
# Software Configuration & Change Management

## A White Paper: Ten Things to Consider

## Introduction

Although most Software Configuration & Change Management (SCCM) solutions address the common issues of source control, many do not address the entire SCCM process requirements. This white paper will explore ten important aspects that are worth considering before proceeding with the selection/implementation of a SCCM solution.

## #1 The change process is universal

We don't always think about Change Management as being a complex process. However, given the diversity of today's international laws and regulations, it's very complex most of the time. Let's define four different "states" pertinent to the change process.

- ✧ Every change process commences with something already in existence. That can be called the **production state** or the **monitoring state**.
- ✧ Because something is broken or no longer provides the full functionality required, a component must depart from the monitoring state and enter the **requirements** state. There we discover what is required to repair or enhance the component.
- ✧ After the requirements are defined and documented, we enter the **development** state where the component can be fixed; that's where the component resides until it is fully repaired.
- ✧ Finally, after the fix is complete, the component must be inserted back into the production state. That insertion is accomplished in the **deployment** state (sometimes it's called the **transfer** state).

Whatever you set out to change, whether it be software or anything else, these four change states are always the same. There is merit to making sure these four states are well defined. In like manner, eengineers are taught to break down complex matters to their component parts and when each piece is understood, the assembly is probably understandable too.

# #2: Desired SCCM result: documentation

In an automated SSCM process, each of the change states is typically managed by electronic documents. For example, if a user reports a bug, that is logged as a problem. Based on this problem the process will issue a change request. In a sophisticated environment, this is flagged in the problem document and details such as the date/time the change request was raised will be logged.

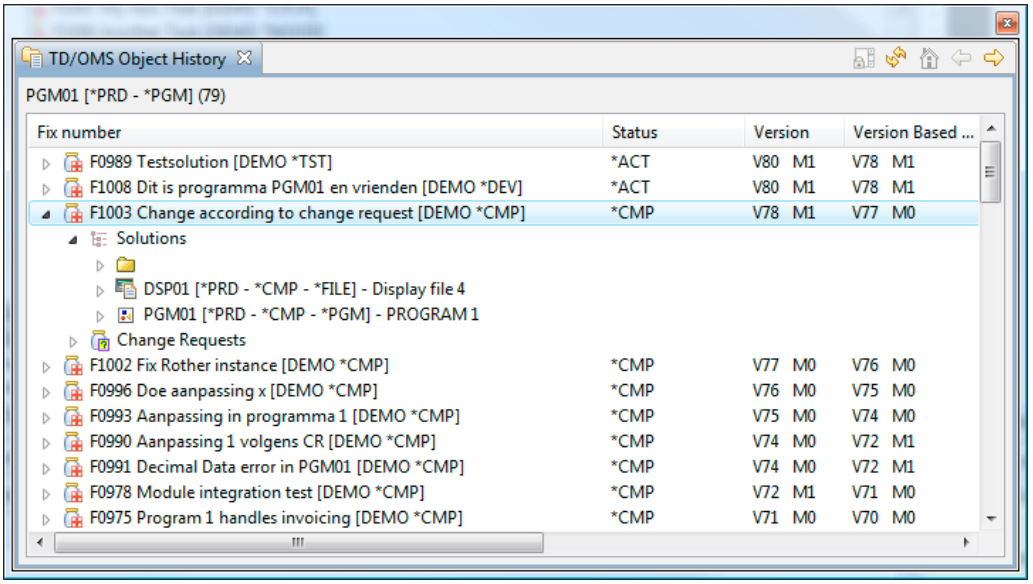
In most organizations, the change request must typically pass several review stages before authorization to assigning tasks to our developers is given. All this activity also requires proper and thorough documentation ("logging").

After the developers have received their assignments, the first chore is to figure out which software components must change to accomplish the requirements. Before allowing any change, a thorough copy must be made of the previous version so that a roll back can occur, in case something goes haywire. This previous version copy is also useful a useful benchmark to measure differences between the beginning state and subsequent versions.

After programming revisions are completed, the testers can perform their work. All of those testing efforts and results should be logged by an automatic SCCM system. Final testing signoff must also be logged before the change can enter the deployment state.

If your enterprise adopts this SCCM process using at tool like **Tight as a Drum**® – TD/OMS, then after a period of time you can assemble some very compelling software change management data for end users, for developers, and/or for your auditors (Sarbanes-Oxley or any other flavor).

