UNIT 13 SOFTWARE PROCESS IMPROVEMENT

Structure

13.0 INTRODUCTION

The process of Software engineering is undergoing rapid changes. Cloud platforms, Tools, open source frameworks, AI-driven automation are acting as catalysts to the software engineering changes. As there is a high risk in the software projects, it is imperative that we should adopt the best practices during software engineering and continuously improve the software engineering processes.
In this unit we discuss the improvements and optimizations for the software engineering processes.

We have depicted the continuous improvement process in Figure 13.1.

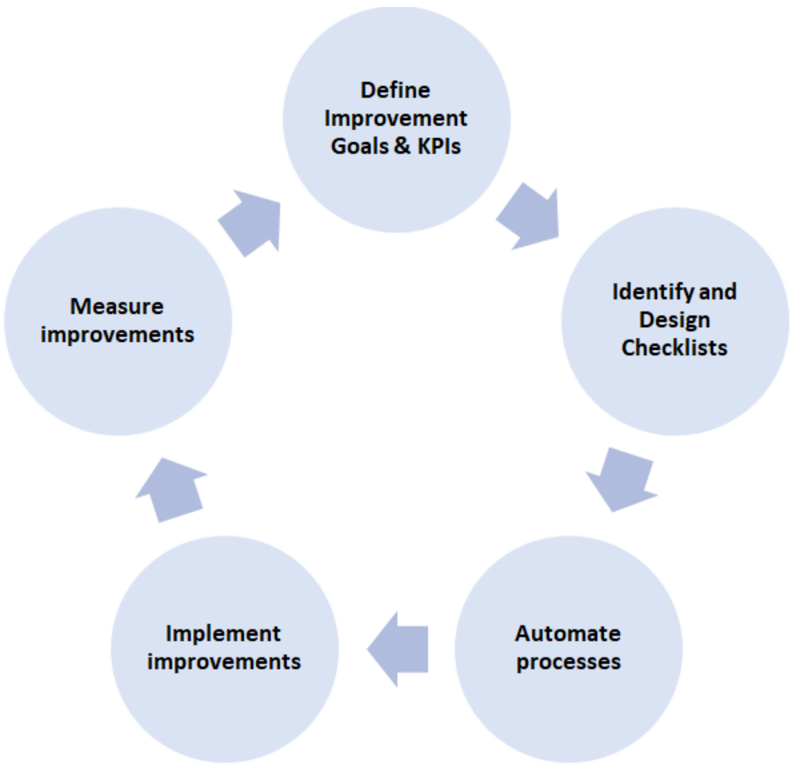

Define Improvement Goals & KPIs
Identify and Design Checklists
Automate processes
Implement improvements
Measure improvements

Figure 13.1 : Continuous Improvement Process

As part of the continuous improvement process, we define the main improvement goals and KPIs. The goals and KPIs should be aligned with the overall business vision. As a next step we identify the design checklists, naming conventions, tools, frameworks, methods and others. We then automate the processes and implement the planned improvements. We then measure the improvements and continuously fine tune it iteratively.

13.1 OBJECTIVES

After going through this unit, you should be able to

- understand key concepts of software process improvement,
- understand details of SEI CMMi process levels,
- understand details of software process optimization through first time right framework,
- software process optimization for validation process
- Understand SEI CMMi methods and best practices for requirements development, technical solution, requirement, requirements management, product integration and project planning.

13.2 SOFTWARE ENGINEERING INSTITUTE CAPABILITY MATURITY MODEL INTEGRATED (SEI – CMMI)

The CMMi is a process improvement model developed by Software Engineering Institute (SEI) at Carnegie Mellon University. The SEI-CMMi integrates the software engineering processes, assesses the maturity levels, assesses the process readiness and provides the coverage for technical, physical and functional aspects of the system.

SEI institute was setup in 1982 at Carnegie Mellon University as an effort to resolve the software process challenges at the US Department of Defense (DoD). The first version of SEI-CMMi was published in 1991.

SEI-CMMi provides a phase-wise steps for each of the levels. Each level defines the capabilities, focus areas, process area and maturity needs for the organization. We can identify a process improvement area and apply the SEI -CMMi process for it to improve the process.

Given below are the key advantages of SEI- CMMi model:

- Improve the software engineering processes using the best practices suggested by SEI-CMMi model.
- Apply the proven processes and methods for software engineering
- Standardize and benchmark software engineering processes.
- Assures the quality of the deliverable.
- Assess the current maturity level of the organization and the process capability readiness.
- Helps us to compare the organizations using the maturity levels.
- Mitigates the risks associated with processes by following the proven methods

We have depicted the various levels of the SEI CMMi in the Figure 13.2.

