2015 IEEE/ACM 1st International Workshop on Software Engineering for Smart Cyber-Physical Systems

Accelerating Regression Testing for Scaled Self-Driving Cars with Lightweight Virtualization – A Case Study

Christian Berger
Department of Computer Science and Engineering
University of Gothenburg
christian.berger@gu.se

***Abstract*—Engineering software for smart cyber-physical systems (sCPS) challenges developers as they have to deal with uncertain and volatile stimuli data perceived by sensors. Regression testing of a sCPS is time-consuming on sequential execution. However, sequential testing can be parallelized depending on the system calls used in the system-under-test. In a case study about accelerating regression testing for scaled self-driving cars, we evaluate the use of namespace-separation based lightweight virtualization that powers solutions like *Docker* or Google's *lmctfy*. After transparently adding lightweight virtualization to CxxTest that is used for regression testing, the total test execution time could be reduced from previously over 12min by more than 62% to less than 5min. Thus, the technology for today's lightweight virtualization can also be used to safely accelerate test-runners without changing existing test cases.**

I. Introduction

The data processing chain of smart cyber-physical systems (sCPS) like scaled self-driving cars, drones, robots, or wearable devices consists typically of *perceive - decide - act* in a *pipe-and-filter* architecture: Sensors like cameras or accelerometers help intelligent algorithms to perceive their environmental context to derive acting decisions. Depending on the device class, actions could range from simple monitoring tasks in the case of alert systems up to complex control tasks to hover drones over ground or to follow the road with scaled self-driving cars like the one depicted in Fig. 1 (cf. [1]).

Fig. 1. Scaled self-driving car "Legendary" used in education and as demonstrator for software engineering research (cf. [1], [2]).

```
void testCase1() {
    SharedMemory* m = createSharedMemory("mem", 5);
    assert(m != NULL);
    snprintf(m->buf, m->size, "%s", "ABCDE");
    assert(strncmp(m->buf, "ABCDE", m->size) == 0);
    release(m);
}

void testCase2() {
    SharedMemory* m = createSharedMemory("mem", 5);
    assert(m != NULL);
    snprintf(m->buf, m->size, "%s", "FGHIJ");
    assert(strncmp(m->buf, "FGHIJ", m->size) == 0);
    release(m);
}
```

Fig. 2. Test cases that cannot be safely parallelized by forking a separate child process.

A. Problem Statement and Motivation

On resource-constrained devices, this pipe-and-filter chain can be designed in a data-driven manner to react on new stimuli ranging from simple data logging up to control tasks. Using shared memory as inter-process communication (IPC) allows for low latency times and a predictable behavior between components comprising the pipe-and-filter chain.

To extensively test such pipe-and-filter chains, potential test scenarios for the sCPS can be provided as test cases in an xUnit environment to enable regression testing. Such test cases range from model-based stimuli, over embedded simulations to assess the behavior of the system-under-test (SUT) over time, up to the use of real world recordings played back for the SUT to also include unanticipated effects from real-world sensor data processing (cf. [3]).

Such regression tests for the pipe-and-filter chain can be parallelized on powerful development machines to avoid a sequential and time-consuming test process.

However, the test suite excerpt in Fig. 2 illustrates an example that cannot be safely parallelized in the same process context of the test-runner because both test cases use IPC system calls to access a shared memory segment with an identical name. Despite the fact that this illustrative example could have been solved by simply changing the name of the shared memory, this change would have to be made to the actual test cases requiring domain knowledge to fix unwanted side-effects.

As the aforementioned example already demonstrates, a domain-independent parallelization for executing test cases is not simply realizable for the general case. This can also be seen in the documentation of other xUnit frameworks like Google Test, which explicitly states that "...[it] doesn't try to solve the problem of running tests in parallel."[1]. Instead, it is suggested to "[...] better run them in different processes."; however, splitting the aforementioned test cases into separated processes does not solve the illustrated IPC problem as in this particular case, the second test case will fail on a POSIX.1-2001 system with the return code `SEM_FAILED` and `errno` set to `ENOENT`.

B. *Research Objectives*

The aforementioned example is an illustrative excerpt from the middleware in use with our scaled cars (cf. [1]). They use shared memory to efficiently process captured video frames and to record sensor data in parallel to the time-critical pipe-and-filter processing chain.

