Integrating an UML tool in an Industrial Development Process - a Case Study

John Knudsenjohnk@cs.auc.dk, Rene Gøttler, Michael Jacobsen, Mads W. Jensen, Jens G. Rye-Andersen, and Anders P. Ravn

CISS Aalborg University
DK-9220 Aalborg
{johnk,reneg,shapeles,blonde,jgra,apr}@cs.auc.dk

Abstract. This paper presents our experience with successfully integrating the UML CASE-tool, Rhapsody, into the software development process of a small company producing high quality embedded systems. The company had not been using Object Oriented Analysis Design and Programming, and this problem was attacked by developing a prototypical application that can be used as a pattern for developers in the company. Documentation of the application, including test specifications and generated and hand coded programs, is kept within the tool, such that the tool truly assists in the full development process. However, the most challenging aspect in our case study was to customize the tool such that it could support concurrent engineering and special platforms. We discuss these matters in detail.

Keywords: Software Engineering Tools, UML, Embedded Systems.

1 Introduction

Many small and medium sized companies are producing embedded systems which are increasingly software intensive. This software needs to be highly dependable, because it is an integral part of a mass-produced product which the customers expect to work smoothly for their lifetime. For this reason, such a company tend to have a defined software process with documented procedures for design, development and test; they appreciate the cost of software development, and are keen to find ways of making the process more efficient. However, some of the more experienced staff members have bad memories of CASE-tools from the late 1980'ies which just produced heaps of "documentation" which was hard to relate to the actual implementation and tests of the system. Such tools are sitting around as 'shelfware', while the actual documentation is done with word processors and code commenting conventions. However, with the advent of UML based tools that incorporate semiautomatic code generation, simulation of scenarios and report generation, it may be time to get some real benefits from a model based design and development process. When we were approached by the company, they formulated their expectations as follows: "We have invested in a license for the Rhapsody tool [1] and we would like you to help us to fit it into our software development process, such that this process is made just as dependable and manageable as our hardware development process (which is done using Cadence tools)". The contribution of this paper is thus a description and discussion of the steps that may be taken to successfully integrate tool supported, model based design and development methods into a small company producing high quality embedded systems.

Our analysis of the requirements for a successful integration gave us three important success criteria:

1. smooth embedding of *model-driven development* into the existing software process,
2. better control of *test and validation* efforts, in particular at the system level, and
3. version controlled *integration*.

Code generation is not a key issue; but keeping track of *code-design correspondence* is very important.

Related Work: Model-based development and especially UML [2] have been subject of much research. Yet, the related issue of ensuring industrial take-up, which is a success criteria for software engineering research, has not been widely discussed for UML based tools. In 1998, a seminar *"Werkzeuggestützte Modellierung des Tamagotchi"* at the University of Kaiserslautern had a workshop on different design methods and CASE-tools which resulted in several reviews of these next generation CASE-tools [3]. The group working with Rhapsody concluded that the tool allows for better software specification than simple diagram applications. The improvement recommendations were targeted at specific tool options and they mention a lack of compiler support, which is an issue also targeted in the case study.

A project was conducted in 2001 in order to gain familiarity with Rhapsody [4]. Rhapsody was explored through a case study, similar in size and scope to that of our work. The project did not evaluate Rhapsody, but had some technical advice as to how specific problems were solved or circumvented. For instance, it is recommended that a GUI is used in order to make insertion of events easier, and that class instances should be checked for null-values to prevent addressing errors. This level of detail is not included in this paper; but we agree with a key conclusion of the previous study: *"UML in Rhapsody provides for a structured method of developing object-oriented software"*.

Another related work is an UML success story from 2002, in which Motion Control Engineering Inc. appraises Rhapsody. An I-LOGIX Rhapsody expert was brought in to provide in-house training for three developers. They found that the Rhapsody tool worked well for concurrent program development, as they only had one developer on each branch, and they made sure to merge the branches periodically. In addition, they only used the configuration management (versioning control) interface in Rhapsody late in the development process. During our case study we have found reason to argue that concurrent development in Rhapsody is not fully satisfactory.

Overview: The remainder of the paper is structured as follows. Section 2 gives a short introduction of the company, FUTARQUE, their product and development process and applied tools. In Section 3 we present the path followed to the successful integration of the CASE-tool in FUTARQUE. In Section 4, we conclude with a discussion of what features of FUTARQUE were crucial for success and the qualities of the specific Rhapsody tool.

