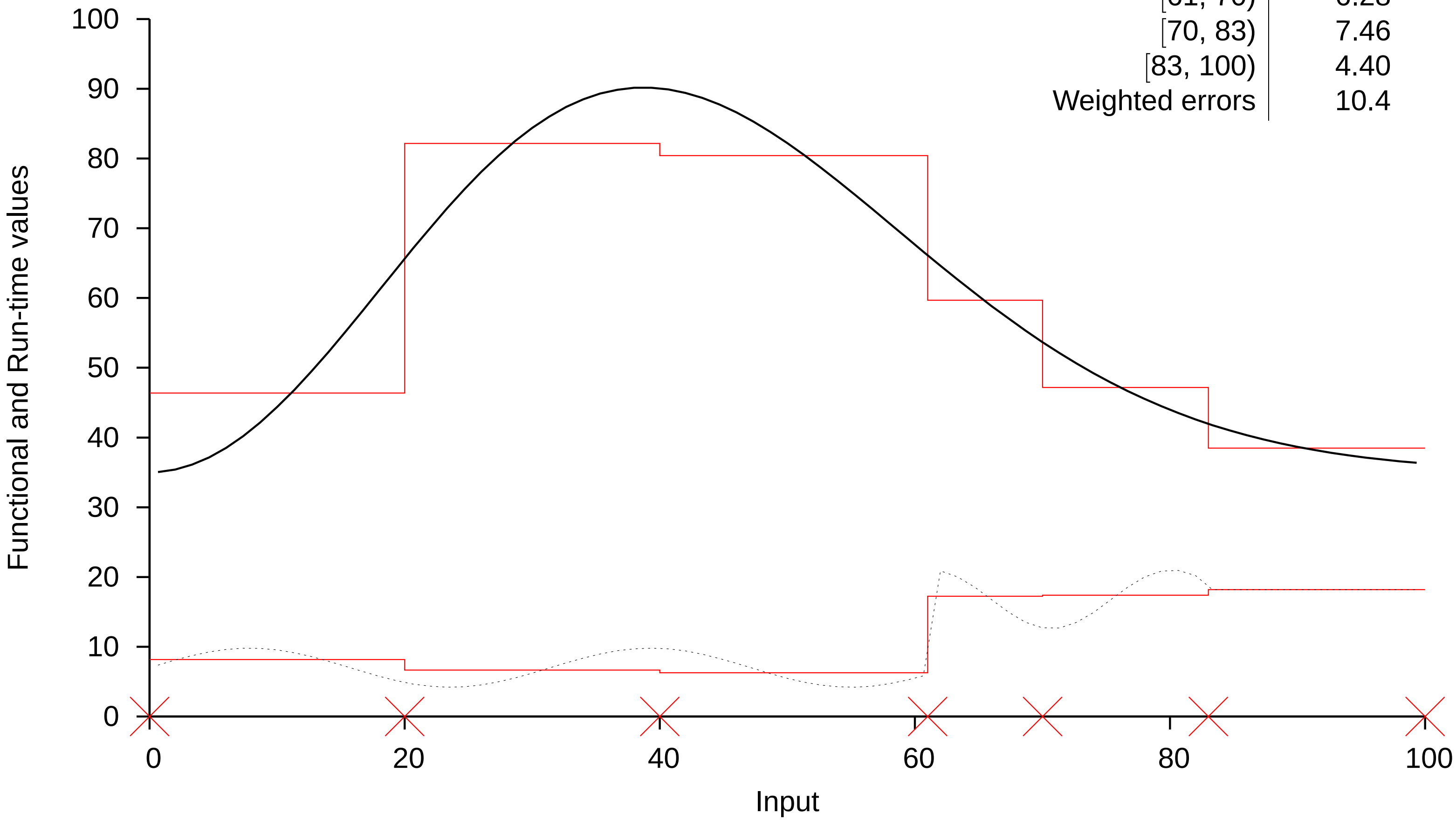
- Subdomain list
- Component code



Approximating Component Behavior

Step-function approximation

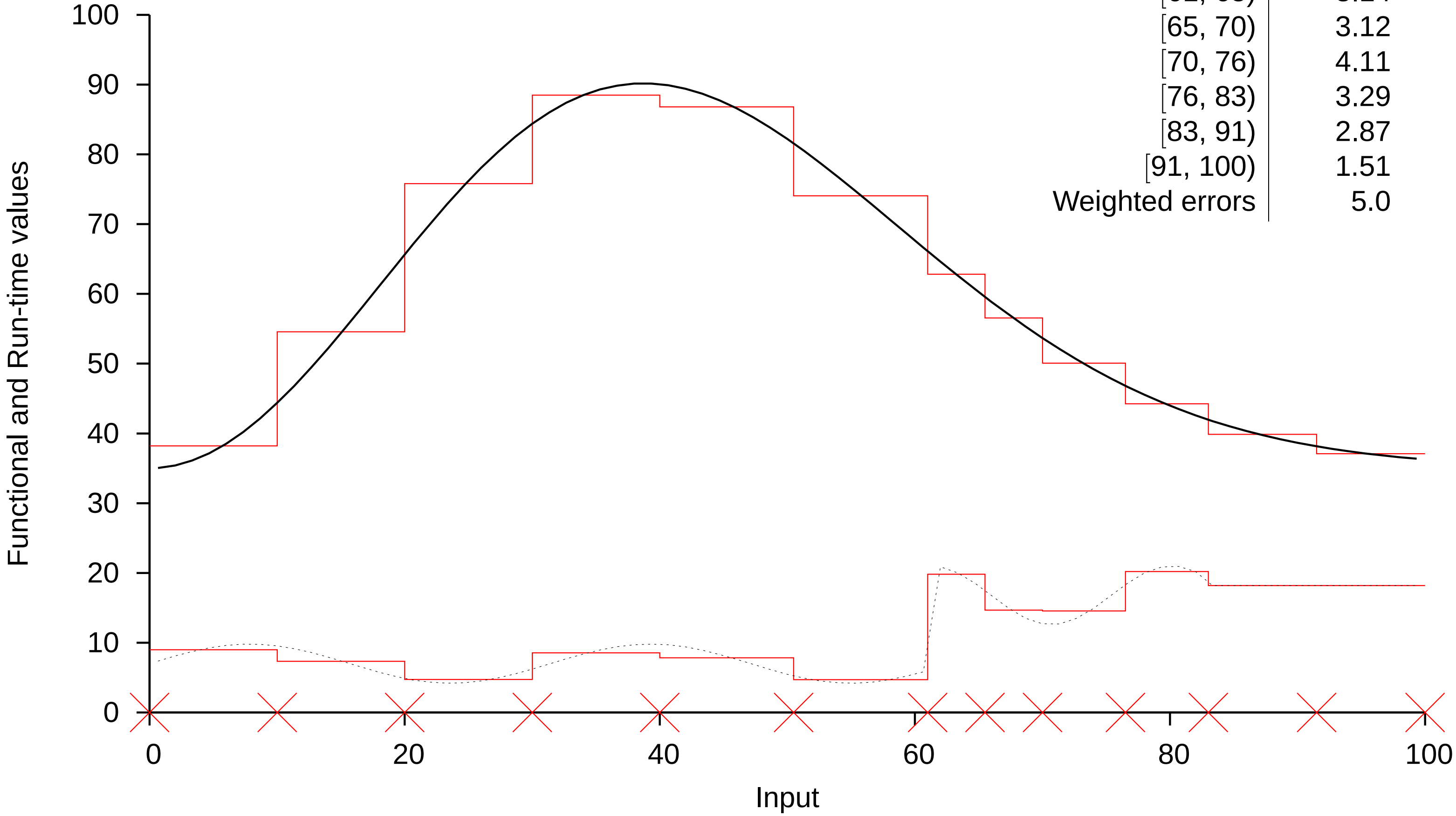
Subdomain	Root-mean-square % errors Function	Run time
[0, 20)	21.28	18.38
[20, 40)	9.52	32.97
[40, 61)	9.49	31.37
[61, 70)	6.28	17.32
[70, 83)	7.46	18.86
[83, 100)	4.40	0.00
Weighted errors	10.4	20.9



Approximating Component Behavior

