

Tool Adoption Issues in a Very Large Software Company

Jean-Marie Favre Jacky Estublier Remy Sanlaville

*Adele Team, Laboratoire LSR-IMAG
University of Grenoble, France
<http://www-adele.imag.fr>*

Abstract

Tool adoption is a major issue in software engineering. In the last decades many ideas and tools have been developed by academics but only a few have had a direct impact on software industry. This paper describes the major issues in tool adoption and presents some technological approaches to cope with these issues. The focus is then on adoption-in-the-large. The results of a ten-years collaboration between the LSR laboratory and Dassault Systèmes are presented. Dassault Systèmes is one of the major software companies in Europe. Two scenarios in tool adoption are described. The first one describes the successful adoption of a configuration management tool, the second one describes adoption issues related with a reverse architecting tool.

1. Introduction

Industry often says “no thanks” to software engineering (SE) research, in particular when tools are proposed [1]. Technology maturation and adoption is known to be a very long process [2][3]. Nevertheless, it is still disappointing to see that while many SE tools are developed, their adoption in software industry is quite an exception. In fact, while researchers concentrate on designing and building *tools*, industry is looking for *solutions*. Even when a tool is very close to a solution, it is actually very hard to get this tool used in industry. The “last mile” [4] is a very difficult step in the technology transfer process. It is indeed a crucial one.

To cope with this problem, a new research trend called *adoption centric software engineering* (ACSE) aims to address the adoption issue in the first place. Successes and failures in software engineering adoption should be studied. Innovative ways to ease adoption must be found.

This paper describes a case study led over the last decade in a very large software company. It describes tool adoption successes and failures in the context of a collaboration between the LSR-IMAG research laboratory and the Dassault Systèmes (DS) company. DS is one of the largest software editors in Europe. This paper shows how the size of the company can lead to specific problems that are unlikely to occur in small or medium-size companies.

The paper is structured as following. Section 2 describes the main issues in tool adoption. Section 3 presents different ways to cope with technical issues in tool integration. Section 4 shortly describes the main characteristics of Dassault Systèmes which constitutes the context of the case study. Section 5 describes a positive scenario in which a configuration management tool is adopted. Section 6 describes problems related to the adoption of an architecting tool. A discussion is provided in Section 7 and finally Section 8 concludes the paper.

2. Issues in software engineering tool adoption

Over the last decades, Software Engineering (SE) research has produced many methods and tools. To evaluate the actual impact of this body of research on software industry, a large study called IMPACT is being held at the international level. This study shows that the topic of tool adoption is quite complex.

Even when SE tools are quite close to solutions and are based on well-defined concepts, there are still important barriers to their actual adoption. Many factors should be taken into account :

- *Scalability issues.* Very often, a large number of unexpected problems are discovered when applying good tools at a large scale. This includes not only the size of the software but also the size of the company. While a tool could perfectly suit the needs of a single user, its use by hundreds of software developers may unveil new issues.
- *Usability issues.* Many software engineering tools focus on functionality, not on usability. However the user interface plays a very important role. Software engineers are under time pressure. They will not use a tool if they can't get easily the result they need from the tool.
- *Tool integration issues.* It is unlikely for a tool to be adopted if it is not tightly integrated with the tools already in use. Data, control and interface integration are required.

- *Process integration issues.* A very good tool will not be adopted if it does not fit well in the development process of the company. This is especially true in companies with well defined and strict software processes.
- *Customization issues.* Large companies often define their own set of concepts and rules. Customization could be of paramount importance at the company level. A systematic way to use tools over the whole company should be enforced. In some situations customization is also a desired property at the end-user level.
- *Deployment issues.* In a very large company deploying a tool could be a real issue, especially since different teams may use different platforms, different tools, etc.
- *Administration issues.* Once installed, some complex tools require a great amount of administration. This could include tasks such as backups, error recovery, creation of new users, new projects, etc. The amount of work and the skills required could be a serious barrier to the adoption of complex tools.
- *Evolution and continuity issues.* Just like any other software, a SE tool has to evolve to meet the company evolving requirements. A company will not invest time and money in a tool if not convinced that the tool will be maintained and enhanced for years. Universities are hardly convincing on that aspect.
- *Training issues.* The introduction of any new tool implies a learning process. Except for very simple tools, learning new concepts and advanced features requires a full training program. This is a very serious issue in large companies since learning represents a temporary loss of productivity with uncertain return on investment.
- *Strategical issues.* At the level of the company, many factors could also hamper the adoption of a particular tool. This might occur for instance to avoid dependencies towards a particular organisation or tool vendor, licensing problems, or any other political issues.

