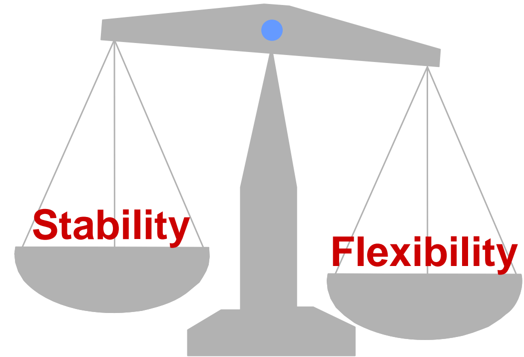
# Risk-Based Configuration Control – Balancing Flexibility with Stability

By Linda Westfall

www.westfallteam.com




There is a dichotomy in software configuration management. On one side, individual developers need the flexibility necessary to do creative work, to modify code to try out what-if scenarios, and to make mistakes, learn from them and evolve better software solutions. On the other side, teams need stability to allow code to be shared with confidence, to create builds and perform testing in a consistent environment, and to ship high-quality products with confidence. This requires an intricate balance to be maintained. Too much flexibility can result in problems including, unauthorized and/or unwanted changes, the inability to integrate software components, uncertainty about what needs to be tested and working programs that suddenly stop working. On the other hand, enforcing too much stability can result in costly bureaucratic overhead, delays in delivery, and may even require developers to ignore the process in order to get their work done.

This paper explores risk-based software configuration control. It also examines techniques that can be used to help maintain this necessary balance between flexibility and stability, as software moves through the life cycle. These techniques include:

- Selecting the appropriate type and level of control for each software artifact
- Selecting the right acquisition point for each configuration item
- Utilizing multiple-levels of formal control authority

## Risk-Based Configuration Control

So how much flexibility can we afford when it comes to controlling change to our software products and components? How much stability do we need? The answer to this, like many questions in software development, depends on risk. As illustrated with the examples in Figure 1, there are many risk indicators that need to be considered when determining the amount of necessary configuration control. For example, lower risk projects with small teams that communicate constantly while implementing small increments of functionality that are built and tested frequently can safely select a more flexible configuration control philosophy. Agile software development projects are typically an example of this type of project. Higher risk projects (or programs), with large, geographically

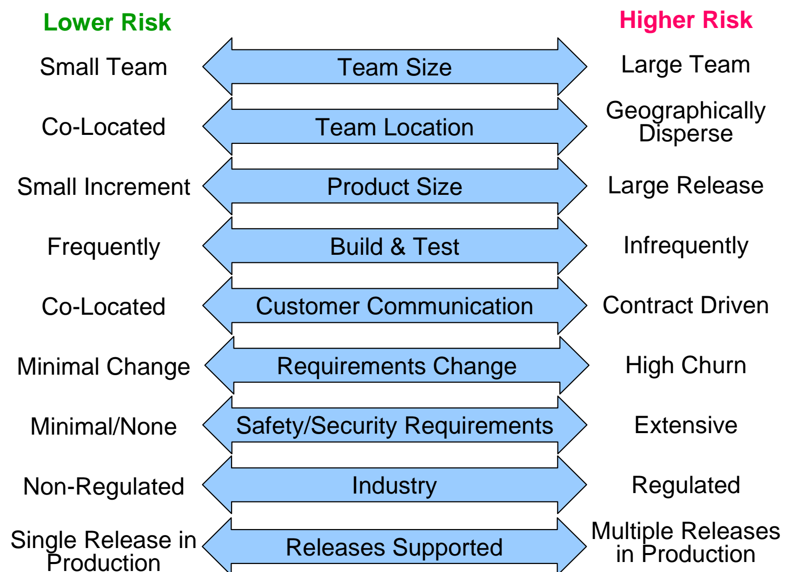



**Figure 1: Examples of Risk Indicators**

Copyright © 2007 The Westfall Team. All Rights Reserved.

dispersed teams that are implementing large software systems following a more traditional life cycle, will typically require more stability and therefore more rigorous configuration control techniques. Projects/programs in a regulated environment or with requirements for high levels of safety or security will also typically require more stability.

