Configuration Management: White Paper

DECEMBER 2002

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk
Version: 1.0

trinem consulting limited

Table of Contents

Table of Contents 2
Configuration Management: White Paper 3
  Introduction 3
The History of Configuration Management 4
  A Brief History 4
The Software Development Life Cycle 5
  Version Control & CM 5
    Version & Source Control 5
    Configuration Management: The Life Cycle 6
    Implementing Configuration Management 8
    Supporting Configuration Management 9
    Process Improvement 11
    Return on Investment 13
  Development Processes & Methods 14
    Concurrent Development 14
    Parallel Development 16
    Third Party Development 17
    Object Dependant Development 17
    Common Object Development 17
    Emergency Fix Development 18
    Coding Standards 20
  Change & Defect Management 21
    Change Control 21
    Defect Control 21
Summary 22

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk

Version: 1.0

trinem consulting limited

Configuration Management: White Paper

Introduction

The purpose of this document is to provide an overview of the relevance of Configuration Management (CM) in any organisation that undertakes application development, in any form.

CM is usually considered something that takes a lower priority in many projects; this is mainly due to the perception that is linked with CM. Project Managers, Senior Managers, etc., concentrate on the delivery of a Project, or product, without the consideration being made to the control of *how* that Project, or product, is *controlled*. This inevitably leads to techniques such as 'fire fighting' development – the development of code to fix reoccurring defects of endless problems.

These issues stem from not controlling the application code throughout the Life Cycle – the control being lost once the code 'leaves' development. Many Projects focus on the version control of their application code but have no strategy for other aspects of the Life Cycle; such as approval mechanisms, audit, software distribution, process control, environmental control, etc. Projects inevitably miss milestones and overrun their expected delivery dates; again, this could be attributed to the inefficiencies within the control of the application development. For example, development teams that do not have a stable, effective process for developing one source item by multiple developers may engineer a work around that is considerably less efficient – organisations may do this as they do not have a tool to control this process; or, in some cases, they do not have adequate knowledge of the current tool.

Another example would be the scenario where a Development Manager is tasked with the objective of maintaining a Project whilst delivering substantial changes for a future planned release schedule. Many development teams try to work these changes into the maintenance release schedule, which eventually results in the introduction of unwanted, untested and unplanned code changes.

Another area where control is greatly neglected is where emergency fixes (EFixes) are required to Production systems. This particular scenario shall be discussed in detail as almost every organisation has, or will, experience difficulties when trying to control, audit or retrospectively develop an EFix.

This White Paper will illustrate the importance of controlling application development throughout the complete Life Cycle, across the organisation's enterprise. As applications grow increasingly more complex, and involve such a wide array of technologies, the control of such applications has also become more complex. Many processes, and products, have become redundant and inefficient; this in turn has made the development of such applications more costly, more error prone and the assured quality less predictable.

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk
Version: 1.0

The History of Configuration Management

A Brief History

The first standard for Configuration Management came in 1962 from the manufacturing industry. As a result of the increasing complexity in the manufacturing of military equipment, the American Air Force authored, and published, it's configuration management standard; AFSCM 375-1.

As society grew increasingly dependant on software, the need to control software development became a priority. The manufacturing and engineering industries had imposed controls; the software engineering industry was to follow.

Today there are a number of organisations that have published their own standards – some of these are discussed in more detail later in this document – these include:

- ANSI/IEEE
- MIL
- ISO9000-3
- EIA
- AIA
- NSIA
- AEA
- EPRI
- ECMI
- SEI

The benefits for the standards that are used today are numerous; the reduced defect deployments, reduced project costs, shorter project Life Cycles, quicker defect resolution, etc. Today, especially in IT, customers expect problems to be minimal, and when they do occur, for them to be resolved quickly. Many organisations rely heavily on the availability of their systems; if there were no controls in place and defects were to be inadvertently distributed, this would have substantial financial consequences. In more extreme cases it can literally be the difference between life and death.

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk

The Software Development Life Cycle

Version Control & CM

Version & Source Control

Many organizations will develop software using in house development teams. This development is, generally, subject to some form of version control. A common example is the control of Visual Basic code using Visual Source Safe. Other controls are put in place by simulating this process; this may be accomplished by developing the source items, and subsequent compiled code, through the Integrated Development Environments (IDEs) and placing them into file structures on a designated server – this, obviously, would be done if no tool exists to control the developed items.