Introducing a new tool in a company represents indeed a risk. In many cases, it is actually quite difficult to get effective managers and software engineers involvement.

3. Tool integration

Many issues listed above are intimately linked to the organisation and the strategy of the company. Researchers should avoid trying to address directly these issues because it is very unlikely for them to have the appropriated skills and knowledge. It is extremely hard for a researcher to have an impact on the company major decisions.

To have an actual impact, ACSE research should therefore concentrate on those parts that can be controlled in a research environment. It is therefore better to focus on technology rather than strategy.

3.1. Integration to existing tool sets

One way to ease the adoption of a tool is to integrate it into the set of tools already used in the company. Currently they are at least 4 main groups of tools used in almost any company producing software:

- *Web-based and communication tools.* Web browsers and mailers are everywhere. They are used by every actor in the company. They play a very important role since all communications rely on these tools.
- *Integrated Development Environments (IDE).* These tools are now widely accepted among the developer community. They are primary tools for most software engineers.
- *Office Tool Suite.* Software engineers and managers share in common the use of text editors like Word, FrameMaker or WordPerfect. They also use tools like PowerPoint. Documents are exchanged using standard formats such as HTML, Postscript, PDF, etc. Managers deal with metrics and other indicators using spreadsheets and bar chart generators.
- *Modelling Tools.* Tools supporting analysis and design are becoming increasingly popular. This is in part due to the success of the UML standard in software industry. There are now numerous UML workbenches such as Rose, Objecteering, Together, etc. These environments support not only the drawing of models, but they also integrate code generators, documentation generators, metrics, and many other tools. Modeling environments still have to be adopted at large but there is no doubt that there is an increasing interest in UML and its associated tools.

Users are more likely to adopt a tool that works in the same environment they use on a daily basis. This means that SE tools should be integrated to the existing set of tools.

3.2. Integration Levels

To be effective, different kinds of integration [5] should be supported between SE tools and existing tools:

- *Data integration.* Data consumed and produced by SE tools should be shared with other tools. For instance, the result of a metric or profiling tool should be easily exportable to a spreadsheet for further manipulations. It should also be very easy to insert an architectural view into an existing document, to publish it on the web or to send it via email for further annotations.

- *Control integration.* It should be possible to call a tool from another one. For instance, it should be possible to call the functions of a metric tool from an IDE, and to display the result through a visualisation tool.
- *User-interface integration.* All tools should be accessible from a consistent user-interface with a common look and feel. The popularity of today IDEs is in part due to the fact that many programming tasks can be performed easily through a unique user interface.

Work on tool integration is far from new. The different kinds of integration were identified in the 80's. Interoperability techniques such as RPC and Corba take their roots from then. In the 80's and 90's a large amount of effort was dedicated to Computer Assisted Software Engineering (CASE).

In particular, many research projects focused on syntax-directed environments. In such environments syntax-based descriptions played a central role in supporting both data integration and control integration. These environments have not been adopted widely, though many concepts and techniques they have been introduced, form the basis of modern IDEs. It is interesting to note that many techniques initially developed in the CASE field have been successfully applied to other domains. This includes for example office suites which present nowadays many of the desirable features of CASE tools such as multiple synchronized views. In fact, one of the mistakes in the CASE vision might have been to believe that the CASE market was large enough per se to support the development and adoption of a rather immature technology.

3.3. Approaches to SE tool integration

It appears now clearly that the market is driven by web technologies, office suites, and to a lesser extent by IDEs and UML modelling tools. New and innovative SE tools must be integrated into these existing suites. The other way around is very unlikely.

The move from Field [6] to Desert [7] is representative of this shift. Field is one of the precursors of CASE environments with its strong emphasis on control integration. Desert, its successor, seeks to integrate programming facilities into the FrameMaker text editor. More recently a few attempts have been made to use PowerPoint as a design tool. These approaches provide good illustrations of Adoption Centric Software Engineering.

