DEVELOPING PROJECT MANAGEMENT AND QUALITY CONTROL APPROACHES FOR IMPROVING THE EE4 BUILDING SIMULATION SOFTWARE

Robert Calla, Phylroy Lopez, and Paul Wyndham-Wheeler
CANMET Energy Technology Centre
Ottawa, Ontario
rcalla@nrcan.gc.ca, plopez@nrcan.gc.ca, pwheeler@nrcan.gc.ca

ABSTRACT

In 1997, Natural Resources Canada (NRCan) developed a highly specialized building energy compliance-checking software called EE4. The need to release a better quality product and to make enhancements and maintenance of the EE4 software more efficient led to several initiatives on behalf of the CANMET Energy Technology Centre (CETC): the development of a detailed project management database to track the various types of work and their level of progress, the implementation of a coding standard, the incorporation of some software configuration management practices and the development of an automated set of testing procedures. These are the initiatives which are discussed in this paper.

INTRODUCTION

Complex building simulation software which are actively used to support energy efficiency programs can become a nightmare to maintain if you do not establish well thought out procedures and strategies for managing the product and improving its quality. This paper describes 4 major initiatives, which CETC incorporated in order to better manage the EE4 building energy compliance-checking software:

- Documenting and implementing formal procedures for tracking the project's progress and other related day-to-day maintenance activities.
- Adopting some software configuration management (SCM) practices.
- Adopting a coding standard.
- Moving towards better-structured testing procedures.

PROJECT MANAGEMENT INITIATIVES

There are different levels of maturity within any organization with respect to project management procedures. When CETC began work on the EE4 software, its level of application of project management procedures to the project was relatively immature.

During this initial phase, we had a highly informal process of requesting and tracking changes to the software. Our bug reporting procedure was (and still is) based on clients' e-mail and telephone-based reporting of problems to the project manager. Changes were then verbally requested by the project manager, based on these comments. They were however uncontrolled and the priority level attached to them was often vague. Two variations of the EE4 software were being supported; one which was used to demonstrate compliance with Canada's Model National Energy Code for Buildings (EE4 CODE) and the other for NRCan's Commercial Buildings Incentive Program (EE4 CBIP). English and French versions were supported for each. Separate project managers were managing EE4 CODE and EE4 CBIP at the time. If changes were required to any of these 2 software, the manager of the project in question would make the request to the programmer - sometimes unbeknown to the other manager. This informal method of requesting changes began breaking down when the number of change requests highly increased as the programs which EE4 supported increased in exposure. The programmer had no indication as to which tasks took priority and in some cases did not know all the required details concerning the requested changes. It was becoming apparent that a better process for handling these requests was needed.

The next phase consisted of developing a template file for documenting change requests. The file contained various types of information fields that were required to describe the problem and the required solution. The template allowed the user to specify which program and linguistic version of EE4 required the change. The communication level between the 2 project managers was also better than in the initial phase in that they now were deciding amongst themselves which tasks took priority. This was necessary as there was only one programmer conducting changes to both versions of

EE4 at the time. Each change request required that a new file be created with the template and submitted to the programmer. All files were stored in a folder on the programmer's local computer. However, no one else knew where they were located and neither did either project manager know what the level of progress was with respect to each task without directly approaching the programmer to enquire about it. Figure 1 shows a partial view of the template.

