Lessons Learned from Automated Analysis of Industrial SW for 15 Year

Moonzoo Kim

Software Testing and Verification Group

KAIST, South Korea

http://swtv.kaist.ac.kr

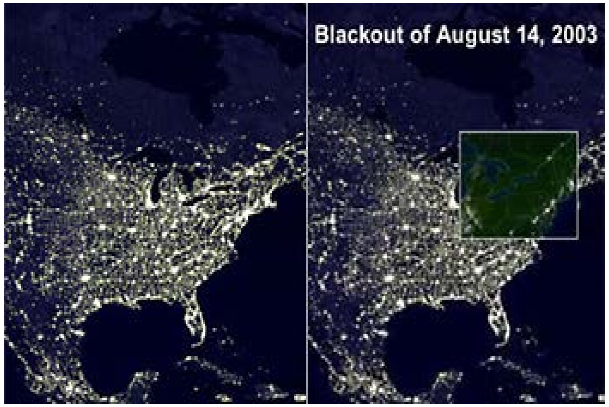
Thanks to Hotae Kim, Youil Kim, Sacheon Park, Yoonkyu Jang, Dongju Lee, Yunseok Choi, Joohyun Lee, Junki Baek and more @ IT industry in South Korea

Part I: Motivation of Applying Formal Techniques to Industries
Social and Economic Loss due to **High Complexity of SW**

Although most areas of modern society depend on SW,
reliability of SW is not improved much due to its high complexity

(1980s)Medical accident: Therac 25

- For 1985-1987, excessive radio reactive beam enforced.
- 6 persons died due to the problem
- Data race bug was the cause of the problem



(2003) US & Canada Blackout

- 7 states in US and 1 state in Canada suffered 3 days electricity blackout
- Caused by the failures of MISO monitoring SW
- 50 million people suffered and economic loss of 6 billion USD

(2010s) Toyoda SUA (sudden unintended accelaration)

- Dozens of people died since 2002
- SW bugs detected in 2012
- Fined 1.2 billion USD in 2014

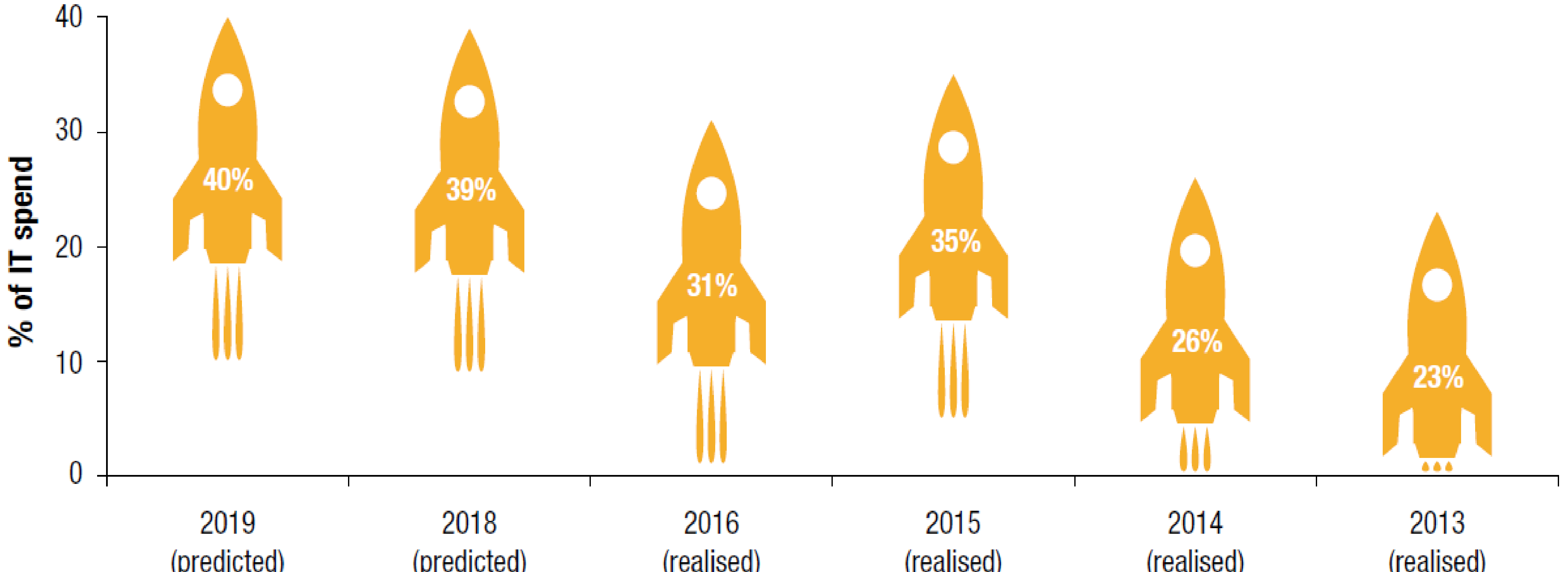
SW Testing is a Complex and Challenging task!!!

Object-Oriented Programming, Systems Languages, and Applications, Seattle, Washington, November 8, 2002

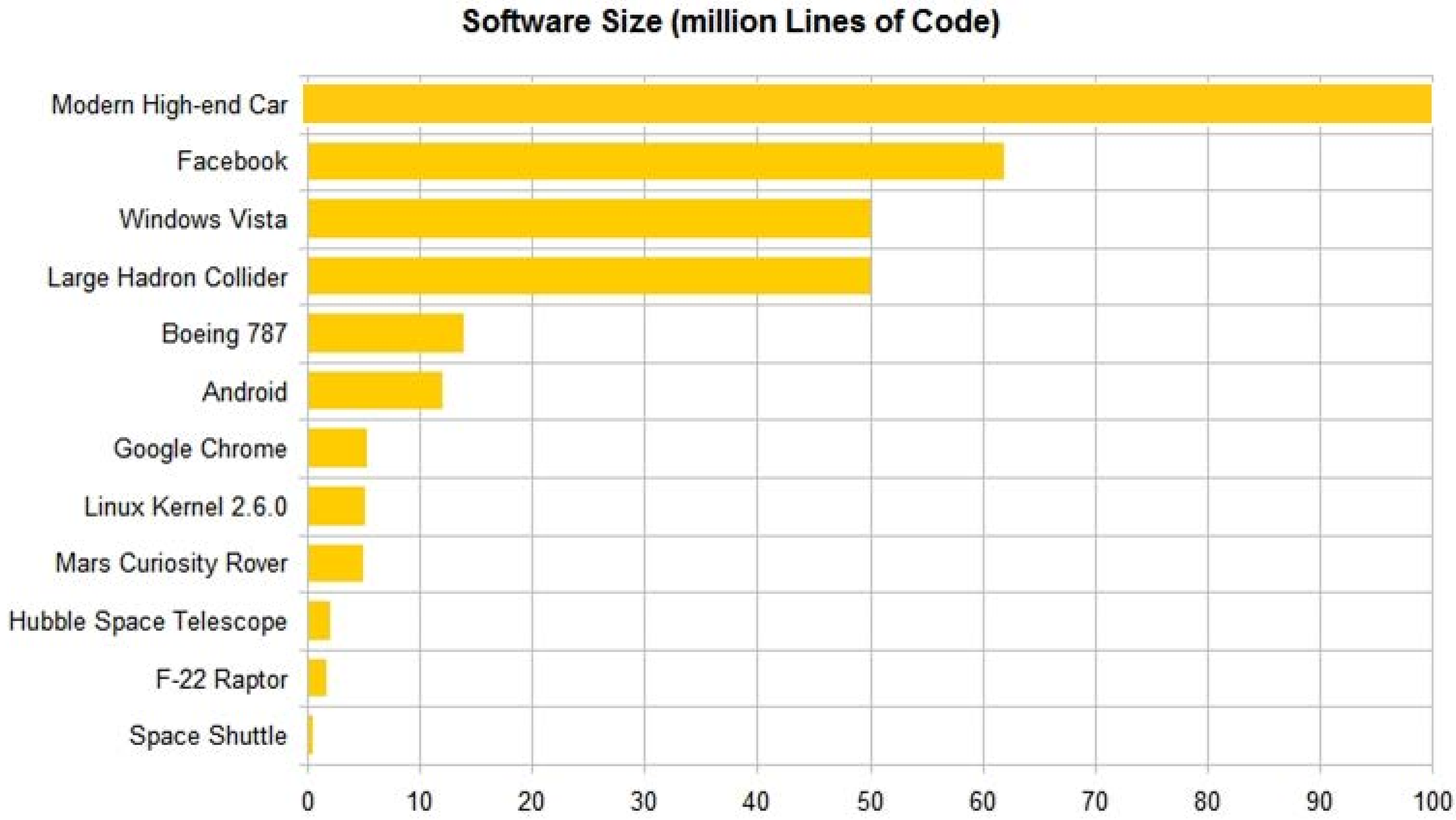
- "… When you look at a big commercial software company like Microsoft, there's actually as much testing that goes in as development. We have as many testers as we have developers. Testers basically test all the time, and developers basically are involved in the testing process about half the time…"
- "… We've probably changed the industry we're in. We're not in the software industry; we're in the testing industry, and writing the software is the thing that keeps us busy doing all that testing."
- "…The test cases are unbelievably expensive; in fact, there's more lines of code in the test harness than there is in the program itself. Often that's a ratio of about three to one."

