# 1st International Workshop on Quality-Aware DevOps (QUDOS 2015)

## Proceedings

Danilo Ardagna, Andreas Brunnert, Giuliano Casale,
and Andre van Hoorn

September 1, 2015
Bergamo, Italy

The Association for Computing Machinery, Inc.
2 Penn Plaza, Suite 701
New York, NY 10121-0701



**Notice to Past Authors of ACM-Published Articles**

ACM intends to create a complete electronic archive of all articles and/or other material previously published by ACM. If you have written a work that was previously published by ACM in any journal or conference proceedings prior to 1978, or any SIG Newsletter at any time, and you do NOT want this work to appear in the ACM Digital Library, please inform permissions@acm.org, stating the title of the work, the author(s), and where and when published.

ACM ISBN: 978-1-4503-3817-2

Additional copies may be ordered prepaid from:

| | |
|---|---|
| | Phone: 1-800-342-6626 |
| ACM Order Department | (U.S.A. and Canada) |
| P.O. BOX 11405 | +1-212-626-0500 |
| Church Street Station | (All other countries) |
| New York, NY 10286-1405 | Fax: +1-212-944-1318 |
| | E-mail: acmhelp@acm.org |

**Production:** Conference Publishing Consulting
D-94034 Passau, Germany, info@conference-publishing.com

# Message from the Chairs

It is our great pleasure to welcome you to the first edition of the International Workshop on Quality-aware DevOps (QUDOS 2015), held on September 1, 2015 in Bergamo, Italy, co-located with the 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2015).

DevOps has emerged in recent years as a set of principles and practices for smoothing out the gap between development and operations, thus enabling faster release cycles for complex IT services. Common tools and methods used in DevOps include infrastructure as code, automation through deep modeling of systems, continuous deployment, and continuous integration. As of today, software engineering research has mainly explored these problems from a functional perspective, trying to increase the benefits and generality of these methods for the end users. However, this has left behind the definition of methods and tools for DevOps to assess, predict, and verify quality dimensions.

The QUDOS workshop focuses on the problem of how to best define and integrate quality assurance methods and tools in DevOps. Quality covers a broadly-defined set of dimensions including performance, reliability, safety, survivability, cost of ownership, among others. To answer these questions, the QUDOS workshop wants to bring together experts from academia and industry working in areas such as quality assurance, agile software engineering, and model-based development. The goal is to identify and disseminate novel quality-aware approaches to DevOps.

QUDOS 2015 is a one-day workshop. In total, we have accepted four technical full papers proposing novel approaches for quality-aware DevOps and four short papers presenting tools in the scope of the workshop. These papers have been selected based on the reviews provided by the QUDOS 2015 program committee members. On average, each submitted paper received three reviews. In addition to the talks presenting the accepted papers, QUDOS 2015 features an invited keynote by Petr Tuma (Charles University of Prague, CZ), sharing his thoughts on performance awareness for DevOps. Moreover, ample space will be devoted to discussions on quality-aware DevOps.

QUDOS 2015 is organized by the consortia of the two EU Projects DICE and MODAClouds, as well as the DevOps Performance Working Group of the Standard Performance Evaluation Corporation's Research Group (SPEC RG). QUDOS 2015 is kindly sponsored by NovaTec Consulting GmbH and technically supported by SPEC RG. We thank the program committee members, who helped with timely and constructive reviews, as well as each author and presenter who submitted their work to the QUDOS 2015 workshop.

Danilo Ardagna, Andreas Brunnert, Giuliano Casale, Andre van Hoorn
(QUDOS 2015 Chairs)

# QUDOS 2015 Organization

## Workshop Chairs

| | |
|---|---|
| Danilo Ardagna | Politecnico di Milano, Italy |
| Andreas Brunnert | fortiss GmbH, Germany |
| Giuliano Casale | Imperial College London, UK |
| Andre van Hoorn | University of Stuttgart, Germany |

## Program Committee

| | |
|---|---|
| Varsha Apte | IIT Bombay, India |
| Matej Artac | XLAB, Slovenia |
| Simona Bernardi | Centro Universitario de la Defensa, AGM, Spain |
| Andre Bondi | Siemens Corporate Research, USA |
| Francesco D'Andria | ATOS, Spain |
| Wilhelm Hasselbring | Kiel University, Germany |
| Samuel Kounev | University of Wuerzburg, Germany |
| Klaus-Dieter Lange | HP, USA |
| Zhen Ming (Jack) Jiang | York University, Canada |
| Manoj Nambiar | Tata Consultancy Services, India |
| Richard Paige | University of York, UK |
| Dana Petcu | IEAT, Romania |
| Dorina Petriu | Carleton University, Canada |
| Meikel Poess | Oracle Corporation, USA |
| Matteo Rossi | Politecnico di Milano, Italy |
| Arnor Solberg | SINTEF, Norway |
| Catia Trubiani | Gran Sasso Science Institute, Italy |
| Petr Tuma | Charles University of Prague, Czech Republic |
| Liming Zhu | NICTA, Australia |

# Contents

# A DevOps Approach to Integration of Software Components in an EU Research Project

Mark Stillwell
m.stillwell@imperial.ac.uk

Jose G. F. Coutinho
gabriel.figueiredo@imperial.ac.uk

Imperial College London
United Kingdom

## ABSTRACT

We present a description of the development and deployment infrastructure being created to support the integration effort of HARNESS, an EU FP7 project. HARNESS is a multi-partner research project intended to bring the power of heterogeneous resources to the cloud. It consists of a number of different services and technologies that interact with the OpenStack cloud computing platform at various levels. Many of these components are being developed independently by different teams at different locations across Europe, and keeping the work fully integrated is a challenge. We use a combination of Vagrant based virtual machines, Docker containers, and Ansible playbooks to provide a consistent and up-to-date environment to each developer. The same playbooks used to configure local virtual machines are also used to manage a static testbed with heterogeneous compute and storage devices, and to automate ephemeral larger-scale deployments to Grid'5000. Access to internal projects is managed by GitLab, and automated testing of services within Docker-based environments and integrated deployments within virtual-machines is provided by Buildbot.

## Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement; D.2.9 [**Software Engineering**]: Software configuration management

## Keywords

DevOps, Configuration Management, Automated Testing, Ansible, Vagrant, Docker, OpenStack, GitLab, BuildBot

## 1. INTRODUCTION

In the past most academic software was developed for specific purposes by individuals or small teams of developers. However, current funding policies of agencies such as the EPSRC, EU FP7 commission, and HORIZON 2020 encourage both multi-partner coalitions and development of high-quality software intended for distribution and reuse [14, 15]. While these trends are encouraging in terms of increasing knowledge exchange and reducing wasted or repeated effort, meeting the new requirements poses challenges for project leaders who need to ensure that work undertaken by independent organizations is coordinated to an appropriate degree in order to assure quality. Furthermore, a recent study [19] compared four research projects conducted by academics over the span of nine years, and concluded that there is a strong correlation between publication output and the software development effort invested three years prior to publication, thus highlighting the importance of software engineering practices to boost the number of papers stemming from large research projects.

In this paper we describe how developers on the HARNESS project [7] address this issue through an approach based on version-controlled configuration management, automated software deployment, and continuous integration. While software development in industry faces similar difficulties, there are differences in the objectives, incentives, and measures of success between commercial and academic research projects, and this work is intended primarily to address the needs of the latter. HARNESS is an EU FP7 research project with the objective of making it easier for cloud providers and consumers alike to take advantage of heterogeneous resources, including, computational accelerators (GPGPUs and FPGAs), programmable routers and heterogeneous storage devices. The key motivation for incorporating heterogeneity is to offer a richer context for price/performance trade offs, and to bring wholly new degrees of freedom to the cloud resource allocation and optimization problem.

A key challenge of the HARNESS project, and indeed most EU research projects [17], is that it requires bringing together specialists from a number of geographically distributed partner institutions in academia and industry. Individual components addressing different classes of heterogeneous resources or requirements can be developed independently, but there is a need to coordinate effort and ensure that API specifications are adhered to and consistently interpreted, so that components can be deployed in a such a way as to provide a coherent distributed computing infrastructure. While responsibility for the quality of the individual components is shared among a number of lead institutions and is managed by the work package leader for each component, there is a need to provide a way to test and evaluate the fully integrated deployment as well. Finally, as this is an academic research project, deployments and benchmarking experiments should be made as reproducible as possible in order to facilitate verification of research results. To address these challenges we present in this paper a development and operations (DevOps) workflow that allows: **(a)** teams of developers to work autonomously on specific parts of the software architecture; **(b)** automated testing of individual projects as well as the integrated system deployments; **(c)** reproducible automated deployment on heterogeneous and large-scale testbeds.



*QUDOS'15*, September 1, 2015, Bergamo, Italy
ACM. 978-1-4503-3817-2/15/09...$15.00
http://dx.doi.org/10.1145/2804371.2804372

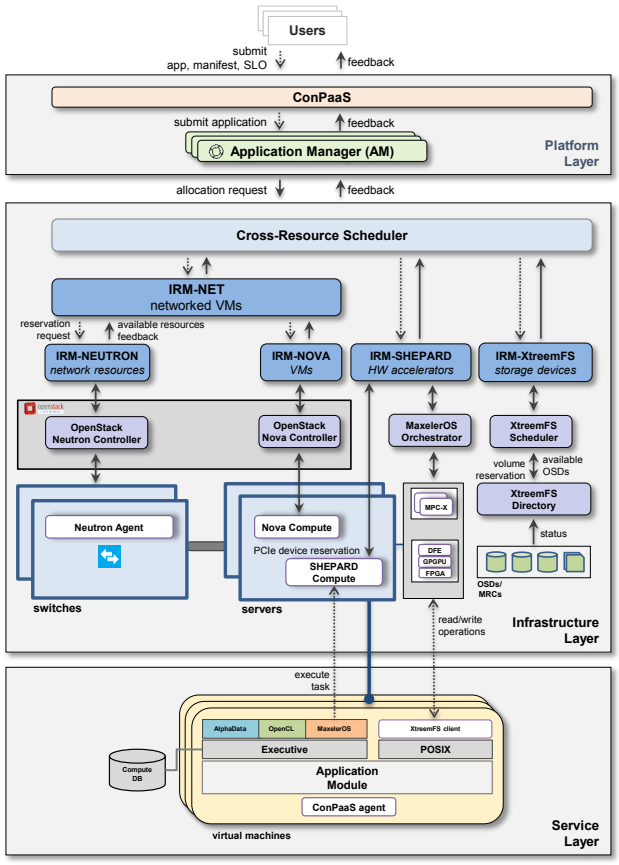



**Figure 1: The HARNESS cloud architecture consists of a suite of loosely coupled distributed services that interact with each other through an HTTP/REST API. This microservice architecture puts considerable overhead in managing the development and deployment processes.**

The remainder of this paper is structured as follows. In Section 2 we describe the HARNESS cloud architecture. and explain why a DevOps approach is needed to maintain high quality output. In Section 3 we present a high-level view of the HARNESS DevOps workflow. In Section 4 we describe how various deployment tools help achieve reproducibility and why this is important for testing and quality assurance. In Section 5 we report our automated testing infrastructure and our novel methodology for testing full systems deployments. In Section 6 we describe two platforms where HARNESS is being deployed. Finally, we conclude with a summary and plans for future work in Section 7.

## 2. HARNESS CLOUD ARCHITECTURE

The HARNESS cloud architecture is divided into three main parts: a *platform layer* in charge of managing applications that works on behalf of the cloud tenant, an *infrastructure layer* responsible for managing resources that works on behalf of the cloud provider, and a *service layer* where cloud applications are deployed and executed. An example HARNESS cloud platform, currently being developed as a proof-of-concept, is presented in Figure 1. This platform combines (a) VM and network resources managed by OpenStack [10], (b) networked FPGA devices managed by MaxelerOS Orchestrator [9], (c) OpenCL accelerators managed by SHEPARD [21], and (d) heterogeneous storage devices managed by XtreemFS [23].

The HARNESS architecture is designed to be open and modular, allowing arbitrary types of resources (compute, storage and network) to be integrated and leased to cloud users. Each resource type has a specially designed *Infrastructure Resource Manager* (IRM) that understands its internal semantics while presenting a uniform API to the *Cross-Resource Scheduler* component (CRS). The CRS acts as the central scheduling module and has a global view of all resources available in the data-center. This structure, wherein a larger project is made up of a suite of fine-grained collaborative services, each running on its own process and communicating with each other through a well-defined HTTP/REST API, is commonly referred to as a *microservice* software architecture [18]. As a consequence of this design decision, services are loosely coupled, allowing different developer teams to work and maintain each service autonomously. As an alternative to the microservice architecture, there is the *monolithic* architecture, in which applications can be realized into a single logical executable.

Cloud tenants submit their applications and performance/cost objectives through the ConPaaS frontend interface [4]. ConPaaS is responsible for providing an entry point to cloud users, as well as handling user authentication and creating a context for application management. An *application manager* instance is generated whenever an application is submitted on the HARNESS platform, and is responsible for overseeing the life-cycle of the application. For instance, the application manager automatically runs a suite of micro-benchmarks to generate a performance profile for a submitted application, and then works with the CRS module to determine the best way to run it, taking into account both user-specified performance and cost objectives and the demands of competing users. Once resources have been provisioned by the CRS, the application manager deploys the application on allocated VMs. The application can access other resources (such as GPGPUs, FPGAs and heterogeneous storage) allocated in the previous step by interacting with management systems (daemons) running on the VMs.

While there are many benefits to following a microservice architecture, it does require each service to be built and tested individually, and then deployed with other services in order to ensure that all these modules stay up and collaborate with each other. Hence, managing and rolling out all these services puts considerable overhead on the development and operations processes, requiring a high degree of deployment automation to ship and configure the integrated system. While this problem is also commonly encountered in real world systems, dealing with it is still an area of active development and research and there are no widely accepted standard solutions. In the next section, we describe our development and operations processes in the context of the HARNESS project.

## 3. DEVOPS WORKFLOW

Figure 2 illustrates the HARNESS DevOps workflow, which captures the process of developing the HARNESS cloud system from version control to release. Our key goal is to allow different teams of programmers to develop and maintain each service autonomously, while also enabling collaboration between teams and providing feedback as soon as possible about the flow of changes and their impact on the combined system. The HARNESS project has four *development* teams and one *integration* team. Each development team is responsible for maintaining a specific part of the architecture belonging to one of four technical areas: compute, network, storage and platform. The integration team is responsible for ensuring that all these services are properly tested when combined and deployed on the HARNESS testbeds (see Section 6).

All the development and integration teams share a single HARNESS **GitLab** server. GitLab is a web-based Git repository manager

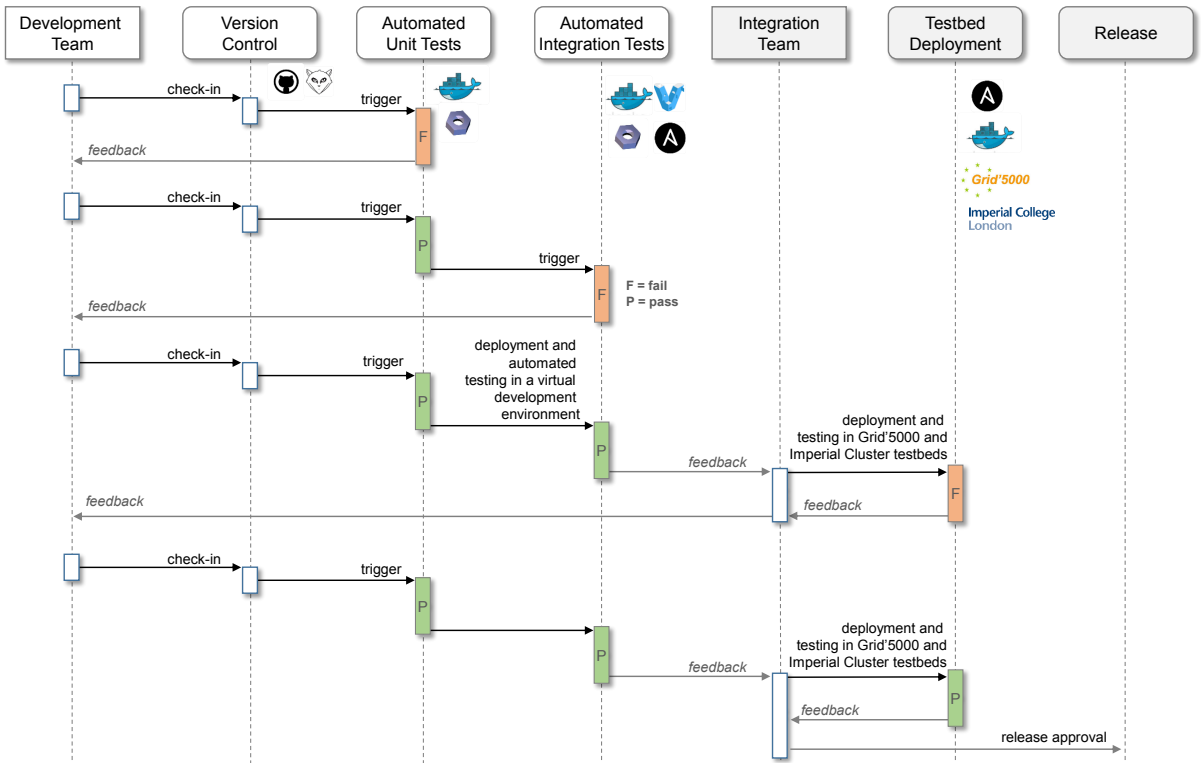



**Figure 2: The HARNESS DevOps workflow.**

that supports multiple users, groups, and owner-specified access controls for repositories. GitLab provides an open-source alternative to GitHub that can be installed on private infrastructure. With GitLab, each developer team stores a specific project (such as an architecture component or an automated deployment project) in its own **Git** repository, with all HARNESS software contributions aggregated into a single central server acting as the authoritative reference. In HARNESS, each project has one *owner* who is responsible for maintaining the project and has exclusive access to the master branch. The *contributors* (other members of the development team) can only make changes, such as adding experimental features or fixing bugs, by forking the master branch. Hence, rather than logging an issue, contributors can fork (copy) the repository, make updates, and then submit a pull (merge) request to the project owner. The project owner can then review the changes and accept or reject the merge request by exploiting GitLab's advanced tracking of the relationship between forks and its code reviewing facilities.

Whenever the owner of a project pushes a commit to GitLab or merges in changes, it triggers a set of automated tests (see Section 5). First, unit tests associated with each project are executed. If any of these tests fail, then the project owner is notified. Otherwise, automated integration tests are queued to run at specific times. In this case, an integration deployment project pulls the latest version of all HARNESS components from various master branches, deploys the HARNESS software stack on virtual machines, and runs tests that aim to exercise all critical features of the system. If any integration test fails, then both the integration team and the project owner (whose commit triggered the integration tests) are notified by email.

Periodically, the integration team deploys and tests the HARNESS cloud system in two HARNESS testbeds (see Section 6), namely Grid'5000 and the Imperial Cluster, depending on the types of updates submitted by the development teams. This process is manually initiated but almost entirely automated. In the case of Grid'5000, deploying HARNESS requires requesting nodes from the batch scheduler and provisioning them with a base Ubuntu 14.04 operating system [16], but after that point the deployment infrastructure can take over to install and configure all of the HARNESS software and its dependencies without human intervention. If any deployment test fails, then the integration team writes one or more integration tests that can flag a particular fault the next time the tests are executed. In addition, the integration team also notifies the development team of any bug, who in turn can write one or more unit tests to flag the problem at the component level.

## 4. REPRODUCIBLE DEPLOYMENT

One of the defining characteristics of a DevOps-based approach is that the testing environments should reliably reflect the production environments. That is, the environments themselves should be, to the extent feasible, *stateless* and *reproducible*. Statelessness in this case refers simply to the idea that the runtime environment should not change over time in a way that might affect the behavior of running applications. It is difficult, if not impossible, to achieve true statelessness on real-world machines, but many of the same benefits can be achieved by isolating running services from each other and the host environment through either virtualization or operating system specific methods of containerization. In order to achieve reproducibility we focus on automation and version control. Automation allows us to ensure that all configuration steps are fully documented, while version control lets us see how configurations have evolved over time, and potentially to review the differences in configuration between running systems. An additional benefit of automation is that it can ease deployment to new target systems, potentially increasing the number of operating production systems and reducing downtime in the event of catastrophic failure.