Integrating new tools in a proprietary toolset could be far from obvious. Fortunately, a lot of improvements have been made over the last years, making the integration of SE tools possible. This results from efforts made both by tool vendors and by standardization bodies

- *Standard exchange formats.* Different de-facto standards are widely used to exchange documents such as RTF, MIF, HTML or PDF. The XML standard will clearly have a major impact on data integration. Specialized formats such as GXL [8] could be useful to improve interoperability between research tools dealing with graphs. Nevertheless, there is currently no indicator that it will be adopted by software industry.
- *Standard schemes:* Specifying a file format is not enough to ensure data integration. It is also necessary to specify the schema or the meta model used to represent data. Fortunately these concepts are well supported by current technologies such as XML and the OMG' Model Driven Architecture (MDA) [9][10]. For instance MDA provides XMI to represent and exchange UML models and meta models. It also includes some standard meta models for different application domains including the Software Process Engineering Metamodel (SPEM) and the Common Warehouse Metamodels (CWM).
- *Standard APIs.* Using standard exchange formats and standard schemes enables data integration but not control integration. One way to cope with this issue is the publication of APIs and in some case the standardization of the APIs. Most of the tools described in Section 3.1 provide APIs and include a "developer kit". The MDA approach is based on the MOF standard [11] which describes how to generate a standard API for each particular meta model.
- *Scripting languages.* Using APIs can be quite complex. Scripting languages and macros offer a much cheaper alternative for customization and automation of common tasks. Nowadays, almost all commercial environments include some sort of scripting capabilities, although most of the time the scripting languages they provide are proprietary. For instance office tool suites typically include extensions of the Basic language. UML workbenches like Rose or Objectteering include proprietary scripting languages that support the manipulation of UML models and the addition of new features.
- *Plugin and component technologies.* Most environments also support the concept of "plugin". which enables the addition of new features in predefined points of extension. More generally, a large amount of research is devoted to component technologies including for instance Microsoft' COM, Sun JavaBeans and Enterprise Java Beans, Corba CCM, Microsoft .NET, etc. While these technologies are not specifically oriented towards the development of SE tools, component technologies will certainly play an increasing role in the future. It is interesting to

notice that the COM component model was originally designed in the context of an office tool suite to enable the inclusion of “live documents” in other documents (e.g. the inclusion of a spreadsheet in a word processor document). These kind of technologies obviously present a strong interest to embed software engineering tools or views in other documents. With respect to the MDA approach, a new research trend tries to define the notion of MDA components.

- *Standard infrastructures.* While component technologies enable the integration and assembly of new tools, they does not ensure per se a strong consistency between the applications being built. For instance good user-interface integration is not possible without some standard rules. Fortunately, the emergence of very open infrastructures such as Eclipse [13] can cope with these issues.

As pointed out before, tool integration has been a primary objective of CASE research. In the 80’s and 90’s the problem was to integrate together SE tools like editors, debuggers, profilers, etc. This objective has been met, but mostly in situations in which the tools are built by a single tool vendor, or by a small set of tool vendors with very close partnerships.

In the context of ACSE, the problem has changed: it seems now necessary to be able to integrate an arbitrary SE tool to the suites already used in a given company. Fortunately, the generalization of the technologies described above leads to new opportunities. In fact, for each set of tools described in Section 3.1, (web-based tools, office tools, IDEs and modeling tools), efforts have been undertaken to improve their openness. Web-based and office tools are increasingly based on XML technologies, Eclipse could have a strong impact on IDEs, MDA could help the interoperability between modelling tools. One important approach in ACSE is then to use these emerging standards in order to facilitate the adoption of SE tools.

3.4. Summary

There are many barriers to tool adoption and the “last mile” is always a difficult step. Some issues are due to the organisation and strategy of the company itself and should not be addressed directly by researchers. Others approaches are related to the technology used. A major trend in ACSE is to use industrial standards as a basis for the development of innovative SE tools. Though promising this approach will not solve all the problems. The following case study shows that SE tool adoption is a real challenge, especially in large software companies in which there is a real shift from adoption-in-the-small to adoption-in-the-large.