SW Verification & Testing Market Trends

- SW verification and testing market: 19.3 Million USD @ 2015, annual growth: 15% (expected) [IDC]
- 31% of total expenses of IT companies is due to QA and SW testing, increasing to 40% (expected)
  [World Quality Report 2016-2017]



Size and Complexity of Modern SW



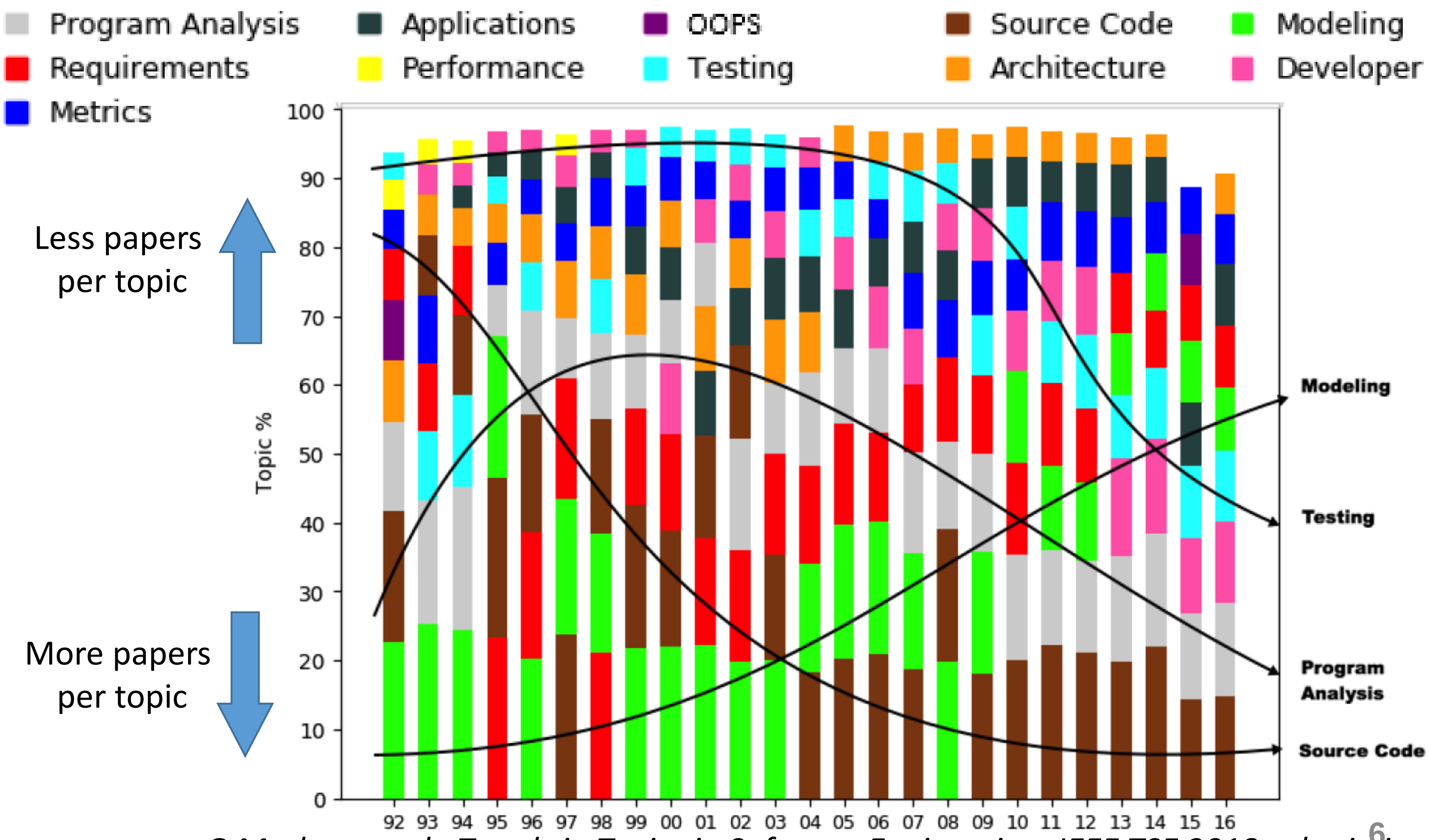
A.Busnelli, Counting, https://www.linkedin.com/pulse/20140626152045-3625

http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

SE Research Topic Trends among 11 Major Topics (1992-2016)

Less papers per topic

More papers per topic



G.Mathew et al., Trends in Topics in Software Engineering, IEEE TSE 2018 submission

Limitations of the Conventional Quality Assurance Techniques

- Most QA techniques in industry practice target easy-to-detect bugs at low cost (i.e., light-weight approaches)
  - Secure coding rule checkers:
    - MISRA C, CERT C Coding standard, CERT Oracle secure coding standard
  - Static analyzer:
    - Coverity, Sparrow, Code sonar, Findbugs
  - Various random + heuristic test case generation techniques
    - Suresoft codescroll
- Such techniques often fail to detect hard-to-detect corner case bugs because
  - These techniques focus to analyze the syntax (i.e., pattern/structure) of a target program
  - However, the difficulty/complexity is caused by the semantics (i.e., meaning) of a target program.

SOFTWARE CAUSES OF MEMORY CORRUPTION

Type of Software Defect	Causes Memory Corruption?	Defect in 2005 Camry L4?
Buffer Overflow	Yes	Yes
Invalid Pointer Dereference/Arithmetic	Yes	Yes
Race Condition (a.k.a., "Task Interference")	Yes	Yes
Nested Scheduler Unlock	Yes	Yes
Unsafe Casting	Yes	Yes
Stack Overflow	Yes	Yes

Static analysis falls short of detecting such complex bugs accurately

- High false negatives
- High false positives

⇒ Systematic and dynamic analysis is MUST for high quality SW

Significance of Automated SW Analysis

- Software has become more ubiquitous and more complex at the same time
- Human resources are becoming less reliable and more expensive for testing highly complex software systems
- Computing resources are becoming ubiquitous and free
  - Tencent @ China provides 10TB storage free
  - Amazon AWS price: you can use thousands of CPUs at 0.05 $/hr for 3.2Ghz Quad-core CPU
- To-do:
  - To develop automated and scientific software analysis tools to utilize computing resource effectively and efficiently

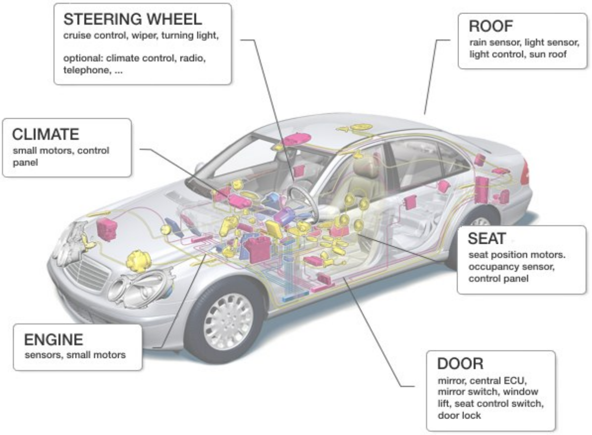
Strong IT Industry in South Korea

SAMSUNG

SK hynix

KIA

KIA MOTORS



STEERING WHEEL
cruise control, wiper, turning light,
optional: climate control, radio, telephone, ...

ROOF
rain sensor, light sensor, light control, sun roof

CLIMATE
small motors, control panel

SEAT
seat position motors. occupancy sensor, control panel

ENGINE
sensors, small motors

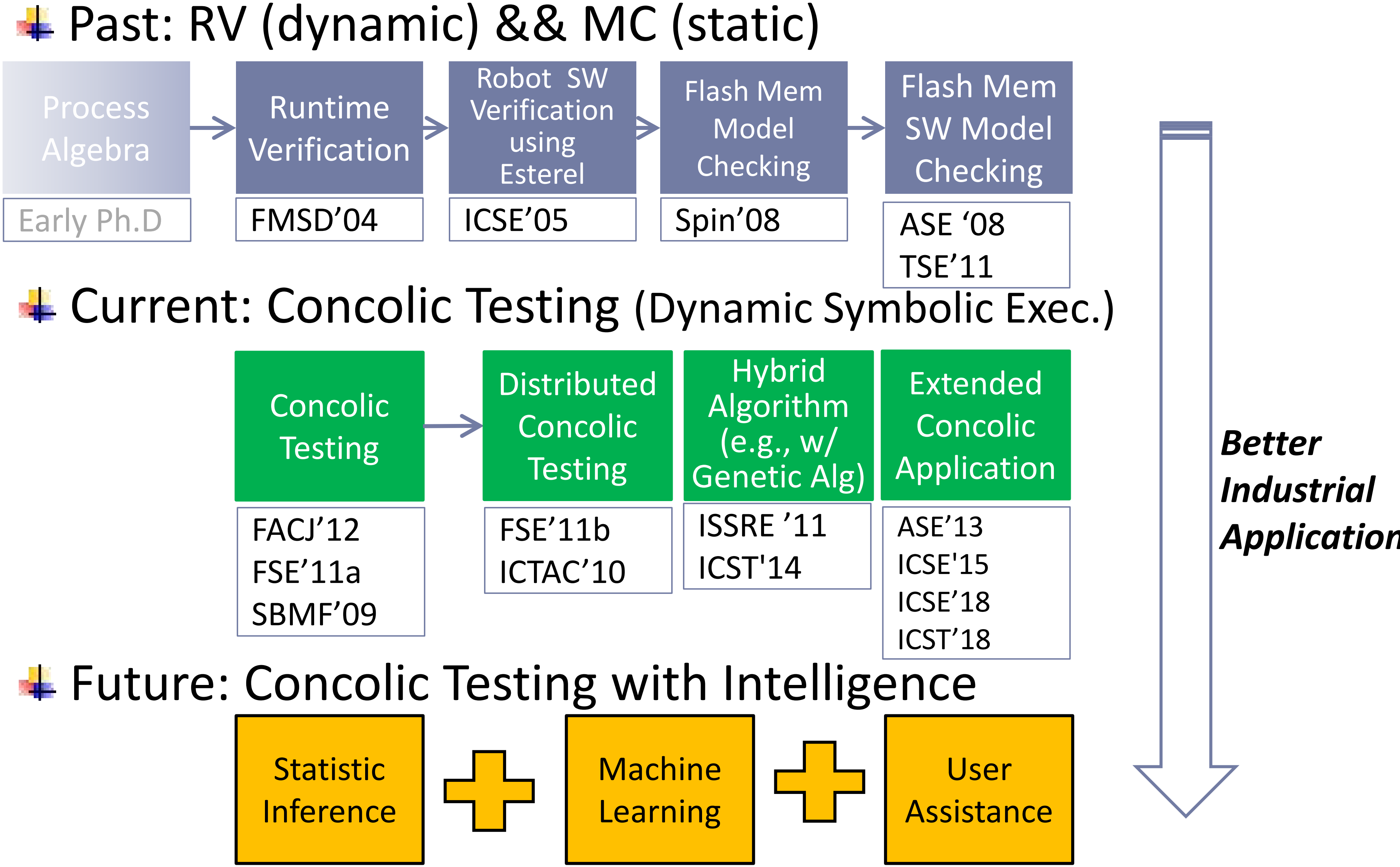
DOOR
mirror, central ECU, mirror switch, window lift, seat control switch, door lock

LG

MOBIS

HYUNDAI

Personal Research Roadmap



- Past: RV (dynamic) && MC (static)

Process Algebra → Runtime Verification → Robot SW Verification using Esterel → Flash Mem Model Checking → Flash Mem SW Model Checking

Process Algebra	Runtime Verification	Robot SW Verification using Esterel	Flash Mem Model Checking	Flash Mem SW Model Checking
Early Ph.D	FMSD'04	ICSE'05	Spin'08	ASE '08 TSE'11

- Current: Concolic Testing (Dynamic Symbolic Exec.)