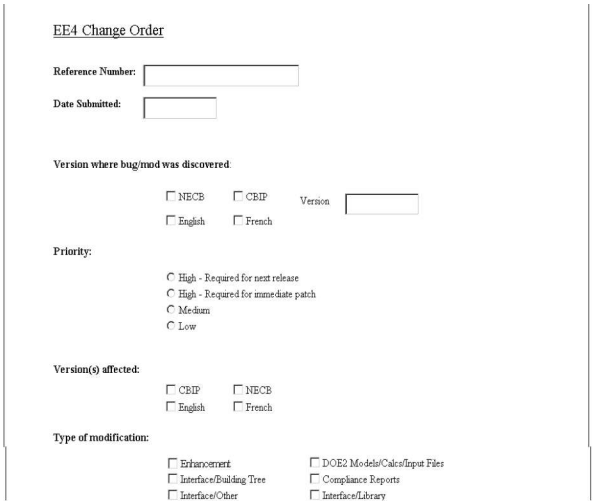

Figure 1: EE4 Change Request Template

The third phase consisted of developing a database to centrally store the data contained in the various files created using the template. Originally, the purchase of an off-the-shelf application was considered, but to allow us the flexibility to make changes as we progressed through various levels of our project management initiatives, it was decided to develop an in-house solution. A Canadian university's IT department provided us with a model database they had devised to track problems related to one of their software. They developed a format that was easy to use and modify and which was an acceptable solution for the multiple platforms they were running off of.

Before actually jumping into the development of the database, additional care was taken in exploring what other items should be included in the change request forms. Once this was finalized, a highly formal database was compiled and the information was transferred to it from the various "loose files". Figures 2 and 3 show the database we are currently using to create and track change requests.

The project manager determines which requests actually make it into the database and sets the priority level. All bug reports are not automatically entered into the database upon report from a client. One of the engineers will usually investigate the request to determine if it actually is a problem before submitting it to the project manager for approval and entry into the database. The project manager together with the team members then determines who is responsible for which part of the task. This is done during periodic (weekly to bi-weekly) meetings. The database is fully searchable and

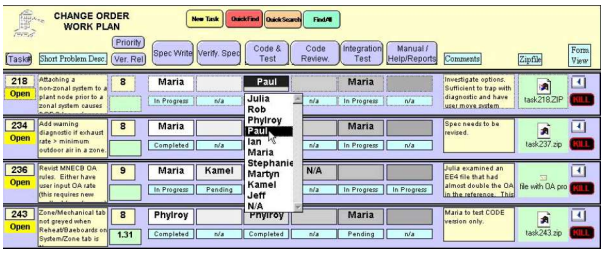

Figure 2: Change Request Database - List View

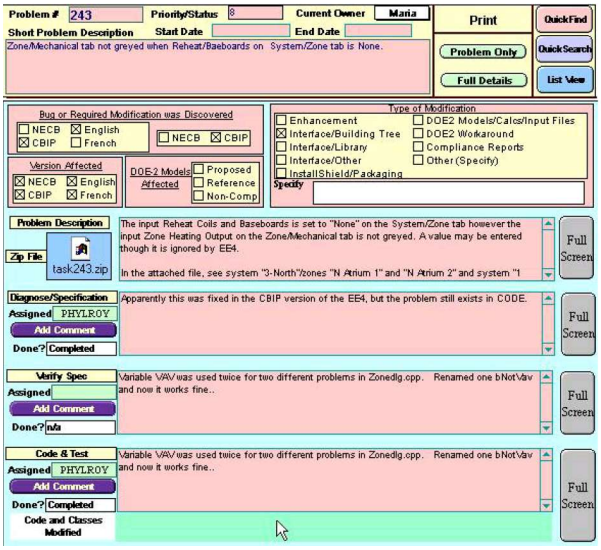

Figure 3: Change Request Database – Form View

there are two project views allowing us to see different levels of detail. One view is primarily used for the project manager's resource allocation strategy for each of the tasks, and the other for describing the task's unique details.

Developing the change request database was a big part of our project management initiative but not the only one that was set in place. Our organization had a long-term goal of developing a formal project management culture, which if expanded and documented, could eventually help us along our path towards ISO-certification in the future.