But the choice isn't really between complete flexibility (anyone can change anything at any time) and complete stability (everything is locked down so that change only happens through rigorous control processes). As the software product is being created, reviewed, tested internally to development, independently tested and finally released, components of that product can move through a continuum from complete flexibility, through various levels of more rigorous control, to a stability at shipment to the end-users. Decisions about when and how to implement those levels of control are part of risk based configuration control.

The level of risk for each project/program should be analyzed to determine the level of control necessary at any point in time during the life cycle to strike the appropriate level of balance between flexibility and stability for that project/program. This risk-based analysis can then be utilized to make decisions about the types and levels of control, acquisition points, and number of levels of control authority that are appropriate for each software artifact produced by that project/program. These choices are then documented as part of that project/program's software configuration management plans.

### Types & Levels of Control

The software development process produces many different software artifacts. Software artifacts are any tangible output from the software process. Examples of software artifacts include documents, code, models, reports, minutes, data, logs and notes. As illustrated in Figure 2, these artifacts may be controlled at various levels. In fact, some artifacts are temporary and therefore pose no risk if they change, so they are never placed under any kind of control. Examples of temporary software artifacts include printed program listings with the programmer's hand written annotations, weekly status reports or a scribe's personal notes from a meeting that are later used as input to the formal meeting minutes.

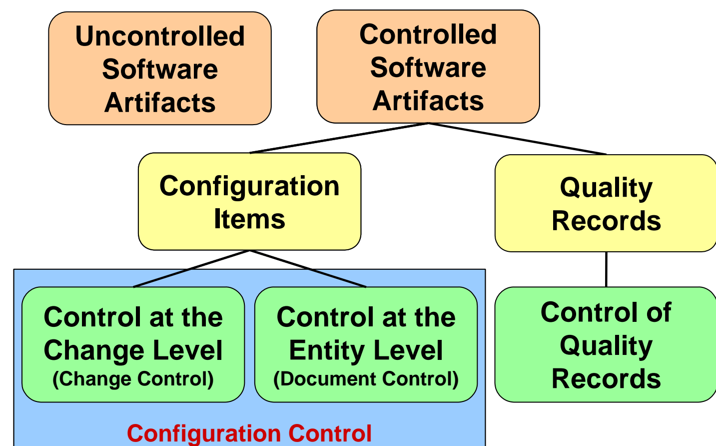



**Figure 2: Levels of Software Artifact Control**

Software artifacts that are controlled fall into two major categories. The first category of controlled software artifacts is called quality records. Quality records provide the evidence, used by management, auditors, assessors or regulators, that the appropriate quality and process activities took place and that the execution of those activities met required standards, policies and/or procedures. Examples of quality records include meeting minutes, reports, change requests, completed checklists or forms, formal sign-off/approval pages, and logs. Artifacts designated as quality records must be controlled, but are not considered to be under formal configuration control. For example, it would be considered overkill to report a correction to a set of meeting minutes by opening up a change request in the change request tool and formally approve and track that change to closure. However, procedures should be established to define the controls for the identification, storage, protection, retrieval, retention time, and disposition of quality records. Quality records should also remain legible, readily identifiable and retrievable. [Based on ISO 9001:2000]

The second category of controlled software artifacts is called configuration items. A configuration item is a controlled software artifact placed under formal configuration management and treated as a single entity. Because of the high risk of direct impact on the customers/end-users if issues arise, externally delivered software products and data (e.g., executables, source code, data, user documentation) should always be designated as configuration items. The following are examples of other software artifacts that could be classified as configuration items and be formally controlled using configuration control procedures as necessary based on risk analysis:

Copyright © 2007 The Westfall Team. All Rights Reserved.