Concolic Testing → Distributed Concolic Testing

Concolic Testing	Distributed Concolic Testing	Hybrid Algorithm (e.g., w/ Genetic Alg)	Extended Concolic Application
FACJ'12 FSE'11a SBMF'09	FSE'11b ICTAC'10	ISSRE '11 ICST'14	ASE'13 ICSE'15 ICSE'18 ICST'18

- Future: Concolic Testing with Intelligence

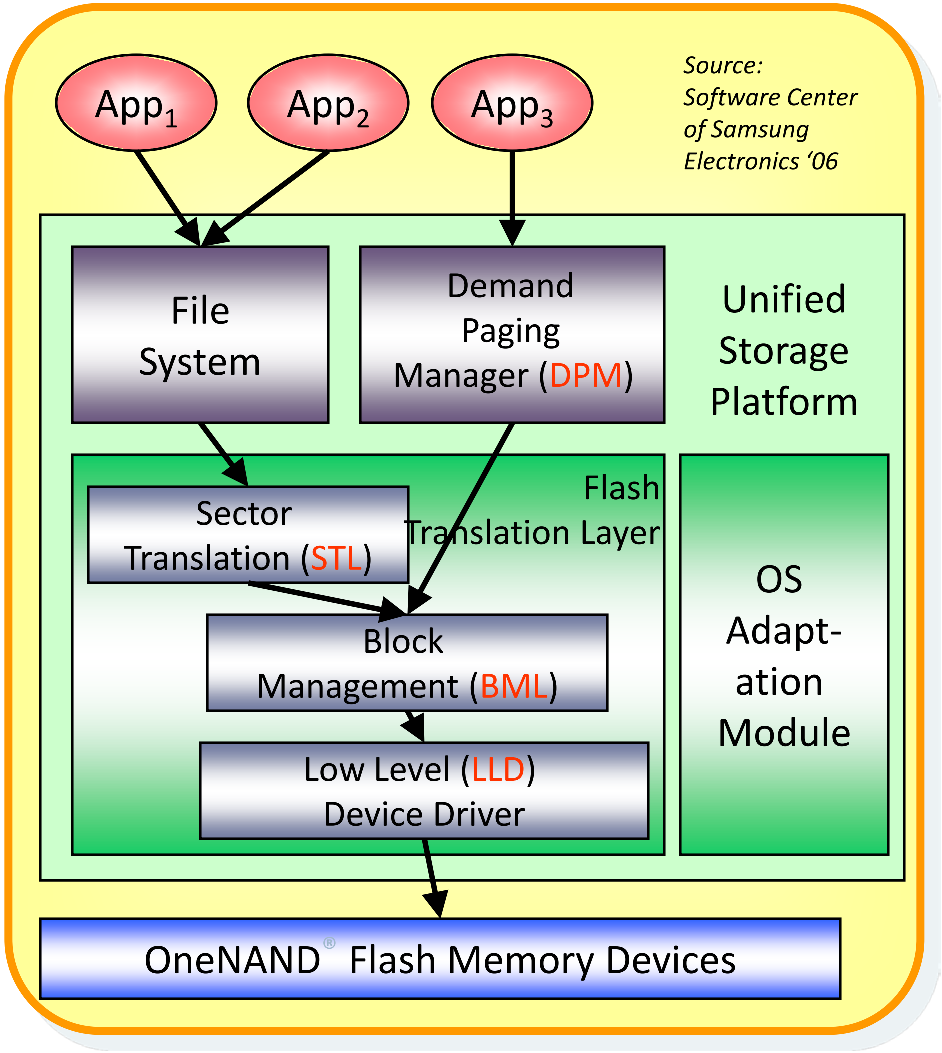
Statistic Inference + Machine Learning + User Assistance

Better Industrial Application

Part II: Industrial Experience w/ Model Checking

Target system: Samsung Unified Storage Platform (USP) for OneNAND® flash memory (around 30K lines of C code)

- Characteristics of OneNAND® flash mem
  - Each memory cell can be written limited number of times only
    - Logical-to-physical sector mapping
    - Bad block management, wear-leveling, etc
  - Concurrent I/O operations
    - Synchronization among processes is crucial
  - XIP by emulating NOR interface through demand-paging scheme
    - binary execution has a highest priority
  - Performance enhancement
    - Multi-sector read/write
    - Asynchronous operations
    - Deferred operation result check



Source: Software Center of Samsung Electronics '06

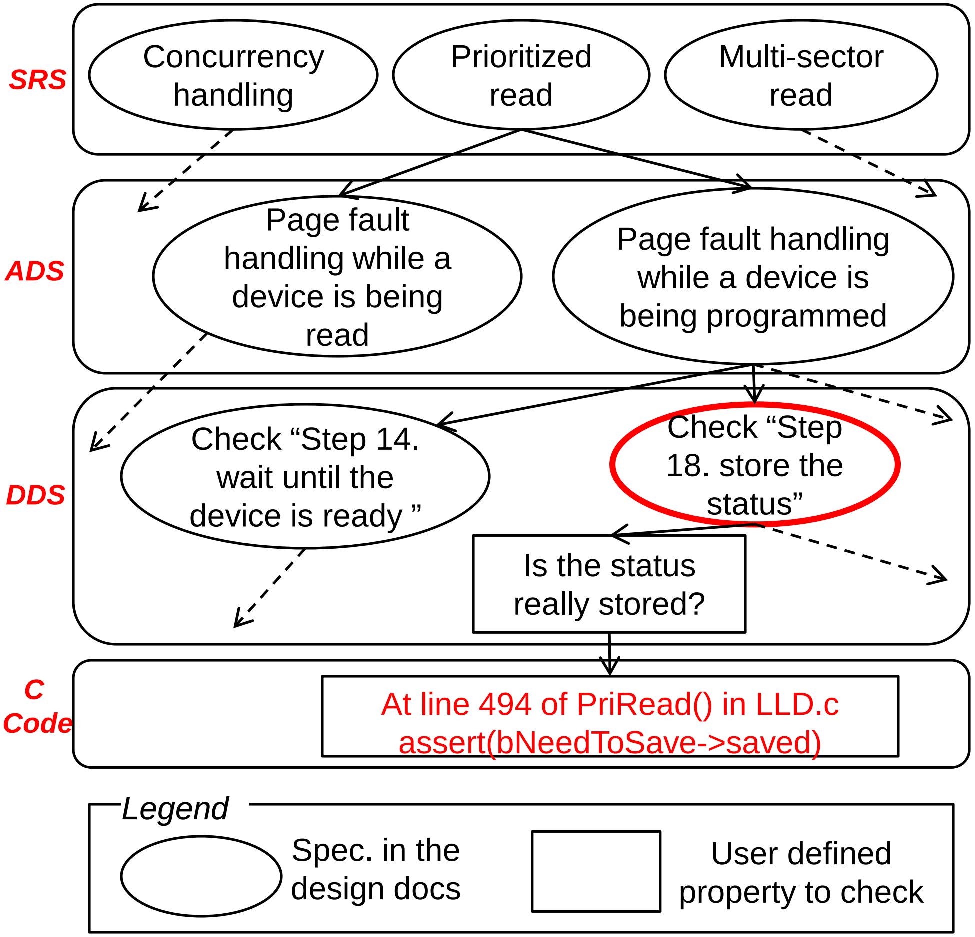
Project Overview

- The goal of the project
  - To check whether USP conforms to the given high-level requirements
    - we needed to identify the code-level properties to check from the given high-level requirements
- A top-down approach to identify the code level properties from high-level requirements
  - USP has a set of elaborated design documents
    - Software requirement specification (SRS)
    - Architecture design specification (ADS)
    - Detailed design specification (DDS)
      - DPM, STL, BML, and LLD

Three High-level Requirements in SRS

- SRS specifies 13 functional requirements, 3 of which have “very high” priorities
  - Support prioritized read operation
    - To minimize the fault latency, USP should serve a read request from DPM prior to generic requests from a file system.
    - This prioritized read request can preempt a generic I/O operation and the preempted operation can be resumed later.
  - Concurrency handling
    - BML and LLD should avoid a race condition or deadlock through synchronization mechanisms such as semaphores and locks.
  - Manage sectors
    - STL provides logical-to-physical mapping, i.e. multiple logical sectors written over the distributed physical sectors should be read back correctly.

Top-down Approach to Identify Code-level Property



SRS

Concurrency handling

Prioritized read

Multi-sector read

ADS

Page fault handling while a device is being read

Page fault handling while a device is being programmed

DDS

Check "Step 14. wait until the device is ready "

Check "Step 18. store the status"

Is the status really stored?

C Code

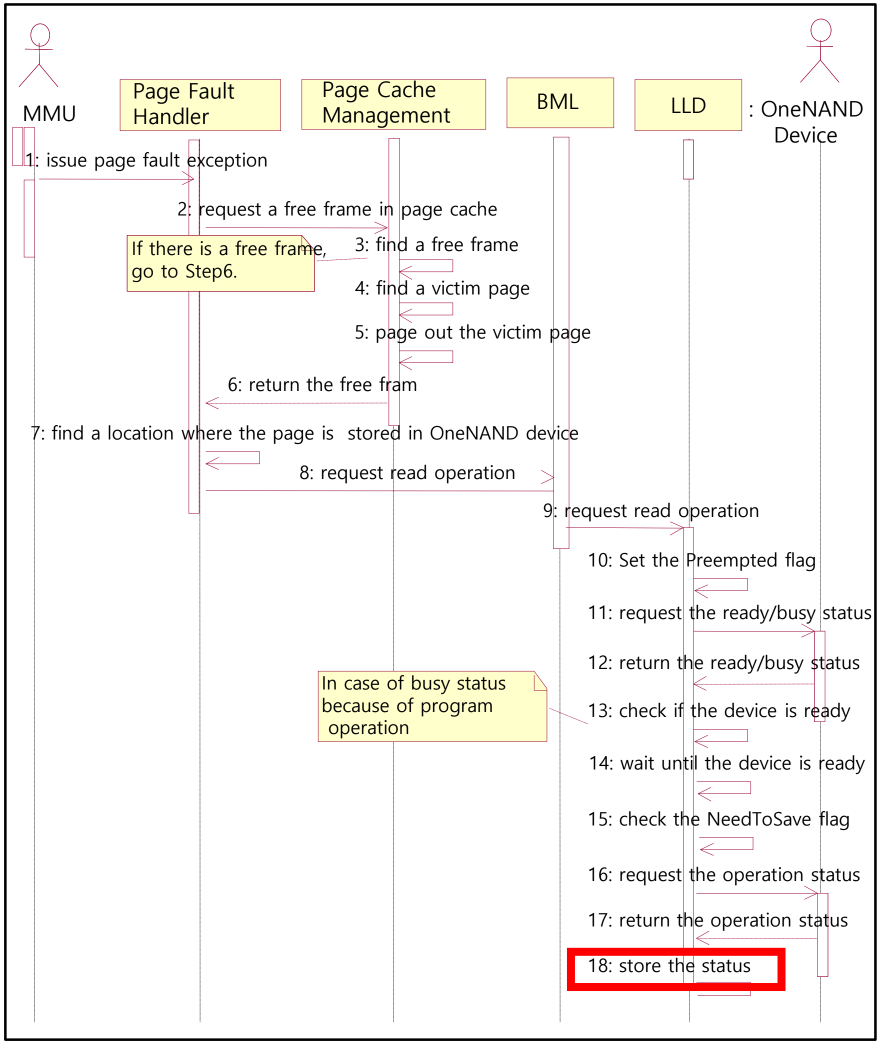
At line 494 of PriRead() in LLD.c
assert(bNeedToSave->saved)

Legend

Spec. in the design docs

User defined property to check

- Total 43 code-level properties are identified



MMU

Page Fault Handler

Page Cache Management

BML

LLD

: OneNAND Device

1: issue page fault exception

2: request a free frame in page cache

If there is a free frame, go to Step6.

