From the Proceedings of the 5th Inter. Conf. on Software Configuration Management, LNCS 1005, Springer-Verlag, Berlin, 1995

Does Configuration Management Research Have a Future?

André van der Hoek, Dennis Heimbigner, and Alexander L. Wolf

Department of Computer Science, CB 430
University of Colorado
Boulder, Colorado 80309 USA
{andre,dennis,alw}@cs.colorado.edu

Abstract. In this position paper we raise the question of whether Configuration Management (CM) research has a future. The new standard in CM systems—typified by commercial products such as Adele, ADC, ClearCase, Continuus/CM, and CCC/Harvest—largely satisfies the CM functionality requirements posed by Dart. This implies that research in the area of CM is either unnecessary or that we must find new challenges in CM on which to focus. We believe that these challenges indeed exist. Here we present some areas that we feel are good opportunities for new or continued CM research, and therefore conclude that CM research *does* have a future.

Introduction

Numerous attempts have been made to lay out the overall functionality requirements for CM systems. Dart [1, 2] defines eight areas.

- **Component**: identifies, classifies, and accesses the components of the software product.
- **Structure**: represents the system model of the product.
- **Construction**: supports the construction of the product and its artifacts.
- **Auditing**: keeps an audit trail of the product and its process.
- **Accounting**: gathers statistics about the product and the process.
- **Controlling**: controls how and when changes are made.
- **Process**: supports the management of how the product evolves.
- **Team**: enables a project team to develop and maintain a family of products.

The first generation of widely-used CM systems—namely SCCS, RCS, Build, and Make—clearly do not meet these requirements. After a decade of research, a second generation has emerged as the new standard in CM systems. Commercial products such as Adele, ADC, ClearCase, Continuus/CM, and CCC/Harvest,

This material is based upon work sponsored 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 Government and no official endorsement should be inferred.

which represent a coalescing and distillation of many research and commercial prototypes, largely satisfy the CM functionality requirements posed by Dart.

Of course, each of the second-generation CM systems mentioned above has its strengths and weaknesses. All can still be expected to improve over time. But their general coverage of Dart's eight functionality requirements raises the important question of whether there is a future to CM *research* as opposed to CM *engineering*. The answer is certainly "no" if research continues to stay focused on the eight areas. While there are engineering improvements to be made in those areas, research must turn its attention to new challenges.

In the following sections we propose some areas that we believe are the future of CM research. Exploration of these areas should eventually lead to new capabilities, and a new generation, of CM systems.

CM System Architecture

The second generation of CM systems is based on a three-level model: a low level that provides data storage, a middle level that provides basic CM services or mechanisms, and a high level that consists of processes or policies for using the mechanisms in the middle level.

At this point, CM systems allow some restricted flexibility at the low level (e.g., one can choose to use RCS, the file system, or a DBMS), and even less flexibility at the middle level (e.g., the naming and locking mechanisms are usually fixed). At the high level of process, second-generation CM systems either provide no explicit support for expressing policies or they provide particular processes for a specific task, such as change control. (Adele is a notable exception to this.)

Two important research questions arise. First, is it possible to provide more flexibility at the various levels and, second, is there even a better CM system architecture than the three-level model that might, for example, allow for this greater flexibility? The three-level model implies that the primitives of CM are data storage, basic mechanisms, and specific policies. An alternative view is a set of cooperating services, such as general relationship maintenance, dependency reasoning, and artifact access brokering, as well as process specification, monitoring, and control. Such alternative architectural views might lead to novel solutions in other areas of CM.

Another much needed topic of research is interoperability among CM systems. More and more, companies are starting to jointly develop products. Clearly, the need arises for companies to share CM artifacts. Yet each company will continue to use its own CM system to preserve its investment (of money, training, and trust) in that system. Therefore, it must become possible to form (temporary) alliances among different CM systems. This is a problem at all levels of the CM system architecture, from sharing of data to integration of processes. Notice that security issues arise as well in this context: it can be assumed that companies will not be willing to share all their CM artifacts. Again, an interesting question is whether the three-level model helps or hinders in solving the interoperability problem or whether there is an alternative architecture that is better suited to the solution.