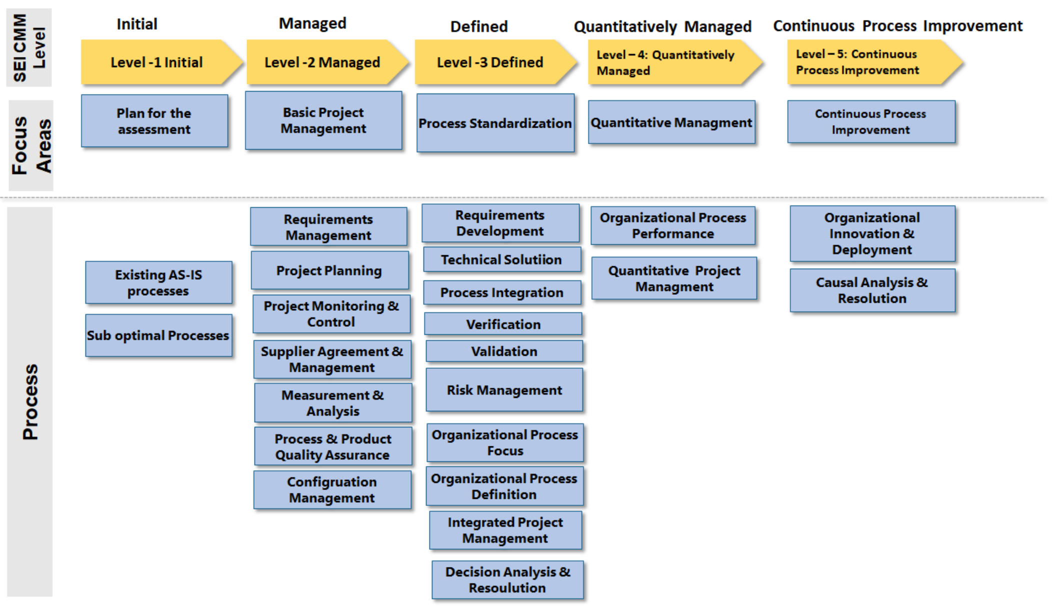

SEI CMM Level
Focus Areas
Process
Initial
Managed
Defined
Quantitatively Managed
Continuous Process Improvement
Level -1 Initial
Level -2 Managed
Level -3 Defined
Level – 4: Quantitatively Managed
Level – 5: Continuous Process Improvement
Plan for the assessment
Basic Project Management
Process Standardization
Quantitative Managment
Continuous Process Improvement
Existing AS-IS processes
Sub optimal Processes
Requirements Management
Project Planning
Project Monitoring & Control
Supplier Agreement & Management
Measurement & Analysis
Process & Product Quality Assurance
Configruation Management
Requirements Development
Technical Solutiion
Process Integration
Verification
Validation
Risk Management
Organizational Process Focus
Organizational Process Definition
Integrated Project Management
Decision Analysis & Resoulution
Organizational Process Performance
Quantitative Project Managment
Organizational Innovation & Deployment
Causal Analysis & Resolution

Figure 13.2 SEI CMM Levels

The figure 13.2 elaborates all the SEI CMM levels and the key focus areas and processes at each of the levels. The initial stage is the as-is stage of the organization. Level 1 is called "*Managed*" wherein the key focus area is basic project management. At this stage, we optimize and improve the processes related to requirements management, project planning, project monitoring, supplier agreement, and measurement, process and product quality assurance and configuration management.

Level 3 is "*Defined*" stage wherein we standardize the processes. We standardize the processes related to requirements development, technical solution, process integration, verification, validation, risk management, organizational process focus, organizational process definition, and integrated project management and decision analysis.

Level 4 is called "*Quantitatively managed*" stage where quantitative management is the main focus area. We aim to optimize the processes such as organizational process performance and quantitative project management.

The final level 5 is called "*Continuous process improvement*" wherein we focus on iteratively improving the organizational processes such as organizational innovation and deployment and causal analysis and resolution.

We look at examples for various SEI CMM levels in later section.

13.3 SOFTWARE PROCESS OPTIMIZATION THROUGH FIRST TIME RIGHT FRAMEWORK.

We have detailed the first time right (FTR) framework that incorporates the best practices at various phases of the software engineering. As part of the FTR framework we have explained the key best practices, methods, tools and improvement techniques during SDLC phases including requirements gathering, architecture and design, testing, DevOps, Infrastructure, project management and governance.

Figure 13.3 depicts the FTR framework.

