Impact of the Research Community On the Field of Software Configuration Management

Summary of an Impact Project Report

**Jacky Estublier (Univ. Grenoble), editor[1], David Leblang, co-editor,
Geoff Clemm (Rational), Reidar Conradi (NTNU, Norway),
Walter Tichy (Univ. Karlsruhe), André van der Hoek (Univ. California at Irvine),
Darcy Wiborg-Weber (Telelogic, Irvine)**

Abstract

Software Configuration Management (SCM) is an important discipline in professional software development and maintenance. The importance of SCM has increased as programs have become larger and more complex and mission/life-critical. This paper discusses the evolution of SCM technology from the early days of software development to present and the impact university and industrial research has had along the way. It also includes a survey of the industrial state-of-the-practice and research directions.

The paper published here is not intended to be a definitive assessment. Rather, our intention is to solicit comments and corrections from the community to help refine the work. If you would like to provide further information, please contact the first author. A longer version of this report can be found at *http://www-adele.imag.fr/SCMImpact.pdf.*

Keywords

Software configuration management, software engineering, software quality, industrial impact.

1 PREFACE: WHAT IS IMPACT?

During the preparation of this report, there was a lively debate among the authors about what defines impact and which research to include. The impact is easy to discern: A successful, multi-million dollar industry in software configuration management tools has arisen. SCM tools have become so pervasive that virtually all major projects use them. SCM provides tangible and recognized benefits for software professionals and managers; software development standards even prescribe their use. There are now over a dozen textbooks dealing exclusively with SCM, and most textbooks on software engineering include a chapter on that topic. Software engineering conferences and workshops regularly seek contributions in this area. Finally, there is a lively international community of SCM researchers and an international SCM workshop series.

Identifying the research that had caused this impact was more difficult. We (the authors) came quickly to the conclusion that citing only academic research would be far too narrow, because corporate research had contributed a great deal of results and ideas. In fact, the software configuration management community owes much of its liveliness to a healthy and competitive mix of researchers and developers from both academia and industry. Our first ground rule was therefore to take an inclusive view of research.

We discovered quickly that it was futile to determine who contributed "more", academia or industry. Our opinions about what was more important diverged widely, depending on our personal points of view. We will therefore leave the evaluation of the relative merits of research contributions to our readers and to historians. Our goal is to provide an honest and accurate picture of the major research ideas in SCM and show where they had an impact.

Given the long lead time for research results to show up in practice, some as yet unused results may have their impact still ahead of them. Thus, we decided to discuss current research even if it has not yet had any discernible impact on practice.

So far, these three ground rules, though important, do not help identify relevant research. After some debate we decided to concentrate on publications in the open literature. This criterion is not perfect, since it leaves out unpublished work, in particular results that go directly into products. However, fortunately for the community, research results were vigorously published even by industry, notably Bell Labs, Apollo, Atria, and others. We are fairly certain that the major research ideas and results in the domain of SCM have actually been published.

A major impact is caused by people, in particular university graduates and researchers in SCM, moving to industry. Many of the academics in the SCM field have enjoyed long and fruitful relationships with the SCM industry. Industrial contacts are a source for new problems to solve and act as a corrective of what is important and what is not.

Last but not least, workshops and conferences have had a significant impact on the SCM community. Given the competitive nature of the software business, it has fallen to the academics to organize vendor-independent meetings in which the latest results in SCM are presented and new, unpublished ideas discussed. It is unlikely that even the authors of this report would all have met without the SCM workshops that brought together researchers and developers in the SCM field for over a decade.

In sum, we hope that this paper gives a flavor of the software

[1] Addresses of authors: Jacky.Estublier@imag.fr, Leblang@alum.mit.edu, GClemm@rational.com, Reidar.Conradi@idi.ntnu.no, Tichy@ira.uka.de, andre@ics.uci.edu, darcy@telelogic.com

configuration management story, a success story that is yet unfolding. We hope that many more chapters will be added to it in the future.

Section 2 describes SCM and how the discipline evolved since the early times. Section 3 summarises the impact of research work. Chapter 4 concludes the report.

2 STATE OF PRACTICE AND EVOLUTION

Configuration management (CM) is the discipline of controlling changes in large and complex systems. Its goal is to prevent the chaos caused by the numerous corrections, extensions, and adaptations that are applied to any large system over its lifetime. The objective of CM is to ensure a systematic and traceable development process, so that at all times a system is in a well-defined state with accurate specifications and verified quality attributes.

CM was first developed in the aerospace industry in the 1950s, when production of spacecraft experienced difficulties caused by inadequately documented engineering changes. Software Configuration Management (SCM) is CM tailored to systems, or portions of systems, that consist predominantly of software [Whi91].

At first, different colored punch cards were used to indicate changes; fortunately, software became an "on-line entity" and hence could easily be placed under automatic, programmed control. Currently, SCM systems are sophisticated tools, which emphasize process support, concurrent engineering control, distributed and remote workspace control and other high-level functions.

Thus, SCM has evolved significantly since its introduction, but a number of characteristics remained constant. SCM is a technology in charge of most of the issues that are raised by large software product development and maintenance, and not addressed by programming languages. SCM does not typically manage early life-cycle phase objects (requirements, design, analysis, etc.) or deal with final stage issues (deployment, dynamic evolution and reconfiguration). However, some SCM systems maintain the relationships between these "other phase" objects and source code.

2.1 Files

Originally, SCM's duty was to manage the many files involved in a software product, their *versions* and the *building* of the system out of these files. Fundamentally this has not changed.

Currently, there is a tendency to hide the concept of file, found to be too low level, at the benefit of concepts closer to actual programming practices, as found in programming environments (project, folder, package, etc.). Nevertheless, the file system did not disappear and software engineering tools still rely on this file system view, which remains, for the foreseeable future, the underlying structure.