4. Case study in a large software company

In the 80’s a Software Configuration Management (SCM) tool called Adele was developed by our team at the University of Grenoble. This tool was very generic and it included many innovative ideas. Adele was adopted by different companies including Matra and Sextant. The expertise gained with industrial partners has resulted in a long and tight collaboration between the LSR-IMAG laboratory and Dassault Systèmes [14][16]. While the first part of the collaboration has been dedicated to configuration management, the second part has been devoted to software architecture. The rest of this paper uses the whole collaboration as a case study to summarize our experience in SE tool adoption over the last decade.

4.1. The Dassault Systèmes (DS) company

Dassault Systèmes (DS) is one of the largest software editors in Europe. DS is also the world leader on CAD/CAM with more than 19 000 clients and 180 000 seats. DS constitutes a very interesting context for a case study because of the size of the company, and the architecture of its software.

The company is indeed very large: 1000 engineers are working simultaneously on the same software product. CATIA is one of the main software products with more than 5 MLOC. The requirements on CATIA software architecture are also very strong, especially since many customers around the world contribute to the development of the CATIA product line. CATIA is sold to companies that have an important know-how in their respective domains. Boeing, for example, is a specialist in plane construction and owns many software tools and rules. DS customers must be capable of adapting CATIA, integrating their own functions into existing DS applications. These extensions constitute a significant part of the software. Boeing alone is said to have developed more lines for CATIA adaptation and extension than DS for CATIA itself.

Actually DS and its partners constitute a large virtual software factory in which thousands of software engineers collaborate to the development and the evolution of a very complex software product line. From the software engineering perspective this implies very strong needs both in configuration management and in software architecture.

The collaboration between the Adele team and DS has been centred around these two themes. Section 5 describes the process leading to the adoption of the Adele SCM tool, while Section 6 describes the difficulties we met in deploying the OMVT, architecting tool.

But let us first review how tools are usually introduced in DS since it may be representative of a typical organisation for adoption-in-the-large.

4.2. Tooling support in large software companies

In small companies tool adoption issues are mostly related to individual software engineers adopting individual tools. This could be referred to as adoption-in-the-small. This contrasts with tool adoption-in-the-large that takes place in very large companies and that requires a much more complex organisation. There are various reasons for that:

- The collaboration between hundreds of software engineers is possible only if the company has a rather well defined process, the tools playing an important part in that process.
- Scalability issues could be so huge that only few tools in the market, if any, fit the company needs. Tool evaluation is indeed a very important issue.
- Large companies often have specific needs related to their process and their culture. Customizing existing tools could be a complex yet essential task. Developing new tools is sometimes necessary.
- Many issues that can be solved rather easily in the context of small companies, can lead to very complex problems in large companies. This includes for instance deployment on hundreds of machines, learning programs over thousands of developers, administration of hundreds of projects and thousands of user accounts, etc.

Obviously programmers must not be in charge of managing SE tools; they must concentrate on their jobs, that is developing software. In large software company this usually leads to the existence of a *Tool Support Team* (TST). The TST is in charge of all activities related to tool support including tool evaluation, customization, integration, deployment, administration, learning, support, etc. Most of the time, this team is in charge of evaluating commercial tools. In some occasions some TSTs develop tools internally.

In such a context tool adoption is not only restricted to end-user adoption (that is adoption by software engineers). A tool will be adopted only if it meets the needs of three kinds of actors:

- *Managers*. They define the strategy of the company at various levels. Without their agreement a tool will not be included in the company toolset. Researchers have to convince them of the actual benefits that the tool is supposed to bring. And this should not be in technical terms but in terms of actual benefits.
- *TST members*. They deal with all technical aspects of tools. Most of the time, failures in tool adoption will happen at that level, because this team is in charge of evaluating the tool. Scalability issues, deployment issues, integration issues, etc. will be discovered here. Researchers must collaborate closely with the TST during the tool adoption phase.

- *Software engineers*. Ultimately software engineers are those who use the tool. Some usability issue could appear at this level because software engineers may have different habits in performing their development activities. The pressure on them to use a given tool could be important or not. Typically a company will oblige all software engineers to use a critical tool such as a configuration management tool. It will be much less strict on second-class tools like a browser for instance.

