DOWNLOAD PDF METHODS AND TOOLS FOR SOFTWARE CONFIGURATION MANAGEMENT

Chapter 1 : An Overview of the SWEBOK Guide - SEBoK

Shem J. Ochuodho, Alan W. Brown, A process-oriented version and configuration management model for communications software, Proceedings of the 3rd international workshop on Software configuration management, p, June , , Trondheim, Norway.

Configuration audits These terms and definitions change from standard to standard, but are essentially the same. Configuration identification is the process of identifying the attributes that define every aspect of a configuration item. These attributes are recorded in configuration documentation and baselined. Baselining an attribute forces formal configuration change control processes to be effected in the event that these attributes are changed. Configuration status accounting is the ability to record and report on the configuration baselines associated with each configuration item at any moment of time. Configuration audits are broken into functional and physical configuration audits. They occur either at delivery or at the moment of effecting the change. A functional configuration audit ensures that functional and performance attributes of a configuration item are achieved, while a physical configuration audit ensures that a configuration item is installed in accordance with the requirements of its detailed design documentation. CMDBs are used to track Configuration Items CIs and the dependencies between them, where CIs represent the things in an enterprise that are worth tracking and managing, such as but not limited to computers, software, software licenses, racks, network devices, storage, and even the components within such items. Information assurance[edit] For information assurance , CM can be defined as the management of security features and assurances through control of changes made to hardware, software, firmware, documentation, test, test fixtures, and test documentation throughout the life cycle of an information system. Maintenance systems[edit] Configuration management is used to maintain an understanding of the status of complex assets with a view to maintaining the highest level of serviceability for the lowest cost. Specifically, it aims to ensure that operations are not disrupted due to the asset or parts of the asset overrunning limits of planned lifespan or below quality levels. In the military, this type of activity is often classed as "mission readiness", and seeks to define which assets are available and for which type of mission; a classic example is whether aircraft on board an aircraft carrier are equipped with bombs for ground support or missiles for defense. Operating System configuration management[edit] Configuration management can be used to maintain OS configuration files. Many of these systems utilize Infrastructure as Code to define and maintain configuration. Preventive maintenance Understanding the "as is" state of an asset and its major components is an essential element in preventive maintenance as used in maintenance, repair, and overhaul and enterprise asset management systems. Complex assets such as aircraft, ships, industrial machinery etc. This serviceability is often defined in terms of the amount of usage the component has had since it was new, since fitted, since repaired, the amount of use it has had over its life and several other limiting factors. Understanding how near the end of their life each of these components is has been a major undertaking involving labor-intensive record keeping until recent developments in software. Predictive maintenance Many types of component use electronic sensors to capture data which provides live condition monitoring. This data is analyzed on board or at a remote location by computer to evaluate its current serviceability and increasingly its likely future state using algorithms which predict potential future failures based on previous examples of failure through field experience and modeling. This is the basis for "predictive maintenance". Availability of accurate and timely data is essential in order for CM to provide operational value and a lack of this can often be a limiting factor. Capturing and disseminating the operating data to the various support organizations is becoming an industry in itself. The consumers of this data have grown more numerous and complex with the growth of programs offered by original equipment manufacturers OEMs. These are designed to offer operators guaranteed availability and make the picture more complex with the operator managing the asset but the OEM taking on the liability to ensure its serviceability. A number of standards support or include configuration management, [16] including:

Chapter 2 : Software Configuration Management and ISO

10 Most Popular SCM Tools (Software Configuration Management Tools): In Software Engineering Software Configuration Management is the task of tracking and controlling changes in the software part of the larger disciplinary field of Configuration Management.