2 Case Study Preliminaries

FUTARQUE is a company that develops various solutions for the digital TV market with the main focus being turn-key solutions for digital set-top boxes (STB) [5]. FUTARQUE's development is divided equally among a hardware and a software division. The hardware division has benefited greatly from introducing advanced development tools and the improvements attained in the hardware division by adopting such tools, has inspired FUTARQUE to achieve similar improvements in the software division.

Development at FUTARQUE is currently done using a simple text editor for editing the source code and a compiler supplied by vendor of the target chip. The compiler tool-set is a command line driven tool-set, which can be used through the command prompt in Microsoft Windows or Linux. FUTARQUE uses Cygwin [6], a Linux-like environment for Windows providing a basic shell, alongside a wide selection of applications. The usage of Cygwin in Windows and the GNU Make system from Cygwin makes it somewhat transparent whether the development is conducted with Windows or Linux hosts.

Debugging the software is primarily done through a clear text interface either through a JTAG [7] or using a UART interface from the STB. Furthermore, STMicroelectronics supplies a graphical debugger that lets developers see what happens during application execution.

FUTARQUE uses a versioning system called Concurrent Versions System (CVS) [8]. CVS administers every version of the source files and as an important feature, it supports branching. Branching means that a team of developers can branch out and at a later point merge the branch into the rest of the code and more developers can work on the same code components in parallel.

The envisioned development in FUTARQUE is to rely completely on Rhapsody using Windows as the host operating system and the target platform is a MIPS-based [9] core CPU using Linux as operating system.

At the beginning of our work, FUTARQUE set the following requirements for integration of Rhapsody in their development environment:

Versioning system: FUTARQUE has always used CVS and would like to continue using CVS as their versioning system. The reasons are that they are already familiar with this system, and servers and back-up are running. Furthermore, CVS supplies all the functionality currently required.

Compiler: FUTARQUE uses Windows and Linux as host development operating systems, thus Windows will be used as the host operating system for Rhapsody. The target platform is a CPU with a MIPS-based core, running Linux as operating system. The optimal solution is to get a cross-compiler tool-chain that is hosted on Windows, but a solution with the cross-compiler on a Linux server and remote execution is also acceptable.

Remote file access: Instead of moving the compiled software components to the target all the time it is easier to remotely mount a file system on the target. This can be done either through NFS [10] or SAMBA [11].

In order to give FUTARQUE developers a nice introduction to model based design and development, we developed a demonstration based on a scaled down, but yet realistic STB application. We shall not discuss this example further, because it does not differ in essence from other UML examples in the literature, and we find that solutions to the integration aspects outlined above much less discussed, although they are very important for acceptance. However, we add the following remarks about the example:

- it uses tool facilities for documenting the design, such that report generating gives full documentation of the software,
- Sequence Diagrams are developed to define complete tests at the module interface and integration levels, and
- code is generated for the target platform.

This has been our answer to the demands for introducing model-driven development into the existing software process, and gain better control of test and validation efforts. The remaining issue is integration of the tool.

3 Tool Integration

This section is a description of the integration of Rhapsody into FUTARQUE according to the requirements set in Section 2.

3.1 Compiler

Rhapsody supports several commercial compilers and compiler tool-sets, some examples are Microsoft Visual Studio [12], Metrowerks CodeWarrior [13] and Green Hills Software Inc [14]. The latter two support cross-compiling to a MIPS-based core CPU, but only to the native core level. What FUTARQUE needs is a cross-compiler that compiles to a Linux target operating system running on a MIPS-based core CPU. Currently, there is no commercial compiler tool supported in Rhapsody that supports cross-compiling to a MIPS-based core CPU running Linux, and it was not possible to find a commercial solution, even among products not supported natively by Rhapsody.

GCC is the GNU Compiler Collection, which currently contains front ends for C, C++, Objective-C, Fortran, Java, and Ada, as well as libraries for these languages [15]. Besides the native targets, e.g., GNU/Linux, GCC also supports cross targets, e.g., ARM [16], SPARC [17], MIPS, and many more. Not all of the cross targets are fully supported and dependent on setup configuration patches might be necessary, especially for minor architectures such as MIPS.