This organisation makes tool adoption even more difficult. Researchers should face tool adoption by managers, TSTs and end-users. They should cooperate therefore with many different actors in the company, and this at different points in time. This could be quite difficult, especially since researchers are usually neither aware of the precise organisation of the company nor of the exact role of each person they meet. In each situation, the discourse should be adapted to the concerns of the interlocutor.

The situation is even more complex in an organisation such as Dassault Systèmes, because of the various partners constituting the virtual software factory. A tool used in the company, could be later included in the development kit delivered with the product sold. For instance, the Adele tool, after being adopted by DS, was included in the CATIA toolset and delivered to customers such as Boeing. The next sections describe two scenarios of tool adoption in the context of Dassault Systèmes.

5. A successful story in configuration management

All adoptions of the Adele SCM tool [15] started at the initiative of the company. This was also the case of DS. This company really needed a configuration management system to manage the parallel development of CATIA by hundreds of software engineers. Actually, before the first contact with our team, DS first asked other users of Adele what they thought of the product and the support. They checked many other informations, technical ones but also, if not essentially, non technical ones. Such precautions are natural and common. An SCM tool is a critical tool, involving a large training, with deep influence on the software process, and even in the software structure. Any failure of the tool can have dramatic consequences for the company, ranging from few hours of unavailability, to loss of sources, or delivery of inconsistent products. The investment is heavy, the choice risky. Evolution and continuity of the tool are of paramount importance in this context. Any large company needs to be convinced the product will live for long. It was improbable a tiny academic team could satisfy these requirements.

DS took the risk to rely on a research team. Actually, this is mainly because they had to. On one side, their analysis showed Adele was the only system capable to satisfy their requirements, mostly because of its flexibility and its capability to adapt to their very unusual process and characteristics. Solving the customization issue was indeed considered as one of the major issues. On the other side DS had to select an SCM tool. The evaluation of existing tools showed that commercial tools were not well-suited to fit their uncommon size and requirements. During the evaluation period DS crashed almost all evaluated tools. Scalability issues in existing commercial tools was therefore another argument to invest on Adele.

Having a tool they could tailor to their needs, and having a team capable of making it evolve in their direction was a very strong point. It is surprising to consider that we were the only ones (apart from the other tool vendors) who tried to discourage them to use Adele, arguing that it was not designed for such a huge software and large software team.

Indeed, the first full scale test of the tool was almost a disaster; efficiency being well below requirements. Scalability was also a major concern. In this first phase the tool was also clearly rejected by software engineers because they were feeling that the tool was not helping them in their work. Usability was also a major issue. A number of problems had to be fixed in a panic mode, because the tool was already used at large. Then, major parts of the tool were redesigned and reimplemented to cope with scalability and usability issues. This was done quite quickly and efficiently. This ability to make the tool evolve was probably the main factor that convinced the managers to continue the experience. The product was under control and the relation between the TST and the research team was good.

The situation thus improved, to reach an acceptable level in which the acceptance by end-users was relatively good. In parallel, different customizations were experimented by the TST. The tool proved to be flexible enough to provide solutions for all new requirements. One of the main features of Adele is that it included an event-based system that enabled to attach a trigger and a reaction to any arbitrary event raised during the software process. The final version of the parametrized system was used for years. It included about 10 000 triggers describing the complete development process of the whole company. It is probably the world's largest trigger-based industrial application.

A new version of the system was later developed from scratch by the TST. The goal was to retain the exact same features, but with an order of magnitude in size of software to be supported, and in efficiency. Of course that new version has no flexibility nor genericity at all, but it meets all other requirements.

6. A missed opportunity in software architecture

The collaboration continued after this successful story in configuration management. In the mid 90's, Dassault Systèmes decided to move its products from Fortran to an object oriented design with C++. This huge redevelopment effort succeeded at the beginning of 1999 with a commercial release of CATIA V5. CATIA is made of approximately 50 000 C++ classes and more than 8 000 components.

In order to develop this complex product line, DS decided to create its own framework. The OM component model is a fundamental part of this infrastructure. This component model is similar to Microsoft's COM but includes more powerful constructs.

