# GitHub Account Name: 7519436451@qq.com

Configuration management (CM) focuses on establishing and maintaining consistency of a product's performance, and its functional and physical attributes with its requirements, design, and operational information throughout its life. CM streamlines the delivery of software and applications by automating the build out of systems quickly and efficiently. It can be used by management and engineers to check which components have been changed and why, ensuring an audit trail of changes done to the system. This helps with quickly identifying bad configuration changes and allows for rollbacks to well-known working ones to ensure rapid restoration of service(s). This also helps developers with debugging to check if configuration changes impacts the product’s functionality.

CM does take time to setup, but if done correctly, it allows for ease of scalability and reduces the time to build out additional resources for your product without the worry of user-prone errors. CM may seem at sometimes daunting to setup and implement, but coming up with a strategy and starting out small can open the door of opportunities to remove the human abstraction layer and automate as much as possible. This allows you to really focus on making your product or service better for your customers instead of misspending time, money, and resources on maintaining your system infrastructure.

Version Management also called  Version Control or Revision Control, is a means to effectively track and control changes to a collection of related entities.  The term "Versioning" is also sometimes used but the difference is that "Versioning" typically refers to when someone manually applies a version number or label to something for easier communication or simplification such as "draft", "beta", or "1.0". In the context of an information solution version control is most often used to track and control changes to source code.  It is a very important tool within an overall life cycle management strategy for information solutions.

Even though it is most often used for source code, in reality, there are many other entities within an information solution to which version control can and where practical be applied.  This is especially true if those entities are stored within text-based files or if they can be exported to and imported from text-based files (e.g., configuration settings stored in a database).  These additional entities include but are not necessarily limited to the following:

* configuration settings (.conf, .cfg, .ini, etc.)
* templates (e.g., web page layouts, forms, etc.)
* rules (e.g., access, permissions, processing, filtering, scheduling, etc.)
* shared inclusions (e.g., headers, footers, menus, etc.)
* documentation (e.g., technical specs, user guides, operations guides, etc.)
* web sites

Over the years many different version control systems have been developed, especially in the areas of source code management and document management.  These systems may be commercial or open-source and often run as stand-alone applications; however, some software applications/suites have version control embedded as an essential feature (e.g., SAP).

Traditionally version control systems have relied upon a central repository for storing and tracking changes to managed entities (e.g., CVS, SVN, etc.).  For these systems individual users check out portions of the central repository into their own local workspaces where they make the desired changes.  Once the changes are ready they commit those changes back into the central repository.

In recent years distributed version control systems have been developed wherein there are multiple peer repositories, any one of which could potentially act as the central repository (e.g., Git).  For these systems individual users get a complete copy of a repository (i.e., clone it) and store it locally.  When making changes they check out entities from their local repository into a local workspace where they make the desired changes.  Once the changes are ready they commit those changes back into their local repository.  Those changes are then pushed/pulled into the original repository to be available to other team members.

Collaborative environments have also been developed (e.g., GForge, GitHub, GitLab, etc.) around version control systems to facilitate a team-based approach to the life cycle management of applications.  These environments generally provide the means to establish collaborative projects which include central repositories, issue tracking/management, and threaded discussions/forums.  UNL IS currently uses GitLab for its collaborative development environment.

 Key benefits of version control, especially in the area of source code, include:

* organized, coordinated management of changes to software assets by one or many individuals, some of whom may be geographically dispersed
* organized, coordinated management of changes to software assets for emergency hot-fixes, routine maintenance, upgrades, and new features with potentially overlapping development timeframes (e.g., work on new features occurs simultaneously with work on routine maintenance and/or hot-fixes)
* an auditable change history (e.g., what changed, when, and by whom)
* a reliable master copy of what assets are currently in production
* a reliable master copy of assets from which to build and/or configure the production environment
* reliable copies of previous production versions of assets
* ability to see the specific differences between distinct versions of a given asset

Version control systems are essentially tools and tools by themselves are not sufficient for effective version management; you also need a well-defined version management process.