Step-function approximation

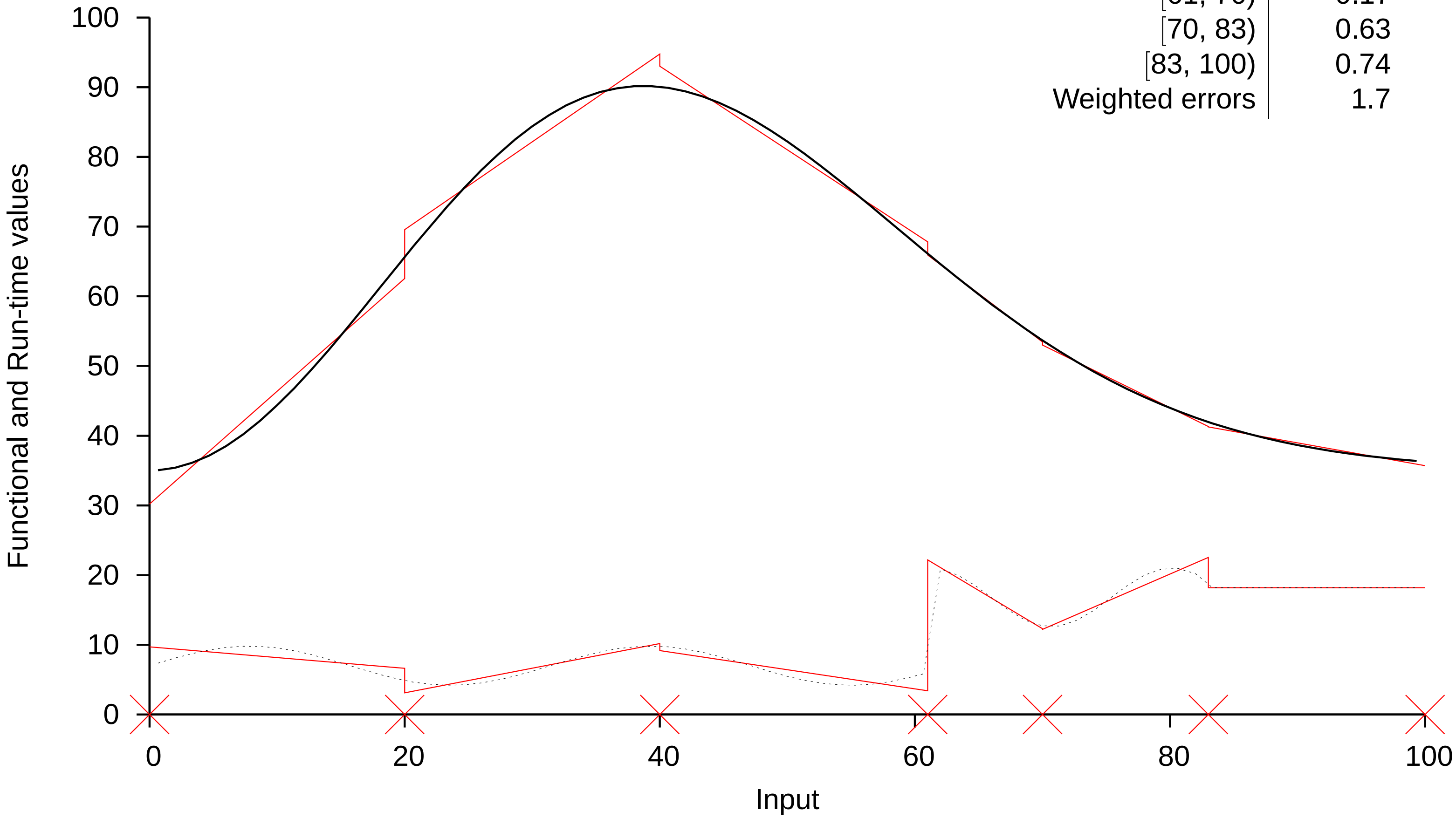
Subdomain	Root-mean-square % errors Function	Run time
[0, 10)	7.65	9.87
[10, 20)	11.84	20.53
[20, 30)	7.50	12.16
[30, 40)	2.03	13.74
[40, 50)	3.03	18.84
[50, 61)	6.31	10.51
[61, 65)	3.14	5.72
[65, 70)	3.12	10.18
[70, 76)	4.11	12.97
[76, 83)	3.29	3.70
[83, 91)	2.87	0.00
[91, 100)	1.51	0.00
Weighted errors	5.0	10.5