The consequence is that the main structure of a system, as perceived by SCM tools, is the file system structure, and a basic SCM function is to be able to (re)generate a file system view of a software system so that desktop tools and development environments can work.

2.2 Version Control

To avoid the confusion resulting from the same identifier assigned to two versions of the same item, a new and unique identifier is issued whenever an item is changed. But the proliferation of items requires a system to manage them, recording their relationships and their common properties. This is what version control is about.

Originally, each file was individually managed as a tree of versions (revisions and branches) following the SCCS [Roc75] and RCS [Tic82] systems. These systems are still at the core of almost all commercial SCM systems. Over time, SCM systems have come to manage *configurations,* which are a "consistent" and complete collection of file versions.

Change packet versioning came from industrial practices in the 70s (IBM and CDC update systems), extended by Hough [Hou81]. The idea relies basically on conditional compilation technique;[1] at check-in all "bundled" changes are tagged with the same label, which allows one to later (re)build files, given an arbitrary list of labels. Thus it becomes possible to build files that never existed before, by combining changes performed independently, maybe concurrently. This technique was introduced to the market under the name *change set* [ADC88]

Change sets open possibilities that go far beyond classic versioning, and thus were heavily studied by academia. In the Norwegian EPOS project, under the name of *change-oriented versioning* [Lie89] [Con97], version fragments (deltas) are tagged by boolean attributes called *options*. In this system it becomes possible to manage the evolution not only of files, but of any data structure, any kind of attribute or relationship, and to make explicit relationship between options (compatibilities, incompatibilities, requirements). Boolean expressions containing options determine the visibility of a logical change [Wes01]. The product structure (called product space) becomes completely independent from the versioning structure (called version space).

Recently, Zeller and Snelting [Zel97] have explored similar approaches in their ICE prototype, using *feature logic* to express the version rules. Many attempts have been made at offering fully versioned databases including Damokles OODBMS [Dit87] and ObjectStore [Lam91].

Pure change-set systems do not work well in practice for several reasons. First, labels (options, deltas) sometimes overlap and conflict. Although the system can physically construct any combination, some combinations may not compile. For some binary formats, there is no way to sensibly combine any deltas except with the version on which they were based. Second, in a software project with tens of thousands of components and hundreds of thousands of changes, it is too difficult to reliably make a system by naming change sets. In fact, only a tiny percentage of change combinations are actually useful.

Commercial change-set systems like TrueChange [McC00] and Bellcore's (in-house) Asgard [Mic96] system make heavy and frequent use of reference product versions, called baselines, to define a starting point for new changes. These baselines are usually tested and stable. A developer need only specify a baseline and a few additional changes. Although this reduces the number of

[1] Conditional compilation is well represented by cpp, the C preprocessor. Cpp analyses special constructs *#if label* and *#endif*, and if *label* is defined, the text between *#if* and the corresponding *#endif* is left in the file, otherwise it is removed.

possible permutations of changes it vastly simplifies version selection and increases confidence in the result.

It is expensive and error prone to build and maintain special software for each customer, even with change sets. The trend has been to build a single version whose special features can be enabled dynamically (usually for a fee).

In the mean time, it was understood that a special use of the merging technique could provide a subset of the facilities found in the change-set approaches. Called *change packages* [Dar97], this technique has the big advantage that is uses the standard versioning technology, which is mature and efficient, and can allow for occasional change selection in an hybrid approach. This is why this technology has increasing acceptance in current SCM tools.

Although change sets could be used to uniformly implement conditional compilation as well as revisions, in practice, users view conditional compilation as a programming language issue distinct from ordinary revisions. In general, software development organizations now try to reduce the number and variety of releases to save costs and increase quality. In addition, organizations are more concerned with change management and process control than advanced versioning models.

Nevertheless, many commercial SCM systems have added the ability to track logical changes rather than individual file changes. But the name and features are not always similar. ADC/Pro uses the term "change set", CCC/Harvest uses "package", CM Synergy uses "task", ClearGuide uses "activity", PCMS uses "work package", and StarTeam uses "subproject". Even lower-end SCM tools such as Intersolv's PVCS and StarBase's StarTeam have the ability to mark a source-code change with the corresponding defect report or enhancement request. Within the next years, we expect that tracking software by units of logical change will be the state of the practice for most commercial software configuration [Dar97] regardless of the underlying versioning technique.

The driving force behind the acceptance of logical change tracking may be the desire to manage changes in a process-oriented way, not as a version selection mechanism.

Only recently, with the advent of task-based approaches, versioning evolved again, integrating change-set features, but on an RCS-based internal representation.

2.3 Product Modeling

Most SCM systems maintain only the version trees, the configuration (a lists of version identifiers) and a makefile, for the compile-time dependencies.

Recently, for their internal representation of a software product, high-end SCM systems support features that add semantics to items. They allow types, attributes, and relationships to be associated with items according to a *data model.*

Consequently, an SCM system needs to translate (fast) between the file system view and the internal representations of the same software product. This is a limiting factor, which explains why even high-end commercial tools use simple data models close to the file system view.

2.4 System Model and Rebuild

As software systems have grown to include many thousands of components, keeping track of the pieces and how they fit together has become a difficult but critically important task. SCM systems accomplish this task with *system modeling* facilities. System modeling was originally used to help rebuild a system from source files (Make [Fel79]), and even now, in most SCM systems, the makefile is the only "system model".

A rich system model also provides a high-level view of the product organisation (its architecture), which can be used to better control product evolution, to compute "consistent" configurations and so on.