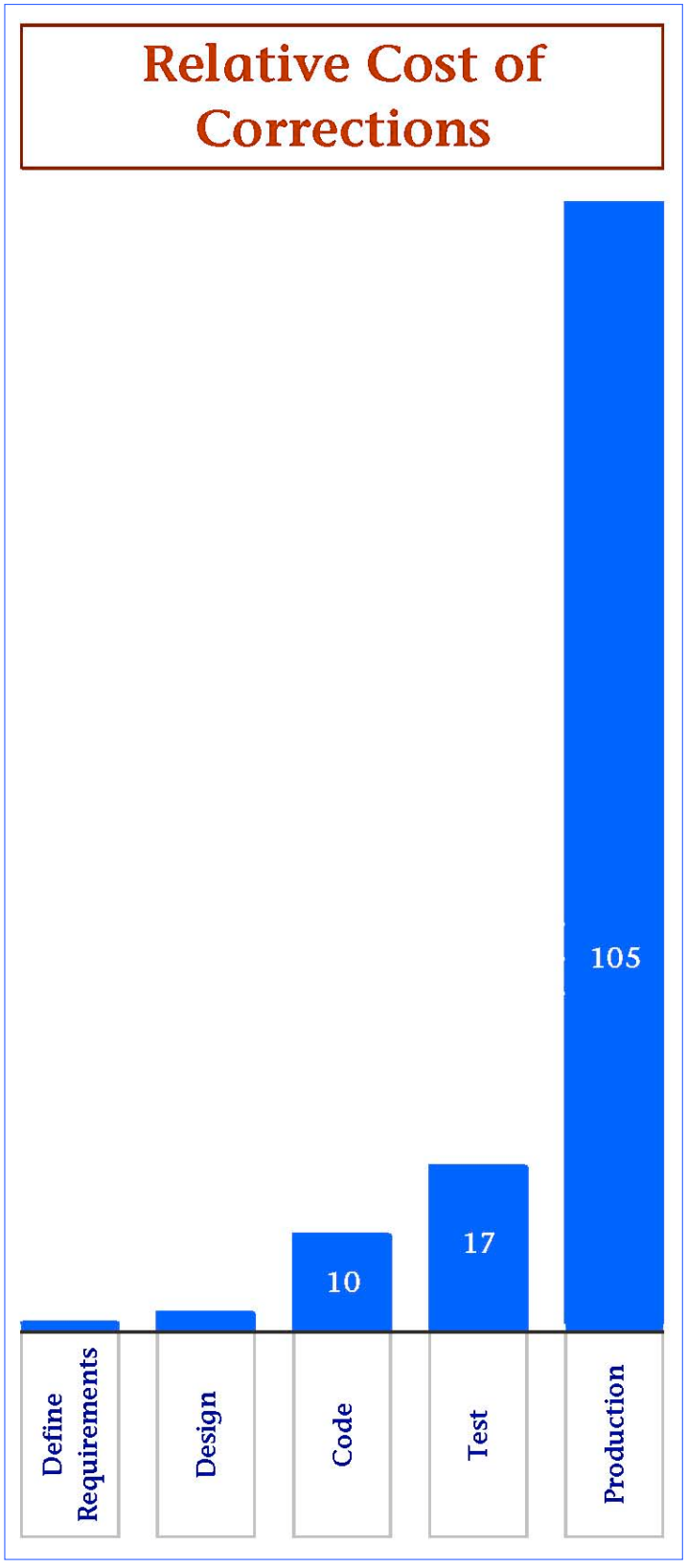
Here's an example of a drill down view from the TD/OMS system for a program called PGM01. This display can be created because the SCCM process provides automatic documentation.



## #3 Keeping bugs as inexpensive as possible

If you can address a bug early in the development life cycle, it's going to be a much less expensive bug. IT professionals know this is true, but companies don't act like they fully believe it. Why? Because companies don't measure this with a post-completion bug cost analysis.

Various studies have arrived at the data depicted in this diagram. If you fix a bug during the Test phase it will have a relative cost factor of 17. However, fixing the same bug later in production will have a relative cost factor of 105 measured on the same scale.




Here's the rationale behind the study findings:

Typically, when a serious bug is reported in production: anyone who can contribute to the solution drops attention to the productive tasks they'd been working on and re-focuses on gathering evidence about the bug and addressing the consequences of that bug. It consumes time that could have been spent more productively. That's a real opportunity cost.

So, if you must fix a bug in production instead of finding it in test, then that bug will have a cost impact which is seven times greater.


## #4 Challenges of multi-tier software development

An increasing percentage of IT shops develop for multiple tiers. "Multiple tiers" simply means that some code runs on a database backend server like the System i and other code runs on another platform such as a Wintel application server or a desktop.

Blind spots are the major challenge of multi-tier development. Dependencies between software components on the various platforms/operating systems can be (tragically) opaque. Without a comprehensive SCCM system to help, how can a developer be sure he/she knows which web application uses DB2-based data or which corporate data consolidations rely on recording transactions a particular way?

When it is time to return finished work to the production state, a comprehensive SCCM system must provide assurance that all the tiers will move over to production together. Furthermore, if one platform fails to install, we must be prepared to roll back all platforms. That's the way TD/OMS operates.

