Journal of Information Technology (2007) 22, 465–478
© 2007 JIT Palgrave Macmillan Ltd. All rights reserved 0268-3962/07 $30.00
palgrave-journals.com/jit

Teaching case

Re-engineering at LeCroy Corporation: the move to component-based systems

Julia Kotlarsky

Warwick Business School, Warwick University, Coventry, UK

Correspondence:
Julia Kotlarsky, Warwick Business School, Warwick University, CV4 7AL Coventry, UK.
Tel: +44-2476-524692;
E-mail: Julia.Kotlarsky@wbs.ac.uk

Abstract
This case study discusses two related aspects that are becoming increasingly important in today's software development practice: re-engineering of a monolithic system into a component-based system (the focus of this case) and globally distributed work. Component-based (software) development (CBD) involves (i) the development of software components and (ii) the building of software systems through the integration of pre-existing software components (developed in-house or procured from the component market). For companies involved in software development, CBD offers agility in design by basing software development on methodologies that support the recombination of reusable components, being an approach that rapidly expands product variation and sustains the build-up of product families. It also promises significant improvements in software development through shorter time-to-market and reduced development costs. However, being an innovative approach to software development that emerged in the mid-1990s, the adoption of CBD requires companies to re-engineer existing software systems (products) or to develop new systems from scratch using component technologies and to deal with additional challenges associated with the management of CBD. LeCroy Corporation was one of the early adopters of CBD that decided to adopt CBD to gain competitive advantage in its markets. The case describes an actual situation at LeCroy Corporation, involving several decisions, challenges and opportunities faced by the managers of a globally distributed software development team over a period of time when they re-engineered a monolithic system into a component-based system.

Journal of Information Technology (2007) **22,** 465–478. doi:10.1057/palgrave.jit.2000099
Published online 15 May 2007
Keywords: component-based architecture; system re-engineering; globally distributed team

Introduction

In the software industry, component-based (software) development (CBD) is a relatively new trend. It emerged in the mid-1990s with the introduction of software component technologies such as Enterprise JavaBeans, Microsoft COM and CORBA (Peters and Pedrycz, 2000). CBD involves (i) the development of software components and (ii) the building of software systems through the integration of pre-existing software components (developed in-house or procured from the component market) (Kotlarsky *et al*., 2007). CBD was presented as a revolutionary approach to software development, promising dramatic improvements in software development, such as the endless possibilities to reuse and recombine software components, shorter time-to-market, better quality and reduced development costs (Crnkovic and Larsson, 2002; Huang *et al*., 2003; Vitharana, 2003). In this regard, CBD offers agility in design by basing software development on methodologies that support the recombination of reusable components, being an approach that rapidly expands product variation and sustains the build-up of product families (Kotlarsky *et al*., 2007). In the light of these potential benefits, many companies involved in software development considered the adoption of CBD. However, being an innovative approach to software development, the adoption of CBD requires companies to re-engineer existing software systems (products) or to develop new systems from scratch using component technologies that required significant investment (budget and time-wise) and presented additional difficulties associated with the management of CBD, such as a lack of stable standards, lack of reusable components and problems related to the granularity and generality of components (Crnkovic and Larsson, 2002; Vitharana, 2003). Furthermore, empirical research on CBD has shown that 'it often took longer to develop a reusable component than to develop a system for a one-off purpose' (Huang *et al.*, 2003). It is argued that the benefits are difficult to achieve in the first place, and that they cannot be achieved immediately, but only in the long run (Crnkovic and Larsson, 2002). Therefore, companies considering the adoption of CBD have to carefully assess any challenges lying ahead and find ways to deal with them to achieve the potential benefits of CBD.

LeCroy Corporation was one of the early adopters of CBD. The company decided to adopt CBD to gain competitive advantage in its markets. System re-engineering at LeCroy took place in a globally distributed environment. Therefore, in addition to the challenges related to the adoption of CBD, the global software development team of LeCroy had to address several constraints associated with globally distributed work, such as distance, time-zone and cultural differences (Carmel, 1999).

This case study describes an actual situation at LeCroy Corporation, involving several decisions, challenges and opportunities faced by the managers of a globally distributed software development team over a period of time when they re-engineered a monolithic system into a component-based system. The case takes a historical perspective, describing the transitions in a software architecture from the mid-1980s until early 2002:

- *Period 1*: a monolithic system
- *Period 2*: a modular system (the monolithic system is broken into three modules)
- *Period 3*: a component-based system (components are reused across a number of products)

For each period, this case study covers three themes:

- *Product architecture* – advantages and disadvantages of the architecture at that specific time period. Opportunities, long- and short-term goals, and strategic decisions made by managers.
- *System re-engineering* – human and technological aspects related to system re-engineering and adoption of new technologies at LeCroy.
- *Global distribution* – human and technological aspects related to managing globally distributed teams.

The company

The LeCroy company was created in 1964 by Walter LeCroy, a physicist. He launched a small business in an old laundromat in Irvington, New York (NY), USA. The new business, LeCroy Research Systems, was quickly recognized as an innovator in instrumentation. In 1972, the company established an instrument design and production facility in Geneva, Switzerland. In 1976 (after changing locations twice in 1965 and 1967), the corporate headquarters moved to its present location in Chestnut Ridge, NY.

Initially, LeCroy developed technology to capture, measure and analyze sophisticated electronic signals in a stringent scientific environment. In 1985, the company began transferring this technology to a popular line of general-purpose instruments. Growth in the commercial test and measurement market really took off when the company introduced its first digital storage oscilloscope (DSO) products. Since that time, the core business of LeCroy has been the design and production of oscilloscopes and oscilloscope-like instruments: signal analyzers, signal generators and others.

An oscilloscope (Figure 1) is a laboratory instrument commonly used to display and analyze the waveform of electronic signals. In effect, the device draws a graph of the instantaneous signal voltage as a function of time. Oscilloscopes are used extensively for industrial, scientific and medical purposes (e.g. for design and testing in the high-tech industry, in research labs and universities).

By the early 1990s, LeCroy had become one of the world's largest designers, manufacturers and distributors of electronic measuring instruments for both scientific and industrial (commercial) applications. Other companies developing test and measurement equipment, and the main competitors of LeCroy, are Textronix and Agilent.

During the last 20 years, LeCroy has opened a number of sales offices in Europe (in France, Italy, Germany, Switzerland and the UK): these offices are responsible for sales in all European countries. There are also several offices in Japan, South Korea, China and Singapore (see LeCroy organizational structure in Appendix A). LeCroy now employs more than 400 people worldwide. In 2006, the company reported annual revenues of more than $160 million.

Figure 1 Digital oscilloscopes of LeCroy: WaveMaster (left) and Serial Data Analizer (right).

At present, oscilloscopes are designed by teams located in Geneva and NY: the hardware is designed in NY, and the software is designed by a team distributed between NY, Geneva and Maine (USA). In Geneva there are 14 software developers, 13 in NY and one (the main software architect) is telecommuting from Maine. The manufacture of all oscilloscopes is done in NY.