Recent achievements in namespace separation-based virtualization technology allow lightweight virtual machines that can be set up and instantiated rapidly as in solutions like *Docker* [4], [5] or Google's *lmctfy* ("let me contain that for you") [6]. This article investigates how the concept that enables the aforementioned solutions can be transparently added to an xUnit test-runner to safely parallelize test cases to avoid the problems pointed out before. Therefore, a case study was conducted using the source code of our scaled self-driving cars and reported here following the guidelines from Runeson and Höst [7].

C. *Context and Limitations*

The study was conducted in an explorative manner to implement and evaluate the concept of namespace separation-based virtualization with an xUnit test environment. The motivational examples are chosen from the domain of embedded systems on the example of scaled self-driving cars. Such examples can be found in further robotic and ambient device applications; other domains such as web systems might be affected as well but are not studied here.

D. *Contributions of this Study*

The contributions of this article can be summarized as follows:

- A concept for safely parallelizing test cases for software systems that use system resources, which cannot be concurrently accessed with test-runners from currently available xUnit environments.
- Results from a case study where the aforementioned concept was evaluated.

E. *Structure of this Study*

The rest of the article is structured as follows: Sec. II describes the relevant technical background as well as related work. In Sec. III, the design of the case study is described followed by the presentation of its results in Sec. IV. The article is concluded in Sec. V

II. Background and Related Work

Lightweight virtualization based on namespace separation is a technique that uses several low-level Linux kernel features available from version 2.6.24 in the official branch released on January 24, 2008. These features allow the isolation of namespaces for IPC and network stacks for example and thus, enable user-space applications to share the same kernel while having strictly separated process and communication contexts. These features were used for example by OpenVZ [8] and LXC [9]. With tools like Google's lmctfy [6], administrative tasks for creating, running, and maintaining such lightweight virtual machines were significantly simplified; with Docker [4] released as open source recently, an entire software ecosystem was established simplifying the packaging and exchange of applications wrapped into containers.

Dua et al. [10] provide an overview and comparison of different lightweight virtualization techniques and regular virtual machines. They have a clear focus on the needs for platform-as-a-service environments (PaaS). In their evaluation, they only provide a qualitative-descriptive comparison.

Felter et al. [11] compare in their work the performance of virtual machines with lightweight containers on Linux. They conclude that the additional overhead for separating processes into different namespaces can be neglected regarding effects on CPU and memory performance. However, they observe additional overhead in the communication during high packet rates. Another performance comparison is conducted by Tang et al. [12] who have similar conclusions as Felter et al. In direct comparison with Docker's technology, lmctfy from Google performed slightly better in the network performance according to their observations due to a different implementation design that does not rely on virtualized network devices.

Marinescu et al. [13] have used Docker as technological basis in their repository analysis framework Covrig to conduct a large-scale and yet safe inspection of the revision history from six selected Git code repositories. For their analysis, they fetched several revisions and started one lightweight container per instance to ensure an isolated execution for compiling and running the test code while collect data about lines-of-code and code coverage. As their motivation is similar, their focus was on providing a clean and thus, comparable execution environment to the different compilation and test runs; we are interested in investigating how test suites in general can be accelerated.

Bell and Kaiser [14] propose an approach for Unit Test Virtualization by shortening the start-up time of individual test cases. To achieve this, they monitor potential changes to an initial state of the SUT after a test case's setup phase during its execution; if there are no modifications to the system's initialization state, the next test case can be executed without having the need of re-running the setup phase again. As their overall goal is similar, they focus on the test case initialization phase by using code instrumentation while this work is focusing on parallelizing the test case execution;

[1]Google Test FAQ: http://goo.gl/mm2Q1F

additionally, the approach described here is not depending on code instrumentation.

III. Case Study Design

The related works let us safely assume that lightweight virtualization might be a promising approach to address safe parallelization of individual test cases. Thus, we designed a case study with the goal to investigate how the concept from lightweight virtualization can be incorporated into test-runners and to evaluate the performance gains. In this section, the design of the case study is described according to the guidelines from Runeson and Höst [7].