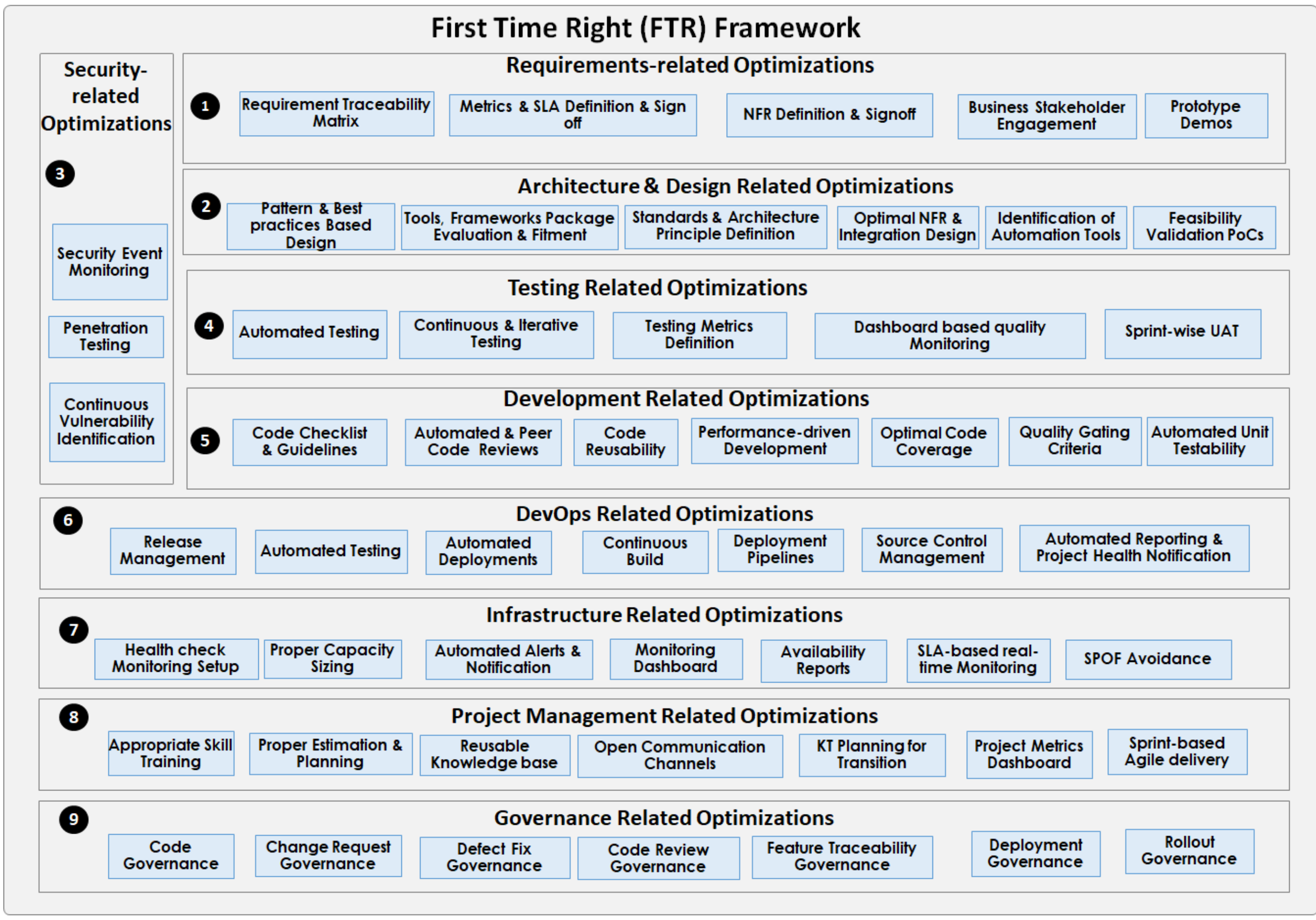

First Time Right (FTR) Framework
Security-related Optimizations
3
Security Event Monitoring
Penetration Testing
Continuous Vulnerability Identification
Requirements-related Optimizations
1
Requirement Traceability Matrix
Metrics & SLA Definition & Sign off
NFR Definition & Signoff
Business Stakeholder Engagement
Prototype Demos
Architecture & Design Related Optimizations
2
Pattern & Best practices Based Design
Tools, Frameworks Package Evaluation & Fitment
Standards & Architecture Principle Definition
Optimal NFR & Integration Design
Identification of Automation Tools
Feasibility Validation PoCs
Testing Related Optimizations
4
Automated Testing
Continuous & Iterative Testing
Testing Metrics Definition
Dashboard based quality Monitoring
Sprint-wise UAT
Development Related Optimizations
5
Code Checklist & Guidelines
Automated & Peer Code Reviews
Code Reusability
Performance-driven Development
Optimal Code Coverage
Quality Gating Criteria
Automated Unit Testability
DevOps Related Optimizations
6
Release Management
Automated Testing
Automated Deployments
Continuous Build
Deployment Pipelines
Source Control Management
Automated Reporting & Project Health Notification
Infrastructure Related Optimizations
7
Health check Monitoring Setup
Proper Capacity Sizing
Automated Alerts & Notification
Monitoring Dashboard
Availability Reports
SLA-based real-time Monitoring
SPOF Avoidance
Project Management Related Optimizations
8
Appropriate Skill Training
Proper Estimation & Planning
Reusable Knowledge base
Open Communication Channels
KT Planning for Transition
Project Metrics Dashboard
Sprint-based Agile delivery
Governance Related Optimizations
9
Code Governance
Change Request Governance
Defect Fix Governance
Code Review Governance
Feature Traceability Governance
Deployment Governance
Rollout Governance

Figure 13.3 : First Time Right (FTR) Framework

Essentially the FTR framework encompasses various SDLC phases and comprehensive program concerns. We have categorized the key components of FTR framework into 9 logical categories:

13.3.1 Requirements Related Optimizations

In many scenarios the requirement gaps snowball into production issues. Hence it is crucial to plug all the gaps in the requirements stage. We have proposed below given optimizations during requirements elaboration stage:

- **Requirement traceability matrix**: The business analysts and the project manager should jointly own the requirement traceability matrix that maps each use case/Jira story to the corresponding test case, code artefact and the release details to ensure that there are no gaps in the delivery.
- **Metrics and SLA definition and sign-off**: We need to quantify the functional and non-functional requirements with accurate metrics and SLAs. For instance, a requirement like "*performance should be good*" is vague and ambiguous; we should formally quantity it as "*the total page load time for the home page should be less than 2 second with the maximum concurrent user load of 100 users in the North American geography*". We need to define the metrics related to application response time, availability, scalability, security and get a formal signoff from business ness.
- **NFR definition and sign-off**: All the non-functional requirements such as *security, performance, scalability, accessibility, multi-lingual capability* and such should be properly defined and signed off by the business.
- **Business stakeholder engagement**: The business stakeholders should be actively engaged throughout the requirement elaboration phase. Without active stakeholder engagement we would miss the requirements and we would face challenges in getting the sign-off.
- **Prototype demos**: We should prepare the mockups/prototypes iteratively and should do frequent demos to showcase the design, user journeys and multi-device experience. This helps us to pro-actively solicit the feedback comments from all the stakeholders and incorporate them.
- **Design Thinking Approach**: Leverage the design thinking approach to empathize with the user and iteratively define the optimal solution exploring the alternatives. Use working ideation and prototype sessions to test the solutions.

13.3.2 Architecture and Design Related Optimizations

During the architecture and design phase, we have has to ensure the following for delivering the first time right design and architecture:

- **Patterns and best practices based design**: The architect has to identify all the application architecture patterns, Industry best practices and design patterns applicable for the solution. As part of this exercise, we has to identify various layers, components and the responsibilities.
- **Tools, frameworks, package evaluation and fitment**: The architect has to evaluate the market leading products, frameworks, open source libraries that are best fit for the requirements. Leveraging proven frameworks, tools and open source libraries greatly contribute to the improved productivity, turnaround time and improved quality. For example as part of the architecture phase, the architect has to critically evaluate technologies such as Angular vs React vs Vue for UI; Spring Boot vs Serverless services, SQL vs NoSQL, mobile web vs hybrid app vs native app, PaaS vs IaaS vs SaaS etc.
- **Standards and architecture principles definition**: The architect has to define the applicable standards for the solution and for the business domain. At this stage the architect has to define the architecture principles such as headless design, stateless integration, token based security etc. Few application domains need to

implement few standards for regulatory compliance (such as HIPAA for health care domain)

- **Optimal NFR and Integration design**: The architect has to design the application to satisfy all the specified NFR (performance, scalability, availability and such) and the integration SLAs.
- **Identification of automation tools:** The architect has to identify all the automation tools that can be used for the project. This includes automation tools for code review, IDE, functional testing and such.
- **Feasibility validation PoCs:** For complex requirements, we need to carry out the feasibility assessment through proof-of-concepts (PoCs). This helps us to finalize the tool, technology, integration method, performance and assess the scalability and performance of the method.

