Eliciting Operations Requirements for Applications

Len Bass, Ross Jeffery, Hiroshi Wada, Ingo Weber, Liming Zhu
NICTA and University of New South Wales
Sydney, Australia

***Abstract*—The DevOps community advocates communication between the operations staff and the development staff as a means of ensuring that the developers understand the issues associated with operations. This paper argues that "communication" is too vague and that there are a variety of specific and well known sources that developers can examine to determine requirements to support the installation and operations of an application product. These sources include standards, process descriptions, studies about sources of failure in configuration and upgrade, and models that include both product and process.**

***Index Terms*—operations processes, applications requirements, devops**

I. Introduction

When determining requirements for a new release of a product, operations sources are becoming as important as business sources. According to Gartner, "Through 2015, 80% of outages impacting mission-critical services will be caused by people and process issues, and more than 50% of those outages will be caused by change/configuration/release integration and hand-off issues." [1]. Making applications operations process aware is one technique for reducing the number of problems caused by change/configuration/release integration and hand-off problems.

Release Engineering is defined as "activities in between regular development and delivery of a software product to the end user, i.e. integration, build, test execution, packaging and delivery of software".[1] This definition assumes that "regular development" is focused only on product features of use to the end user. In fact, as we argue here, regular development should include features of use to the delivery process as well as to the operations process since including those features will simplify the release process and reduce the number of associated errors. For example, making the product test for correctness of configuration dependencies (one of our examples) will reduce errors in the installation process which is considered a portion of Release Engineering.

Our sources for eliciting operations requirements, in addition to asking operations personnel, are: standards, organizational process descriptions, and academic studies. This provides the organization of this paper. What standards are relevant and how do we use them to elicit requirements? How do we utilize process descriptions? What can we gather from academic studies on configuration and upgrade failures? What have we learned from our internal development efforts? We end with a description of how we are using operations process/product modeling as a vehicle for determining product requirements to support operations.

II. Standards

Operations processes are standardized in ISO 15504 (SPICE) [7] and ITIL [8] yet these standards are concerned mainly with managing the operations process and are not specific enough to get explicit requirements. Mining these standards for requirements that could be incorporated into products has not been productive for us.

What *is* productive is mining standards in other areas.

NIST 800-53 [9] provides a widely used list of requirements for security. It has both technical (product related) and operational requirements. We sample two operational requirements from this reference to demonstrate that product requirements can be generated from operational considerations. For each operational requirement, we ask the question: how can the product under development support this requirement? We first quote the operational requirement from NIST 800-53 (in italics) and then we propose (in bold) a product requirement that supports the NIST operational requirement.

Requirement CM-8. INFORMATION SYSTEM COMPONENT INVENTORY

The organization develops, documents, and maintains an inventory of information system components that:

a. Accurately reflects the current information system;

b. Is consistent with the authorization boundary of the information system;

c. Is at the level of granularity deemed necessary for tracking and reporting;

d. Includes [Assignment: organization-defined information deemed necessary to achieve effective property accountability];

e. Is available for review and audit by designated organizational officials.

Product Requirement: **The product shall provide a method to create an inventory of information system components for auditors' review. The product shall provide a method to verify that its current version is registered in the component inventory.**

Requirement CP-6 ALTERNATE STORAGE SITE

[1] http://releng.polymtl.ca/

The organization establishes an alternate storage site including necessary agreements to permit the storage and recovery of information system backup information.
Product Requirement: **The product shall maintain consistent backup information in an alternate storage site. The product shall provide a method to assess the recoverability of information system.**

III. Organizational Process Descriptions

Organizations routinely develop extensive processes for operations. These processes provide the detail lacking in SPICE or ITIL. We will use a process description provided by Amazon Web Services (AWS) for migrating resources to a new region (a geographically distant data center) [6]. From this we will derive a new requirement for a product. The key idea is that the product be made migration aware. Again, we show the relevant quote in italics from the description followed by a proposed new requirement in bold.

… the user must take care to update any Auto Scaling launch configuration …to use keys that are available in a new region, or deploy the public key with the same key pair name to the new region p5.