Even though this process was not originated in the IT industry, the term is broadly used to refer to server configuration management. Automation plays an essential role in server configuration management. Another common term used to describe the automation features implemented by configuration management tools is Server Orchestration or IT Orchestration, since these tools are typically capable of managing one to hundreds of servers from a central controller machine. There are a number of configuration management tools available in the market. Puppet, Ansible, Chef and Salt are popular choices. Although each tool will have its own characteristics and work in slightly different ways, they are all driven by the same purpose: Benefits of Configuration Management for Servers Although the use of configuration management typically requires more initial planning and effort than manual system administration, all but the simplest of server infrastructures will be improved by the benefits that it provides. To name a few: Quick Provisioning of New Servers Whenever a new server needs to be deployed, a configuration management tool can automate most, if not all, of the provisioning process for you. Automation makes provisioning much quicker and more efficient because it allows tedious tasks to be performed faster and more accurately than any human could. Quick Recovery from Critical Events With quick provisioning comes another benefit: When a server goes offline due to unknown circumstances, it might take several hours to properly audit the system and find out what really happened. In scenarios like this, deploying a replacement server is usually the safest way to get your services back online while a detailed inspection is done on the affected server. With configuration management and automation, this can be done in a quick and reliable way. No More Snowflake Servers At first glance, manual system administration may seem to be an easy way to deploy and quickly fix servers, but it often comes with a price. With time, it may become extremely difficult to know exactly what is installed on a server and which changes were made, when the process is not automated. Manual hotfixes, configuration tweaks, and software updates can turn servers into unique snowflakes, hard to manage and even harder to replicate. By using a configuration management tool, the procedure necessary for bringing up a new server or updating an existing one will be all documented in the provisioning scripts. Version Control for the Server Environment Once you have your server setup translated into a set of provisioning scripts, you will have the ability to apply to your server environment many of the tools and workflows you normally use for software source code. Version control tools, such as Git, can be used to keep track of changes made to the provisioning and to maintain separate branches for legacy versions of the scripts. You can also use version control to implement a code review policy for the provisioning scripts, where any changes should be submitted as a pull request and approved by a project lead before being accepted. This practice will add extra consistency to your infrastructure setup. Replicated Environments Configuration management makes it trivial to replicate environments with the exact same software and configurations. This enables you to effectively build a multistage ecosystem, with production, development, and testing servers. You can even use local virtual machines for development, built with the same provisioning scripts. Overview of Configuration Management Tools Even though each CM tool has its own terms, philosophy and ecosystem, they typically share many characteristics and have similar concepts. Essentially, the controller directs the configuration of the nodes, based on a series of instructions or tasks defined in your provisioning scripts. Below you can find the most common features present in most configuration management tools for servers: Automation Framework Each CM tool provides a specific syntax and a set of features that you can use to write provisioning scripts. Most tools will have features that make their language similar to conventional programming languages, but in a simplified way. Variables, loops, and conditionals are common features provided to facilitate the creation of

more versatile provisioning scripts. Idempotent Behavior Configuration management tools keep track of the state of resources in order to avoid repeating tasks that were executed before. The objective is that after each provisioning run the system reaches or keeps the desired state, even if you run it multiple times. This is what characterizes these tools as having an idempotent behavior. This behavior is not necessarily enforced in all cases, though. System Facts Configuration management tools usually provide detailed information about the system being provisioned. This data is available through global variables, known as facts. They include things like network interfaces, IP addresses, operating system, and distribution. Each tool will provide a different set of facts. They can be used to make provisioning scripts and templates more adaptive for multiple systems. Templating System Most CM tools will provide a built-in templating system that can be used to facilitate setting up configuration files and services. Templates usually support variables, loops, and conditionals that can be used to maximise versatility. For instance, you can use a template to easily set up a new virtual host within Apache, while reusing the same template for multiple server installations. Instead of having only hard-coded, static values, a template should contain placeholders for values that can change from host to host, such as NameServer and DocumentRoot. Extensibility Even though provisioning scripts can be very specialized for the needs and demands of a particular server, there are many cases when you have similar server setups or parts of a setup that could be shared between multiple servers. Most provisioning tools will provide ways in which you can easily reuse and share smaller chunks of your provisioning setup as modules or plugins. Third-party modules and plugins are often easy to find on the Internet, specially for common server setups like installing a PHP web server. CM tools tend to have a strong community built around them and users are encouraged to share their custom extensions. Using extensions provided by other users can save you a lot of time, while also serving as an excellent way of learning how other users solved common problems using your tool of choice. Choosing a Configuration Management Tool There are many CM tools available in the market, each one with a different set of features and different complexity levels. Popular choices include Chef, Ansible, and Puppet. The first challenge is to choose a tool that is a good fit for your needs. There are a few things you should take into consideration before making a choice: Infrastructure Complexity Most configuration management tools require a minimum hierarchy consisting of a controller machine and a node that will be managed by it. Puppet, for example, requires an agent application to be installed on each node, and a master application to be installed on the controller machine. For smaller projects, a simplified infrastructure might seem like a better fit, however it is important to take into consideration aspects like scalability and security, which may not be enforced by the tool. Some tools can have more components and moving parts, which might increase the complexity of your infrastructure, impacting on the learning curve and possibly increasing the overall cost of implementation. Learning Curve As mentioned earlier in this article, CM tools provide a custom syntax, sometimes using a Domain Specific Language DSL , and a set of features that comprise their framework for automation. As with conventional programming languages, some tools will demand a higher learning curve to be mastered. The infrastructure requirements might also influence the complexity of the tool and how quickly you will be able to see a return of investment. Cost Most CM tools offer free or open source versions, with paid subscriptions for advanced features and services. Some tools will have more limitations than others, so depending on your specific needs and how your infrastructure grows, you might end up having to pay for these services. You should also consider training as a potential extra cost, not only in monetary terms, but also regarding the time that will be necessary to get your team up to speed with the tool you end up choosing. Advanced Tooling As mentioned before, most tools offer paid services that can include support, extensions, and advanced tooling. Management panels, for instance, are a common service offered by these tools, and they can greatly facilitate the process of managing and monitoring all your servers from a central point. Community and Support A strong and welcoming community can be extremely resourceful for support and for documentation, since users are typically happy to share their knowledge and their extensions modules, plugins, and provisioning scripts with other users. This can be helpful to speed up your learning curve and avoid extra costs with paid support or training. Overview of Popular Tools The table