As part of process improvement and optimization, we need to track the trends along various dimensions such as architecture shift, technology shift, integration shift and others and marked the recommended tenets for the solution. We have depicted the sample architecture trends in Figure 13.4.

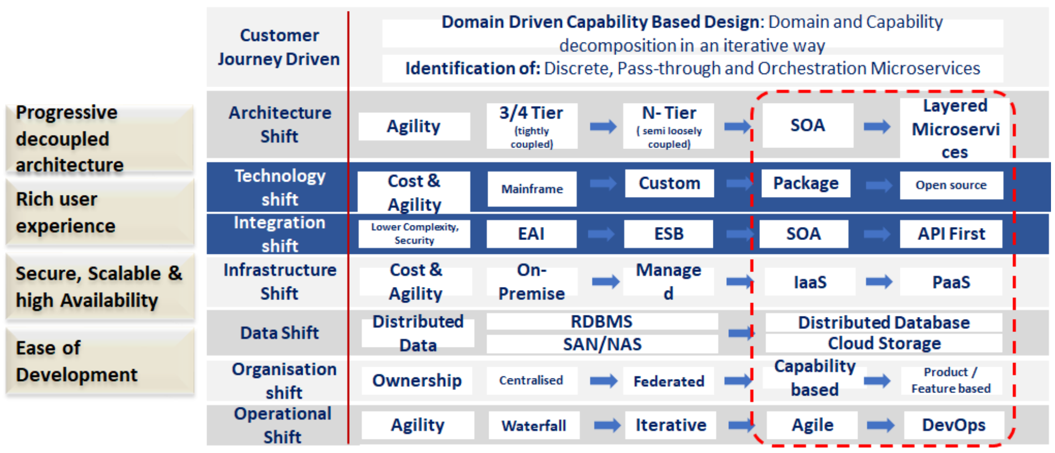


Figure 13.3 : Architecture Trends

13.3.3 Security Related Optimizations

On the security front, we need to continuously monitor and audit for critical security events (such as failed login attempts, password change events, impersonation events, role change events etc.) and setup a notification infrastructure for the same.
We should also carry out the penetration testing and vulnerability testing iteratively to discover the security vulnerabilities early.

13.3.4 Testing Related Optimizations

A comprehensive validation is one of the key success factors for delivering the first time right deliverable. The validation team needs to follow the below-given optimizations:

- **Automated testing**: The testing team has to use the automated tools such as Apache JMeter or Selenium to automate the regression scenarios and other functional test scenarios to improve the overall productivity.
- **Continuous iterative testing**: The testing has to be a continuous and iterative process across all sprints. The helps us to discover the defects in the early stage.
- **Testing metrics definition**: The testing team has to define the quality metrics such as defect rate, defect slippage rate, defect density and track the metrics with each sprint.

- **Dashboard based quality monitoring**: The test lead should pro-actively monitor all the testing metrics in a dashboard and notify the project manager in case of any critical violations.
- **Sprint-wise early UAT**: We have noticed that involving the business stakeholders with each sprint delivery in the UAT phase helps us to uncover any business related gaps early in the game. Additionally the team can incorporate the feedback in the next sprint.

13.3.5 Development Related Optimizations

The big chunk of responsibility for first time right deliverable lies with the development team. I will design and implement the below given optimizations for the development team:

- **Code checklist and guidelines**: Developers should use the code guidelines, naming conventions and coding best practices in a checklist format. Before the start of the project, the architect along with project manager has to create the checklist, naming conventions, best practices and such. Some of the most common code checklist are as follows:
  - Language specific coding best practices
  - Performance checklist
  - Security coding checklist
  - Code naming conventions
  - Design checklist
- **Automated and Peer Review process**: Developers should use automated static code analyzers such as PMD/SonarQube to ensure the code quality. The developers should also get their code reviewed by their peers and leads on a frequent basis.
- **Code reusability:** Developers should actively explore the code reusability opportunities in the following order:
  - Look for reusable libraries, code modules, components offered by the underlying platform/product/framework/accelerators.
  - Explore the availability of reusable libraries, code modules, components at the organization level.
  - Look for approved open source libraries that satisfies the functionality.
  - If nothing is available, develop the code in modular way so that it can be reused.
- **Performance driven development**: Performance should not be an afterthought, but it must be implemented from the very early stages. Developers need to pro-actively carry out the performance testing of the integrated code to ensure that their code is free of memory leaks, integration bottlenecks and others. We have detailed the performance based design and web optimization framework in the NFR section.
- **Optimal code coverage**: Developers should ensure that their unit test cases provide more than 90% code coverage. This ensures high code quality with minimal defects. Developers should also try to use an automated tool for generating unit test cases.
- **Quality gating criteria:** We need to define multi-level code quality gating criteria as follows (this could also go in as code check-in checklist):
  - **Developer-level code quality**: The developer has to use the defined coding checklist and naming conventions to adhere to those guidelines. The developer can also use the IDE for the same.
  - **Automated Local code quality analyzer**: The developer has to use the static code analyzers such as PMD, SonarQube to analyze all coding issues and fix the major and critical issues.
  - **Manual code review**: The developers can request for peer review and lead review of the code. Once all of the above is completed, the developers can check in the code to the source control system.

   - **Integrated code review**: We could setup a Jenkins job with SonarQube to continuously review the integrated code and generate the report. The Jenkins job can notify the developers and project managers for major and critical violations.
- **Automated unit testability**: Developers should use automated unit test generators (such as EvoSuite, Veracode) to improve the quality and productivity of the developer.