Unfortunately, any information that has to be provided and maintained by hand represents a (high) cost that may rapidly outweigh its expected benefit. For this reason, in the current state of practice, system models are not widely adopted by practitioners. High-level system models will be used only if they can be extracted automatically from source code, or if they are available, for other reasons.

2.5 Rebuild and Selection

With so many components and versions, a *configuration selection* mechanism is needed to select which versions to include in a new configuration.

The number of possible configurations far exceeds the number of useful configurations. The general problem of determining the "consistent" combinations would require some semantic knowledge of all artifacts. This is beyond the state of the art. In fact, engineers often are unable to determine if a configuration is correct without building and testing it.

Of course, SCM tools do not address this issue in its generality. Instead, a new configuration is built starting from a consistent reference configuration called a *baseline*. The baseline is used to create a *workspace* in which a few changes are performed. Once tested, the changes are "checked-in" and a new baseline configuration is created. This simple schema is the basis of almost all SCM systems, with extensions for concurrent engineering support (see below).

2.6 Workspace and Concurrent Engineering

Workspaces are "naturally" independent areas where users can perform their job isolated (or insulated) from the activities of other colleagues. This is a desirable property, especially in large projects.

Eventually, the work performed, simultaneously or not, in different workspaces must be combined (merged). Thus, high-end systems propose features, like rules, to flexibly combine the work performed in different workspaces. Sometimes the work can be combined through the careful selection of versions in a new configuration. Other times the physical file changes must be merged. Since the late 1980s, this is one of the important research topics.

2.7 Change Control

From the beginning, auditing and tracking have been considered as basic functions of SCM systems. They require that any change performed on a large software system be related with a formal change request. Originally, a per-file change log kept information about changes, but over time change control has evolved into the set of procedures governing how a change is decided, implemented, tested and released. Most current SCM systems

support change control based on a set of states assigned to software items or change request items. It is a "product based" process control.

More recently, in high-end systems, workspaces were considered as a unit of work with sub-workspaces corresponding to sub-activities. Thus a workspace becomes the context for *task* or *activity* concepts. Therefore, in these systems, change control can be expressed in terms of tasks assigned to users in specific workspaces; it is an "activity based" process control.

Process support is a generalisation of change control applied to other activities. Activities are currently assimilated to workspaces so process support in SCM is limited to activities performed in a workspace.

2.8 High-End Versioning and Task Support

Because tasks and workspaces are strongly related, tasks and version selection are also related.

Usually, a workspace is created to perform a task (often a change request) on a baseline. The workspace is populated by "checking-out" the baseline, and the resulting change performed is a change set. Using change sets for version selection can be a way to unify the SCM concepts of file management, version control, and workspace management with the process of change control.

Interest in integrating SCM and change control has grown over time and the field of study is still maturing.

2.9 Distributed and Remote Development

Distributed SCM coordinates the work of geographically distributed teams. Distribution requires that the main SCM functions introduced above are network enabled. In local-area networks, client/server implementations of these functions will do, but in wide-area networks the bandwidth between sites may not be sufficient. In this case, each site, even mobile, needs to replicate the relevant parts of the database and a periodic update process must synchronize them [All95]. Updates that travel over public networks should be encrypted.

Distributed development places a strong emphasis on process support. If team members rarely meet, telephone calls are difficult because of time-zone differences, and email round-trips take a day, informal arrangements on who works on what for how long break down. In this situation, much more emphasis must be placed on automated support for scheduling, tracking work, and preventing information loss.

Web technology and the generalization of the virtual enterprise increase the need for a good support for remote development. Remote development is an important area of work for vendors.

Web site management, and web enabled cooperative work became a new area where some SCM techniques are used. However, web-content management is more concerned with maintaining hyperlinks, page templates, and pooling graphic images than it is with recreation of old releases, frequent baselines, and long audit trails. This is why companies other than the SCM vendors are the actual leaders in the "content management" market[1].

Web technology also addressed issues related to cooperative work on common files. Not surprisingly, some SCM vendors had a definitive influence in the definition of this norm.

2.10 Evolution in Response to the Market

Although current tools build on past inventions, changes in the hardware, software, and business environment have made the evolutionary path of SCM far from a straight line.

Throughout the 1980s debates raged on the best type of delta storage and retrieval mechanisms. New tools were built extending the groundwork laid by earlier tools as rapid progress was made. However, in the 1990s non-textual objects became more common and totally new algorithms were required. By 2000, disk storage became so inexpensive, CPUs so fast, and non-text objects so common that deltas became unimportant – many new tools use simple (zip-like) compression.

Feature selection versus branching as a way to build customized configurations of a product was a popular subject in the 1980s and 1990s. However, in recent years the business environment has made building custom software releases much less popular. Even the creation of "patches", a mainstay of the past, has largely been replaced by annual "service updates".

Development environments like Visual Studio have taken over Make responsibilities and also some system modeling features, so new tools rarely include proprietary build features.

These changes do not mean that SCM research is dead, but rather that it needs to solve new problems more than it needs to improve on solutions to old problems. Some of those new problems include dealing with teams collaborating over the internet, allowing software developers to take their work home at night or on an airplane by docking and undocking, dealing with web content, integrating with other development tools, and providing a GUI that is easy to use but still provides the needed power.

The success of SCM is also due to the fact that SCM systems have been carefully designed to be independent from programming languages and application semantics, and to take the best of simple algorithms (like the version tree) and simple heuristics (like line merging). This turns SCM systems into general and efficient tools, while avoiding the intrinsic complexity of syntactic and semantic issues. Instead, SCM systems seek to integrate the tools that need syntactic or semantic knowledge (such as syntactic editors).

2.11 Summary

SCM is one of the most successful branches of software engineering. There is a lively international research community working on SCM and a billion dollar commercial industry has developed. Nearly all corporate and government software projects use some SCM tool.