below should give you a quick overview of the main differences between three of the most popular configuration management tools available in the market today: Ansible, Puppet, and Chef.

DOWNLOAD PDF METHODS AND TOOLS FOR SOFTWARE CONFIGURATION MANAGEMENT

Chapter 3 : What is Configuration management tools?

A comprehensive guide to the principles and practice of configuration management--the management of software system components during updating or replacement of elements. Features of commercially available tools are described enabling critical evaluation of their effectiveness.

Each KA includes an introduction, a descriptive breakdown of topics and sub-topics, recommended references, references for further reading, and a matrix matching reference material with each topic. An appendix provides a list of standards most relevant to each KA. An overview of the individual KAs presented in the guide is provided in the next two sections. Knowledge Areas Characterizing the Practice of Software Engineering Software Requirements The Software Requirements KA is concerned with the elicitation, negotiation, analysis, specification, and validation of software requirements. It is widely acknowledged within the software industry that software engineering projects are critically vulnerable when these activities are performed poorly. Software requirements express the needs and constraints placed on a software product that contribute to the solution of some real-world problems. Software Design Design is defined as both the process of defining the architecture, components, interfaces, and other characteristics of a system or component and the result of [that] process IEEE The Software Design KA covers the design process and the resulting product. A software design the result must describe the software architecture -- that is, how software is decomposed and organized into components and the interfaces between those components. It must also describes the components at a level of detail that enables their construction. Software Construction Software construction refers to the detailed creation of working software through a combination of detailed design, coding, unit testing, integration testing, debugging, and verification. The Software Construction KA includes topics related to the development of software programs that will satisfy their requirements and design constraints. This KA covers software construction fundamentals; managing software construction; construction technologies; practical considerations; and software construction tools. Software Testing Testing is an activity performed to evaluate product quality and to improve it by identifying defects. Software testing involves dynamic verification of the behavior of a program against expected behavior on a finite set of test cases. These test cases are selected from the usually very large execution domain. The Software Testing KA includes the fundamentals of software testing; testing techniques; human-computer user interface testing and evaluation; test-related measures; and practical considerations. Software Maintenance Software maintenance involves enhancing existing capabilities, adapting software to operate in new and modified operating environments, and correcting defects. These categories are referred to as perfective, adaptive, and corrective software maintenance. The Software Maintenance KA includes fundamentals of software maintenance nature of and need for maintenance, categories of maintenance, maintenance costs ; key issues in software maintenance technical issues, management issues, maintenance cost estimation, measurement of software maintenance ; the maintenance process; software maintenance techniques program comprehension, re-engineering, reverse engineering, refactoring, software retirement ; disaster recovery techniques, and software maintenance tools. It can also be considered as a collection of specific versions of hardware, firmware, or software items combined according to specific build procedures to serve a particular purpose. Software configuration management SCM is thus the discipline of identifying the configuration of a system at distinct points in time for the purposes of systematically controlling changes to the configuration, as well as maintaining the integrity and traceability of the configuration throughout the software life cycle. The Software Configuration Management KA covers management of the SCM process; software configuration identification, control, status accounting, auditing; software release management and delivery; and software configuration management tools. Software Engineering Management Software engineering management involves planning, coordinating, measuring, reporting, and controlling a project or program to ensure that development and maintenance of the software is systematic, disciplined, and quantified. The Software