Recognizing that this was a strong possibility, various procedural documents were compiled over a period of time for the EE4 software and stored in a central location, which could be accessed by anyone in the group. The goal of these documents was to outline proper practice in working with various aspects of the EE4 software – from source code documentation and coding to maintaining the compliance reports that come bundled with the software. Here is a list of some of our documented procedures:

- EE4_01 Repository and Document Control Procedures
- EE4_02 New Document Template
- EE4_04 Change Request Database
- EE4_06 Coding Standard
- EE4_09 Design Documents
- EE4_10 On-line Help

SOFTWARE CONFIGURATION MANAGEMENT

It became apparent to us that we were going to need to centralize all documents once we realized how rapidly the resources we had working on the project and the amount of work that was on the go was growing. Each team member had their assigned tasks and as a natural course of practice, had their working files stored on their individual PC's or on the organization's central server in a location they usually did not publicly share with the rest of the group.

Once the number of people and tasks start growing in this fashion you begin running scenarios like this through your head:

Joe: "I'm leaving – I just got a job offer that pays me $15 000 a year more than what I get here. So long!"

Manager: "What? How about giving some notice ahead of time? Wait! Where is that algorithm we worked on 2 days ago for calculating HVAC system sizing?"

Joe: " I think it's on the back of the Tim Horton's box we scribbled on. Check the second or third drawer of my desk. Careful not to smudge it. I think there's a couple of jelly doughnuts still in there. Gotta go. Have a golf game scheduled with the new boss this afternoon."

Manager: "Good grief – now we're really up the river without a paddle with our star team member gone."

What do you do in this situation? Where is all of Joe's work stored and what state is it in?" When you come to the realization that you don't like the answers to these questions, then you begin exploring ways of sharing information and keeping others in your group aware of the state of your work.

This is where the concept of software configuration management becomes very useful. SCM is a formal process that involves:

- Identifying the configuration of software products and their descriptions at given points in time.
- Systematically controlling changes to the configuration.
- Maintaining integrity and traceability of the configuration throughout the life of the software.[1]

In formal SCM, there are 4 functions: identification, control, configuration auditing and status accounting. The process of SCM can become very heavy and cumbersome and rightfully so if you have a fairly large project which you would like to put under the strictest of control. It is beyond the scope of this paper to explain the process of software configuration management but the reader is encouraged to visit the following Department of Defence mirrored website to get an introduction as to what software configuration management entails:

http://www.pogner.demon.co.uk/mil_498/

In Karl Wiegers' Paper entitled "Twenty-One Project Management Success Tips"[2] one of the tips he presents is "Identify project drivers, constraints, and degrees of freedom." Although it is ideal to follow all the functions of SCM in their entirety, many times, reality does not permit it. So it is left to the judgement of the person developing the SCM plan to evaluate constraints within the organization and implement those parts of SCM which would have the most positive impact. Typical constraints may be limited budget or lack of employer awareness with respect to the advantages of good project management and software configuration management processes.

Some organizations do not have the resources to allocate a full time person(s) to the overseeing of proper SCM processes. They instead rely on the individual team members to do their own part in following agreed upon SCM procedures. This means that extra work is usually loaded onto already very burdened team members. Don't be surprised if you come across

resistance from your team members in attempting to implement an SCM process because of this very reason. When implementing change within an organization, no matter how small, you will usually come up against some resistance. Positive motivation usually works a lot better than negative reinforcement in getting aspects of SCM implemented (in other words bribe your team with goodies like donuts or cookies or other gifts). In our particular case, getting the team to go through stage 1 – identifying configuration items (CI's) was somewhat relatable to pulling teeth and required this type of positive reinforcement. Once all the items were identified and were confirmed to be items that we wanted to control, they were placed in the central repository on a dedicated server for EE4.What are some examples of configuration items? Here are a few of many with respect to EE4: source code, weather files which we distribute with EE4, files which will be compiled to make up the on-line help, compliance report template files and user guides.

Next, we had to tackle the control function which consists of reviewing and assessing the impact of proposed changes to CI's, the tracking of approved changes and the recording of rejected changes. Explaining how we implemented this function for all our CI's would be too lengthy for this paper. Instead we will demonstrate how this function was handled from a "tracking" perspective using EE4 source code as the configuration item being controlled.