Product Requirement: **The product shall provide a method to verify that Auto Scaling Launch configuration utilizes SSH keys that are available in the region in which it is hosted.**

The point of this example is not that this is a reasonable requirement, in fact it might not be. The point is that an operations process description provides a large number of opportunities for product requirements and these descriptions should be mined to determine which such requirements provide an appropriate cost benefit.

IV. Academic Studies

There have been academic studies of the incidence and causes of failures from upgrades and from configuration errors. These studies provide another source for product requirements

A. Upgradability failure modes

A 2007 survey of system administrators showed an average of 8.6% of upgrades fail [4]. Upgrade failures can be devastating to a business. Recently, an upgrade failure cost the Knight Trading company US $440 Million [11].

Crameri and his colleagues [2] found that the most common source of upgrade problems is the difference between the environment (i.e. version of operation system and libraries, configuration settings, environment variables, etc.) at the developer's site and the users' sites. The term "hidden dependencies" is used to identify these differences. The fact that most upgrade problems occur because of differences between the development environment and the production environment suggests that the appropriate location to detect these problems is not at the developer's site but at the user's site – either in a test bed or during activation. It further suggests that these problems are easiest detected by the product during execution rather than through other means.

Dumitras and Narasimhan [4] provide a list of the most common hidden dependencies. For each of these hidden dependencies we can generate a product requirement to potentially detect the problem and report it to the operations staff. The results for the first three dependencies are shown in Table 1.

Table 1: Hidden dependencies and related product requirement.

Hidden Dependency	Product Requirement
Incorrect file path	The product shall, during initialization, verify that all file path information results in an access to the correct type of file. Failures shall be reported.
Incorrect network address	The product shall, during initialization, verify that all network addresses are accessible. Failures shall be reported.
Library conflicts	The product shall, during initialization, verify that the libraries used by the product are available in a version compatible with the current version of the product. Failures shall be reported.

To check the compatibility of libraries, we envision different implementation options. Depending on the criticality of the product, the requirement might be to have exact versions of libraries available – which can be checked by a list of permitted version numbers for each library. In other cases, the following techniques might be used: (i) in the case of dynamic-link libraries, during initialization, collecting the signatures of all operations used by the product, and making sure the signatures match those of the libraries present; (ii) explicitly stating a range of version numbers required, e.g., at least 3.5 – [3.5, ∞) – or between 3.5 and 4 – [3.5, 4), and the product checks if the library present is compatible with; (iii) using a compatibility repository, such as [13], where the product can check if the version required (e.g., 3.5) is compatible with the one present (e.g., 4.2) – the product could issue a warning if compatibility is at a high percentage, but less than 100%, or an error if the compatibility is below some threshold; or any combination of the above (i) – (iii).

B. Configuration errors

Yin [14] and his colleagues have studied misconfigurations in open source and commercial systems. Configuration correctness is an area where many products have requirements already. Parameters are routinely tested for syntactic correctness and sometimes even semantic correctness – e.g. does the system have appropriate access rights for the necessary files.

Performance is an area, however, highlighted by Yin where the errors are subtle and difficult to detect. This is especially true when the application stack comes from multiple providers.

Yin provides an example where a configuration setting in PHP is inconsistent with a setting in MySQL.

A generalized requirement comes out of Yin's work with respect to performance. Products should monitor their important performance measures and compare the values before and after a configuration modification to determine whether there is significant degradation.

V. Developer Problems

In this section, we describe a requirement that emanated from the standard view of DevOps – the interaction between operations personnel and developers. In this case, the operations personnel are also developers so the communications channels are very short.

Yuruware Bolt[2] is a product that provides disaster recovery and migration services for AWS. We had two problems during the development and execution of Bolt.

1. Unit testing of Bolt was very difficult. Each test requires a collection of cloud resources in certain state before it runs. It was not a negligible cost to set up each environment and to clean up the environment properly upon any test failures.
2. Failure is a common occurrence within the cloud and so any operations process must deal with the possibility that the execution of that process will fail. Unexpected failures occurring in cloud often require operational personnel to clean up the cloud environment properly by hand before restarting the product.