Versions of the application may be recorded by building subsequent file structures for new releases. This process, in our experience, is commonplace in many institutions. It does not, however, accommodate varying development methods; or, at least, make them easily achievable. These various development methods will be discussed later in this document.

The objective of either using a tool, or accomplishing the same outcome of a tool, is to identify what version of one, or more, source item has been subject to change from one release to the next. Now that we start to highlight why we are trying to identify these changes, we start to uncover more requirements that are needed for effective version control.

A straightforward example might be that a development manager wishes to see the history of changes made to one, or more, source items. Once the changes have been identified, how do the changes that are made reference the change details; was the change a result of a defect, did a Business Analyst request the change, did an End User request the change, was the change a result of an EFix, etc.?

Simple version control may be adequate to version each source item within an application, but once other considerations are introduced, when does Version Control become inadequate and Configuration Management become a necessity? In our experience, simple Version Control is never enough.

Version control may manage the source items for your application; how does this relate to the Technical Specifications, or the Functional Specifications, how are the change details referenced, how are the results of System Test referenced, how are the changes approved, how is the propagation into Production managed? These are a small number of key areas that require consideration that go beyond simple Version Control. We will discuss these issues throughout this document to educate you, the reader, how best to manage your application development.

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk

Configuration Management: The Life Cycle

Throughout this document we will be referring to 'The Life Cycle'. The Life Cycle is built from the processes, procedures and approvals that are necessary within application development. The process may start from the pre-project analysis stage, through to the deployment of the application to the Production systems. Once development is underway, the initial process may be to raise the change requests and initiate development. From this point, we would need to consider the process for initiating changes from either one, or more than one, tool.

An example of this is using Helpdesk systems, Action Remedy say, to record the change request, or defect details, from the End User. At this point the change request would normally undergo some form of validation; once it has been validated as a legitimate request it would require some approval for development to begin.

Once this change has been made the code will have to be deployed throughout the relevant testing environments and subsequently into the Production systems.

This is a very crude, high-level description of what may be considered a typical Life Cycle. From here we can introduce, and summarize, various other components such as:

- Change Management Tools
- Defect Tracking Tools
- Release Schedule
- Approval Process
- Development Method
- Software Distribution
- Technical Notes
- Audit Trail
- Back Out Procedures
- Disaster Recovery
- Archive

This fractional list starts to illustrate the fundamental differences between simple Version Control and Configuration Management. It becomes more apparent that to effectively control, and therefore ensure, the stability and integrity of the application changes, we need to maintain control of all faculties of the Application Life Cycle.

It is considered by some that because their Life Cycle is so effortless, they do not require CM. No matter how simple the perceived Life Cycle is, some form of control has to be put in place for one reason or another. These reasons are usually for the following:

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk

- Audit – legislation depicts it; the FSA, for example, requires a certain level of control of financial organizations.
- Reoccurring Defects – a defect may be resolved in a simple Life Cycle, but if the change is not correctly recorded it will be easily reintroduced.
- Parallel Development – the ability to develop code in isolation, without impacting ongoing maintenance.
- Concurrent Development – the ability to allow more than one developer to make code changes without restrictions.
- Accountability – ensuring that changes are made, approved and deployed by the authorized people, as intended.
- EFix – ensuring that the changes made in an emergency are recorded, approved and are integrated with the 'normal' development, retrospectively.

This short list does not define *all* the reasons why control needs to be implemented; what it does illustrate is that regardless of size, all application development *needs* control.

If you were the person made responsible for the Production systems, would you have the confidence, given the logistics, to rebuild your systems from the source code you have stored (either in your source control tools, or the file systems you maintain)? How many changes have been applied to your Production systems without your knowledge.

In essence, no matter how simple or complex your perceived Life Cycle is, if there are one or more people involved in your application development you will want to control everything that happens. You will want to know at least all, or some, of the following:

Ø What changes have been approved?
Ø Who approved the changes?
Ø When are the changes being implemented?
Ø Who is authorized to implement the changes?
Ø What are the change details?
Ø Who made the changes?
Ø When were the changes made?
Ø What changes were made?
Ø How many changes were made?
Ø When were the changes tested?
Ø Who tested the changes?
Ø Who raised the change request?
Ø Where, in the Life Cycle, are the planned changes?
Ø How does this release differ from the last?
Ø How does my file system differ from my source libraries?

Simple Version Control may answer a few of these questions; effectively implemented CM would answer all of them.