After having centralized the source code on our dedicated EE4 server, we needed to develop a systematic and orderly procedure for 3 programmers to access the source code in order to effect the required changes. The following are properties that we felt were required for the third party tool which would help us control and track our source code changes and additions. It had to have the ability to:

1. label all components of the source code at any point in time (baseline) and rollback to this state if need be
2. limit modifications of any specific file to one person at a time and lock out any users from making simultaneous changes to that file
3. track who made changes to which file and for what purpose (have a change history list)
4. easily compare differences between various versions of a file (differential analysis tool)
5. be quick and easy for a new user to learn
6. be relatively inexpensive

In searching for such a tool we easily came across half a dozen different options. In looking closely at their capabilities, most had the basics which we listed above. Others also had more specialized features which we may or may not have required in the future.

Once we purchased the tool, we had to agree on how to effectively use it. Here were some of the basic rules we agreed upon:

1. All users would get the latest version of source code on a periodic basis to load onto their local computer. Once per day (usually at the beginning of the day) was considered good practice.
2. Changes or enhancements with respect to source code would be performed from the user's local computer and tested. Once the programmer felt that changes were implemented properly, only then would he check out the file(s) from the central repository database, make the changes and check it (them) back in.
3. When a file went through the checkin/checkout procedure, we agreed that a comment would be appended to the file concerning the type of change effected to it. The tool automatically documents when the new/modified file was added to the database and by whom.
4. At every new release, the entire source code set is baselined so that if need be, we could rollback and recompile that exact release version.

A few issues had to be clarified as we began investigating setting up a dedicated central server for EE4. Did we want to set up the server as a file server or as an application server? Our budget was a determining factor. Because there was not big support for purchasing an expensive server, we decided to convert one of our current computers into a file server and performed some minor upgrades. We used an existing Pentium Pro computer which had 256 MB RAM and upgraded the hard drive to a 20 Gb capacity. The processor speed was a determining factor in having it used as a file server and having all users do actual compiling, processing, etc. of files from their local computers. We also needed to designate one of the team members as the server administrator. The administrator would maintain the levels of access and perform other day-to-day maintenance activities.

A CODING STANDARD - WHAT IS IT AND WHAT IS IT USED FOR?

We all have a clear understanding of the role software programs like EE4 play in the building industry. The mystery is, how is a well written and robust software application developed?

The coding standard is a small part of the software project, but it has a large impact on the final result and in the software's maintenance and extendibility. It provides the rules by which code is written. If these rules are followed, then the result is a consistent, maintainable and extendable piece of software. In brief, the rules you will find outlined in a coding standard should be similar to the following:

1. Each item in the standard should be labelled as [M] mandatory, [R] recommended or [S] suggested
2. **General recommendations** - (i.e. optimize code only if performance is a problem).
3. **Naming convention** - (i.e. C++ file extension must be cpp; variables must be prefixed with a type descriptor – uiDeviceCount). Source code should be self-documenting through the use of clear variable, subroutine and constant names.
4. **Coding syntax and format** - (i.e. white space and brace placement). A consistent style contributes to clarity, readability and understanding.
5. **The use of function prototypes -** (i.e. void GetDeviceName(char *pName)). This step will help avoid function calling syntax errors.
6. **Compilation rules** - try to avoid conditional compilation. Conditional compilation includes or excludes code at compile time and makes the understanding of code difficult and perhaps misleading.
7. **Fault handling** – check the return codes from all functions, even if these functions seem foolproof. Bad data can be caused by a variety of things and must always be trapped.

Implementing these rules will help software developers produce readable code and develop good consistent coding practices.

To understand the impact of a coding standard, you first have to understand the rules by which software developers work. We are now going to let you into the secret world of software development, by revealing some software developers' working paradigms. In this way we hope to illustrate the value of a good coding standard.