Approximating Component Behavior

Piecewise-linear approximation

Subdomain	Root-mean-square % errors Function	Run time
[0, 20)	3.87	13.97
[20, 40)	2.37	8.17
[40, 61)	1.30	14.55
[61, 70)	0.17	2.05
[70, 83)	0.63	6.07
[83, 100)	0.74	0.00
Weighted errors	1.7	8.5



Tools for System Designers

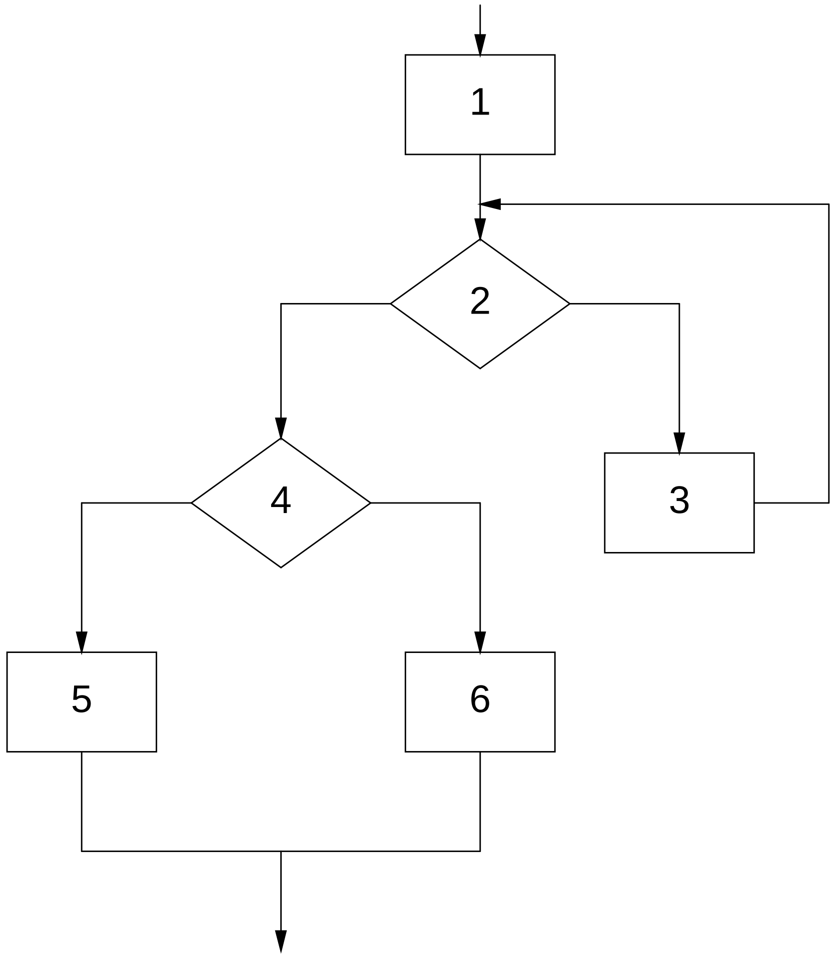
(CAD) Synthesize Equivalent component from:

- System structure description
- Component-specification list (catalogue)

Tools for System Designers

(CAD) Synthesize Equivalent component from:

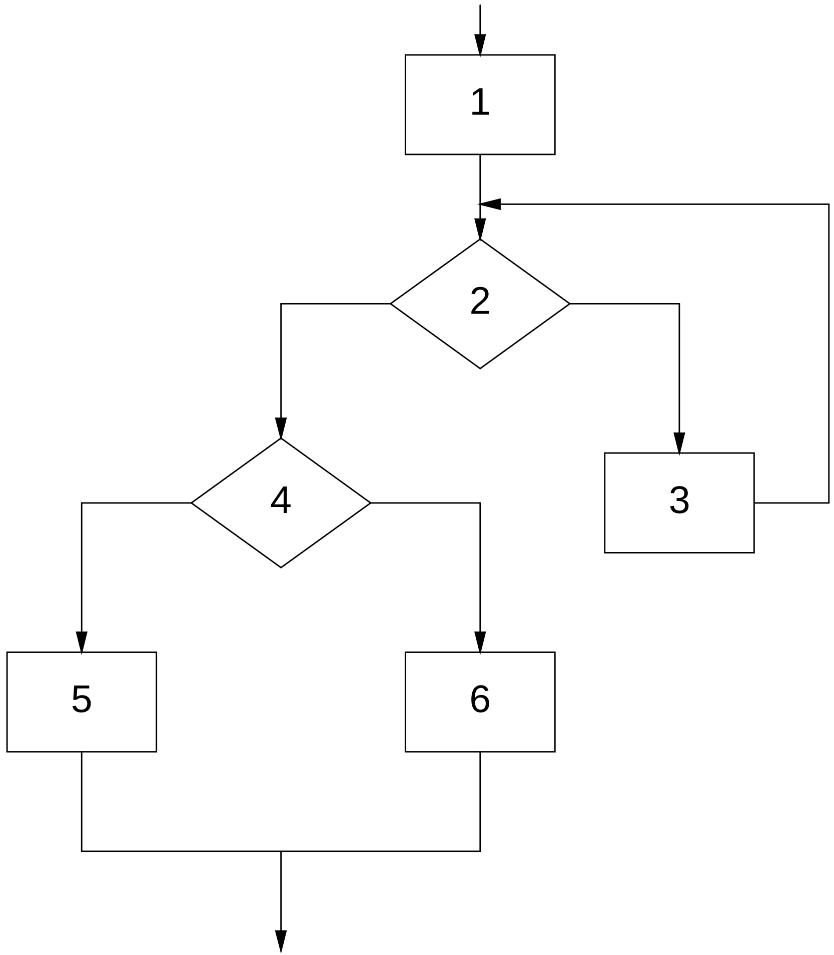
- System structure description
- Component-specification list (catalogue)



Tools for System Designers

(CAD) Synthesize Equivalent component from:

- System structure description
- Component-specification list (catalogue)




```
1 2 3 L S 4 5 6 C S
1.ccf
2.ccf
3.ccf
4.ccf
5.ccf
6.ccf
```

‘Validation’ Experiments

- Tool/theory debugging
- Creation of artificial components
- To be investigated:
  - Accuracy vs. subdomain size
  - Step-function vs. piecewise-linear approximation
  - Choice of subdomains

Artificial Components

Monitoring behavior in experiments (UNIX)
1. Time component externally

```
#! /bin/tcsh shell script
# get time in $Elapsed0 variable

perl ComPonent
# execute code, output to stdout

# get time in $Elapsed1
# compute $Elapsed = $Elapsed1 - $Elapsed0

echo $Elapsed > /dev/stderr # send time to stderr
```

Artificial Components

Monitoring behavior in experiments (UNIX)
2. Wrap component to time itself

```
#! /usr/bin/perl
# call UNIX timer function

# code for ComPonent here:
while ($LongTime) { ... } # runs 200 ms
print "99\n"; #output to stdout

# call UNIX timer function
$Diff = ...; # subtract times
print STDERR "$Diff\n"; # time to stderr
```

Artificial Components

Monitoring behavior in experiments (UNIX)
3. Fake the actual behavior

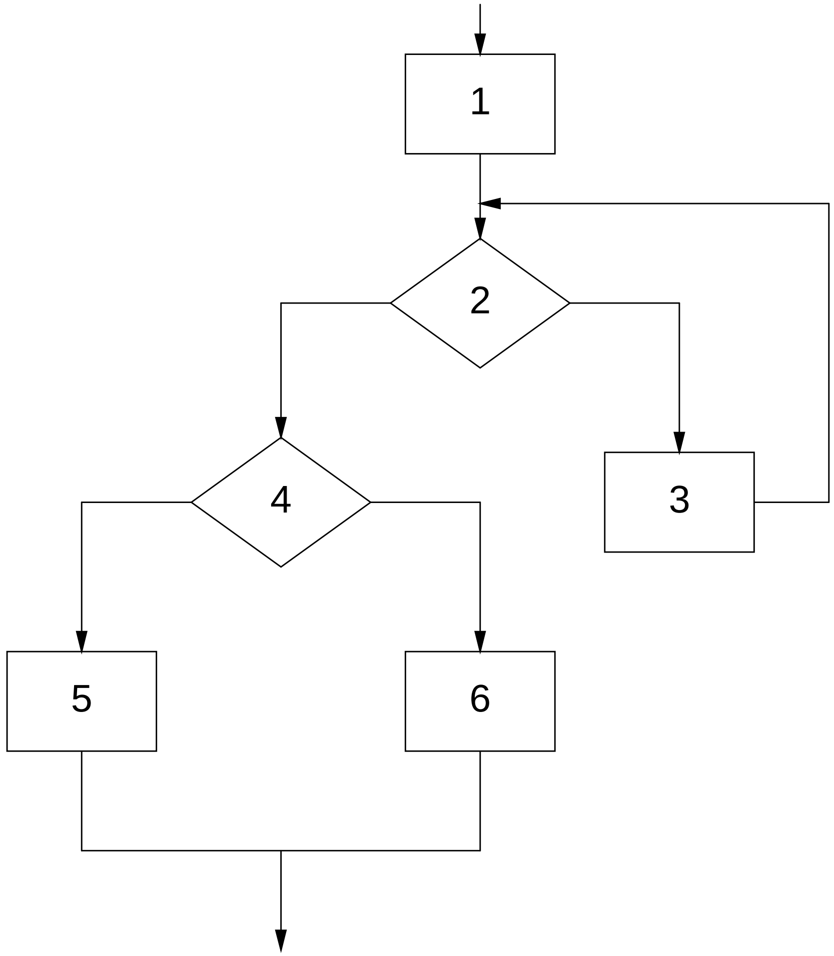
```
#! /usr/bin/perl
print "99\n"; #output to stdout
print STDERR "200\n"; # time to stderr
```