Just like other component technologies, the OM is quite difficult to understand and to teach. Using the OM requires experienced and skillful engineers. It introduces new conceptual entities programmers are unfamiliar with (e.g. OM components, bases, extensions, delegations, etc.). Moreover these concepts are described using low-level mechanisms like C++ entities, naming conventions, macros in source code, specific implementation patterns, etc. An additional issue is due to the lack of centralized description for these concepts. Due to concurrent development within the virtual software factory, information about a single OM entity is often spread among a large set of different files, that may come from different companies. Though the introduction of the OM component technology was a major step towards the definition of the CATIA product line, managing this complexity quickly became a difficult task.

The collaboration between Adele-DS concentrated therefore on software architecture. The initial goal of was to review which recent advances in software architecture were applicable in the context of DS. A survey of existing ADLs was conducted to see if one of them could be adopted to describe the architecture of CATIA. None of the existing ADLs appeared to be satisfactory [20].

First of all, there were serious customization issues. Some concepts like the OM inheritance between OM components could not be described in the existing ADLs. Handling all the specificity of the OM was an essential requirement. In fact, most ADLs introduce high level concepts such as connectors, but this concept for instance was not perceived as being useful within the company.

It also appeared that the primary objectives of many ADLs did not fit the requirements of DS. Many ADLs focus on the verification or simulation of interaction protocols between components. Though this is an important topic this approach does not scale up and describing the behaviour of all components was not considered as feasible.

Finally ADLs are tailored to fit in a forward engineering process, but they usually do not take into account existing software. It makes no sense in a large company to maintain an architectural description if the link to source code is not maintained. In fact DS, just like almost every other company, is essentially code-centric. Moving the an architecture-centric approach was not even considered since it would require a major risk with an absolutely unclear return on investment.

To cope with this problem, we defined an architectural notation for the OM concepts, but took a reverse engineering approach. The architecture had to be extracted from the code, not the other way around. This process had to be fairly automatic to show immediate benefits. We developed a specific tool called OMVT [22] to explore CATIA at the architectural level. This tool offers different architectural views and analysis features with a custom graphical syntax. A special emphasis was put on the scalability and usability of this tool, providing for instance a clean and easy to use interface.

Various demonstrations were performed to show the tool to different actors in the company. DS software engineers were very positive: for the first time they were "seeing" the CATIA architecture. Comments were very encouraging: "it is very promising for a controlled definition of components"; "it provides both global and detailed views ... not available today without cumbersome browsing of many files ..." [21]. Although the OMVT tool is close to DS' requirements and has positive assessments, it is still not adopted in the company. The "last mile" is indeed a difficult step.

7. Discussion

Two different scenarii have been described: the successful adoption of Adele configuration management tool, and the issues in deploying the OMVT in the company. Actually, a closer look on these examples reveals that there are at least two main categories of tools: *critical tools* and *non-critical tools*. This characteristic obviously has an impact on its adoption by the company.

7.1. Adoption of critical tools

SCM tools are critical for large companies like DS. Adele was adopted because solving SCM issues was urgently needed and because no other tools with similar features were available on the market. In fact the adoption process was quite long. The TST ultimately implemented a specialized version of the tool from scratch.

The size at DS clearly plays a very important role. Almost all tools, including popular and effective commercial tools, suffer from serious issues when applied

at large at DS. DS has to tailor existing tools to their needs. This often implies special partnerships with tool vendors. These partnerships are fundamental in the adoption of a given tool.

In fact, if a tool is critical for a company, the company will deploy considerable energy in getting the tool and customizing it. If the company is large enough, developing an in-house tool is sometimes considered. One of the main qualities of Adele were its genericity and flexibility. They enabled DS to define and implement their own development process. The integration with existing tools in the company was not considered a primary issue because the TST could handle these problems. In fact, existing tools were integrated into the ADELE environment, which reflects the fact that SCM tools are always critical ones in large companies.

7.2. Adoption of non critical tools

Tools that do not have a direct and immediate impact on source code are usually not considered as critical ones. They are plenty of ways to avoid using a tool in a company.

Reverse engineering and architectural tools might fall in the "non critical tool" category (except when a major reengineering effort is needed, but this is not common).