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk

Implementing Configuration Management

Many organizations procrastinate when it comes to CM; “It’s not a good time for us”, “We’re in the middle of a project”, “We’re going to see how the other development team do it”, “It will delay the delivery of the project”.

Inevitably, when implementing CM for a client, there will always be the issue of how the transition will unfurl. This can be achieved through straightforward planning; our experience has told us that the ‘big bang’ approach is never wise. As previously discussed, CM incorporates many elements of the application Life Cycle. If the ‘big bang’ approach were to be used this could have a very detrimental affect on the entire project, and teams.

The recommendation would be to identify the most suitable breaks within projects and convert the development teams on a team-by-team basis. Before this can happen, the planning has to involve ratifying the processes, procedures, approvals, and distribution techniques throughout the complete Life Cycle. This, therefore, does not just mean agreeing the development processes, but also the Testing teams, the Business Analysts, the Distribution teams and the Operations teams, for example.

A further consideration when implementing a new CM tool is to identify source libraries that may be shared and to distinguish between common code and shared code.

Many organizations, when implementing CM, are doing so by the direction of a tool. When an organization decides that a tool will be implemented to control the application development, it is usually at that point the organization would want to review the current working practices. There would be no advantage to an organization implementing a tool that would enforce the users to work to existing, potentially inefficient, processes.

There are few organizations, if any, that would be able to transpose their current processes directly into a CM tool without any process improvement taking place. Before the implementation of a CM tool can take place, analysis has to be carried out to ensure that the processes that will be mapped, and therefore controlled, are; identified, ratified, detailed, and that the people who will have to adhere to these processes are sufficiently educated.

Once the implementation is complete there will be a period of time when the business will have to adapt to the new working practices. The business may be tempted to revert back to old, poor, practices should there be insufficient support to guide the users throughout the implementation. Later in this document we shall highlight the importance of post-implementation support.

The implementation of CM processes and tools is not a straightforward practice; it requires many conversations and interviews to assess the correct level of control. It becomes too easy for organizations to impose so many controls that they become difficult to understand – which makes them very difficult to follow – and therefore cause bottlenecks, reducing the productivity of all those involved.

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk

In addition to implementing the correct level of business controls, it is important to understand the technical impact of implementing a CM tool. Many organizations may already have systems that control their Change Management or Software Distribution, for example. For complete control, the CM tool has to be able to integrate with these products to share important information about the application development process. There are, however, tools on the market today that are capable of controlling all of this information; making it very simple to audit, report and reference, for example, all the necessary meta data that is available.

Management, culture, or established products may dictate that integration is required. If so, it is imperative, as with most Software Development, to combine the technical knowledge with the CM knowledge.

Supporting Configuration Management

As briefly mentioned earlier in this document, the support of CM is vital for the initial implementation to succeed. It is the same principle for any new software; if an organization purchases a new tool, some level of training will be required. Most CM tools are intuitive, but now matter how basic a tool may appear there will always be the need for some form of training. CM is no exception, especially considering the various teams of users and the utilities that modern CM tools provide. One example would be the ability to model parallel development; at some point, a merge of source code is inevitable - we are familiar with one tool on the market that allows for three way interactive merges.

Effective support of the CM processes has many benefits. It may be that you are a financial organization and you are obligated to adhere to constraints and regulations set by the Financial Services Authority (FSA). Another reason may be that your internal, or external, audit teams require particular information. If the CM process is supported efficiently with an effective tool; controlling source libraries for the FSA and providing information for the audit teams becomes a effortless task.

Support can take on many guises; we have referred to the support of the CM processes and the tool that controls these processes. In addition to this particular support, we need to consider the support of the CM tool as a product. Potentially, large organizations could have hundreds of developers relying on the CM tool to control all development. The Test Managers will rely on it for the approved code changes, the Operations Team will rely on the CM tool for code that has been successfully tested, and so on. All this information is stored in a central place; this has the advantage that it is easy to maintain. The disadvantage is that if the sufficient support is not in place, everyone related to the application development process will suffer. The issues are the same for smaller teams, and therefore the support issue is just as important.

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk

Even after a successful implementation, it is possible that not all processes may have been captured. If this is the case, it is important that there is suitable support to be able to add, amend or delete the necessary processes. A proficient practice for any organization is to reassess its processes and strive to continually improve them. When implementing for an organization in a 'drip feed' affect, there is a learning curve for both the implementers and those using the CM tool. As the roll out reaches more teams the "Organization's Method" may evolve as each implementation is made.