Artificial Components

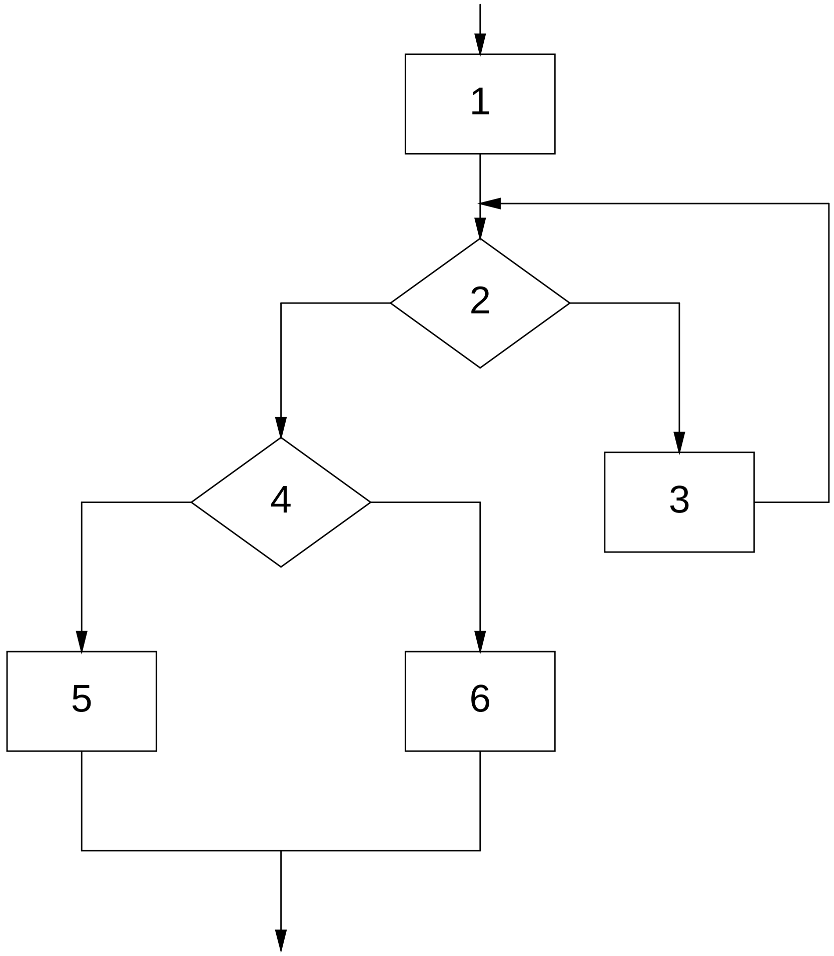
Monitoring behavior in experiments (UNIX)

- Can't tell the difference!
- Fake is *much* faster
- No operating-system variability
- Measure and replace real components with fakes

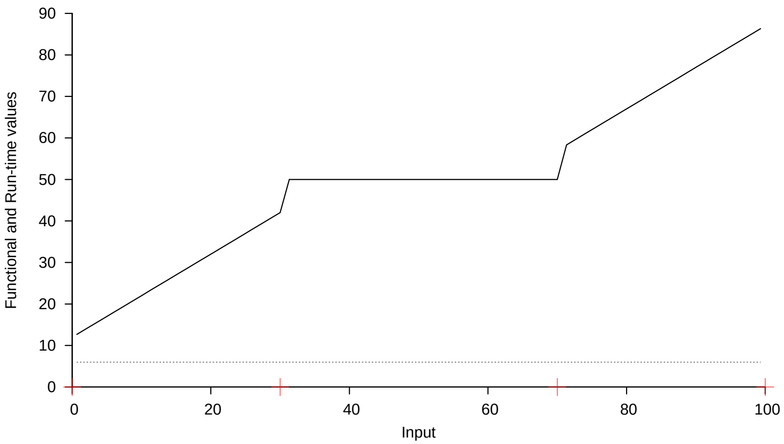
Case-study System and Components



Case-study System and Components



Comp	Function $y = f(x)$
1	$y = \frac{1}{10}x^2e^{-x^2/1500} + 35$
2	*false* $30 \leq x < 70$
3	(see graph)
4	*true* $x \geq 50$
5	$y = \begin{cases} 90 - 2.5x, & x < 30 \\ 15 + 1.1(x - 30), & x \geq 30 \end{cases}$
6	(same as 1)