This case study focuses on the software development team, which is globally distributed between NY, Geneva and Maine.[1]

17–19 December 2001; Lecroy Office, Chestnut Ridge, NY: Snapshot

Monday 17 December 2001, 1 week before Christmas, five working days before the Christmas vacation. Downtown Manhattan is crowded: it seems everybody in NY is busy buying presents for Christmas, for family, friends and colleagues. At LeCroy office in Chestnut Ridge, NY, the atmosphere is very similar to what is happening in the city: last-minute preparations. Everybody is busy. But this is not only because of the approaching Christmas, or maybe not because of Christmas at all. LeCroy is preparing the launch of a new product. Developers are mostly busy with bug fixes, and managers are busy with the last preparations before the release of the new product: Aladdin, a Windows-based digital oscilloscope, the first of the new generation of oscilloscopes based on Windows. The launch of Aladdin (officially WaveMaster) is scheduled for 10 January 2002, right after the Christmas–New Year vacation.

There is still some work to be done before the product can be released. This causes tension among the software team. However, the overall atmosphere in the software team, and also among all the LeCroy employees in the NY office, is very cheerful. And there is something to be proud of. During the last couple of years, companies producing test and measurement equipment have been competing in creating a Windows-based oscilloscope. By the end of 2001 it had become clear that LeCroy was the first to offer the market the new product based on Microsoft COM technology that enables data transfer, processing and integration of external algorithms up to 100 times faster than other scopes.

Back to this, Monday, 17 December 2001: Larry (Director of Software Engineering who is responsible for all the software at LeCroy) and Anthony (Chief Software Architect of the company and head of the Geneva software team) who was visiting the NY office during this period, both seem to like the idea of telling the story of how they proceeded from the old oscilloscopes LeCroy produced in the mid-1980s to Aladdin – the first oscilloscope based on the component-based architecture. This is the story of complete system re-engineering: from a monolithic to a component-based system. As Anthony commented on their decision to fully re-engineer the monolithic system: ‘we did something which very few software companies have a chance to do.’

Period 1 – history: mid-1980s to mid-1990s

How development was organized: organizational perspective

From the organizational perspective, historically the development of oscilloscopes was distributed between Geneva and NY. Three teams – software, hardware and manufacturing – were involved in the production of oscilloscopes. Initially, all three teams were located in NY and Geneva and worked together from these two locations.

Since the mid-1980s, the software for the oscilloscopes has partly been developed in Geneva, and partly in NY. Initially there were about five–six people in Geneva and five–six people in NY, and these two teams interacted frequently. Originally, interactions involved shipping tapes and floppy disks between the two sites. Later, the software team used a 2400 baud modem to interchange files. The interactions progressed as the teams acquired e-mail. Later on, they replaced modems with a wide area network (WAN) connection between the two sites.

Until 1999, LeCroy had manufacturing and hardware development teams in both Geneva and NY. However, some key hardware engineers in Geneva had left LeCroy in 1998. At that point, LeCroy would have needed to replace the engineers and re-establish the hardware team in Geneva. At the same time, LeCroy was reexamining why they were manufacturing in two locations – especially in two very expensive locations. The decision was made to stop manufacturing in Geneva and consolidate all manufacturing in NY. Since hardware design is closely related to manufacturing, a decision was made not to rebuild the hardware team in Geneva but instead to close down hardware engineering in Geneva and bring as many of the engineers as they could to NY. Some key manufacturing personnel moved to NY to run manufacturing, and seven people (about half of the group) from the hardware group moved to NY. Four of those people have remained and now permanently live in NY.

It was not as important to have the software team physically located close to manufacturing. Also, since the software team had developed very good ways of working together over distance, it was decided to leave the software team in Geneva. At that time there were about 10–15 people in the Geneva software team and the same number in NY.

In the software team, one of the advantages was that a couple of members of the Geneva software group had originally worked in NY. Anthony was there: he had started in NY in 1986. Another senior person, Martin, the chief scientist currently based in Geneva, had worked in NY for many years (since the mid-1970s). Anthony is talking from his experience:

> To take people with experience, I think, working in the group, and then move them into another group, is a good way to seed the other group, to make sure that everything works together.

In the late 1998, Jon, the main software architect, who had worked in NY since 1990 and had spent a year (1998) in Geneva, moved to Maine (USA) from where he continued to work for LeCroy by telecommuting. Since then, LeCroy software team has worked from three geographical locations: Geneva, NY and Maine.

Looking from the perspective of cultural differences, the software team is multinational:

> There are many different nationalities in the team. If I put everyone together from Geneva, NY and the Maine – we

have Swiss guys, American guys, British guys, German guys, French guys, Spanish guys, Polish guys, Indian guys, Chinese guys, Israeli guys, Russian guys (Gilles).

Previous products

From the mid-1980s, starting with the first scopes of the company, and for about 10 years, the software for oscilloscopes was written in the C-programing language, using structured design techniques. The system was a typical embedded system. Larry explained: 'it was a simple "loop" operating system, which controlled the flow of the waveform data from the acquisition system to the display.' Initially it was based on a home-grown operating system (not an off-the-shelf real-time operating system).

In the mid-1990s LeCroy started using C++ programing language, but every time LeCroy came out with a new model with a new feature it was built on top of the software that was initially developed in the mid-1980s for the first scopes (internally referred to as 'Core' software). This Core of software grew and became a platform for all scopes that LeCroy shipped up until the end of 2001. As a result, the software for scopes turned into a huge monolithic program. With every new version of the scope, the system became more complex and every time it proved more difficult to add new features to it because it was becoming more difficult to identify all the areas (files definitions and source code in the monolithic program) that would be affected by the new feature. And when identified, additional work was required to 'plug' the new features into the existing program. Moreover, LeCroy had multiple teams working on the acquisition systems of different scopes at the same time.[2] In addition, there was some source code from the Core software that every team used and sometimes changes were required in this common chunk of code. As a result, it was becoming more difficult to deal with dependencies between the teams. Furthermore, LeCroy also wanted another company in Japan, in partnership with LeCroy, to write their own acquisition system. However, LeCroy did not want the Japanese company to have access to the key intellectual property of LeCroy – the Core software.

Larry commented on the situation LeCroy managers faced at that time: 'We needed to split the existing platform up into at least three different sections that could be linked together in the end.' So, LeCroy started a project called ORACLE:[3] they took the monolithic program and broke it up into three components.

Period 2 – ORACLE project: January–June 1997

The main objective of the ORACLE project was to divide the original scope software into three modules (see Figure 2). One of the modules was the *operating system*. For the original scopes LeCroy actually wrote the operating system. As part of the ORACLE project, LeCroy switched to VxWorks as the operating system. VxWorks is a commercial real-time operating system. It replaced the home-grown operation system LeCroy used to have for previous scopes. Another module was what they called *Core*. Core contained the functionality common to all oscilloscopes (analysis and display capabilities), regardless of the operating systems. The third piece was the *acquisition system*. Each scope has a different acquisition system (the acquisition system is the part that changes every time a new scope is produced).

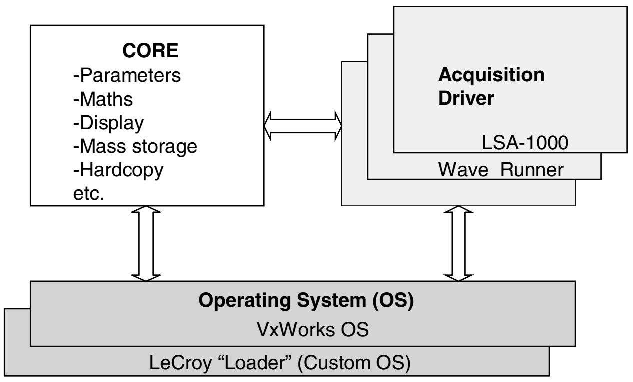


Figure 2 DSO architecture after the ORACLE project (schematic).