It should be recognized for instance that, CATIA development and evolution is very successful even though DS does not have a clear vision of the full architecture of CATIA, at least in the ADL sense of the term. Communication among highly skilled teams reveals itself to be very effective in practice. Though the OMVT could represent a helpful tool, until now it has not been adopted.

Although it is quite difficult to determine which are the most important barriers to adoption of OMVT, it is clear that it is not currently a priority in the strategy of the company. Let us review however which technical issues should have received more attention in an ACSE perspective:

- *Deployment issues.* Deploying a tool at DS is clearly complex. Though the OMVT tool is quite simple to install on one computer, more attention should have been dedicated to automatic deployment. Another alternative would have been to integrate the OMVT in the intranet of the company since this solution would not require any kind of installation on client machines. This kind of solution is widely used at DS for other tools. The intranet is quite rich in terms of SE data.
- *Integration issues.* Communication is of paramount importance at DS. Providing architectural views of CATIA as live documents would have also greatly helped in the diffusion of the architectural notation and the corresponding concepts. An architectural view that cannot be annotated and shared among various

software engineers is not really useful in practice. Similarly an important aspect would have been to integrate the OMVT as a plugin in the DS toolset.

- *Customization issues.* Demonstrations and interviews among various actors of the company revealed very different needs. Many enhancements and customizations were required. We implemented 22 view points of software architecture, but though the tool is based on a clean object-oriented framework, adding and customizing new features still required significant development efforts. The ease of customization would have been a strong point, especially since just like in the case of Adele, the needs of the company are not clearly identified.
- *Evolution and continuity issues.* The evolution of CATIA V5 infrastructure is continuous. To continue to be useful, the OMVT should have been capable of evolving accordingly. In fact new architectural concepts are sometimes introduced while some others are removed. For instance, some interviews revealed a strong interest in taking into account the concepts of the “Feature Modeller,” that is additional layer built later on top of the OM.

Actually, one of the major problem we have faced is that we haven’t been able to achieve major TST involvement. No TST engineer has been assigned to the deployment, administration, support and evolution of the OMVT. Such involvement is a prerequisite for the adoption of a tool. Once again, since such kind of tools is not considered as being critical, their use is not considered as a top priority.

8. Conclusion

Adoption Centric Software Engineering addresses a very important issue in software engineering: how to cope with the fact that research produces many tools but that, most of the time, these tools are not adopted by the software industry. A large list of issues in tool adoption can be established. Some barriers are related to the organization and strategy of the company. Others are related to technological aspects.

Researchers should not expect to change the way companies are making business. Research provides tools while industry is looking for solutions to support their strategies. This constitutes indeed a very big gap.

Research should rather concentrate on concrete and technological aspects, especially since an actual impact is possible there. Tool integration is an important issue. It is however not clear if it always constitutes the most important barrier to tool adoption. Much ACSE research focuses on this issue and relies on the use of industry standards to improve the chance of adoption. This is a promising way, but this approach should not be considered in isolation.

In fact, the case study described in this paper reveals that the size of a company plays an important role in tool adoption. Adoption-in-the-small and adoption-in-the-large are quite different. End user, that is software engineers, are the natural targets in small companies. They will decide if the tool is worth it or not. In a large company, the tool adoption process is much more different. At least three different parties have to be convinced: the managers, the tool support team, and the software engineers. Even if the software engineers would not adopt naturally a tool, tool adoption could be decided at the managing level and implemented by the tool support team. The distinction between critical tools and non critical ones is important. If a tool is considered as critical, a large company will spend a vast amount of energy in deploying it in the whole company. In a company like DS, it is illusory to believe that a research prototype will be adopted as is. A close collaboration with the tool support team is a key to success.

Our experience also suggests that to be adopted in a large company a research tool should be generic and should support a high level of customization, at least in the initial phase. Actually a rewarding collaboration schema with a tool support team would be to help them to define their own requirements by using a generic tool. If the company is large enough and the results of the experiments are convincing enough, the company might implement later a specific tool using a full engineering power. Industrial strength tools and research tools do not play in the same category.

Our current research includes the design of G^{SEE} [23][24], a Generic Software Exploration Environment. This is a meta-model-driven environment. It enables a very fast integration of various kind of data sources and tools as well as the interactive definition of new architectural views.

