Stop-Gap Configuration Management

Ted Gill
Puget Sound Naval Shipyard

Configuration management can be highly difficult to implement at a site where the software development effort has been at the ad hoc level. Particularly for pre-existing development projects, to attempt to start a full-blown configuration management process is likely to be a lethal cure. This article outlines an exceptionally simple, goal-oriented "stop-gap" configuration management plan. The purpose of the plan is to gain immediate control of the most critical aspects of a development project and build a foundation of methods and understanding upon which a more complete configuration management plan can be built.

This article describes the most vital steps to bring essential aspects of the formal configuration management (CM) discipline to a project that has had none. This is emergency information—*triage*, pure and simple. If you use this information, the probable reason is that you are having a CM emergency. The typical condition in which a project finds itself at this point is something like this:

- The schedule is already extremely tight.
- The resources are barely adequate for the basic tasks, let alone "extras" like CM activities.
- No one on the project has extensive formal CM training or experience.
- The need for CM has just become apparent due to a big CM blunder on this or another project, a management CM mandate, or a milestone looming in the near future, and CM controls do not exist.

To team members in this situation, CM may appear big and scary. However, the essentials activities of CM are probably already being carried out in the project—albeit in an informal manner.

The suggestions and steps outlined here will in no way fulfill the requirements of a complete, coordinated CM plan. But they will go a long way toward establishing core processes and will mitigate the risks posed by a loss or lack of configuration control.

The following sections describe the important items and concepts for stop-gap configuration management.

Establish Version Control of the Code

Your primary goal must be to control the actual code: source, object, and executable. The following concepts will help you gain version control.

- Baseline the current code at some meaningful point in the development.
- Define what your computer software configuration items (CSCIs) are: individual modules, subsystems, etc.
- For each CSCI, determine its lowest level of decomposition, i.e., configuration units (CUs). This decomposition sets the granularity or resolution of configuration control.
- Determine an ironclad method to identify your CUs and their revision level. (Filename, date, and time will work if there is nothing else; if you have PVCS or a source code control system, use it.)
- Document your baseline with a version description document. This document should identify the baseline to which it refers (baselines need to be uniquely identified and catalogued), the CSCI it documents, and every CU in the CSCI and its revision level.
- Assign ownership of modules to specific developers—document the ownership and hold to it.

Establish Formal Control

Establish a formal method of change control for *all* changes to the code—accept no other method to propose changes. "Formal" means "with a form" and with a specific review and approval process. A change request form must include the following fields:

Submitter fields

- Project.
- Type (problem or enhancement). *(May include more detail.)*
- Severity.
- Date submitted.
- Description.
- Reporter or submitter.
- Unique identifying number.

Change control board fields

- Action (approve/disapprove/defer[until when]).
- Date acted on.
- Justification for disapprove/defer actions *(optional).*

Fields for those who act on the change request

- Date changed.
- CSCIs and CUs affected.

Additional Change Control Issues

- Always provide feedback to the submitter.
- Always document and save change requests, even if disapproved.
- When code is modified in response to change requests, developers should always document the change request number(s) on which they are working.
- The configuration control board (CCB) (see following section) must review and approve all changes to baselined CSCIs.

Establish a Project CCB

The CCB is responsible to approve all configuration changes to baselined configuration items. This ensures that all proposed changes receive a technical analysis and review and that they are documented for tracking and auditing purposes. The board also has final responsibility for release management, e.g., establishing new baselines.

The elements of the CCB already exist in a project that does any amount of formal change control. The goal must be to officially establish the board to gain formal control over the change approval process as it affects configuration control.

The basic tasks of the CCB are to declare baselines on CSCIs (promotions, releases, etc.), to review changes to baselined CSCIs, and approve, disapprove, or defer their implementation.

The above is a short but extremely important task list. The CCB must have a stranglehold on the project. Nothing can be changed without their approval—end of discussion. For this reason, board members must be chosen carefully. The board must be composed of representatives from all affected organizations or concerns (stakeholders) such as

- Functional or user community.
- Developers.
- Test group.
- Hardware design and operation personnel.
- Interface groups.
- Database administrators.

Although every member of the CCB may not be excited about every change, be certain that some change will affect every member of the board at some time. It should not be difficult to recall past experiences when an unwise or costly mistake could have been avoided if the right people had known about a proposed change.

The chair of the CCB must be from project management—a person who can unambiguously resolve conflicts within the board and enforce the board's decisions on the project. Decisions on change implementation and CSCI promotion translate directly to fundamental project cost, schedule, and quality issues. The CCB will find that their efforts are an infuriating exercise in futility if their decisions are continually reversed or ignored by an outside entity with the real decision-making authority. Do not let this happen; put that entity in charge of the board. By doing so, those who have decision authority are directly coupled to those who have expertise on the details. Decisions of the CCB should be reached by consensus whenever possible. The group dynamic must reflect the cooperative nature of a development project. The chair must nurture this cooperative vision and take unilateral action only when all other methods have been exhausted.

Document What You Do

As a treatise on triage, this article does not suggest elaborate plans. Beyond the specific documents required above, a written record is needed of what you are doing or plan to do. These plans do not need to be more formal than memorandums to all project team members—memorandums are adequate if they convey the plan and its execution.