Four people worked full time on the ORACLE project – three software engineers and Anthony (he was the project manager). According to Anthony, ORACLE 'was basically the splitting of this 15 years old mass, this big ball of strength, into 3 major components [modules].'

As the ORACLE project plan states: 'the idea is not to start from scratch and re-write our Digital Storage Oscilloscope (DSO) software, it is instead a plan to reconstruct and reengineer our existing DSO software so that it is more modular, extensible, and maintainable.'

For LeCroy ORACLE was an interim solution, before they could come up with an oscilloscope of the next generation. The ORACLE project 'was a kind of a stop-gap or solution that would buy us a few more years anyway – about 5 more years' life of this platform' (Larry). And during these 5 years they need to come up with a new platform, new product.

The ORACLE project made it possible for LeCroy software engineers in NY and in Geneva and their development partner in Japan to work on sets of acquisitions in parallel, without encroaching on each other's developments. Sometimes, though, this was difficult. In some cases, where there was a common code, the teams had to manage the check-in and check-out manually:

> If the guys working in NY made changes in the acquisition or core files that were being modified in Geneva, at the same time, they would manually have to merge the changes (Larry).

As a result of various changes and new versions of scopes, there were multiple versions of software that needed to be identified, kept track of and managed (e.g. for fixing bugs reported by customers and the maintenance of the product, it is necessary to know which version of software was installed in which model of oscilloscope). Moreover, at any time, both test and development versions and working versions of the software could exist.

These multiple versions were managed using a Version Control System (VCS). The VCS uses version trees to keep track of all new and past versions of the software. LeCroy had different trees of their VCS. Each acquisition system could either be a separate tree or a branch of the tree. And the Japanese development partner had their own separate VCS where LeCroy would put the object files for Core, so that in Japan they would just work on the acquisition files.

To limit access to the source code (Core files), LeCroy gave the Japanese partner object files for Core and the source code to one of the acquisition systems as an example on which to base their acquisition system. In this way, they did not reveal any internal Core code. The Japanese partner developed a scope simply by replacing the acquisition system software. Based on the modular architecture created during the ORACLE project LeCroy, with the Japanese partner, developed and shipped several products. At the same time, other teams (sub-teams of NY and Geneva teams) worked on acquisition systems for other oscilloscopes which were shipped by LeCroy.

Summer 1997

The lessons learned and the development of a long-term vision for the product and the development approach

LeCroy managers realized that ORACLE was just a beginning in the re-engineering of the monolithic architecture of the previous scope into several pieces (modules) that could be linked together. As Anthony summarized it, it was: 'a start on this road towards a system that really was Object-Oriented and reusable and modular and all these good words. That is what ORACLE was.' As the ORACLE project approached completion, Larry, together with the group of people who had worked on the ORACLE project (Anthony was one of them) decided that 'now we need to sit down and figure out a longer term vision – what we are going to do down the road.' There were several decisions to be made at this point.

The product development team was to conceptualize the next generation of scopes. On which technologies should it be based? How should it operate? The market for testing and measurement equipment is highly competitive, so to be better than a competitor 'we have to stay at the top edge of the technology.' There were many advantages in moving towards a scope that ran Windows as its operating system. Larry explained:

> There was a desire to be Windows-based because we knew that our competitors were going towards Windows: we knew it provided a lot of advantages in terms of user interface and all the I/O (i.e graphics, network, etc). We would be able to develop on PCs and won't need a cross compiler if we develop using standard Windows tools such as Microsoft Developer's studio which has a native compiler and debugger.

Furthermore, using Windows-operating system for oscilloscopes and oscilloscope-like instruments would enable LeCroy software team to purchase commercial (software) libraries and objects instead of writing their own software, and would allow users (LeCroy's customers) to add and run third party applications, such as Matlab, Mathcad and Excel.

Therefore the question was 'How do we create a Windows-based scope?' One of the options that LeCroy had was to take their existing scope, keep its source code and on the top of this to write a simple program to provide the user interface. It actually turned out that this was exactly what their leading competitor (Tektronix) had done. The LeCroy managers said: 'we realized that in order to grow further, we need to continue the division process, we needed to split it into much finer kind of components.' But there were aspects of the original design, the original architecture, that were very difficult to remove:

> In the 15 years working with this architecture we knew that in order to really move forward, there were so many constraints in there, we just wanted to throw them all away, we wanted to start with a clean page. [...] And we got to a fork in the road where we had to say, 'ok, do we want to do all that with Oracle, which was evolutionary (we took what we had and carefully pieced it up into 3 modules). Or do we cleanly start from scratch?' And that's what we did. [...] So we threw out all the old stuff and we started afresh (Anthony).

Period 3 – Towards a component-based architecture

Studying product requirements and available technologies: Summer 1997

In the summer of 1997, when ORACLE was completed, LeCroy started the Maui project. 'Maui' stands for Massively Advanced User Interface. The goal of the Maui project was to develop and implement a platform for the next generation of Windows-based software for digital scopes that would be component-based, and thus would enable reuse of components in a number of products:

> The whole idea is that we can take the bunch of different components and create a different instrument, within weeks is kind of optimistic but, within a few months rather than in a few years (Larry).

Schematically, the major phases of the Maui project are presented in Figure 3.

The first step was to work with marketing and other people in the company to determine the requirements for the new system. In parallel, Anthony, Jon[4] and Joe[5] investigated what component technologies were available for LeCroy to work with:

> We needed to be on Windows, so that limited our options. We did have several discussions on the possible choices. In particular, we knew of Microsoft COM, and had an idea about what it could do for us. But we were not convinced that it could be used to accomplish our goals. We were worried about JAVA because it does things in the background (e.g. garbage collection, which is the process of freeing dynamically allocated memory that is no longer used. Since it happens in the background, it can cause unpredictable time delays). After looking at the choices, we decided to do the feasibility

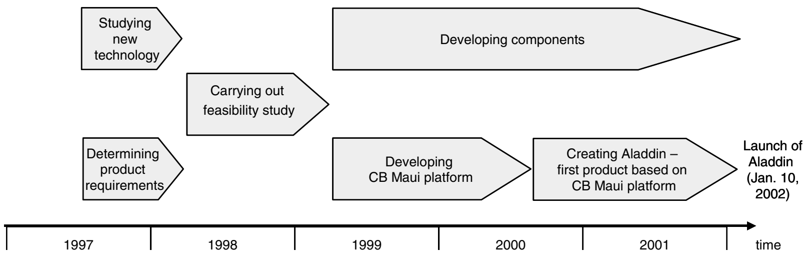

Studying new technology
Developing components
Carrying out feasibility study
Determining product requirements
Developing CB Maui platform
Creating Aladdin – first product based on CB Maui platform
Launch of Aladdin (Jan. 10, 2002)
1997
1998
1999
2000
2001
time

Figure 3 Major chronological phases of the Maui project.