## 4.1 Containerized Services

An operating system container is an isolated environment provided by an operation system kernel rather than a hypervisor. While hypervisors achieve isolation between environments through high-overhead techniques like intercepting interrupts in order to provide the appearance of a physically isolated machine, containers provide a lighter-weight approach. In the case of the Linux operating system, containerization is primarily achieved by replicating various kernel data structures to provide separate *namespaces* to running processes. From the perspective of the operating system, processes running within containers are no different from processes running outside of containers, they just have a more constrained view of the system. Thus, containerized services can run in isolation at native or near-native performance (there may be some small overheads due to an extra layer of abstraction for some operations) [24]. Containers have an additional benefit in that, since services can be launched directly from the host, and links can be made easily between containers or between a container and the host [5], there is no need to deploy a complete stack of running services in every container, which improves efficiency as compared with full virtual machines. While some work may be needed to give containers direct access to hardware devices, it is nonetheless simpler to do so than to implement similar functionality for virtual machines. Some drawbacks of containers relative to virtual machines are that environments must all share the same kernel version, and there may be some difficulty in management of access to the kernel module space.

**Docker** is a software technology for managing containerized software deployments. As with OpenStack, Docker provides an interface and various databases to track conceptual objects, while leaving implementation primarily up to other lower-level technologies [5]. Application specific software runtime environments are described in a "Dockerfile" that can be committed to the software repository or maintained in a separate project. Dockerfiles give instructions for reproducibly creating an image from a standard binary base. There are base images available representing the environments provided by most of the mainline Linux distributions, though it should be noted that applications may not function in precisely the same way within a container as on the equivalent full operating system. In particular, through experience we have learned that in the standard Ubuntu image the upstart service does not work correctly, so daemons need to be started either directly or by using a third-party application.

The main advantages of deploying services within containers rather than directly on the host system are 1) that the services themselves are implemented and tested within the same environment, which includes all dependencies and so there is no need to worry about the configuration of the host and 2) deploying services does not affect the host's operating environment, and so there is no concern, for example, that deploying a service B will result in breaking some unrelated service A because of dependencies on incompatible libraries. This latter benefit also means that services can be un-installed and reinitialized cleanly, without worrying that they are leaving behind old versions of data or configuration files that may affect the running of future service deployments. Within HARNESS, many components are implemented as python daemons that need to talk or be available on different network interfaces, which is the ideal situation for docker based deployment. In recent versions of the platform we are moving away from simply running these daemons directly on the host to having Dockerfiles embedded within the projects and having docker as the preferred means of deployment.

## 4.2 Service Orchestration and Configuration

In recent years there has been a movement toward increasing the use of automation for systems administration and configuration management tasks. This has been motivated by a number of factors, including: the difficulty of tracking configuration changes across multiple systems, the need to ensure configurations are applied consistently to ephemeral cloud-based systems, and the desire to make configurations reproducible across platforms. Leading technologies in this area include Puppet [11], Chef [3], Salt [12], and Ansible [1], among others. While each of these has its advantages and disadvantages, **Ansible** stands out for its relatively low barrier to entry for new projects: client systems need only ssh and Python; there is no need to install a client service or manage a separate trusted certificate registry as with Puppet or Chef; and it is conceptually simple: nodes are listed and categorized into groups within an "inventory" (usually a static configuration file, but potentially a dynamic script), while configuration changes are described within "playbooks" as sequences of tasks applied in parallel to one or more nodes or groups of nodes, with checks implemented within modules to ensure idempotency (that is, if a configuration change is applied once then it should not be applied a second time, even if the same set of Ansible tasks are run multiple times on the same system) [1]. In Puppet, by contrast, there is a need to fully describe dependencies between configuration directives to ensure that changes are consistently applied in the same order when there are possible side-effects [20].

As discussed in the previous section, container-management technologies like Docker are extremely useful for creating reproducible environments for individual services, but there is still a need for higher-order orchestration and configuration management. For one thing, not every service can be deployed within a Docker container: examples include Docker itself (which can run in a container, but there first needs to be an installation on the base operating system) and services that need to cross standard container boundaries, like OpenStack Neutron, which must be able to control the host networking interface. It should be emphasized that services like Neutron *can* run in containers, but getting them to function correctly requires significant effort and is not yet widely supported. Another reason that an orchestration is required is that even if all services are containerized, there is still a need to place them on particular hosts and make sure that required configuration information (particularly secret information, like passwords) is distributed correctly to the services that need it.

The HARNESS deployment project contains several Ansible playbooks and related configuration files describing how the various components are instantiated on different nodes within the distributed system. There is an inventory file for each deployment target, including the automated testing environment. Each inventory groups hosts in the target environment by services run in the deployment and sets deployment-specific configuration variables. Currently, all of the deployment targets make use of the same set of playbooks. The "getreqs.yml" playbook first fetches related projects, or roles, each of which describes the tasks required to instantiate a number of standard services: mysql (database), rabbitmq (messaging), docker (container management), keystone (OpenStack authentication and identity), glance (OpenStack virtual machine image service), nova controller (OpenStack virtual machine frontend API and management services), neutron controller (OpenStack virtual network frontend API and management services), neutron network (OpenStack virtual network gateway service) and nova compute (OpenStack virtual machine management service). The source code and Dockerfiles for each of the HARNESS services are also fetched, so that these can ultimately be synced to appropriate target nodes in order to build the required Docker images. The "deploy.yml" playbook contains instructions for actually deploying the HARNESS platform, while the "test.yml" playbook should be run after the deployment to ensure that the integrated system functions as expected.

### 4.3 Virtual Machine Environments

Yet another advantage of automated, reproducible deployment is that it makes it possible for developers to create personal testbeds so that they can see how their individual components function within the larger system and make changes without fear of causing problems for others. The most practical way to go about this is to provision the full system in a virtual machine based environment on the developer workstation—this way the environment can be destroyed and recreated in a pristine state relatively quickly, without having to worry about reconfiguring the base operating system on a physical system. Of course, setting up virtual machines, particularly multiple machines connected to each other by virtual network links, is in itself a complicated process, but fortunately one that is also amenable to a certain level of abstraction and automation.

**Vagrant** is used to manage the creation and configuration of virtual machines for testing environments (see Section 5). Vagrant is a software tool that provides a simple and consistent configuration and command line interface for developers to manage virtual machines [13]. For each project, a "Vagrantfile" is created within the root directory of the software project and can be included within the source control system. After this, developers can simply run "vagrant up" to have all of the virtual machines required for local testing made available, with networking and local file synchronization set up automatically. Vagrant has hooks that are configured to call Ansible to configure a full HARNESS software deployment automatically. Critically, Vagrant can be used to manage the creation of both local virtual machines in VirtualBox on developer workstations and virtual machines that run within the testing environment.

## 5. AUTOMATED TESTING

Automating tests is a fairly standard procedure with the aim of minimizing the number of defects of a software system within a standard environment. These tests need to ensure that not only individual services or packages function correctly, but that the individual parts work together as part of a coherent whole. In this context, development and integration teams can be automatically notified by email or access a web service interface to verify if their projects are passing—or failing—the automated tests.

There are a number of standard solutions for automating testing and continuous integration. Perhaps the most popular and widely deployed of these is Jenkins [8]. However, the normal way to configure Jenkins and set up projects is through its interactive web console, whereas we wished to take a DevOps-oriented approach and manage the configuration and deployment of the testing environment using Ansible. For this reason, along with its Python implementation, we use **Buildbot** to provide an automated testing service for the HARNESS project.

Figure 3 illustrates the HARNESS Buildbot architecture. With Buildbot, it is possible to define a number of automated tests to run, and to have those tests invoked whenever changes are made to a specified repository. The architecture of Buildbot consists of one or more *master* nodes which monitor repositories and generate tasks, and multiple *buildslave* nodes to run queued tasks [2].

For the HARNESS unit tests, the Buildbot master monitors specified projects. When changes are detected, Buildbot checks them out, and looks for a `run_tests.sh` script in the root directory. If such a file is found then the script is run within a **Docker** container using the standard Ubuntu 14.04 image as illustrated in Figure 3(a). The buildslaves, in this case, run the unit tests on standard virtual machines deployed within the Imperial DoC cloud infrastructure.

The integration tests, on the other hand, require all the HARNESS cloud services to be deployed, including OpenStack, incurring a

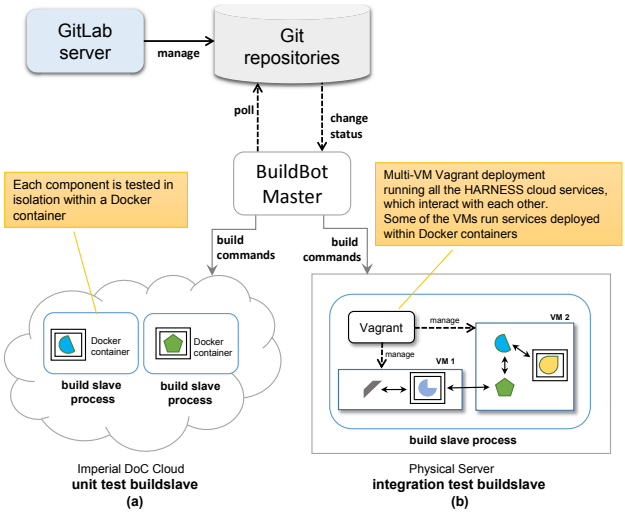



**Figure 3: Automated testing workflow using Buildbot. Individual components tested include the services illustrated in Figure 1, including the Application Manager, the Cross-Resource Scheduler, and the IRM components which interface various resource managers.**

higher runtime overhead than running isolated unit tests. For this reason, we run the integrated tests on dedicated physical nodes as illustrated in Figure 3(b). In this case, the Buildbot service runs Vagrant with the same configuration file as used by developers to create and test systems locally, and the output is monitored to ensure both that the Ansible configuration and testing scripts run successfully, and that the implementation is idempotent (that is, that subsequent runs of the `vagrant provision` command do not result in changes to the configuration of the deployed virtual machines). Our current automated integration tests run on top of multiple Vagrant spawned VMs, where some of these VMs run HARNESS cloud services within Docker containers. These integration tests are triggered and queued whenever a change is pushed to any project on which the integrated environment depends. However, due to high overheads, these builds are only run when all of the required projects have passed their unit tests, to a maximum of a few times per hour.

## 6. HARNESS TESTBEDS

There are two main deployment testbeds for the HARNESS cloud platform: the Imperial Cluster, which offers a relatively small-scale static testbed with heterogeneous compute and storage devices, and Grid'5000, which is a large-scale research testbed to support parallel and distributed computing experiments.

The Imperial Cluster testbed infrastructure, which is partly managed by the Custom Computing Group at Imperial College London, consists of a total of 6 compute nodes, 16 CPU cores, 2 GPGPUs, heterogeneous storage devices (HDD and SSD), and 3 MPC-X boxes harboring a total of 24 Dataflow Engines (DFEs). A Dataflow Engine is a general purpose reconfigurable device using an FPGA at its core and RAM for bulk storage. In our testbed, DFEs are co-located in MPC-X appliances which are in turn connected to select hosts via an Infiniband network. Dataflow computing technology [22] has been used successfully in fields such as oil and gas exploration and financial risk analytics, while research has been conducted in scientific areas as diverse as fluid dynamics and quantum chemistry.

The other main deployment target, Grid'5000, is a large-scale research testbed to support parallel and distributed computing exper-

iments. This testbed is distributed across 10 sites (mostly in France), with 1000 compute nodes and 8000 cores. It features a diverse set of technologies, including 10G Ethernet, Infiniband, GPUs, Xeon PHI, and data clusters. One key capability of Grid'5000 is that it is highly reconfigurable, providing bare-metal deployment that allows a fully customized software stack (including the operating system) and isolation at the network layer [16].

Deploying HARNESS to Grid'5000 requires an almost fully automated approach. The allocation is ephemeral, so there is an emphasis on speed of deployment and a need to dynamically generate an inventory. The dynamic nature of Grid'5000 means that there is no single set of static nodes that form the test bed. Rather, developers must request sets of nodes and other resources from the OAR batch scheduler. The following commands show an example of how HARNESS can be deployed on Grid'5000:

```
% oarsub -t deploy -I \
         -l slash_22=1+cluster=1/nodes=3,walltime=4:00:00
% kadeploy3 -f $OAR_NODE_FILE -e ubuntu-x64-1404 -k
% ansible-playbook -i inventories/g5k.sh deploy.yml
```

The first command (`oarsub`) requests a reservation of three Grid'5000 nodes, all on the same cluster, with a /22 subnet, for 4 hours. The second command (`kadeploy3`) puts a fresh install of Ubuntu 14.04 on the reserved nodes. Finally, the third command (`ansible-playbook`) deploys the HARNESS cloud to the reserved nodes. The `g5k.sh` script, which is passed as an argument, is a shell script that reads environment variables set by the batch job scheduler to dynamically generate an inventory based on the user's reservation.

# 7. CONCLUSION

In this paper we have described the development and deployment infrastructure created to support the integration effort of the FP7 HARNESS project. This infrastructure addresses a number of challenges commonly found in EU research projects that aim to develop high-quality software intended for distribution and reuse, with a focus on dissemination activities (such as proof-of-concept demonstrations and experiments for research papers) rather than following the imperatives of commercial clients. Currently, the primary deployment targets for HARNESS consist of a static testbed hosted at Imperial College London and Grid'5000, which presents a larger-scale but ephemeral environment with direct access to machine hardware. Future plans for the project include deploying HARNESS to the EGI Federated Cloud, a multi-site cloud computing infrastructure for research within the European Union [6].

While it is difficult to fully quantify how our DevOps workflow has affected our development process, and we are only now starting to collect metrics, we can see a qualitative difference in the way teams operate as compared to the previous year, before putting in place automated testing and deployment. In particular, in the beginning of this year considerable changes in the HARNESS cloud architecture where planned with the introduction of a new API specification and updated features such as monitoring and feedback, which required extensive modifications across the whole cloud software stack. We found development teams more able to operate autonomously, while simultaneously being more willing to accept changes and providing more frequent updates. There also seemed to be a reduction in communication overhead for making coordinated changes. The combination of automated testing and deployment appears to have contributed greatly to improving the speed and efficiency with which we converge toward the project milestones.

# 8. ACKNOWLEDGMENTS

This work was supported by the European Union Seventh Framework Programme under Grant agreement number 318521.

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see `https://www.grid5000.fr`).

# 9. REFERENCES

[1] Ansible: DevOps made simple. `http://ansible.com`.

[2] Buildbot: The continuous integration framework. `http://buildbot.net`.

[3] Chef: Automation for web-scale IT. `http://chef.io`.

[4] ConPaaS Project. `http://www.conpaas.eu/`.

[5] Docker. `http://docker.io`.

[6] European grid infrastructure federated cloud. `http://www.egi.eu/infrastructure/cloud/`.

[7] FP7 HARNESS project. `http://www.harness-project.eu/`.

[8] Jenkins: An extensible open source continuous integration server. `http://jenkins-ci.org`.

[9] Maxeler Technologies. `http://www.maxeler.com`.

[10] OpenStack: Open source software for creating private and public clouds. `http://openstack.org`.

[11] Puppet Labs. `http://puppetlabs.com`.

[12] SaltStack. `http://saltstack.com`.

[13] Vagrant: Development environments made easy. `http://vagrantup.com`.

[14] European Commission: Software technologies, the missing key enabling technology, 2012. `http://cordis.europa.eu/fp7/ict/docs/istag-soft-tech-wgreport2012.pdf`.

[15] EPSRC: Software as an infrastructure, 2015. `https://www.epsrc.ac.uk/newsevents/pubs/software-as-an-infrastructure/`.

[16] D. Balouek et al. Adding Virtualization Capabilities to the Grid'5000 Testbed. In *Cloud Computing and Services Science*, volume 367, pages 3–20. 2013.

[17] A. Bubeck et al. Implementing Best Practices for Systems Integration and Distributed Software Development in Service Robotics. In *IEEE/SICE Inter. Symp. on System Integration (SII)*, pages 609–614, Dec 2012.

[18] M. Fowler. Microservices. `http://martinfowler.com/articles/microservices.html`.

[19] D. Groen et al. Software Development Practices in Academia: A Case Study Comparison. *CoRR*, abs/1506.05272, 2015.

[20] R. Harrison. How to avoid Puppet dependency nightmares with defines. `http://www.webcitation.org/6a1doDLla`.

[21] E. O'Neill et al. Cross resource optimisation of database functionality across heterogeneous processors. In *Proc. IEEE on Parallel and Dist. Processing with Applications*, 2014.

[22] O. Pell et al. Maximum Performance Computing with Dataflow Engines. In *High-Performance Computing Using FPGAs*, pages 747–774. Springer, 2013.

[23] F. Schintke. XtreemFS & Scalaris. *Science & Technology*, (6):54 – 55, 2013.

[24] M. G. Xavier et al. Performance Evaluation of Container-based Virtualization for High Performance Computing Environments. In *Proc. of Euromicro Inter. Conf. on Parallel, Distributed and Network-Based Processing (PDP)*, pages 233–240, 2013.

# DevOps Meets Formal Modelling in High-Criticality Complex Systems

Marta Olszewska
Åbo Akademi University
Joukahaisenkatu 3-5A
20520 Turku, Finland
+358442806858
mplaska@abo.fi

Marina Waldén
Åbo Akademi University
Joukahaisenkatu 3-5A
20520 Turku, Finland
+35822154675
mwalden@abo.fi

## ABSTRACT
Quality is the cornerstone of high criticality systems, since in case of failure not only major financial losses are at stake, but also human lives. Formal methods that support model based-development are one of the methodologies used to achieve correct-by-construction systems. However, these are often heavy-weight and need a dedicated development process. In our work we combine formal and agile software engineering approaches. In particular, we use Event-B and Scrum to assure the quality and more rapid and flexible development. Since we identified that there are more prerequisites for a successful IT project, we use DevOps to embrace the development, quality assurance and IT operations. In this paper we show how formal modelling can function within DevOps and thus promote various dimensions of quality and continuous delivery.

## Categories and Subject Descriptors
D.2.1 [Software Engineering]: Requirements/Specifications — Elicitation methods, Methodologies; D.2.4 [Software Engineering]: Verification methods — Formal methods; D.2.10 [Software Engineering]: Design — Methodologies.

## General Terms
Management, Design, Reliability, Verification.

## Keywords
Agile, Scrum, formal modelling, Event-B, DevOps.

## 1. INTRODUCTION
Development of complex systems of high-criticality requires not only vast domain knowledge, but also use of appropriate methods and tools that would ensure high quality of the produced systems. Therefore, formal methods are often employed to build software models in the early phases of the software development and guarantee that the system is correct by construction. However, nowadays there are other requirements added to the development, i.e. reducing friction in the development time, delivering artefacts faster, improving communication within development team and with stakeholders. Formal methods alone provide firm software engineering approaches; however, they need support on behalf of the development process and resource management.

Formal methods are mature enough and ready for being integrated in the development with other methods [1]. Agile methods, on the other hand, are the most appropriate means for engineering such a merge [2]. However, they both need some more support for the interdependencies within the development, including issues regarding a project set-up (tools, methods and decisions on collaboration), overcoming the learning curve of formal methods, identifying bottlenecks and handling standardization issues, etc. The merge of formal methods and agile approaches is meant, among others, to speed up the delivery of artefacts while ensuring their quality.

In our previous work we have deepened the understanding of agile concepts set in the context of safety-critical development by (i) providing evidence of such development through related work and (ii) relating agile principles, practices and values to formal environment in order to create a synergy between these two (FormAgi framework) [3]. We chose Event-B as a formal method to be used within the agile development process.

Here we continue our work with the FormAgi framework by setting up a Scrum-based process for development of systems of high criticality with the use of Event-B. We present the merge of formal modelling with Scrum in DevOps perspective. We show how formal modelling can function within DevOps and contribute to its quality assurance and continuous delivery. We also point out how DevOps can support certain aspects of formal development.

This paper is structured as follows. Section 2 provides description of approaches for achieving shorter release cycles leading towards continuous delivery and integration. Section 3 describes methods supporting correctness and quality of development. In section 4 we present how we adapted Scrum to a formal modelling context. Section 5 depicts formal modelling with Scrum in perspective of DevOps. We conclude our paper in section 6 and provide directions for future work.

## 2. STRIVING FOR RESPONSIVENESS

### 2.1 Iterative and Flexible Development
Agile software development philosophy [4] introduced in 2001 occurred to be successful due to the capabilities of substantially increasing the project success rate, while reducing the development time and cost. Although criticised for disregarding existing practices of traditional software engineering, it is still popular, mainly because of its responsiveness and ability to meet stakeholders' needs within the given time. Agile methods are known for facilitating the collaboration within the development



*QUDOS'15*, September 1, 2015, Bergamo, Italy
ACM. 978-1-4503-3817-2/15/09...$15.00
http://dx.doi.org/10.1145/2804371.2804373

team and boosting internal morale. Moreover, the communication between the team and stakeholders is strengthened, thus leading to higher customer satisfaction [5].

