An Approach to Practical Validation of Control Software Specification

Tomoyuki Kaga
Toyota Motor Co.

Masakazu Adachi
Toyota Central R&D Labs
Contents of this talk

1. Automotive control software development
2. Specification validation and open issues
3. Concept of the practical approach
4. Tool implementation
5. Application
6. Summary and future direction
Definition of Verification and Validation

Standard definitions

- **Verification**
  - Confirmation by examination and provision of objective evidence that specified requirements and are fulfilled
  - Ensuring implementation satisfies the requirements for that step. Can include testing, analysis and review
  - You built product right

- **Validation**
  - Confirmation by examination and provision of objective evidence that particular requirements for a specific intended use are fulfilled
  - Ensuring requirements are complete and correct
  - You built the right product

Definition in this talk

- **Validation**
  - Source code (C program)
  - ECU

- **Verification**
  - Coding

Software specification

Impl.
1. Automotive control software development
2. Specification validation and open issues
3. Concept of the practical approach
4. Tool implementation
5. Application
6. Summary and future direction
Automotive control system and software

Major automotive control systems

- Interacting with physical environment and various drivers
- Growing contribution of elaborate control systems to deliver attractive products and to meet regulations

- Coping with growing complexity is a big challenge
Traditional Development Process

- Repetitive prototyping loops
- Requires different domain skills: system and software
- Incremental

Target

Concept study

Control design

SW spec authoring

System eng. <OEM>

SW eng. <Supplier>

System evaluation

Integration

Unit test

Coding

SW level design

PlanA

PlanB

Pro1

Con1

Pro2

Con2

Con3

Con1

Con2

Con3
Example: Developing Cold Start Control of Gasoline Engine

**Target**
Reduce 5% Hydro Carbon emission

**Concept**
Activate catalyst converter as fast as possible

**Control design**
Optimize reference profiles of throttle, fuel injection and spark timing.

**Supposed prototyping loops**

- Standard engine system
- Select related specification
- Throttle
- Fuel inj.
- Spark
- Modification
- Throttle
- Fuel inj.
- Spark
- Coding
- Implement
- Evaluate!

Experimental engine system

if successful, Standardization

2nd prototyping loop
System engineering is exploration of feasible solution

- Due to growing complexity, hard to foresee exploration paths without prototyping
- Solution space is getting narrower

**Improvement of prototyping loops is the key**
Promotion of Model-Based Development

Model-Based Development

Virtual World

- Engine Performance Specification
- Control Software Specification

Combination

- Plant Model
- Controller Model

SILS / MILS

Real World

- Plant (Engine, Transmission etc.)
- Controller (Hardware, Software)

Rapid Prot. ECU

HILS

Combination

Validation
Model-based Development of Control Software

Small virtual loops reduce big physical loops which are much expensive

More intuitive representation to system engineers

Model-based code verification

Auto code generation
Areas where further improvement is needed

- Validation of code specification
- Lack of documents/evidence

Spec validation & SW level design

Target

Concept study

Control design

Virtual system evaluation

Proto modeling and implementation

Legacy SW (C, SL)

Proto SW model

System perspective

Software perspective

Integration

Spec Validation

Code Verification

Unit test

Coding
Summary - Automotive control software development

• Requires repetitive prototyping loops for feasible solution exploration

• Promotion of Model-based Development to improve prototyping loop efficiency

• Specification validation is a big challenge
1. Automotive control software development
2. Specification validation and open issues
3. Concept of the practical approach
4. Tool implementation
5. Application
6. Summary and future direction
Question is simple:

Is the sample set sufficient?