These two problems led us to introduce an "undo" capability for AWS. Implementing such a capability is non-trivial since not every operation in AWS is reversible. For example, if auto scaling is turned on, then reversing the creation of a new instance by deleting that instance will not succeed since the auto scaling rules will result in the instance being restored. For another example, deleting a resource cannot be reversed since the resource is gone and may have been re-allocated.

Our solution involves creating a checkpoint of a consistent state and using artificial intelligence planning as a technique to return to the known state from the state from which an undo is desired. See [12] for more details.

VI. Modelling Process and Product in a Single Model

Products and operations processes impacting these products can be modeled with the same formalism in a single model. This allows determining and verifying requirements in the product that are intended to automate or support steps in the operations process.

As an example, we describe our work in preventing mixed version race conditions. This work is ongoing and so the results we describe here are preliminary.

The mixed version race condition was identified during the 2nd ACM Workshop on Hot Topics in Software Upgrade [5]. It might occur during a rolling upgrade when a client interacts both with version N and version N+1 during a single session. Specifically the sequence shown in Figure 1 can occur.

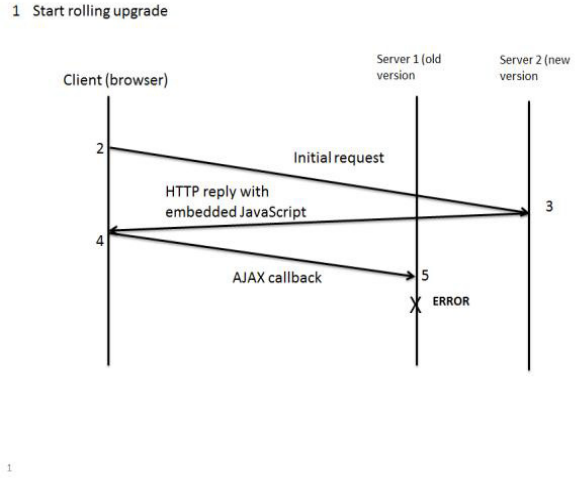

Figure 1: Mixed Version Race Condition

In this sequence, version N and version N+1 of an application are both executing within a cluster. A client is initially routed to an instance of version N+1. This version sends embedded JavaScript reflecting new functionality back to the client which, in turn, uses this new functionality to make another request. The new request is routed to an instance of version N and the race condition occurs.

Intuitively a solution that prevents this problem would make the application version aware, make the client version aware, and make the load balancer version aware. Similar to the cases discussed in the previous sections, this is also a clear example where an operations requirement (e.g., a rolling upgrade with no mixed version race condition) raises a new product requirement and the implementation.

The next question is how to verify the production requirement/implementation satisfies the operations requirement. The difficulty rises from the fact that those two are traditionally considered in separated activities and analysis methods are less studied. To address the issue, we created a Colored Petri Net[3] [10] model that modelled both the interactions of the application with the client (the top portion of Figure 2) and the concurrent execution of the rolling upgrade (the bottom portion of Figure 2). That is, the model contains the product and the process and can be used to understand their interaction as well as to derive specific requirements for the product.

The Petri Net is used to verify the correctness of the solution. Specifically, making the application version aware, making the load balancer version aware, and reflecting that awareness in the client.

[2] Yuruware Bolt— http://www.yuruware.com/

[3] cpntools.org

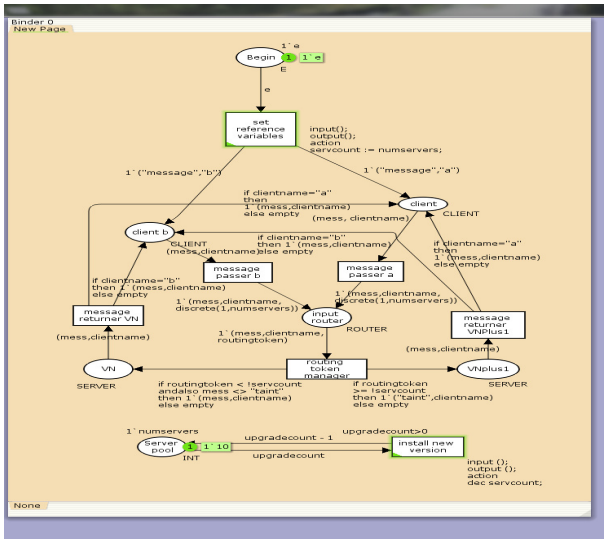

Figure 2: Petri Net model of a technique for prevent the mixed version race condition.