There is a plethora of agile software development methods available, all of which encourage adaptive planning, evolutionary development, early delivery, continuous improvement, and promote rapid and flexible response to change [3]. The methods themselves need to be carefully chosen and tailored to the needs of the case development in order to be successful.

Even though agile methods are meant to improve product quality by e.g. leading to less defects in the development, there are still ongoing discussions whether this is really the case [6]. Some projects, for instance the ones of high-criticality, need to adhere to certain standards and follow well-defined practices [7] [8].

Boehm et al. [9] stated that neither the agile nor the traditional disciplined approach can alone be the optimal solution. Although they seem contradicting [10], a software project needs both agility and discipline [9] [11]. Therefore, a comprehensive approach is essential to embrace all the necessary methods and raise the understanding what is needed in the development – from the perspective of tools, methodology and people.

## 2.2 A Synergy Demand – DevOps

A rapid creation of software products and services, as well as improvement in operations performance, is at the core of today's IT. DevOps [12] answers these needs by putting special emphasis on collaboration, integration, communication and automation. There is no clear definition of DevOps yet [13], however, it is described as a software development method that combines quality assurance mechanisms with IT operations within software engineering practices (see Figure 1). It is based on the ideas from agile development movements and supports rapid development and deployment cycles.

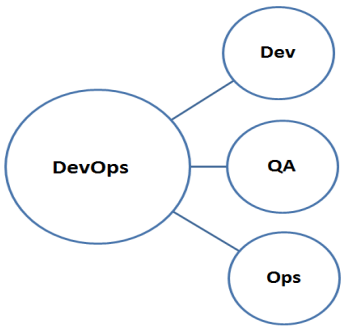



**Figure 1. DevOps concept**

Since systems become more complex, we need to increase our *comprehension* of what needs to be mapped from the empirical world to the model or code, as well as how to do it effectively. Moreover, we should improve our *communication and collaboration*, so that we can (i) raise the understanding of the system to be developed, (ii) foster awareness between people involved in the development and operations, (iii) progress with our development, and (iv) build our infrastructure. We require tool support for our work, to *automate* some of our activities. And finally we need to understand that our models, code, teams and organization are constantly in flux, thus we need to continuously address all the above mentioned issues.

# 3. CORRECTNESS AND QUALITY

There are various methods available for assuring quality in systems of high-criticality. Application of formal methods [14] brings high quality to critical systems, in particular when certain system behaviour and properties need to be guaranteed. However, some experience and mathematical background is needed in order to properly utilise the existing modelling solutions.

## 3.1 Iterative Formal Development

*Refinement* [15] [16] [17] [18] is a stepwise formal development method, which allows the system to be created iteratively following certain rules called *refinement rules* (also referred to as proof obligations) [19] [20]. *Stepwise refinement* is a top-down approach [16], which aids handling all the implementation matters and *complexity* by splitting up the problems to be specified and gradually introducing details of the system to the specification. In the refinement process, an abstract specification is created from requirements. It is then transformed into a more concrete and deterministic system that preserves the functionality of its specification in consecutive refinement steps. For each refinement step an invariant is given that states the properties of the system. Hence, also the invariant is created in an iterative manner. The refinement process is presented in Figure 2.

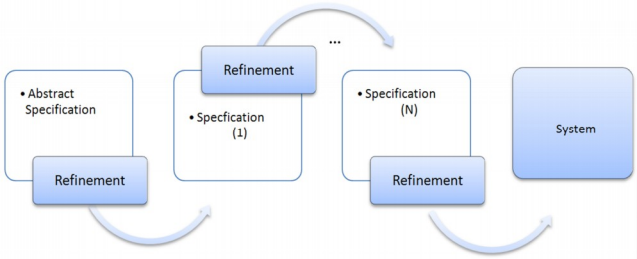



**Figure 2. Refinement process**

The correctness of each step, resulting in a system that is correct by construction [15], is ensured by mathematically proving that the abstract model is consistent and feasible. It involves proving that each refined model preserves its invariant. Even if proving is tool-supported, there are still some proofs that cannot be automatically discharged, but will require human interaction. The amount of involvement needed heavily depends on the chosen modelling strategy and is a subject of our current work.

The complexity of proofs depends not only on the problem and the complexity of the system to be modelled, but also on the refinement strategy utilised and e.g. on the decomposition mechanisms [21]. Therefore, assisting the modelling activity by facilitating the development process would help dealing with the complexity issues.

## 3.2 Modelling in Event-B

Event-B [22] is a formal method and modelling language for stepwise system-level modelling and analysis, based on the Action Systems formalism [18] [23] [24]. It is derived from the B-Method [25], with which it has several commonalities, e.g. set theory and the refinement idea. Event-B is dedicated to model complete systems, including hardware, software and environment [26] and has gained appreciation in industrial settings [27].

An Event-B specification uses a pseudo-programming notation – Abstract Machine Notation – and consists of a dynamic and a static part, *machine* and *context* respectively. The formal development starts from specifying an abstract machine from a set of requirements and then refining it in a number of steps (see Figure 2). It identifies the machine being refined, so that the refinement chain and the modelling process can be tracked and

controlled. The static part of the specification is also extended with respect to the development of the machine.

Event-B utilises refinement to represent systems at different abstraction levels, which enables us to gradually introduce details to the constructed system and to represent new levels of a system with more functionality. Mathematical proofs are used to verify consistency between the refinement levels. Event-B provides rigour to the specification and design phases of the development of critical systems. It is effectively supported via the *Rodin platform* [28], an Eclipse based tool, which is an open source "rich client platform" that is extendable with plug-ins, e.g. ones providing simulation and animation, as well as visualisations of the model. Finally, code generation from models to various programming languages is supported.

## 3.3 Standardisation

Systems of high criticality typically need to be qualified or certified in order to be deployed. Formal methods are recommended practices when licensing critical software, just to mention the IEC 61508 standard ("Functional Safety of Electrical, Electronic and Programmable Electronic Safety-related Systems") [29] or ISO 26262 standard for automotive domain ("Road vehicles — Functional safety") [30]. The application of formal methods is additionally followed by measures and techniques that are obligatory for the product to be certified.

Although agile methods are not explicitly considered in standards (e.g. IEC 61508 standard), their use in safety-critical development is already present and needs to be transferred to standards (see e.g. the goals of the European Project RECOMP [31]).

## 3.4 Need for Speed

Developing a system has never been an easy task, but now, when the complexity of the surrounding world is increasing and the requirements given by the stakeholders are expanding, it has become even more intricate [32]. There is a constant struggle between development cost, quality and time, where the choice is made on the expense of one of these characteristics (note that we do not consider cost in this paper).

Achieving high quality seems like a prerequisite for stakeholder satisfaction. However, the time pressure and need for fast delivery of a product can lead to "good enough" solutions and acceptable quality. This attitude cannot suffice for the systems of high criticality, where human lives or major financial losses can be at stake. Yet, development of this kind of systems is also required to be responsive to change, actionable, providing faster delivery, as well as enabling communication and collaboration. This as a consequence inevitably leads to the amendments in the development process, as well as organisational changes.

The formal modelling process itself is thought to be heavy-weight and thus may seem far from the *need for speed* that we experience nowadays. Therefore, a suitable development process, along with some principles and practices tailored for the specifics of the formal modelling environment and development domain is necessary to facilitate the change and aid in adapting to the challenges set by the contemporary IT world.

# 4. TOWARDS AGILE

A combination of (semi) formal and agile approaches has been discussed in a number of publications [9] [11] [7] [8] [33], where capabilities of traditional software development process models or development methods were merged with agile methods, principles and practices.

We observed that having a well-described and flexible development process is not sufficient for formal methods to be successful. Additional mechanisms are needed, which are specific to the development setting. In this section we first shortly describe the *FormAgi* [3] framework, which is the basis of our work. Next, we depict Scrum as the agile software development method of our choice and then adapt it to the formal modelling setting. Finally, we describe the merge of formal modelling of critical systems with the Scrum development process in the context of DevOps.

## 4.1 FormAgi Framework

In our previous work we explored the values, principles and practices of agile development methods and placed them in the context of formal, refinement-based developments. We analysed several of the agile methods with respect to their feasibility in development of critical systems. [3]

We provided a mapping between the characteristics of these two, which established FormAgi [3], a high-level framework consisting of (i) guidelines on what concerns should be tackled before committing to a certain agile method and (ii) pointers in which aspects an agile method can be a facilitator in the formal development. Additionally, we mentioned Event-B as a formal method of our choice to be utilised within the agile process. Although Event-B is thought as being far from lightweight approach, we argue that by conducting the development in small refinement steps [22], and by decomposing the models [21] e.g. using abstractions [40], it is a good candidate to be applied in a rapid manner. Our current work on an example is meant to investigate this claim.

## 4.2 Scrum

In this work we chose to use Scrum [34], an iterative agile development framework, which relies on frequent releases and short development cycles, as well as supports process improvement. Since some of the characteristics conceptually overlap with the ones of Event-B, e.g. iterations and refinement steps, we consider the integration of the two to be seamless.

One of the reasons for which we chose Scrum was the clear definition of time frames for iterations (organisation of sprints) and the set of meetings to be held during the development process [35]. An overview of a sprint within Scrum is shown in Figure 3.

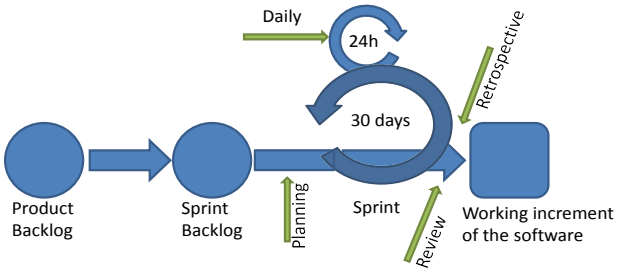



**Figure 3. The overview of a Scrum sprint**

During each sprint the development team takes a set of features from the product backlog, which is a set of high-level requirements. Then the stakeholder informs the developers of the features that should be completed in this iteration. From this subset the team selects the features that are realistic to be implemented. Next, when the goals for the sprint are determined, they are shifted to the sprint backlog, which remains fixed for the time of the sprint. The sprint lasts for a specific amount of time (2-4 weeks) and at the end it produces a potentially shippable product increment, which is presented by the team to the

stakeholder. Any of the unimplemented features are returned to the product backlog.

Communication, one of the cornerstones of agile approaches, is also important within Scrum. Various meetings are held throughout the development: before the sprint starts (Planning), during the sprint (Daily Scrum), after the sprint (Review and Retrospective). Retrospective is a process improvement-oriented meeting, while the other meetings concern the development itself.

In Scrum there is a strong involvement of the representative of the end user (or the stakeholder), so that at the end of each sprint the directions for further development can be indicated. The issue of how much functionality will be implemented during each iteration is controlled and decided by the development team. The contents of a sprint do not change during its duration. Thus, the moments at which functionality changes can occur are limited and well-defined.

The goal of this development methodology is to increase the relative effectiveness of development practices for improvement purposes and at the same time delivering a framework for development of complex products [36].

## 4.3 Adapting Scrum

One of the purposes of agile methods is to be tailored to fit the characteristics of the environment they are utilized in. These adjustments aid in getting the most out of the used methodologies and tools. In case of developing systems in Event-B we aim at smoothening the development by supporting shorter iterations, as well as facilitating the intra-project communication. Moreover, we consider supporting the communication between the team and the stakeholders as crucial to obtain high quality software.

Although formal modelling differs from coding in programming languages, creation of *artefacts* (be it a subset of requirements, specification, model on specific abstraction level or implementation of a feature) remains as the main goal.

We adapt Scrum to fit the specifics of Event-B development, which is depicted in Figure 4. In practice, formal modelling involves not only transforming requirements into models, which are then proven to be correct, but also elicitation and modelling of requirements themselves. For this reason we use the term *item*.

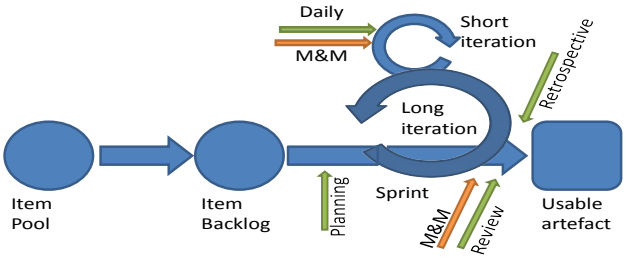



**Figure 4. Scrum Sprint adapted to formal modelling (within the FormAgi framework)**

The requirements in the *item pool* are considered as a set. The item pool acts as product backlog. However it contains not only high-level requirements, but also lower-level requirements, safety cases, environment descriptions. Furthermore, the *item backlog* is a subset of the item pool and consists of requirements chosen for this sprint, but not prioritised. Since prioritization may take more time than what is scheduled for regular Scrum process, we believe it should be done within sprints. The reasoning is twofold: (i) we do not want to rush decisions which would lead to a complex and hard to prove model and (ii) the work on the requirements and their structuring with respect to the modelling strategy will pay off later, when the model needs to be extended. Thus, a sprint includes modelling of the requirements, as well as developing and proving a model. For the verification purposes, model animation and simulation can also be a part of the sprint. It is well supported via plug-ins to the Rodin platform.

The duration of *long* and *short iterations* should be decided before the development starts. There is a risk that some requirement or property is too complex to be processed within a short iteration. In this case it should be discussed during the short *daily* meeting so that the team is informed. The sprint *review* resembles the discussions in a regular sprint. However, some issues like model walkthroughs, or demonstrating the results to stakeholder as model simulation or animation should also be included at this stage. The *retrospective* is meant to reflect upon the sprint and highlight the areas of the sprint for future improvement.

It can be noted that the relation between a refinement step and a development process iteration is certainly not a one-to-one mapping. There can be several refinement steps in one iteration. Moreover, a refinement step might be too large for an iteration, so that the problem to be modelled needs to be decomposed into sub-problems and only as such placed into the item backlog.

Finally, although not present in original Scrum, a feedback system is included in the sprint via the *Monitoring and Metrics* mechanisms *(M&M)*, which is to raise understanding on what is being done (short iterations), as well as to facilitate the process improvement and provide evidence on the development (long iterations). We are aware that metrics and measurements within agile developments are sometimes considered as harmful to the team morale and against agile philosophy. However, we use them for informational purposes rather than "plunger of blame".

# 5. EMBRACING IT ALL

Nowadays it is not only the software system development itself that matters, regardless if it is the work on requirements, specifying the system, modelling its properties and behaviour, or proving it afterwards. It needs certain methods and tools, practices and principles, as well as some external mechanisms for making the development work smoothly and progress towards creation of a quality product. In Figure 5 we present our view on formal modelling set in the DevOps context. Note that the formal development method is only a part of the whole development.

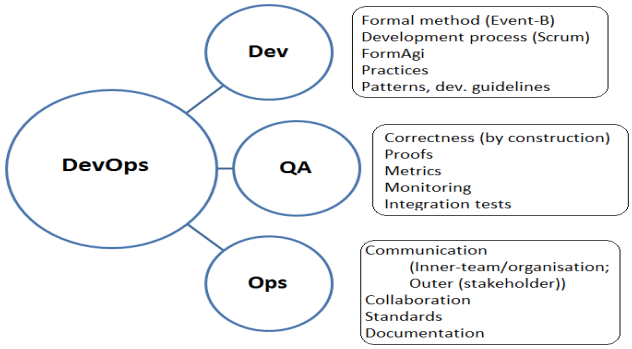



**Figure 5. Formal Modelling in DevOps context**

Apart from the formal method (Event-B), and the development process (Scrum) supported by the FormAgi framework, there are other necessary elements that need to be taken into consideration in order to achieve a quality product in the domain of highly-critical systems. DevOps encapsulates what we believe is vital for the development to be successful, from technical, social and quality perspectives.

While dependability, herein reliability, is ensured by Event-B, Scrum increases the speed of delivery of artefacts and facilitates their deployment. Iterations supported by the development process and model refinement promote continuous delivery. Once the model is created and proved throughout the development, all the modelling decisions are documented in a stepwise manner in form of refinement steps [37]. They can serve as artefacts for standardisation purposes. Moreover, runtime information in the Ops phase can be tracked all the way to requirements due to the way the system is developed. In case of a failure at runtime, it can be traced to a flaw in the modelling process, the specification or even the requirements. Modifications of a system by either adding new or altering existing components or requirements will cause the whole system to be remodeled and (partially) proved again.

We can facilitate the modelling and help continuous delivery of artefacts by application of patterns, be it the lower level ones [38], or the ones related to the modelling strategy [39]. We are currently working on a library of components to support the reuse and modularity of development.

We noticed that there is quite a natural transition from push to pull flow with respect to modelling requirements and building a model. We focus only on necessary properties related to a modelled artefact in a certain refinement step. Thus refinement helps us in (i) concentrating on what matters the most at a particular point in the development and (ii) matching the level of abstraction with the current development stage. As a consequence, we provide better control over the model complexity, and by that contribute to higher quality. Moreover, by employing requirement prioritisation and providing strategy in modelling, e.g. by decomposition and abstraction mechanisms [40], we avoid waste. So far we have identified two cases of generating it that can be avoided: (i) when insufficient time is spent on requirements modelling, it can lead to spending excessive time on modelling and then cause cumbersome proving; (ii) when detailing the model too early, it increases the complexity of the model and its related proofs.

Identifying bottlenecks and prioritising the improvement areas, which are identified as important in DevOps, are supported by the retrospectives in Scrum. We additionally propose monitoring and metrics to be used as a feedback system for improvement. So far we provide metrics for measurement of the complexity of the Event-B models [41], as well as their UML representations [42].

Visualisations of models are available via Rodin tool plugins for animation and simulation. This enables us to show the results of the modelling to team members and stakeholders after a sprint, without the need to provide executable code. The platform offers code generation to various programming languages, with different level of technical detail, once the model is at a lower level of abstraction.

Although tests are not necessary for the critical development modelled with formal methods (proofs are ensuring correctness), integration tests should be used in order to ensure quality in case the critical part is integrated with a non-critical one [31].

Post-mortems are one of the mechanisms that should be incorporated in the formal modelling and should be done with two groups (team and stakeholders). However, one would need an additional "check" mechanism that could be incorporated in the development process once a bigger milestone is achieved. Particularly, a more in-time feedback could be provided if this check is integrated with some other part of the system.

So far we have shown how formal modelling can work within DevOps, but we also see how DevOps (and agile methods) can contribute to formal modelling. What is often missing in the formal process is the emphasis on communication between the team members (and also with the stakeholders). After receiving requirements, be it safety cases or functional properties, the experts are singlehandedly working out *a good way* of modelling them, but usually without a common strategy for the development. We observed that intra-team communication largely enhances the modelling capabilities. We also observed that stand-ups are a good approach of pinpointing difficulties with the modelling or proving. Thus the collaboration means not only knowledge sharing, but also raising understanding and awareness in the project. It also leads to the concept of "reusable team", where the expertise of every group member is known and can be utilised whenever needed.

# 6. CONCLUSIONS AND FUTURE WORK

As observed by many DevOps followers, having no clear definition of this concept has its advantages. Firstly, it raises discussions, and secondly, it opens new possibilities of combining methods and tools within one high level development method, which in consequence enables to smoothen out the gap between development and operations. We believe that regardless of the application domain, it is important to utilise existing methodologies and tools through a well-structured merge (DevOps umbrella), so that not only the development is supported, but also the people, processes and the artefacts inevitably bound to it (operations and quality management tasks).

In our work quality in formal modelling within DevOps is achieved by building a correct-by-construction software system, incorporating monitoring and metrics mechanisms to the development process, providing tool support for modelling and proving, as well as facilitating communication and collaboration across organisation and with stakeholders.

We are currently planning to perform a case study, where we will employ all the methodologies and practices mentioned in this paper (see Figure 5). It will be a development of a high-criticality system, but executed in an academic environment. Moreover, we would like to investigate further what can be interpreted as a waste in formal modelling and how to minimise it. Since the information about how to model in Event-B is spread over a number of publications, we are currently working on providing more comprehensive guidelines, suitable also for inexperienced Event-B users. Finally, we believe that deeper research on how refinement steps relate to Scrum sprints has a potential of shortening the deployment time for artefacts.

# 7. ACKNOWLEDGMENTS

This work was carried out within the project ADVICeS, funded by Academy of Finland, grant No. 266373. We would like to thank PhD Mikołaj Olszewski, Vaadin Expert and a Scrum enthusiast, for constructive discussions on agile topics.