SCM is actually unanimously considered as an essential tool for the success of any software development project. It is required to get the second CMM maturity level. Practitioners consider SCM tools as helpful, mature and stable. Consequently, the SCM market is increasing pretty fast (see table 1).

The success of SCM can also be measured by the fact its basic techniques became pervasive and found in many tools. All 4GL tools, most programming environment tools, and even text editors now include a basic version manager. New domains such as "web content management" also apply SCM techniques, as does the new web protocol WebDAV/DeltaV.

[1]BroadVision, Vignette, ATG, Allaire, InterWorld, Interwoven, Blue Martini, Open Market, BSCW.

Company (product)	Annual Revenue $M	Share %	Annaul Growth %
Rational (Atria ClearCase)	293	32.4	50.3
MERANT (Intersolv PVCS & Harvest CCC)	115	12.6	14.9
Computer Associates (Endevor)	113	12.5	5.2
SERENA (ChangeMan)	94	10.4	38.2
Telelogics (Continuus)	65	7.1	23.1
Microsoft (SourceSafe)	31	3.4	2.3
Total (with Others)	906	100	22.7

Table 1: Worldwide SCM Tools ($M) [IDC 2000]

3 SCM IMPACT

In this section, we provide an overall view of the evolution and impact of SCM research.

3.1 Some Successful Transitions

SCM is unanimously considered an essential tool for the success of any software development project. Practitioners consider SCM tools as helpful, mature, and stable. The SCM market has increased in value year after year (see Table 1). Everyday tools such as Microsoft Word now incorporate basic versioning capabilities. All in all, this level of success is remarkable. It is interesting to correlate this success to the key research contributions that have made the transition into industrial SCM products. In particular, industrial products all have as their cornerstones the following four, well-researched issues: versioning, and merges; selection; process support; and distributed and remote development.

3.1.1 Versioning and selection

Change sets, for long a curiosity, are slowly becoming a standard feature in high-end SCM systems, but in a simplified form, often associated with the concept of task/activity. Indeed, focusing on the concept of activity has two major consequences. First, the grain for a version is the complete workspace, thus "naturally" a change set. Second, a selection is based on a known release, and a version is a change with respect to that release. This explains why the simple change-set approach (in its change-package implementation) is gaining acceptance, and why the advanced change-set approach, which does not fit practice, is not used.

Similarly, advanced selection features made possible by advanced versioning, are not fully used because they are not required by actual practice, nor by actual product models, nor promoted by actual tool interfaces. Nevertheless, current evolution goes toward hiding low-level mechanisms (revisions and branches) and relying on higher-level concepts (workspace, task) making implicit use of change sets, and requiring more and more of the features experimented by researchers.

The basic versioning schema (revisions and branches) initially provided by SCCS [Roc75] and improved by RCS [Tic82][Tic85] and still used today in most commercial SCM systems, have been provided by researchers.

3.1.2 Differences, compression and merges

Initial SCM systems relied on traditional differencing and merging technology that involved comparing sequences of lines of ASCII text [Hunt76][Meyers86]. These algorithms were not invented within the SCM domain, but SCM research has since improved the simple line comparison algorithm. These improvements were twofold: incorporating more semantics to provide more accurate differencing and merging technology, and extending the differencing and merging algorithms to also handle binary artifacts.

Context- [Knuth84], operation-, syntactic- and semantic-based comparisons [Reps88] have been proposed to make differencing and merging algorithms more accurate [Buff95]. Syntactic and semantic comparisons seek to find the "relevant" differences and to ensure a merge will produce a consistent result (i.e., a source file that will compile and exhibit the intended behavior). It is easy to prove that a syntactic and semantic merge can avoid errors produced by classic line-based mergers [Hor89]. Despite this clear advantage, commercial SCM systems have not adopted these kinds of algorithms: the need to remain neutral with respect to which kinds of artifacts are versioned prevents the incorporation of semantic differencing and merging techniques. Nonetheless, this research has made a transition into industry in a different domain, where it has been used for several years now: version-sensitive editors ("multi-version editors") [Kruskal84] [Fraser87] [Sar88] [Atkin98].

The classic "diff3" algorithm performs well for textual files, but cannot handle binary files (a Word document is stored as a binary file!). This has prevented use of diff3 for storage optimization in SCM systems that must handle both source and binary files. SCM research addressed this problem from two angles: new algorithms to detect block moves [Tichy84], and new algorithms to operate on binary files [Rei91]. Bdiff [Hunt96] and Vdelta [Korn95] integrated these algorithms into a single algorithm that can detect block moves and handle binary files. The result is a differencing and merging algorithm that not only is extremely efficient, but also serves to compress any kind of file stored in an SCM system. These important benefits explain why commercial SCM products are starting to incorporate these algorithms. Clearly, industrial and academic research has had its impact in this area.

3.1.3 Process support

Over the years, SCM research focused on incorporating process support in the tools. Early SCM tools only provided basic mechanisms to version artifacts. Users would use attributes to label the artifact with the name of the particular life-cycle phase in which it resides (initial, test, release) or with a state. Clearly, improvements were needed to incorporate process support into an SCM system.

The first logical evolution was to incorporate state transition diagrams to model, control and automate the succession of states for artifacts, using a triggering mechanism as the process engine [Bel87][Bel91]. Most SCM systems use this technology now. The next step, currently under way, was to use "activity based" approaches, borrowed from the software process community research [Din98] [Fin94] [Pro98]. The goal was to provide powerful modeling capabilities and associated engines so that customers define and control their own SCM processes [Est97].