3: find a free frame

4: find a victim page

5: page out the victim page

6: return the free fram

7: find a location where the page is stored in OneNAND device

8: request read operation

9: request read operation

10: Set the Preempted flag

11: request the ready/busy status

12: return the ready/busy status

In case of busy status because of program operation

13: check if the device is ready

14: wait until the device is ready

15: check the NeedToSave flag

16: request the operation status

17: return the operation status

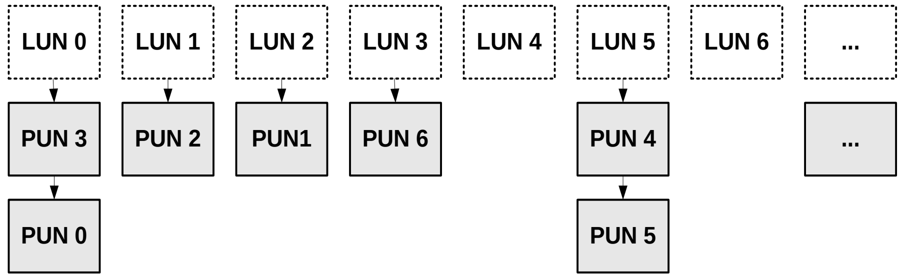
18: store the status

A sequence diagram of page fault handling while a device is being programmed in LLD DDS

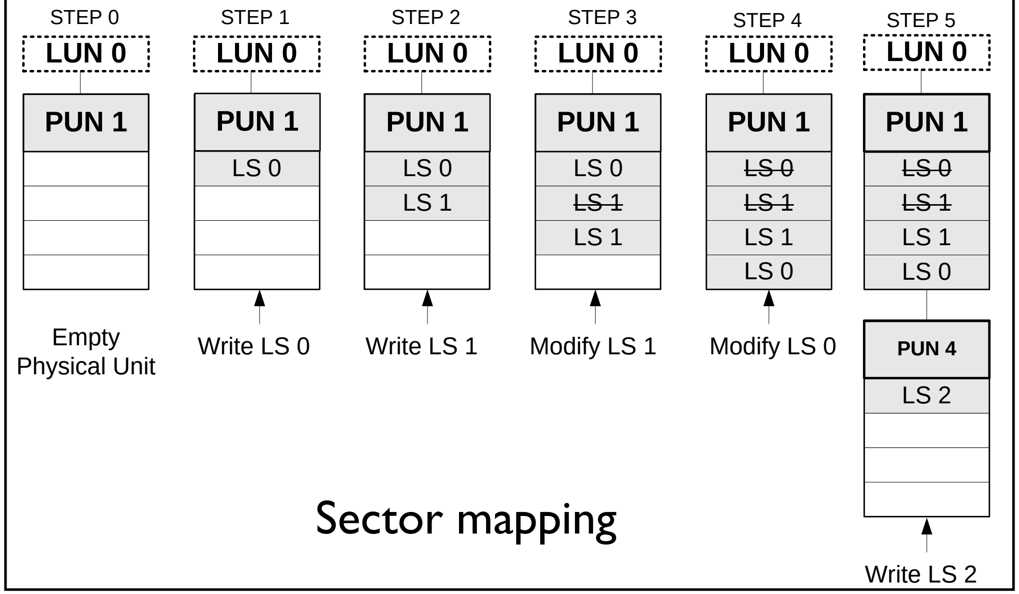
Results of Applying CBMC and BLAST [TSE'11]

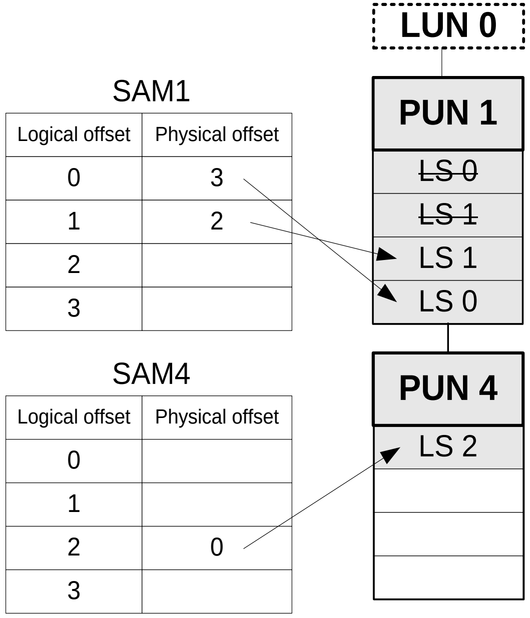
- Demand paging manager (234 LOC)
  - Detected a bug of not saving the status of suspended erase operation
- Concurrency handling
  - Confirmed that the BML semaphore was used correctly in all 14 BML functions (150 LOC on average)
  - Detected a bug of ignoring BML semaphore exceptions in a call sequence from STL (2500 LOC on average)
- Multi-sector read operation (MSR) (157 LOC)
  - Provided high assurance on the correctness of MSR
    - no violation was detected even after exhaustive analysis (at least with a small number of physical units(~10))
- In addition, we evaluated and compared pros and cons of CBMC and BLAST empirically

Logical to Physical Sector Mapping



1:N mapping from a LUN to PUNs





SAM1

Logical offset	Physical offset
0	3
1	2
2	
3	

SAM4

Logical offset	Physical offset
0	
1	
2	0
3	

Sector Allocation Map (SAM)

- In flash memory, logical data are distributed over physical sectors.

Multi-sector Read Operation (MSR)

	SAM0~SAM4					PU0~PU4				
Sector 0	1			0					E	
Sector 1		1			1	A	B			F
Sector 2		2					C			
Sector 3			3					D		

a) A distribution of "ABCDEF"

SAM0~SAM4					PU0~PU4				
	3			3	B				
0			2			D			
		3						F	
	1					A	C		E

b) Another distribution of "ABCDEF"

SAM0~SAM4					PU0~PU4				
	0							E	
	1				A	B			F
	2					C			
	3						D		

c) Invalid distribution of "ABCDEF"

- MSR reads adjacent multiple physical sectors once in order to improve read speed
  - MSR is 157 lines long, but highly complex due to its 4 level loops
  - 4 parameters to specify logical data to read (from, to, how long, read flag)
- The requirement property is to check
  - after_MSR -> (∀ i. logical_sectors[i] == buf[i])
- We built a **verification environment model** for MSR

Environment Modeling

1. **One PU is mapped to at most one LU**
2. **Valid correspondence between SAMs and PUs:**

   If the *i* th LS is written in the *k* th sector of the *j* th PU, then the *i* th offset of the *j* th SAM is valid and indicates the k'th PS ,

   Ex> 3rd LS ('C') is in the 3rd sector of the 2nd PU, then SAM1[2] ==2

   i=2 k=2 j=1

3. **For one LS, there exists only one PS that contains the value of the LS:**

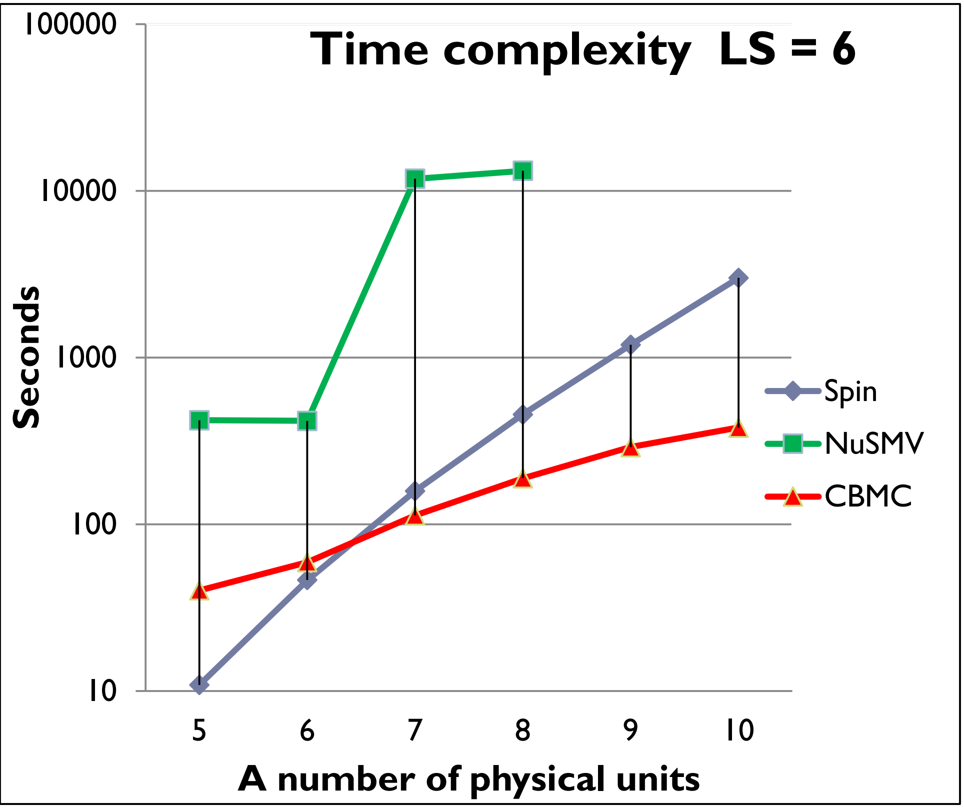
   The PS number of the *i* th LS must be written in only one of the (*i* mod *4*) th offsets of the SAM tables for the PUs mapped to the corresponding LU.