A. Research Questions

The following research questions were of interest for this case study:

RQ-1: *How can the fundamental concept for namespace separation-based lightweight virtualization be transparently added to an xUnit test environment on the example of CxxTest [15]?*

RQ-2: *What are the costs and benefits of using lightweight virtualization to accelerate regression testing?*

B. Case and Subjects Selection

We conducted the study using the source code and test environment that is used on our scaled self-driving cars (cf. [3], [1]).[2] The software was also used with real scale self-driving cars as described here [16] and influenced by design decisions and experiences obtained during the 2007 DARPA Urban Challenge (cf. [17]). The motivation to select the software that we are using on our cars is manifold: Firstly, the software is in use since several years on robotic miniature vehicle platforms that are participating in international competitions (cf. [2]); secondly, the software is of reasonable size as summarized in Tab. I and the functionality to realize sCPS can be considered as relevant for this domain as demonstrated at various occasions (cf. [3], [1]); finally, the software environment is easily accessible from our end to conduct and evaluate the required changes for this study.

LoC	Middleware		Simulation		Visualization	
headers	6,530	28.5%	6,801	22.0%	2,106	32.4%
sources	10,507	45.8%	22,380	72.3%	4,392	67.6%
tests	5,886	25.7%	1,778	5.7%	-	-
total	22,923	-	30,959	-	6,498	-

TABLE I
Overview of the source lines of code; the domain code (i.e. applications to realize the self-driving functionality) accounts for additional 1,702 source lines of code.

We collected the data from 61 test suites in total consisting of 141 test cases in total. The test suites contained a little more than 4 test cases on average; and approximately 30% of the test suites contained up to two test cases according to the histogram shown in Fig. 3; such test suite classification is relevant to decide when to add parallelization to test-runners. Furthermore, the test suites names `RuntimeControl*` contain test cases that substitute the real system time with a virtual one for a complete system-under-test virtualization.

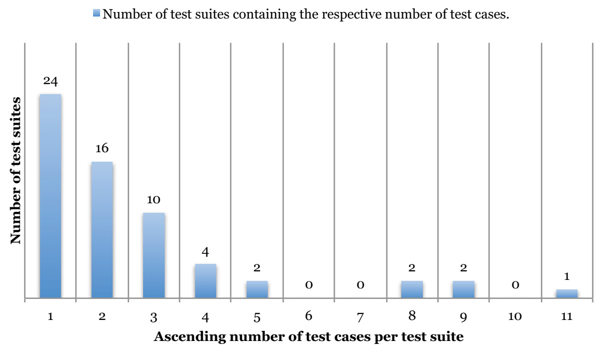

Fig. 3. Histogram about the number of test suites containing the respective number of test cases; this chart also shows how many child processes were started at most.

C. Data Collection Procedure

To address *RQ-1*, the implementation of Google's lmctfy as well as Docker was studied. The resulting technical approaches were compared and prototypically realized in CxxTest as described in Sec. IV-A. *RQ-2* was addressed by measuring the duration for executing the single test cases; for the sequential execution, the existing CxxTest's TestRunner.h header file was modified accordingly. The time measurement for the parallelized execution of test cases included also the time for spawning the clones for the respective test cases in one test suite.

D. Analysis and Validity Procedure

The data was post-processed and further analyzed using a spreadsheet tool. Here, the data from the sequential execution served as ground truth and validation to enable a comparison for the transparently added cloning procedure for the test cases. The performance comparison was conducted in logarithmic scale using the ground truth data per test case and per test suite in ascending order, respectively.

IV. Results

In the following, we present the results to the respective research questions.

A. Namespace Separation-based Lightweight Virtualization for xUnit

Lightweight virtualization utilizing namespace separation is based on the Linux system call `clone(2)`, which is neither available on other POSIX-compliant Unix-like operating systems nor on Microsoft Windows. To enable parallelized execution of test cases within a test suite without changing the respective test cases, the CxxTest's TestRunner was adapted. Therefore, the executing part for the test suites was located and modified to delegate the call to a wrapping method, which is spawning the child processes in separate namespaces. The entire implementation is available in Appendix A.