13.3.6 DevOps Related Optimizations

DevOps defines a set of tools and processes to ensure the proper delivery and release of the application. Given below are the key optimizations from DevOps standpoint:

- **Release management**: DevOps setup provides an opportunity for setting up automated release management system
- **Automated testing**: We can configure unit test jobs and functional testing jobs (such as Selenium) to test the code from source control system
- **Automated deployment**: We can use Jenkins pipelines and deployment jobs to automate the deployment.
- **Continuous build**: We can enable the continuous build to catch any errors with integrated code.
- **Deployment pipelines**: The pipelines enable and automate the code deployment.
- **Source control management**: As part of DevOps process we also define the source control management processes related to pull request, approval, code check-in, code merge and such.
- **Automated reporting and project health notification:** We could configure the project health notification, reporting features to trigger the notification in case of any SLA violation.

We have depicted the process improvement and optimizations for Azure DevOps in figure 13.5.

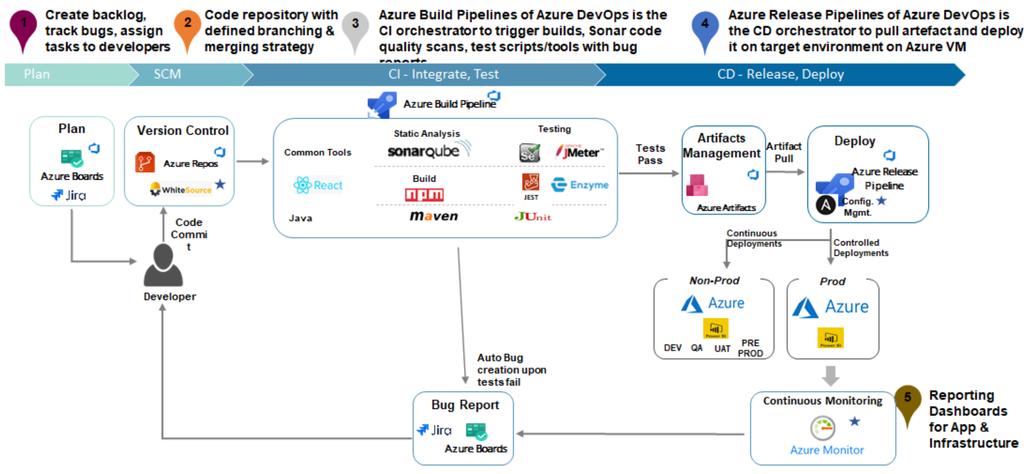


Figure 13.5 : DevOps Pipeline in Azure DevOps

13.3.7 Infrastructure Related Optimizations

A right sized infrastructure is critical for the optimal application delivery. In order to deliver the first time right solution, given below are the infrastructure related optimizations:

- **Health check monitoring setup**: We need to setup the pro-active healthcheck/heartbeat monitoring infrastructure to continuously monitor the availability of the servers.
- **Proper capacity sizing**: All the server and network capacity should be sized based on the user load and related non-functional requirements (such as scalability, availability, performance). Based on the availability requirements, we should also setup the disaster recovery (DR) and the corresponding sync jobs.
- **Automated alerts and notification**: We should be able to configure notification triggers and the monitoring setup should notify the administrators in case of SLA violation.
- **Monitoring dashboard**: The monitoring dashboard should provide a snapshot of all key parameters such as availability percentage, performance, CPU/memory/network utilization, request rate and such.
- **Availability reports**: The monitoring setup should be able to generate on-demand monitoring reports for various infra-related parameters.
- **SLA based real time monitoring**: We should also setup the real-time application monitoring infrastructure across various geographies to understand the performance and availability issues.
- **Single Point of Failure (SPOF) avoidance**: Ensure that none of the systems in the end to end request processing pipeline form the bottleneck leading to single point of failure (SPOF). We can ensure high availability through cluster setup, redundancy setup, DR setup, regular backup and by other means.

Additionally we should setup the automatic elastic scalability, optimal load balancing, CDN based caching based on the application requirements.

13.3.8 Project Management Related Optimizations

Project managers shoulder greater responsibility in defining and enforcing the processes to implement first time right deliverable. Given below are the key optimizations from project management standpoint:

- **Appropriate skill training**: The project manager has to train the project team members to equip them with suitable technology skills. It is recommended to have right mix of skillset (lead to developer ratio) in the team to ensure timely delivery and high quality deliverable.
- **Proper estimation and staffing:** The project manager has to do the proper effort estimates and staff resources accordingly. The project manager has to define proper risk management plan for all known contingencies.
- **Reusable knowledge base:** The project manager has to maintain the best practices, Standard Operating Procedures (SOP), key design decisions (KDD), How-to documents, troubleshooting documents, learnings, process documents, KT documents and others in a centralized knowledge base. This helps the team members to build upon the collective knowledge and leverage it for faster and better deliverable.
- **Open communication channels**: In order to avoid information silos, the project manager has to establish open communication channels across all tracks. Various tools such as Slack, MS Teams can be leveraged for this.
- **KT Planning for transition:** If the team takes over any new engagement from an incumbent, we need to plan for proper KT from the incumbent.
- **Project metrics dashboard**: The project manager has to track the overall project health in the metrics dashboard. Typically a project dashboard consists of burn rate, code quality score, open defect count, average defect fix time, sprint run rate and such. This helps the project manager to get a holistic view of all open issues and prioritize them accordingly.
- **Sprint based Agile Delivery:** Modern digital projects are complex and are end-user focused. Hence the project manager has to plan for sprint based agile delivery so that we can mitigate the risk and get the early user feedback.

- **Continuous Improvement model:** The project manager has to constantly seek to improve the overall productivity and quality using the learnings, metrics from earlier sprints. The project manager has to continuously look out for automation opportunities.