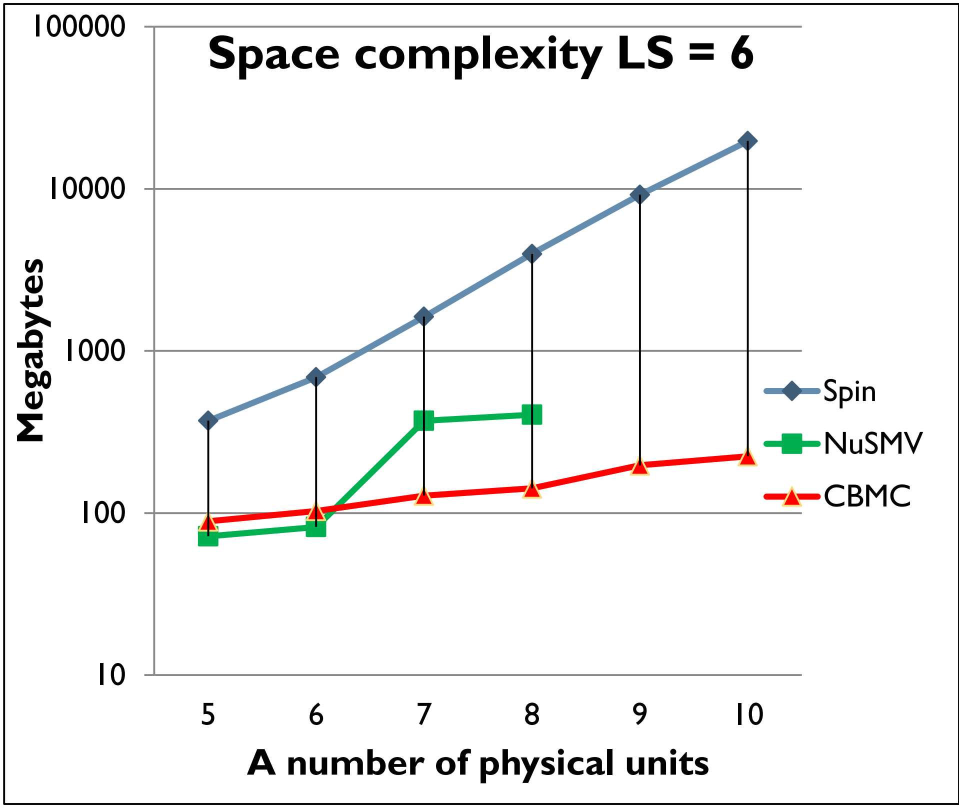
$$
\begin{aligned}
\forall i, j, k\ (LS[i] = PU[j].sect[k] \rightarrow (&SAM[j].valid[i\ mod\ m] = true \\
&\&\ SAM[j].offset[i\ mod\ m] = k \\
&\&\ \forall p.(SAM[p].valid[i\ mod\ m] = false) \\
&\text{where } p \neq j \text{ and } PU[p] \text{ is mapped to} \lfloor \frac{i}{m} \rfloor_{th}\ LU))
\end{aligned}
$$

	SAM0~SAM4					*PU0~PU4*				
Sector 0	1			0					E	
Sector 1		1			1	A	B			F
Sector 2		2					C			
Sector 3			3					D		

Model Checking Results of MSR [Spin'08]

- Verification of MSR by using NuSMV, Spin, and CBMC
- No violation was detected within |LS|<=8, |PU| <=10
  - 10^{10} configurations were exhaustively analyzed for |LS|=8, |PU|=10





Feedbacks from Samsung Electronics

Main challenge :

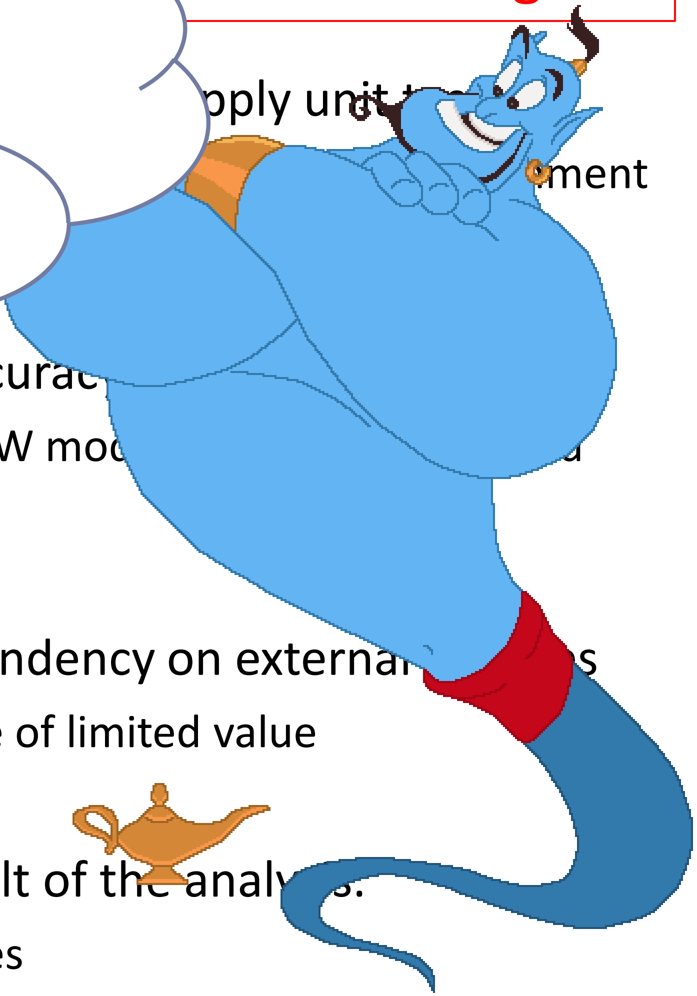
- IT indus luct unit testing

Concolic testing

can resolve most

of the problems ♬

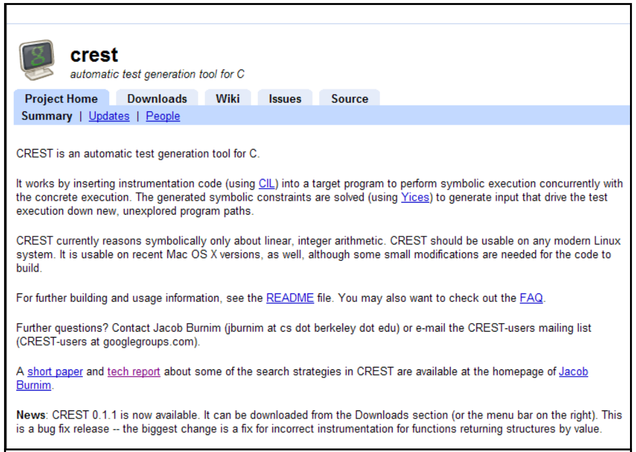
1. Curren pply unit t
   - Tight r ment
     mode
2. Needs large scalability even at the cost of accurac
   - Rigorous automated tools for small unit (i.e., SW mod d
     practical value
3. Many embedded SW components have dependency on external s
   - Pure analysis methods on source code only are of limited value
4. It is desirable to generate test cases as a result of the analy s.
   - Current SW V&V practice operates on test cases



Background on Concolic Testing

- **Conc**rete runtime execution guides symb**olic** path analysis
  - a.k.a. dynamic symbolic execution (DSE), white-box fuzzing
- Automated test case (TC) generation technique
  - Applicable to a large target program (no memory bottleneck)
  - Applicable to testing stages seamlessly
  - External binary library can be handled (partially)
- Explicit path model checker
  - All possible execution paths are explored based on the generated TCs
  - Anytime algorithm
    - User can get partial analysis result (i.e., TCs) anytime
  - Analysis of each path is independent from each other
    - Parallelization for linear speed up
    - Ex. Scalable COncolic testing for Reliable Embedded Software (SCORE) on thousands of Amazon EC2 cloud computing nodes [FSE'11b]

Concolic Testing Tools



crest

automatic test generation tool for C

CREST is an automatic test generation tool for C.

It works by inserting instrumentation code (using CIL) into a target program to perform symbolic execution concurrently with the concrete execution. The generated symbolic constraints are solved (using Yices) to generate input that drive the test execution down new, unexplored program paths.

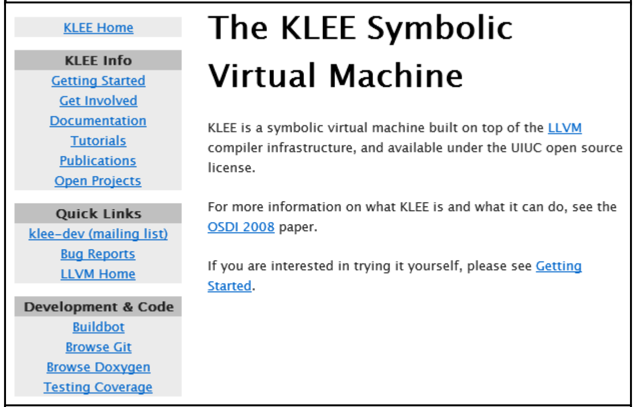
CREST currently reasons symbolically only about linear, integer arithmetic. CREST should be usable on any modern Linux system. It is usable on recent Mac OS X versions, as well, although some small modifications are needed for the code to build.

For further building and usage information, see the README file. You may also want to check out the FAQ.

Further questions? Contact Jacob Burnim (jburnim at cs dot berkeley dot edu) or e-mail the CREST-users mailing list (CREST-users at googlegroups.com).

A short paper and tech report about some of the search strategies in CREST are available at the homepage of Jacob Burnim.

News: CREST 0.1.1 is now available. It can be downloaded from the Downloads section (or the menu bar on the right). This is a bug fix release -- the biggest change is a fix for incorrect instrumentation for functions returning structures by value.

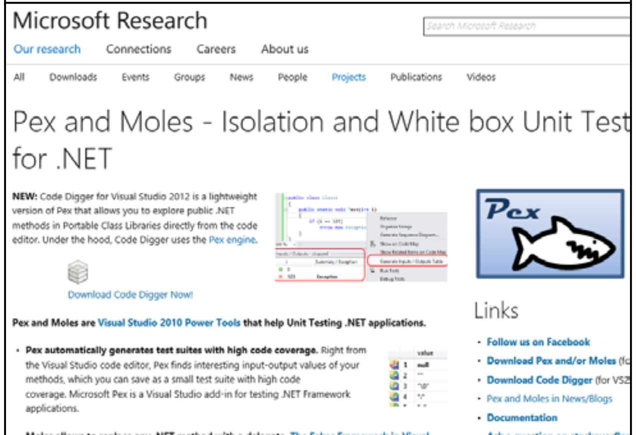


The KLEE Symbolic Virtual Machine

KLEE is a symbolic virtual machine built on top of the LLVM compiler infrastructure, and available under the UIUC open source license.

For more information on what KLEE is and what it can do, see the OSDI 2008 paper.

If you are interested in trying it yourself, please see Getting Started.



Pex and Moles - Isolation and White box Unit Test for .NET

NEW: Code Digger for Visual Studio 2012 is a lightweight version of Pex that allows you to explore public .NET methods in Portable Class Libraries directly from the code editor. Under the hood, Code Digger uses the Pex engine.

Download Code Digger Now!

Pex and Moles are Visual Studio 2010 Power Tools that help Unit Testing .NET applications.

- **Pex automatically generates test suites with high code coverage.** Right from the Visual Studio code editor, Pex finds interesting input-output values of your methods, which you can save as a small test suite with high code coverage. Microsoft Pex is a Visual Studio add-in for testing .NET Framework applications.