[2] OpenDaVINCI: http://www.cse.chalmers.se/~bergerc/opendavinci/

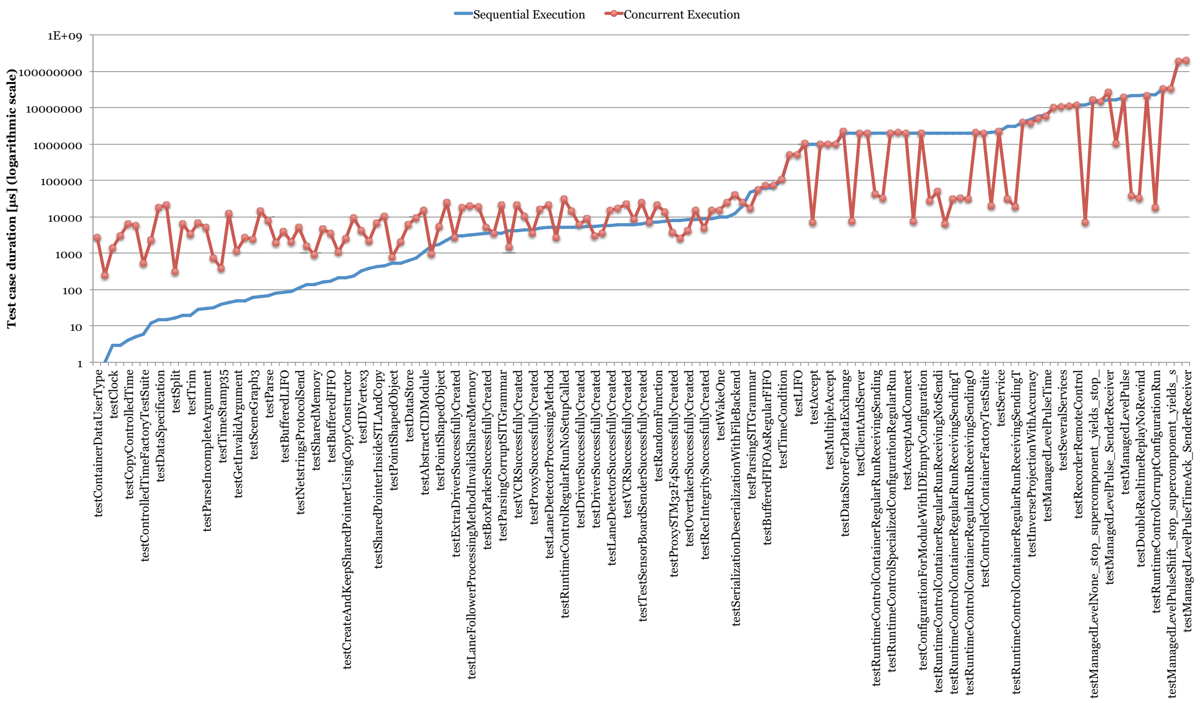

Fig. 4. Duration of test cases executed in the same process context as the test-runner vs. execution with a lightweight virtual machine.