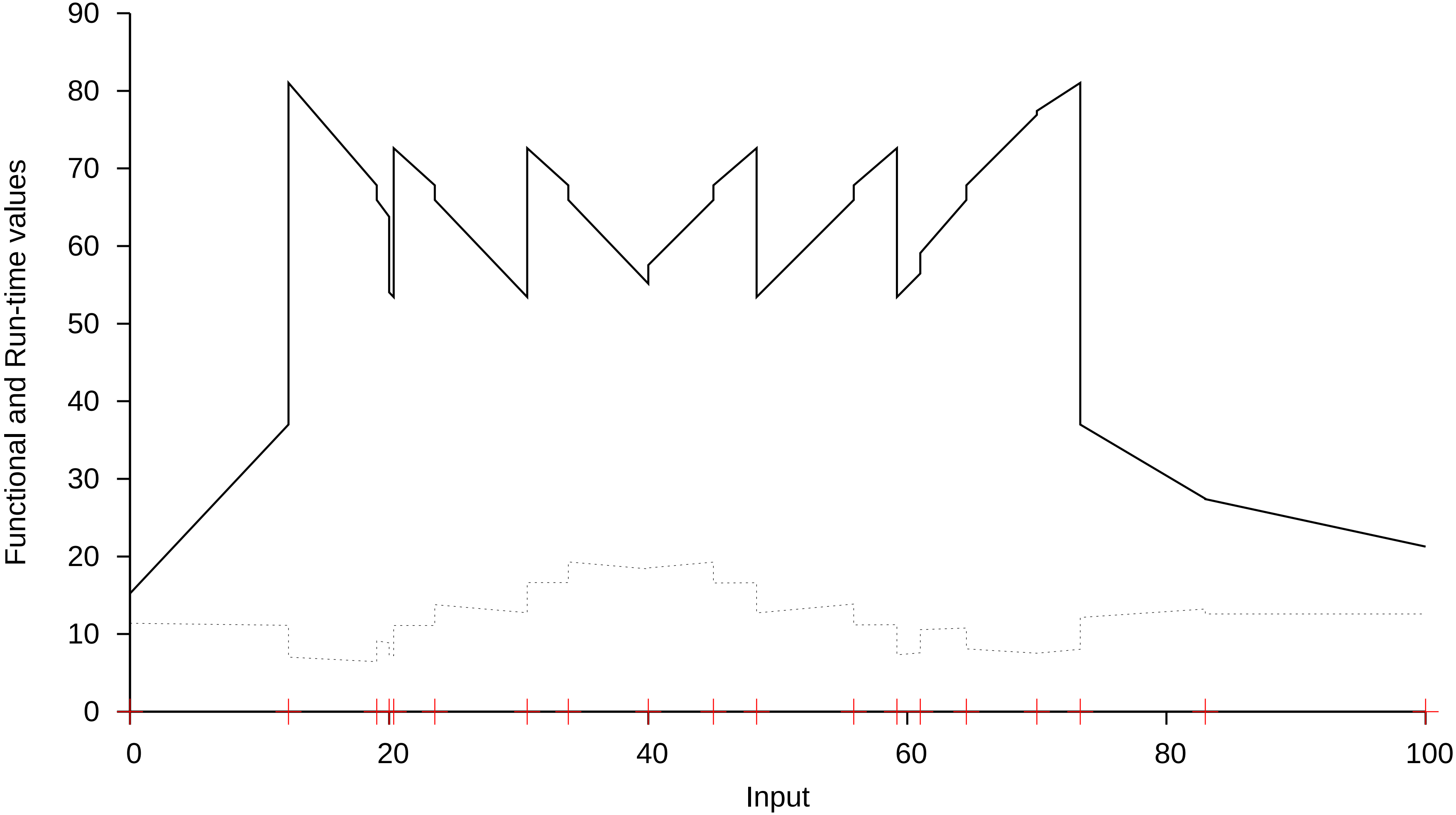
CAD Tool Execution

```
Beginning system:  Polish 1 2 3 L S 4 5 6 C S
Using piecewise-linear approximation

Loop:  WHILE 2.bin DO 3.bin OD -> theory1
  Conditional:  IF 2.bin THEN 3.bin ELSE  FI ->
                once.ccf -> again.ccf
  (again.ccf stripped of false subdomains -- 2 true)
  Series: again.ccf; once.ccf -> another.ccf ->
          again.ccf (2 still true)
  Series: again.ccf; once.ccf -> another.ccf ->
          again.ccf (2 still true)
  Series: again.ccf; once.ccf -> another.ccf -> again.ccf

Series: 1.bin;theory1 -> theory2

Conditional:  IF 4.bin THEN 5.bin ELSE 1.bin FI -> theory3

Series: theory2;theory3 -> theory4
```

CAD Tool Execution

Subdomain	Function	Run time
[0.000,12.239)	26.11	11.25
[12.239,19.039)	74.43	6.74
[19.039,20.000)	64.85	9.01
[20.000,20.351)	53.74	7.24
[20.351,23.525)	70.23	11.11
[23.525,30.666)	59.68	13.27
[30.666,33.840)	70.23	16.63
[33.840,40.000)	60.54	18.86
[40.000,45.026)	61.74	18.91
[45.026,48.359)	70.23	16.61
[48.359,55.857)	59.68	13.30
[55.857,59.189)	70.23	11.19
[59.189,61.000)	54.94	7.45
[61.000,64.551)	62.51	10.67
[64.551,70.000)	72.37	7.82
[70.000,73.345)	79.22	7.79
[73.345,83.000)	32.22	12.67
[83.000,100.000)	24.32	12.60

CAD Tool Execution



Subdomains and Accuracy

Successively halve subdomains:

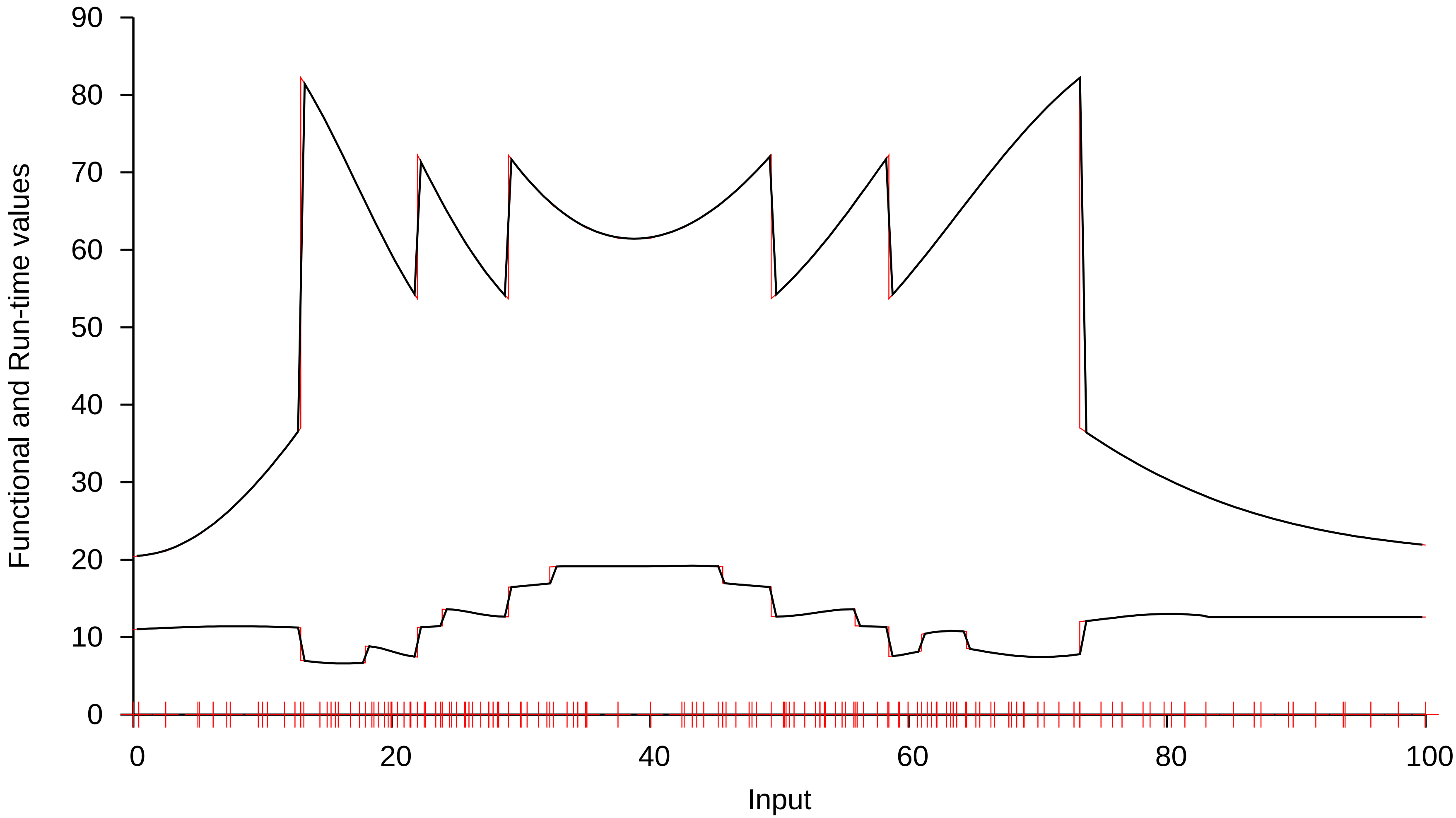
Step-function Approximation

Piecewise-linear Approximation

Subd count	Functional % error Overall	Max	Mean	> 5	Run-time % error Overall	Max	Mean	> 5
6	– ABORTED –							
12	22.13	50.21	16.59	6	9.37	21.39	7.32	5
18	5.16	12.96	3.85	6	3.56	18.87	4.49	7
24	9.69	24.61	7.86	13	8.05	40.37	6.27	8
32	0.70	21.35	1.65	3	0.40	15.71	1.49	3
48	8.47	19.01	6.13	24	3.22	17.03	2.22	7
76	0.52	55.01	0.97	2	0.55	53.55	1.00	2
96	4.42	28.85	2.82	20	1.20	18.42	0.93	5
152	0.04	55.00	0.40	1	0.02	54.02	0.41	1
192	1.40	25.01	1.55	6	0.50	15.56	0.58	9
384	0.64	34.43	0.79	5	0.09	21.73	0.32	8
768	0.69	30.46	0.42	5	0.50	27.24	0.19	8