Links

- Follow us on Facebook
- Download Pex and/or Moles (fc
- Download Code Digger (for VS2
- Pex and Moles in News/Blogs
- Documentation

- CROWN (open source)
  - Target: C
  - Instrumentation based extraction
  - BV supported
  - https://github.com/swtv-kaist/CROWN
- KLEE (open source)
  - Target: LLVM
  - VM based symbolic formula extraction
  - BV supported
  - Symbolic POSIX library supported
  - http://ccadar.github.io/klee/
- PEX (IntelliTest incorporated in Visual Studio 2015)
  - Target: C#
  - VM based symbolic formula extraction
  - BV supported
  - Integrated with Visual Studio
  - http://research.microsoft.com/en-us/projects/pex/
- CATG (open source)
  - Target: Java
  - Trace/log based symbolic formula extraction
  - LIA supported

Part III. Industrial Experience w/ Concolic Testing

- System level testing
  1. Samsung Linux Platform (SLP) file manager [FSE'11]
     - Covered 20% of the branches and detected an infinite loop bug
  2. 10 Busybox utilities
     - Covered 80% of the branches with 4 different search strategy and 10,000 TCs in 20 min each (a buffer overflow bug in grep was detected)
  3. Libexif [ICSE'12]
     - Covered 43% of the branches with 4 different search strategy and 10,000 TCs in 8 hours each
     - 2 null pointer dereferences and 5 divide-by-0 bugs were detected
     - Reported 2 security bugs to Common Vulnerability Exposure (CVE)
  4. Failure to control LG home appliances [ICSE'2015 SEIP]
  5. Hyundai automotive software (2015- 2017) : inconsistency between spec. and impl. found

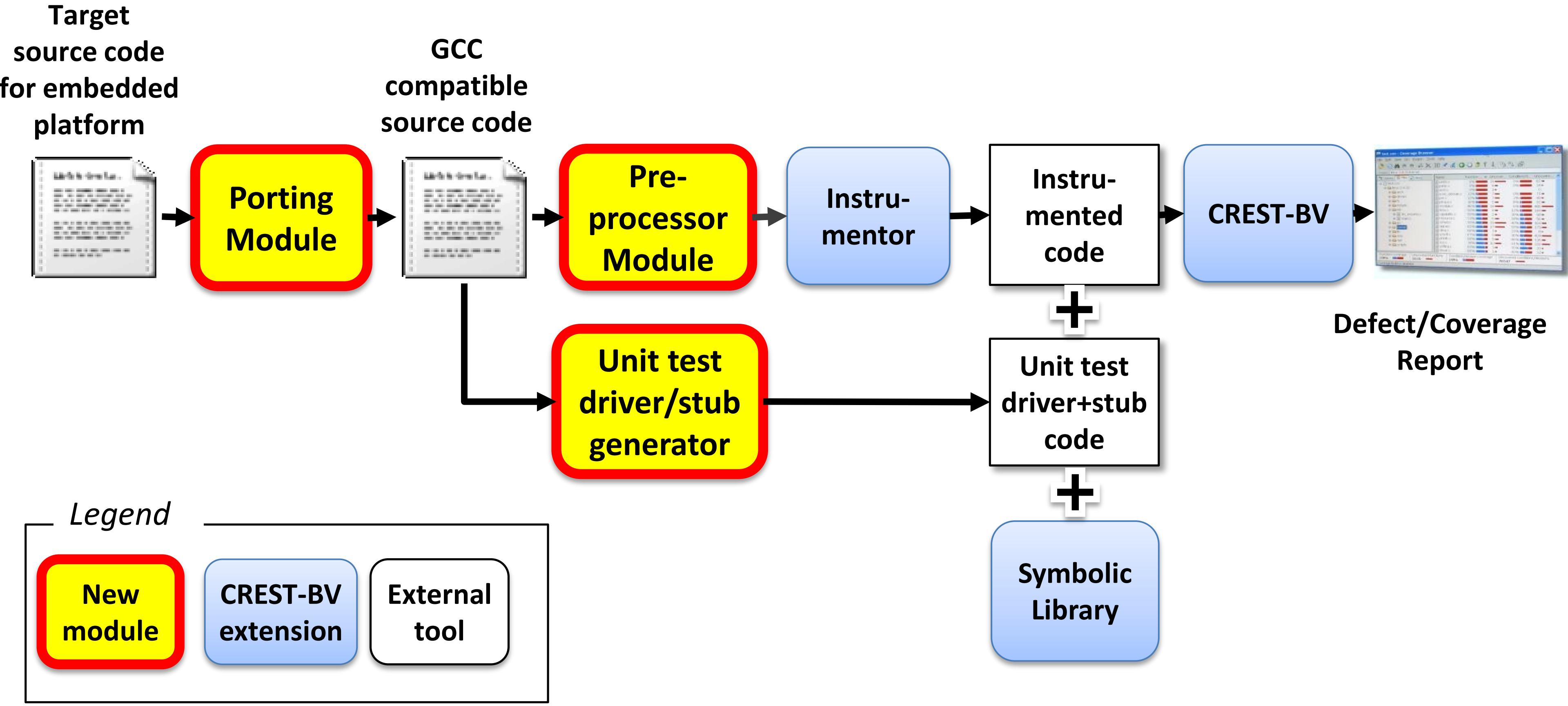
- Unit-level testing
  1. Busybox ls (1100 LOC) [ICST'12]: 98% of branches covered and 4 bugs detected
  2. Samsung security library (2300 LOC) [ICST'12]
     - 73% of branches covered and a memory violation bug detected
  3. **CONcrete and symBOLic (CONBOL) framework [ASE'13]**
     - **Detected 24 crash bugs in 4 MLOC embedded program**
     - **Detected 6 crash bugs in 0.2 MLOC in another embedded program**
  4. Automated unit testing framework for Hyundai Mobis automotive software (2017 - now):
     - reducing manual testing time more than 50% (= 0.5 million USD per year @ Mobis India) [ICSE'19 submitted]

Summary of the Case Study

- Embedded SW is becoming larger and more complex
  - Ex. Android: 12 MLOC, Tizen > 6 MLOC
- Smartphone development period is very short
  - No time to manually test smartphones sufficiently
- Solution: Automated unit test generation for industrial embedded SW using CONBOL (CONcrete and symBOLic testing)
  - CONBOL automatically generates unit-test driver/stubs
  - CONBOL automatically generates test cases using concolic testing
  - CONBOL targets crash bugs (i.e. null pointer dereference, etc.)
- CONBOL detected 24 crash bugs in 4 MLOC Android SW in 16 hours

Overview of CONBOL

- We have developed the CONcrete and symBOLic (CONBOL) framework: an automated concolic unit testing tool based-on CREST-BV for embedded SW



Target source code for embedded platform

Porting Module

GCC compatible source code

Pre-processor Module

Instru-mentor

Instru-mented code

CREST-BV

Defect/Coverage Report

Unit test driver/stub generator

+

Unit test driver+stub code

+

Symbolic Library

Legend

New module

CREST-BV extension

External tool

Unit Test Driver/Stub Generator(1/2)

- The unit test driver/stub generator **automatically generates unit test driver/stub** functions for unit testing of a target function
  - A unit test driver symbolically sets all visible global variables and parameters of the target function

Type	Description	Code Example

Primitive	set a corresponding symbolic value	int a;
		SYM_int(a);

Array	set a fixed number of elements	int a[3];
		SYM_int(a[0]); … SYM_int(a[2]);

Structure	set NULL to all pointer fields and set symbolic value to all primitive fields	struct _st{int n,struct _st*p}a;
		SYM_int(a.n);
		a.p=NULL;

Pointer	allocate memory for a pointee and set a symbolic value of corresponding type of the pointee	int *a;
		a = malloc(sizeof(int));
		SYM_int(*a);

  - The test driver/stub generator replaces sub-functions invoked by the target function with symbolic stub functions

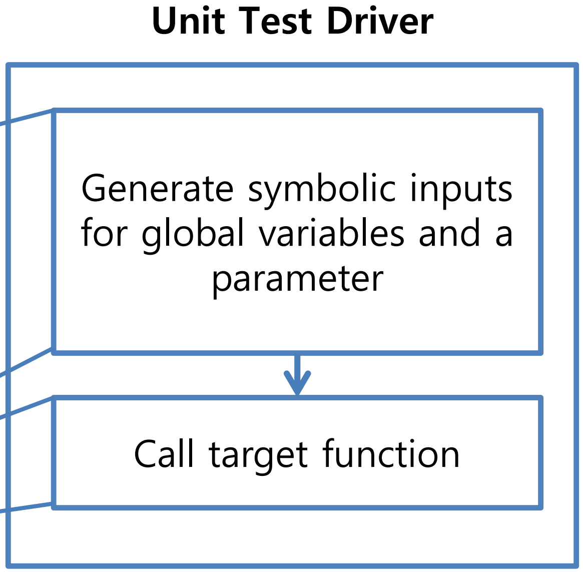
Unit Test Driver/Stub Generator(2/2)

- Example of an automatically generated unit-test driver

```
typedef struct Node_{
  char c;
  struct Node_ *next;
} Node;
Node *head;
// Target unit-under-test
void add_last(char v){
  // add a new node containing v
  // to the end of the linked list
  ...}
// Test driver for the target unit
void test_add_last(){
  char v1;
  head = malloc(sizeof(Node));
  SYM_char(head->c);
  head->next = NULL;
  SYM_char(v1);
  add_last(v1); }
```

Set global variables

Set parameter



Unit Test Driver

Generate symbolic inputs for global variables and a parameter

Call target function

Statistics of Project S

- Project S, our target program, is an industrial embedded software for smartphones developed by Samsung Electronics
  - Project S targets ARM platforms

Metric		Data
Total lines of code		About 4,000,000
# of branches		397,854
# of functions	Total	48,743
	Having more than one branch	29,324
# of files	Sources	7,243
	Headers	10,401

Test Experiment Setting

- CONBOL uses a DFS strategy used by CREST-BV in Kim et al. [ICSE12 SEIP]
- Termination criteria and timeout setting
  - Concolic unit testing of a target function terminates when
    - CONBOL detect a violation of an assertion, or
    - All possible execution paths are explored, or
    - Concolic unit testing spends 30 seconds (Timeout1)
  - In addition, a single test execution of a target unit should not spend more than 15 seconds (Timeout2)
- HW setting
  - Intel i5 3570K @ 3.4 GHz, 4GB RAM running Debian Linux 6.0.4 32bit

Results (1/2)

- Results of branch coverage and time cost
  - CONBOL tested **86.7%(=25,425)** of target functions on a host PC
    - 13.3% of functions were not inherently portable to a host PC due to inline ARM assembly, direct memory access, etc
  - CONBOL covered **59.6%** of branches in **15.8 hours**

Statistics	Number
Total # of test cases generated	About 800,000
Branch coverage (%)	59.6
Execution time (hour)	15.8
# of functions reaching timtout1 (30s)	742
# of functions reaching timtout2 (15s)	134
Execution time w/o timeout (hour)	9.0

Results (2/2)

- CONBOL raised 277 alarms
- 2 Samsung engineers (w/o prior knowledge on the target program) took 1 week to remove 227 false alarms out of 277 alarms
  - We reported 50 alarms and **24 crash bugs** were confirmed by the developers of Project S
- Pre-conditions and scoring rules filtered out **14.1% and 81.2%** of likely false alarms, respectively
- Note that Coverity prevent could **not** detect any of these crash bugs