Our work in component-based software engineering is also applied in this context. One of our goals is to build a component-based framework that enable the TST to build interactively they own environements by assembling and customizing very generic components. We do not refer here to component programming, but on the contrary to component assembly. The first approach still requires programming skills and a non trivial development effort. By contrast, component assembly refers to the *interactive* assembly and customization of existing components via an assembly tool. This tool could provide significant help to support these tasks by using for instance wizard-style dialogs and very interactive prototyping.

We believe that this kind of tool could considerably improve the relationships between research teams and the tool support teams. Research team would bring the generic framework, the tool support team would bring their knowledge about the company. The ultimate goal is to help large software companies to assemble their own SE tool suited to their specific needs.

9. References

- [1] A. M. Davis, "Why Industry Often Says 'No Thanks' to Research", IEEE Software, November. 1992
- [2] L.B.S. Raccoon, "Fifty Years of Progress in Software Engineering", ACM Sigsoft, Software Engineering Notes, Vol 22, No 1, pp 88-104, January 1997
- [3] S.T. Redwine, W.E. Riddle, "Software Technology Maturation", 8th International Conference on Software Engineering, pp 189-200, August 1985
- [4] H. Muller, "Adoption Centric Software Engineering", presentation at the Dagstuhl seminar on Software Architecture Reconstruction and Modeling, February 2003
- [5] A. Wasserman, "Tool Integration in Software Engineering Environments", LNCS 467, Springer-Verlag, pages 138-150, 1990.
- [6] S.P. Reiss, "Interacting with the FIELD Environment", Software - Practice and Experience, 20(S1), 1990.
- [7] S.P. Reiss, "The Desert environment", Brown University <http://www.cs.brown.edu/software/desert>
- [8] R. Holt, A. Winter, A. Schürr, S. Sim, "GXL: Towards a Standard Exchange Format", 7th Working Conference on Reverse Engineering, November 2000
- [9] OMG, "MDA: the OMG Model Driven Architecture", <http://www.omg.org/mda/>
- [10] OMG, "Model Driven Architecture - A Technical Perspective", ormsc/01-07-01, 2001
- [11] OMG, "Meta Object Facilities (MOF) Specification, Version 1.4", April 2002
- [12] OMG, "Common Warehouse Metamodel (CWM) Specification, v1.1", March 2003
- [13] <http://www.eclipse.org>
- [14] Dassault Systèmes, <http://www.3ds.com/>
- [15] J. Estublier, R. Casallas, "The Adele Software Configuration Manager", book chapter in *Trends in Software*. J. Wiley and Sons, 1994
- [16] J. Estublier, J.M. Favre, R. Sanlaville, "An Industrial Experience with Dassault Systèmes' Component Model", Book chapter in Building Reliable Component-Based Systems, I. Crnkovic, M. Larsson editors, Archtech House publishers, 2002
- [17] C. Hofmeister, R. Nord and D. Soni. *Applied Software Architecture*. Addison-Wesley Publisher, 2000.
- [18] IEEE Architecture Working Group. "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems". IEEE Std 1471-2000, October 2000.
- [19] P. B. Kruchten. The 4+1 view model of architecture. IEEE Software, 12(6): 42-50, November 1995.
- [20] Y. Ledru, R. Sanlaville, J Estublier, "Defining an Architecture Description Language for Dassault Systèmes", 4th International Software Architecture Workshop, 2000.
- [21] R. Sanlaville, "Software Architecture: An Industrial Case Study within Dassault Systèmes", PhD dissertation in french, University of Grenoble, 2002
- [22] J.M.Favre, F. Duclos, J. Estublier, R. Sanlaville, J.J. Auffret, "Reverse Engineering a Large Component-based Software Product", European Conf. on Software Maintenance and Reengineering, CSMR'2001
- [23] J.M. Favre, "A New Approach to Software Exploration: Back-packing with G^{SEE}", European Conference on Software Maintenance and Reengineering, CSMR'2002
- [24] J.M. Favre, "G^{SEE}: a Generic Software Exploration Environment", 9th International Workshop on Program Comprehension, IWPC'2001