- Designated internal software work products and data (e.g., plans, specifications, test cases and procedures)
- Designated support tools used to create or support the software product (e.g. compliers, linkers, build files) Supplier/vendor supplied software
- Customer supplied software/equipment

As also illustrated in Figure 2, there are two levels of rigor for configuration control:

1. Configuration control at the change level, called change control
2. Configuration control at the entity level, called document control

Change control is the most rigorous level of control and therefore provides a higher level of stability for configuration items controlled at that level. Change control proactively manages changes by reviewing each proposed change before it is implemented and allowing only authorized changes to be made to the configuration items. Higher risk configuration items are typically placed under this level of control. Examples of configuration items that are typically controlled through change control include requirements, interface and design specifications, and source code.

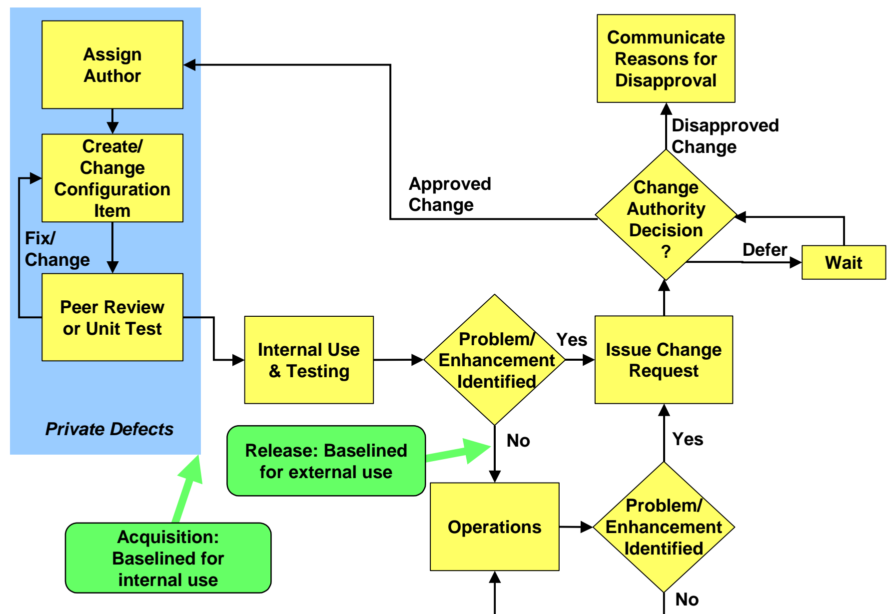



Figure 3: Change Control Process

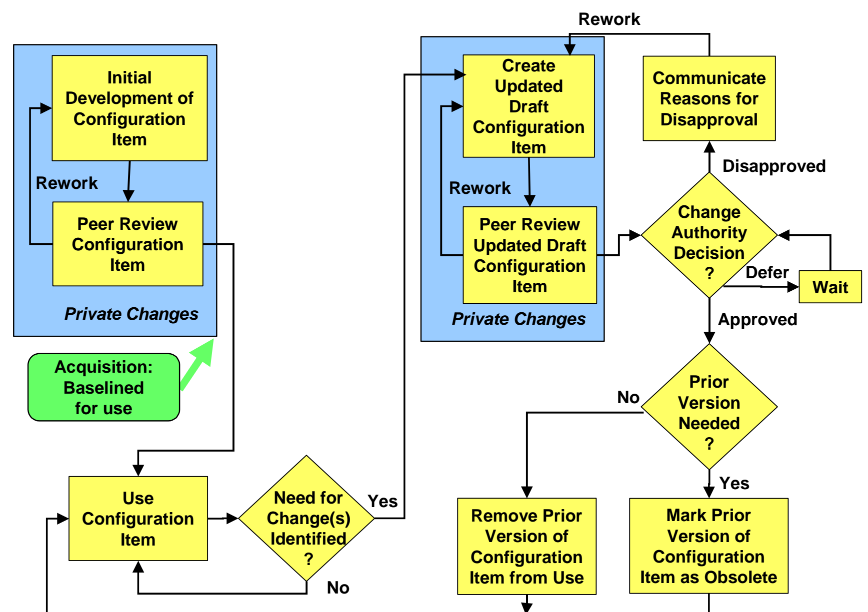



Figure 4: Document Control Process

Figure 3 illustrates the change control process. When an author is assigned to create a new configuration item, that author can make any changes necessary to create and update that product as it is being created. However, at some point that configuration item is acquired (baselined for internal use) and placed under confirmation control. As that item (and other items) is being used and tested internally by the development organization or later used in operations (production), problems or enhancements may be identified. The change control process requires that each change to baselined configuration items be formally documented in a change request and that the appropriate change authority review that request. That authority can defer that request to a later time, disapprove or approve the change. If the change request is approved, one or more authors are assigned to make the changes to the configuration items impacted by the requested change. The authors can

Copyright © 2007 The Westfall Team. All Rights Reserved.

update their assigned configuration items as needed to implement the approved change. However, if while they are implementing the approved change, another defect or enhancement is identified, that new defect or enhancement must be documented in another change request that goes back through the change control process.

