

Major U.S. Chemical Company Uses Eiffel to Powerfully Interface ERP System with Process Control Computers

Abstract

This case study involves two projects at a chemical company headquartered in the United States. In Project 1, an automatic data communication capability was required between a state-of-the-art ERP system at corporate headquarters and a set of older process control host computers at a manufacturing plant. The system must keep communication with the older plant computers as simple as possible, while supporting twenty-four-by-seven operations with comprehensive store-and-forward and error recovery capabilities. In Project 2, a new manufacturing process is added at the plant. This in turn requires a new generation process control host computer system. This new system becomes a client of the system created in Project 1. Very high levels of software reuse are achieved simultaneously with very low defect rates.

Highlights

- 1) **High level of project-to-project reuse of locally produced software components.**
- 2) **Low software defect rate.**
- 3) **Short development time.**

Project 1: Problem Description

The Company has implemented an Enterprise Resource Planning (ERP) system at its corporate headquarters. Users of the ERP system can cause process orders to be issued which dictate the manufacture of certain products. This information needs ultimately to reach the process control host computers at manufacturing sites. Once the product is manufactured, a report of production must be returned to the ERP system.

The process control host systems in place at one manufacturing plant are from a generation of technology prior to the deployment of large-scale ERP systems. Custom software was needed on the process control hosts to receive process order information and return reports of production.

The process orders that come from the ERP system contain much information in a complex structure. The process control host needs only a very small amount of this information to do its job. But it cannot simply dispose of the remainder of the information, because the report of production that goes back to the ERP must contain much of the data from the original process order, transferred again in a complex format.

Project 1: Solution

The Project 1 system was constructed using the Eiffel Development Framework™. The system handles the complex communication with the ERP system, stores the process order information, and creates and communicates a simplified process order to the appropriate process control host computer at the manufacturing plant. Once the material designated by the process order has been produced, the process control host computer passes a simple acknowledgment back to the Eiffel system. The Eiffel system combines it with the stored process order information and then creates and sends the complex report of production message required by the ERP system.

Extenuating Circumstances

This system runs twenty-four hours per day, seven days per week. It includes the ability to work around or recover from several different types of external failures. For example, communication links and databases, although mostly reliable, can at times become unavailable.

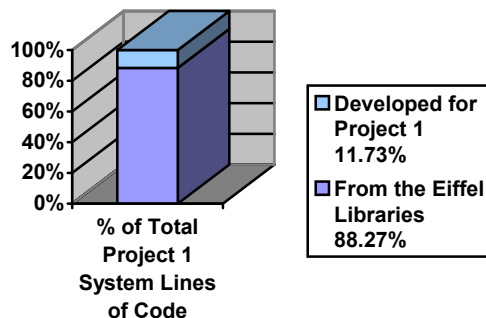
The system also provides a way for humans to intervene when invalid inputs are received. That is, some method for browsing, correcting, and resubmitting these inputs.

Results

Writing Less Software

The ability to accomplish a given task by constructing a minimum quantity of new software means less opportunity for defects and less software to maintain in the future. The Eiffel method encourages software reuse making this a reality.

Because the Eiffel code is so well controlled, it is easy to make measurements of the software. In fact, the most commonly needed software measurements can be made very easily using the facilities of EiffelStudio™. Because measurement is so time consuming and tricky when using other technologies, no significant measurement of the characteristics of software had been done within the Company IT organization previously. The Eiffel system for Project 1 was constructed by reusing extensively the rich software libraries distributed with EiffelStudio™, particularly the EiffelBase, EiffelStore, and EiffelTime. Using the measurement tools in EiffelStudio™, it was shown that about 88% of the lines of code that comprise the system were code that were delivered in the Eiffel libraries ... that is, code that the Company programmers did not have to write. Only the remaining 12% were new code written by the Company programmers.



Reliability

Since the system was placed in production (approximately three years prior to this writing), it has been supporting manufacturing 24X7, and only one software defect has been reported. This success can be attributed directly to the use of the Eiffel method and language. Because less new code is written, there are fewer places for bugs to arise. More importantly though, the Eiffel method specifically addresses reliability issues by providing a facility called Design by Contract™. The Company software engineers who implemented this project are as fallible as any of their peers. But using Design by Contract™ helped them reveal software defects early in the development cycle, reducing the probability that production code would be faulty. The engineer responsible for maintenance on Project 1, describes its reliability with this anecdote: “At our weekly teleconference roundtables, I would listen to other engineers all over the company telling about the problems that they are having with their software. Every week the manager would ask me how [Project 1] was running. Every week I could say, ‘It’s running fine ... no problems.’ That’s a very good feeling. Eventually, the manager quit asking.”