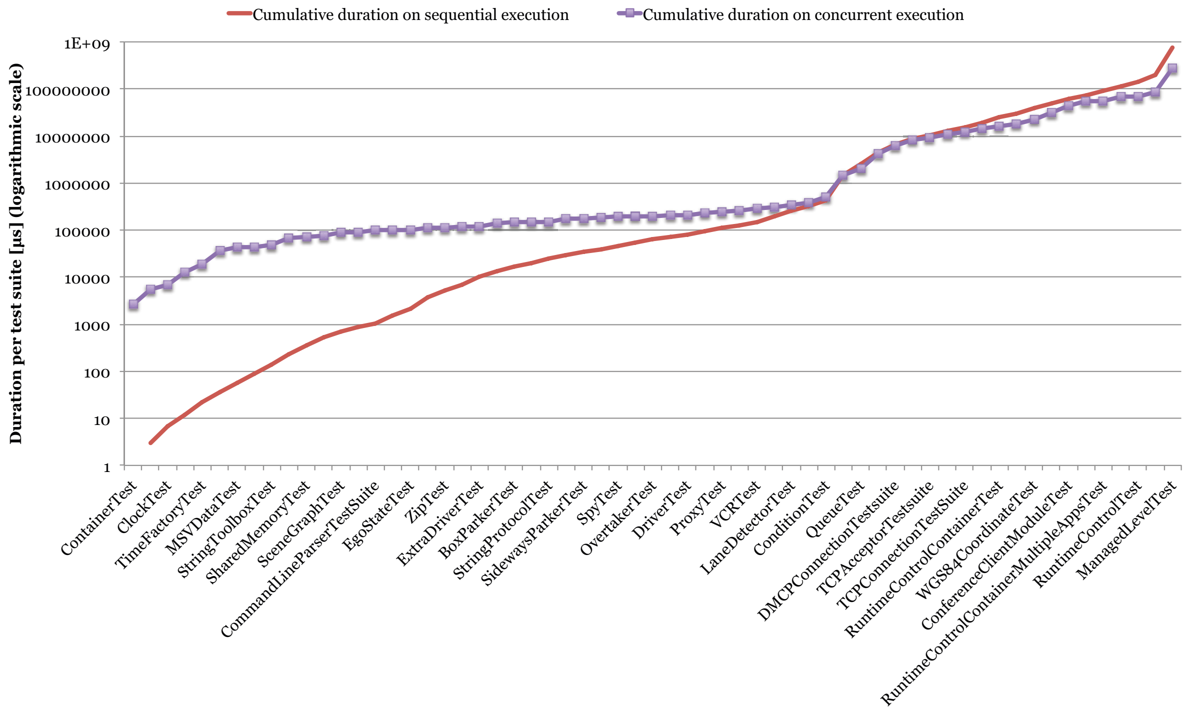

Fig. 5. Cumulated duration of executing test suites: The solid chart depicts the cumulated duration on sequential execution and the chart with the squares shows the cumulated duration on concurrent execution.

B. Performance of Namespace Separation-based Lightweight Virtualization for xUnit

The results from running the test cases in lightweight virtual machines based on namespace separation on an Intel Core i7 at 1.8 GHz with 1GB RAM restricted to use only one core are shown in Fig. 4. The solid chart presents the duration of the individual test cases sequentially executed in ascending order on a logarithmic scale. In addition, the duration of the respective test case executed in a separate child process is shown; the additional runtime overhead thereof is apparently in the range of approximately 5ms on average.

Fig. 5 shows on a logarithmic scale the cumulated duration of the individual test suites for the sequential and the parallel execution: The solid chart shows the totally cumulated duration for all test suites on sequential execution; the chart with the squares shows the cumulative duration for all test suites on concurrent execution.

In both charts, sorting the data by duration and connecting the individual execution times with a continuous line was chosen to better visualize the break-even in terms of when the costs of wrapping a test case into a child process start to pay off. This point is reached after the test suite `QueueTest`, where the sequential execution has accumulated 2,578ms while the parallel execution is at 2,020ms. In total, the parallel execution requires only 38% of the original execution time of more than 12min resulting in less than 5min.

C. Threats to Validity

Regarding construct validity, the setup of the case study can be considered as valid to evaluate the concept of adding namespace separation-based lightweight virtualization to test-runners on the example of an xUnit environment. This is due to the chosen example from the sCPS domain dealing with self-driving scaled cars, which can be considered relevant and representative for such robotic sCPS as reported in [1].

Considering internal validity, the lightweight virtualization based on namespace separation added to the xUnit environment does neither account for the number of test cases per test suite nor their specific requirements regarding resources. Thus, further improvements might be expected by modeling such constraints properly.

Regarding external validity, the results need to be validated with further examples from the sCPS domain to generalize the findings. In addition, the observations are only valid on Linux as the namespace separation-based lightweight virtualization concept is not present on other platforms as such.

V. Conclusions and Future Work

This work presented and evaluated an approach to accelerate the execution time of test suites by running the contained test cases in lightweight virtual machines based on separating namespaces. The approach was transparently added to an xUnit test-runner without changing the actual test cases and thus, the total test suites' execution duration was reduced by more than 62%.

A. Summary of Conclusions

Lightweight virtualization by namespace separation of processes is an approach to safely separate processes from each other. As this approach adds only neglectable overhead in comparison to the native execution of a process, this concept helps to accelerate test suites significantly. In addition, it ensures that individual test cases do not interfere with each other as they are strictly separated from each other running in isolated process contexts.