Though true validity can only be confirmed in actual uses…
Making development process formal and accountable

- Systematic breakdown from system requirement to software spec
- Make design artifacts traceable to higher requirements as evidence
- Use formal models to apply advanced verification technology: property checking
  - Model checking
  - Exhaustive testing

Formal assumption (set of inputs) → Formal property (set of outputs)

Target

System requirement

Software specification

Verification

Refinement

No need to care about samples! – exhaustiveness assured
Formal process and verification - lessons learned

Limitation on model scale and type of arithmetic

Difficult to define property for continuous and/or dynamic behaviors

Property is too obvious for small models/codes

Requisite
- Design is complex enough, but not too much
- Property must be simply describable relative to the Design

So far applicable area is limited.
Rabbit vs. Turtle

Agile
Informal
Small system
Not scalable
Integral
Unaccountable
Individual/Skill oriented
Efficient

Waterfall
Formal
Large system
Scalable
Modular
Accountable
Team/Rule oriented
Redundant

On the way to finding the best level of formality

In Japan:
- Low communication cost
- Cultural strength
- …
A practical problem - latent function

Under a rare condition, hunting behavior had occurred. The cause was an unrecognized dependence loop.

Due to complex dependence among function, especially via shared memory, one function was overlooked and the condition hadn’t been exercised due to its rareness.

- C code is to blame for … ?
Another practical problem - latent function in a Simulink model

It seems there is no latency in models, but it is not as obvious as we suppose.

Supposed reasons of incomprehensibility:
- Lack of software design skill or less care about model quality
- Essential paths are obscured by software level design details (e.g. type guard)
- Functional grouping is not an easy task
Summary - Specification validation and open issue

- “Is the sample set enough?”
- Current usage of formal verification is limited
- On the way to finding the best level of formality
- One of validation problem: Latent function
1. Automotive control software development
2. Specification validation and open issues
3. Concept of the practical approach
4. Tool implementation
5. Application
6. Summary and future direction
A direction to go - whitebox

Testing with software coverage metrics

Make sure branches and conditions of each switch are exercised

Problems:
- Are functional components really covered?
- Bad S/N for functional coverage
  (importance of branches are not even)
- Hard to infer validity without knowing which function was stimulated

There is a chance of div by 0 error with 100% branch coverage

Hard to get a sense of functional coverage
Functional coverage

Example: 1 input, 1 output

A point exercised by the sample
(trajecory if dynamic)

Coverage is supposed to be sufficient if each of equivalence class of function is exercised in proper manner respectively.
Specification validation by design interest extraction

Design intension

Prototype software

C

Simulink

Other models

Identify equivalence class

Model analysis

Exercise equivalence class

Behavioral analysis

Design Interest

which has similar abstraction level to design intension
Defect category and our target

(a) Defect of recognized function
(b) Absence of function
(c) Defect of latent function
(d) Potential absence of function

Class of function

<table>
<thead>
<tr>
<th>Scenario Specified</th>
<th>Class of function</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Exist</td>
</tr>
<tr>
<td>(a) Recognized</td>
<td>Exist</td>
</tr>
<tr>
<td>(b) Absent</td>
<td>Exist</td>
</tr>
<tr>
<td>(c) Latent</td>
<td>Exist</td>
</tr>
<tr>
<td>(d) Potentially absent</td>
<td>Exist</td>
</tr>
</tbody>
</table>

Matter of requirement engineering

Our target: Are functional components covered?
Expected benefits of the approach

- Interactive process with visual support stimulates engineer’s awareness
- Mechanized interest extraction serves as the baseline of coverage standard
- Quality of validation can be improved by tuning extraction mechanism

Next development cycle
Summary - Concept of the practical approach

- Our target is latency problem
- Covering equivalence class of function
- Extract design interest: equivalence class of function and their behaviors
- Visual support enhances engineers’ awareness
1. Automotive control software development
2. Specification validation and open issues
3. Concept of the practical approach
4. **Tool implementation**
5. Application
6. Summary and future direction
Outline of current tool implementation

- A kind of dataflow graph (DFG) as one of the abstract functional model of C code
- Snapshot DFG as one of the equivalence class
- Oneshot testing which stimulate the path corresponding to each DFG
foo() {
    t = 0;
    if (P1) {
        t = u1;
    }
    if (P2) {
        t = bar(u1, u2);
    }
    if (P3) {
        t = t + 1;
    }
    _sc_ = t;
}
DFG generation algorithm