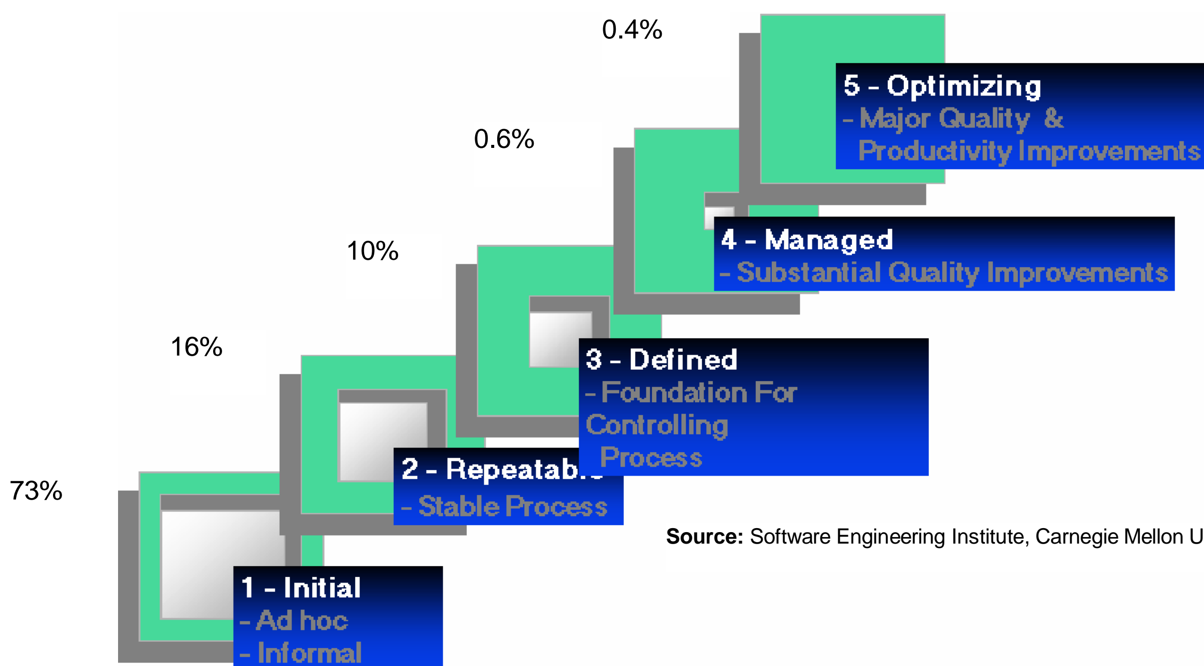
Our experience tells us that it is prudent to corroborate a minimum standard that may be used across the enterprise. This may then form the basis for a generic collection of processes and standards that are implemented across the organization for all Development Life Cycles. If the standards are generic the support is generic and therefore more easily managed; from here the process improvement is added and an assessment may be to validate the suitability for all Development Life Cycles.

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk

Process Improvement

The ability to continually assess and improve the working practices within an organization is crucial. One institution that has prepared a great deal of research in process improvement is the Software Engineering Institute (SEI) at Carnegie Mellon University (http://www.sei.cmu.edu/sei-home.html). The SEI has many proven methods for efficient, cost effective and integrated process improvement. The SEI uses a model to assess an organization on its ability to perform a process; this is called the Capability Maturity Model (CMM). This assessment would classify an organization's processes anywhere from **Initial** (or sometimes referred to as '**Chaotic**') *Ad Hoc* and *Informal*, to **Optimized** *Major Quality & Productivity Improvements.*



Source: Software Engineering Institute, Carnegie Mellon University.

This diagram illustrates the percentage of assessed organizations and the classification of their processes. From looking at this diagram, if your organization was assessed according to the SEI's standards, in which level do you think your organization would be categorized?

The benefits of implementing effective processes, according to the SEI, is that if an organization can attain level two of the CMM, the 'Repeatable Processes' stage, then the number of deployed defects would decrease by 50% and the project cycle time would reduce by 30%.

If an organization manages to reach level three, the 'Defined Processes' stage, then deployed defects are reduced by a further 20% and the project cycle is reduced by a further 10%.

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk

We can use these figures to illustrate the range of savings that may be made by process improvement. If we make a fair assumption that in today's IT market the following is true:

- Average annual salary of a permanent member of staff: £40,000[1]
- Average annual cost of a contractor: £88,400[2]

[1]Source: Computer Weekly http://www.cw360.com
[2]Source: Trinem Consulting survey.

