Regression Test Selection for Tizen RT

Ahmet Celik\textsuperscript{1}, Young Chul Lee\textsuperscript{2}, and Milos Gligoric\textsuperscript{1}

\textsuperscript{1}The University of Texas at Austin
\textsuperscript{2}Samsung
Regression Testing

- Widely practiced in industry
- Checks that changes made to the project do not break the existing functionality
- Google, Facebook, Samsung, and many others uses Regression Testing extensively, since they have thousands of engineers making changes to the same project
Regression Testing for TizenRT

- TizenRT is an open-source lightweight RTOS-based platform implemented in C to support low-end Internet of Things (IoT) devices
- TizenRT includes a collection of test suites (Integration and Unit) that checks different functionalities
- A Samsung IoT platform is required to execute all tests

<table>
<thead>
<tr>
<th>Test Suite</th>
<th>Time [s]</th>
<th>#Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arastorage I-Tests</td>
<td>2.02</td>
<td>54</td>
</tr>
<tr>
<td>Arastorage U-Tests</td>
<td>1.01</td>
<td>46</td>
</tr>
<tr>
<td>Drivers Tests</td>
<td>3.02</td>
<td>26</td>
</tr>
<tr>
<td>Filesystem Tests</td>
<td>23.21</td>
<td>76</td>
</tr>
<tr>
<td>System I/O U-Tests</td>
<td>4.04</td>
<td>90</td>
</tr>
<tr>
<td>Network Tests</td>
<td>2.02</td>
<td>180</td>
</tr>
<tr>
<td>Kernel Tests</td>
<td>136.26</td>
<td>405</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>171.58</strong></td>
<td><strong>877</strong></td>
</tr>
</tbody>
</table>

Results for ARTIK 053
Regression Test Selection (RTS)

- Optimizes Regression Testing by analyzing the change
- Executes only tests that are affected by the change (and newly added tests)
- Is considered safe if it does not miss any test affected by changes
Using Existing RTS for TizenRT is Challenging

- Available RTS tools target managed languages, e.g. Java and C#.
- Additional constraints for TizenRT:
  - GNU Arm Embedded Toolchain does not support compiler plugins.
  - Limited memory, processing and storage space in IoT device used to execute tests.
  - Transfer between device and host.
Our Solution: Selfection

- Targets projects written in C
- Analyzes Arm ELF binaries using readelf and objdump tools provided by GNU Arm Toolchain
- Analyzes code *statically* and thus does not require extra space and memory
- Works in three phases:
  - *Analysis Phase* Select tests those are affected by the change
  - *Execution Phase* Execute the selected tests
  - *Collection Phase* Collect dependencies for all tests
Testing in TizenRT

Code A

```c
.../le_tc/kernel/kernel_tc_main.c
int tc_kernel_main(int argc, char*argv[])
{
    ... mqueue_main();
    ...
}
```

Code B

```c
.../le_tc/kernel/tc_mqueue.c
int mqueue_main(void)
{
    ...
    tc_mqueue_mq_notify();
    tc_mqueue_mq_timedsend_timedreceive();
    ...
    return 0;
}
```

Code C

```c
.../le_tc/kernel/tc_mqueue.c
static void tc_mqueue_mq_timedsend_timedreceive(void)
{
    int ret_chk = OK;
    timedsend_check = timedreceive_check = 0;
    ret.chk = timedsend_test();
    TC_ASSERT_EQ("timedsend_test", ret_chk, OK);
    ret.chk = timedreceive_test();
    TC_ASSERT_EQ("timedreceive_test", ret_chk, OK);
    mq.unlink("t_mqueue");
    TC_SUCCESS_RESULT();
}
```
Arm ELF Binary Example

- Example, disassembled Arm ELF Binary

04110e0c <tc_wqueue_work_queue_cancel>:

- 4110e0c: e92d41ff push {r0, r1, r2, r3, r4, r5, r6, r7, r8, lr}
- 4110e10: e59f021c ldr r0, [pc, #540]
- 4110e14: ebff127f bl 40d5818 <tc_skip_function>

- 4110e28: e1a06000 mov r6, r0
- 4110e2c: e3a00020 mov r0, #32
- 4110e30: ebff0727 bl 40d2ad4 <malloc>