# 8. REFERENCES

[1] Larsen, Peter Gorm, Fitzgerald, John S., and Wolff, Sune. Are Formal Methods Ready for Agility? A Reality Check. In *Second International Workshop on Formal Methods and Agile Methods* (Pisa 2010), Springer.

[2] Paige, Richard F. and Brooke, Phillip J. Agile Formal Method Engineering. In *Integrated Formal Methods* (Eindhoven 2005), Springer.

[3] Olszewska, Marta and Waldén, Marina. *FormAgi – A Concept for More Flexible Formal Developments*. Åbo Akademi University, Turku, 2014.

[4] *Manifesto for Agile Software Development*. 2001.

[5] Olszewski, Mikołaj. *Scaling Up Stepwise Feature Introduction to Construction of Large Software Systems*. TUCS Dissertation Series, Turku, 2013.

[6] Olszewska, Marta, Jeanette, Heidenberg, Weijola, Max, Mikkonen, Kirsi, and Porres, Ivan. *Did It Actually Go This Well? A Large-Scale Case Study on an Agile Tranformation.* TUCS, Turku, 2014.

[7] Glazer, Hillel, Dalton, Jeff, Anderson, David, Konrad, Mike, and Shrum, Sandy. *CMMI or Agile: Why Not Embrace Both!* Carnegie Mellon University (CMU) and Software Engineering Institute (SEI), 2008.

[8] Fritzsche, M. and P., Keil. Agile methods and CMMI: compatibility or conflict? *e-Informatica, Software Engineering Journal*, 1, 1 (2007), 9-26.

[9] Boehm, Barry and Turner, Richard. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley, 2003.

[10] Turner, R. Agile development: good process ot bad attitude? In Oivo, Markku and Komi-Sirviö, Seija, eds., *Product Focused Software Process Improvement*. Springer, Rovaniemi, 2002.

[11] Boehm, Barry and Turner, Richard. Observations on balancing discipline and agility. (Salt Lake City 2003), IEEE Computer Society.

[12] Loukides, Mike. *What is DevOps?* O'Reilly Media, Sebastopol, 2012.

[13] Loukides, Mike. What is DevOps (yet again)? *Radar*, radar.oreilly.com (Feb. 2015).

[14] Butler, Ricky W. What is Formal Methods? In *NASA LaRC Formal Methods Program*. 2001.

[15] Dijkstra, Edsger W. A Constructive Approach to the Problem of Program Correctness. *BIT Nmerical Mathematics*, 8(3) (1968), 174-186.

[16] Wirth, Niklaus. Program Development by Stepwise Refinement. *Communications of the ACM*, 14(4) (1971), 221-227.

[17] Back, Ralph-Johan. *On the Correctness of Refinement Steps in Program Development; PhD thesis*. University of Helsinki, 1978.

[18] Back, Ralph-Johan. Refinement Calculus, Part II: Parallel and reactive programs. Stepwise Refinement of Distributed Systems. In de Bakker, J. W. et al., eds., *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*. Springer-Verlag, 1990.

[19] Metayer, Christophe, Abrial, Jean-Raymond, and Voisin, Laure. *Event-B Language, RODIN Deliverable 3.2 (D7)*. 2005.

[20] Waldén, Marina and Sere, Kaisa. Reasoning about Action Systems using the B-Method. *Formal Methods in System Design*, 13 (1998), 5-35.

[21] Yeganefard, Sanaz and Butler, Michael. Problem Decomposition and Sub-Model Reconciliation of Control Systems in Event-B. In *IEEE International Workshop on Formal Methods Integration* (Turku 2013).

[22] Abrial, Jean-Raymond. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.

[23] Back, Ralph-Johan and Kurki-Suonio, R. Decentralization of process nets with centralized control. *2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing* (1983), 131-142.

[24] Back, Ralph-Johan and Sere, Kaisa. From modular systems to action systems. *Software - Concepts and Tools*, 17 (1996), 26-39.

[25] Abrial, Jean-Raymond. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[26] Event-B, http://www.event-b.org/index.html. *Home of Event-B and the Rodin Platform*. 2008.

[27] Romanovsky, Alexander and Thomas, Martyn. *Industrial Deployment of System Engineering Methods*. Springer Heidelberg, 2013.

[28] RODIN and http://www.event-b.org/platform.html. *RODIN - Rigorous Open Development Environment for Complex Systems*. 2006.

[29] Commission(IEC), International Electrotechnical. *IEC 61508 1-7 - Functional Safety of Electrical, Electronic, Programmable Electronic Safety-related Systems*. IEC, 2010.

[30] ISO/FDIS. *26262 Road Vehicles - Functional Safety*. ISO/FDIS, 2011.

[31] RECOMP. RECOMP Project - Reduced Certification Costs Using Trusted Multi-core Platforms. In *https://artemis-ia.eu/project/21-recomp.html*.

[32] Olszewska, Marta. *On the Impact of Rigorous Approaches on the Quality of Development*. Turku Centre for Computer Science, 2011.

[33] Mandal, Ardhendu and Pal, S. C. Achieving agility through BRIDGE process model: an approach to integrate the agile and disciplined software development. *Innovations on System Software Engineering*, 11, 1 (March 2015), 1-7.

[34] Schwaber, Ken. *Agile Project Management with Scrum*. Microsoft Press, 2004.

[35] Shore, James and Warden, Shane. *The Art of Agile Development*. O'Reilly Media, Sebastopol, 2008.

[36] Schwaber, Ken and Sutherland, Jeff. *Scrum. The Official Guide.* Scrum.org, 2010.

[37] Pląska, Marta, Waldén, Marina, and Snook, Colin. Documenting the Progress of the System Development. (Heidelberg 2009), Springer-Verlag.

[38] Snook, Colin and Waldén, Marina. Refinement of Statemachines using Event-B semantics. In *Formal Specification and Development in B* (Besançon 2007), Springer.

[39] Iliasov, A., Troubitsyna, E., Laibinis, L., Romanovsky, A., Varpaaniemi, K., Ilic, D., and Latvala, T. Supporting Reuse in Event B Development: Modularisation Approach. In *Abstract State Machines, Alloy, B and Z: Second International Conference (ABZ)* ( 2010), Springer.

[40] Parsa, Masoumeh, Snook, Colin, Olszewska, Marta, and Waldén, Marina. Parallel Development of Event-B Systems with Agile Methods. In Mousavi, Mohammad Reza and Taha, Walid, eds., *Proceedings of 26th Nordic Workshop on Programming Theory, NWPT '14*. Halmstad University, Halmstad, 2014.

[41] Olszewska (Pląska), Marta and Sere, Kaisa. Specification Metrics for Event-B Developments. In *13th International Conference on Quality Engineering in Software Technology (CONQUEST 2010)* (Dresden 2010).

[42] Olszewska, Marta and Waldén, Marina. Measuring the Progress of a System Development. In Petre, Luigia et al., eds., *Dependability and Computer Engineering: Concepts for Software-Intensive Systems*. IGI Publishing House, Hershey, 2011.

# Modelling Multi-tier Enterprise Applications Behaviour with Design of Experiments Technique

Tatiana Ustinova
Imperial College London
Exhibition road, South Kensington
London, UK, SW7 2AZ
tatiana.ustinova12@imperial.ac.uk

Pooyan Jamshidi
Imperial College London
Exhibition road, South Kensington
London, UK, SW7 2AZ
p.jamshidi@imperial.ac.uk

## ABSTRACT
Queueing network models are commonly used for performance modelling. However, through application development stage analytical models might not be able to continuously reflect performance, for example due to performance bugs or minor changes in the application code that cannot be readily reflected in the queueing model. To cope with this problem, a measurement-based approach adopting Design of Experiments (DoE) technique is proposed. The applicability of the proposed method is demonstrated on a complex 3-tier e-commerce application that is difficult to model with queueing networks.

## Categories and Subject Descriptors
C.2.4, C.4, D.2.8, D.4.8

## Keywords
Multi-tier enterprise applications, design of experiments, two-level factorial designs, response surface models, linear regression, software performance testing

## 1. INTRODUCTION
DevOps is defined as a set of practices and principles bridging the gap between application development and operation stages [8]. One way to achieve this is continuous application performance modelling and prediction combined with automated feedback of the models to the developer and their update via continuous testing. A large body of work exists that employs Machine Learning algorithms [9] and tools, as well as linear regression, to obtain performance models based on measurements. In this paper we propose application performance modelling and prediction algorithm based on the Design of Experiments (DoE) technique.

DoE – widely used in engineering and industry for optimising processes – looks very promising for the use in DevOps, as it utilises measurements obtained at runtime to build performance models. These models can be fed to the application developer and updated in an automated way through continuous testing. However, its use is rather sparse in computer science, especially in the area of application performance modelling and prediction. This technique involves choosing a number of input parameters called 'factors', designing a set of experiments and then carrying them out on the system-under-study. The experiment results, called 'response variables', are then used to construct linear regression model representing a relationship between system output ('response variable') and inputs (factors). In this approach system under study is treated as a black box.

A number of studies exist that explore the capabilities of the DoE technique and DoE-based models in performance modelling, evaluation and prediction. Li et al. [4] presented a factor framework for performance evaluation of commercial Cloud services. This framework establishes factors that are currently used in the performance evaluation of clouds and can help facilitate designing new experiments for evaluating cloud services. However, this work does not provide any quantitative or qualitative assessment allowing to conclude which of these factors may be important for software performance testing.

Westerman et al. [10] apply statistical inference techniques to adaptively select experiments resulting in the optimal performance model. The approach automatically selects and conducts experiments based on the accuracy observed for the models inferred from the currently available data. The results demonstrate that this approach can automatically infer a prediction model with a mean relative error of 1.6% using only 18% of the measurement points in the configuration space. However, this work is focused only on the design of experiments and does not investigate predictive capabilities of the obtained model.

Molka and Casale [in revision] applied DoE techniques to generate response surfaces (non-linear models constructed using linear regression) that describe database performance as a function of workload and hardware parameters for in-memory databases. The response variables this study reported include response times, server utilisation, energy consumption and memory occupancy. They found out that the queueing network and response surface models yield mean prediction errors in the range 5%-22% with respect to response times and mean memory, but the accuracy for the latter deteriorates in response surfaces as the number of experiments are reduced, whereas model-based simulation is effective in all cases. This suggests that simulation can be more effective in performance prediction for in-memory database management. However, this queueing network model was tailored to describe in-memory database, which required significant effort and knowledge of the system under study.

The proposed method described in details in the following sections is based on the design of experiments technique, which is first used to establish the design space (screening procedure) – a set of factors and their low and upper bounds – that influence response variable(s). Then a linear regression is used to construct a model describing relationship between input parameters and performance metrics based on the experiment results obtained



*QUDOS'15*, September 1, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3817-2/15/09...$15.00
http://dx.doi.org/10.1145/2804371.2804374

during the screening. Afterwards, the model prediction accuracy is assessed. Additionally, the model prediction error is then compared to prediction made by the out-of-the-box Mean Value Analysis algorithm for queueing network models. To the best of our knowledge none of the work presented in this paper has been done before.

The rest of the paper is organised as follows. Section 2 presents the methodology of the proposed approach; Section 3 is dedicated to the case study – load simulation for the web-based e-commerce 3-tiered application; Section 4 provides analysis of the model prediction accuracy and discussion of the analysis results; Section 5 draws conclusions and gives suggestions for further work.

# 2. METHODOLOGY

Design of Experiments (DoE) starts with determining the objectives of an experiment and selecting the factors for the study. The choice of the experimental design would influence the amount of runs required to obtain sufficient information about system under test [1]. For example, if software performance tester is interested only in identifying the parameters that significantly influence application's performance, then two-level factorial design would suffice. The objective in this case would be to find out parameters (factors) that cause significant change in the output by shifting from one (low) level to another (high). Additionally, because in order to investigate all possible combinations of levels, $2^k$ runs (where k = number of factors in the experimental design) would be needed, the so-called fractional factorial designs are often used, where only a part (fraction) of the $2^k$ (full factorial) design is used. These designs, however, should be treated with care, as they are constructed under a number of assumptions, which may not hold for the given system.

Two-level factorial designs are also widely used for construction of linear regression models, but their use implies that relationship between system inputs and output is linear. If there is a chance that this relationship is not linear, other designs, allowing to construct polynomial regression models (e.g. Response Surface Methodology), might be considered instead. Therefore, it can be summarised that well-chosen experimental design would involve minimum possible number of runs required to obtain necessary information about the system under test. Also on this step response variables should be agreed on.

Taking into consideration everything said above, the following actions are needed to implement the method:

a) Define response variables: those would be performance metrics (e.g. response time, CPU utilisation, throughput);

b) Create design space via screening for important factors and their interactions using two-level fractional factorial design: choose a number of factors that might influence performance metrics, set the low and high levels for them (the levels are chosen based on the experimenter's experience and knowledge of the system).

c) Validate results of the screening with full factorial design for the chosen subset of important factors (may or may not require additional runs) and allocation of variation [7]. Allocation of variation shows how much variation each of the factors causes in the response variable when changed from low to high level.

d) Construct linear regression model based on the experiment results from b) and c) (may or may not require additional runs).

# 3. CASE STUDY

## 3.1 Objective

The objective of this case study is to build the model allowing to describe and predict performance of the web-based 3-tier e-commerce application following the methodology presented in the Section 2. The outputs (response variables) considered are application response time and CPU utilisation.

## 3.2 Test Environment

### *3.2.1 Testbed Description*

The testbed consists of workload generator syntactically generating requests to a backend web-based application. Experiments in this study were performed using model-driven workload generator called MDLoad [5]. MDload automatically generates requests to an application under test by simulating a set of users. Since the workload generator needs to create considerable number of virtual users, MDLoad was deployed on a Virtual Machine (VM) with 12 CPU and 3GB of memory. This VM is located on the private cloud at Imperial College London. The hosts of the private cloud are Intel Xeon with CPU E5-2450 2.10 GHz. The capacity of the VM machine was chosen based on the previous experience with MDload such a way that relatively small number of users would saturate the application, resulting in significant increase in application response time and CPU utilization, but not leading to the MDload outage. This decision allowed to reduce execution time needed for each experimental run while still obtaining sufficient samples to estimate mean values of performance metrics.

The software stack of workload generator comprises JAVA and shell scripts for submitting HTTP requests and controlling the behaviour of virtual users by creating session-based workload. The request composition of the sessions for the three MDload user classes adopted in this study is shown in Table 1:

**Table 1: Request mix per session for 3 MDload user classes.**

| Request | Class I (light) | Class II (medium) | Class III* (heavy) |
|---|---|---|---|
| Home | + | + | + |
| Login | + | + | + |
| Login details | + | + | + |
| Main | | + | + |
| Order History | | + | + |
| QuickAddMain | | + | + |
| CartAddAll | | + | + |
| Checkout | | + | + |
| CheckoutAddressNext | | + | + |
| CheckoutPaymentNext | | + | + |
| CheckoutShippingNext | | + | + |
| Logout | + | + | + |

*Class III has higher number of Checkout requests per session than Class II

### *3.2.2 Application Under Test*

The application under test is Apache OFBiz [6] - an open source web-based e-commerce system. The OFBiz instance is

deployed on a VM with 1 CPU and 3GB of memory on the same private cloud at Imperial College London. Keeping both workload generator and backend application on the hosts in the same private cloud and connected through high-speed broadband network allows to remove 'noise' in the system response time (collected on the MDload side using tool's features) caused by network latencies. Therefore, measurements for the system response time can be considered response time on the application level.

## 3.3 Screening Procedure

There are a number of parameters (in DoE known as 'factors') that might influence application performance. These may be external inputs, such as, for example, number of users, user think time, or system parameters (e.g. hardware configuration on which application is deployed). Such parameters may be controllable (can be changed by the experimenter) and uncontrollable. For example, network delay, mentioned above, can be viewed as the noise factor, influencing response time as it is experienced by the user. An extensive taxonomy of factors is given in [4]. However, to explore all possible combinations of these factors would require $2^k$ experimental runs, as was mentioned in the Section 2. In the example from [4] that would be $2^{38}=274\text{x}10^9$ runs, which is, of course, infeasible. Therefore, not only fractional factorial design is needed, but also careful consideration for the choice of the candidate factors for the screening procedure, based on the experience of the experimenter.

In this study it was decided to start with a small set of well-known factors, such as number of users, user think time and workload mix. Additionally it was tested if the testbed set up described in 3.2.1 would allow to decrease execution time of the experimental run without causing deterioration of estimates. The low and high levels for the number of users were chosen based on the N*, where N*, following the definition from [3] is the point where application starts exhibiting saturation behaviour. Levels for other factors for the two-level design were chosen based on the authors' experience with application load testing and MDload. The summary of factors and their levels chosen for the screening procedure is given in the Table 2.

**Table 2: Factors and their levels.**

| | Levels | |
|---|---|---|
| | **Low (-1)** | **High (1)** |
| **Number of users*** | 3** | 20 |
| **User think time, s** | 10 | 1 |
| **Execution time, min (steady state)** | 10 | 30 |
| **Workload mix (user class)** | I | III |

* N*=16 for user think time = 5 s.

** N_users = 3 instead of 1 is chosen to obtain more samples for averaging.

To investigate all possible combinations of these four factors would require $2^4=16$ runs, which is theoretically feasible, but with half of the runs requiring execution for 30 minutes it would take 5 h 20 min. Therefore it was decided to use fractional factorial design in line with the commonly-used procedure. It is important to note, though, that the price for the reduction in runs is so-called confounding of effects. This means that the effects (factors and their interactions) estimated based on the results of fractional factorial design are a combination of two or more effects. Hence it is important to choose fractional factorial design in such a way so that main effects are confounded with higher-order interactions. The higher-order interactions (interactions of N-1 factors in design for N factors) are generally considered negligible. The fractional factorial design of resolution IV (all main effects will be confounded with higher-order interactions, low order interactions will be confounded with each other) for the example data from Table 2 along with the confounding pattern is presented in Table 3:

**Table 3: Fractional factorial design for four factors.**

| Exp. run | Number of users (A) | Think time (B), s | Execution time (C), min | User class (D) | Confounding pattern |
|---|---|---|---|---|---|
| 1 | 3 | 10 | 10 | I | I=I+ABCD |
| 2 | 3 | 10 | 30 | III | A = A + BCD |
| 3 | 3 | 1 | 10 | III | B = B + ACD |
| 4 | 3 | 1 | 30 | I | C = C + ABD |
| 5 | 20 | 10 | 10 | III | D = D + ABC |
| 6 | 20 | 10 | 30 | I | AB = AB + CD |
| 7 | 20 | 1 | 10 | I | AC = AC + BD |
| 8 | 20 | 1 | 30 | III | AD = AD + BC |

As was mentioned above, higher-order interactions are considered negligible. Therefore, based on the results of the experimental runs it should be possible to make conclusion about significance of main effects (significance of interactions should be treated carefully as they are confounded with each other).

Response variables response time and CPU utilisation, obtained in the screening experiments can be analysed graphically (numerical analysis such as ANOVA or p-values is not recommended because of confounding). Example analysis for the response time is shown in the Figures 1 and 2.

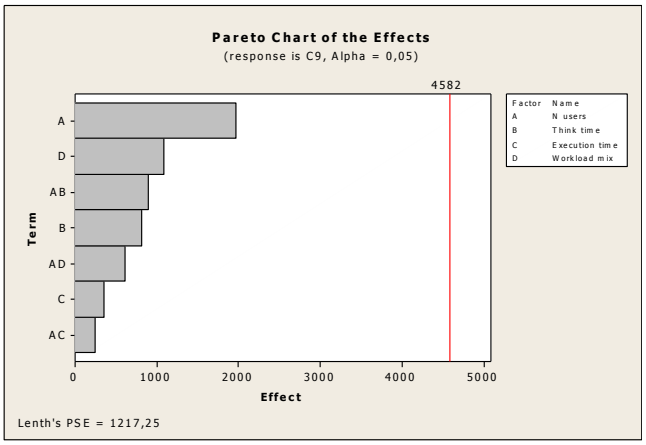



**Figure 1. Ranking of effects.**

On the Figure 1 estimated effects are ranked by their magnitude. Red line represents Lenth's PSE – pseudo-standard error. All effects that cross this line are deemed significant. From the Figure 1 it is obvious that none of the factors are deemed significant, which is suspicious, because from the Figure 2 it is seen that at least number of users, think time and workload mix make an impact on the response time. To investigate this problem 8 more runs were conducted to create a full factorial design for 4 factors. The results (for the response time) of the full factorial design for 4 factors are shown on Figure 3.