If we then assume that an organization attains level two of the CMM, Repeatable Processes, then we should expect the number of defects to be reduced by 50%. Again, assuming that an organization has a development team that attempts to fix 16 defects per month (4 per week), and the two developers in the team take an average of six hours per defect; this would require a development time of 96 hours.

Two developers would, therefore, take 48 hours to fix their allocation of the defects. The savings from the CMM Level One to the CMM Level Two may be summarized as follows:

- Original Costs Per Permanent Staff: £999.56 (hourly rate*48 hours)
- Original Costs Per Contractor: £2,331.43 (hourly rate*48 hours)

- Level Two Permanent Staff Savings: £499.78 (orig. costs – cost of eradicated defects)
- Level Two Contractor Savings: £1,165.71 (orig. costs – cost of eradicated defects)

From this example, it is easy to see how costs may be lowered and productivity increased – costs to the project being halved, and the number of defects that may be resolved in the same period being doubled. Bearing these figures in mind, it is easy to visualize how the quality of software systems may be assured and the productivity within the Development Life Cycle increased.

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk

Return on Investment

As discussed earlier in this document, organizations may implement CM because they need to control, audit or become more accountable for their application development activities. Our experience with financial organizations has made us aware of the regulations that are required according to the Bank of England (BoE) and the Financial Services Authority (FSA).

The Code of Practice, published by the BoE, has many directives on security and source library control, etc. Examples of a very small number of these controls are as follows[1]:

- Restricted access of support staff to program source files.
- Maintenance of audit logs including access to program source libraries.
- Archiving of old versions of source programs including the operational change.
- Maintenance & copying of program libraries with strict change control procedures.

[1]Source: Bank of England Regulations, Paragraph 7.6.3.

These directives are a requirement of financial institutions. Many organizations fall short of satisfying these directives because of the lack of control throughout the CM process. By investing in adequate CM controls, penalties are avoided and, probably more importantly, the quality of software delivered to an organization's customers is of a higher standard – this in turn results in lower development and maintenance costs, and timely delivery dates.

Studies in the IT industry have shown that out of 80% of development resources spent on maintenance, nearly half of that is based on day-to-day CM type activities. This, therefore, means that 40% of an organisation's development costs are attributable to CM activities.

Bearing in mind these statistics, it is astonishing to think that a recent study has found that only 4% of organisations had a CM solution in place.

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk

Development Processes & Methods

Concurrent Development

Within Software Development there are various terms and phrases used. These terms may be interpreted slightly differently depending on which Consultant you use, or which product you implement. The following sections will define Trinem's interpretation of these terms.

The first method that requires definition is Concurrent Development. Concurrent Development is; where more than one developer develops one source item at any one time within a single Project Life Cycle, i.e. Maintenance.

Trinem Consulting is particularly familiar with the CM tool All Fusion Harvest Change Manager (**Harvest**). The basis for this familiarity is the tool's ability to perform virtually every utility required for various, and somewhat complex, development processes – a number of these will be discussed in the following sections.

Harvest has the ability to control Concurrent Development. Concurrent Development is essential in development environments where developers are continually modifying the same code base. Experience has demonstrated how, due to lack of knowledge or product capability, development teams contrive processes to 'control' the concurrency issue. It is not unknown for groups of developers to 'lock' a particular source item, make their changes, and then email it on to the next awaiting developer. This would carry on until all developments are complete and the source item 'unlocked', or checked back in.

This type of working practice is extremely unproductive and substantially increases the costs of development. CM tools, such as Harvest, should allow the ability to control the concurrency issue effectively and intuitively. Harvest allows developers to interactively merge source items, three ways – branched, trunk and combined versions.

The naming standard that Harvest uses for Concurrent Branches is fixed. If two developers wish to change the same Item of code simultaneously they could both check out the latest version for Concurrent Update. If the latest version were '2', for example, the naming would be interpreted as follows:

Branch One (Developer1)		**Main Trunk**		**Branch Two (Developer2)**
		0		
		1		
2.1.1	ç	2	è	2.2.1
2.1.2	è	3		
		M	ç	

In this example, the Main Trunk Development has occurred up until version two. At this point Developers 1 & 2 each check out version two concurrently. When a delta is checked out concurrently it will be assigned a three-part reference; the first digit referring to the Main Trunk version of which it was derived, the second digit is the Branch number, and the third digit refers to the version of that branch.