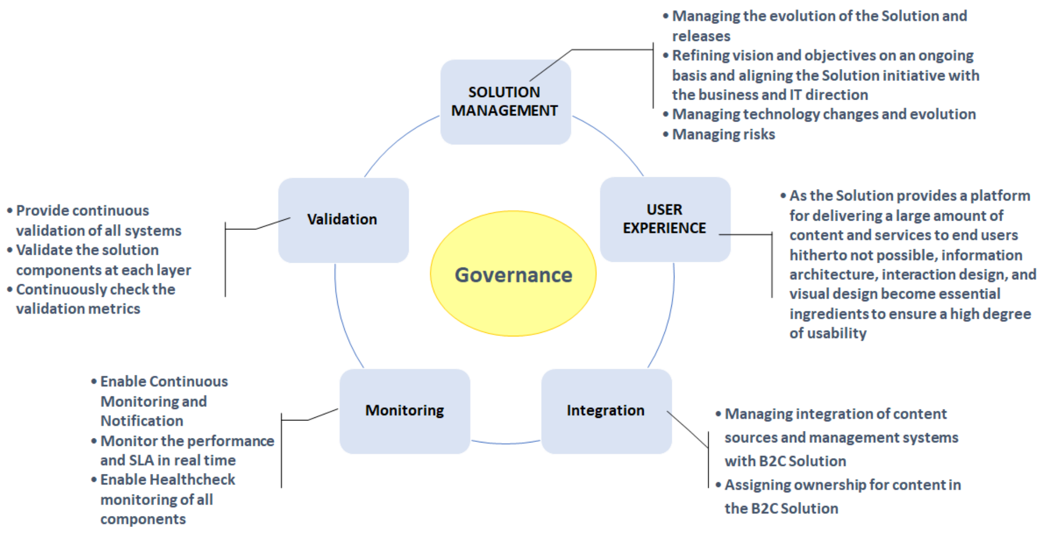
The project manager also has to pro-actively understand the regulatory and compliance requirements (such as accessibility standards), browser and device compatibility (browsers and mobile devices that need to be supported), multi-lingual requirements (list of left-to-right and right-to-left languages) that need to be supported and accordingly plan for the same.

13.3.9 Governance Related Optimizations

Project governance broadly details the well-defined processes to streamline the complex project activities. Given below are the governance-related optimizations:

- **Code Governance** defines the code management related processes such as code merging, code check in, code versioning and such.
- **Change request governance** broadly defines the scope creep management process. It defines the criteria for change request management, its prioritization, and impact management and implementation plan.
- **Defect fix governance** defines the processes related to defect prioritization, SLAs and such.
- **Review governance** defines the processes related to code review and approval.
- **Feature traceability governance** defines the process to trace the requirement to the final release.
- **Deployment governance** provides the deployment plan for code artefacts.
- **Rollout governance** defines the rollout plan across various geographies, features, languages, rollback plan and such.

We have depicted the key elements of solution governance in Figure 13.6.



• Managing the evolution of the Solution and releases
• Refining vision and objectives on an ongoing basis and aligning the Solution initiative with the business and IT direction
• Managing technology changes and evolution
• Managing risks
SOLUTION MANAGEMENT
• Provide continuous validation of all systems
• Validate the solution components at each layer
• Continuously check the validation metrics
Validation
Governance
USER EXPERIENCE
• As the Solution provides a platform for delivering a large amount of content and services to end users hitherto not possible, information architecture, interaction design, and visual design become essential ingredients to ensure a high degree of usability
• Enable Continuous Monitoring and Notification
• Monitor the performance and SLA in real time
• Enable Healthcheck monitoring of all components
Monitoring
Integration
• Managing integration of content sources and management systems with B2C Solution
• Assigning ownership for content in the B2C Solution

Figure 13.6 : Solution Governance

13.3.10 Tools

We have given various tools that can be used for implementing the First time right framework in Table 13.1.

Table 13.1: FTR Tools

Tool Category	Open Source/ Commercial Tool(s)
Web Page Analysis tools (HTML analysis, performance benchmarking, improvement guidelines)	Yahoo YSlow , Google PageSpeed , HTTPWatch , Dynatrace AJAX Edition
Page development tools (analysis of page load times, asset size, asset load times and such.)	Firebug, Google Chrome Developer toolbar, Fiddler , HTTP Archive WEB PAGEiddle, CSSLint , JSLint , W3 CSS Validator , W3 HTML validator
Asset merging and minification tools (JS/CSS minification)	Yahoo UI (YUI) minifier, JSMini , JSCompress
Page Performance Testing tools (load simulation)	JMeter, LoadUI, Grinder, Selenium
Image compression tools	PNGCrush, Smush It , Img min , JPEG Mini
Web Server Plugins (for automatic compression, minification, merging, placement, caching etc.)	Mod_pageSpeed , mod_cache, mod_SPDY mod_expiry , mod_gzip
Website Performance Testing	GTMetrix , Pingdom
Synthetic monitoring (transactions simulation & performance statistics)	Web Page test , DynaTrace Synthetic monitoring
CDN	Akamai, CloudFlare, KeyCDN,
Web Analytics (track user behavior, performance reporting)	Google Web Analytics, Omniture, Piwik
CSS Optimization tools	CSS Sprites , SpriteMe , SpritePad
Bottleneck Analysis (dependency & bottleneck analysis)	WebProphet , WProf
Real User Monitoring (RUM) (monitoring & bottleneck analysis)	New Relic , Dynatrace , Gomez
Network analysis (network traffic, HTTP headers, request/responses, protocol analysis)	Wireshark , Charles Proxy
Application Performance Monitoring (APM) (Layer wise monitoring of application code)	New Relic , Dyna Trace Monitoring, Nagios

☞ **Check Your Progress 1**

1. The focus of SEI CMMi level 2 is ______
2. Requirements development is included in level ______
3. The focus areas of level 3 of SEI CMMi is _____
4. ______ maps each use case/Jira story to the corresponding test case
5. ______approach to empathize with the user and iteratively define the optimal solution exploring the alternatives
6. We assess the feasibility of the tool/method through _______

13.4 SOFTWARE PROCESS OPTIMIZATION FOR VALIDATION PHASE

In this section we discuss the software process optimization for the performance validation phase.

A robust web performance test methodology defines the sequence and activities for various performance testing phases. We have detailed the performance test methodology in Figure 13.7.