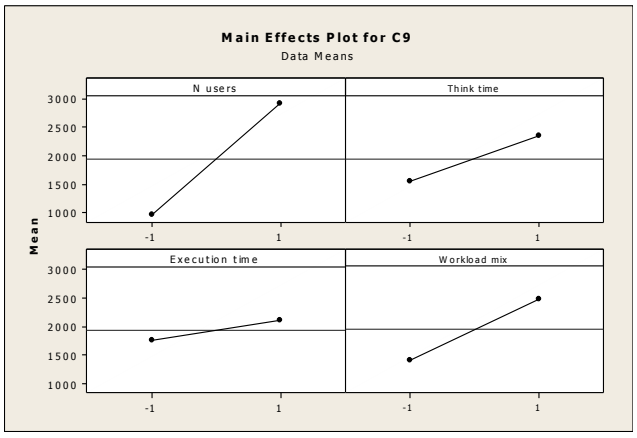



**Figure 2. Main effects plot.**

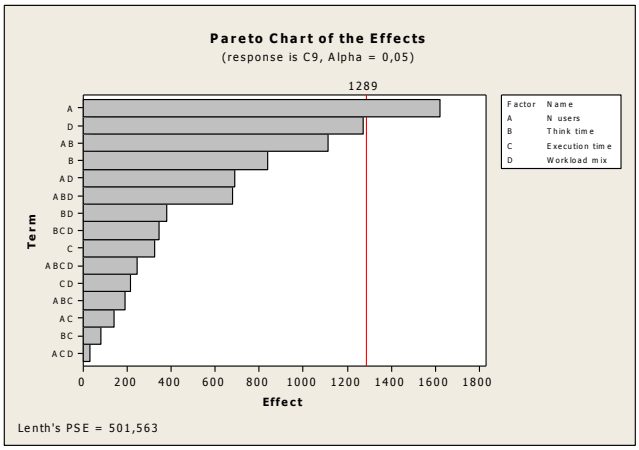



**Figure 3. Ranking of effects.**

It is clearly seen from the Figure 3 that number of users is significant, user class is close to significance, as well as the interaction between number of users and think time. Additionally, it can be seen that high-order interactions ABD and BCD (and even ABC) are not negligible as had been assumed. This resulted in the distortion of main effects and the value of Lenth's PSE, which is based on the effects' magnitudes. In the case of 4 factors, where at least two of them turned out to be significant (number of users and workload mix) as well as the two-way interaction for the third factor(think time), the combined influence ABD of these three factors turned out to be large. Such occurrence can be mitigated by screening for large number of factors, especially with deliberate addition of factors which should not be significant, because then there is a small chance that combined interaction of, e.g., 5 factors for 6-factor design would be present.

After screening test is conducted, and significant main effects are found, the full factorial design with replications should be conducted for this subset. If there are significant interactions (or close to significance), the factors that cause them also should be included into full factorial design, even if they themselves were not identified as significant. In the example think time (B) would be taken into the subset of significant factors, even though it is on itself wasn't flagged as significant, because the interaction AB (between number of users and think time) is very large. Execution time did not show any significant influence either on response time or CPU utilisation, therefore it was set at the low level (10 minutes). The full factorial design with 3 replications and response variables are presented in the Table 4. This design is needed to validate analysis conducted on the fractional factorial design stage and required 4 additional runs (2, 3, 5 and 8).

**Table 4: Full factorial design for 3 factors.**

| Exp. run | N_users | Think time, s | User class (D) | Execution time, min |
|---|---|---|---|---|
| 1 | 3 | 10 | I | 10 |
| 2 | 3 | 10 | III | 10 |
| 3 | 3 | 1 | I | 10 |
| 4 | 3 | 1 | III | 10 |
| 5 | 20 | 10 | I | 10 |
| 6 | 20 | 10 | III | 10 |
| 7 | 20 | 1 | I | 10 |
| 8 | 20 | 1 | III | 10 |

The analysis of results confirmed that all three factors, as most of their low-order interactions were significant. Additional analysis was conducted to estimate the allocation of variation: how much variation each of the factors causes in the response variable when changed from low to high level [7]. Variation of responses (in %) due to factors and their interactions is shown in Table 5:

**Table 5: Variation of responses (in %) due to factors and their interactions.**

| Effect | Response time | CPU utilisation |
|---|---|---|
| **N_users** | 26.03 | 54.27 |
| **Think time** | 4.53 | 42.99 |
| **User class** | 36.25 | 1.14 |
| **N_users:Think time** | 19.13 | 0.59 |
| **N users:User class** | 6.63 | $7.886*10^{-6}$ |
| **Think time:User class** | $1.5x10^{-8}$ | $1.8917*10^{-4}$ |
| **N_users:Think time: User class** | 5.42 | 0.91 |
| **Error** | 2.01 | $7.6946*10^{-4}$ |

Error term in the Table 5 contains both random error and influence of any factors that were not considered when constructing screening design. As this error term is very small for both response variables, it is safe to assume that all major sources of variation were identified.

## 3.4 Constructing the Model

As both response time and CPU utilization exhibit non-linear behaviour, Response Surface (RS) design, namely central-composite Box-Wilson design, was chosen. This design contains full factorial design for 3 factors and centre points, therefore can be used to construct both linear, quadratic and polynomial models. Additionally, the 'faced' configuration of the design was implemented. This configuration does not use points outside of the design space. The prediction capabilities of the model constructed based on this design can be worse than of a combination using the points outside the design space, but in our case this combination is impossible to implement (we can't go beyond user classes I and III). This design requires 24 runs in total (centre points are run 10 times to allow for a more uniform estimate of the prediction variance over the design space). The design is shown in the Table 6 (shaded area shows full factorial design):

**Table 6: Box-Wilson central composite 'faced' design.**

| N | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $X_1$ | -1 | -1 | -1 | -1 | 1 | 1 | 1 | 1 | -1 | 1 | 0 | 0 |
| $X_2$ | -1 | -1 | 1 | 1 | -1 | -1 | 1 | 1 | 0 | 0 | -1 | 1 |
| $X_3$ | -1 | 1 | -1 | 1 | -1 | 1 | -1 | 1 | 0 | 0 | 0 | 0 |
| N | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| $X_1$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $X_2$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $X_3$ | -1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

As was mentioned above, chosen RS design allows to construct various types of regression models. We want to investigate how they fare in prediction. Summary of the constructed regression models is given in the Table 7:

**Table 7: Regression models constructed from the experiment results and used in subsequent analysis.**

| Name | Description | Formula |
|---|---|---|
| **Linear** | Model contains an intercept and linear terms for each factor | $y=I+a_1x_1+a_2x_2+ +a_3x_3$ |
| **Interactions** | Model contains an intercept, linear terms, and all products of pairs of distinct factors | $y=I+a_1x_1+a_2x_2+ +a_3x_3+a_4x_1{:}x_2+ +a_5x_1{:}x_3+a_6x_2{:}x_3$ |
| **Pure Quadratic** | Model contains an intercept, linear terms, and squared terms | $y=I+a_1x_1+a_2x_2+ +a_3x_3+a_4x_1^2+ +a_5x_2^2+a_6x_3^2$ |
| **Quadratic** | Model contains an intercept, linear terms, interactions, and squared terms | $y=I+a_1x_1+a_2x_2+ +a_3x_3+a_4x_1{:}x_2+ +a_5x_1{:}x_3+a_6x_2{:}x_3+ +a_7x_1^2+a_8x_2^2+a_9x_3^2$ |
| **Full Polynomial** | Model is a polynomial with all terms up to degree 3 in the first factor, degree 3 in the second factor, and degree 3 in the third factor* | $y=I+a_1x_1+a_2x_2+ +a_3x_3+a_4x_1{:}x_2+ +a_5x_1{:}x_3+a_6x_2{:}x_3+ +a_7x_1{:}x_2{:}x_3+a_8x_1^2++a_9x_2^2+a_{10}x_3^2+ +a_{11}x_1^2{:}x_2+ +a_{12}x_1{:}x_2^2+ +a_{13}x_1^2{:}x_3$ |

*$x^3$ terms are zero, the third level was chosen to include 3-way interaction between number of users, think time and user class into the model.

Prediction curves R=*f*(N users) and U_cpu=*f*(N users) were constructed for each model type for every combination of user class and user think time. As an example, the curves for 'full polynomial' model type and user class III are shown in the Figures 4 and 5.

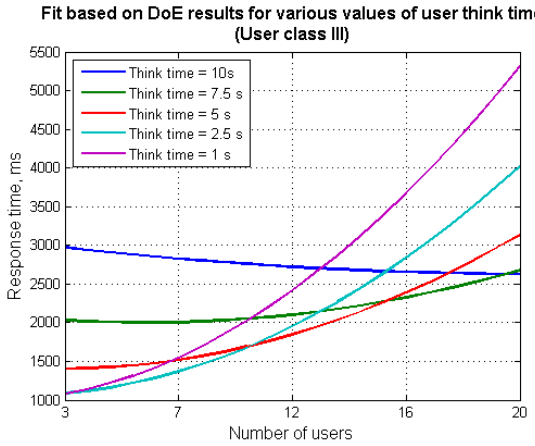



**Figure 4. Prediction for the response time.**

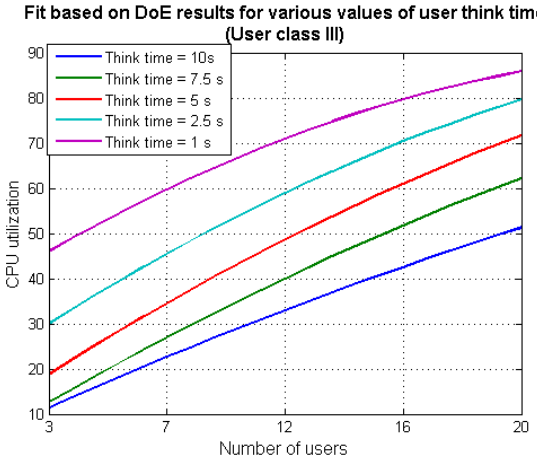



**Figure 5. Prediction for CPU utilization.**

# 4. ANALYSIS AND DISCUSSION

## 4.1 Collect Independent Observations.

In order to assess the model prediction capabilities, a series of experiments with parameter values from the design space was run. One experiment point was run per each prediction, i.e. pair {User think time, user class}, for N users = 16. The points collected for the model verification are given in the Table 8:

**Table 8: Independent observations.**

| Think time, s | User class I | | User class II | | User class III | |
|---|---|---|---|---|---|---|
| | RT, s | Ucpu, % | RT, s | Ucpu, % | RT, s | Ucpu, % |
| **10** | 0.79 | 31.8 | 2.72 | 33.1 | 2.7 | 39.8 |
| **7.5** | 0.83 | 41.0 | 3.14 | 39.9 | 2.02 | 37.0 |
| **5** | 0.88 | 50.7 | 2.72 | 53.5 | 2.00 | 67.0 |
| **2.5** | 1.29 | 76.1 | 2.03 | 77.8 | 2.83 | 64.8 |
| **1** | 1.43 | 80.1 | 2.91 | 80.99 | 3.71 | 92.0 |

## 4.2 Prediction Accuracy.

Prediction error for each {User think time, user class} pair is defined as a relative standard error

$$\delta = \left|\frac{(Y-\bar{Y})}{Y}\right| * 100\%$$

where Y is an observation and $\bar{Y}$ is predicted value. Accuracy of prediction for the entire model is estimated as a standard deviation of the sum of squares of prediction errors

$$\sigma = \sqrt{\frac{\sum \delta^2}{N-P}}$$

where N = 15 (3 user classes, 5 think time values (1, 2.5, 5, 7.5 and 10 s)) and P = 4 (intercept and 3 independent variables).

Additionally, these observations were compared to the prediction based on the out-of-the-box Mean Value Analysis (MVA) algorithm for queueing network models, implemented in the Java Modelling Tool [2].

Total prediction error and estimation bias (average of all differences between observed and predicted values, not their absolute values) for response time and CPU utilisation for each

model type, full factorial design (FF) for 3 factors and MVA prediction are summarised in the Table 9:

**Table 9: Total prediction error and bias for various model types.**

| | | Total prediction error σ, % | | Bias, % | |
|---|---|---|---|---|---|
| | | RT | CPU | RT | CPU |
| Response Surface models | Linear | 6.51 | 4.3 | -3.62 | -0.75 |
| | Interactions | 6.32 | 4.09 | -2.6 | -0.65 |
| | Pure quadratic | 5.11 | 4.93 | -2.02 | -0.79 |
| | Quadratic | 5.42 | 4.09 | -1.0 | -0.69 |
| | Full polynomial | 5.12 | 4.06 | -1.97 | -0.96 |
| FF | | 6.896 | 3.987 | -4.96 | -0.32 |
| MVA | | 40.0 | 11.4 | -234.6 | 7.29 |

From the Table 9 it may be seen that prediction error σ for the response time is a bit higher in the case of linear models ('linear', 'interactions' and full factorial design), which is to be expected since relationship between number of users and response time is not linear. As for the CPU utilization, all DoE models showed error 4-5%. This may be explained by the fact that within most of the design space CPU utilization increases linearly with increase in the number of users. However, prediction by MVA algorithm produced the error of 40% for the response time.

In order to investigate this phenomenon, we looked into independent observations and predicted values obtained from both DoE models and MVA algorithm. From the Table 8 and Figure 4 it may be seen that for the response time both observed and predicted response times do not follow classical trend of monotonous increase with decrease in user think time [3]. The comparison between independent observations, DoE RS 'full polynomial' model and MVA algorithm predictions, along with prediction errors (on the example for the user class III) are presented in the Table 10. Comparison of results in Table 10 revealed that both independent observations and values, predicted by RS model, follow the same trend. It indicates that there is some persistent (i.e. constantly present) behaviour, which RS model, having no knowledge of the system under test, however, is able to capture based only on the application inputs and outputs. MVA algorithm also captures this trend, however, it drastically overestimates response time values.

**Table 10: Trend for response time (s) in predicted values and observations.**

| | User think time, s | | | | |
|---|---|---|---|---|---|
| | 10 | 7.5 | 5 | 2.5 | 1 |
| Observed | 2.7 | 2.02 | 1.9 | 2.8 | 3.7 |
| RS model | 2.66 | 2.33 | 2.39 | 2.84 | 3.67 |
| Error,% | **1.1** | **-15.3** | **-25.8** | **-1.4** | **0.8** |
| MVA | 33 | 24.9 | 25.5 | 43.8 | 58.4 |
| Error, % | **-1122** | **-1124** | **-1216** | **-1464** | **-1478** |

All RS models demonstrate negative bias, which means that overall prediction tends to overestimate both response time and CPU utilisation, except MVA algorithm, which underestimates CPU utilisation.

# 5. CONCLUSIONS AND FUTURE WORK

This study highlighted the importance of software performance modelling and prediction, identified existing gap in the knowledge and proposed a new performance modelling approach, based on the Design of Experiments technique.

The results demonstrate that proposed method produces good prediction of an application performance while treating it as a black box, even in the presence of an anomalous behaviour. Additionally, it showed much better prediction capabilities compared to the out-of-the box Queuing Network model. This allows to suggest that proposed method may be used for the performance modelling and prediction on the application development stage, where models based on measurements may be a better alternative as they provide a good trade-off between efforts required for model specification and accuracy of estimation and prediction.

Considering the study outcomes, some of the directions for further work may be investigation of the approach predictive capabilities in multiclass models and as a tool for anomaly detection.

# 6. ACKNOWLEDGMENTS

This work was supported by the funding from the European Union's Horizon 2020 research and innovation programme [grant agreement No. 644869]. Authors also would like to thank Dr. G. Casale and Dr. J.F. Perez-Bernal from Imperial College London for their support and invaluable comments.

# 7. REFERENCES

[1] NIST/SEMATECH e-Handbook of Statistical Methods http://www.itl.nist.gov/div898/handbook/

[2] Java Modelling Tools. http://jmt.sourceforge.net/

[3] Lazowska E et. al. 'Quantitative system performance'. Available online from: http://homes.cs.washington.edu/~lazowska/qsp/

[4] Z. Li, L. O'Brien, H. Zhang, and R. Cai. A factor framework for experimental design for performance evaluation of commercial cloud services. In Cloud Computing Technology and Science, 2012 IEEE 4th International Conference on, pages 169, 176.

[5] MDload load generation simulator. https://github.com/imperial-modaclouds?query=modaclouds-mdload

[6] OFBiz web-based 3 tier e-commerce application. http://ofbiz.apache.org/

[7] J. Rai. Art of Computer Systems Performance Analysis Techniques For Experimental Design Measurements Simulation And Modeling. Wiley Computer Publishing, John Wiley & Sons, Inc. ISBN: 0471503363 Pub Date: 05/01/91

[8] Software Engineering Institute - Blog https://blog.sei.cmu.edu/post.cfm/continuous-integration-in-devops

[9] Spinner S., Casale G., Zhu X., and Kounev S. LibReDE: A Library for Resource Demand Estimation (Demonstration Paper). In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*, Dublin, Ireland, March 22-26, 2014. ACM. March 2014

[10] D. Westermann, R. Krebs, and J. Happe. Efficient experiment selection in automated software performance evaluations. In Computer Performance Engineering, pages 325-339. Springer, 2011.

# A Proactive Approach for Runtime Self-Adaptation Based on Queueing Network Fluid Analysis

Emilio Incerto
Gran Sasso Science Institute
Viale Francesco Crispi, 7
L'Aquila, Italy
emilio.incerto@gssi.infn.it

Mirco Tribastone
IMT Institute for Advanced Studies Lucca
Piazza S. Francesco, 19
Lucca, Italy
mirco.tribastone@imtlucca.it

Catia Trubiani
Gran Sasso Science Institute
Viale Francesco Crispi, 7
L'Aquila, Italy
catia.trubiani@gssi.infn.it

## ABSTRACT

Complex software systems are required to adapt dynamically to changing workloads and scenarios, while guaranteeing a set of performance objectives. This is not a trivial task, since run-time variability makes the process of devising the needed resources challenging for software designers. In this context, self-adaptation is a promising technique that work towards the specification of the most suitable system configuration, such that the system behavior is preserved while meeting performance requirements.

In this paper we propose a proactive approach based on queuing networks that allows self-adaptation by predicting performance flaws and devising the most suitable system resources allocation. The queueing network model represents the system behavior and embeds the input parameters (e.g., workload) observed at run-time. We rely on fluid approximation to speed up the analysis of transient dynamics for performance indices. To support our approach we developed a tool that automatically generates simulation and fluid analysis code from an high-level description of the queueing network. An illustrative example is provided to demonstrate the effectiveness of our approach.

## Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*performance measures*; C.4 [**Performance of Systems**]: Modeling techniques, Performance Attributes

## Keywords

Runtime Self-Adaptation, Queueing Networks, Fluid Approximation Analysis

## 1. INTRODUCTION

In the software development process it is fundamental to understand if performance requirements are fulfilled, since they represent what end users expect from the software system. In fact, if performance targets are not met, a variety of negative consequences (such as damaged customer relations, business failures etc.) can impact the project success.

The performance evaluation of software systems is critical in most application domains, such as distributed web systems and services, enterprise applications or cloud computing, since such domains are increasingly required to adapt dynamically to changing workloads, scenarios and objectives. Guaranteeing performance requirements with a high degree of unpredictability and dynamism in the execution context is a not trivial task, since the presence of runtime variability (e.g., workload peaks) makes the whole process challenging for software designers.

Several approaches have been proposed in literature for performance evaluation [14], however the current status of the proposed approaches is far from an ideal situation. Current methodologies rely on: (i) model-based predictions that may not approximate real systems; (ii) measured runtime values that are very expensive in terms of resource usage. Our work tries to overcome these issues by proposing a proactive approach based on queuing networks (QN) [16] that allows model-based predictions while considering runtime variability by means of fluid approximation analysis. In this way we are able to efficiently predict performance flaws and devise the most suitable system resources allocation.

We propose an approach to guarantee the performance requirements of software systems by explicitly considering variability in the performance analysis process. The performance predictions are leveraged to pro-actively place the software in the optimal configuration with respect to changing conditions and parameter runtime changes. In particular, the QN model represents the system behavior and embeds the input parameters (e.g., workload) observed at runtime. We rely on fluid approximation to speed up the analysis of transient dynamics for performance indices. The goal is to build a framework able to anticipate performance flaws by adapting software on the basis of the monitored runtime variabilities.

In this paper self-adaptation is currently implemented with a repository of design changes that impact the system performance, such as the reallocation of software components to hardware platforms, and mirroring the software components to better distribute incoming requests. To support our approach we developed a tool that automatically generates simulation and fluid analysis code from a high-level description of the queueing network.