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk

In this particular example Developer one has continued to develop the Branched version, hence the '2.1.2', before it is Concurrently Merged back into the Main Trunk to produce version three. Now that the latest version that exists is later than the Branches of which they were derived, when Developer 2 checks in the Branched version a Merge Tag will be produced.

Harvest is capable of identifying when conflicts occur; it is also able to interactively merge any necessary changes developed by the programmer. In addition, this particular CM tool is able to automatically merge according to rules set by the Development Manager, i.e. passive or aggressive merging.

Effectively handling the concurrent development issue is critical to increasing the productivity of development teams; even more so when the teams are larger and the changes are more frequent, on the same code base.

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk
Version: 1.0

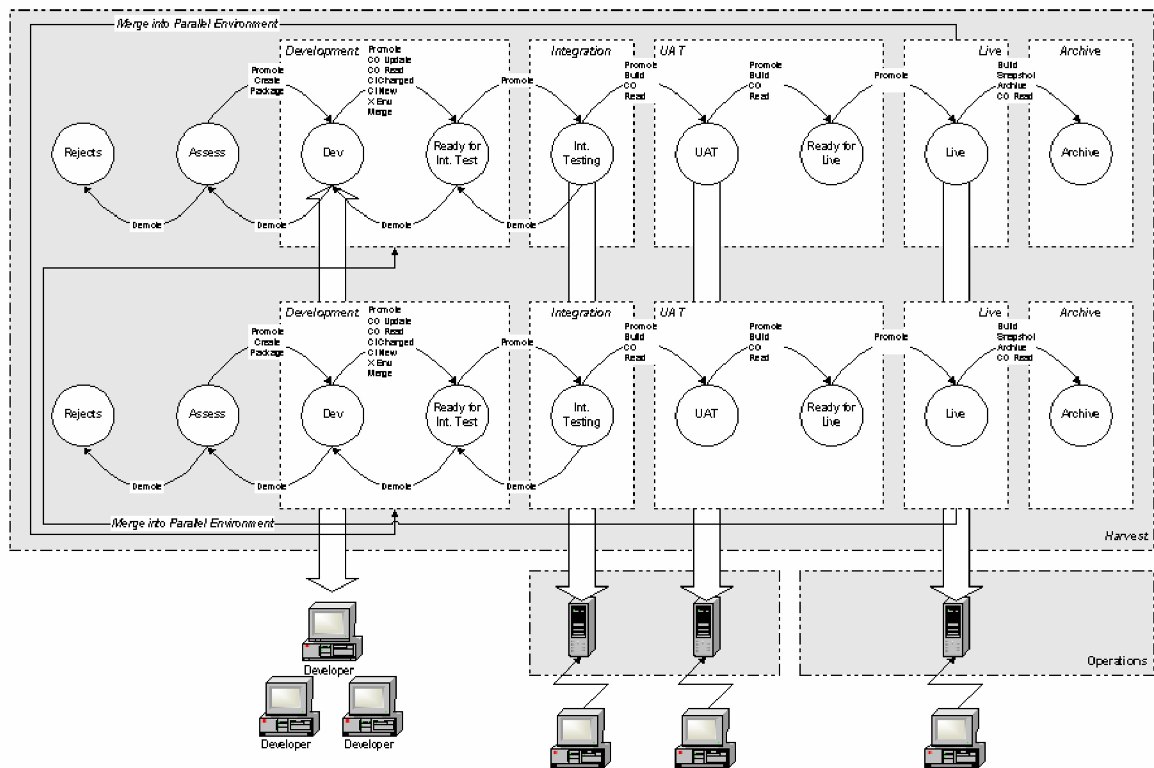
trinem consulting limited

Parallel Development

As with Concurrent Development, Parallel Development has to be interpreted and defined as its own method. Trinem's definition of Parallel Development is; the development of the same code *Base*[1] through more than one Development Life Cycle.

[1]A *Base* refers to the underlying source repository. A Base may be an initial load of code or it may be a 'virtual repository' based upon the snapshot from another Project. For example, ***Project1*** is the maintenance project; additional functionality is deemed necessary so the latest version of the application, a snapshot of Production in ***Project1***, is used to *Base* ***Project2*** – the additional functionality Project.

The key purpose of Parallel Development is to allow the multiple development of a singular application without impacting the Life Cycle that delivers the code into Production. As above, the necessity for such a method is as a result of either a complex Release Schedule, the requirement to develop for non-planned releases, or it can even be used in the event of an Emergency Fix (EFix) – EFixes will be discussed later in this document; Parallel Development is not the preferred way of resolving an EFix.