> study with Microsoft COM and only pursue other choices if Microsoft COM didn't meet our goals … It did) (Larry).

One of the major issues they were concerned in relation to Microsoft COM was that having separate objects could add considerable processing overheads to the system, and therefore might not give the performance they needed.

Another concern was the granularity issue: 'If you use this technology, where would you draw the line? How many objects are reasonable – do you take everything about the display and put it in one object or you break it up into many smaller objects?' (Larry).

What components should be built so that they could be used in most of the scopes?

Furthermore, Larry and Anthony had to deal with people-related issues: 'How do you take all the guys that we had – pure embedded programers – and teach them all about Windows at the same time?' (Larry). The LeCroy managers had to address all these issues.

Team-building exercise and learning new technology: August 1997

Larry and Anthony organized a conference in the Alps where they took all the software developers from NY and Geneva. Learning Tree, a company that offers training courses, was contracted to give the course: a customized concentrated course that was given in 1 week in August 1997. The course provided training in Windows and COM programing. At that time 'everyone was still working on other projects but it was sort of an introduction to them for this stuff.' Larry, reflecting on the goals of the course, said:

> We all got together in the mountains of France and it was a real fun week, and it had two purposes: one was to teach us all this new technology. The other, which was equally important if not more important in some ways, was to really try to build relationships between people. Because what we found over the years is, whenever people had worked face-to-face, even if it was only for a few days, the fact that you could put someone's face to the caller, made it much easier for someone to pick up the phone and ask the question than if it was just a name that you heard. And so, it was sort of a team-building experience as well, and that was the real fun aspect to it.

Most of the software engineers in NY and Geneva did not know each other. Larry and Anthony tried to make sure that the members of both teams interact: 'we increase the possibility that they are really getting to know each other' (Anthony). The course in the Alps was an occasion to get all of the software team together. Anthony said, about the importance of such interactions:

> It makes a big difference, when the guys know each other. And more importantly, when the guys trust each other and know what their capabilities are. I think that makes a huge difference. It is because there are very clever guys in the group. And when you get fairly clever guys talking to each other, there needs to be certain degree of trust, I guess respect is maybe a better word, for each other. And where that is lacking, there is really a communication problem. And when there is a lot of trust and respect, people get on very well, they are very productive.

Anthony observed:

> Meeting has got a lot to do with it. In fact, I would say that some of our most valuable time spent meeting is probably in the local bar rather than in the meeting room. Because most of getting to know each other, getting respect, happens over a few beers. And that develops into professional aspects. I think that is a sort of an important thing, very important thing. And that was the idea, one of the ideas, behind the conference in the Alps, was to get people in an environment where there was plenty of time for that.

Meeting and working side by side even for a very short time ensures significant progress in building relationships between remote counterparts: 'We understand each other better now. And it is a learning experience: learning about how we work. About how we work and about how we communicate,' Adrian, Web master of LeCroy, said. Adrian works a lot with all LeCroy locations in Europe and with the office in Japan.

Further study of new technology – learning strategy: Autumn–Winter 1997

After the software conference in the Alps, where all the developers had an introduction to internal Windows and Microsoft COM programing, the software managers assigned a small team to work closely with a consultant in COM to study the COM technology in depth. Their goal was to learn all about COM while applying COM to oscilloscopes (trying to create an oscilloscope prototype using COM). The small team worked with the consultant over several rounds of meetings (question and answer sessions). First, the small team worked on different areas of COM and then they had the consultant come in for a couple of days. As Larry explained: 'And they [the small team], having got their hands dirty in it, had a whole list of questions that they needed to sit down and get answered.' The small team worked with the consultant for several days to answer all those questions and then they developed a list of things that they wanted to accomplish for his next visit a month or two later. They did this three times: 'The third time he came, we started asking him questions that he couldn't answer. And that's when we knew what we were talking about' (Larry).

Soon a key weakness of Microsoft COM became apparent. There was no proper way to manage components and their dependencies. With COM they could create components, but there was no facility to manage inter-dependencies between components and related files. This was not a problem as long as the number of components was small: in this case, the dependencies could be modeled and understood visually. However, when the number of components became hundreds, visual understanding was no longer an option. Components created in COM are connected into 'projects' under Visual Studio. As Anthony explained:

> Imagine building one DLL in one project under Visual Studio. It is very easy to do. Building two or three project DLLs that depend on each other is fairly easy to do. Building 300 or 500 of these things is impossible.

Therefore, during one of the meetings with the COM consultant, he was asked about development environments: 'How do people develop projects with a lot of COM components?' (Larry). There was no good answer to this, at least not at that time. What the consultant said was: 'While you build your components, you register them, and then you debug them… .' 'We knew that, we had a lot of people working on it, we needed a better answer than that,' said Larry. LeCroy software managers also went to a Microsoft conference and asked one of the Microsoft developers how they (in Microsoft) managed the components dependency problem. 'He told us they already had a solution, an internal tool but they never shipped it,' Anthony explained.

Feasibility study for a Windows-based scope – proof of concept: January 1998–February 1999

In January 1998, four people were assigned to carry out a feasibility study, which took about 1 year. They created experimental prototypes and then performed experiments to try various things and test out various concepts related to the scope's functionality. It was understood that these prototypes would only be for experimental purposes and would not become a production code: 'At the end of the year we took everything they did, basically, and threw it out. And we started again knowing what we wanted to do,' Larry explained.

The year of the feasibility study brought Larry and Anthony to the point where they said: 'we believe we proved that we can do what we need to do' (Larry). They were comfortable with the major review at that point.

The next step in the feasibility study was to check if they could get the performance (i.e. fast enough processing of data and display) they needed using Windows and the COM architecture. Larry explained how the performance of the COM-based solution was verified:

> COM has different threading models, so we looked at the inter-process communications times between components for each of the models and components of different models. We also looked at the time required for the different types of marshalling across the component interfaces. These were key in determining how to pass 'chunks' of data from one processor to another very efficiently.
>
> Additionally, we tested the different methods that Windows provides to display things on the display. We tested 'interrupt latency' – time from a hardware interrupt till the software can respond. For example, if a user pressed a button on the front panel, how long did it take to notify the code which needs to respond.
>
> We also investigated how to make 'lightweight' widgets which could be embedded in our dialogs for the user controls and corresponding 'Cvars' (control variables) to keep that state of each control.
>
> Another area we looked at was could we use VBScript to save/recall the state of the instrument, from how we could actually extract the state of the control variables to how long it took to save and recall.
>
> There were many other experiments run to understand the time required to do things on Windows as well as the variations in time (since each process only gets a slice of the processor's time). We also looked at the effect of changing the priority of processes.
>
> A lot of the concepts developed in this study are now patented by LeCroy (e.g. CVars, streaming processors, etc).

Once LeCroy managers were sure that the concept would work, they started working on the high-level architecture using what they had learned during the feasibility studies.

However, the feasibility study could not be completed without a solution for managing Microsoft COM components:

> Other things we looked for were tools and things to help to deal with a large component system. We didn't have any and we couldn't find any at that time – to manage the building and the inter-relationships of the components (Larry).

Early in 1999, Larry and Anthony assigned one or two software developers who spent a few months working on a tool for LeCroy to manage the COM objects and their dependencies: they called it *COM Project Manager* (*COM-ProjMgr*). This tool is described in greater detail further in the case study.