There are two essential attributes of the documentation portion of stop-gap CM:

- The documents must describe what to do and how to do it.
- The documents must be published and disseminated so that all involved know the project status and what is expected of them.

Software Configuration Management
What to Do When You Know You Are in Trouble

Software configuration management (CM) is a task often left until last. Consequently, motivation to do CM often comes from a fear of a project disaster rather than from faith in the control and visibility CM provides. This management by fear is akin to battlefield triage. Triage is a system to assign medical treatment priority to battlefield casualties based on urgency and chance for survival.

In the software battle, what will keep your project from bleeding to death? The answer is "stop-gap CM." Ted Gill's article is careful to point out that although stop-gap CM increases your project or organization's odds of survival over the short term, stop-gap CM will not produce the vibrant and healthy project or organization you need to compete for scarce Department of Defense funding year after year.

The Software Technology Support Center (STSC) does triage to see you through a CM crisis by assessing your situation, working with your leadership to plan a solution, and doing the necessary "hand holding" at the project level. And we do it with an eye toward the long-term health, stamina, and resilience your organization needs. On a cost-recovery basis, the STSC sees the job through to your desired state in which

- A CM organizational entity has been established.
- All CM policies and procedures are fully documented and implemented.
- The CM polices and procedures are tailorable to the size and scope of specific software development or maintenance projects.
- All software development or maintenance projects are conducted in compliance with CM policies and procedures.

The STSC – A practical approach to CM.

For more information, call
Paul Hewitt
Voice: 801-775-7775 ext. 3039
DSN 775-7775 ext. 3039
Reed Sorensen
Voice: 801-775-5555 ext. 3049
DSN 775-5555 ext. 3049
E-mail: scm@software.hill.af.mil

S T S C

Everything beyond this level of documentation is window dressing. An easy way to approach documentation is to assemble the appropriate project team members (possibly with the software engineering process group [SEPG] CM process lead) and decide how to attack a particular part of the process. Once you have a method, create a memorandum that details the decisions and send it to all project members. Keep a copy of these memorandums in a CM process notebook. That is it—nothing fancy.

If a method does not work as it should, do not be afraid to change it. Just be sure to document the process change.

Following are examples of topics that should be documented:

CSCIs
- Lists of CSCIs and their baseline status.
- Levels of decomposition of CSCIs (what constitutes a configuration unit).
- Naming conventions and standards and version and revision identification.
- Configuration unit ownership lists and policies.

Changes
- Forms to be used and guidance on filling them out.
- Process flow descriptions for change submission and processing.
- Points of contact.

CCB
- Members.
- Duties and tasks, process flow descriptions.
- Meetings schedules.
- Record of actions (minutes).

Conclusion

This set of processes is not intended for the mature software development environment. It is aimed primarily at the organization that is taking those first difficult steps toward software process improvement. These processes serve as a consciousness-raising device as much as anything else. At the early stages, I have found that the two most essential ingredients to initiate process improvement were education aimed at people not steeped in the vocabulary and dogma of the Capability Maturity Model (CMM), and a plan of action that could be achieved by beginners. If you are in a situation where you have no configuration management, these processes provide the subject matter for education and a simple plan to execute. ◆

About the Author

Ted Gill is currently employed by the U.S. Navy at Puget Sound Naval Shipyard. He was the configuration management key process lead in the shipyard's SEPG. As a charter member of the group, he was involved with the initial efforts of assessing software practice, educating developers, and moving the shipyard's software development processes toward CMM Level 2. He served as project manager for the shipyard's training and qualification tracking system development project and has developed propulsion plant demonstrator software for the shipyard's nuclear training division. He now works in database administration and as a general process consultant for the shipyard's information resources management department.

Puget Sound Naval Shipyard
Code 1233
1400 Farragut Avenue
Bremerton, WA 98314-5000
Voice: 360-476-2072
Fax: 360-476-2275
E-mail: gillt@psns.navy.mil

NSWC PHD Dam Neck Receives CMM Level 3 Rating

On Sept. 19, 1997, The Naval Surface Warfare Center (NSWC) Port Hueneme Division (PHD), Dam Neck Detachment completed its external assessment to earn a Software Engineering Institute (SEI) Capability Maturity Model (CMM) Level 3 rating. NSWC PHD, Dam Neck is the first U.S. Navy tactical real-time program developer to attain Level 3.

The road to CMM Level 3 began in 1987 with an association between NSWC PHD, Dam Neck and the SEI, whereby the former served as a CMM development beta test site and an internal software process assessment reviewer. In 1992, NSWC PHD, Dam Neck continued down the software process improvement path by conducting a benchmark assessment using CMM as a guide. This assessment resulted in the development of a Software Process Improvement Plan, and training was provided to introduce all employees to CMM. The next step was completed in 1994 with the chartering of a full-time software process engineering group, which established a process asset library and developed the Standard Software Process Definition.

In 1995, NSWC PHD, Dam Neck reorganized along product lines, which allowed the new departments to focus on quality, process, and training. As a result, process improvement was institutionalized at the organizational level. NSWC PHD, Dam Neck continued with periodic internal assessments and achieved CMM Level 2 status in 1996. The final stop on the road to CMM Level 3 was achieved on Sept. 19, 1997 when an external assessment rated the directorate CMM Level 3. Throughout this entire evolution, NSWC PHD, Dam Neck has maintained its affiliation with the SEI.