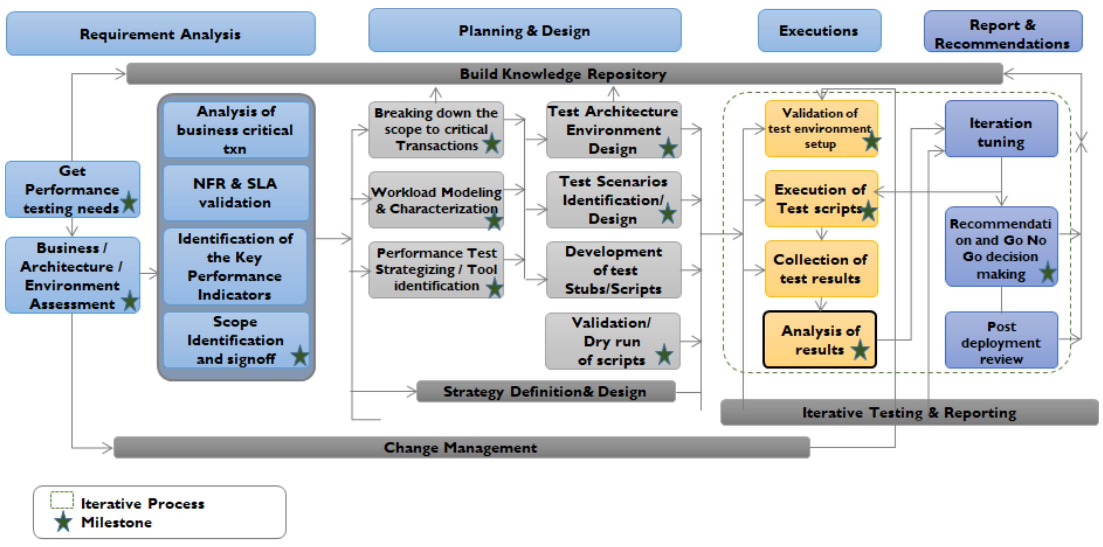

Requirement Analysis
Planning & Design
Executions
Report & Recommendations
Build Knowledge Repository
Get Performance testing needs
Business / Architecture / Environment Assessment
Analysis of business critical txn
NFR & SLA validation
Identification of the Key Performance Indicators
Scope Identification and signoff
Breaking down the scope to critical Transactions
Workload Modeling & Characterization
Performance Test Strategizing / Tool identification
Test Architecture Environment Design
Test Scenarios Identification/ Design
Development of test Stubs/Scripts
Validation/ Dry run of scripts
Validation of test environment setup
Execution of Test scripts
Collection of test results
Analysis of results
Iteration tuning
Recommendation and Go No Go decision making
Post deployment review
Strategy Definition& Design
Iterative Testing & Reporting
Change Management
Iterative Process
Milestone

Figure 13.7 : Web Performance Test Methodology

During ***requirement analysis phase***, the performance testing team will get the complete performance test needs and document the requirements. During this phase, the testing team assesses the architecture and design documents. The team would also analyze the business critical and performance critical transactions. The performance validation team gets sign-off on performance-related NFRs (such as peak user load, peak content volume and such) and performance-related SLAs (timing metrics for the defined NFRs) during this phase. The team identifies the key performance indicators (KPI) such as average page response time across all geographies, availability metrics, performance metrics, performance SLA adherence rate and such. Some of the common performance related questions are given below:

- Which pages receive highest traffic?
- How many pages have acceptable performance above configured performance thresholds?
- Which pages have bad performance?
- What is the caching strategy used?

The ***planning and design phase,*** the performance testing team will break down the scope into business transactions that need separate performance scripts. We also characterize the workload model based on the load numbers we have gathered. The performance testing team creates the overall performance test plan. The performance environment will be setup after all the performance-testing tools are finalized. Performance testing team develops the performance test scripts.
During the ***execution phase,*** the performance test team executes all the scripts and record various timing metrics that are defined in the requirements analysis phase. Various performance tests such as load testing, stress testing, endurance testing will be carried out in this phase. Automation scripts will be used to automatically execute the performance tests after each major release. The results will be analyzed and document.
Reports will be published in the **reporting and recommendation** phase. Performance testing teams recommends a "go" or a "no-go" decision based on the overall performance and adherence of the performance results to the defined SLAs. Performance testing and reporting will be done iteratively for various sprints of the release.

13.4.1 Performance Testing Tools

Given below are the key testing tools that can be used for web performance testing in Table 13.2

Table 13.2 : Performance Testing Tools

Category	Performance testing tools
Web Page Testing	Google Chrome Lighthouse Google PageSpeed Insights
Application Profiling	Open Source: JProbe, Eclipse Profiler Commercial: Jprofiler, OptimizeIt,
Load testing	Open Source: Apache JMeter, Grinder, Apache Bench, HTTPPerf Commercial: HP LoadRunner, NeoLoad, BlazeMeter
Service testing	LoadUI, SOAPUI
Client side performance testing	https://www.webpagetest.org/
Real time web performance monitoring	Open Source: Nagios, Hyperic – HQ, Perfmon, NMon Commercial - HP SiteScope, Wily Introscope
Application Performance Monitoring (APM)	Dynatrace, AppDynamics, New Relic
Mobile App testing	Appium, UI Automator
Analysis tools	GC Analyzer, JVM Analyzer

13.4.2 Performance Monitoring and Notification

A robust monitoring and alerting setup should be able to capture the system metrics (CPU, memory), log metrics (system logs, application logs), errors, performance and such. The monitoring setup should also be flexible to monitor various systems such as Linux OS, database server, stand-alone server, performance and such. The alert and notification setup should be flexible enough to send notifications to various channels such as email, pager, incident management system and such. We should be able to configure the performance thresholds and resource utilization thresholds that can trigger the notification.
A comprehensive monitoring tool should support these features:

- Core Monitoring capabilities
  - Resource (CPU, Memory) monitoring
  - Network (Router, Switch, and Firewall etc.) Monitoring
  - Server Monitoring
  - Windows Event Log Monitoring
  - Applications Monitoring
  - Virtual instances
- Application Server monitoring capabilities
  - Database Monitoring
  - Web Page, Web Server/ Web Services Monitoring
    - Middle Ware Monitoring
    - Custom Application Monitoring
- Reporting Dashboard
  - Business service management views

  - Comprehensive dashboard
  - Real Time trends and availability of devices
  - Events and Correlated Alarms
- Reporting
  - Standard daily, weekly, monthly, quarterly, and yearly reports

13.4.3 Performance Monitoring Tools

We could leverage many open source and commercial tools for performance monitoring. Table 13.3 lists the popular performance monitoring tools.