The incorporation of powerful process engines in SCM systems did not succeed, because users found it too cumbersome and difficult to define their own processes properly. Instead, process

support now is based on predefined processes developed by SCM vendors. Hiding many of the details of this process behind a powerful user interface, it became acceptable for users to start using and enforcing these processes. An example is ClearGuide, which is a powerful generic process engine integrated with ClearCase. Nevertheless most ClearCase users prefer the "simple" UCM, which provides predefined processes said to embody best practices. Even though users can customize processes, most users choose one of the out-of-the-box processes to institute for their organization. Progress both in customer maturity and in process support concepts and interfaces will be required before generic process support tools will be widely accepted.

Clearly, academic research had its impact in this domain: the issue of process support was raised, example solutions were created, and the fundamental basics were laid. However, it was not until industrial research took the ideas and made them viable for real-world use that software process grew into a mature part of standard, repeatable SCM practice.

SCM is one of the very few fields were process support proved to be critical; it became one of the major vendor selling arguments, and one of the major client expectations [Con98].

3.1.4 Distributed and remote development

Distributed and remote development represents an area in which industrial research clearly took the lead in solving the problem. Initial academic approaches focused on rather complex distributed and replicated repositories. There was also work on adding a simple web interface to an SCM system to provide remote access to a repository with artifacts. However, it was ClearCase that researched and developed MultiSite, a solution that relies on peer-to-peer repositories that are periodically synchronized with each other. Similar solutions are now in place in almost every high-end CM system and research on the topic has quieted down. An adequate solution seems to be in place that resulted from well thought out industrial research, as well as background in academic research and timestamp techniques used in database field.

3.1.5 Web

It is interesting to note that the web provides a different kind of success story. First, web-site management resembles software development closely: rapidly changing resources that are authored and managed by a variety of people and need releases that must be closely controlled. Although it is curious that none of the SCM vendors is among the current market leaders of content management tools (such as BroadVision, Vignette, ATG, Allaire, InterWorld, Interwoven, BSCW, and DreamWeaver), it is not surprising that all of these tools incorporate basic SCM techniques to manage the evolution of web sites. While using a different data model, the basic principles and techniques that they use are still the same and were adopted straight from existing SCM systems.

WebDAV and DeltaV provide another success story in the realm of the web. WebDAV is a protocol that extends HTTP with distributed authoring facilities and DeltaV is a proposed extension that adds versioning capabilities [Whi99][Web99]. SCM research has had a definite impact in this arena: early incarnations of the interface functions in the protocol of WebDAV were partially inspired by NUCM [Hoe96] and DeltaV is actively being developed under the partial leadership of SCM vendors. Clearly, the field is having its impact and the two standards incorporate many of the good practices that have been researched and developed over time.

3.2 Some Failed Transitions

SCM research also has produced a number of ideas that have not been able to succeed in making the transition to industrial practice. The root cause of these failures lie in the level of complexity required to master the ideas, or the level of effort required by customers to use the feature, or that the typical customer does not feel an actual need for the feature, or because vendors think the feature is outside the field of SCM.

Industry tends to ignore such ideas, until customer practices meet the need for the feature, and until a complex idea can be transformed in a way that hides most of the actual complexity. None of these conditions are granted, nor can they be easily forecast.

3.2.1 Smart recompilation

It was Tichy that coined the term "smart recompilation" in 1986 [Tic86]. His work was shortly thereafter followed by a number of other approaches [Sch88, App93], a survey of which is presented by Adams et al. [Ada94]. Unfortunately for the domain of SCM, the syntactic analysis needed to perform smart recompilation often takes more time than the time saved by avoiding unnecessary recompilations. Especially with today's hardware, advanced recompilation algorithms are simply not needed—in the general case, compilation "as is" is fast enough.

Nonetheless, smart recompilation cannot be labeled a failure altogether. The idea is appealing and has started to be incorporated into language-dependent programming environments. In these environments, syntactic information is available "for free". The Ada and Chill programming environments, for example, were among the first to adopt smart recompilation techniques [Bret93]. Now, most modern programming environments, including the popular Visual Studio, rely on these techniques. Smart recompilation, thus, is an area of research that has found its way into industry in a domain other than SCM.

3.2.2 Version models, data models, system models

Extending and generalizing versioning capabilities clearly has been a core topic of SCM research since its early beginnings. Much work has been dedicated to advanced versioning models and associated selection techniques, including interesting formalizations of these approaches [Bie95, Nav96, Zel97]. From a researchers point of view, these approaches improve over the current state of the art by offering new or alternative modeling capabilities. From a practitioner point of view, however, some of these approaches are overkill: they provide more power than actually needed, at a cost of extra complexity and reduced efficiency.

Data and system models used by today's commercial SCM systems only capture the files and directories that represent a software product, the compile-time dependencies, and a small set of attributes. Logically, researchers hypothesized that a more powerful data and system model would allow the SCM system to provide better support for precisely capturing the evolution of the artifacts it manages. This simple idea fostered a number of contributions of dedicated data and system models for SCM, models in which everything is versioned, including files, attributes, general relationships, configurations, and workspaces [Est85, Dit87, Bou88, Tho89, Gul91, Est94].

Commercial SCM systems improved their data model, but are still far from these research attempts for two reasons. First, no commercial database exists that can support these kinds of advanced models, and building such a database, either from scratch or on top of a commercial relational database, is a daunting undertaking. Second, the models are simply too complicated or inefficient. We have experience of vendors who provided users with versioned relationships, only to abandon these efforts: managing such relationships was too cumbersome for the users. Unless research invents automated techniques that support users in updating and maintaining advanced data and system models, they will never be put into practice.