Having a correct solution is only a portion of showing that the solution is effective. Three other issues remain:

1. Do the modifications to the load balancer introduce unacceptable overhead? Preliminary results are that the modifications to the load balancer added no additional overhead.
2. Will the load balancer continue to distribute requests in a balanced fashion? This work is continuing.
3. What will happen if a message reflecting version N+1 is routed to a cluster with no occurrences of an instance that has been upgraded. A Petri Net has been created to reflect this situation but the experimental work is continuing.

Our point for this paper is that treating the product and operational processes involving the product in a uniform and unified fashion allows one to determine additional requirements for the product and to verify that those requirements do in fact improve the process.

VII. Summary

The DevOps community advocates communication between operations personnel and developers as one means for improving operations process. In this paper, we go much further. We claim that operations processes can be improved by making products operations process aware. This awareness is manifested by adding requirements to the product.

Sources of operations requirements include not only the developers but also standards, organizational process descriptions, academic studies of types of operational failures, and treating the product and operation process in one model. For each of these categories we have given examples of how this might work.

This paper only sketched an approach to improving operational processes. The approach needs to be fleshed out and validated with examples of improvements to existing operational processes. Furthermore, the approach we advocate needs to be compared to other approaches to improving operations processes. This is all work to be done.

Acknowledgments

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program

VIII. References

[1] Colville, R, and Spafford, G. Configuration Management for Virtual and Cloud Infrastructures, http://www.rbiassets.com/getfile.ashx/42112626510

[2] Crameri, O., Knezevic, N., Kostic, D., Bianchini, R., Zwaenepoel, W., Staged Deployment in Mirage, an Integrated Software Upgrade Testing and Distribution System, In Symposium on Operating Systems Principles, pages 221–236, Stevenson, WA, Oct 2007.

[3] Debois, P., Devops: A software revolution in the making. http://www.cutter.com/itjournal/fulltext/2011/08/index.html

[4] Dumitras, T., Narasimhan, P. Why do upgrades fail and what can we do about it?: toward dependable, online upgrades in enterprise system. In *Proceedings of the ACM/IFIP/USENIX* Middleware'09,

[5] Dumitras, T., Neamtiu, I., Tilevich, E. Report on the Second ACM Workshop on Hot Topics in Software Upgrades (HotSWUp'09). Proceeding of the 24th ACM

[6] Elisa, S., Bromberger, J., Stunski, P. Migrating AWS Resources to a New Region. http://media.amazonwebservices.com/AWS_Migrate_Resources_To_New_Region.pdf

[7] International Organization for Standardization. ISO/IEC 15504 Information technology - Process assessment

[8] Information Technology Infrastructure Library http://en.wikipedia.org/wiki/Information_Technology_Infrastructure_Library

[9] National Institute of Science and Technology, Recommended Security Controls for Federal Information Systems and Organizations, NIST SP800-53

[10] Petri Nets, http://en.wikipedia.org/wiki/Petri_net

[11] Ruhle, S., Harper, C., Mehta, N. http://www.bloomberg.com/news/2012-08-14/knight-software.html

[12] Weber, I., Wada, H., Fekete, A., Liu, A., Bass, L., Automatic Undo for Cloud Management via AI Planning, Proceedings Usenix Workshop on Hot Topics in System Dependability, 2012.

[13] Upstream Tracker http://www.cnet.com/techtracker-free/

[14] Yin, Z., Ma, X., Zheng, J., Zhou,Y., Bairavasundaram, L., Pasupathy, S. An empirical Study on Configuration Errors in Commercial and Open Source Systems, Proceedings 23rd SIGOPS, 2011