*QUDOS'15*, September 1, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3817-2/15/09...$15.00
http://dx.doi.org/10.1145/2804371.2804375

The remainder of this paper is organized as follows. Section 2 presents related work; Section 3 provides foundations for the QN fluid approximation analysis; Section 4 describes our approach; Section 5 shows the approach at work on an illustrative example to demonstrate its effectiveness; Section 6 discusses the relevant open issues raised by the approach and provides directions for future research.

## 2. RELATED WORK

The work presented in this paper relates to two main research areas: (i) runtime self-adaptation, and (ii) queueing network fluid analysis.

*Runtime Self-Adaptation.* The need of runtime self-adaptation has been widely recognized in literature. In [11] the problem to dynamically self-adapt software systems (due to resource variability, changing user needs, system faults, etc.) is addressed with a framework using software architectural models to monitor and adapt a running system. Our approach differs from [11], since we are interested in performance properties of running systems and our proactive approach makes use of QN models to avoid performance flaws. In [28] authors claim that one of the major challenges in self-adaptive systems is to assure the required Quality-of-Service (QoS) properties (e.g., performance, reliability), and a relevant part of the studies use formal methods at runtime. However the usage of such methods is limited to modeling and analysis, whereas our approach includes a set of adaptation strategies aimed at executing changes in the running system. In [1] approaches for performance engineering of self-adaptive systems are surveyed and classified (design vs runtime [19], and reactive vs proactive [18]), outlining that model-driven tools including modeling of self-adaptive systems with explicit adaptation strategies are missing in the current state-of-the-art. Several approaches have been defined in the context of QoS driven runtime adaptation, in particular for service-oriented systems [4, 17], for the evolution of runtime parameters [9], for the quantitative verification [3]. The novelty of our approach is that it exploits the QN fluid approximation to speed up the analysis of transient dynamics for performance indices.

*Queueing Network Fluid Analysis.* We consider fluid analysis in the sense of Kurtz [15], providing a system of ordinary differential equations that associates one equation to each service center in the network. The solution to this equation gives an estimate to the average queue length as a function of time. We refer to [7, 26, 2, 23] for a more in-depth discussion of the derivation of the fluid QN models considered here. The refactoring approach presented in this paper in principle requires the evaluation of each possible QN design alternative from scratch, i.e., without the possibility of reusing the analysis across different alternatives. This is a problem that has been recently tackled in previous literature, for instance in [24] and [13]. However, the approach in [24] allows to reduce the complexity of the exploration of the design space when specific conditions on the constraints for adaptation are met. The symbolic technique of [13] computes the solution of every possible design alternative at once, but it works for Jackson-type queuing networks (see e.g., [22]). Furthermore, it supports only steady-state performance measures, which prevents to apply adaptation when transient violations are detected (as shown in our running example).

Lastly, Table 1 reports some of the QN analysis tools that can be found in literature. We can notice that such tools make use of simulation (providing transient behavior analysis) and/or analytic (providing exact or approximate steady state solutions) evaluation techniques. The novelty of our approach is that it relies on QN fluid approximation analysis that can be seen as an analytical technique for studying the transient behavior of the QN model.

**Table 1: Other QN Tools**

| Tool | Evaluation technique |
|---|---|
| JMT [6] | Analytic and Discrete Event Simulation |
| SHARPE [25] | Analytic |
| JINQS [10] | Discrete Event Simulation |
| PEPSY-QNS [12] | Discrete Event Simulation and Analytic |
| TANGRAM-II [5] | Simulation and Analytic |
| QSIM [21] | Discrete Event Simulation |

## 3. BACKGROUND

The purpose of this section is to briefly describe the main concepts underlying the fluid approximation analysis of queuing networks. Roughly speaking, the goal of this analysis technique is to translate a QN model in a system of ordinary differential equations (ODEs), then solving them to derive the performance indexes of interest. The role of the defined equations is to analytically describe the evolution of the queue length at each service center composing the QN model. The main idea is to view a QN model $M$ as a *Markov Population Process* (MPP) where:

i) $x_i(t)$ is the number of jobs in the queue of service center $i$ at time $t$, and $N(t) = \sum_{i \in M} x_i(t)$ is the total population of jobs circulating in the network of $M$ centers;

ii) each service center $i$ modifies the number of jobs waiting in the queue, according to a *jump vector* $I_i \in \mathbb{R}^{|M|}$;

iii) the rate of the changes caused by each server center $i$ is proportional to its actual jobs population $x_i(t)$, and is expressed by the definition of the infinitesimal generator $q_{x,x+I}$.

Let us now consider, as an example, a tandem QN model with $N$ circulating jobs. Assume that this network consists of a pure delay station (station 1) and a single server station (station 2) with services rates $\mu_1$ and $\mu_2$ respectively. From the topology of the QN model we can derive the jump vectors $I_1 = (-1, +1)$ and $I_2 = (+1, -1)$ where each component of the jump vector $I_i$, with $i \in \{1, 2\}$, describes the change of number of jobs at each service center. The elements of the infinitesimal generator for this model are $q_{x_1,x_1+I_1} = \mu_1 x_1$ and $q_{x_2,x_2+I_2} = \mu_2 \min(1, x_2)$. Putting it all together we can write the ODE system as:

$$\frac{dx_1(t)}{dt} = -\mu_1 x_1(t) + \mu_2 \min(1, x_2(t))$$
$$\frac{dx_2(t)}{dt} = +\mu_1 x_1(t) - \mu_2 \min(1, x_2(t))$$

By solving this system, we can compute the queue length of the two stations over time. The transformation of the QN

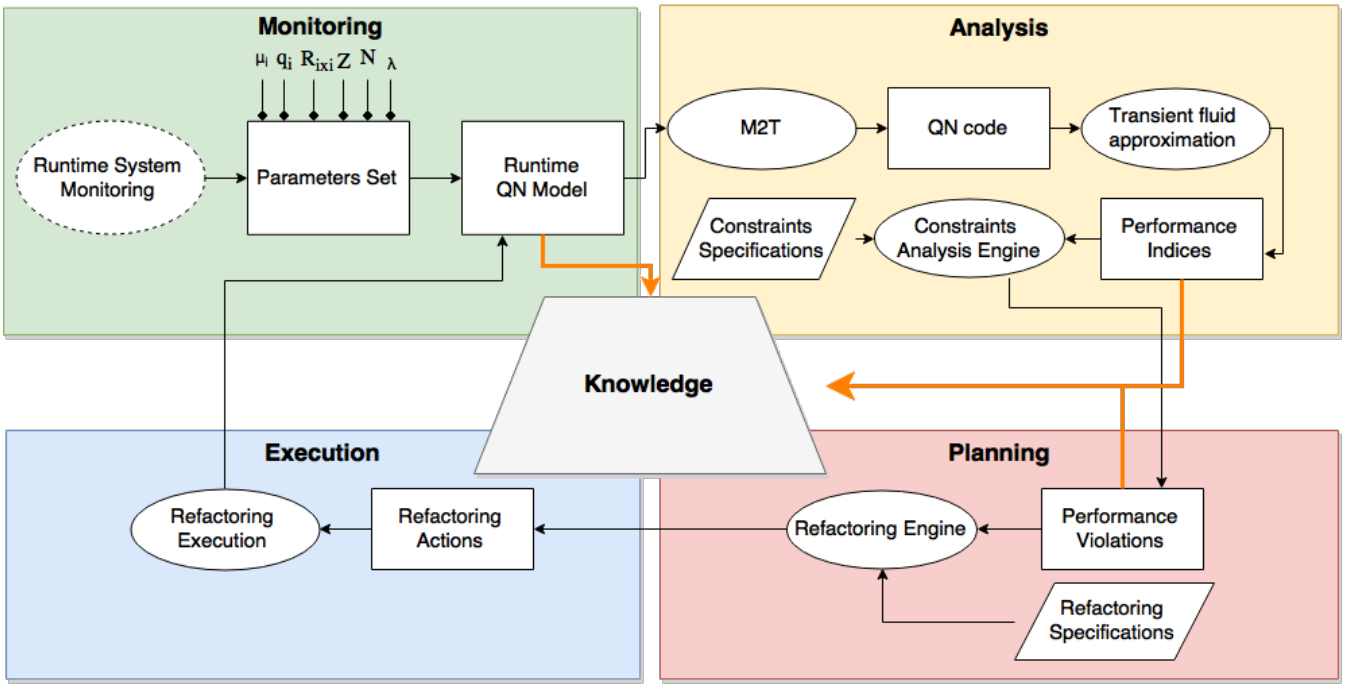



**Figure 1: Overview of the proposed technique**

model in ODE implies that the solution process shifts from a discrete-state characterization and mean value analysis to a continuous-state representation (based on a system of ordinary differential equations) and fluid approximation analysis. Its computational complexity depends directly on the algorithm used for solving the ODE system that, by referring to the original QN model, is related to the product between the number of stations in the network and the transitions among them.

## 4. OUR APPROACH

The purpose of this section is to describe a framework that provides the self-adaptation capability to software systems. The goal of this adaptation process is to keep the performance indices of the system within the stated requirements.

The key idea of our approach is to exploit the predictive power of queuing network models by analyzing them at runtime. In particular, feeding a model of the running system with runtime parameters allows us to see if its future evolution will results or not in the degradation of the performance indices of interest.

Our work, can be considered as an implementation of the MAPE-K [8] loop that is a very well assessed methodology in the context of *self-adaptive systems*. The overview of the proposed approach is depicted in Figure 1 in which we can identify the following phases:

- **Monitoring:** the idea of this phase is to monitor the running system using some kind of runtime system profiler tool, similarly to [27]. Ideally, beyond the choice of a particular tool, we expect to periodically extract a set of parameters from the running system. More precisely we want to collect $\mu_i$ as the service rate for each service center, $q_i$ as the estimated queue length at service center $i$ and $R_{ixi}$ as the routing matrix of the QN model. In case of closed networks, we estimate the number of customers in the system $N$ and the thinking time $Z$. On the contrary, in case of open networks, we identify the jobs arrival rate $\lambda$. Then we use these parameters to feed the *runtime QN model* built in order to describe the running system.

- **Analysis:** in this step using an ad-hoc defined model-to-text transformation (M2T) we translate the high-level queuing network model of the previous phase, in a low-level representation (*QN code*) suitable for automatic analysis. Executing this code we can analyze the transient behavior of the QN exploiting the fluid approximation technique. Result of this step is the set of *performance indices* of interest that are influenced by the previously estimated parameters. Now giving the computed indices and the performance constraints model as input to the *constraint analysis engine* we detect the eventual performance violations.

- **Planning:** starting from the *performance violations* previously computed and from a refactoring specification model, the *refactoring engine* component is able to compute a set of refactoring actions that represent the plan for the current adaptation iteration. We can think at this actions as the adding or removal of a service center, the modification of the service rates or for example the change of the routing probability for some branch in the model.

- **Execution:** executing the previously defined *refactoring actions* we are able to devise the new QN model that is aimed at avoiding violations of performance constraints. Several refactoring actions may be available and each action gives rise to a new runtime QN

model that undergoes the same process of the initial one. Our approach aims to evaluate several QN models and then reflect in the running system the change actually beneficial only.

We identify as **knowledge** of the MAPE-K loop the information that is required to describe the system in each self-adaptation iteration $t$. Such information is identified by the runtime QN model, the computed performance indices and the detected performance violations.

Note that the presented approach is tool supported. We developed an Eclipse-based tool used for the definition of the Queuing network models and to automatically execute the M2T transformation on them. Our tool can be downloaded here `http://sourceforge.net/projects/qnml`.

# 5. ILLUSTRATIVE EXAMPLE

In this section we show a simple but meaningful illustrative example suitable to demonstrate our approach. In particular we consider a constraint model requiring that the percentage of jobs in a queue of every service center do not exceed 0.5% of the total jobs population. In the following the example is described step by step.

## 5.1 Monitoring

Figure 2 depicts the QN model describing the running system used in our illustrative example. It represents a closed network composed by: one delay station ($D1$), four service centers (*Server1, Server2, Server3, Server4*) and four routing stations ($R1, R2, R3, R4$). In Figure 2 the solid lines are used to represent the connections between the routing station and service center; on the contrary, the dashed line is used to represent the opposite direction.

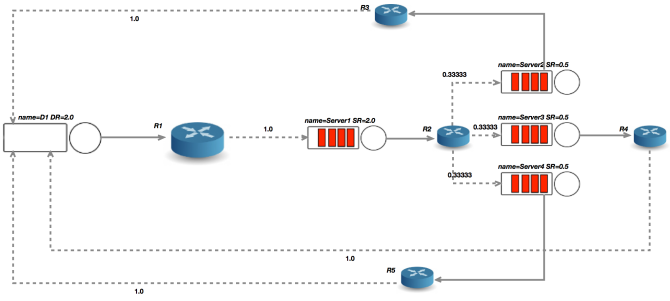



**Figure 2: Initial Model**

Although, in our vision, the parameters embedded by the model should to be automatically and periodically extracted from the running system, for sake of simplicity, we fed the model for the example with specific parameters suitable to trigger interesting behavior into the system. More precisely the used parameters are listed in Table 2 while the routing matrix is reported hereafter.

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 1/3 & 1/3 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

**Table 2: Initial Model Parameters**

| Station | Type | Init. Pop. | $\mu_i$ | Z |
|---|---|---|---|---|
| D1 | Delay Center | 10 | n.d. | 0.5 |
| Server1 | Service Center | 0 | 2.0 | n.d. |
| Server2 | Service Center | 0 | 0.5 | n.d. |
| Server3 | Service Center | 0 | 0.5 | n.d. |
| Server4 | Service Center | 0 | 0.5 | n.d. |

## 5.2 Analysis

Goal of this step is to produce the performance indices and to detect the eventual constraints violation of the system. Figure 3 shows the structure of the Eclipse plug-in used in order to implement the Model to Text Transformation (M2T) through the ACCELEO framework [20].

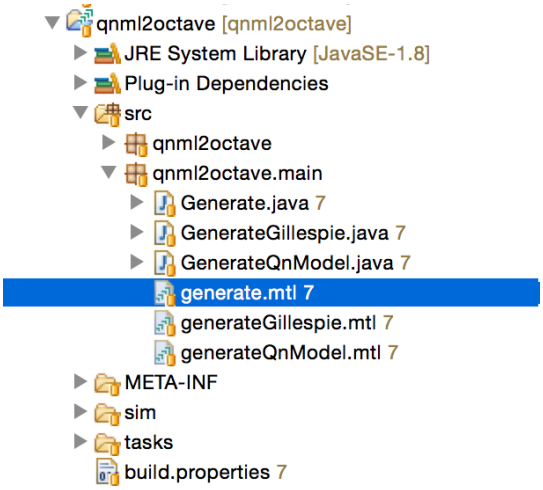



**Figure 3: M2T plug-in structure**

Using this transformation we can translate the previously defined QN model into Octave[1] executable code, shown in Listing 1. It is worth to notice that this low-level representation of the model allows us to analyze the transient behavior of the system through the fluid approximation technique.

**Listing 1: Octave Code For The Initial Model**

```
function dx = qn_ode(x,t)
dx = zeros(5,1);
dx(1) = -mu1*x(1)+mu3*min(x(3),1)
   +mu4*min(x(4),1)+mu5*min(x(5),1);
dx(2) = +mu1*x(1)-mu2_1*min(x(2),1)
    -mu2_2*min(x(2),1)-mu2_3*min(x(2),1);
dx(3) = +mu2_1*min(x(2),1)-mu3*min(x(3),1);
dx(4) = +mu2_2*min(x(2),1)-mu4*min(x(4),1);
dx(5) = +mu2_3*min(x(2),1)-mu5*min(x(5),1);
endfunction
```

Figure 4 shows how the queue length at each service center changes during the observation time. We can see that the population of the jobs in the network, at the beginning of the analysis, is almost in the queue of *server*1. This happens due to the fact that the thinking time $Z$ of $D1$ is lower than the time needed by *Server*1 in order to complete a job.

[1] `http://www.gnu.org/software/octave/`

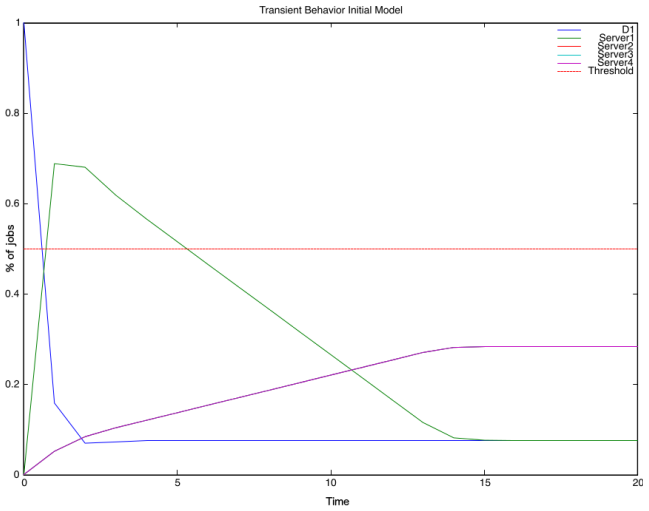



**Figure 4: Transient behavior for the initial model**

Considering our constraints model, the behavior shown in Figure 4 is not acceptable and represents a performance violation. The set of perfomance violations are used as input for the planning phase in order to produce the appropriate set of refactoring actions.

## 5.3 Planning

The refactoring engine relying on the refactoring specification is able to generate a set of refactoring actions suitable to react to the identified performance violation. In our example the refactoring specification suggests to duplicate the queuing center suffering of these kind of performance violations. Figure 5 depicts the refactored QN model for our example.

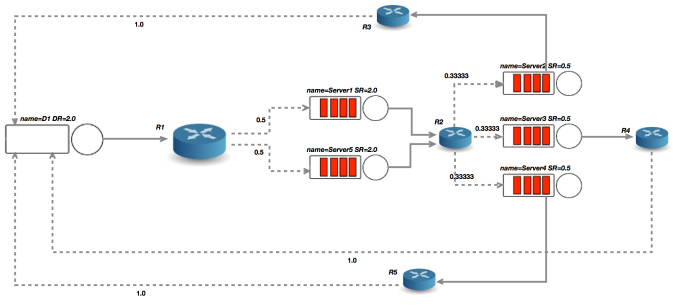



**Figure 5: Refactored Model**

In particular, we added a new service station in parallel to $Server1$ with an equal routing probability. In this way the jobs coming from $D1$ are equally distributed among $Server1$ and $Server2$.

## 5.4 Execution

In this phase we check if the new QN topology is suitable to avoid the performance violations and we actually execute the refactoring actions. This means to update the QN model and, once verified the effectiveness of the refactoring, to report this change in the structure of the running system. Figure 6 shows that the executed refactoring action is suitable to fulfill the constraints specification.

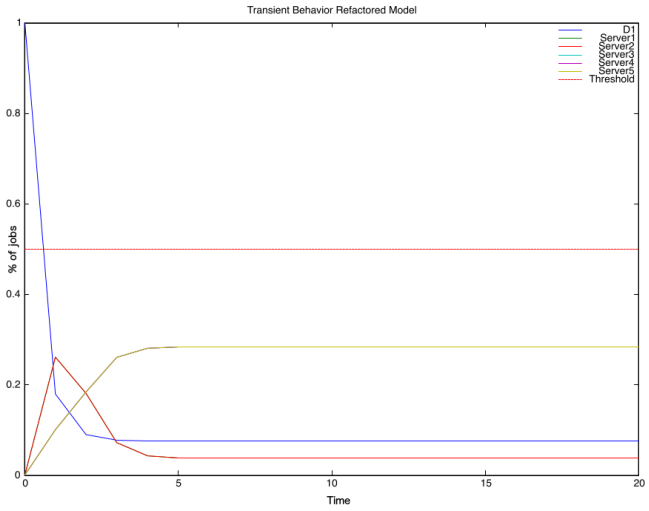



**Figure 6: Transient behavior for the refactored model**

This change in the QN topology (i.e., the addition of a further server) makes the peak no longer happening. After this phase the adaptation iteration ends and the self-adaptation process continues with new iterations using this model as the starting point for the future analysis and adaptation. We are aware that our illustrative refactoring action is straightforward, but the experimental results seem promising to implement further actions for self-adaptation purpose.

# 6. CONCLUSION

In this paper we presented a proactive approach that provides self-adaptation capabilities to software systems in order to guarantee the fulfillment of performance requirements.

The novelty with respect to current state-of-art is that our approach exploits the QN fluid approximation to speed up the analysis of transient dynamics for performance indices. To this end, we developed a tool that automates the generation of simulation and fluid analysis code from a high-level description of QNs.