Creation of a cross-compiler tool-chain is not a straightforward task, but because of the work done by Dan Kegel through the creation of `crosstool` [18] it has been made a not too difficult task. `crosstool` is a script collection for automating the creation of a cross-compiler tool-chain based on GCC and the GNU C Library [19]. `crosstool` supports the creation of both a native mipsel-elf compiler and mipsel-linux which is the one needed for this work.

Creation of a mipsel-linux cross-compiler tool-chain hosted on a Linux machine works out of the box with `crosstool`, this means that the option with a Linux server as compiling server is possible. But creation of a mipsel-linux cross-compiler tool-chain hosted on Cygwin/Windows is not possible, because the GNU C Library is not compatible with Cygwin. The reasons are with the libtool utility and incompatible reserved keywords, see news group posting [20]. There is another widely used C library used for cross-compilers, Newlib [21], but Newlib supports only the i386 architecture, when using Linux as operating system, i.e., a i386-linux cross-compiler. It was however possible to create a Cygwin/Windows hosted mipsel-linux cross-compiler, by using the GNU C library files created on Linux, thus avoiding compiling of the GNU C library on Cygwin/Windows.

3.2 Remote Access

When the source files have been created and compiled for a model, the binary executable has to be executed by the target STB, i.e., the STB has to have access to the executable. The simplest solution is to copy the binary executable to the target and then execute it, but this is not very practical, and it is time consuming. A better solution is to use remote access via an Ethernet interface. Specifically, this can be done in two ways; either by exporting a Linux root file system via NFS and then have the STB use this as its root device, or by having the STB run a stand-alone Linux version from hard-disk and then mount a remote directory via NFS or SAMBA.

Figure 1 shows a compiler setup that is working. The setup has not been completely tested, but the Cygwin/Windows mipsel-linux compiler works, the STB executes the model, and the animation feature works in Rhapsody.

3.3 Concurrent Development

Concurrent development is an important issue because software projects are often developed by a team, and sometimes across great distances. The level of management needed is unique from project to project and some of the influential

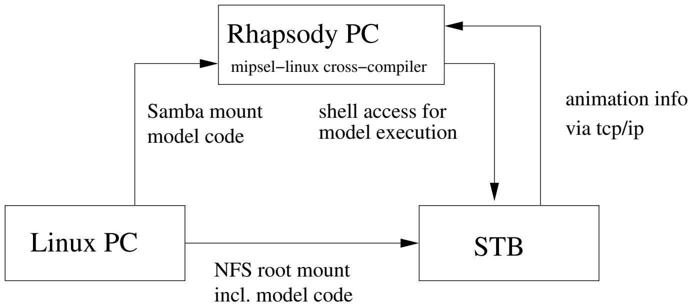

Fig. 1. Working compiler setup.

factors are: project size, project life time, number of developers, development procedures, and geographical diversification.

CVS and Rhapsody: FUTARQUE has several developers working on the same project, thus the requirement for CVS support. The code base at FUTARQUE has a significant life time, because many products are configured from the main code base. Most of the development projects, be it addition of new functionality or bug fixing, span across several modules in the code base and because of this characteristic the development process rely on a robust versioning system.

The development procedures at FUTARQUE exploits the branching feature in CVS. Branching is used for development tasks that are estimated to take more than two days because it enables the developer to be sure that other changes that might effect the current work does not interfere, and that temporary and incomplete solutions can be committed without limiting the usage of the code base. Branching in CVS works by creation of *co-existing* versions of the branched files, with the possibility of parallel development on both versions. Several branches can co-exist and it is also possible to create branches on branches. Figure 2 illustrates the creation of a branch on a file, the file is edited in the main sequence as well as the branch, and finally the branch is merged into the main file version.

Rhapsody supports version control software, but only as a front-end in the form of a configuration management menu, therefore, it needs a back-end to function. By using CVS and Perl (Practical Extraction and Report Language) and customizing some of the Rhapsody configuration files it is possible to use CVS as back-end. Currently, only the basic CVS functionality is supported in this integration and it is not possible, e.g., to compare two versions in the repository.