Product Representation

As a rule, current CM systems represent the relationship among product versions through a directed acyclic graph (DAG), where the nodes are the version objects and the edges represent the *is-version-of* relationship. Implicitly, the edges also represent the deltas among versions. An interesting question is whether there are additional graphs—or, more generally, views of the available information—that might be useful.

The change set approach (as found in ADC, for example) provides an alternative view over similar information. A product is represented by an initial instance of a product combined with a constellation of change sets, which are sets of deltas to be applied to the product as a whole. Using change sets, one is able to do such things as apply a single bug fix to multiple branches. This is not possible, strictly speaking, with the traditional DAG model, and therefore the change set notion clearly adds useful functionality.

Software process provides examples of relationships that would be useful in an extended CM system. A software product actually consists of a rich web of related artifacts such as requirements, designs, and test cases, as well as the standard code artifacts. A CM system should be able to model relationships among related versions of these varied artifacts. In addition, it should be possible to tag relationships with process activities. For example, if a new version of some code has been created, then it is possible that the corresponding test cases also need to have new versions. A CM system should be able to traverse links to locate those test cases and interact with a process execution system to put the creation of new versions of test cases into a task agenda.

Product Software Architecture

The relationship between software architecture and CM systems has not been explored in any depth. However, knowledge of a product's architecture might have very interesting implications for a CM system. To date, the system model is the closest that CM systems come to representing product architectures. But system models primarily deal with the *building* of executable object modules, including the selection of versions. They do not represent other important architectural information, such as data or control flow connections among the modules.

Giving CM systems greater knowledge about the architecture of the software that they are managing would allow them to provide useful capabilities. For example, a geographically distributed effort to develop a certain product might apportion responsibility for different architectural pieces of the system to different sites in order to allow for parallel development. With the current generation of CM technology, each site must be assigned its own development branch. This implies that later in the development process an extensive, and possibly error-prone merge must take place to give the various sites consistent views of the entire system. In contrast, by using knowledge of a product's architecture, a CM system can make decisions about distributing artifacts based on, for example, module interconnections.

Domain Extensions

The capabilities provided by sophisticated CM systems should be applicable to domains outside of software development. Systems engineering is an example of such a domain. Systems engineering is concerned with products that are combinations of both software and hardware. Conceptually, second-generation CM systems already should be capable of maintaining hardware configurations. Of course, there is the problem of controlling and monitoring changes to hardware, which is not an issue for software (i.e., since software is intrinsically on the computer, unlike hardware, changes are immediately evident to the CM system). But ignoring that problem, there is still a serious question about how to represent and version the relationships *among* hardware and software.

The post-deployment phase for software provides another interesting domain ripe for configuration management research. There is a wide variety of problems involved in distributing software into the field and then maintaining it once there. Dynamically reconfigurable systems provide an example. For such systems, only part of a product may need to be upgraded, while the rest of the product should continue running. Suppose the installation of a product consists of a collection of client and server programs, where a certain client or server needs to be replaced by a new version. The CM system should offer support to decide whether the new client or server is compatible with the other clients and servers in the environment. Also, the CM system should indicate which clients and which servers are dependent on each other. This can determine the extent to which part of the whole product needs to be brought down in order to let the upgrade succeed.

Conclusion

Configuration management systems have come a long way since SCCS and Make. Second-generation CM systems have raised the level of sophistication dramatically, creating much greater challenges to CM researchers. Many problems and opportunities remain and, indeed, the second-generation systems themselves raise new questions. Configuration system architecture, product representation, product software architecture, and extended application domains are just a few of the areas that should keep researchers busy for the next several years.

References

1. S. Dart. Spectrum of Functionality in Configuration Management Systems. Technical Report SEI-90-TR-11, Software Engineering Institute, Pittsburgh, Pennsylvania, December 1990.
2. S. Dart. Concepts in Configuration Management Systems. In *Proceedings of the Third International Workshop on Software Configuration Management*, pages 1–18. ACM SIGSOFT, 1991.