3.2.3 Generic Platform

SCM is meant to provide a language- and application-independent platform that can handle any kind of artifact. The focus of SCM research on managing source code, however, has lead to a platform that tends to be more specific than desired. Limitations in the data and system model, even though necessary from the perspective described above, make it difficult to manage complex structures: too much additional manual work is needed to capture the extra information that is necessary in, for example, supporting product data management (PDM) [Est98]. Similarly, existing SCM tools cannot effectively support web site management or document management. Even an integration with existing tools in those domains that already include some of the basic SCM services (e.g., data and version management, process management, rebuilding) has proven to be difficult.

There is hope, however. Since software is an increasing part of almost any complex manufactured object, the need to consistently and conjointly manage software and hardware may force SCM researchers to reconsider the situation, and to start research into advanced data modeling facilities. Some research has started to look into this direction, but much additional research will be required to reach a satisfactory result.

3.3 What is Next?

SCM research has addressed many different topics and it is fair to say that by now the basic principles, concepts, and techniques are set. Consensus also seems to be emerging for more advanced functionality, evidenced by the fact that most high-end CM systems are closing in on satisfying the spectrum of functionality laid out by Dart [Dart91] [Est00]. Nonetheless, many research issues remain to be addressed. In particular, the field as a whole is now sorting out its relationship to, and place within, the overall picture of software development.

This research is breaking two fundamental assumptions that underlie current SCM systems: (a) the focus on managing the implementation of software and (b) the basic philosophy of SCM being language and application independent. Breaking the first assumption requires careful management of artifacts produced earlier in the life cycle (e.g., requirements, design) and later in the life cycle (e.g., deployment, dynamic evolution and reconfiguration). Breaking the second assumption involves the integration of SCM functionality into particular environments (for example, integrated development environments) and representations (for example, product line architectures).

At the forefront seem to be the issues of unifying SCM and PDM, a better management of component-based software development, and a better understanding of the relationship between SCM system models and software architecture [Hoe98, Hoe98b]. It is clear why these issues are currently being addressed: SCM no longer is a stand-alone discipline. To survive, it needs to stay abreast of new developments, trends, and technologies. Of course, in no way can we predict the future of SCM. It may be that sorting out some of these issues turns out to be trivial, not relevant, or far too difficult for practical application. However, this is the way of research and providing an answer to the questions that are raised is what is important, even if those answers close, rather than open, doors.

Finally, although conferences, workshops, and personal interactions undoubtedly play a tremendous role in research transition, it is impossible to quantify their impact. Continual attendance of the SCM workshop series by chief architects of some of the most influential SCM systems, the transition of academic researchers to industry and vice-versa, and anecdotal evidence brought forth in our personal interviews, indicate that these kinds of interactions are absolutely necessary for any kind of research impact to occur. Conference and workshop venues create a community of researchers and practitioners, raise new issues to be addressed, set high-level expectations for new directions, and, in the case of SCM-1, have set the standard terminology still in use today [Win88].

4 CONCLUSION

A report like this can (and did) generate a lot of dissent. Indeed, writing this report made obvious the difficulties in reaching agreement between researchers and vendors. A caricature of our starting point was, from researchers, to claim the ownership of almost all ideas, dismissing tool realization as "engineering common sense", and from vendors to claim they (re)invented everything they needed, dismissing concepts, ideas, architectures as "engineering common sense". From the intensive discussion in our group emerged a much more balanced perspective.

We came to the conclusion that both camps require engineering creativity, and that tracking down where ideas came from and what was influential is virtually impossible. We also rapidly agreed that research have been fundamental in the success of SCM; and that industrial research, both from corporate research labs and vendors have had a definitive influence. Thus, we have tried to document the flow of ideas, based on evidence such as publication and implementations, an attempt that we hope comes as close as possible to reality.

SCM is arguably one of the most successful software engineering disciplines, and it is difficult to imagine this kind of success would have prevailed without research fueling continuous innovations. This report demonstrates that the impact of this research, whether industrial or academic, is undeniable—many of the fundamental techniques underlying current SCM systems were first published in one form or another.

Like any other field, SCM research has had its successes and failures. Some ideas are universally adopted, others have had limited impact, and yet others never saw fruition. Timing has been critical: whereas most contributions were rooted in practical, day-to-day problems, others were too early for their time and not practically relevant for the problems at hand. Nonetheless, the actual evolution of the field does let us think that some those ideas will eventually be useful. As demonstrated by the remarkable time delay in the adoption of change sets, it is often market readiness

that determines success: over time, however, most ideas have trickled through.

SCM research is still alive. The basic concepts and technologies have been settled, but much work remains to be done. In particular, the field as a whole is now sorting out its relationship to other domains, such as product data management, component-based software engineering, and software architecture. We look forward to the advances that will come from this research, and are proud to be a part of a field with such a rich legacy as SCM.

The international research community has contributed many ideas to the field of SCM but more importantly, it has provided a forum for the publication and discussion of ideas. The ICSE and SCM series of conferences enabled people from academia and industry to interact and exchange ideas. The set of important ideas may be changing but the need for an active research community remains essential.

Acknowledgements: We would like to thank all in the field of SCM, whether researcher, industrial vendor, or customer. Knowingly or unknowingly, most if not all of you have advanced our field to where it is now—a standard and accepted part of any serious software development project.

This work was supported in part by the U.S. National Science Foundation under grants CCR-00-10041 and CCR-01-37766, and by IEE

REFERENCES

[ADC88] Aide-De-Camp: *A software Management and Maintenance System*, National Software Quality Assurance Conference, the National Institute for, Software Porductivity, Washington DC, April 1988.

[Ada94] R. Adams, W. Tichy, A. Weiner. *The cost of selective recompilation and environment processing*.. ACM Trans. Soft. Engineering Methodology. 3, 1, Jan 1994, P 3-28.