Engineering Management KA covers initiation and scope definition determining and negotiating requirements, feasibility analysis, and review and revision of requirements ; software project planning process planning, estimation of effort, cost, and schedule, resource allocation, risk analysis, planning for quality ; software project enactment measuring, reporting, and controlling; acquisition and supplier contract management ; product acceptance; review and analysis of project performance; project closure; and software management tools. Software Engineering Process The Software Engineering KA is concerned with definition, implementation, assessment, measurement, management, and improvement of software life cycle processes. Topics covered include process implementation and change process infrastructure, models for process implementation and change, and software process management ; process definition software life cycle models and processes, notations for process definition, process adaptation, and process automation ; process assessment models and methods; measurement process measurement, products measurement, measurement techniques, and quality of measurement results ; and software process tools. Software Engineering Models and Methods The Software Engineering Models and Methods KA addresses methods that encompass multiple life cycle stages; methods specific to particular life cycle stages are covered by other KAs. Topics covered include modeling principles and properties of software engineering models; syntax vs. In addition, the Software Quality KA includes fundamentals of software quality software engineering cultures, software quality characteristics, the value and cost of software quality, and software quality improvement ; software quality management processes software quality assurance, verification and validation, reviews and audits ; and practical considerations defect characterization, software quality measurement, and software quality tools. Software Engineering Professional Practice Software engineering professional practice is concerned with the knowledge, skills, and attitudes that software engineers must possess to practice software engineering in a professional, responsible, and ethical manner. The Software Engineering Professional Practice KA covers professionalism professional conduct, professional societies, software engineering standards, employment contracts, and legal issues ; codes of ethics; group dynamics working in teams, cognitive problem complexity, interacting with stakeholders, dealing with uncertainty and ambiguity, dealing with multicultural environments ; and communication skills. Knowledge Areas Characterizing the Educational Requirements of Software Engineering Software Engineering Economics The Software Engineering Economics KA is concerned with making decisions within the business context to align technical decisions with the business goals of an organization. Topics covered include fundamentals of software engineering economics proposals, cash flow, the time-value of money, planning horizons, inflation, depreciation, replacement and retirement decisions ; not for-profit decision-making cost-benefit analysis, optimization analysis ; estimation, economic risk and uncertainty estimation techniques, decisions under risk and uncertainty ; and multiple attribute decision making value and measurement scales, compensatory and non-compensatory techniques. Computing Foundations The Computing Foundations KA covers fundamental topics that provide the computing background necessary for the practice of software engineering. Topics covered include problem solving techniques, abstraction, algorithms and complexity, programming fundamentals, the basics of parallel and distributed computing, computer organization, operating systems, and network communication. Mathematical Foundations The Mathematical Foundations KA covers fundamental topics that provide the mathematical background necessary for the practice of software engineering. Topics covered include sets, relations, and functions; basic propositional and predicate logic; proof techniques; graphs and trees; discrete probability; grammars and finite state machines; and number theory. Engineering Foundations The Engineering Foundations KA covers fundamental topics that provide the engineering background necessary for the practice of software engineering. Topics covered include empirical methods and experimental techniques; statistical analysis; measurements and metrics; engineering design; simulation and modeling; and root cause analysis. The related disciplines are those that share a boundary, and often a common intersection, with software engineering. SWEBOK V3 does not characterize the knowledge of the related disciplines but, rather, indicates how those disciplines interact with the software engineering discipline. The related disciplines include

Computer Engineering.

Chapter 4 : Configuration Management Tools

Regardless of what tool you use for configuration management, the way to start your automation project is to discover what you have. Automating poor processes or poorly understood infrastructure is a fast and expensive way to multiple your problems.