It is possible to store both text-based and binary files in CVS, but CVS supports only merging of text-based files, i.e., handling of parallel development, because it generally makes no sense to merge binary files where developers cannot understand the format and solve unavoidable conflicts. The models in Rhapsody are stored in a text-based format, thus applicable for CVS merging, but the format is not easily readable by developers. This means that when two developers check out the same file and make changes in the same parts, the last to commit

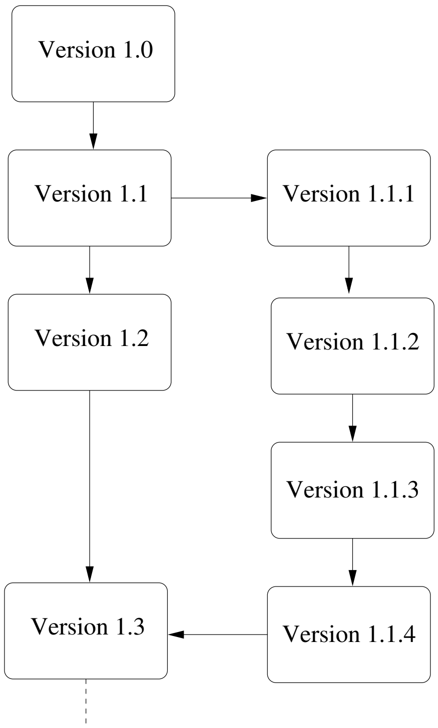

Fig. 2. CVS branching example.

will get conflicts, and it is not always easy to resolve these manually. Some elements in the model are in such a complex format, that it is not recommended to risk creating conflicts in them. Instead, it is recommended that they are under strict process management rules.

During the development of the case, it was discovered that Rhapsody creates a time stamp within a file every time it is saved. This time stamp is another reason that the mentioned conflicts occur. Although an annoying effect, it is a fairly easy conflict to handle, especially if there are other conflicts and the file have to be opened anyway. Also, it appears that if two developers change some graphical elements in the same file, e.g., object model diagrams or statechart diagrams, a conflict occurs which is very hard to handle manually, as the graphical representation on file is complex. Many conflicts can be corrected without much trouble, especially by using a text diff tool. Furthermore, as the number of concurrent developers increase, so does the number of conflicts. Organization of the model, which is described later in this section can help to ease this problem.

The DiffMerge Tool: The models in Rhapsody are not really suitable for the line-by-line comparison that CVS uses when it updates and merges files. Rhapsody provides an external viewer called DiffMerge, which can be used to see the difference between two models. This enables developers to compare two versions of a Rhapsody model in a configuration management system and merge them if necessary. The DiffMerge tool presents a graphical overview of the differences, and the user may view the differences in a step-by-step fashion and choose which to keep and which to dismiss. When the merge is complete, a third model is created and the two "old" models are unaffected by the merge.

To be more specific, DiffMerge can compare any Rhapsody files of the same file type or compare a unit to another unit. It compares the meta elements of a unit, e.g., the attributes, properties, and relations like associations and aggregations. Units can be compared as files on disk or in their transient form, i.e., a live object in RAM.

The DiffMerge tool is important for a development team, and therefore it is important for us to integrate it into the CVS front-end. Unfortunately, it is only possible to integrate the basic CVS functions such as check in, check out, and add - not the DiffMerge tool. Furthermore, the configuration management interface in Rhapsody does not support all needed CVS administration functionality, thus an external CVS front end is required. FUTARQUE already uses WinCVS, so this application is maintained for all configuration management.

A problem with DiffMerge is that it does not have an equivalent to the conventional merge function in CVS. It does have an automatic merge function, which is a form of absolute merge that merges two models into a third model that contains the two models including all their differences. This behavior is usually not desirable because only the changes from the newest model are relevant, e.g., functionality deleted in the newest revision should not be present in the final model. Therefore, the user must go through the differences manually and choose which to keep in the merge. This manual step-through of the model differences is possible but very tedious and error-prone, because the user doing it has to have a thorough knowledge about the meta elements in the models, in order to perform the manual diff correctly.

Development Process Management: Because of the limitations regarding the DiffMerge tool and the problems with conflicts during CVS merge, parallel development on shared files is not possible without using the manual DiffMerge process or by using the plain CVS merge. This requires strict development procedures to avoid developers constantly being occupied by solving complex CVS conflicts. I-LOGIX has provided some guidelines for configuration management of important project files, these can be seen in [22, Sec.1-11].