The Diagram above illustrates the ability to develop code within two separate 'streams' without impacting each other. If maintenance fixes, for example, have to be integrated with the additional functionality changes, or vice versa; this can be accomplished through Cross Project Merges. Again, Harvest has the ability to fully control and audit the entire process. In addition, all the necessary approvals and access levels may be applied so that only users with the adequate execute permissions, may invoke sensitive processes.

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk
Version: 1.0

Third Party Development

Many organizations outsource their development. It is very easy to lose sight of what changes, and for what reason, code developments were made. This is generally because the teams that are responsible for delivery of the code, to the client, usually do so in the form of compiled code. This, again, adds to the lack of control the client has of what, and when, changes were made.

Another alternative is to have the code delivered as either source, or source *and* compiled code. These deliveries may then be recorded in the client's repositories and the developments then stored. Harvest would then be able to compare all source items and identify the exact nature of the change. Many outsourced teams are less favorable to this approach as this level of control may easily highlight the development team's CM inadequacies.

Object Dependant Development

Earlier in this document we discussed the major differences between simple Version Control and Configuration Management. Another advantage that some CM tools have is that they can identify the dependencies between code objects.

If you are in the scenario where your development team use a process driven CM tool; you may come across the situation where ***Developer1*** makes a change with the knowledge that a colleague, ***Developer2***, requires to make a dependant change to another object, or module. If ***Developer1*** finishes all his, or her, necessary changes and then immediately promotes his changes through the Life Cycle; the Test Systems would be damaged as a result of the missing dependant changes. To avoid this, Harvest can be set to identify that ***Developer1's*** change are reliant on ***Developer2's*** and therefore stop the promotion of individual modifications.

Common Object Development

Many organization's produce software internally. One objective of development teams is usually to streamline the development process by identifying, and rationalizing code that may be shared throughout the organization.

For this to work successfully in an organization, there has to be the control of the common code repository to be used, and the repositories that will control team level development. Inevitably, there has to be some integration of processes to allow the common code to be built around the team level development; and, sometimes, vice versa.

The leading CM tools on the market may have the correct utilities available to control the software changes; the important feature is to ensure that the correct controls are mapped to allow the flexibility to change common code and team level code, without impacting the other systems. This is generally worked by adding the approvals on the common code repositories that will have to be ratified by all development managers.

The key element is to set this process up so that it does not restrict the development, but it allows developers the flexibility to makes the changes (which are continually audited) without the concern of impacting other systems. This form of Parallel Development, if you like, coupled with Concurrent Development would allow multiple teams to work from multiple repositories but only using one common code repository.

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk
Version: 1.0

Where this type of implementation is successful, many benefits are achieved; more productive development, fully audit changes across the organization, standardization of code, coding standards, etc.

Emergency Fix Development

We briefly touched on the ability to make Emergency Fixes (EFixes) using the Parallel Development Model. We also mentioned how this would not be the preferred solution to achieve an EFix. The main reason for this is that once Development Teams start realizing the benefits of Parallel Development, Snapshots, Cross Project Merges, etc., they find that they accumulate a number of redundant Life Cycles, or Projects.

A Project would become redundant if, for example, it was used to develop Additional Functionality (AF) and then the changes were merged with the ***Maintenance Project***. At this point the ***AF Project*** would be redundant, assuming, of course, that no further additional functionality would be required. If further additional functionality were required, another ***AF Project*** could be set up (i.e. ***AF Project II***) Based on the newly released software from the Maintenance Project.

This illustrates how easily Projects may be created – there is no limit or, necessarily, anything wrong with creating multiple Projects. Although there is nothing preventing the creation of multiple Projects, the most effective way to achieve an EFix is to make, and control, these changes through the Production state of the Life Cycle. To do this Concurrent Development would need to be implemented within the Production state of the Project. From here the CM tool would; audit the process, control the changes, and sustain the levels of access and approvals.

Although the changes are made within Production, they cannot be deployed unless done so by an authorized user. This allows the flexibility for the EFix Developer to make the necessary changes, yet still maintains the accountability with the Operations team – some sites allow EFix Developers update access to Production systems; our recommendation would be to follow the Operations procedures, i.e. only the Operations team may deploy changes to Production. The purpose of an EFix is to fast track a change that resolves a system critical defect, this should not be an opportunity to relax the controls and processes and therefore jeopardize the integrity of the systems.