B. Relation to Existing Evidence

The concept and case study described here complement previous works from a different perspective to reduce the execution time of test suites. In this regard, this work contributes to address the need of rapidly testing systems with a growing software complexity where fast feedback from regression testing is needed.

C. Impact/Implications

The described approach also provides a way to safely run test cases in parallel without interfering with each other. As also popular xUnit test environments have not addressed this issue in a generic manner, this work contributes to this challenge of practical relevance as well.

D. Limitations

The presented concept depends on a specific technological approach that is provided by the Linux kernel. While there might be similar concepts on different POSIX-compliant platforms, the approach was only evaluated on the Linux platform and has a technological dependency. Furthermore, namespace separation requires the `CAP_SYS_ADMIN` privilege for execution.

E. Future Work

So far, the presented approach does not account for context or domain related information from the individual test cases. Therefore, future work needs to investigate how additional information about the individual test cases can be used to better plan the scheduling, selection, and spawning of children in separated namespaces.

References

[1] C. Berger, "From a Competition for Self-Driving Miniature Cars to a Standardized Experimental Platform: Concept, Models, Architecture, and Evaluation," *Journal of Software Engineering for Robotics*, vol. 5, no. 1, pp. 63–79, Jun. 2014. [Online]. Available: http://arxiv.org/abs/1406.7768

[2] S. Zug, C. Steup, J.-B. Scholle, C. Berger, O. Landsiedel, F. Schuldt, J. Rieken, R. Matthaei, and T. Form, "Technical evaluation of the Carolo-Cup 2014 - A competition for self-driving miniature cars," in *Proceedings of the IEEE International Symposium on Robotic and Sensors Environments (ROSE)*. Timisoara, Romania: IEEE, Oct. 2014, pp. 100–105. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6952991

[3] C. Berger, M. Chaudron, R. Heldal, O. Landsiedel, and E. M. Schiller, "Model-based, Composable Simulation for the Development of Autonomous Miniature Vehicles," in *Proceedings of the SCS/IEEE Symposium on Theory of Modeling and Simulation*, San Diego, CA, USA, Apr. 2013, p. 7. [Online]. Available: http://dl.acm.org/citation.cfm?id=2499651

[4] Docker, "Docker," Jan. 2014. [Online]. Available: www.docker.com

[5] D. Merkel, "Docker: lightweight Linux containers for consistent development and deployment," *Linux Journal*, no. 239, Mar. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2600241

[6] Google, "lmctfy - Let Me Contain That For You," Jan. 2014. [Online]. Available: github.com/google/lmctfy

[7] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, Dec. 2008. [Online]. Available: http://link.springer.com/10.1007/s10664-008-9102-8

[8] OpenVZ, "OpenVZ," Jan. 2015. [Online]. Available: www.openvz.org

[9] LinuxContainers, "LXC LinuxContainers," Jan. 2015. [Online]. Available: linuxcontainers.org

[10] R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs Containerization to Support PaaS," in *Proceedings of the IEEE International Conference on Cloud Engineering*. IEEE, Mar. 2014, pp. 610–614. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6903537

[11] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An Updated Performance Comparison of Virtual Machines and Linux Containers," IBM Research Division, Austin Research Laboratory, Austin, TX, USA, Tech. Rep., Jul. 2014.

[12] X. Tang, Z. Zhang, M. Wang, Y. Wang, Q. Feng, and J. Han, "Performance Evaluation of Light-Weighted Virtualization for PaaS in Cloud," in *Proceedings of the 13th International Conference on Algorithms and Architectures for Parallel Processing*, X.-h. Sun, W. Qu, I. Stojmenovic, W. Zhou, Z. Li, H. Guo, G. Min, T. Yang, Y. Wu, and L. Liu, Eds., Dalian, China, Aug. 2014, pp. 415–428.

[13] P. Marinescu, P. Hosek, and C. Cadar, "COVRIG: A Framework for the Analysis of Code, Test, and Coverage Evolution in Real Software," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, San Jose, CA, USA, Jul. 2014, pp. 93–104.

