From the Proceedings of the 10th International Software Process Workshop (ISPW-10), Nancy, France June 17–19, 1996

Software in the Field Needs Process Too

Dennis Heimbigner and Alexander L. Wolf
Software Engineering Research Laboratory
Department of Computer Science
University of Colorado
Boulder, CO 80309 USA
{dennis,alw}@cs.colorado.edu

Abstract

A software system spends its life in two, very different worlds: development and operations. Traditional software process technology has focused on supporting the activities associated only with development. We believe that there are significant benefits to be gained by also supporting processes in the field. Furthermore, once support is in place for operation processes, we can begin to consider how to profitably integrate development and operations.

1. Introduction

Software process research has traditionally focused its attention on supporting the activities that occur within development organizations. It is there that we find the design, code review, build, regression testing, change control, and bug tracking processes that have been the mainstay of our challenge examples, prototypes, and successes.

But software sometimes ventures beyond the relatively comfortable confines of the development world and enters the dark, cold, and lonely world of operation in the field. It is there that the software, rent from its nurturing developers, faces the enemy: the real users, not the use-case scenarios; the real uses, not the functional test cases; the real environment, not the experimental laboratory. And, perhaps most significantly, it is there that real maintenance occurs, with rapid ports to unanticipated platforms and quick changes to functionality that one large corporation has taken to calling "field features".

But imagine if instead of facing the enemy alone, the software was able to take some protective gear with it into the field. Perhaps, for example, it could take a configuration management system along to help with unanticipated ports. Perhaps it could even carry a communication mechanism that could be used to report back to the developers on the real conditions encountered in the field so that succeeding versions could be better prepared.

2. The Deployment Gap

We believe there is a need for innovative process technologies that can bridge what we call the *deployment gap*. The deployment gap is the barrier that divides the development activities of requirements analysis, design, code, and test from the operation activities of localization, installation, use, and rapid fix. Integration across the gap would allow new requirements, constraints, and experience to be more accurately and routinely communicated from operations back to development, at the same time as it allows development to nimbly respond with more appropriate updates.

Figure 1 depicts our vision of how the deployment gap might be bridged. The left side of the figure shows that development involves a cadre of *personnel* using a set of well-defined and articulated *processes* for carrying out the organization's responsibilities by making use of an *environment* of tools and data. The primary tasks for development are to methodically plan and perform evolutionary changes to the gross functionality and performance of software systems, and to carefully test and then package the software for deployment. The right side of the figure shows that operations has a corresponding structure of personnel, processes, and tools. The primary tasks of operations are to localize, install, and operate the software. In addition, they monitor the system and interact with users with an eye toward understanding and documenting improvements that can be made to the system; this is depicted in the figure by the way operations cradles the operational system within its (conceptual) boundaries. Operations has the added chore of having to make rapid fixes in the field to the software, such as developing workarounds for critical failures and performing unanticipated ports to new hardware.

Figure 1 shows the deployment gap that exists between development and operations, but also the existence of var-

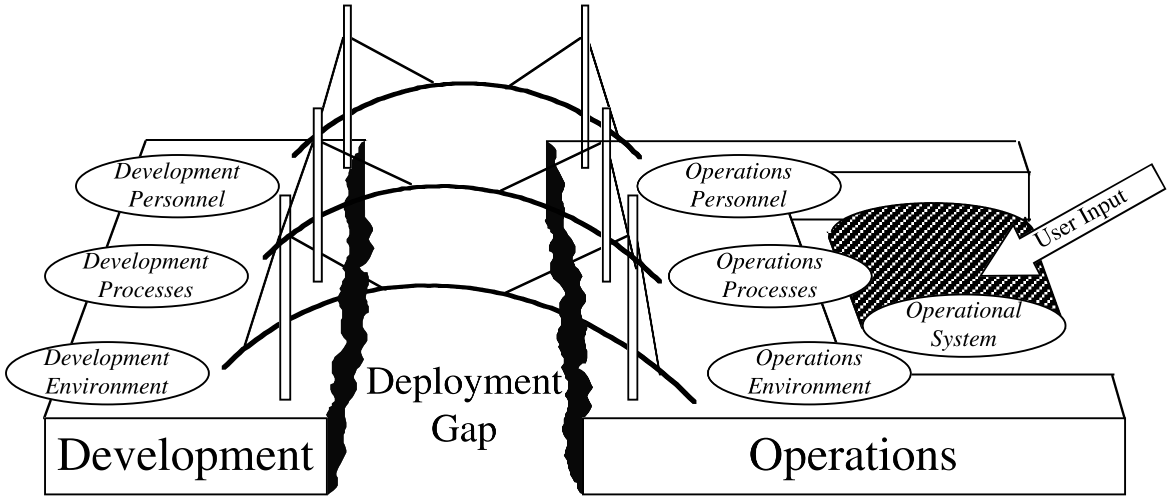

Figure 1. A Vision for Bridging the Deployment Gap.