[All95] L. Allen, G. Fernandez, K. Kane, D. Leblang, D. Minard, and J. Posner. *ClearCase MultiSite: Supporting Geographically-Distributed Software Development*. ICSE SCM-4 and SCM-5, Seattle USA, May 1995.

[App93] Twentieth ACM Symp. on Principles of Programming Languages, Charleston, SC, ACM Press, page 439-450, January 1993.

[Atkin98] D. Atkin. *Version Sensitive Editing : Change History as a Programming Tool*. SCM-8, 1998, Brussels, LNCS 1439.

[Bel87] N. Belkhatir and J. Estublier. "Software management constraints and action triggering in Adele program database." In *1st European Software Engineering Conf.*, pages 47–57, Strasbourg, France, Sept. 1987.

[Bel91] N. Belkhatir, J. Estublier, and W. L. Melo. "Software process modeling in Adele: The ISPW-7 example." In I. Thomas, editor, *Proc. of the 7th Int'l Software Process Workshop*, San Francisco, CA, October 16–18 1991. IEEE Computer Society Press.

[Bie95] N. Bielikova, P. Navrat. *Modelling software systems in configuration management*. In Applied Mathematics and Computer Sciences. 5(4) : 751-764, 1995.

[Bret93] B. Bret. *Smart recompilation: what is it?, its benefits for the user, and its implementation in the DEC Ada compilation system*. Conference proceedings on TRI-Ada '93 September 18 - 23, 1993, Seattle, WA USA

[Bou88] Gerard Boudier, Ferdinando Gallo, Regis Minot, and Ian Thomas. *An Overview of PCTE and PCTE+*. In Proc. ACM/SIGSOFT Software Engineering Symposium on Practical Software Development Environments, Boston, 28-20 Nov. 1988, pp. 248-257.

[Buff95] J. Buffenbarger. *Syntactic Software Mergers*. SCM-5, Seattle USA, May 1995, LNCS 1005.

[Clem95] G. Clemm. *The Odin System*. SCM-5, Seattle June 1995, pp241-263. Springer Verlag LNCS 1005.

[Con97] R. Conradi and B. Westfechtel. *"Toward an Uniform Model for Software Configuration Management"*. In SCM-7 Workshop.pages 1-17. Springer LNCS 1235. May 1997.

[Con98] R. Conradi, A. Fuggetta, M.L. Jaccheri. Six theses on Software Process Research. EWSPT 98. Weybridge, UK, September 1998.

[Dar91] Susan Dart. *Spectrum of Functionality in Configuration Management Systems*. CMU/SEI-90-TR-11 ESD-90-TR-212. http://www.sei.cmu.edu/legacy/scm/tech_rep/TR11_90.

[Dar97] Darcy Wiborg Weber . *Change Sets Versus Change Packages: Comparing Implementations of Change-Based SCM*. Proc. 7th International Workshop on Software Configuration Management (SCM-7), Boston, USA, Springer Verlag LNCS 1235, 18-19 May 1997.

[Dit87] Klaus R. Dittrich, W. Gotthard, and P. C. Lockemann. DAMOKLES - *The Database System for the UNIBASE Software Engineering Environment*. Database Engineering, 10(1), March 1987.

[Din98] E. Di Nitto and A. Fuggetta (Eds). *Process Technology*. Kluwer Academic Publishers, Boston.. January 1998.

[Est84] Jacky Estublier, S. Ghoul, and S. Krakowiak. Preliminary Experience with a Configuration Control System for Modular Programs. In Peter B. Henderson, editor. *Proc. 1st ACM SIGSOFTSIGPLAN Software Engineering Symposium on Practical Software Development Environments (Pittsburgh), 197 p.*, April 1984. ACM SIGPLAN Notices 19(5)149-156, May 1984.

[Est94] Jacky Estublier and Ruby Casallas. *The Adele Software Configuration Manager*. In [Tic94], pp. 2-11. 1994. http://www-adele.imag.fr/Les.Publications/BD/adele1994est.html

[Est97] J. Estublier and S. Dami and M. Amiour. *High Level Process Modeling for SCM Systems*. SCM 7, LNCS 1235. pages 81--98, May, Boston, USA, 1997

[Est98] J. Estublier and J.M. Favre and P. Morat. *Toward PDM / SCM: integration?*. In proc SCM8, Bruxelles, Belgium, July 1998. Springer Verlag, LNCS 1439, pp75-95.

[Est00] Jacky Estublier. *Software Configuration Management: A Road Map*. In Anthony Finkelstein, editor, The Future of Software Engineering (supplementary Proc. for 22th Int'l Conf. on Software Engineering), Limerick, Ireland, June 2000, ACM Press, Order No. 592000-1, pp. 279-289.

[Fel79] Stuart I. Feldman. *Make -- a Program for Maintaining Computer Programs*. Software -- Practice and Experience, 9(3):255-265, March 1979.

[Fin94] A. Finkelstein, J. Kramer, B. Nuseibeh. *Software Process Modeling and Technology*. John Wiley, Advanced Software

Development Serie. ISBN 0 471 95206 0. 1994.

[Fraser87] C. Fraser, E. Myer, *An editor for revision control.* ACM Transactions on Programming Languages and Systems. 9(2) April 1987.

[Gul91] Bjørn Gulla, Even-André Karlsson, and Dashing Yeh. *Change-Oriented Version Descriptions in EPOS.* Software Engineering Journal, 6(6):378-386, November 1991.

[Hoe96] A. Van der Hoek, D. Heimbigner, and A.L. Wolf. *A Generic, Peer-to-Peer Repository for Distributed Configuration Management.* Proceedings of the 18th International Conference on Software Engineering, Berlin, Germany, March 1996.