Purposes[edit] This section is in a list format that may be better presented using prose. You can help by converting this section to prose, if appropriate. Editing help is available. June Configuration identification - Identifying configurations, configuration items and baselines. Configuration control - Implementing a controlled change process. This is usually achieved by setting up a change control board whose primary function is to approve or reject all change requests that are sent against any baseline. Configuration status accounting - Recording and reporting all the necessary information on the status of the development process. Configuration auditing - Ensuring that configurations contain all their intended parts and are sound with respect to their specifying documents, including requirements, architectural specifications and user manuals. Build management - Managing the process and tools used for builds. Environment management - Managing the software and hardware that host the system. Teamwork - Facilitate team interactions related to the process. Defect tracking - Making sure every defect has traceability back to the source. With the introduction of cloud computing the purposes of SCM tools have become merged in some cases. The SCM tools themselves have become virtual appliances that can be instantiated as virtual machines and saved with state and version. The tools can model and manage cloud-based virtual resources, including virtual appliances, storage units, and software bundles. The roles and responsibilities of the actors have become merged as well with developers now being able to dynamically instantiate virtual servers and related resources. Early software had a physical footprint, such as cards , tapes , and other media. The first software configuration management was a manual operation. With the advances in language and complexity, software engineering , involving configuration management and other methods, became a major concern due to issues like schedule, budget, and quality. Practical lessons, over the years, had led to the definition, and establishment, of procedures and tools. Eventually, the tools became systems to manage software changes. With the growing use of computers, systems emerged that handled a broader scope, including requirements management , design alternatives, quality control, and more; later tools followed the guidelines of organizations, such as the Capability Maturity Model of the Software Engineering Institute.

Chapter 5 : An Introduction to Configuration Management | DigitalOcean

A description of the concepts, principles and practice of software configuration management, with particular emphasis on the use of tools and techniques for the development and maintenance of.

These are called codeline policies. The codeline policies also dictate what tests and integrations need to occur before code is released upstream. These requirements should be automated as much as possible, such as automating tests in the Integration build. Codeline patterns circumvent the primary development line. Source Codeline patterns aka branches Core patterns and workspace patterns represent a single path of development, or a single release. It is like a river where boats stop at ports along the banks to fill up and check their inventory. Codeline patterns are like creeks and streams that feed into the river. They are separate from the primary development path and are also called branches. Branches are used when activities could destabilize the main line or considerably slow down development. Examples of when you should use a branch line include: Earmarking the current software version while working on a new version Working on a large, significant change Starting work on a new release before the current release is complete Software Configuration Management Best Practices Along with an organizational pattern chart, effective SCM is based on a set of principles. They map very closely to a number of Agile principles. Development should be done with as few codelines as possible One of the main values of SCM is to root out redundancy and integration issues. Test early and often Testing is fundamental to good software configuration management. Each pattern represents an opportunity for testing before code is released to the next pattern in the hierarchy. Performance testing can be done right on your workstation with pre-prod performance tools like Prefix. Integrate early and often Private workspaces are critical because they allow developers to test integration before actually integrating with the main line. Developers also need to frequently integrate the main line with their workstation to account for any changes. Use tools to automate testing, build, and integration There are dozens of tools that help teams automate critical aspects of the development process. Software Configuration Management Tools Steve Berczuk identifies five categories of tools you will need for effective software configuration management: Source code management repository This tool will serve as your source code repository and help you manage your software versions. Examples of source code management systems include Git , Subversion , and Mercurial. These tools will turn your source code into an artifact, test it for errors, and integrate it with higher-level patterns. Note that not all of these tools have overlapping capabilities and multiple can be used at once. Documentation system A documentation system will contain all codeline policies. A simple Wiki will do, or a well-known spot in your SCM repository. Artifact repository Artifact repositories allow your team to store, organize, and distribute software artifacts and not just source code. Like any good agile practice, you should start by identifying the top priorities for your organization. SCM patterns are the most fundamental piece of software configuration management. If your patterns are poorly laid out, then processes and tools will not help. Patterns should be your top priority. Processes like codeline policies and automated testing are almost as important as the patterns themselves. Make sure everyone on your team knows the processes for working through the SCM patterns. Finally, look at your tool stack for chances to improve efficiency. This should be the last stage of improving your SCM, because the tools will only work as well as you configure them to do so. Here at Stackify, we are passionate about agile, automation, and building better software. Follow our blog to learn more about these practices and more. His company, WeContent , helps technology companies build passionate audiences through irresistible content.