Emerging Software Assets

As stated previously, a major advantage of the Eiffel Development Framework™ is its emphasis on software reuse. This is confirmed by the very high rate of reuse of the Eiffel library code noted above. But the Eiffel class libraries were written by highly skilled engineers specifically intending to create Eiffel classes (software modules) that are applicable to a number of software applications. What about classes created by ordinary software engineers during the process of constructing a new software system? Will those elements be reusable?

When organizations use Eiffel to develop a new system, they create new classes to serve specific purposes in the system being built. Often subsequent projects appear as “variations on a theme”. Modules built for specific purposes for one project can often be used with slight variations for the next project. This is no secret; software engineers have been doing it for years. Prior to Eiffel and other purely object-oriented methods, the state of the art was to make a copy of the first module and modify the copy to suit the second purpose ... a process sometimes referred to as “clone-n-hack”. The disadvantage to “clone-n-hack” is that every time it occurs, a new maintenance load is created. If a bug is discovered in the original module, then all clones must be examined, analyzed, and fixed if necessary. However the Eiffel method defines a seamless software lifecycle in which modules go through a process of generalization as they are reused. During generalization, modules are strengthened (documented and contracted more fully) and abstracted (made applicable in more diverse situations). It is this process of generalization that starts a module on the path from being a specialized part of a single system to being part of a corporate library of reusable software assets. In Eiffel the second use (i.e., the variation on the theme) contains *only* those features that are new or different from the first.

During the development of Project 1, the Company engineers felt that some of the modules they were creating would probably have reuse potential in the future. Among these were the classes that model the complex data streams that go to and from the ERP. Also, the engineers had produced object-oriented encapsulations of two third-party products. These products are message-oriented middleware (MOM) used for store-and-forward, guaranteed delivery communications between systems. Both products expose complex C language programming interfaces. By applying the Eiffel principle of Operand/Option Separation, the team produced interfaces that are greatly simplified from the originals. For example, a call in the original C interface which requires 16 arguments only takes one argument in the Eiffel version. Because these products are used widely within the Company, the engineers felt that the new interface classes had reuse potential.

So, the team anticipated reuse potential in the areas just mentioned. Indeed in Project 2 they did reuse these modules. But to their surprise, they were able to reuse much more Project 1 software than they had imagined.

Project 2: Problem Description

The engineers developing the Project 1 system were unaware that they would be involved in supporting the installation of a new manufacturing process. The new process was designed to allow the manufacture of one of the site’s key intermediate chemical products in a significantly more cost-effective fashion.

This new manufacturing process required a new process control host computer. Of course the new process host would be current generation technology, unlike those older, Fortran-based systems that were clients to the Project 1 system. Yet this new process itself also needed to be a client to the Project 1 system. That is, it would be required to receive process order information from the ERP system and report production back to the ERP system.

Project 2: Solution

In its most intuitive form, the need for Project 2 involved analyzing the software on the older process control host computers that were clients to Project 1, and then building software that would do the same type of thing, but using the new tools available on the new process host.

Results

Writing Even Less Software

So, it was thought initially that the Project 2 system would be a completely different type of system than the one created in Project 1.

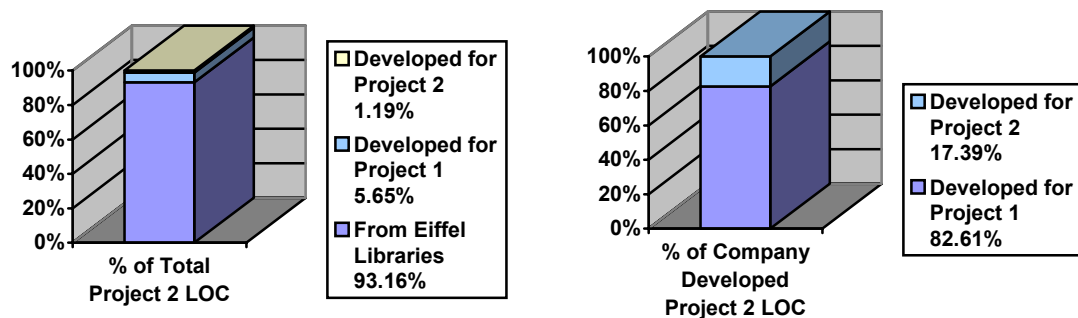
Of course the team anticipated that the delivered Eiffel libraries would provide many components useful in Project 2, just as in Project 1.

