**CSC584 Assignment 3**

By,  
Neil Duraiswami

**Contents**

[**Chapter 24: Quality Management** 2](#_Toc159847905)

[**Chapter 25: Configuration Management** 5](#_Toc159847906)

[**Chapter 26: Process Improvement** 8](#_Toc159847907)

[**Reference** 10](#_Toc159847908)

# **Chapter 24: Quality Management**

1.  
  
  
Standards are necessary to preserve organizational knowledge of successful software development techniques because they make them institutional and formal. Learned lessons, best practices and successful approaches are institutionalized through standards. Here are some ways through which this wisdom can be captured:

* **Documented Best Practices:** In documenting the best practices that have been found and demonstrated effective within an organization, standards may come in handy here. For example, if the company has realized that doing thorough reviews improves software quality, there could be a standard specifying their periodicity, who should participate in them, and how they should be done.
* **Insights from previous projects (lesson learned repositories):** Standers can highlight both best practices and pitfalls encountered during past projects. Such lessons can range from technologies used to methods for developing or managing projects that have worked out well or failed.
* **Optimized Procedures:** One possible way of going about this is to develop company-specific processes and methods, which can be standardized. For example, they can describe how the organization develops software in relation to stakeholder expectations, project complexities and team sizes.
* **Guidelines for Tools and Technologies:** Standards may provide recommendations on technologies, tools and frameworks that are appropriate. The set of such knowledge also extends to testing frameworks, version control systems, development environments, among other software engineering tools based on the company history.

2.

The template/prototype for a report that suggests suitable metrics and how they can be collected is  
provided below.

**Report on Software Development Metrics for Database Products**

**Introduction:**

Brief overview of the organization's focus on developing database products for individuals and small businesses. Importance of quantifying software development processes for enhancing product quality and organizational efficiency.

**Metrics:**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Metric Category*** | **Metric Name** | **Description** | **Collection Method** |
| **Product Metrics** | Size Metrics | Measure the size of the database products in terms of lines of code, number of tables, or data volume. | Automated tools (code analysis), manual counting |
|  | Complexity Metrics | Assess the complexity of database structures and queries using metrics such as cyclomatic complexity. | Automated tools (code analysis), manual assessment |
|  | Reliability Metrics | Track the frequency of database failures or errors encountered by users. | Database monitoring, user feedback |
| ***Process Metrics*** | Defect Density | Measure the number of defects per unit size of the database product to gauge the quality of development. | Automated tools (code analysis), manual review |
|  | Time Metrics | Track the time taken to develop and deploy new features or resolve issues. | Version control system logs, manual tracking |
|  | Code Review Metrics | Assess the frequency and effectiveness of code reviews or inspections in identifying defects. | Manual review (code inspections), checklists |
| ***Database Metrics*** | Performance Metrics | Monitor database performance, uptime, and error rates. | Database monitoring tools, system logs |
|  | User Interaction Metrics | Track user interactions, query execution times, and system usage patterns. | Logging and analytics features |

**Conclusion:**

Emphasize the importance of ongoing measurement and analysis of software development metrics for driving continuous improvement. Highlight the potential benefits of adopting a data-driven approach to software development, leading to higher-quality products and improved organizational performance.

3.

Program inspections are a great way of discovering errors in programs for a few reasons:

* **Initial Verification:** Inspections occur at the start of the software development life cycle when defects can be discovered early. Faults identified early during development can be fixed for a small fraction of what it would cost to correct them later or during deployment.
* **Review:** During inspections, team members from different backgrounds go through the program source code comprehensively and line-by-line. This kind of collaboration ensures that faults are detected from different perspectives, thus increasing chances of catching minor mistakes that lone developers might not identify.
* **Supplementary Testing:** Besides testing, inspections play an important role by uncovering bugs that may elude testing. They could be done on incomplete system versions or design models resulting in detecting defects at earlier stages before execution, unlike tests which call for running the program.
* **Standardized Checklists:** Inspections usually use standard checklists of common programming errors that are based on best practices in the industry and recognized error patterns. In order to ensure that many probable mistakes are considered during the inspection process, these checklists give reviewers a systematic framework to work with.