### Code

1: ...
2: t = gin
3: ...
4: If (P) {
5: ...
6: }
7: if (Q) {
8: t = t+1
9: }
10: ...
11: gout = t

### Reaching-definition

1: \{ \}  
2: \{ 2 \}  
3: \{ 2 \}  
4: \{ 2 \}  
5: \{ 2 \}  
6: \{ 2 \}  
7: \{ 2 \}  
8: \{ 8 \}  
9: \{ 2, 8 \}  
10: \{ 2, 8 \}  
11: \{ 2, 8 \}  

### Backward tracing

1: \{ \}  
2: \{ 2 \}  
3: \{ 2 \}  
4: \{ 2 \}  
5: \{ 2 \}  
6: \{ 2 \}  
7: \{ 2 \}  
8: \{ 2 \}  
9: \{ 2 \}  
10: \{ 2, 8 \}  
11: \{ 2, 8 \}  

### DFG

[Diagram of DFG]
Points unique to embedded control software

1: repetitive_task(){
2:   foo()
3:   bar()
4: }
5:
6:   foo(){
7:     if (P) {
8:       x = 0
9:     }
10:    y = x
11:  }
12:
13:   bar(){
14:     x = gin
15:     gout = y
16:  }

Slicing criteria

Inter-procedural dependence

Dependence to previous definition
Example

```c
int _sc_, gvar1, gvar2, gvar3;
int * const gvar_tbl[3] = {&gvar1, &gvar2, &gvar3};

void timed_task(void){
    foo();
    bar();
    baz();
}

void foo(void){
    gvar1 = 1;
}

void bar(void){
    int i, P, sum;
    sum = 0;
    for ( i = 0 ; i < 3 ; i++ ) {
        sum += *(gvar_tbl[i]);
    }
    _sc_ = sum;
}

void baz(void){
    gvar2 = 2;
}

void occasional_event(void){
    gvar3 = 3;
}
```

Reference to previously executed value

Order is undecidable
Model abstraction

Replace typical function patterns to compact representation:
- to omit trivial branches
- to help comprehension
- Type guard
- Absolute
- Rounding
- Max/Min
- Summation
- Cast
- …

On Control flow graph

On Dataflow graph
Snapshot DFG as a decomposed functional component

Enumerate possible dataflow patterns by taking edge combinations

Snapshot breakdown

Snapshot 1:

- A
  - e1
  - e2
- B
- C
  - e3
- D
  - e4
- Out

Condition: e1 e2

<table>
<thead>
<tr>
<th>Condition</th>
<th>e1</th>
<th>e2</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

Condition: e3 e4

<table>
<thead>
<tr>
<th>Condition</th>
<th>e3</th>
<th>e4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

Snapshot 2:

- B
  - e2
- C
  - e3
- Out

Condition: e1 e3

<table>
<thead>
<tr>
<th>Condition</th>
<th>e1</th>
<th>e3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>Q</td>
<td>F</td>
</tr>
</tbody>
</table>

Snapshot 3:

- D
  - e4
- Out

Condition: e4

<table>
<thead>
<tr>
<th>Condition</th>
<th>e4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>T</td>
</tr>
</tbody>
</table>
Meaning of the snapshot breakdown

Original C code

```c
if (P) {
    x = fun1();
} else {
    x = fun2();
}

if (Q) {
    y = fun3();
} else {
    y = fun4();
}

if (R) {
    out = x;
} else {
    out = y;
}

_sc_ = out;
```

Slicing criteria

- `out = x`
- `out = y`
- `T F`
- `x = fun1()` `P`
- `x = fun2()`
- `T F`
- `y = fun3()` `Q`
- `y = fun4()`

Doesn’t matter

Narrowing focus of interest with visual comprehension

Simplification by slicing out relevant portion

Condition “\( Q \land \neg R \)” guarantees that the particular snapshot is functioning.
Behavioral analysis for extracted snapshot

Regarding the snapshots as functional components...