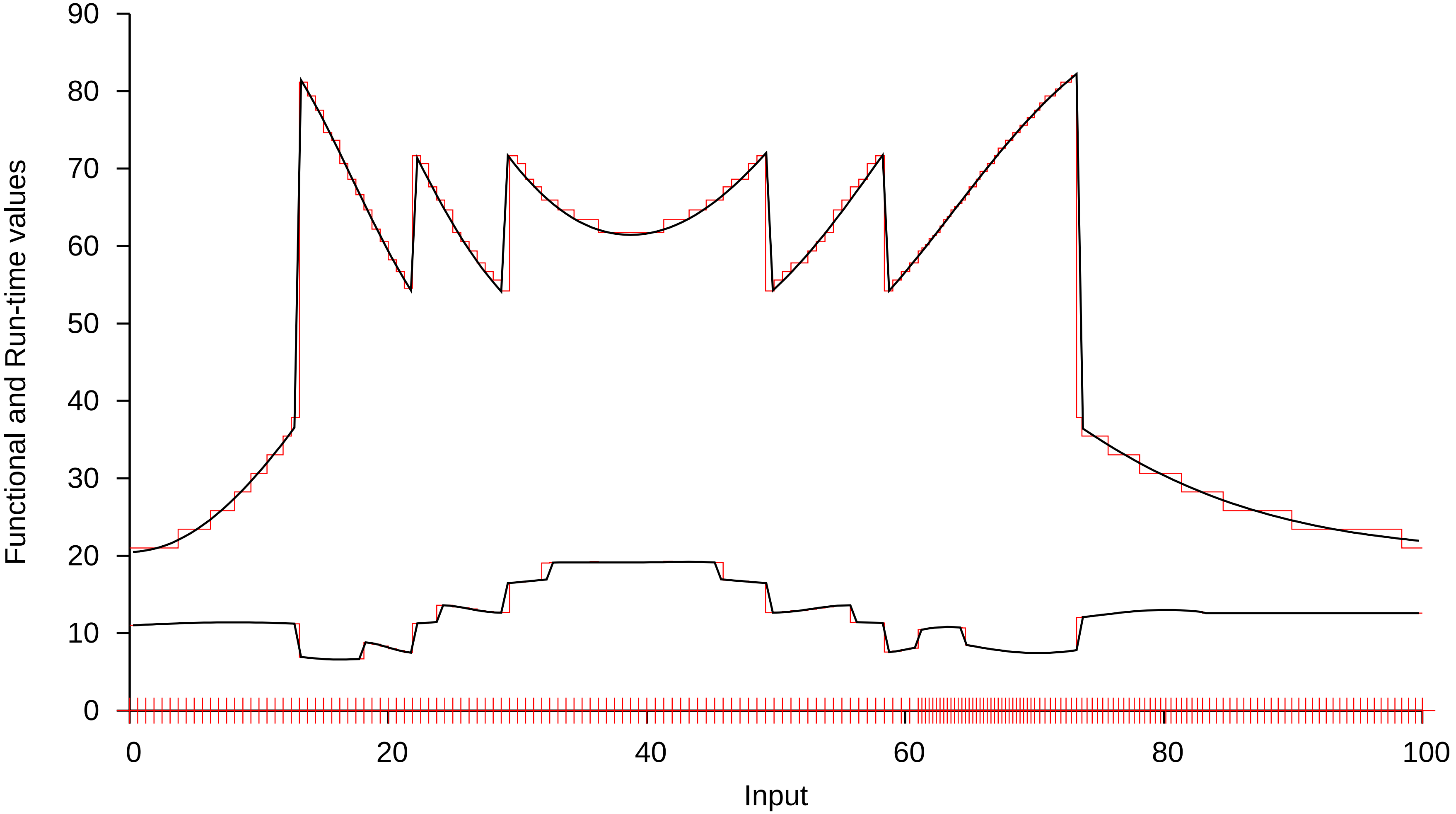
Subdomain Boundary Problems

Even small subdomains miss discontinuities



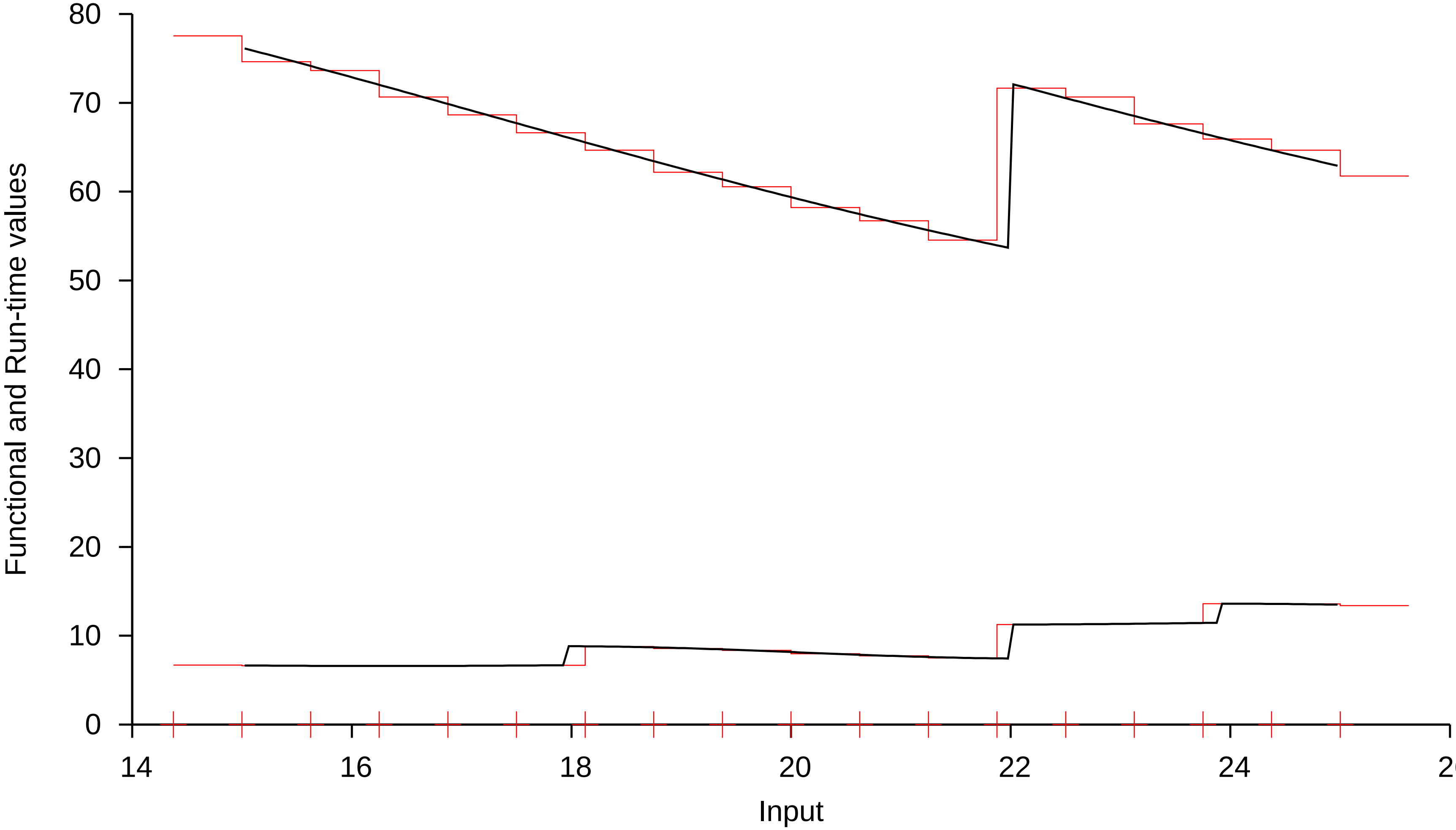
Subdomain Boundary Problems

Even small subdomains miss discontinuities



Subdomain Boundary Problems

Even small subdomains miss discontinuities



Open Problems

- Error prediction
  - Rms approximation error predictive?
  - Functions always badly behaved
- Components retaining state
  - Most real components have persistent state
  - State is not just an extra input variable

QUESTIONS? COMMENTS?