As future work, our short-term research agenda includes the task of providing a more formal definition of the constraints analysis and refactoring engines, together with the introduction of a language for constraints and refactoring specifications. For the last mentioned task we plan to investigate the use of formal specifications such as Signal Temporal Logic. Furthermore, a systematic comparison between the proposed approach and other simulation techniques is needed. In the long-term we intend to apply our approach to other case studies, possibly coming from real-world domains. This wider experimentation will allow us to deeply investigate the scalability and the usefulness of our approach for self-adaptation, thus quantifying its effectiveness.

# 7. REFERENCES

[1] M. Becker, M. Luckey, and S. Becker. Model-driven performance engineering of self-adaptive systems: a survey. In *Proceedings of the 8th international ACM SIGSOFT conference on Quality of Software Architectures (QoSA)*, pages 117–122, 2012.

[2] L. Bortolussi and M. Tribastone. Fluid limits of queueing networks with batches. In *WOSP/SIPEW International Conference on Performance Engineering (ICPE)*, pages 45–56, 2012.

[3] R. Calinescu, C. Ghezzi, M. Z. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Commun. ACM*, 55(9):69–77, 2012.

[4] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. L. Presti, and R. Mirandola. MOSES: A framework for qos driven runtime adaptation of service-oriented systems. *IEEE Trans. Software Eng.*, 38(5):1138–1159, 2012.

[5] R. M. Carmo, L. R. de Carvalho, E. d. S. e Silva, M. C. Diniz, and R. R. Muntz. Tangram-ii: A performability modeling environment tool. In *Computer Performance Evaluation Modelling Techniques and Tools*, pages 6–18. Springer, 1997.

[6] G. Casale and G. Serazzi. Quantitative system evaluation with java modeling tools. In *Proceedings of the 2Nd ACM/SPEC International Conference on Performance Engineering*, ICPE '11, pages 449–454, New York, NY, USA, 2011. ACM.

[7] G. Casale, M. Tribastone, and P. G. Harrison. Blending randomness in closed queueing network models. *Performance Evaluation*, 82(0):15 – 38, 2014.

[8] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, et al. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*, pages 1–32. Springer, 2013.

[9] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Model evolution by run-time parameter adaptation. In *International Conference on Software Engineering ICSE*, pages 111–121, 2009.

[10] T. Field. Jinqs: An extensible library for simulating multiclass queueing networks, v1. 0 user guide, 2006.

[11] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, Oct. 2004.

[12] M. Kirschnick. The performance evaluation and prediction system for queueing networks-pepsy-qns. 1994.

[13] M. Kowal, I. Schaefer, and M. Tribastone. Family-based performance analysis of variant-rich software systems. In *Fundamental Approaches to Software Engineering (FASE)*, pages 94–108, April 2014.

[14] H. Koziolek. Performance evaluation of component-based software systems: A survey. *Perform. Eval.*, 67(8):634–658, 2010.

[15] T. G. Kurtz. Solutions of ordinary differential equations as limits of pure jump markov processes. *Journal of Applied Probability*, 7(1):pp. 49–58, 1970.

[16] E. D. Lazowska, J. Zahorjan, G. Scott Graham, and K. C. Sevcik. *Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., Englewood Cliffs, 1984.

[17] D. A. Menascé, H. Gomaa, S. Malek, and J. P. Sousa. SASSY: A framework for self-architecting service-oriented systems. *IEEE Software*, 28(6):78–85, 2011.

[18] A. Metzger, O. Sammodi, K. Pohl, and M. Rzepka. Towards pro-active adaptation with confidence: Augmenting service monitoring with online testing. In *Proceedings of ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS, pages 20–28, New York, NY, USA, 2010. ACM.

[19] R. Mirandola and C. Trubiani. A deep investigation for qos-based feedback at design time and runtime. In *IEEE International Conference on Engineering of Complex Computer Systems ICECCS*, pages 147–156, 2012.

[20] J. Musset, É. Juliot, S. Lacrampe, W. Piers, C. Brun, L. Goubet, Y. Lussaud, and F. Allilaire. Acceleo user guide, 2006.

[21] P. Saxena and L. Sharma. Simulation tool for queuing models: Qsim. *International Journal of Computers and Technology*, 5(2):74–79, 2013.

[22] W. J. Stewart. *Probability, Markov Chains, Queues, and Simulation*. Princeton University Press, 2009.

[23] M. Tribastone. A fluid model for layered queueing networks. *IEEE Trans. Software Eng.*, 39(6):744–756, 2013.

[24] M. Tribastone. Efficient optimization of software performance models via parameter-space pruning. In *ACM/SPEC International Conference on Performance Engineering (ICPE)*, pages 63–73, 2014.

[25] K. S. Trivedi and R. Sahner. Sharpe at the age of twenty two. *SIGMETRICS Perform. Eval. Rev.*, 36(4):52–57, Mar. 2009.

[26] M. Tschaikowski and M. Tribastone. Insensitivity to service-time distributions for fluid queueing models. ICST, 1 2014.

[27] A. Van Hoorn, J. Waller, and W. Hasselbring. Kieker: A framework for application performance monitoring and dynamic software analysis. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, pages 247–248. ACM, 2012.

[28] D. Weyns, M. U. Iftikhar, D. G. de la Iglesia, and T. Ahmad. A survey of formal methods in self-adaptive systems. In *Proceedings of the International Conference on Computer Science and Software Engineering*, C3S2E, pages 67–79, New York, NY, USA, 2012. ACM.

# Model-Based Performance Evaluations in Continuous Delivery Pipelines

Markus Dlugi, Andreas Brunnert
fortiss GmbH
Guerickestr. 25
80805 Munich, Germany
{dlugi|brunnert}@fortiss.org

Helmut Krcmar
Technische Universität München
Boltzmannstr. 3
85748 Garching, Germany
krcmar@in.tum.de

## ABSTRACT

In order to increase the frequency of software releases and to improve their quality, continuous integration (CI) systems became widely used in recent years. Unfortunately, it is not easy to evaluate the performance of a software release in such systems. One of the main reasons for this difficulty is often the lack of a test environment that is comparable to a production system. Performance models can help in this scenario by eliminating the need for a production-sized environment. Building upon these capabilities of performance models, we have introduced a model-based performance change detection process for continuous delivery pipelines in a previous work. This work presents an implementation of the process as plug-in for the CI system Jenkins.

## Categories and Subject Descriptors

C.4 [**Performance of Systems**]: measurement techniques, modeling techniques

## General Terms

Measurement, Performance

## Keywords

Performance Evaluation, Performance Change Detection, Palladio Component Model, Continuous Delivery

## 1. INTRODUCTION

The modern software engineering landscape has been reshaped significantly in recent years by new paradigms such as agile software development. This led to a shift from the traditional model of releasing new features in few, major versions to rapid release cycles pushing new features and bug fixes to the user in increasingly shorter intervals. In order to support such short release cycles and still ensure adherence to all requirements during the entire development process, new concepts like continuous integration (CI), continuous delivery (CD) or continuous deployment have emerged.



*QUDOS'15*, September 1, 2015, Bergamo, Italy
ACM. 978-1-4503-3817-2/15/09...$15.00
http://dx.doi.org/10.1145/2804371.2804376

However, while functional requirements are rigorously evaluated in these systems to avoid breaking the application, non-functional requirements like performance (i.e., response time, utilization and throughput) are only rarely examined. In the case of software performance, one frequent reason for this is the lack of an adequate performance test environment [3]. Performance models such as the Palladio Component Model (PCM) [1] bear the potential to help in this scenario. By automatically generating a performance model and running a simulation, a prediction of an enterprise application's (EA) performance can be made which is independent of the environment used to generate the model [4]. By integrating such a model-based performance evaluation process into a CD deployment pipeline, changes in an EA's performance can be detected for every build of the EA. This capability reduces development costs and risks as performance regressions introduced by new features or bugs can be detected immediately, thus avoiding expensive fixes once the EA is in production.

This work presents the integration of an existing performance change detection process [2] into Jenkins[1] as an exemplary CI system. It builds upon the previous work that demonstrated the feasibility of such an approach and presents an integrated tool for the whole process.

## 2. CONTINUOUS INTEGRATION, DELIVERY & DEPLOYMENT

CI describes the practice of ensuring a usable application during all stages of the development process [5]. This means that every time a developer commits their changes, they are instantly integrated with the rest of the application and acceptance tests are performed to guarantee the soundness and stability of the system.

CD takes the idea of CI one step further and demands that the application not only be usable, but also in a deployable state for every successful release candidate [5]. A key concept of CD is a so-called deployment pipeline. It consists of multiple steps such as acceptance testing, capacity testing or packaging to produce a deployable artifact in the end.

Finally, continuous deployment is the practice of actually deploying every application version that has passed the necessary tests to production [5].

## 3. PERFORMANCE CHANGE DETECTION

The tool for performance change detection is realized as a Jenkins plug-in. The process necessary for providing the

[1] http://jenkins-ci.org

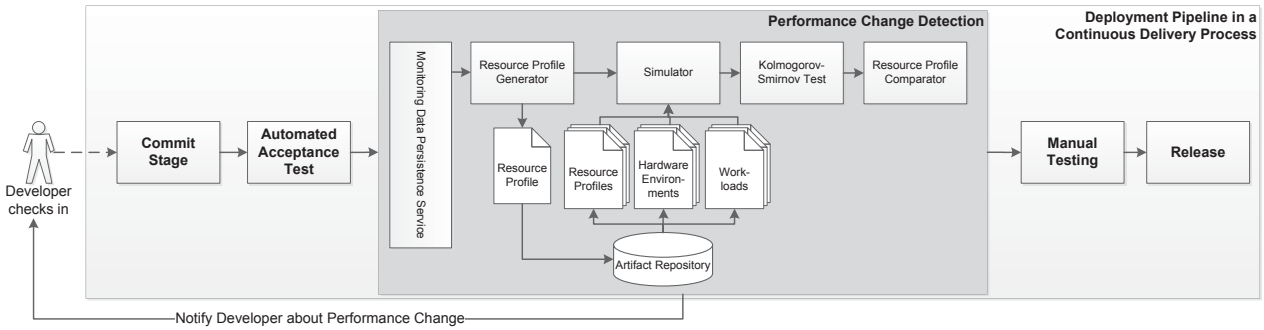



**Figure 1: Performance Change Detection Process (adapted from [2, 5])**

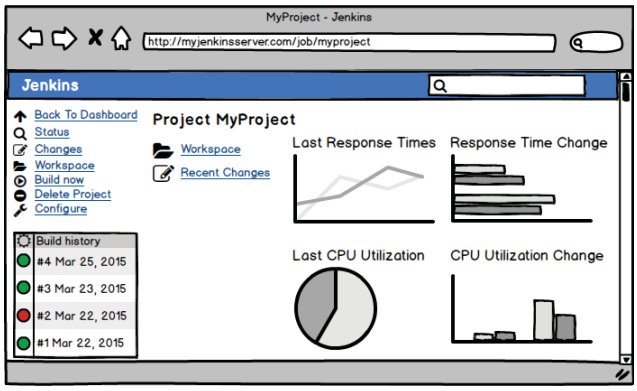



**Figure 2: Mockup of the Web UI**

change detection functionality is illustrated in figure 1.

Once the build process and the automated acceptance tests are completed, the first process step is to generate specifically formed performance models called resource profiles. A resource profile describes the performance-relevant aspects of an EA by depicting the control flow and resource demand of single transactions on the level of single component operations. Users can specify the workload for a resource profile on the level of single transactions and the deployment topology using deployment units. The data necessary for the generation is gathered using application performance monitoring (APM) tools. In order to support different monitoring technologies, the Jenkins plug-in provides a configuration to define which APM solution is used to collect the necessary data. Out of the box, the plug-in provides support for dynatrace[2] as well as a custom monitoring solution. However, it can be extended to additional APM tools.

The resulting resource profile is stored in an artifact repository [2]. The artifact repository is responsible for holding performance model information about each of the EA's revisions and is realized as EMFStore server. Besides the various resource profile versions, it also contains the hardware environments and workloads designed for the performance evaluation; these artifacts need to be manually created prior to the change detection. The Simulator retrieves the generated resource profile as well as the predefined hardware environments and workloads and executes a simulation for each combination. This step is repeated for all resource profile versions to which the latest version should be compared.

After all simulations have finished, the resulting performance metrics of all examined application versions are compared using two-sample Kolmogorov-Smirnov tests to determine whether the samples differ significantly. If a performance regression is found during one of the tests, the Resource Profile Comparator tries to find the reason for it by analyzing the control flow of the business transactions containing a regression, comparing the involved resource profiles using EMFCompare and computing their difference. Finally, a report is generated containing visualizations of the examined performance metrics as well as a list of the methods that are likely to be responsible for the observed regression. A mockup of the information available to the developer is depicted in figure 2.

[2]http://www.dynatrace.com

## 4. CONCLUSION AND FUTURE WORK

This work presented the integration of a model-based performance evaluation process into an exemplary CI system. While the basic process has been implemented and is working as described in section 3, there are still many aspects of the prototype that can be refined and improved. One major task is the development of a configuration interface to enable the developer to easily manipulate the various process parameters as well as the hardware environment and workload models.

## 5. REFERENCES

[1] S. Becker, H. Koziolek, and R. Reussner. The palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82(1):3–22, 2009.

[2] A. Brunnert and H. Krcmar. Detecting performance change in enterprise application versions using resource profiles. In *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS '14, pages 165–172, ICST, Brussels, Belgium, 2014.

[3] A. Brunnert, C. Vögele, A. Danciu, M. Pfaff, M. Mayer, and H. Krcmar. Performance management work. *Business & Information Systems Engineering*, 6(3):177–179, 2014.

[4] A. Brunnert, K. Wischer, and H. Krcmar. Using architecture-level performance models as resource profiles for enterprise applications. In *Proceedings of the 10th International ACM Sigsoft Conference on Quality of Software Architectures*, QoSA '14, pages 53–62, New York, NY, USA, 2014. ACM.

[5] J. Humble and D. Farley. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation.* Addison-Wesley Professional, 1st edition, 2010.

# Continous Deployment of Multi-cloud Systems

Nicolas Ferry, Franck Chauvel, Hui Song, Arnor Solberg
Department of Networked Systems and Services, SINTEF, Oslo, Norway
{name.surname}@sintef.no

## ABSTRACT

In this paper we present our mechanism and tooling for the continuous deployment and resource provisioning of multi-cloud applications. In order to facilitate collaboration between development and operation teams as promoted in the DevOps movement, our deployment and resource provisioning engine is based on the Models@Runtime principles. This enables applying the same concepts and language (*i.e.*, CLOUDML) for deployment and resource provisioning at development-and operation-time.

## Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Configuration Management

## Keywords

Cloud computing, deployment, model-driven engineering, Models@Runtime, CloudML

## 1. INTRODUCTION

Multi-cloud systems are typically large scale, distributed, and dynamic. The need to evolve and update such systems after delivery is often inevitable, for example due to: changes in the requirements, needs for maintenance, or demands for advancing the quality of service such as scalability and performance. In order to ensure the quality of the system, validation and analysis tasks have to be carried out during all development steps towards the delivery of the system. This includes testing the deployment as well as the various features of the system in a pre-production environment. Such testing usually involves both software developers that are responsible for designing and implementing the system according to the functional and non-functional requirements, and software system operators that are responsible for operating the system at runtime.

Traditionally, these teams are working separately, typically using specific languages and tools they feel comfortable with [3]. In addition, languages and tools typically vary within these teams since various cloud platforms offer different languages and tools [3]. This can hinder knowledge sharing, making it difficult: i) for designers to obtain and understand feedback on the status of the operated system that would be useful in order to evolve the system, ii) for operators to analyse and comment on the impact of proposed or implemented changes to the system, and iii) for operators of different cloud platforms to exchange knowledge and specific operation management decisions. The DevOps movement [4] promotes to facilitate collaboration between developers and operators, for example, through aligning concepts and languages used in development and operation and supporting them with automated tools that help reducing the gap and improving the flexibility and efficiency of the delivery life-cycle.

In this tool paper we present the CLOUDMF mechanism and tooling to reduce the gap between development and operation teams by supporting continuous deployment of multi-cloud applications. In order to reduce this gap, we apply the same concepts and language at development-time and at operation-time. To automate the continous deployment and resource provisioning, we have developed an engine based on the principles of the Models@Runtime approach [1]. The deployment engine "speaks" the language of CLOUDML [2], thereby, it enables to apply the same concepts and abstractions for the operators as applied by the developers. Moreover, it enables seamless operation of multi-cloud deployment and provisioning since our previously developed CLOUDML modelling language provides abstractions enabling to specify deployment and resource provisioning in a cloud platform agnostic way.

## 2. CONTINUOUS DEPLOYMENT USING MODELS@RUNTIME

Models@Runtime [1] is an architectural pattern for dynamic adaptive systems that leverage models as executable artefacts that can be applied to support the execution of the system. Thus, Models@Runtime can be applied to reduce the developer-operator gap by providing a unique model-based representation of the applications that can be applied for both design- and run-time activities. As depicted in Figure 1, Models@Runtime enables to provide abstract representations of the underlying running system, which facilitates reasoning, analysis, simulation, and adaptation. A change in the running system is automatically reflected in the model of the current system. Similarly, a modification to this model is enacted on the running system on demand. This causal connection enables the continuous evolution of the system with no strict boundaries between design-time and run-time activities.

In our tooling we exploit Models@Runtime for the continuous deployment of cloud-based applications following the process depicted in Figure 1. A developer team specifies a model of the deployment and provisioning of its application applying CLOUDML, and then automatically enacts this deployment and provisioning in a test environment. Developers exploit the test environment to



*QUDOS'15*, September 1, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3817-2/15/09...$15.00
http://dx.doi.org/10.1145/2804371.2804377

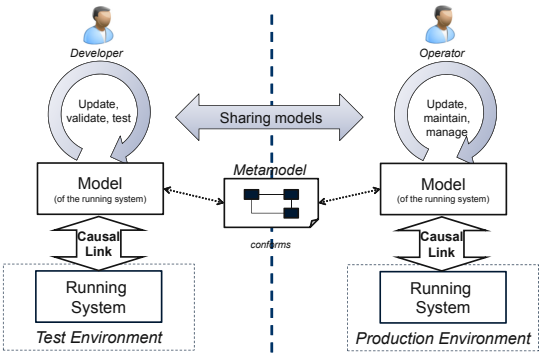



**Figure 1: Continuous deployment using Models@Runtime**

check, analyse and validate the deployment and resource provisioning as well as the system's features, and continously tune its development and redeploy it automatically. Any change made to the deployment model will be enacted on demand on the running system whilst its status will be reflected in the model providing useful feedbacks. Once the new release is validated, it can be provided together with the associated CLOUDML model to the operation team. The latter can in turn exploit the model to deploy the new release in a production environment. The operators can further tune this model to maintain, manage and evolve the running system. Because the models shared by the developers and operators conform to the same metamodel, at any time they can share and exchange information. Moreover, any automation of the ''hand-over'' from the Test Environment to the Operation Environment can be provided seamlessly.

## 3. THE DEPLOYMENT ENGINE

Figure 2 depicts the architecture of our Models@Runtime based deployment engine. A reasoning engine (*e.g.*, operator, analysis engine) can read the current model provided by the deployment engine (step 1), which describes the actual running system, and then produces a target model (step 2). Then, the deployment engine calculates the difference between the current and the target model (step 3). Finally, the deployment engine enacts the changes modifying only the parts of the system necessary to account for the difference, and the target model becomes the current (step 4).

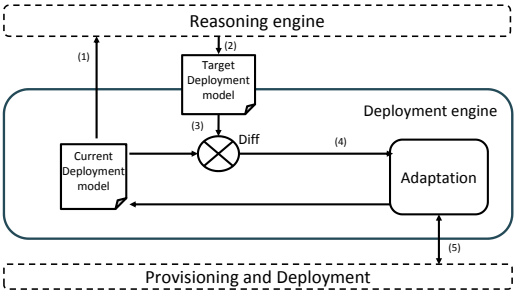



**Figure 2: The Models@Runtime based Deployment Engine**

Figure 3 shows the web interface for viewing and manipulating the deployment and resource provisioning either in the test environment at design-time or in the operation environment at run-time.

Adaptations in the deployment of a cloud-based application can be specified by providing a new deployment model describing the

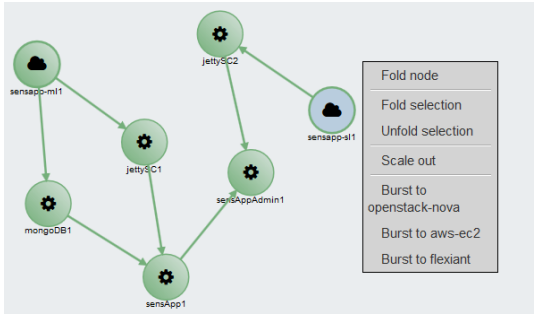



**Figure 3: Snapshot of the CLOUDMF Web-based editor**

desired deployment. A Comparison engine compares the current and target models and derives a plan describing how to reach that deployment. The resulting plan modifies only the parts of the system necessary to account for the difference thus minimizing the impact on the system in production (from a quality point it is important to not touch the running system more than necessary).