Development of a component-based Maui platform: February 1999–December 2001

By February 1999, the feasible concept had been found and the first version of the COMProjMgr for managing dependencies between software components was in place.

The attention now shifted to the development of the component-based Maui platform.

Division of work
Development of the Maui platform started in early 1999 (see Figure 3). There were only a few people who developed the basics of the new platform: Anthony and two more developers in Geneva and Jon from Maine (at that period he spent 1 year in Geneva). Anthony describes the assignment in the following way: 'What happened was, part of it was based on who was free at that time, and part of it was where the expertise was.' Anthony was managing the ORACLE project, and the logical step for him was to move onto Maui. He happened to be based in Geneva at that time. The software team in Geneva was responsible for most of the core code in the oscilloscope: these were the developers who wrote the original code 15 years ago: 'So they were also the natural guys to work on the defining the next generation' (Anthony).

As additional people started work on the new platform, each worked on different parts of the system. Gilles explained: 'So we have kind of specificities, we know better one domain then another one.'

Planning of the Maui project
Typically for any project carried out at LeCroy – whether it is adding a new feature, development of a new product or of an internal tool – the project phases that are defined and followed are a Proposal phase, a Planning phase, an Implementation phase and a Deployment phase. There are work flowcharts defined for each of these phases, describing the steps to be taken, responsible roles, and key milestones and deliverables associated with each milestone (e.g. documents such as proposal template and preliminary design review guidelines). At the same time, these flowcharts are generic and flexible enough to accommodate everyday dynamics. These flowcharts were designed some years ago by Larry, Anthony and some other senior managers. Within LeCroy, they are used as guidelines for project planning. However, 'especially with a project this size [Maui project], we are not very good in sticking through this all the way through the project,' Anthony said. Larry explained:

> The problem we find in huge projects in particular – there are so many dynamics – things dynamically changing on any given day. If you try to fully maintain the project at micro level, it would be a full-time job for someone. Typically what we used to do, and what we still do for some projects, is to break it down into three-week tasks.

Going back to the time when the Maui project started:

> It was more of a research project. When the Maui project started, we didn't really have a product in mind, not in the sense of the product that you can ship. But we knew that we wanted to use this [Maui architecture] on several products which would be defined in the future (Anthony).

It should be recognized that the Maui project represented a family of products. However, the same steps – proposal phase, planning and implementation – were used at a high level for this project, but they 'were not really due to any particular product' (Anthony).

What is Maui? How to create a scope in Maui?
Maui is a *platform*, consisting of a collection of hundreds of *Maui components*[6] that can be used to create different instruments (applications). However, these components are not enough to create an oscilloscope, or oscilloscope-like instrument: a Maui-based oscilloscope consists of large numbers of Maui components (most of them common to all scopes) that are integrated through *standard Maui interfaces*[7] with Acquisition and Application systems. Figure 4 illustrates schematically the Maui product architecture. A specific oscilloscope product such as Aladdin (officially called WaveMaster) or X15 can be constructed by integrating the components from Maui with an Acquisition system and designing the user interface for a specific application (Application system). For example, an Aladdin scope would be built by combining the Aladdin Acquisition system and Aladdin Application with the components selected from the Maui components. The same would apply for another product called X15: it requires X15 Acquisition and X15 Application and components from Maui (see Figure 4).

There are four types of components in products based on the Maui architecture. One category is called *processors*: mathematics functions, there are hundreds of them, one component per functionality. The second category is the *core components* that serve as an operating system for a scope. The third category of components is *Graphical User Interface (GUI)* components, which are combined to provide the user interface (connect to and control an Application system shown on Figure 4). They allow the systems to work together, and provide the basic instrument capabilities. And finally there are the components that comprise the *acquisition board driver*. These are responsible for controlling the acquisition hardware (part of an Acquisition system shown on Figure 4).

The components are written in C++ and the interfaces between them are in COM. Maui describes these interfaces, they are also part of the Maui architecture. Anthony explained:

> I guess, really the root of Maui, are these standard Maui interfaces which describe how these components talk to each other. That is really the heart of Maui. If you want to make a component for Maui, whether it will be something to display waveforms, to control the front panel, an

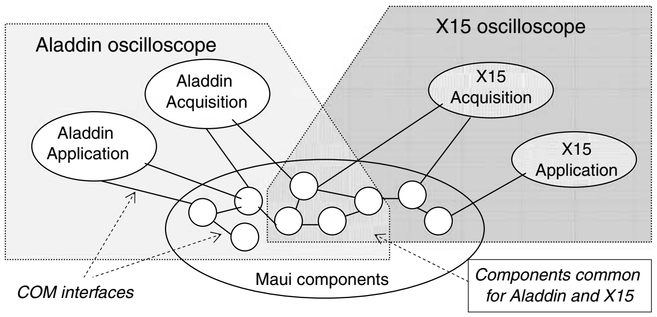


Figure 4 Maui product architecture (schematic).

> acquisition system, any of these things, in order to integrate them into the system and to attach them to the rest of the system, they have to implement or use one of the Maui interfaces. It is a bunch of standards, and it is a tool kit.

Switching to a new platform – bringing people 'on board Maui': Spring 2000–November 2001

When the Maui platform was developed, LeCroy still had to produce oscilloscopes based on the old system (including new versions of those oscilloscopes). When the development of the new platform started, everybody was still working with the old system. Gilles explained: 'When we started to do the transition from the old scope to the new scope, we didn't move every worker at once into the new platform, because we still had to develop for the old system.' And at the same time they had to teach all software developers how to work with the new platform.

Basically, there were two major dilemmas related to how to make a smooth transition from old product architecture to a new Windows-based scope. One was: in the transition phase, how to develop and produce oscilloscopes based both on the old system and the new (Windows based) scopes in parallel. The other dilemma LeCroy faced was how to move people on to the Maui project so that they could develop in Maui and (hopefully) be as productive as they were with the old system.

After the basics of the Maui were developed, Anthony explained:

> We started taking away people from existing products and moving them onto the Maui team. This happened first in Geneva until everyone in Geneva was involved, and then in the last year or two years [2000-2001] – we started bringing people on board Maui in NY.

To make the transition easier, Anthony, together with others who initially developed the Maui platform, created a document 'Maui Software Developer's Guide,' a Bible for developers coming on board Maui. This document explained the Maui Software Development Philosophy and 'describes the complete software-development process for the Maui Software Platform. It is intended that a developer new to Maui development can follow the directions in this guide to configure a "virgin" workstation for Maui development, create Maui Components, debug them, and evaluate their performance' (quote from the 'overview' section of the Maui Software Developer's Guide).

> One of the problems we had is: that our old system was a heavily embedded system based on embedded operating systems and embedded compilers. And moving those developers into Maui – using tools like Visual C, things like Rational Rose for the UML diagrams, just when everything changed, everything that we were used to and lived in for years – changed. So this Guide is explaining how to move into this new world (Anthony).