<table>
<thead>
<tr>
<th>Snap</th>
<th>P ∧ R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Snap2</td>
<td>¬P ∧ R</td>
</tr>
<tr>
<td>Snap3</td>
<td>Q ∧ ¬R</td>
</tr>
<tr>
<td>Snap4</td>
<td>¬Q ∧ ¬R</td>
</tr>
</tbody>
</table>

Oneshot test to cover snapshots (ATG1)

Output

Cause analysis when fail

Outline of the function surface

Grid test in a snapshot (ATG2)

Gap detection

Dynamic test w/ snap traverse (ATG3)

Output

Coverage monitoring for closed-loop test

ATGX

Property checking in snapshots

Snap3 (Q ⊧ ¬R) ⊧ PROP

Model checker

yes

no

already implemented
Instrument C code and find inputs which passes the target path.

```c
void main (void){
    int temp;
    if ((gvar1 + gvar2) > 10) {
        temp = 1;
    } else {
        temp = 2;
    }
}
```

Ex. Target branch

(gvar1 + gvar2 <= 10)

Model checking

```
gvar1 = 2
int gvar1
int gvar2
```

Test inputs that stimulates the target branch!!

Instrumentation

```
int gvar1
int gvar2
void main (void){
    int temp;
    if ((gvar1 + gvar2) > 10) {
        temp = 1;
    } else {
        temp = 2;
    }
}
```

Ex. Target branch

(gvar1 + gvar2 <= 10)

```
assert (!(flag == 1));
flag = 1;
```  

flag becomes 1 when this branch is executed

An assertion to find counter example that falsifies “!(flag == 1)”
Prototype tool architecture

- External Model checking tool (CBMC*2)
- ATG module
- Snapshot breakdown module
- Abstraction module
- DFG generation module
- Control flow analysis module
- Data dictionary module
- C parser


*2 http://www.cprover.org/cbmc/
Example

Input: gin1, gin2
Output: _sc_

```c
1: int gin1, gin2, _sc_;  
2:  
3: void foo (void)  
4: {  
5:   int P, Q, R, x, y, out;  
6:  
7:   P = ( gin1 == 10 );  
8:   Q = ( gin1 * gin2 > 0 );  
9:   R = ( (gin1 + gin2 < 0) & (gin2 > 5) );  
10: 
11:   if (P) {  
12:     x = 1;  
13:   } else {  
14:     x = 2;  
15:   }  
16:   if (Q) {  
17:     y = 3;  
18:   } else {  
19:     y = 4;  
20:   }  
21:   if (R) {  
22:     out = x;  
23:   } else {  
24:     out = y;  
25:   }  
26: 
27:   _sc_ = out;  
28: }
```
Conflict analysis

Statistics:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>267</td>
</tr>
<tr>
<td>OK</td>
<td>21</td>
</tr>
<tr>
<td>NG</td>
<td>246</td>
</tr>
<tr>
<td>Coverage</td>
<td>7.86516853933%</td>
</tr>
</tbody>
</table>

Test generation result:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>#1_1</td>
</tr>
<tr>
<td>NG</td>
<td>#2_1</td>
</tr>
<tr>
<td></td>
<td>#3_1</td>
</tr>
<tr>
<td></td>
<td>#4_1</td>
</tr>
<tr>
<td></td>
<td>#5_1</td>
</tr>
<tr>
<td></td>
<td>#6_1</td>
</tr>
<tr>
<td></td>
<td>#7_1</td>
</tr>
<tr>
<td>OK</td>
<td>#8_1</td>
</tr>
<tr>
<td>OK</td>
<td>#9_1</td>
</tr>
</tbody>
</table>