The comparison engine processes the entities composing the deployment models on the basis of their logical dependencies. In this way, all the components required by another component are deployed first. For each of these components, the engine compares the two sets of instances from the current and target models. This comparison is achieved on the matching of both the properties of the instances and their types as well as on the basis of their dependencies (*e.g.*, if the host of a software component has changed it might be redeployed). For each unmatched instance from the current model a `remove` action with the instance as argument is created. Similarly, for each unmatched instance from the target model an `add` action with the instance as argument is generated. In addition, by analysing the difference between the two models, the comparison engine identifies the high level operations to be performed. For instance, when bursting to a new provider, the engine triggers a classical deployment, whilst when scaling within the same cloud, it creates and provisions image of the VM to be scaled and then reconfigures and restarts the components hosted on it.

## 4. CONCLUSION

Our Models@Runtime approach leverages upon MDE techniques and methods at runtime to support the continuous design and deployment of multi-cloud applications. Thanks to the proposed approach, it becomes possible to exploit the same concepts and language for deployment and resource provisioning at both development and operation time.

*Acknowledgements.* The research leading to these results has received funding from the European Community's FP7 program under grant agreement number: 318484 (MODAClouds).

## 5. REFERENCES

[1] G. Blair, N. Bencomo, and R. France. Models@run.time. *IEEE Computer*, 42(10):22–27, 2009.

[2] N. Ferry, H. Song, A. Rossini, F. Chauvel, and A. Solberg. CloudMF: Applying MDE to Tame the Complexity of Managing Multi-Cloud Applications. In *UCC*, 2014.

[3] M. Httermann. *DevOps for developers*. Apress, 2012.

[4] J. Humble and D. Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010.

# SPACE4Cloud: A DevOps Environment for Multi-cloud Applications

Michele Guerriero
Politecnico di Milano,
Dipartimento di Elettronica,
Informatica e Bioingegneria.
Via Golgi 42 20133, Milano,
Italy
michele.guerriero@polimi.it

Michele Ciavotta
Politecnico di Milano,
Dipartimento di Elettronica,
Informatica e Bioingegneria.
Via Golgi 42 20133, Milano,
Italy
michele.ciavotta@polimi.it

Giovanni Paolo Gibilisco
Politecnico di Milano,
Dipartimento di Elettronica,
Informatica e Bioingegneria.
Via Golgi 42 20133, Milano,
Italy
giovannipaolo.gibilisco@polimi.it

Danilo Ardagna
Politecnico di Milano,
Dipartimento di Elettronica,
Informatica e Bioingegneria.
Via Golgi 42 20133, Milano,
Italy
danilo.ardagna@polimi.it

## ABSTRACT

Cloud computing has been a game changer in the design, development and management of modern applications, which have grown in scope and size becoming distributed and service oriented. New methodologies have emerged to deal with this paradigm shift in software engineering. Consequently, new tools, devoted to ease the convergence between developers and other IT professional, are required. Here, we present SPACE4Cloud, a DevOps integrated environment for model-driven design-time QoS assessment and optimization, and runtime capacity allocation for Cloud applications.

## Categories and Subject Descriptors

D.2.2 [**Design Tools and Techniques**]: Computer-aided software engineering (CASE); D.2.9 [**Management**]: Software quality assurance (SQA)

## Keywords

Model-Driven, Cloud, QoS, design-time, runtime, DevOps

## 1. INTRODUCTION

In recent years we have witnessed a paradigm shift in the software creation and management. Projects have progressively grown in size and scope, and the legacy model of standalone applications has lost much of its relevance in favor of more flexible, distributed, and web-based architectures. Another factor to consider in this change is the emergence and the success of Cloud computing. Overall, new ideas for application development and management have appeared, which resulted in DevOps, *i.e.*, a software development method based on collaboration, more often on a real convergence, between software developers, system administrators, and performance engineers. Under these circumstances, tools that simplify the early quality evaluation at design-time and systems for managing the quality at runtime have become especially important. In this work we propose SPACE4Cloud, a collection of tools developed within the MODAclouds[1] EU FP7 project for design-time modeling and analysis, and runtime quality management of multi-Cloud applications.

The rest of the paper is organized as follows: in Section 2 we introduce SPACE4Cloud; some experimental results are presented in Section 3 whereas conclusions are finally drawn in Section 4.



*QUDOS'15*, September 1, 2015, Bergamo, Italy
ACM. 978-1-4503-3817-2/15/09...$15.00
http://dx.doi.org/10.1145/2804371.2804378

## 2. SPACE4CLOUD

SPACE4Cloud (System PerformAnce and Cost Evaluation on Cloud) is an integrated environment for model-driven design-time QoS assessment, optimization, and runtime capacity allocation of Cloud applications. It is composed of two main tools: at design-time **SPACE 4Cloud**$^{Dev}$ takes in input models in extended PCM format [2] describing the application under development in terms of functionalities, Quality of Service (QoS) requirements, and end-user workload profile defined over a 24-hour time horizon. Such models are converted in Layered Queueing Networks [4] and evaluated in terms of cost and performance. The tool is also able to perform, through a local-search-based metaheuristic, a fully automatized exploration of the space of possible Cloud offers, seeking for the configuration that minimizes the execution costs fulfilling at once the QoS requirements. The outcome of this module is a new set of models describing the Cloud deployment and the runtime adaptation actions. These models are, in turn, fed into **SPACE4Cloud**$^{Ops}$,

[1] www.modaclouds.eu

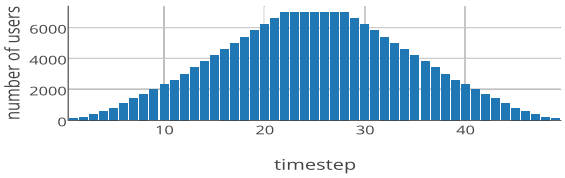


Figure 1: Test workload

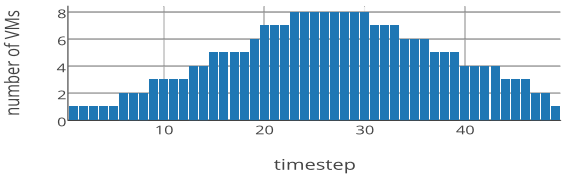


Figure 2: Varying number of VMs

which is the tool in charge of guaranteeing at runtime the pledged QoS levels by enacting suitable policies. This solutions implements the so-called Receding Horizon Control paradigm. Simply put, it solves a Mixed Integer Linear Programming (MILP) optimization problem over a finite time window; this process generates a set of scaling actions to be implemented within the MODAClouds Runtime Platform [3], using a suitable IaaS interface, in order to obtains a satisfactory QoS during the time window, yet only the adaptations for the first time step are enforced. Eventually, the process is repeated considering the second time step as the beginning of a new time window. The MILP problem, in its basic formulation, is a capacity allocation problem in charge to determine the number of VMs suitable to serve the expected incoming workload, minimizing costs and guaranteeing that the QoS requirements are met. The optimization model, based on queueing theory results, is characterized by performance parameters that are continuously updated at runtime by an appropriate monitoring system (see [3]) in order to cope with the time-varying behavior of the Cloud. Moreover, since the workload varies over time and it can not be known in advance, the allocation problem is solved based on a prediction, also provided by the monitoring system.

The entire SPACE4Cloud environment is implemented in Java, released under Apache License 2.0[2] [3]

## 3. EXPERIMENTAL RESULTS

In this section we briefly describe some of the experiments performed to prove the soundness of our approach. The experiments are all performed using a simplified single-tier web application and leveraging Amazon EC2 services. Embracing a model-base approach, we start by modeling our application and the operational environment at design-time using the extended PCM format. Along this path, initial estimates of service times are derived by performing some preliminary experiments on a prototype environment and relying on state-of-the-art parameter estimation techniques [5]. In this way we could describe the application and the runtime environment, and feed with the resulting models **SPACE4Cloud**$^{Dev}$. Once the application has been automatically deployed in the MODAClouds runtime environment, **SPACE4Cloud**$^{Ops}$ starts creating the optimization model and enacting the runtime control loop with a 5 minute timescale against a synthetic workload generated by Apache JMeter[4]. A timescale of 5 minutes has been set up since it has been proved in [1] to provide better performance with respect to larger ones, essentially due to more accurate workload predictions available at this scale. Moreover in [1] a comparison with a heuristic currently implemented by some IaaS providers is reported, showing that our approach always provides better solutions. Here, focusing on the effectiveness of our approach, a basic experiment carried out using the 4-hour workload profile shown in Figure 1 is presented; the incoming workload is basically a ramp up with a maximum number of users equal to 7,000, followed by a ramp down. Figure 2 shows the number of running VMs varying in the range [1, 8] following the shape of the workload, proving the effectiveness of the adaptive mechanism provided by **SPACE 4Cloud**$^{Ops}$.

[2]github.com/deib-polimi/modaclouds-space4cloud
[3]github.com/deib-polimi/modaclouds-autoscalingReasoner
[4]jmeter.apache.org

## 4. CONCLUSIONS

In this work we presented a joint DevOps environment for design-time modeling and optimization, and runtime control for Cloud applications. The aim of the tool is to minimize the execution costs of Cloud applications providing QoS guarantees by design. The most distinguished characteristic of the Cloud, such as variable workload, congestion due to multi-tenancy, and performance variability, are considered. Future work will be devoted mainly to extend the runtime adaptive actions to PaaS platforms and multi-Cloud applications, both already managed at design-time. Moreover, a feedback mechanism will be implemented in MODAClouds runtime environment to provide a better estimation of design-time parameters. In this way the user, dealing with an application model closer to reality, can further improve the deployment, discovering and solving possible performance bottleneck. Finally, the tool will be extended for design-time modeling and optimization of data intensive applications within the framework of the DICE[5] project.

## 5. ACKNOWLEDGMENTS

The research reported in this article is partially supported by the European Commission grants no. FP7-ICT-2011-8-318484 (MODAClouds) and H2020-ICT-2014-1-644869 (DICE).

## 6. REFERENCES

[1] D. Ardagna, M. Ciavotta, and R. Lancellotti. A receding horizon approach for the runtime management of iaas cloud systems. In *SYNASC-MICAS 2014*.

[2] D. Ardagna, M. Ciavotta, M. Miglierina, G. Gibilisco, G. Casale, J. Pérez, F. D'Andria, and R. S. González. MODAClouds D5.2.2 - MODACloudML QoS abstractions and prediction models specification, 2014.

[3] G. Iuhasz, S. Panica, G. Casale, W. Wang, P. Jamshidi, D. Ardagna, M. Ciavotta, D. Whigham, N. Ferry, and R. S. González. MODAClouds D6.4.2 - runtime environment final release, 2015.

[4] J. A. Rolia and K. C. Sevcik. The method of layers. *IEEE Trans. Softw. Eng.*, 21(8):689–700, Aug. 1995.

[5] L. Zhang, X. Meng, S. Meng, and J. Tan. K-scope: Online performance tracking for dynamic cloud applications. In *ICAC 2013*.

[5]www.dice-h2020.eu

# Filling the Gap: A Tool to Automate Parameter Estimation for Software Performance Models

Weikun Wang, Juan F. Pérez, Giuliano Casale
Department of Computing
Imperial College London
UK
{weikun.wang11,j.perez-bernal,g.casale}@imperial.ac.uk *

## ABSTRACT

Software performance engineering heavily relies on application and resource models that enable the prediction of Quality-of-Service metrics. Critical to these models is the accuracy of their parameters, the value of which can change with the application and the resources where it is deployed. In this paper we introduce the Filling-the-gap (FG) tool, which automates the parameter estimation of application performance models. This tool implements a set of statistical routines to estimate the parameters of performance models, which are automatically executed using monitoring information kept in a local database.

## Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*performance measures*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Computer-aided software engineering*

## General Terms

Theory

## Keywords

Software Performance Engineering; Quality of Service

## 1. INTRODUCTION

DevOps [5] is a recent trend in software engineering that bridges the gap between software development and operations, putting the developer in greater control of the application operational environment. To support Quality-of-Service (QoS) analysis, the developer may rely on software performance models. However, to provide reliable estimates, the input parameters must be continuously updated and accurately estimated. Accurate estimation is challenging because some parameters are not explicitly tracked by log files, requiring deep monitoring instrumentation that poses large overheads, unacceptable in production environments.

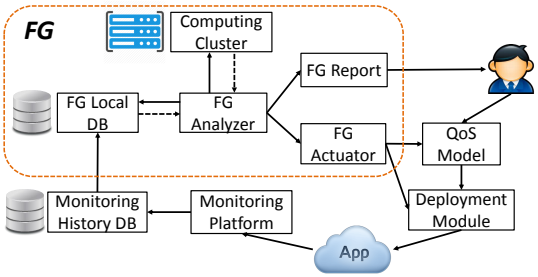



**Figure 1: FG Architecture**

In this paper we present the first release of the Filling-the-Gap (FG) tool, a tool for continuous performance model parametrization that implements the research agenda set in [4]. The FG tool implements a set of statistical estimation algorithms to parameterize performance models from runtime monitoring data. Multiple algorithms are included, allowing for alternative ways to obtain estimates for different metrics, but with an emphasis on resource demand estimation, which has recently been contemplated also in tools such as LibReDE [6]. FG tool supports advanced algorithms to estimate parameters based on response times and queue-length data, which makes the tool useful in particular for applications running in virtualized environments where utilization readings are not always available.

Figure 1 shows the four main components of the FG Tool.

**FG Local DB:** This local database periodically acquires and stores the monitoring data that is relevant for FG analysis and used by the FG Analyzer. The Local DB is a Fuseki[1] database, which keeps data in RDF format.

**FG Analyzer:** The FG Analyzer executes the statistical methods necessary to obtain the estimates of the performance models parameters, relying on the monitoring information available on the Local DB. It has the ability to connect with a Condor cluster to process estimation routines in parallel.

**FG Actuator:** The FG Actuator updates the parameters

*The research leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement no. 318484. and DICE H2020 under grant agreement no. 644869. The data reported is available at http://dx.doi.org/10.5281/zenodo.20280.



*QUDOS'15*, September 1, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3817-2/15/09...$15.00
http://dx.doi.org/10.1145/2804371.2804379

[1] `http://jena.apache.org/documentation/serving_data/`

of the models, e.g., resource demands, population, etc., obtained from the FG Analyzer. The update is performed on the performance and the deployment model.

**FG Reporter:** The FG Reporter is in charge of providing the developers with information regarding the application runtime behavior to help them evaluate how well the application responds under different conditions. The FG Reporter is based on DynamicReports[2].

## 2. SUPPORTED DEMAND ESTIMATION ALGORITHMS

Queueing networks are popular abstractions used as application performance models. To parametrize these models, the FG tool uses the monitoring data collected at runtime and executes statistical methods, improving the accuracy of the models. Among the set of model parameters, resource demand is difficult to estimate since extensive monitoring poses unacceptable overheads. To tackle this problem, the FG Analyzer implements several algorithms, relying on different monitoring metrics. Here we use $D_r$ to represent the service demand of request class $r$.

**CI:** the Complete Information (CI) method [3] requires a full trace, that is, the times at which every request arrives and departs from the resource. Consider a class-$r$ request that arrives at $t_1$ and departs at $t_I$, where $I$ is the number of observed events (request arrivals or departures) during the request execution. Then the demand of that request on that resource is:

$$D_r = \sum_{i=1}^{I-1}(t_{i+1} - t_i)\min(n(t_i^+), V)/n(t_i^+)$$

where $n(t_i^+)$ is the number of requests in execution just after time $t_i$ and $V$ is the number of CPUs.

**GQL:** the Gibbs sampling with Queue Lengths (GQL) method [7] uses queue-length samples collected at run time to estimate the service demand with the Bayes' theorem:

$$P(\boldsymbol{D}|\boldsymbol{N}) = P(\boldsymbol{N}|\boldsymbol{D})P(\boldsymbol{D})/P(\boldsymbol{N}) \approx \prod_{\boldsymbol{n}\in\boldsymbol{N}} P(\boldsymbol{n}|\boldsymbol{D})P(\boldsymbol{D}),$$

where $P(\boldsymbol{n}|\boldsymbol{D})$ is the steady state probability of a product form queueing network, $\boldsymbol{N}$ is the observed queue length dataset and $\boldsymbol{n}$ is one entry of the dataset $\boldsymbol{N}$, i.e. $n_{ir}$ is the number of class-$r$ jobs at station $i$. Gibbs sampling is employed to obtain the demand $\boldsymbol{D}$.

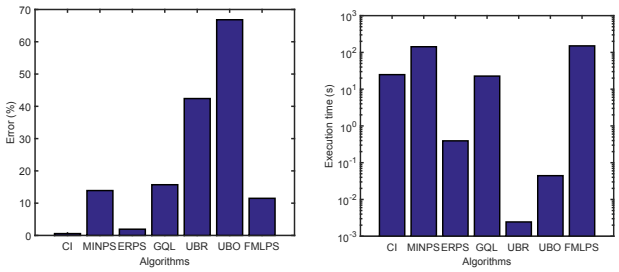
**MINPS/FMLPS:** the MINPS method [3] is a maximum likelihood method based on a Markov Chain representation of the response time given the observed queue length. It requires response times and queue lengths observed upon arrival. Similarly, FMLPS [3] is also a maximum likelihood estimator but uses a fluid approximation to obtain estimates for large systems.

**ERPS:** the Extended Regression-Based (RPS) approach [3] makes use of the response times and queue lengths observed at arrival times as in the equation

$$E[R_r] = E[D_r]E[\bar{A}^r]/\min\left\{V, 1/I\sum_{i=1}^{I} n(t_i)\right\},$$

where $A^r$ is the queue-length seen upon arrival by class-$r$ jobs, and $R_r$ is their response time. The service demand is estimated using linear regression.

[2] `http://www.dynamicreports.org/`




(a) Error (%) (b) Execution time (s)

**Figure 2: Comparison between demand estimation algorithms**

**FCFS:** estimations for FCFS servers [1] rely on

$$E[R_r] = E[T_r] + \sum_{k=1}^{K} E[D_r](1_{k,r} + E[A_k^r]),$$

where $1_{k,r} = 1$ if $k = c$ otherwise $1_{k,r} = 0$, $T_r$ is the residual time to completion of a class-$r$ request, and $A_k^r$ is the queue-length of class-$k$ jobs seen upon arrival by each class-$r$ request. Demand estimates $E[D_r]$ are obtained with regression methods given $A_k^r$ and $R_r$.

### 2.1 Evaluation

One advantage of the FG tool is the availability of different estimation algorithms in a common environment. We make use of the tool to provide a novel comparison of these algorithms. We simulate the underlying Markov chain of a closed network with 4 request classes, 2 queueing stations, and one delay node. The queueing stations have 2 servers and the number of jobs for each class is $(7, 7, 1, 5)$. We simulate a total of 200000 arrival and departure events, and generate all the metrics required by the estimation algorithms.

Figure 2 shows the experiment result, including the UBR [8] and UBO [2] methods, which use CPU utilization measurements. We observe that CI achieves the highest accuracy while UBR is the most efficient one.

## 3. REFERENCES


[1] Kraft, S., Pacheco-Sanchez, S., Casale, G., Dawson, S.: Estimating service resource consumption from response time measurements. In: VALUETOOLS, 2009.

[2] Liu, Z., Wynter, L., Xia, C.H., Zhang, F.: Parameter inference of queueing models for IT systems using end-to-end measurements. Perf. Eval, 63(1):36–60,2006.

[3] Perez, J.F., Casale, G., Pacheco-Sanchez, S.: Estimating computational requirements in multi threaded applications. IEEE TSE 41(3): 264–278, 2014.

[4] Perez, J.F., Wang, W., Casale, G.: Towards a devops approach for software quality engineering. In: WOSP, 2015.

[5] Roche, J.: Adopting DevOps practices in quality assurance. CACM, 56(11), 2013.

[6] Spinner, S., Casale, G., Zhu, X., Kounev, S.: Librede: a library for resource demand estimation. In: ICPE, 2014.

[7] Wang, W., Huang, X., Qin, X., Zhang, W., Wei, J., Zhong, H.: Application-level cpu consumption estimation: Towards performance isolation of multi tenancy web applications. In: IEEE CLOUD, 2012.

[8] Zhang, Q., Cherkasova, L., Smirni, E.: A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In: ICAC, 2007.

# Author Index