Document control is a less rigorous level of configuration control than change control and can therefore be used for less risky configuration items. As illustrated in Figure 4, after the initial version of the configuration item is acquired and placed under document control; all subsequent changes are made to a draft (e.g., non-released or preliminary) version of the configuration item. That updated draft must go through a review cycle and be formally approved by the appropriate level of change authority before it is released for use. Once an updated version of the configuration item is released for use, procedures should be in place to ensure that obsolete versions of the configuration item are removed from use. If the obsolete versions must remain available for reference, they should be clearly marked to indicate that they are not the most current version of the configuration item. The document control process is more reactive in managing change than the more formal change control process because it reviews the changed configuration items after the changes are implemented in draft form. Document control, however, allows for more flexibility because multiple changes can be made to the same draft and all of those changes are approved together as a set when the draft document is approved. Document control also allows problems or enhancements found while making other changes to be implemented in the same draft without going through an approval cycle first.

**Acquisition Points**

Another important part of the software configuration management process is defining when each configuration item is initially acquired (i.e., initially baselined under configuration control). The Software Program Manager's Network says that, "one critical aspect for control of work products is the proper timing for when they enter into configuration management." [SPMN-98] As illustrated in Figure 5, quality gates are used to approve the acquiring of a configuration item (work product) and its baselining under configuration control. Examples of quality gates include the successful completion of:

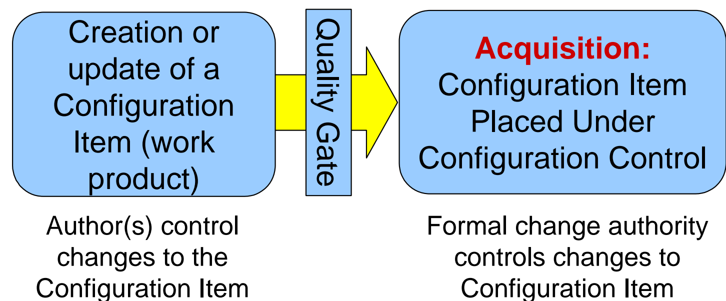



**Figure 5: Configuration Item Acquisition**

- A peer review (e.g., desk check, inspection, walkthrough)
- A test activity
- A project review (e.g., phase gate review, major milestone review)
- An independent product analysis or audit.

The configuration management plans for a project/program should define the acquisition points and associated quality gates for each configuration items. The earlier the acquisition point in the life of a configuration item, the more rigorous the level of control and formal communication about changes to that item (more stability), the later the acquisition point, the easier and quicker it is to make changes (more flexibility). The higher the risk that changes to a configuration item will create potential issues, the earlier in the life cycle the acquisition point is established for that configuration item.