At first glance this did not appear to be the case for much of the Company-developed software from Project 1. Project 2 would need to use the MOM software. Therefore, reuse of the interface modules developed for Project 1 seemed plausible. But that seemed like about it.

However, after an analysis of the software on the older process control host computers, a pattern of commonality began to emerge. It appeared that the types of actions taking place on the process control host computers were similar, at least at an abstract level, to those taking place within Project 1. Essentially, each was waiting on some sort of request to be received from some source or sources as a result of some event. Then after processing the received request, which usually implies passing on a message to another party, the software would continue to wait for additional requests from its sources.

The team realized that by slightly generalizing some of the modules in the Project 1 system, they would be able to reuse those same modules as the heart of the Project 2 system. As a result the abstract notions of Request, Request Processor, Request Source among others, became generalized, reusable software modules. This meant that in Project 2, the developers got the basic request processing functionality, including fault tolerance, browsing, resubmittal, and logging ... all very nearly for free.

The following charts show the reuse levels of code in the Project 2 system. The first shows that only slightly over one percent of the code is new, if the Eiffel Library code is taken into account. The second chart limits the scope to code written by the Company. Less than eighteen percent of the code is new for Project 2. This means that for every line of new code written for Project 2, four lines of code were reused from the Project 1 components.



Short Development Cycle

As a result of reusing Project 1 components, a proof-of-concept for Project 2 was built and running on the new process control host computer in the space of a single afternoon. A Company process control systems engineer had ultimate responsibility for the Project 2 development task. Regarding the efficiency of building new systems based on reusable software components, the process engineer said, "I joked with [the Project 1 leader] that after one afternoon of development with [a different technology], I would still have been coding declarations! The potential for reusing Eiffel code is truly extraordinary. We really produced very little new code to create this new system."

The process engineer also says that in the case of this project, the short development cycle was particularly welcome: "It seems that there are never enough time and resources to do things the way we'd like. Particularly, during the course of this project, we suffered from a sudden personnel change causing the permanent loss of a team member, and the temporary, but unexpected medical leave of another. Without the rapid, high quality development we got from Eiffel, we would not have met our deadlines. My instincts tell me that we built the Eiffel system in half the time it would have taken in [a different technology]."

Conclusion

The study of the two projects at this Company demonstrates clearly the undeniable power of Eiffel to improve software economics.

Working effectively in Eiffel requires some changes to the way that software engineers think about their jobs. These changes are not difficult to understand, but they sometimes take effort to effect. In this case study, two such notions stand out.

Contrary to the view of many software engineers, the goal of every software engineer should be to write less software, rather than more. Less software means less opportunity for failure, less maintenance load, more reuse, more reliable systems ... and, ultimately, improved software economics. Certainly this change of thinking paid off for the teams in this case study with the very high rates of software reuse and reliability in these projects.

In the Eiffel method it is said that an engineer should create every module as if he or she expects that module someday will become a reusable software asset ... but the engineer should never consider a module reusable until it actually has been reused. Certainly, software engineers cannot be expected to be clairvoyant. But by following the tenets of the Eiffel method such as Design by Contract™, engineers can produce software that can be generalized easily and safely into more widely applicable corporate software assets.

About Eiffel Software:

Eiffel Software (a division of ISE) is the world leader in Eiffel true object-oriented programming tools. Founded in 1985, Eiffel Software produces proven professional tools and component libraries for business-critical and enterprise software developments, as well as Eiffel training and consulting services. Eiffel Software's products and methodologies enable their customers to output more and higher-quality software in less time than with any other development tools available. Its users span the globe, in industries ranging from large financial institutions, to technology manufacturing, to government and defense contractors, to health care providers and more.

You can visit Eiffel Software on the Web at <http://www.eiffel.com>, or by telephone in the USA at +1-805-685-4395.