Besides the CVS setup, some general development procedures are necessary. Specifically, a manager needs to specify some rules regarding work on graphical artifacts, such as object model diagrams. These should only be changed by a single developer at a time, and the developer should lock the file in the repository before changing it, and release the lock when finished. This approach is also recommended by I-LOGIX, alongside recommendations for model organization.

Class files, i.e., .cls files, contain both some graphical elements, e.g., if the class has a statechart attached, and also all class information like attributes and operations. These files need especially strong development procedures in order to enforce that only one developer at a time changes the graphical elements.

Model Organization: Concurrent development needs more than configuration management. It is a good idea to have an infrastructure which separates some of the model elements into units. A Rhapsody unit is simply a model element stored in a separate file. Rhapsody allows components, packages, classes, and diagrams (except statecharts and activity diagrams) to be separate units. The advantage of dividing a project into units is that it enhances concurrent development. If an entire model is only one unit, then only one person may update the model at a time, if locks are used. In Rhapsody every package created is by default a unit. A project needs to be organized in a way that achieves re-usability in developing frameworks and components. I-LOGIX suggests several approaches in the organization of a project, two of which are presented below:

Use cases: These could be suitable for organizing the requirements and analysis model, since the later models are refinements of the use case. Packages can divide the use cases into coherent sets such that a package would contain a use case and its actors, activity diagrams, statecharts, and sequence diagrams.

Logical and physical architecture: The logical architecture is often organized by domains such as types, classes, and other model elements. The physical architecture concentrates on source code and subsystems. If an element is large enough it may even be divided further into sub-packages. Figure 3 shows an object model diagram from the example STB application and here the physical architecture of the software is organized into two packages, an upper-level Application layer which contains the classes `UserInterface`, `ChannelDatabase`, and `Application` , and a lower-level DVB layer that contains the `DVB`, `Demux`, `SIDB`, `StreamProvider`, `VideoDecoder`, and `AudioDecoder` classes. All the mentioned classes are also units, as they are organized into one file per class.

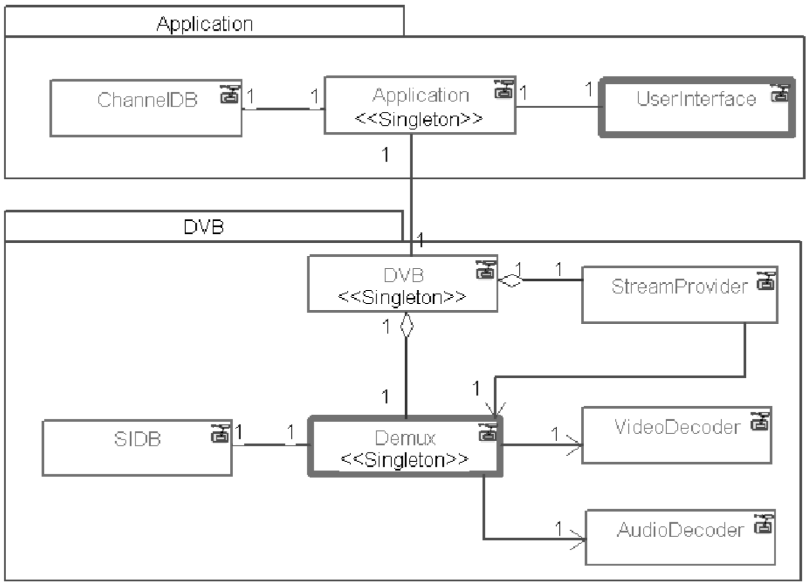

Fig. 3. The object model diagram of the case modeled in Rhapsody.

4 Conclusion

We have given an account of a successful integration of tool supported model based design and development into the company FUTARQUE. The success is documented by the fact that FUTARQUE is now in the process of adapting the procedures for future software development. We believe that the key ingredients in achieving take-over are:

- to provide a realistic demonstration system, in this case the STB, which is recognizable by the developers, and make it complete with respect to documentation, tests and with target platform code generation, and
- to really integrate the tool into the development process.

In the case of FUTARQUE, the work has depended on having a staff of engineers and programmers which during their studies have been exposed to Object Oriented Analysis and Design. They have not used it very actively in their current work; but could see the perspective and have little trouble in understanding the concepts and notations used by the tool. Furthermore it has been very important that the organization had defined procedures for using configuration control tools and specific target platforms, such that tool integration had precise goals. Other organizations may be more diverse. If modern design methods are less known, one may have to supplement with more general courses in analysis and design, although also here, application specific examples may be very important for assimilation of the concepts. A lack of a defined software process would be much more critical, and we see no quick solution.