ious bridges across this gap. The bridges serve to increase commonality, communication, and coordination among the personnel, processes, and tools constituting the development and operations organizations. Bridging the gap in a secure, flexible, and controlled manner will provide three, high-leverage improvements to the state of the art in software evolution.

1. *Cost Reduction through Reuse of Common Infrastructure.* Many of the tasks performed by development organizations overlap in nature and complexity with the tasks performed by the organizations that operate deployed systems. Both must, for example, manage multiple, simultaneous configurations, but only development organizations typically use sophisticated configuration management technologies. Sharing infrastructure across the deployment gap should create significant improvements in software management quality and cost.

2. *Nimble Evolution through Effective Communication.* The separation of users from providers seriously impedes software evolution. Expediting, enriching, and automating the communication across the deployment gap allows new requirements, constraints, and experience to be more accurately and routinely migrated from operations back to development, and helps development to nimbly respond with more appropriate updates.

3. *Coordinated Activities through Integrated Processes.* Common infrastructure and effective communication enable more intimate coordination of the development and operation activities that evolve software systems. Coordination supports more reasoned partitioning of evolution tasks, rapid incorporation of field fixes into the development cycle, and traceability of evolution paths wherever they lead. This coordination is best accomplished by integrating the development and operation organizations' processes into a flexible federation that solidifies the bridge between development and operation.

These leverage points provide significant benefits to numerous activities of the software enterprise, including requirements validation, configuration management, testing, and performance optimization.

3. Examples

In this section we consider several examples where it would be beneficial to have process technology applied to software in the field.

Requirements Validation Requirements analysis is usually seen as an upstream, or early development, process. Customers and sometimes even users engage in a surrealistic attempt to understand and evaluate a system and its environment before it has been created. If we could continue the requirements analysis process after the software has been fielded, then we could use the software itself to validate the requirements. In fact, requirements could be viewed as more than simply assumptions, but as constraints that are continually evaluated and whose violation are automatically reported back to development.

Perpetual Testing Testing is a process that historically ends when a system is placed in the field. Typically, this occurs before the system is considered to have been "fully" tested, due to the inevitable schedule contraction that occurs at the end of a development cycle. If, however, the testing process—and the associated testing environment—were to be delivered along with the system, then testing could continue in the field. Moreover, operational software can encounter situations never anticipated by testers when developing test cases. If these situations could be automatically incorporated into test sets and communicated back to development, then succeeding versions of the software could emerge better, and perhaps more realistically, tested.

Post-deployment Configuration Management Post-deployment configuration management is the process of taking a delivered, but partially instantiated, system and configuring it in the field [1]. If we could apply the full power of existing pre-deployment (i.e., development) configuration management processes, we could provide the following capabilities.

- *Localization*—the ability to alter various parameters of the operational system. Note that this may include, for example, wholesale replacement of some underlying component, such as a database system.
- *Rebinding*—the ability alter the binding of components to executable processes when the actual context makes the existing binding infeasible.
- *Backdown*—the ability to revert to a previous version of a system when a new version proves unusable.

4. Acknowledgments

The ideas contained in this position paper were formed in part with the collaboration of Leon Osterweil and Lori Clarke. Perpetual testing, in particular, is due to Leon Osterweil.

This work was supported in part by the National Science Foundation under grant CCR-93-02739 and by the Air Force Material Command, Rome Laboratory, and the Advanced Research Projects Agency under Contract Number F30602-94-C-0253. The content of the information does not necessarily reflect the position or the policy of the U.S. Government and no official endorsement should be inferred.

References

[1] D. Heimbigner and A. Wolf. Post-Deployment Configuration Management. In *Proceedings of the Sixth International Workshop on Software Configuration Management*. Association for Computer Machinery, Apr. 1995. To appear.