Coverage Testing SDL Models

W. Eric Wong
Department of Computer Science
The University of Texas at Dallas
ewong@utdallas.edu
http://www.utdallas.edu/~ewong
**Speaker Biographical Sketch**

- Professor & Director of International Outreach  
  Department of Computer Science  
  University of Texas at Dallas

- Vice President, IEEE Reliability Society

- Secretary, ACM SIGAPP (Special Interest Group on Applied Computing)

- Principal Investigator, NSF TUES (Transforming Undergraduate Education in Science, Technology, Engineering and Mathematics) Project:  
  *Incorporating Software Testing into Multiple Computer Science and Software Engineering Undergraduate Courses*

- Founder & Steering Committee co-Chair for the SERE conference  
  (*IEEE International Conference on Software Security and Reliability*)  
  (http://paris.utdallas.edu/sere12)
Software Development


Maintenance (standard upgrade, new features, evolution, .......)


Maintenance (standard upgrade, new features, evolution, .......)

Our Vision

Workflow Specification

Petri (semi)automatic

Architecture/Design-Based Analysis of C/C++/Java Program

Assessment and Diagnosis of Architecture/Designs as well as Source Code

SDL Specification

UML Specification

C/C++/Java Program

Program-Based Analysis of C/C++/Java

Trace (Dynamic Run-time Information)
Software Architecture Design in SDL

• SDL (Specification and Description Language) is an object-oriented, formal language for designing complex, real-time, and communicating systems
  – Visit http://www.sdl-forum.org for more details

• The architectural design of a software system in SDL can be viewed as a collection of blocks and processes communicating with each other by exchanging signals through channels.
  – An SDL specification provides a process view of a system's architectural design
Life of an SDL Specification

White box-based Coverage Testing, Debugging, and Test Generation

Black-box Testing Question: Is this enough?

Conformance Testing Q: What if the SDL specification is not correct?
**Graphical and Textual Representations**

- The graphical representation is called GR and the textual representation is called PR (Phrase Representation).
  - Automatic translation between GR and PR can be done

```plaintext
process Game;
dcl
  Count: Integer;
start:
  task Count := 0;
  nextstate Losing;
state Losing:
  input Probe;
  output Lose;
  task Count := Count - 1;
  nextstate -;
  input Bump;
  nextstate Winning;
  endstate;
state Winning:
  input Bump;
  nextstate Losing;
  input Probe;
  output Win;
  task Count := Count + 1;
  nextstate -;
  endstate;
state = ;
  input Result;
  output Score{Count};
  nextstate -;
  input GameOver;
  stop;
  endstate;
endprocess Game;
```
Coverage Testing SDL Specifications (1)

• The textual representation of SDL specifications can be viewed as "programs in a specification and description language," just like programs in C.
  – All the testing methods applied to C programs, including random testing and functional testing (both are black box oriented) as well as control flow-based and data flow-based white box coverage testing, can also be applied to SDL specifications.
  – How much of the design specification is currently tested?
  – What is missing?
  – Need help in creating tests?

Analyzing the control-flow graph of an SDL specification to find the dominant blocks, decisions, etc. For example, when a test case covers highly dominant blocks it will cover many other blocks.

Coverage Testing SDL Specifications (2)

- Visualizing coverage in SDL specification and its control-flow graph

- A control flow graph is generated for each SDL process
- The textual representation displays the SDL source code, whereas the control flow graph makes its flow of control more evident
**Coverage Testing SDL Specifications (3)**

This decision (either the TRUE branch and/or the ELSE branch) has the highest weight.

Double-clicking on the decision shows the corresponding branches with their respective weights.

There are two decisions: one at block 5 and the other at block 3.

Double-clicking on block 5 to view the associated branches.

Here is the decision.

Here are the branches of this decision.

The TRUE branch has already been covered.

The ELSE branch has a weight of 1.

The decision at block 5 has two branches: one from block 5 to block 6, and the other from block 5 to block 7.
Design Philosophy for Testing/Maintenance Tools

• When you develop a testing/maintenance tool, you should consider
  – Ease of use
  – Visualization
  – Prioritization
  – Granularity
  – Incrementability
  – Extensibility
  – Portability
  – etc.
**Alternating Bit Protocol**

- The alternating bit protocol, which is a simple form of the “sliding window protocol” with a window size of 1, is used as the example.

- It can be used to provide **reliable communication over non-reliable network channels** through a **one-bit sequence number** (which alternates between 0 and 1) in each message.
**Alternating Bit Protocol (2)**

- The alternating bit protocol is constituted by a sender and a receiver who exchange messages through two channels, Medium1 and Medium2.
  - When the sender sends a message (containing a protocol bit, 0 or 1) to the receiver through *Medium1*, it sends the message *repeatedly* (with the corresponding protocol bit) until receiving an acknowledgment from the receiver that contains the same protocol bit as the message being sent.
  - When the receiver receives a message, it sends an acknowledgment to the sender through *Medium2* and includes the protocol bit of the message received.
  - The first time the message is received, the protocol delivers the message for processing. *Subsequent messages with the same bit are simply acknowledged.*
  - When the sender receives an acknowledgment containing the same bit as the message it is currently transmitting, *it stops transmitting that message, flips the protocol bit*, and repeats the protocol for the next message.
  - *This implies that the sender associates each message with a protocol bit which is alternated between 0 and 1 to differentiate consecutive messages.*
Alternating Bit Protocol (3)

put('world') from: env

[put]

bit: 0

sender

start

wait_put

wait_am

receiver

get('world') from: receiver

bit: 0

[get]

[put]

medium1

start

wait_dm

medium2

start

wait_am

am(1) from: medium1

am(0) from: medium1

am(1) from: medium1

am(1) from: receiver

am(0) from: receiver

am(1) from: receiver

[am]

am_error

[am, medium_error]

[dm, medium_error]
Questions

• Do you know how much of your SDL code has been tested?
• What is still missing?
• How to generate additional test cases to execute the uncovered code in an effective way?
CAT\textsubscript{SDL}: A Coverage Analysis Tool for SDL Specifications

- Given an SDL specification, CAT\textsubscript{SDL} performs instrumentation on it by inserting a probe, a user-defined function, at appropriate locations.

- The resulting instrumented specification is then exported to an SDL simulator (Telelogic Tau in our case) for simulation.

- A file is created to record the trace information during the simulation.

- As subsequent simulation continues, the trace information is appended to the trace file which is then exported back to CAT\textsubscript{SDL} for coverage analysis.
CATSDL Demo