Now that the changes are being controlled through the CM tool, it should be a simple process of propagating the change straight into Development. Because the EFix was coded as a concurrent change, it is now a simple process to integrate, or retrospectively fit, the EFix with normal development.

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk

When considering the functionality of the CM tool Harvest, the Emergency Fix process may be summarized as follows:

- Create EFix Package
- CO EFix Code
- CI EFix Code
- Promote to Development
- Concurrent Merge (automated/manual)
- Continue Normal Promotions Through Project

The first step is to create the EFix Package; as in Development, a Package has to exist so that code may be checked out. Once the Package exists, code may be checked out for the EFix. The only ‘mode’ available for the check out process is concurrently.

There are a number advantages to this:

- Changes may be made with reasonable flexibility and control.
- Changes cannot accidentally be promoted into Archive without ever being retrospectively fitted.
- EFix changes are easily found.
- The process of retrospective fitting can be automated.

Various scenarios may occur in relation to EFixes being made in Production. When simultaneous changes are made on the same Item, within Development, there will be resulting merge tags; identifying and incorporating the EFix development with the mainstream development will then be required.

If incorporating multiple deltas in development further complicates the scenario, an assessment will have to be made by the Development Manager as to which version should be produced. If the EFix version is merged with the latest development version (assuming development has occurred since the EFix was made), the EFix Package may only be promoted as far as the latest Main Trunk version that was merged within Development. This would mean that when the next Release occurs in Live, the EFix will have to be manually applied at deployment, and for any subsequent Releases until the retrospective fit reaches Live (when the EFix Package is promoted to Live).

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk

The following diagram may illustrate this process:

Coding Standards

When developing code it is important to ensure that all developers, regardless of team sizes, are adhering to a minimum standard. Particular information needs to be captured and the correct spacing and layout needs to be defined. In a similar way to the integration between Change & Defect Management Systems and the CM tool, controls may be put in place to ensure that code is developed to a minimum level – this may be achieved either through development reviews or through available software products.

The main purpose of coding standards is to ensure that where there may be multiple developers, and in large Projects where there is a large turnover of contracted staff, there is a minimum standard. Where there is control in the development of the code, there is a reduction in the deployed defects and the debugging of code; therefore resulting in a higher level of assured quality.

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk

Change & Defect Management

Change Control

As discussed earlier in this document, it is important to control all the information relating to requested changes to application software. Changes are more difficult to track if the actual change details are stored in one system – or a number of systems in some organizations – and the development details are stored in another, for example. If we consider all details that need to be recorded for a change to occur, it is possible that all, or some, of the following may require updated:

- Change Record
- Business Details
- Development Details
- Technical Specifications
- Functional Specifications
- Test Plan
- Test Results
- Implementation Plan
- Back Out Plan
- Roll Out Details
- Change Review

Many organizations record this information in uncontrolled file systems, in the cases of Technical and Functional Specifications. There may not be an integrated Change Management System, which means that change details are stored in multiple areas. Many details are authored and require both reviewed and approved, such as the Business and Development details. Again, these details, if recorded, are stored in third party or in house systems that do not, necessarily, have any direct reference to the application source code.

Because companies house all of this important information in such an open fashion, it becomes difficult to track and audit the entire change process. CM tools, such as Harvest, are capable of storing this information and allowing the consolidation of important change details.

Defect Control

Defect Management is very closely associated with Change Management, and so are the principles. As already discussed in Change Management, it is important to capture all the details relating to a defect fix. This would mean controlling, obviously, the defect information; who found the defect, what severity, and what area of the application, for example.

When a defect is detected, it is just as important to reference the details that describe how the defect was resolved, as finding the defect in the first place. This information may, for example, be used to investigate other detected defects.

If this data is not captured, then the ability to audit, control, fix, and manage an application's defects becomes increasingly difficult.

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk

trinem consulting limited

Summary

After reading this document, you should now have some indication of the various issues that need to be considered if you are involved with software development.

This document is by no means meant to be conclusive; the multitude of topics that are invariably linked with Configuration Management, and the level of detail that each one may be analysed, is practically immeasurable. This document should highlight some of the outstanding subjects and draw attention to the 'pain' that a large proportion of organisations suffer due to a lack of Configuration Management.

© Copyright 2002 Trinem Consulting Limited

Author: James Wilson
Email: jwilson@trinem.co.uk
Version: 1.0