For example, consider a source code module. If the acquisition point is set after peer review, then all defects found in unit, integration and system test must go through formal change control. The peer review acquisition point may be too early for most projects, but for example, if a project has an independent verification and validation (IV&V) team that does unit testing, it may provide the formality needed for the IV&V and development teams to communicate effectively. For many projects, an acquisition point for source code may be more appropriately set after unit test or integration test depending on when the hand off takes place to a testing group outside development. For Agile development teams or other small teams with high levels of internal communications, it may even be

Copyright © 2007 The Westfall Team. All Rights Reserved.

appropriate for the acquisition point to be set at the point of product release, so that only defects reported from operations (production) are subject to formal change control.

For other configuration items, like requirements or design specification, the successful completion of the peer review or of a major phase gate or milestone review may be the appropriate acquisition point. These points act as internal release points where the specification moves from creation by its authors to use by other members of the project team (development, test, technical publications), so the need for more control (stability) and more formal communication about changes and their impacts may be desirable.

### Multiple-Levels of Formal Change Authority

Formal change authorities are referred to by many names, Configuration Control Boards (CCB), Change Control Boards (CCB), Change Authority Boards (CAB) or Engineering Change Boards (ECB). For the purpose of this article, we will use CCBs to refer to these formal change control authorities.

CCBs are beneficial because they:

- Provide the authority for approving/disapproving changes to configuration items
- Provides visibility into the configuration control process and ensure communications with impacted stakeholders
- Provides a vehicle for impact analysis
- Facilitates resource allocation
- Plays an integral role in keeping the software development process under control

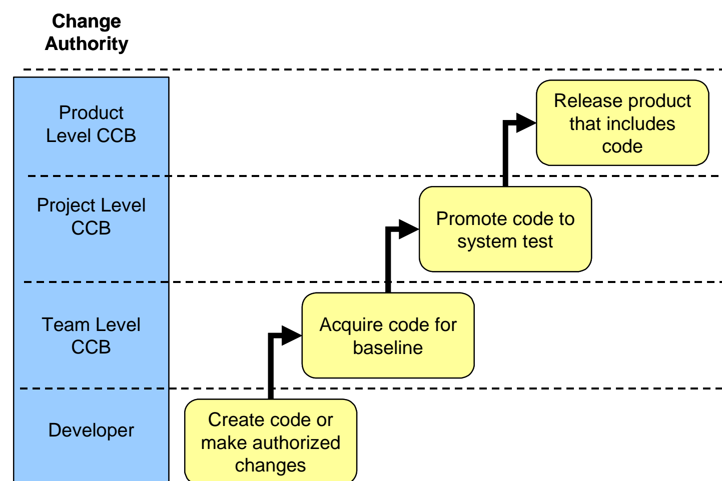



**Figure 6: Multiple-Levels of CCBs – Code Example**

Another way to create a balance between the need for rigorous control and communications (stability) and the need to expedite the change process (flexibility) is to create multiple levels of CCBs. Having multiple levels of CCBs, allows small changes, that have limited scope and impact, to be approved at lower levels of authority. While major changes, that impact multiple stakeholders or multiple work products, can be escalated to higher-level CCBs that have broader scope and involve all effected parties.

For example, as illustrated in Figure 6, the developer can change a newly created source code module as necessary. When it is initially acquired, a Team Level CCB could be assigned control change authority for that module, because typically only team members need to be consulted when the code changes at that level. As illustrated in Figure 7, the membership of the Team Level CCB

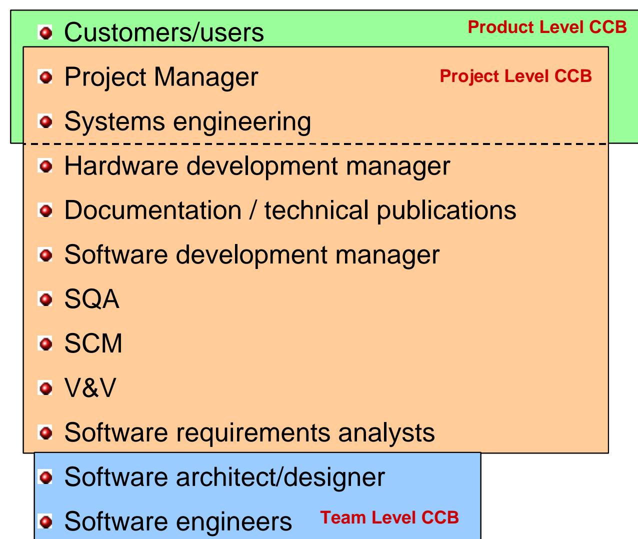



**Figure 7: Membership of Multiple-Levels of CCBs**

Copyright © 2007 The Westfall Team. All Rights Reserved.