## #5 Multiple versions resurrect bugs

Many enterprises must maintain multiple versions of the same application. When you create another version of your application, you replicate the bugs that exist in your original version. Some enterprises don't believe multiple versions are necessary, but their developers still end up doing it that way for a variety of reasons. Here's a hypothetical illustration:

> *Let's say that your sophisticated product pricing/discounting module must be maintained to reflect a new sales policy. The change involves important modifications to a substantial fraction of the pertinent software components. Brian, one of your programmers, has been heads-down working it.*
>
> *In the mean time, one of your customers complains about a bug in the pricing calculations on his latest invoice. The complaint is judged to have implications beyond that single customer. Therefore, immediate remedial attention is demanded. Kirsten is assigned to iron out the just-reported invoicing complaint and her work is accomplished long before Brian's much more complicated pricing policy project can be finished.*
>
> *Now, when Brian finishes the pricing policy changes and users give thumbs-up to the testing, his programming work is deployed to production. But the objects Brian worked on won't have Kirsten's fix in them. The net effect is an impression that bugs sometimes "rise from the dead."*


## #6 A small change can generate a huge effect

The "butterfly effect" theory states that a butterfly flapping wings in Beijing can cause a hurricane in Florida. Mathematicians call it the chaos theory: it describes cause and effect relationships in complex systems (like the weather).

Let's illustrate the theory with something more finite: an automobile engine.

If we randomly punch a hole in a pipe or if we remove several fasteners, what is the effect of that change? Will something happen? Probably … but exactly what will happen depends on relationships between the pipe or the fasteners with other parts of the engine.

> If the punctured pipe is in the supply chain of the windshield wiper fluid, then the consequence of the change won't be severe. If the loosened fasteners hold the radiator together, then we have serious engine damage on the horizon.

The key to projecting the result of a change is knowing how the parts of the automobile engine are related. In like manner, relationships between pieces of your software application must be well defined to fully know (and prevent) unconstructive changes. We all know that minor adjustments out in a remote corner of the system can cause chaos in the middle of a key business process.

Given a comprehensive software configuration database (like that available in Tight as a Drum – TD/OMS software), a developer can conduct a comprehensive impact analysis before embarking on any change, no matter how small it may seem to be.

## #7 Auditors are expensive

Since a crew of SOX auditors has been known to accumulate charges at the rate of $1,000 per hour or more, the strategy is to get the auditors in and out of your office as quickly as possible. The more quickly your enterprise can present requested information to them, the the more quickly the auditors depart.

Diligent users of an SCCM system are well prepared for the question: "Can I see … " SCCM systems like TD/OMS have drill-down reporting to answer any question the auditor can invent.

There's another efficiency ….. since all applications have used the same SCCM process, it's not necessary for the auditors to look at many change episodes. Confirmation of the validity of one change episode confirms the validity of that process for all change episodes.

## #8 Living in the eye of a hurricane

IT Departments have described their situation as living in the "eye of a hurricane." By that they mean that the front side of the eye-wall is a recent painful memory and it won't take long before the other eye wall arrives with gusts of wind that tear at them from the other direction.

A good SCCM system can help an IT Department cope with those wind gusts without losing control of the code. The integrity and security of software changes is sustained despite the chaos visited by these pressures:

- ✧ Business requirements born from the need to sustain a competitive advantage
- ✧ New technology to embrace the promise of improved productivity/reduced costs
- ✧ Regulatory demands

## #9 Software bugs are invented in many departments

How many times have people in your organization immediately pointed a finger at the programmers when a new bug disrupts a business process? In defense of the programmers, why didn't the **users** find that bug before it caused a mess in production?

Unbeaten Path's opinion is that the bug hunt should get started back when the conceptual design for a new system is developed and that bug hunt should stay active until senior managers give the post-testing green light to deploy new code into production.

Testing is the answer to fewer bugs and testing efforts directed with an Impact Analysis tool like the one available with TD/OMS deliver higher quality results. That kind of tool opens the scope of the testing to include all system components which are related to the software change.

## #10 Business rules are enforced by code

Reliability of business rule application is the most compelling reason why your enterprise should consider implementing an SCCM system.

Your customers earn discounts based on a variety of parameters: total revenue over a period of time, duration of business, seasonal programs, special end-of-quarter incentives, and so on. The enforcement of these rules is captured in software.

Some enterprises have complex rules about the expiration date of products. Perhaps the sell date by lot is based on the production date for a given product you are selling. The enforcement of this rule is captured in software.

So if you change your software, you are changing the infrastructure in which your business rules must perform their duties. One minor mistake can void your business rules.

With an SCCM system, you can ensure that your rules are kept ... and ... when something inevitably requires adjustment, you will be able to quickly track down an emerging issue and resolve the problem.

## Questions ?

Click here to learn about the SCCM system we proudly market: **Tight as a Drum** – TD/OMS.

It would be a privilege to answer any questions about Software Change & Configuration Management systems. Here's Unbeaten Path International's contact information:

**Toll free North America: (888) 874-8008**
**International: (+USA) 262-681-3151**
**Send us an email** ( click **here** )

Unbeaten Path®