With respect to the concrete tool, we note that Rhapsody did allow a successful integration, although it was about half the effort spent by us. It seems unfair to blame the tool for the difficulties encountered in building the mipsel-linux cross-compiler tool-chain for Cygwin/Windows, where integration with Rhapsody went rather smoothly. However, with respect to configuration management, we managed to integrate basic CVS functionality with the configuration menu in Rhapsody, but the final solution is based on the usage of WinCVS. DiffMerge provides a graphical merge functionality, but because it is an absolute merge and the merge process can be very tedious, it is not recommended for daily usage. The plain CVS merge is a feasible alternative to the DiffMerge tool, but is still troublesome to use due to the complexity of the text format of some model elements. The lock mechanism, also provided in CVS, combined with model organization and process management, is the chosen solution. This solution does limit concurrent development, but is found to be the better choice.

The successful integration of a version management system in the FUTARQUE case thus relies on a more rigid development process that limits the possibility of concurrent development rather than version control offered by Rhapsody and CVS. The lacking version control is caused by the artifacts produced by Rhapsody or the CVS systems lacking ability to handle such artifacts. Methods to improve either should be addressed in new versions of the tools.

Integrating new development tools and processes is a resource heavy task and for ventures without the specific knowledge on the methods and philosophy behind the development processes, we recommend alliances with consultants possessing such knowledge.

Acknowledgment

We find that thanks are due to management and employees at FUTARQUE, especially Jens Kristian Jensen, for providing the opportunity to work with them and for their efforts in devising case study of a proper magnitude. We look forward to follow the company and their long term experiences with the adopted method.

References

1. I-Logix Inc.: I-Logix Inc. http://www.ilogix.com 21/9 (2003)
2. OMG: (Object Management Group, OMG)
3. Christian Becker, Steffen Glomb, Michael Graf: UML Notation and I-Logix Rhapsody Tool (1999) http://wwwagse.informatik.uni-kl.de/teaching/seminar/ws1998/UML.html, 21/9 2003.
4. G.R. de Boer: *Development of a Cruise Control in UML using Rhapsody.* http://www.ce.utwente.nl/rtweb/publications/Msc2001/pdf-files/002R2001.pdf 15/12 2003 (2001)
5. : futarque. http://www.futarque.com (Visited 21-09-2003)
6. : Cygwin. http://www.cygwin.com/ (Visited 12-10-2003)
7. : JTAG Technologies. http://www.jtag.com/ (Visited 12-10-2003)
8. : Concurrent Versions System. http://www.cvshome.org/ (Visited 11-11-2003)
9. : MIPS. http://www.mips.com (Visited 10-10-2003)
10. : NFS development homepage. http://sourceforge.net/projects/nfs/ (Visited 12-10-2003)
11. : SAMBA. http://us1.samba.org/samba/samba.html (Visited 12-10-2003)
12. : Microsoft Visual Studio. http://msdn.microsoft.com/vstudio/ (Visited 12-10-2003)
13. : Metrowerks CodeWarrior. http://www.metrowerks.com/ (Visited 12-10-2003)
14. : Green Hills Software Inc. http://www.ghs.com/products/compiler.html (Visited 12-10-2003)
15. : GNU Compiler Collection. http://www.gnu.org/directory/gcc.html (Visited 12-10-2003)
16. : ARM Homepage. http://www.arm.com/ (Visited 09-12-2003)
17. : SPARC Homepage. http://www.sparc.com/ (Visited 09-12-2003)
18. : Crosstool. http://www.kegel.com/crosstool/ (Visited 28-10-2003)
19. : GNU C Library. http://www.gnu.org/software/libc/libc.html (Visited 28-10-2003)
20. : Glibc not Cygwin compatible. http://sources.redhat.com/ml/crossgcc/ms-10-2003g00108.html (Visited 25-11-2003)
21. : Newlib. http://sources.redhat.com/newlib/ (Visited 28-10-2003)
22. I-Logix Inc.: Concurrent Engineering Guide - Part of Rhapsody installation. I-Logix Inc. (2003)