How did people perceive the new platform and that they now needed to work with it? 'Some loved it, some hated it,' Anthony said, 'but now, they all I think love it.' He continued:

> The last software guy in this building came on board Maui a couple of weeks ago [in November 2001]. It's an interesting or it's a difficult step for a developer to make when you were the master of your environment for such a long time, and you understood the entire system (and it is – we are talking about half a million lines of code[8]). These guys knew this stuff [the old system], this was their world for 10–15 years, and all of a sudden someone says 'forget all that, we are going to go to this new place which is completely different.' And it is using some standards by Microsoft, that we didn't create and that's not perfect but we have to live with them. And, everything that they were used to day-to-day – changed. Some guys accepted that very, very quickly. Some guys were 'up-and-running', maybe 'climbing' the learning curve within a few weeks. Other guys, they took longer. Somebody from the original senior guys are still not really up to speed in this new environment – they never will be as productive as they were on the old stuff. So the younger guys find it a little easier, they came up to speed literally in weeks (Anthony).

How team environment is maintained

In Geneva, in the summer of 2000, when the majority of the people started coming on board Maui, Anthony started 'around the table' meetings, typically once a week, with all the developers in Geneva. Anthony described this:

> We go to a meeting room and do an 'around the table' where I start with generally what is going on in the company and the project. And then we go around the table and everybody describes what they did the week before, what is holding them up, what they are planning to do the next week. It is to make sure that they all are plugged into the project.

The philosophy that Anthony follows is:

> We generally want that everyone knows what everyone else is working on. And if someone is held up because of a particular problem, somebody else may have a solution. So it is just a way for our guys to coming to structure.

Anthony visits NY about four or five times each year. When he is in NY, he still holds his 'around the table' meetings with his team in Geneva, except that he conducts the meetings via videoconferencing.

In NY, team meetings for the NY team happen less often, partly because the engineers are working on very different projects. They are not as regular and not of the same structure as meetings in Geneva. Anthony commented:

> What happened in Geneva is that among the guys there is a natural feeling that they are kind of unplugged from the rest of the company. Because it is an outpost! In order to handle that we organise regular meetings to let people know

what is going on in the company, what everyone else is working on. It is a big help. Every several months we have a transatlantic videoconference with the software guys in NY and Geneva. It helps everyone, I think, to feel that we are working as a team and that they are part of the LeCroy team.

Frequent visits by Larry to Geneva (about four or five times a year) also help to maintain the team environment between the NY and the Geneva group.

More recently, as the team in NY started working with Maui, every once in a while, usually once every few months, Anthony and Larry have organized trans-Atlantic videoconferences, through which all software developers in NY and in Geneva get together to discuss progress.

How knowledge exchange and coordination between sites are organized

Larry and Anthony travel back and forth between NY and Geneva regularly, for short periods, and usually twice a year for a month or more. Occasionally, they have developers from Geneva coming to NY or from NY going to Geneva for a week or two:

And we even have a few cases where we put someone over, we have one guy [Gilles] right now who is spending a year here from Geneva. And that is real useful sharing experiences and stuff (Larry).

Gilles has been working for LeCroy since the early stages of the Maui project. He is one of the five people who developed the basis of the Maui platform. He knows all the basics and the background of the platform. Therefore, in the summer of 2001, when the last group of people in NY were supposed to start working with Maui, Gilles joined the NY team for 1 year. He mainly provided individual help for software developers. However, in the beginning, when Gilles came to NY, he gave a few hours of classes ‘for general purposes,’ as he said, about the Maui platform. Afterwards, when everyone got assigned to different areas, Gilles helped individuals ‘on demand.’ He explained:

To do their task they come to me to ask some questions – first how to work with the Maui platform, how to use the platform and then how to develop in it a new component.

When the NY team started to develop software using the new platform, Gilles was still in Geneva. At that time, only a few people in NY were working with Maui and they always had a number of questions about the new platform. So they were always in contact with Gilles in Geneva (or Anthony, or Jon in Maine, or the other two people who developed the basics of Maui). When more and more people in NY started working with the new platform, Larry and Anthony decided that Gilles should come over to NY for 1 year to facilitate the contact for everyone. Gilles explained:

I asked to come to NY because I wanted to learn English and to discover a way of working here [in NY], the company here [in NY]. Because it was in the right time for them to start working on the new platform and I have the knowledge that they needed here to start to work on the new platform. Both together, it was accepted.

During the time Gilles spent in NY, he was often contacted by developers from the Geneva team when they have questions or need his help. The team in Geneva had been working in Maui for some time and it was not difficult to help them from NY. On the other hand, for the team in NY, the local presence of the expert was important during the transition period.

The first component-based products based on the Maui architecture: December 2001

With progress in development of the Maui platform, a picture of a new product – a Windows-based scope – was emerging and becoming clearer.

It is only now [end of December 2001], with Aladdin, that the hardware development and the software development are coming together and we actually say – now we have a product (Anthony).

Aladdin[9] (officially called WaveMaster) is the first oscilloscope in the new generation of Windows-based scopes based on the Maui architecture. It was launched in early January 2002. At that time there were more products on the way, for example the X15, the PXI digitizer; and basically everything LeCroy is developing now is based on the new technology implemented in Maui.

Talking about products based on the Maui architecture, Larry says:

What is the product? That, I guess, is really the key. So the products are: we have X15 as a product, WaveMaster or Aladdin is a product. But most of the components are the same in both. There are literally hundreds of these components.

Because Maui the architecture has been designed to be a basis for all scopes, initially core components and functions that could be used in several products were built. When LeCroy started working on specific products, Aladdin and other Windows-based products (X15 and PXI): ‘Most of the people were focused on getting the first couple of scopes out and what we needed for those. This is how we determine what components to build,’ Larry explained.

Software development tools for CBD at Lecroy

The four basic tools the software team uses are COM-ProjMgr, Perforce, BugBase and SoftwareTestHarness for testing components.

Anthony and Larry are both of the same opinion: whenever possible they would prefer to buy a tool if they can:

Whenever we need a tool, we do try to buy it, but most of the time we don’t find a proper solution. Then we make our own, and this goes for most of the tools that we have (Anthony).

Of the main four tools, Perforce is a commercial tool, and COMProjMgr, SoftwareTestHarness and BugBase are all tools created by the LeCroy software team.

To support working in a globally distributed environment, everyone working with Maui uses the same methods and procedures, protocols and tools: 'all are identical, absolutely identical' (Anthony) in NY and Geneva. The tools and methods used (Maui standards) are described in the Maui Software Developer's Guide: it lists the tools used (Lotus Notes, Visual Studio, Perforce, ComProjMgr and Rational Rose), and explains how to create and debug components in Maui using these tools.

COMProjMgr is a tool used for managing interdependencies between components. As one interviewee commented, 'COMProjMgr manages the entire project': it knows the dependencies between all of the files in Maui, of which there were 5000–6000 files (for the end of December 2001), and the various projects,[10] the various components and related files. This is a kind of tree, a hierarchy of components, with each component normally creating the DLL in a dynamic library. The DLL files are grouped into categories such as Utilities, Display and Acquisition. Within each of these categories is a complete list of source files and header files (all of the files that are needed to build this DLL). Anthony explained:

> What COMProjMgr will do is, if one of these files changes, it knows the dependencies about everything from everything else. And it will go through the old build just for things that it needs to be built. Because building everything takes about six hours [as at end of December 2001], even on a high-powered machine.

Therefore, COMProjMgr builds only those files that have been modified or added, and those that depend on them.

> So COMProjMgr basically is a dependency scanner, it will scan through all source and header files looking for dependencies from other files. And it will tell us about these dependencies (Anthony).