Chapter 6 : Software Configuration Management: Patterns, Best Practices, and Tools for Agile and DevOp

The first software configuration management was a manual operation. With the advances in language and complexity, software engineering, involving configuration management and other methods, became a major concern due to issues like schedule, budget, and quality.

In reality, it seems that few are able to actually articulate what is meant by that term software configuration management. To be fair, those who have a record in software development recognize that there is a need to control what is happening in the development process, and, once controlled, there is a sense that the process can be measured and directed. From that recognized need, then, comes a good working definition of software configuration management: Software Configuration Management is how you control the evolution of a software project. Slightly more formally, software configuration management SCM is a software-engineering discipline comprising the tools and techniques processes or methodology that a company uses to manage change to its software assets. SCM constitutes good engineering practice for all software projects, whether phased development, rapid prototyping, or ongoing maintenance. It enhances the reliability and quality of software by: And not only is it easy to change, but it is unconstrained by the physical laws that serve as the guardrails of what is possible with hardware systems. Software is bounded only by the limits of the human imagination. Uncontrolled and undirected, imagination can quickly give rise to nightmare. Today most software project teams understand the need for SCM to manage change to their software systems. However, even with the best of intentions, software projects continue to fail because of problems that could have been avoided through the use of an SCM tool and appropriate processes. These failures are reflected in poor quality, late delivery, cost overruns, and the incapability to meet customer demands. To understand software configuration management, you might find it easier to look first at configuration management in a hardware-development environment. Hardware systems have physical characteristics that make the problems caused by the lack of sound configuration management easier to see. For example, consider a personal computer. A computer has a processor, a mainboard, some memory, a hard drive, a monitor, and a keyboard. Each of these hardware items has an interface that connects it to other hardware items. If the plug on the mouse was not compatible with the port on the computer, there would be no way to connect the two pieces of hardware into a working system. Throughout the computer, there are many other similar interfaces. The processor and memory plug into the mainboard, the hard drive plugs into the computer, and the printer, monitor, and keyboard all have interfaces. When the hardware is manufactured, the interfaces that are essential for the operation of the final system are easily seen. Therefore, they are well known and are carefully examined whenever changes are made to the hardware design. For a hardware system, configuration management has the following aspects. Each system is numbered or identified and also has a version number. Each version number identifies different designs of the same part. For example, the model year for a car is a version number of that car. When the design of a hardware system is changed, the system gets a new version number. A hardware system can be made up of hundreds, thousands, or tens of thousands of parts. The next thing that must be recorded is which versions of these parts go together. In manufacturing terms, this is often called a bill of materials. The bill of materials lists all the parts and specifies which version of each part should be used to build the system. Parts are assembled into bigger parts, which simplifies the manufacturing process for large systems. In the personal computer example, you can say what version of the mouse, hard drive, processor, and monitor go together to make a complete system. Each of these parts, such as a hard drive, is made of many, many subparts, all of which must go together to have a working unit. Software configuration management deals with all of the same problems as hardware configuration management and more because of the lack of the guardrails that the laws of physics provide. Each software part has an interface, and software "parts" are plugged together to form a software system. These software "parts" are referred to by different names, such as subsystems, modules, or components. They must be identified and must have a version