# of reported alarms	**Out-of-bound**		**NULL-pointer-dereference**		**Divide-by-zero**		**Total**	
	# of alarms	Ratio (%)	# of alarms	Ratio (%)	# of alarms	Ratio (%)	# of alarms	Ratio (%)
W/O any heuristics	3235	100.0	2588	100.0	61	100.0	5884	100.0
W/ inserted pre-conditions	2486	76.8	2511	97.0	58	95.1	5055	85.9
W/ inserted pre-conditions + scoring rules	220	6.8	42	1.6	15	24.6	277	4.7
Confirmed and fixed bugs	**13**	**0.4**	**5**	**0.2**	**6**	**9.8**	**24**	**0.4**

Concolic Testing w/ Samsung for 5 Years

- Goal: Developing **automated test generation technique** based on concolic testing to **effectively detect bugs** in **embedded C programs** developed by Samsung
- Project period: 2010 ~ 2014
- Outcome:
  - Tool development: We have developed the concolic unit testing tool CONBOL to **automatically detect bugs** in **embedded C programs**
    - CONBOL has detected more than **100 real crash bugs** in 4M LoC embedded software developed by Samsung and 0.2M LoC open-source C program
  - Publications: We have published **4 papers** to prestigious SE conferences: **ICSE 2012, FSE 2011, ASE 2013**, and **ICST 2012**

Industrial Achievement from the Proejct

Project Years
2010
2011-2012
2013-2014

Phase1(2010)
Concolic testing
Feasibility study

- We applied concolic testing to a pilot project for a feasibility study
- We **detected new bugs** in open source SW (Busybox) and Samsung-developed SW (SLP(Tizen's predecessor) file manager, Samsung security library)

Phase2(2011-2012)
Concolic testing Tool development

- We developed the automated concolic unit testing tool CONBOL for testing embedded SW developed by Samsung
- CONBOL **detected 27 real bugs** in **4M LoC embedded SW developed by Samsung**
  - All the bugs were confirmed and fixed by Samsung developers

Phase3(2013-2014)
Concolic testing
Tool improvement
↑speed & ↓#false alarms

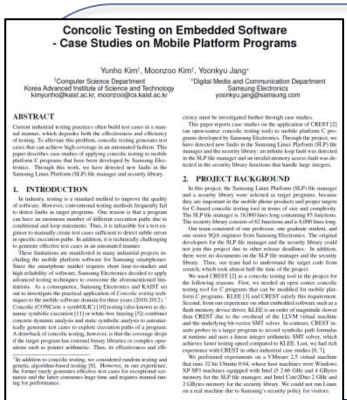
- We developed **false alarm reduction techniques**: unit-testing strategy and pre-condition generation
- CONBOL improvement: 64bit support, speed improvement, etc
- CONBOL **detected real bugs** in **embedded SW developed by Samsung** through weekly basis application

Success of CONBOL at Samsung Electronics

- Bronze Award at Samsung Best Paper Award
- Oct’s Best Practice Award
- Team leader Dr. Yoonkyu Jang received Samsung Award of Honor

Academic Achievement from the Project

Project Year
2010
2011
2012
2013
2014


Concolic Testing on Embedded Software - Case Studies on Mobile Platform Programs

Yunho Kim[†], Moonzoo Kim[†], Yoonkyu Jang[*]
[†]Computer Science Department, Korea Advanced Institute of Science and Technology, kimyunho@kaist.ac.kr, moonzoo@cs.kaist.ac.kr
[*]Digital Media and Communication Department, Samsung Electronics, yoonkyu.jang@samsung.com

- FSE 2011 held in Szeged, Hungary

Industrial Application of Concolic Testing Approach: A Case Study on `libexif` by Using CREST-BV and KLEE

Yunho Kim, Moonzoo Kim, YoungJoo Kim, CS Dept. KAIST, South Korea
Yoonkyu Jang, Samsung Electronics, South Korea

Industrial Application of Concolic Testing on Embedded Software: Case Studies

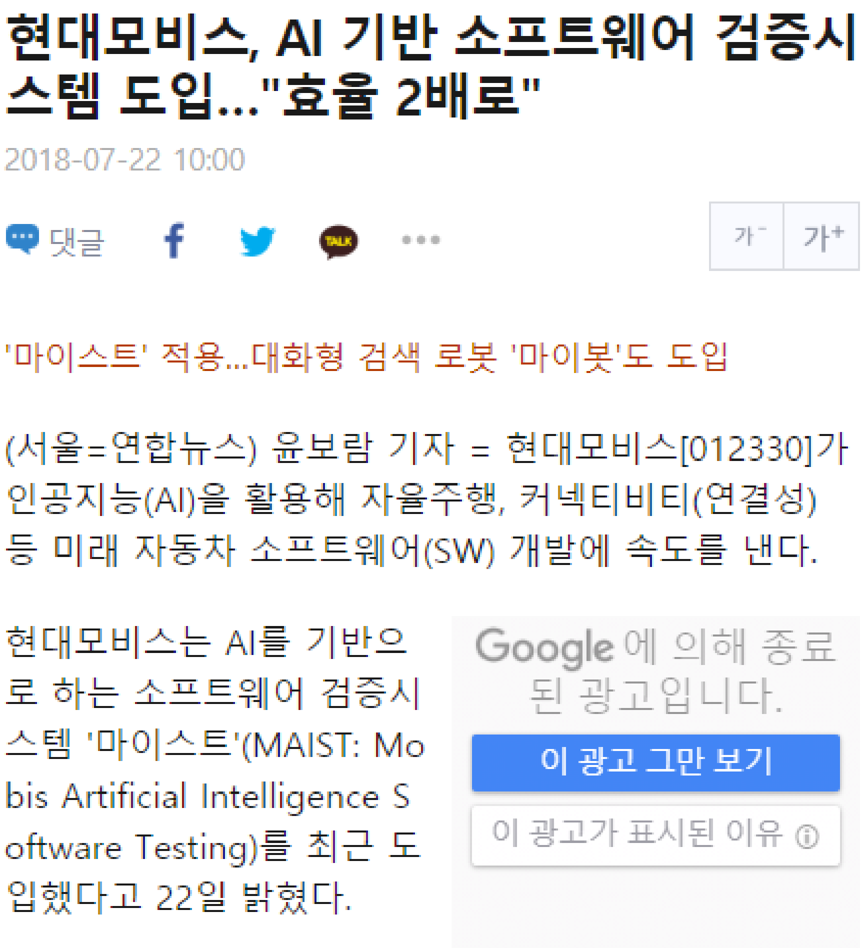
Moonzoo Kim, Yunho Kim, Department of Computer Science, KAIST, South Korea
Yoonkyu Jang, Samsung Electronics, South Korea

- ICSE 2012 held in Zurich, Switzerland,
- ICST 2012 held in Montreal Canada

Automated Unit Testing of Large Industrial Embedded Software using Concolic Testing

Yunho Kim, CS Dept. KAIST, South Korea
Youil Kim, Taeksu Kim, Gunwoo Lee, Yoonkyu Jang, Samsung Electronics, South Korea
Moonzoo Kim, CS Dept. KAIST, South Korea

- ASE 2013 held in Palo Alto, CA, USA
- Bronze Award for Samsung Research competition



현대모비스, AI 기반 소프트웨어 검증시스템 도입..."효율 2배로"

2018-07-22 10:00

'마이스트' 적용...대화형 검색 로봇 '마이봇'도 도입

(서울=연합뉴스) 윤보람 기자 = 현대모비스[012330]가 인공지능(AI)을 활용해 자율주행, 커넥티비티(연결성) 등 미래 자동차 소프트웨어(SW) 개발에 속도를 낸다.

현대모비스는 AI를 기반으로 하는 소프트웨어 검증시스템 '마이스트'(MAIST: Mobis Artificial Intelligence Software Testing)를 최근 도입했다고 22일 밝혔다.

Hyundai Mobis and a research team lead by Prof. Moonzoo Kim at KAIST jointly developed MAIST for automated testing

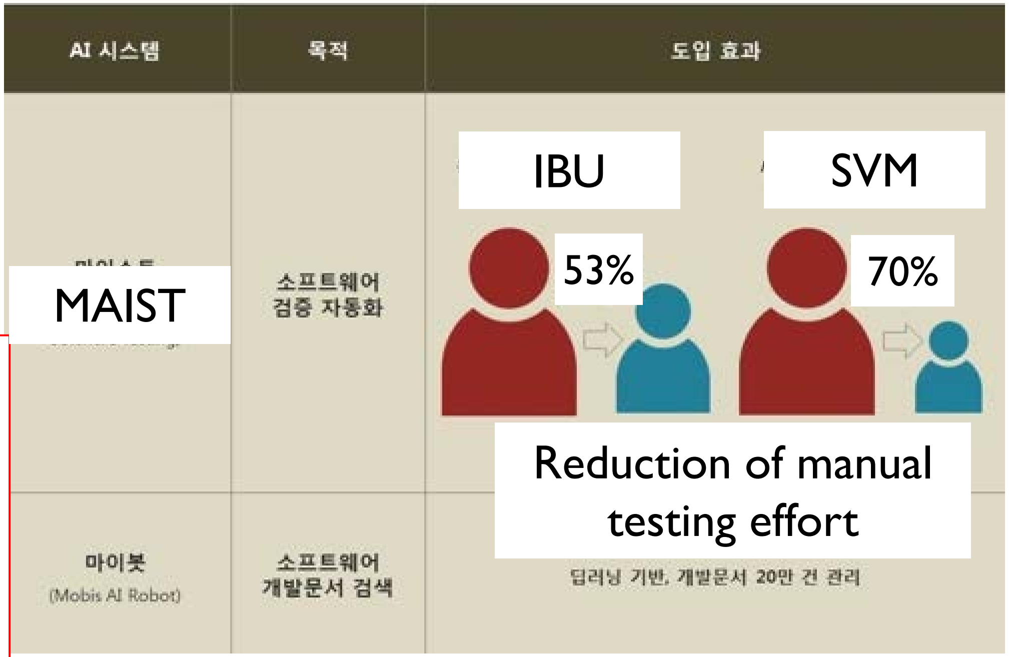
MAIST automates unit coverage testing performed by human engineers by applying concolic unit testing

실제 현대모비스가 통합형 차체제어시스템(IBU)과 써라

MAIST can reduce 53% and 70% of manual testing effort for IBU(Integrated Body Unit) and SVM(Surround View Monitoring)

현대모비스는 하반기부터 소프트웨어가 탑재되는 제동, 조향 등 모든 전장부품으로 마이스트를 확대 적용할 계획이다. 글로벌 소프트웨어 연구기지인 인도연구소에도 적용한다.