Perforce is a VCS and configuration management system used by the LeCroy software team. On older scopes they had used SourceSafe. One of the shortcomings of SourceSafe was dealing with multiple sites. Working with SourceSafe on a WAN was very slow and 'it was very inefficient' (Larry). So LeCroy investigated some other tools: they looked at Perforce and ClearCase. But with ClearCase 'we had some problems with that when we were testing it – it did not meet the requirements for working in a distributed environment' said Larry. The advantage of Perforce is that it is a client–server based system. 'Because, Perforce knows what you have on your local machine and what it has. We found it has been terrific, very reasonable speed for both, people here in NY and people in Geneva,' (Larry) and for Jon who works from Maine. Therefore, LeCroy decided to switch from SourceSafe to Perforce. Physically, Perforce resides on a server at the Geneva office, and the team in NY accesses it over the WAN, so the only difference there is that from NY it takes a little longer to access.

For testing the components an in-house developed tool called *SoftwareTestHarness* is used. 'What it does, it shows you all the LeCroy developed components in your system and you could say "run a test for all of them" or "run the test for any one of them," Larry explained.

Each component LeCroy develops has interfaces that are standard for the component: one *basic self-test* and one for an *advanced self-test*. There are special test components (used for testing of other, functional, components). They typically contain 'a whole bunch of test cases' needed to make sure that the functionality of the tested component is correct. 'So we can test each component by itself in this SoftwareTestHarness, and that runs every single day automatically,' Larry explained.

BugBase is a tool used for tracking bugs. This tool is created in-house, and is based on Lotus Notes database that is accessible and constantly replicated over the Web. In this way, every LeCroy office has access to the BugBase: 'Also all our sales offices, in Japan for instance, they have a copy of it' (Larry). In sales offices employees can enter the bugs, look at their status, but they cannot change anything. And as a bug gets fixed, the one who entered the bug gets notified that it has been fixed.

'So for managing bugs, BugBase is invaluable,' Larry says. It is very convenient for tracking. Therefore, sometimes, the managers use the tool in a manner for which it was not intended, for tracking development tasks: 'Sometimes we put in tasks for people just because it is a convenient way to track things' (Larry).

Besides BugBase, LeCroy engineers have project databases in *Lotus Notes*. Larry explained:

> Because we are working at separate locations and Lotus Notes replicates databases, it is very good for us. And so the big data bases are local to Geneva and here [in NY], and they get replicated constantly over the Web.

Epilogue: evaluating the success of the component-based Maui architecture

The main goal and the main advantage anticipated from the component-based Maui architecture was to 'take the bunch of different components and create a different instrument – within weeks is kind of optimistic – but within a few months rather than in a few years' (Larry). It was a long-term planning that aimed at reducing time-to-market and lowering costs while delivering state-of-the-art products. Therefore, the story would not be complete without reflecting on what has actually happened since the launch of Aladdin (the first WaveMaster) in January 2002, and reflecting on the progress of Maui and the technical, marketing and financial evaluation of products based on the Maui architecture since 2002 and until 2006.

From a technical perspective, comparing expectations and actual achievements several years later (until 2006), it is safe to say that the Maui architecture has been a great success. So far, the expectations of the LeCroy software team have come true. Larry reflected on the products released in 2002:

> We began shipping both the WaveMaster 8300 and 8500 to customers in March, 2002. At the same time we also began shipping a Disk Drive Analyzer (DDA), which is based on the WaveMaster 8500. The DDA has a

customized front panel and analysis software that is targeted to the engineers who design hard disk drives.

During 2002, LeCroy have introduced a wide range of software options for the WaveMaster series. In January 2003, LeCroy launched the WavePro 7000 series of scopes (7000, 7100 and 7300), which replace their 'mid performance' scope line. These are also based on Maui. Since 2002, LeCroy had been able to reduce significantly time-to-market in introducing new products and features (product options). One of the best examples of Maui's power and flexibility was the development of the Serial Data Analyzer (SDA). Like the DDA, the SDA is a customized version of the WaveMaster for analyzing serial data streams. Larry explained:

> The concept for this instrument was developed in June [2002]. In mid July, we completed the product plan and began the implementation. We were able to staff the project with 5 engineers to develop the many processors needed for this instrument and one engineer who was responsible to implement the user interface and connect all the components together. This instrument was introduced on October 1st [2002].

LeCroy's WaveMaster 8600 was announced as one of the top products of the year 2002 by *END* magazine.

On 1 May 2006, LeCroy announced industry's fastest real-time oscilloscope.

Due to the Maui architecture, LeCroy have successfully partnered several commercial software companies (e.g. Mathsoft Engineering & Education, Amherst Systems Associates and MathWorks) to further extend the analytical capabilities of LeCroy products.

From the marketing perspective, with WaveMaster LeCroy entered a new segment of the oscilloscopes market:

> Prior to the launch of the new WaveMaster family of oscilloscopes, LeCroy product portfolio centered on mid-to-high-performance oscilloscopes operating at speeds of 500 MHz to 2GHz. LeCroy's share of this segment of the oscilloscope market (approximately half of the overall market) has grown to about 25%. With the introduction of the WaveMaster, LeCroy entered the 3GHz-6GHz high bandwidth segment of the oscilloscope market with a new line of oscilloscopes operating at these high speeds. This market segment is the highest performance segment of the real-time oscilloscope market and is likely to be the fastest growing segment of the oscilloscope market during the next several years' (Annual Report 2002).

On the financial side, the contribution from products based on Maui is reflected in the financial reports of LeCroy from 2003 until 2005. The revenue of LeCroy has grown from $107.8 million in 2003 to $120 million in 2004 and to $160 million in 2005 and 2006.

Notes

1 This case study is based on the data collected from interviews, internal and external documents, reports and press releases, direct observations in NY and Geneva offices, and follow-up communications for clarifications and feedback during a period from November 2001 until January 2003. Individuals interviewed include Larry (Director of Software Engineering, responsible for the NY team), Anthony (Chief Software Architect, responsible for the Geneva team), Gilles (software engineer from Geneva who came to NY for 1 year in August 2001), Adrian (Web master), Dave (VP, Chief Technology Officer) and Corey (VP of Information Systems, Facilities and Security). Based on the comments of anonymous reviewers, Larry was contacted for additional information and clarifications in October 2006.

2 The acquisition system captures signals: it is the heart of an oscilloscope. To read these signals, analyze and display them, additional combinations of software and hardware components are required (software to read the data, analyzing system, display and the front panel).

3 The name of the project ORACLE comes from Greek mythology, meaning 'foresee a future.' (The ORACLE project has nothing to do with ORACLE databases and the ORACLE company.)

4 Jon was the main architect for Maui.

5 Joe was Project Leader for the Virtical Market software (analysis packages).

6 There were 508 components at the end of December 2001; this number had grown to 1119 by October 2006.

7 There were 219 standard Maui interfaces by October 2006.

8 The previous system was half–a-million lines of code. And Maui is an environment completely different from the earlier system.