Some software developers try to achieve legacy code guru status through the following tried and true methods of coding:

1. *Save file white space wherever possible. Never line up brackets or write code in a neat and tidy fashion.*

i.e.

```
void TDOE2Data::Calculate(TBuilding& Building,
stringFName,stringWName,boolDelete,TDiagnostic
*Diagnostic){stringpath=TApp::Path+"DOE\\";FN
ame=path+FName+"-
"+Name;stringIName=FName+".doe";string
BName=FName+".bdl";stringOName=Fname
+".sim";WName.prepend(TApp::Path+"Weather\\"
);RunDoe2(path,IName,WName,true);GetResults(F
Name,Building,Diagnostic);if(Delete){DeleteFile(INa
me.c_str());DeleteFile(BName.c_str());DeleteFile(O
Name.c_str());}}
```

Code should be readable and clear. Following a few simple format rules helps other people understand the structure and data flow of the program.

2. *Sacrifice clarity and simplicity - simple and straightforward code is to be avoided at all costs. If your design can be understood in a code walkthrough then you have failed.*

   It is a good possibility that other people will eventually have to work with the source code you have written. Try to make their lives as easy as possible.

3. *Vary coding format - never do things in the same way twice.*

i.e:

```
if( y>9 ) { y=100; } else { y-=200; }

y=x>9?100:200;
```

A consistent coding style is a great aid to clarity and readability.

4. *Continually add to the source lines in a file so that it is impossible to comprehend the intent of the functions that are contained in the file.*

It is very difficult to grasp and understand code that continues for hundreds of lines.

5. *Use the minimalist naming convention. Only use one or two characters for variable names, particularly global variables.*

Following a coding standard naming convention contributes to code readability and design understanding.

6. *Do not document your code (it takes too much valuable time and someone might find your comments useful). If you really feel compelled to place a comment in your code, make it glib and cute:*

i.e. /* don't call the police here */

i.e. TestYear=1967; /* my birthday */

7. *Do not implement any error checking (the bugs in your code will be found after the product is delivered and you will probably be gone by then).*

All functions should trap errors. A function may have to deal with bad data originating from an outside source.

A coding standard is of course not the only part of a software project but it is an important part and a project team that embraces a coding standard in conjunction with code walkthroughs, will produce a quality product that is maintainable, expandable and reliable.

Most large corporations, such as Nortel and IBM insist on enforcing the use of a code standard. They know that to meet market demands in a timely fashion, code must be easily modifiable. That means that it must be in a form that is readable understandable and robust.

We have included several URL's for you to investigate coding conventions and standards which are followed by other organizations:

http://www.possibility.com/Cpp/CppCodingStandard.html

http://oss.software.ibm.com/icu/userguide/conventions.html

http://g.cs.oswego.edu/dl/html/javaCodingStd.html

http://www.cs.mcgill.ca/resourcepages/indian-hill.html

SOFTWARE TESTING

Initially, our testing consisted of compiling and creating builds for all 4 versions of EE4 and having six engineers install them on their computers and perform runs with their own set of unique test files. They would then visually compare the results to those obtained using the previous version. The team members would perform this concurrently. When an error was reported, all testing was halted, the engineer/ programmer would patch the problem, and then the procedure would begin again. This testing strategy was flawed in several respects:

1. **The engineers were testing their own work.** A third party not familiar with the intricacies of the code and who can just interpret the output should ideally perform testing. Programmers, while they are often very diligent, can miss things when their 'nose is close to the grindstone' Programmers are largely interested in coding, not testing for bugs. Someone with a fresh perspective may be able to more easily spot potential problems.

2. **A great deal of time was wasted.** 6 engineers testing concurrently in an unstructured fashion can be frustrating especially when their test cases pass but someone else's doesn't. In this event, the entire process has to be repeated by all engineers regardless of their previous testing results once a fix has been implemented. This can be frustrating and lead to carelessness. This in turn allows for bugs to go undetected.

3. **Testing was inconsistent.** Having people manually do most of the testing introduced human error.

To remedy this situation, new measures were taken to make testing more efficient. A testing plan was created which took into account our current constraints and resource availability.