might be limited to only the software architect/designer and the software engineers on the team. Since this CCB has limited members, that work closely together, they can usually meet and make decisions quickly.

To continue this example, once system testing starts, the source code module is promoted to the Project Level CCB (see Figure 6). Changes to that source code module at this phase of the life cycle may impact software written by other teams, the hardware and/or the documentation as well as the work of the testers, software configuration management, software quality assurance and other specialists. Changes this late in the life cycle may also impact project schedules, costs, effort and risks. The membership in the Project Level CCB expands to include representatives from the various stakeholders that may be impacted (see Figure 7). In this example, the individual team architect/design and engineering are not members of the Project Level CCB but may be called upon to participate in that CCB activities as subject matter experts if their software designs or code is impacted by the requested changes.

Finally when the product from this example is released into production, the course code module is promoted to the Product Level CCB along with all of the other configuration items that become part of the product baseline (see Figure 6). Again, the membership of the CCB changes to include the customer/user representative and management level personnel who are making business decisions about the longer-term direction of the product (see Figure 7). CCB meetings at this level typically happen much less frequently, however, and to attempt to control lower level products, early in their life cycle with this level of CCB would add an extreme time burden that would probably grind software development to a halt.

Not every configuration item needs to start at the same level of CCB. Higher risk configuration items be assigned to higher-level CCB when they are acquired. For example, as illustrated in Figure 8, the Software Requirements Specification (SRS) for this project could go directly under the control of the Project Level CCB when it is acquired because of the wider impact that changes to requirements may have across the project. The SRS is promoted to the Product Level CCB along with all of the other configuration items that become part of the product baseline when the product is released into production.

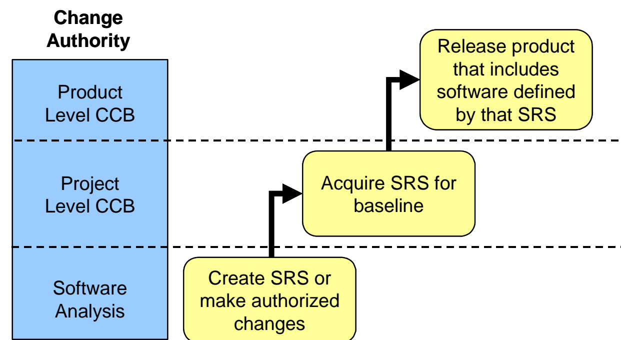



**Figure 8: Multiple-Levels of CCBs – Software Requirements Specification (SRS) Example**

The project/program's configuration management plans should define the promotion points and associated quality gates for each type of configuration item, as well as the formal change authority that owns the configuration item at each level of acquisition/promotion.

## Conclusions

Software configuration control can use multiple techniques to maintain the appropriate balance between flexibility and stability. The level of rigor used for each of these techniques should be determined on a project-by-project (program-by-program) basis depending on the results of a risk-based analysis of the software artifacts that will be produced by that project/program. One of the primary roles of the software configuration management plans for a project/program is to document and communicate these decisions.

## References

ISO 9001-2000 American National Standard, Quality Management Systems – Requirements, ANSI/ISO/ASQ Q9001-2000, American Society for Quality, Milwaukee, Wisconsin, 2000.

SPNM-98 Software Program Manager's Network, *Little Book of Configuration Management*, Computer & Concept Associates,1998 (available at www.spmn.com).

Copyright © 2007 The Westfall Team. All Rights Reserved.