9 The name Aladdin comes from the famous story about a boy, Aladdin, who finds a bottle that contains a Genie and lets the Genie out of the bottle. As Dave explained: 'We code-name all of our development projects. The acquisition channel Integrated Circuits (ICs) are: a very fast (10 Gigasample per second) analog-to-digital converter (ADC), a very fast (7 Gigahertz bandwidth) amplifier and a custom memory that accepts the data from the ADC at a 3.3 Gigabyte per second rate. We refer to these chips as a "chip set." The technology development that made the acquisition channel ICs was called "Genie." As you may recall, a Genie is a mythical being who has magical powers, lives in a bottle and grants wishes to someone who releases them from the bottle. We believed these ICs would give us the power to grant our wish of getting into the very high performance part of the oscilloscope market. So, the Aladdin project that was the first use of the "Genie". ICs was code-named Aladdin because it let the Genie out of the bottle.'

10 Each project is associated with a different product.

References

Carmel, E. (1999). *Global Software Teams: Collaborating Across Borders and Time Zones*, Upper Saddle River, NJ: Prentice-Hall PTR.

Crnkovic, I. and Larsson, M. (2002). Challenges of Component-Based Development, *The Journal of Systems and Software* **61**: 201–212.

Huang, J.C., Newell, S., Galliers, R.D. and Pan, S.-L. (2003). Dangerous Liaisonsc? Component-Based Development and Organizational Subcultures, *IEEE Transactions on Engineering Management* **50**(1): 89–99.

Kotlarsky, J., Oshri, I., van Hillegersberg, J. and Kumar, K. (2007). Globally Distributed Component-Based Software Development: An exploratory study of knowledge management and work division, *Journal of Information Technology* **22**(2): 161–173.

Peters, J.F. and Pedrycz, W. (2000). *Software Engineering: An engineering approach*, New York: John Wiley & Sons, Inc.

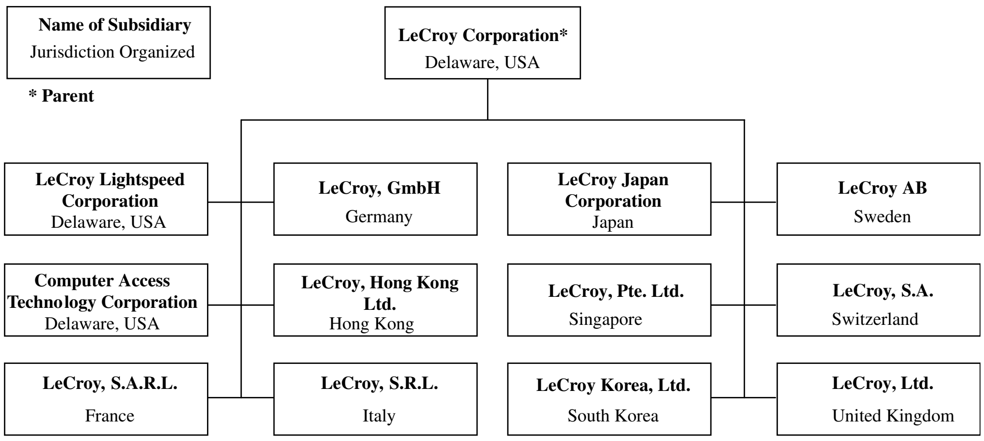
Vitharana, P. (2003). Risks and Challenges of Component-Based Software Development, *Communications of the ACM* **46**(8): 67–72.

About the author

Dr. Julia Kotlarsky is Assistant Professor of Information Systems, Warwick Business School, UK. She holds a Ph.D. degree in Management and IS from Rotterdam School of Management Erasmus (The Netherlands). Her main research interests revolve around social and technical aspects involved in the management of globally distributed IS teams and IT outsourcing. Julia published her work in journals such as *Communications of the ACM, European Journal of Information Systems, Information Systems Journal, IEEE Security & Privacy* and others.

Appendix A

Lecroy Corporation organizational structure



Teaching notes

This case study describes an actual situation at LeCroy Corporation, involving several decisions, challenges and opportunities faced by the managers of a globally distributed software development team over a period of time when they re-engineered a monolithic system into a component-based system.

*The case takes a historical perspective, describing the transitions in a software architecture from the mid-*1980*s until early 2002:*

- *Period 1*: a monolithic system
- *Period 2*: a modular system (the monolithic system is broken into three modules)
- *Period 3*: a component-based system (components are reused across a number of products)

For each period, this case study covers three themes:

(1) *Product architecture* – advantages and disadvantages of the architecture at that specific time period. Opportunities, long- and short-term goals and strategic decisions made by managers.
(2) *System re-engineering* – human and technological aspects related to system re-engineering and adoption of new technologies at LeCroy.
(3) *Global distribution* – human and technological aspects related to managing globally distributed teams.

Teaching goals

The major teaching goals of this case could be summarized as:

System re-engineering: to discuss the main challenges involved in system re-engineering in a highly competitive environment (from technological and human perspectives).

Long-term strategic planning: to discuss issues and challenges in long-term strategic planning in system development and (architectural and product) innovation.

Component-based architecture: to understand the main principles of a component-based system and how it differs from traditional (monolithic) architectures. The case allows students to develop a feel for the technology 'behind the screen.' Furthermore, students can analyze what is a 'good' system architecture and study the advantages of component-based architectures. To understand a lifecycle of the architecture of a software product (a longitudinal perspective) and how a product architecture may change over time.

Global distribution: To analyze the influence of a globally distributed setting on the system development approach from human and technological perspectives.

Learning and knowledge management strategies: To analyze and discuss knowledge management and learning strategies in a globally distributed environment.

Tools and technologies: To analyze what are the major requirements for tools to support CBD, in particular in a distributed environment.

Target audience

Target groups for this Case could be MBA, M.Sc. students of IS, project management or general business. It is likely that these students will be involved in decisions on architectural innovation, system re-engineering and will be involved in globally distributed projects. The case might be suitable for undergraduate electives (e.g. IT Architecture).

Suggested questions for discussion and assignment

Analyze and discuss LeCroy Case from the following perspectives:

The system re-engineering perspective
The objective is to understand the main challenges involved in system re-engineering in a highly competitive environment.

Questions:

1. What are the major challenges LeCroy software managers faced during the different stages of system re-engineering? (Discuss technical and organizational challenges in each of the 3 periods – initial, ORACLE and Maui)
2. What strategies did LeCroy implement to deal successfully with these challenges? Are there any alternative strategies you could recommend?

Component-based architecture
The objective is to understand the major principles of a component-based architecture, how it differs from traditional (monolithic) and modular architectures and what are the advantages of a component-based architecture.

Questions:

3. What is a 'good' software system architecture? (Support your argument showing why it is good). Does the Maui CBD architecture LeCroy developed have the characteristics of good system architecture?
4. What competitive advantages did LeCroy gain from having a component-based product architecture?
5. What are the requirements for tools to support (globally) distributed CBD?

Additional questions may include questions about:

Working in a globally distributed environment

The objective is to analyze the influence of a globally distributed environment on the system development approach.
Questions:

6. What challenges did LeCroy software managers face in managing a globally distributed team? In your opinion, what are the critical issues that need special attention in a globally distributed environment?
7. What strategies did LeCroy implement to deal successfully with these challenges? Are there any alternative strategies you could recommend?

We would recommend distributing the Epilogue section to students only after the case has been discussed.

Suggested teaching framework and sketch answers are available to teaching faculty directly from the author.