A test plan is an integral part of any software development strategy. "Testing verifies that a software deliverable conforms precisely to the functional and design specifications that have been agreed to with the users." [3]

While this is the accepted definition, software testing is also used to detect bugs in the software. Even the best programmers will inadvertently produce bugs in their code. The strategy should not be to detect and eliminate all of them. This would require more resources than most organizations can devote. Generally, "bugs are found and eliminated exponentially".[3] That is, as time invested in finding bugs increases, less bugs will be

found. So a cut-off point for testing must be established based on available resources. Emphasis should be placed on eliminating most bugs before shipping the deliverable. Shipping a stable, capable program is possible, but assuming that it is completely bug free is not.

Before testing begins you need to know the requirements of the program. Requirements are the ranges that the program operates under, and the behaviour that the program must exhibit as specified by the designer. Testing should be performed against these requirements. Many types of testing can be undertaken for quality assurance of a software. Here are a few definitions:

- Unit testing: Testing every file or class of code on it's own. This is expensive and time consuming. Every modification to a class requires a modification to the unit testing strategy.

- Integration testing: Testing modules or groups of classes and how they interact with each other and ensuring that no conflicts arise. It is concerned with confirming that no problems arise out of combining unit components into more complex processes. For example, two separate functions might appear to test successfully on their own, but if function B is fed the output of function A as one of its parameters, it might not perform as expected.

- System testing: Concerned with the full build of an application to ensure that system behaviour is correct. Installation, GUI interfaces, registry access, performance, error recovery and error handling are all checked in this process. .

- Regression testing: "The repetition of previously run tests after changes have been made to the source code". [4] This is to ensure that the product still functions in the same manner as it did before. This does not mean that the results have to be exactly the same as before. The changes made to the code might lead to different results. However the results have to be explainable.

- Inspections/Code Reviews: A team of programmers looks at and evaluates the code. It ensures that code is efficient, meets the requirements and follows the organization's accepted coding standards.

- User Acceptance Testing /Beta Testing: This type of testing allows us to see how the software will function in the wild. There are many different versions of operating systems, hardware and third party software that may interact with your products and you will not know the full effect of these interactions until regular users put it through its paces. Finding good testers is the real work here. Users who are familiar with their hardware and the software are ideal candidates.

Unit and integration testing can ensure a very high quality product if they are done correctly. However, the process is very expensive and time consuming and could conceivably double development time. Implementing unit testing on the existing EE4 code would take months if not years due to the sheer volume of code.

A dedicated staff member now conducts system and regression testing procedures for EE4. The tester manually installs all versions of EE4 and ensures that the installation wizard functions correctly. An automated testing protocol (autotester) was developed to run a series of EE4 building test files which contain HVAC system types and scenarios which users would normally incorporate into the modelling of their buildings. We have called these our 'base cases'. They are kept in read-only format on the EE4 server and are a configuration item and therefore subject to the control function of our SCM process. The autotester runs these files with the current release of EE4 and then compares the results to a saved set of output from the previously released version. It then provides a report which details the differences in the output between the new and the old version of EE4. If new functionality is added to the software that will change the output significantly, for example a humidity control which will change the results of the energy use in buildings, select cases are examined manually to predict the percent difference in other base cases output. With this we can normalise the new result set to the old results set. The tester also verifies all dialogs by iterating though them to automatically create a building and to ensure that all dialogs are correctly presented and in the correct language.

Figures 4 and 5 show the interface of the autotester.