[Hoe98] André van der Hoek, Dennis Heimbigner, Alexander L. Wolf. *Software Architecture, Configuration Management, and Configurable Distributed Systems: A Ménage a Trois.* Tech Report CU-CS-849-98. U. Colorado.

[Hoe98b] André van der Hoek, Dennis Heimbigner, Alexander L. Wolf. *System Modeling Resurrected.* System Configuration Management (SCM-8), Brussels, Belgium 1998, Springer-Verlag LNCS 1439.

[Hor89] S. Horwith, J. Prins, T. Reps. *Integrating non-interfering versions of programs.* ACM Transaction on Programming Languages and Systems. 11(3) July 1989.

[Hou81] H. Hough. *Some Thoughts on Source Update as a Software Maintenance Tool.* IEEE Conference on Trends and Applications, CH1631-1/81/0000/0163 May 1981.

[Hunt76] J. Hunt, M. McIllroy. *An efficient algorithm for differential file comparison.* Technical Report 41, Bell Labs, June 1976.

[Hunt96] J. Hunt, K. Vo, W. Tichy. *An Empirical Study of Delta Mechanisms.* SCM6, Berlin, March 1986. LNCS1167.

[Korn95] D. Korn, K. Vo. *Vdelta: Efficient data differencing and compression.*

[Knuth84] D. Knuth, *Literate Programming,* Computer Journal, pages 97-111, 19984.

[Kruskal84] V. Kruskal. *Managing Multi-Version programs with an editor.* IBM Journal of Research and Development. 28(1), January 1984.

[Lam91] Charles Lamb, Gordon Landis, Jack Orenstein, and Dan Weinreb. *The ObjectStore Database System.* Comm. of the ACM, 34(10):50-63, October 1991.

[Lie89] Anund Lie, Tor M. Didriksen, Reidar Conradi, Even-André Karlsson, Svein O. Hallsteinsen, and Per Holager. *Change Oriented Versioning.*
In Carlo Ghezzi and John A. McDermid, editors. *Proc. 2nd European Software Engineering Conference (Coventry, UK), Springer Verlag LNCS 387, 496 p.*, September 1989, pp. 191-202.

[McC00] McCabe/True Software. Documentation 2000. http://www.mccabe.com/products.htm.

[Meyers86] E. Meyers. An O(ND) *Difference algorithm and its variations.* Algorithmica, 1(2):251-266, 1986.

[Mic96] Josephine Micallef and Geoffrey M. Clemm: *The Asgard System: Activity-Based Configuration Management,* In Ian Sommerville, editor, Proc. Software Configuration Management, ICSE'96 SCM-6 Workshop, Berlin, March 1996, Springer Verlag LNCS 1167, pp. 175-186.

[Nav96] P. Navrat, N. Bielikova. *Knowledge controlled version selection in software configuration management.* Software Concepts and Tools. 17:40-48, 1996.

[Pro98] Promoter group. *Software Process: Principles, Methodology, Technology.* Springer Verlag, LNCS 1500. 1998.

[Rei91] Chris Reichenberger. *Delta storage for arbitrary non-text files.* In Proceedings of the 3rd International Workshop on Software Configuration Management, Trondheim, Norway, 12-14 June 1991 (June 1991), ACM, pp. 144--152.

[Reps88] T. Reps, S. Horwitz, J. Prins. *Support for Integrating program variants in an environment for programming in the large.* Proc. Int. Workshop on Softwae Version and Configuration Control. Grassau, Germany 1988.

[Roc75] Mark J. Rochkind. The Source Code Control System. *IEEE Trans. on Software Engineering,* SE-1(4):364-370, 1975.

[Sar88] N. Sarnak, B. Bernstein, and V. Kruskal. *Creation and Maintenance of Multiple Versions.* In [Win88], pp. 264-275, 1988.

[Sch88] Robert W. Schwanke and Gail E. Kaiser. *Smarter Recompilation.* . ACM Transactions on Programming Languages and Systems (TOPLAS) Pages: 627 - 632 Periodical-Issue-Article 1988 ISSN:0164-0925

[Tic82] Walter F. Tichy. *Design Implementation and Evaluation of a Revision Control System.* In Proc. Sixth International Conference on Software Engineering. 1982.

[Tic85] Walter F. Tichy. *RCS -- A System for Version Control.* Software -- Practice and Experience, 15(7):637-654, 1985.

[Tic86] W. Tichy. *Smart recompilation.* ACM Transactions on Programming Languages and Systems, 8(3):273--291, 1986

[Tho89] Ian Thomas. ***PCTE** interfaces: Supporting tools in software-engineering environments.* IEEE Software, 6(6):15-23, November 1989.

[Wes01] Bernhard Westfechtel, Bjørn P. Munch, and Reidar Conradi. *A Layered Architecture for Software Configuration Management.* IEEE Trans. Software Engineering, to appear in 2001, 24 p.

[Win88] Jürgen F. H. Winkler, editor. Proc. ACM *Workshop on Software Version and Configuration Control,* Grassau, FRG, Berichte des German Chapter of the ACM, Band 30, 466 p., Stuttgart, January 1988. B. G. Teubner Verlag.

[Whi91] David Whitgift. *Methods and Tools for Software Configuration Management.* John Wiley and Sons, England, 1991, 238 p.

[Whi99] Jim Whitehead. *Goals for a Configuration Management Network protocol.*In SCM9, LNCS 1675, pages 186-204, Toulouse September 1999.

[Web99] WebDav. *HTTP extentions for distributed Authoring.* RFC 2518. http://andrew2.andrew.cmu.edu/rfc/rfc2518.htm. February 1999.

[Zel97] Andreas Zeller and Gregor Snelting. *Unified Versioning through Feature Logic,* ACM Transactions on Software Engineering and Methodology, 6(4):397-440, Oct. 1997.