number. They also must have compatible interfaces, and different versions of parts can have different interfaces. Ultimately, you need a bill of materials to see which versions of which components make up the entire software system. However, software configuration management is much harder to get right because software is much easier to change than hardware. Unlike hardware, software manufacturing is very fast and can be performed hundreds of times a day by individuals on a software team. This is usually referred to as "performing a software build" or "building the software. To begin to create that understanding for you, the rest of this chapter discusses key best practices of SCM in detail and introduces the concepts of the SCM tools and processes that are used to implement those best practices. Many years of practical experience have shown that the following best practices are essential to successful software development: Identify and store artifacts in a secure repository. Control and audit changes to artifacts. Organize versioned artifacts into versioned components. Organize versioned components and subsystems into versioned subsystems. Create baselines at project milestones. Record and track requests for change. Organize and integrate consistent sets of versions using activities. Maintain stable and consistent workspaces. Support concurrent changes to artifacts and components. Integrate early and often. Ensure reproducibility of software builds. The rest of this section explains each of these best practices. These artifacts should include both those used to manage and design a system such as project plans and design models and those that instantiate the system design itself such as source files, libraries, and executables and the mechanisms used to build them. IEEE calls this configuration identification: In terms of an SCM tool, identification means being able to find and identify any project or system artifact quickly and easily. Anyone who has managed a development project with no SCM or poor SCM can attest to the difficulty of finding the "right" version of the "right" file when copies are floating around all over the place. Ultimately, losing or misidentifying artifact versions can lead to the failure of a project, either by hindering delivery of the system because of missing parts or by lowering the quality of the system because of incorrect parts. Organizing artifacts and being able to locate them are not enough. You also need fault-tolerant, scalable, distributable, and replicable repositories for these critical assets. The repository is a potential central point of failure for all your assets; therefore, it must be fault-tolerant and reliable. As your organization grows, you will add data and repositories, so scalability and distributability are required to maintain high system performance. The SCM tool, therefore, must be capable of supporting teams that collaborate across these geographically distributed sites. Finally, the repositories should be backed up with appropriate backup and disaster-recovery procedures. Sadly, many companies overlook this last step, which can lead to severe problems. We refer to this as the audit information. This best practice is related to the IEEE configuration management topics configuration control and configuration status accounting, defined, respectively, as "the evaluation, coordination, approval or disapproval, and implementation of changes to configuration items" and "the recording and reporting of information needed to manage a configuration effectively" [IEEE Glossary,]. Using both control and audit best practices, an organization can determine how strictly to enforce change-control policies. Without control, anyone can change the system. Without audit, you never really know what went into the system. The audit information also enables you to more easily make corrections if errors are introduced. Using control and audit in balance enables you to tune your change control approach to best fit your organization. Ideally, you want to optimize for development productivity while eliminating known security risks. These single objects, made up of sets of files and directories, have a number of different names in the software industry, including packages, modules, and development components. For the purposes of this book, an SCM component is a set of related files and directories that are versioned, shared, built, and baselined as a single unit. To implement a component-based approach to SCM, you organize the files and directories into a single SCM component that physically implements a logical system design component. A component-based approach to SCM offers many benefits, as follows: Use of a higher level of abstraction reduces complexity and makes any problem more manageable. Using components, you can discuss the 6 that make up a system instead of the 5, files subsumed under them. When producing a system, you need to select only 6 baselines, one from each component, instead of having to select the right 5, versions of 5, files.

It is easier to assemble consistent systems from consistent component baselines than individual file versions. Inconsistencies result in unnecessary rebuilding and errors discovered late in the development cycle. It is easier to identify the quality level of a particular component baseline than that of numerous individual files. A component baseline identifies only one version of each file and directory that makes up that component. Because a component baseline contains a consistent set of versions, these can be integration-tested as a unit. It is then possible to mark the level of testing that has been performed on each component baseline. This method improves communication and reduces errors when two or more project teams share components. For example, a project team produces a database component, and another team uses it as part of an end-user application. If the application project team can easily determine the newest database component baseline that has passed integration testing, it will be less likely to use a defective set of files. Instantiating a physical component object in a tool helps institutionalize component sharing and reuse. After component baselines have been created and the quality level has been identified, project teams can look at the various component baselines and choose one that can be referenced or reused from one project to the next. Component sharing and reuse is practically impossible if you cannot determine which versions of which files make up a component. Component sharing between projects is not practical if you cannot determine the level of quality and stability for any given component baseline. Mapping logical design components to physical SCM components helps preserve the integrity of software architectures.

DOWNLOAD PDF METHODS AND TOOLS FOR SOFTWARE CONFIGURATION MANAGEMENT

Chapter 7 : What Is Software Configuration Management? | SCM Best Practices | InformIT

About this Course This four day course provides a basic introduction to the theory, principles, and techniques of Software Configuration Management (SCM) as it applies to the entire software lifecycle.