Still, the effectiveness of inspections notwithstanding, there exist some kinds of errors that are unlikely to be found during inspections:

1. **Runtime Errors:** Inspections focus on statically analyzing program source code and design artifacts. Hence, it is not possible that through inspections alone one can come across runtime exceptions or memory leaks which only occur during program execution.
2. **Integration Errors:** Rather than considering how components interact and integrate with each other, typically individual components are evaluated in inspections. Therefore, interface mismatches, interoperability problems or system-level integration issues may occur but are not detected during inspection.
3. **Complex Logic Errors:** Though good at identifying syntax errors and common coding mistakes, inspections may not be effective in catching algorithmic issues or complex logic errors which require deep understanding of the domain and a thorough grasp of program functionality.

# **Chapter 25: Configuration Management**

4.

Some of the problems developers encounter when two programmers are altering different software components at a time and merge their modifications:

* **Overlapping Changes:** Changes made by multiple developers on different components may overlap or conflict. Developers can change the same code portion or make changes that affect related functionalities or dependencies.
* **Interference of Changes:** This is when one developer’s alterations accidentally disrupt another developer’s modifications, causing difficulties in merging. For example, conflicting changes in variable names, function signatures, or data structures can result in compilation errors or runtime issues.
* **Merge Conflicts:** Merging changes coming from several branches or versions of the software might lead to conflicts during the merge process where the version control system fails to automatically reconcile the differences between modified versions. These conflicts must be resolved manually by developers through identification and reconciliation of competing changes which are both time-consuming and error prone.
* **Complexity of Merging:** As the number of modified components and changes increases, so will be the complexity of the merging process. Developers will have to go through multiple versions of the codebase, understand what each modification is about, and ensure that the merged version has not lost the intended functionality for which it was built.
* **Version Control Coordination:** For smooth merging and fast conflict resolution, developers must effectively coordinate their efforts and communicate accordingly. Inadequate synchronization or collaboration among them can result in changes overwriting each other’s work by mistake or inconsistencies in a merged codebase.

5.

The below figure illustrates the system building workflow, encompassing three pivotal elements: the Development System, the Build Server, and the Target Environment. Developers utilize the Development System to make modifications to system components. The Build Server is responsible for generating definitive, executable versions of the system, closely interacting with the version management system. Finally, the Target Environment represents the platform where the system ultimately executes. This diagram underscores the necessity for seamless coordination and compatibility among these components throughout the system building process.

A diagram of a software system

Description automatically generated  
  
The difficulties that may arise when building a system from its components and identifying specific problems that might occur when building a system on a host computer for some target machines are as follows:

* **Building complex systems:** Building involves compiling and linking various components, external libraries, and configuration files. This can be a complicated process with many chances of going wrong due to the necessity of managing dependencies between different things.
* **Multiple System Platforms:** The build process may include multiple system platforms, like the development system, build server, and target environment. These different platforms make building more complex, hence increasing chances of compatibility issues.
* **Version Management Interaction:** In addition, system building tools need to work together with version management systems to download appropriate levels of components. Coordinating this interaction and making sure that correct versions are employed represent a challenge.
* **Build Server Dependencies:** Additionally, the build server relies on external libraries and dependencies which are not necessarily included in the version management system. Keeping track of these and ensuring they are available for use during compilation processes can become a liability.
* **Difference in target environment:** The target environment where the system will work may vary from the development and build environments. This can cause compatibility problems especially for real time or embedded systems with scarce resources.
* **Storage management and compilation:** Building a system involves issues of managing storage space and minimizing compilation time. Tools should conserve storage by retaining only what is necessary for components and minimize recompiling to reduce their build times.
* **Conflict resolution and testing:** Developing a system from multiple components can introduce conflicts and errors which need to be resolved. Thorough examination is needed to identify these problems, thus ensuring that the build system performs properly on the target machine.