No test input activating snap#2 was found

Step-by-step identification of root conflict by solving relaxed constraints
Example of the root conflict

Statistics:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>267</td>
</tr>
<tr>
<td>OK</td>
<td>21</td>
</tr>
<tr>
<td>NG</td>
<td>246</td>
</tr>
<tr>
<td>Coverage</td>
<td>7.86516853933%</td>
</tr>
</tbody>
</table>

Test generation result:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>655 767 819 843 853 864</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>#1_1</td>
<td>T  F  T  F  T  F  F</td>
</tr>
<tr>
<td>Conflict</td>
<td>#2_1</td>
<td>674 772 F  F</td>
</tr>
<tr>
<td>Conflict</td>
<td>#3_1</td>
<td>674 772 F  F</td>
</tr>
<tr>
<td>Conflict</td>
<td>#4_1</td>
<td>674 772 F  F</td>
</tr>
<tr>
<td>Conflict</td>
<td>#5_1</td>
<td>674 772 F  F</td>
</tr>
<tr>
<td>Conflict</td>
<td>#6_1</td>
<td>674 772 F  F</td>
</tr>
<tr>
<td>Conflict</td>
<td>#7_1</td>
<td>710 772 F  F</td>
</tr>
<tr>
<td>OK</td>
<td>#8_1</td>
<td>655 674 677 690 710 713 72</td>
</tr>
<tr>
<td>OK</td>
<td>#9_1</td>
<td>655 674 677 690 710 713 72</td>
</tr>
</tbody>
</table>

#line 674
    if ( flag == 0 ) {
#line 772
    if ( flag == 1 ) {

Existence of conflict is fine.
Unrecognized conflict is the problem.
Summary - Tool implementation

• Dataflow graph and its snapshots as one of the model of functional component

• Auto test generators for behavioral analysis

• Model checking based one shot test generator

• Tests pinpointing the particular snapshot

• Conflict analysis
1. Automotive control software development
2. Specification validation and open issues
3. Concept of the practical approach
4. Tool implementation
5. Application
6. Summary and future direction
Monitoring snapshot coverage on SILS

Hunting behavior

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input space coverage</td>
<td>66</td>
</tr>
<tr>
<td>Condition coverage</td>
<td>54</td>
</tr>
<tr>
<td>Branch coverage</td>
<td>32</td>
</tr>
<tr>
<td># of cases</td>
<td>116</td>
</tr>
</tbody>
</table>

SILS simulation

(a) (b) (c)
Paths reduction in a production code

Source Code

Abstraction (trivial branches)

\[10^{8-9}\]

27,122,688

Snapshots

Irrelevant

267

Not executable paths

Irrelevant

246

Executable paths

Irrelevant

21
Architecture analysis of large scale legacy code

- Extracted from C code: 52 files
- Controlling granularity by grouping
- Model abstraction of typical function
- 60hrs by manual analysis
1. Automotive control software development
2. Specification validation and open issues
3. Concept of the practical approach
4. Tool implementation
5. Application
6. Summary and future direction
Future direction

- **Tool implementation**
  - DFG extraction from Simulink model
  - Integration to SILS/HILS environment

- **Abstraction level of equivalence class of function**
  - Extract more essential function for larger problems
  - Other function models

- **Auto test generators**

- **Software modeling**
  - 
  - 
  - 
  -
Auto test generators

Outline of the function surface

Grid test in a snapshot (ATG2)

Dynamic test w/ snap traverse (ATG3)

Gap detection

Coverage-guided test generator for closed-loop simulator (ATGx)
Software modeling

Demand for software modeling:
- Embeddable (can generate C code)
- Helps intuitive understanding of equivalence class of function
- Separation of concern
  - Implementation details \( \Leftrightarrow \) Essential function
- Unique (no manual synchronization among models)

Centralized mode control

Distributed mode control

+ Clear mode of operation
- Redundant description

+ Compact description
- Ambiguous mode of operation

Is it possible to describe as a static model??
End.