This article puts in reference the configuration management function and the ISO standard. This standard offers a wide range of advice on how to deal with this important, but often neglected, aspect of software engineering. The software engineering practices associated with software configuration management SCM or CM offer a number of opportunities to address requirements found in the International Standard, ISO From a management perspective, the principles and practices of CM represent an accepted and understood foundation for implementing ISO-compliant processes in software engineering organizations. In addition, the growing number of tools for automating CM practices is chance for improving the efficiency and effectiveness of these processes. This article begins with brief, general definitions of configuration management and of ISO Configuration Management While there is no single definition of CM, there are three widely disseminated views from three different sources: In many cases earlier versions still in use must also be maintained and controlled. It describes a minimum set of activities found in companies and organizations that consistently produce products that satisfy customer requirements. The policies, procedures, standards, records, and associated business activities are the quality system. While ISO is written to describe any company providing any product or service, it tends to employ manufacturing terminology, which must be interpreted for non-manufacturing environments, including service and software providers. The key issues ISO addresses are those: Product exists earlier in software during design and development Software product can be proliferated easily Focusing on these issues mirrors the guidance in Clause 7. The process of development, supply, and maintenance of software is different from that of most other types of industrial products in that there is no distinct manufacturing phase. Software does not "wear out" and, consequently, quality activities during the design phase are of paramount importance to the final quality of the product. ISO is the only source of the requirements against which compliance in software engineering practices is assessed. For software product there should be a clear path between a change request spawned by a fault report or enhancement request and a change ina specific product component to correct the fault or to implement the enhancement. An interested party should be able to pick up the path at any point and follow it forward to the released change and backward to the changerequest or to the fault report. Because a fundamental function of CM is making current configuration items available, the CM practices and tools can be applied to thecontrol of product- and process-related documentation and data. This paragraph of ISO also requires a procedure to determine the disposition of nonconforming product at all stages. For software, the bulk of the activity related to non-conforming product is in the correction of faults identified during allphases of development e.

Chapter 8 : What is Software Configuration Management (SCM)? - Definition from Techopedia

Configuration management is about managing change of the multiple items composing an information system. This article puts in reference the configuration management function and the ISO standard. This standard offers a wide range of advice on how to deal with this important, but often neglected.

Tools to support and automate the CM process support and enforce adherence to policies, procedures, and standards. These procedures can also be automated through integrated workflow and groupware tools that increase the effectiveness and efficiency ofthe information exchange. Standard distribution lists and notification procedures linked to specific activities prevent significant missed communication. If appropriate, the subcontractor can be required to follow or implement specific CM practices. Intermediate or final product and all related documentation e. Applying CM practices to third party development is of particular benefit for coordinating in-house integration and verification activities and for fault resolution. This software is specified by the customer and supplied by the customer or by a third party; the software can be a standard, off-the-shelf shrink-wrapped product or one that is custom developed. Depending on how the included software is packaged and distributed, ISO requirements to verify, store, and maintain this software appropriately may met by considering the included software as a configuration item. In software engineering environments, the project management and CM processes combine to address the majority of these requirements. By automating the product build process, a CM system contributes significantly to the effectiveness and efficiency of the software production process, both for intermediate versions of the product and for a released version. This becomes particularly significant, when multiple versions of the product are being developed or maintained in parallel. The same requirements pertain to problems identified in the development process, starting from the point at which the software product or item comes under CM control. Incidents must be tracked from report, through classification, and, if appropriate, to resulting changes in the product. CM practices, particularly those related to change management, product maintenance, and status accounting, ensure that incidents that result in product change are always handled properly. There is significant opportunity for improving the efficiency of product support and software engineering organizations by minimizing the amount of manual intervention and effort in moving information between the problem tracking and the CM systems. For software product, this requirement is interpreted to include activities like virus checking if an outside replication vendor is used and off-site storage of product masters as a minimum level of disaster recovery. Automated support for the build process, included in most CM tools, reduces opportunities for error and can increase confidence in intermediate test results and in final product integrity. Bydefinition, quality records includes records of: Product identification Non-conformity review and disposition All verification and validation activities, including: CM practices ensure that the product is maintained inan orderly manner and that each approved change can be prioritized, tracked, and managed to completion. Analysis of the data in the CM system related to all aspects of product maintenance can support systematic prioritization and planning forproduct and process enhancement e. What modules cause the most problems? Is the effectiveness of testing continuing toimprove? Even if no modern statistical methods are implemented e. The data in the CM system is a primary input for problem analysis and the identification of root causes in products and processes.

Chapter 9 : Configuration management - Wikipedia

Resources about configuration management tools, which fall into two categories: automation-based tools that can automatically configure software, and CMDB or similar tools that help organizations manage and track software configurations.