6.

Essential features to be included in a change management tool include:

1. **Change Requests Submission:** The tool should make it easier for stakeholders or customers to submit change requests by offering a structured format for outlining the necessary change.
2. **Change Validation:** Before moving to further evaluation, it should have procedures for validating change requests that confirm their validity and adherence to predetermined standards.
3. **Changes Evaluation and Costing:** What will be the impact of the proposed changes on the system? How much does it cost to implement them? Will they affect other system components?
4. **Approval of Change:** It would allow only important changes which are approved and given priority by having some designated authority such as, change control board review and approve such changes.
5. **Change Tracking:** It should keep a history of changes, which records decisions made at each stage of change management processes and tracks the status of requests for variation.
6. **Integration and Collaboration:** This encourages communication, document sharing, and decision-making among stakeholders engaged in a process of change.

# **Chapter 26: Process Improvement**

7.

Distinctions between the process maturity approach and the agile approach:

|  |  |  |
| --- | --- | --- |
| **Aspect** | Agile Approach | Process Maturity Approach |
| Focus | Iterative development, rapid delivery, customer value | Process and project management, quality improvement |
| Methodology | Flexible, adaptive, iterative (e.g., Scrum, Kanban) | Structured, plan-driven, sequential |
| Approach to Change | Embraces change, welcomes customer feedback | Resistant to change, emphasizes stability and predictability |
| Degree of Formality | Minimal formalities, values working software | Emphasizes formalities, documentation, and compliance |
| Risk Management | Addresses risks iteratively, continuous testing | Structured risk management, formal risk assessment |
| Communication | Values face-to-face interaction, collaboration | May rely more on formal documentation and reporting |

8.

The three types of software process metric that may be collected as part of a process improvement process are:

* **Time Metrics:** Time metrics involve measuring the duration or time taken to perform specific activities within the software development process. These metrics provide insights into process efficiency and help identify bottlenecks or areas for improvement. For example, time spent in the software testing phase from start to end of testing activities and measured in days or hours.
* **Resource Utilization Metrics:** Resource utilization metrics focus on quantifying the resources, such as human effort, equipment, or budget, allocated to various process activities. These metrics assess resource allocation efficiency and aid in optimizing resource utilization. For example, total person-days spent on software development, which includes coding, testing and debugging efforts over a specific period.
* **Event Metrics:** Event metrics capture the occurrence or frequency of specific events or activities within the software development process. These metrics provide insights into process dynamics, identify trends, and highlight areas requiring attention. For example, number of defects are discovered during code inspection sessions done throughout quality assurance operations.

9.

The approach to process assessment and improvement contained in frameworks such as the CMMI has several advantages and a couple of drawbacks:

* **Advantages:**

1. **Structured Approach:** he CMMI framework offers a meticulously structured approach to evaluating and enhancing software development processes. This structured methodology provides organizations with a clear roadmap for systematically improving their processes.
2. **Best Practices Integration:** By integrating industry best practices, the CMMI framework enables organizations to leverage proven methods and insights accumulated over time. Adhering to these practices enhances the efficiency and effectiveness of software development processes.

* **Disadvantages:**

1. **Complexity:** Implementing the CMMI framework can be daunting due to its complexity and resource-intensive nature. It demands significant time, effort, and expertise, which may pose challenges for smaller organizations or those with limited resources.
2. **Rigidity:** While the CMMI aims to standardize processes for optimization, it may inadvertently introduce rigidity into organizational workflows. This rigidity can hinder innovation and creativity by imposing strict rules and procedures, potentially limiting adaptability in dynamic environments.

# **Reference**

Sommerville, Ian. Software Engineering. Pearson, 2011.