- 4110ffc: ebfef111 bl 40cd448 <work_queue>

- 4111028: e3a00001 mov r0, #1
- 411102c: eb031a99 bl 41d7a98 <sleep>

- 411106c: 041f8499 .word 0x041f8499
Selfection Analysis Phase

- Find the tests to run
- Get executable code of the functions from the binary
- Checksum the code in a smart way by using symbol names instead of symbol addresses
- Compute transitive closure of affected functions using the dependency graph obtained in Collection Phase and check if any test is in this set, and find newly added tests
- Example: The change with SHA aa7f5149 on the left side is from TizenRT, a new test is added to kernel test suite

```c
int tc_kernel_main(int argc, char *argv[]) {
    ...
    wqueue_main();
    ...
}

int wqueue_main(void) {
    ...
    tc_wqueue_work_queue_cancel();
    ...
    return 0;
}
```
Selfection Execution Phase

- Testing framework of TizenRT does not support test filtering
- We added support for test filtering to TizenRT by including functions and macros statically
- Selfection sends the selected tests to device before execution started using serial console, and only those tests will not be skipped

```
.../tash_main.c
#ifdef SELFICATION
...
while(strcmp(line_buff,">>start")!=0){ ... }
...
for(;;){ ...
  if(strcmp(line_buff,"stop<<")!=0){ tc_skip_function_set(line_buff); ... } else { ... }
};
#endif
```

```
.../le_tc/kernel/tc_wqueue.c
static void __attribute__((noclone))
__attribute__((noinline))
tc_wqueue_work_queue_cancel(void) {
  if(tc_skip_function(__func__))return;
  ...
}
```
Selfection Collection Phase

- Selfection statically analyzes binaries to build function call graph
- Example: On the right side, function call instructions are shown as bold
- `tc_wqueue_work_queue_cancel` depends on `tc_skip_function`, `malloc`, `work_queue` and `sleep` functions, and any function they depend on transitively

```
04110e0c <tc_wqueue_work_queue_cancel>:
4110e0c:  e92d41ff        push    {r0, r1, r2, r3, r4, r5, r6, r7, r8, lr}
4110e10:  e59f021c        ldr     r0, [pc, #540]
4110e14:  ebff127f        bl      40d5818  <tc_skip_function>
          ...               
4110e28:  e1a06000        mov     r6, r0
4110e2c:  e3a00020        mov     r0, #32
4110e30:  ebff0727        bl      40d2ad4 <malloc>
4110ff:  ebfef111        bl      40cd448 <work_queue>
          ...               
4111028:  e3a00001        mov     r0, #1
411102c:  eb031a99        bl      41d7a98 <sleep>
          ...               
411106c:  041f8499        .word   0x041f8499
```
Evaluation

- We asked three Research Questions (RQs):
  - RQ1: How many tests does Selfection skip on average across a large number of revisions?
  - RQ2: What is the reduction, on average, in end-to-end test execution time across a large number of revisions?
  - RQ3: How does time for Analysis, Execution, and Collection phases compare to other build steps?
Experiment Setup

- 150 revisions used in the experiment is annotated to support test selection in an automated manner
- ARTIK 053 IoT device by Samsung is used to execute all tests
- QEMU emulator is also used, however only kernel test can be executed without hardware
- For each revision repeat:
  - Checkout the revision
  - Execute all tests (RetestAll) and collect the number of executed tests and time to execute them
  - Apply three phases of Selfection, and collect the number of selected tests and time to execute them
RQ1: How many tests does Selfection skip on average across a large number of revisions?

6% of tests are selected in ARTIK 053

5% of tests are selected in QEMU
RQ2: What is the reduction, on average, in end-to-end test execution time across a large number of revisions?

Execution time reduced to 27% of RetestAll in ARTIK 053

Execution time reduced to 7% of RetestAll in QEMU
RQ3: How does time for Analysis, Execution, and Collection phases compare to other build steps?
Conclusion

- **Selfection**
  - RTS tool for projects in C that compiles to Arm ELF binary
  - Statically analyzes binaries to collect call-graph dependencies and find affected tests
- Substantial savings in testing time and number of executed tests
- Only the execution phase is specific to TizenRT

Ahmet Celik <ahmetcelik@utexas.edu>
Young Chul Lee <yc207.lee@samsung.com>
Milos Gligoric <gligoric@utexas.edu>