[14] J. Bell and G. Kaiser, "Unit test virtualization with VMVM," in *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, vol. 2014, no. June, Hyderabad India, Jun. 2014, pp. 550–561. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2568225.2568248

[15] CxxTest, "CxxTest," Jan. 2015. [Online]. Available: www.cxxtest.com

[16] C. Berger, *Automating Acceptance Tests for Sensor- and Actuator-based Systems on the Example of Autonomous Vehicles*. Aachen, Germany: Shaker Verlag, Aachener Informatik-Berichte, Software Engineering Band 6, 2010. [Online]. Available: http://www.christianberger.net/Ber10.pdf

[17] C. Basarke, C. Berger, K. Berger, K. Cornelsen, M. Doering, J. Effertz, T. Form, T. Gülke, F. Graefe, P. Hecker, K. Homeier, F. Klose, C. Lipski, M. Magnor, J. Morgenroth, T. Nothdurft, S. Ohl, F. W. Rauskolb, B. Rumpe, W. Schumacher, J. M. Wille, and L. Wolf, "Team CarOLO - Technical Paper," Technische Universität Braunschweig, Braunschweig, Germany, Informatik-Bericht 2008-07, Oct. 2008.

Appendix

A. Source Code for Parallelizing Test Cases

```
void runSuite( SuiteDescription &sd ) {
    StateGuard sg;
    tracker().enterSuite( sd );
    if ( sd.setUp() ) {
        runAsClones(sd.firstTest());
        sd.tearDown();
    }
    tracker().leaveSuite( sd );
}
```

Fig. 6. Modification to CxxTest's TestRunner class for delegating the execution of a test suite to a wrapping method.

```
#define STACK_SIZE (1024 * 1024)
class CloneDescriptor {
    public:
        pid_t childPid;
        char *stack;
        CxxTest::TestDescription *td;
        core::data::TimeStamp start;
        core::data::TimeStamp end;
};
std::map<pid_t, CloneDescriptor*> mapOfClones;
```

Fig. 7. Data structure for describing a spawned process; the timestamping feature is used from the OpenDaVINCI framework.

```
static int runAsClones(CxxTest::TestDescription *td)
    {
  if (td != NULL) {
    for ( CxxTest::TestDescription *it = td; it; it
        = it->next() ) {
      if ( td->active() ) {
        CloneDescriptor *desc =
          new CloneDescriptor();
        desc->td = it;
        desc->start = core::data::TimeStamp();
        desc->childPid = createClonedChild(desc);
        mapOfClones[desc->childPid] = desc;
      }
    }
    // Wait after all children have started.
    std::map<pid_t, CloneDescriptor*>::iterator it =
        mapOfClones.begin();
    while ( it != mapOfClones.end() ) {
      waitpid(it->second->childPid, NULL, 0);
      it++;
    }
  }
  return 0;
}

static pid_t createClonedChild(CloneDescriptor *desc
    ) {
  desc->stack = (char*)malloc(STACK_SIZE);
  assert(desc->stack != NULL)
  pid_t retVal = clone(runClone, (desc->stack +
      STACK_SIZE),
       CLONE_NEWIPC | CLONE_NEWNS | SIGCHLD, (void*)
          (desc));
  return retVal;
}
```

Fig. 8. Delegate method that starts the individual test cases (top) and the actual Linux system call to create a lightweight virtual machine based on namespace separation (bottom); in this case, the test case will be executed in a newly created process context that is separated from the existing processes (including the parent context).

```
static int runClone(void *arg) {
  if (arg != NULL) {
    CloneDescriptor *desc = (CloneDescriptor*)arg;
    if (desc->td != NULL) {
      CxxTest::TestRunner::StateGuard sg;
      CxxTest::tracker().enterTest( *(desc->td) );
      if ( desc->td->setUp() ) {
        desc->td->run();
        desc->td->tearDown();
      }
      desc->end = core::data::TimeStamp();

      std::cerr << desc->td->suiteName() << ";"
          << desc->td->testName() << ";"
          << (desc->end - desc->start).
               toMicroseconds()
          << std::endl;
     CxxTest::tracker().leaveTest( *(desc->td) );
    }
  }
  return 0;
}
```

Fig. 9. Function that is finally executed within the child process executed in a separate namespace.