■ 현대모비스 인공지능 도입 사례



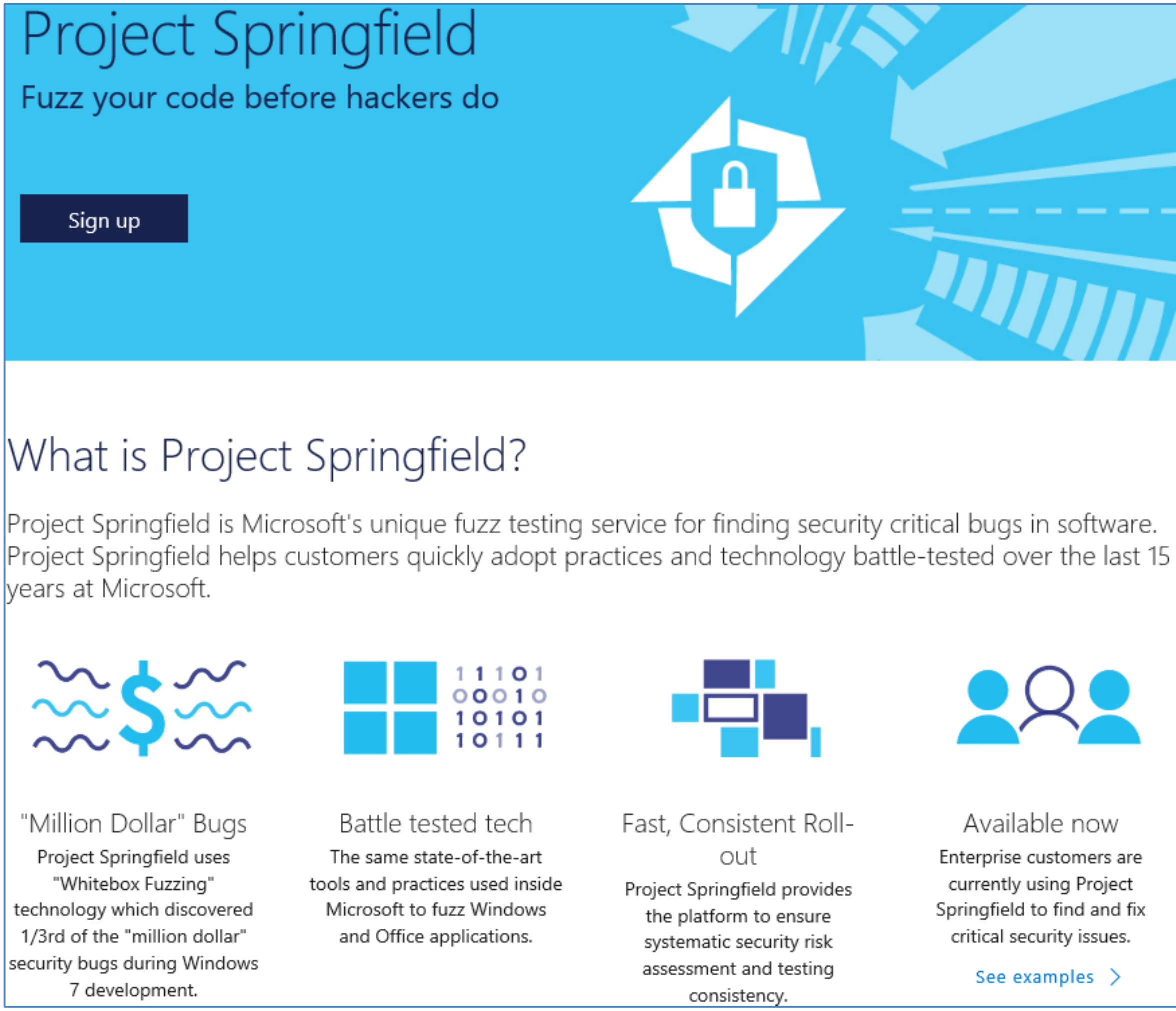
AI 시스템	목적	도입 효과
MAIST	소프트웨어 검증 자동화	IBU 53% SVM 70%
마이봇 (Mobis AI Robot)	소프트웨어 개발문서 검색	딥러닝 기반, 개발문서 20만 건 관리

Reduction of manual testing effort

http://m.yna.co.kr/kr/contents/?cid=AKR20180720158800003&mobile

Microsoft Project Springfield

- Azure-based cloud service to find security bugs in x86 windows binary
- Based on concolic testing techniques of SAGE



Project Springfield

Fuzz your code before hackers do

What is Project Springfield?

Project Springfield is Microsoft's unique fuzz testing service for finding security critical bugs in software. Project Springfield helps customers quickly adopt practices and technology battle-tested over the last 15 years at Microsoft.

"Million Dollar" Bugs

Project Springfield uses "Whitebox Fuzzing" technology which discovered 1/3rd of the "million dollar" security bugs during Windows 7 development.

Battle tested tech

The same state-of-the-art tools and practices used inside Microsoft to fuzz Windows and Office applications.

Fast, Consistent Roll-out

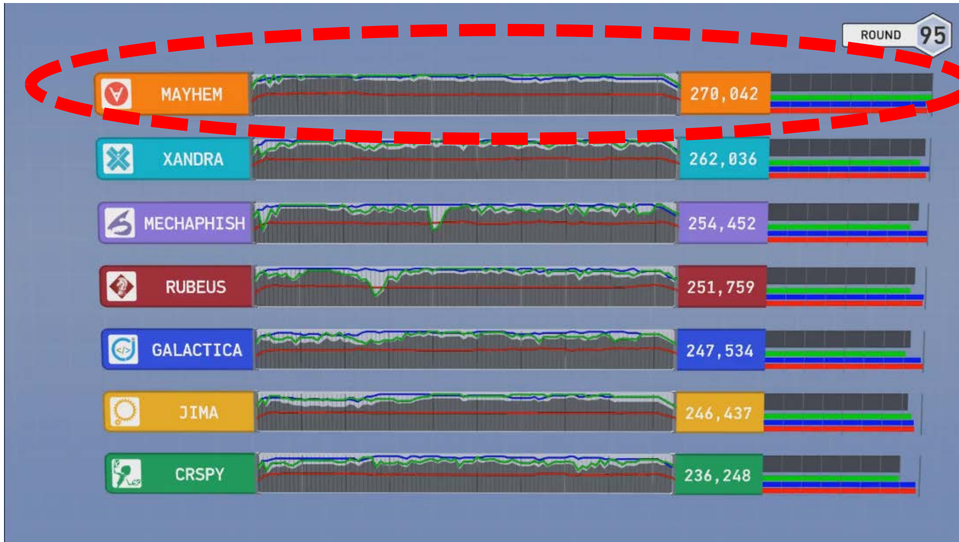
Project Springfield provides the platform to ensure systematic security risk assessment and testing consistency.

Available now

Enterprise customers are currently using Project Springfield to find and fix critical security issues.

2016 Aug DARPA Cyber Grand Challenge -the world's 1st all-machine hacking tournament

- Each team's Cyber Reasoning System automatically identifies security flaws and applies patches to its own system in a hack-and-defend style contest targeting a new Linux-based OS DECREE
- Mayhem won the best score, which is CMU's concolic testing based tool


ROUND 95

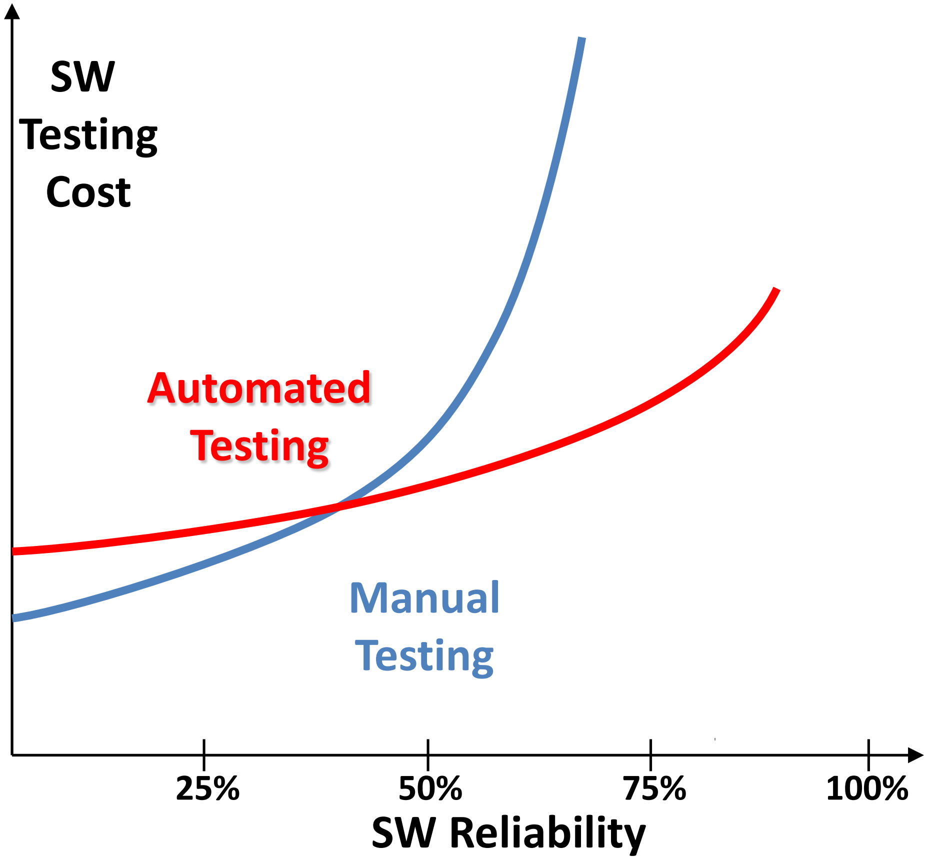
MAYHEM	270,042
XANDRA	262,036
MECHAPHISH	254,452
RUBEUS	251,759
GALACTICA	247,534
JIMA	246,437
CRSPY	236,248

Summary: Lessons Learned for Successful Industrial Application of Automated Analysis Technique

- Expect a large amount of manual work to understand and modify a target program and analysis tools
  - Field engineers may be too busy to explain a target program for you
    - Industrial software may not have good documents
- Be flexible for selecting target techniques to apply
  - Use appropriate techniques for the project, not the one you like
- Remote access right to target code is important
  - Due to IP issue, most companies are reluctant to give remote code access right.
- Expertise of field engineers in both target domain and the techniques is crucial

Conclusion

- Formal verification techniques really work in industry !
  - Software model checking and concolic testing detected hidden bugs in industrial embedded software
- Concolic testing is an industry friendly model checking technique.
  - Concolic testing is effective and efficient for testing industrial embedded software
- Successful application of automated testing techniques still requires expertise of human engineers in the domain



Traditional testing

- Manual TC gen
- Testing main scenarios
- System-level testing
- Small # of TCs

+

Concolic testing

- Automated TC gen
- Testing exceptional scenarios
- Unit-level testing
- Large # of TCs