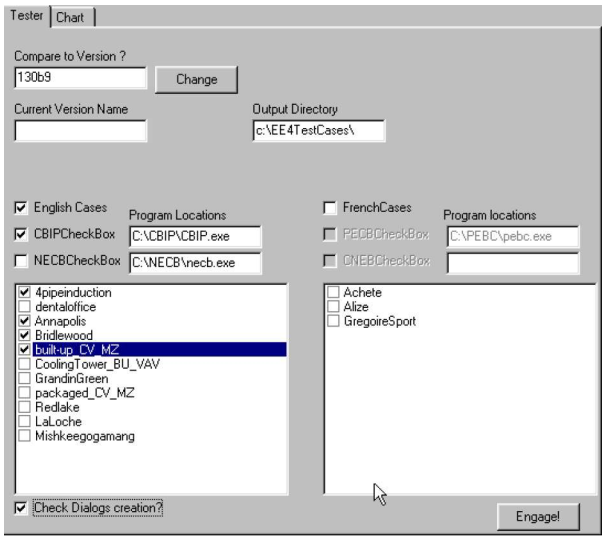

Figure 4: Autotester – User Options Screen

FileName	FileCreated?	FileSame?	Beps % Difference
4pipeinduction_CBIP_Proposed_SIM	True	True	0
4pipeinduction_CBIP_Non-Compliance_SIM	True	False	0
4pipeinduction_CBIP_Reference_SIM	True	True	0
4pipeinduction_CBIP_Proposed_DOE	True	True	
4pipeinduction_CBIP_Non-Compliance_DOE	True	False	
4pipeinduction_CBIP_Reference_DOE	True	True	

Figure 5: Autotester – Results Screen

Our final test is to go through our checklist to verify that all the help files and documentation have been updated and to determine if the program functions correctly. Figure 6 shows the checklist we are currently using.

CONCLUSIONS

Modifying and improving a large and complicated piece of software is not an easy task and should incorporate several initiatives. Making sure that good project management practices have been put in place is an important first step. Communication amongst the team members is enhanced through the use of configuration management practices. The implementation of proper SCM practices allows for a quick way to determine the status of your project at any point in time. An organization that implements a coding standard recognizes the inherent value in writing consistent and clear code – mainly less timely maintenance-related activities. Developing good software testing procedures is also an indication of an organization's commitment to releasing a quality product. These initiatives enable a team to function with synergy for software production. Time is used effectively, confusion is avoided and quality is assured. These were the major initiatives undertaken by the EE4 team in releasing a better quality product to the Canadian building industry.

1. EE4 Environment
- The correct executable is in the Install (check time stamp with most recent compiled version)Notice screen text is up-to-date
- Upon EE4 start-up you do not receive any messages that certain files cannot be loaded
- All default libraries are present and in the appropriate language
- "About EE4" dialogue box indicates the correct version number
- Text in the "User Information" dialogue appears in the correct language
- EE4 Help File can be launched from within EE4
- Windows Help can be launched from within EE4
- The list of recent files under the "File" menu is blank
- The Help File is up to date

2. InstallShield
- Title and version appear correctly in the first "Welcome Screen"
- Software licence text is up-to-date
- Readme text is up-to-date (Known bugs, planned enhancements, FAQ)
- Destination Location directory appears correctly
- EE4 can be opened from Program Groupbox immediately after installation
- EE4 Help File can be opened from Program Groupbox immediately after installation
- Readme file can be opened from Program Groupbox immediately after installation
- Comply can be opened from the Windows Start menu
- Comply Help File can be opened from the Start menu
- Readme file can be opened from the Start menu
- Uninstall works correctly
- User Guide is up to date

3. EE4 Execution
- You are able to load building files from previous versions (test ~ 3)
- The sample building(s) included return simulation results for all 3 building models
- File…Print shows the compliance reports checkbox items in their appropriate language
- The Compliance Reports print in the appropriate language
- The formatting and text for all compliance reports appear properly (test both E/F)
- All context sensitive help and on-line help are up to date (refer to detailed list compiled by Mick)

4. User Guide
- Make sure user guide is up to date

Figure 6: Final EE4 Release Checklist

REFERENCES

1 - Learning Tree International – Software Configuration Management Course Manual p342-1-10

2 - Karl Wiegers, Process Impact "Twenty One Project Management Success Tips" – Proceedings of the Borland 11th Annual Conference

3 - Microsoft – MSDN On-Line Library "The Purpose of Testing"

http://msdn.microsoft.com/library/books/advnvb5/Ch10_2.htm

4 - Microsoft – MSDN On-Line Library "Regression Testing"

http://msdn.microsoft.com/library/books/advnvb5/Ch10_8.htm