Table 13.3 : Performance Testing Tools

Monitoring needs	Monitoring Tool
System Monitoring (CPU, Memory, and disk)	• Windows Performance monitor (perfmon) • NMon for AIX servers • System Activity Report (SAR) for UNIX systems • Node exporter for container pods
File monitoring	Filebeat
Network monitoring, System monitoring and infrastructure monitoring	• Nagios • ELK (Elastic search, Logstash and Kibana) • Grinder • AppDynamics (Commercial) • New Relic (Commercial) • Zabbix (https://www.zabbix.com/)
Statistics dashboard and visualizations Alert and monitoring dashboard	• Kibana • Grafana
Real time event monitoring Visualizations And data queries	Prometheus & Grafana
Search engine	Elasticsearch
Synthetic monitoring tool	• DynaTrace (Commercial) • Selenium • Lighthouse • Webpagetest.org
Database monitoring	• Automatic Workload Repository (AWR) • Fluentd
Log monitoring	• Splunk • Fluentd
Message streaming	Apache Kafka
Notification	Alert Manager
Container Monitoring	• Node Exporter • Docker Stats • cAdvisor • Prometheus
Web Page Monitoring (page size, page response time, number of requests, asset load time etc.)	• Webpagetest.org • Site Speed (https://www.sitespeed.io/) • Google Page Speed Insights (https://developers.google.com/speed/pagespeed/insights/)

	• Pingdom (commercial) • Silk performance Manager (commercial) • Uptrends (commercial) • https://web.dev/measure/
Development tools/Page Auditing	• Google Chrome Developer Tools • Test My site (https://www.thinkwithgoogle.com/feature/testmysite/) • Google Chrome lighthouse • HTTP Watch • https://speedrank.app/en • Fiddler • Firebug • Web Tracing Framework http://google.github.io/tracing-framework/ • Timeline tool https://developers.google.com/web/tools/chrome-devtools/evaluate-performance/timeline-tool#profile-painting
Multi-geo web performance testing	• https://performance.sucuri.net/
Cloud Monitoring	• AWS CloudWatch
Website speed test	• https://tools.keycdn.com/speed
Load testing	• BlazeMeter • Apache JMeter
Web site latency test	• Ping Test (https://tools.keycdn.com/ping)
Real user monitoring (RUM)	• New Relic • SpeedCurve (https://speedcurve.com/) • DynaTrace • Akamai mPulse (Commercial) • AppDynamics (Commercial)
Analyze network timings	• Resource Timing API (https://developer.mozilla.org/en-US/docs/Web/API/Resource_Timing_API/Using_the_Resource_Timing_API) • Network Information https://developer.mozilla.org/en-US/docs/Web/API/NetworkInformation

13.5 SEI CMMi PROCESS EXAMPLES

In this section we shall look at improving the processes at various SEI CMMi levels. We have taken the focus areas and the best practices for each of the processes.

13.5.1 Requirements Development

This is part of level 3 "Defined" stage wherein the expectation is to thoroughly understand the requirements and develop the solution for that.

Key Focus Areas
Given below are the key focus areas:

- Understand the requirements from various perspectives (business, operations, performance) and document the requirements.

- Thoroughly understand the integration requirements and the related nonfunctional SLAs related to security, performance, scalability and availability.
- Understand the exception scenarios for each of the requirements.

Methods for process standardization
Given below are the main methods and best practices for requirements development process standardization:

- Design a comprehensive requirements gathering template to cover the requirements from various dimensions.
- Elaborate the user stories and document the exception flows
- Define bi-directional requirement traceability matrix to trace the requirements to its corresponding development artefacts.
- Validate the requirements with the stakeholders and obtain a formal sign-off from the stakeholders.
- Build quick prototypes to validate the user experience related requirements.

13.5.2 Technical Solution

This is part of level 3 “Defined” stage wherein the expectation is to build the solution based on best practices and best of breed technologies.

Key Focus Areas
Given below are the key focus areas:

- Evaluation of alternate solution options and tools
- Evaluate build, reuse or buy decisions
- Design and develop a scalable and high performing solution

Methods for process standardization
Given below are the main methods and best practices for Technical solution process standardization:

- Evaluate the alternate solution options against the define criteria (such as cost, effort, performance)
- Elaborate the detailed design and leverage the best practices, recommended naming conventions and use patterns.
- Use the automated static code analyzers to ensure code quality.
- Develop unit test cases to ensure adequate code coverage
- Document the code and provide a user guide.

13.5.3 Requirements Management

This is part of level 2 “Managed” stage wherein the expectation is to manage the requirements and resolve any inconsistencies across various requirements.

Key Focus Areas
Given below are the key focus areas:

- Change management for the requirements
- Identify the inconsistency among the requirements
- Manage the requirement change and its impact

Methods for basic project management
Given below are the main methods and best practices for requirements management:

- Leverage various methods such as prototyping, user journey mapping, interviews, focus group discussions, story boarding for gathering requirements.
- Define the change management process.
- Develop bi-directional requirement traceability matrix to ensure the proper implementation of requirements.

13.5.4 Product Integration

This is part of level 3 “Defined” stage wherein the expectation is to provide an optimal integration across various interfaces.

Key Focus Areas
Given below are the key focus areas:
- Identify all the needed interfaces
- Optimal assembly of the product components.
- Define the integration SLAs

Methods for process standardization
Given below are the main methods and best practices for product integration process standardization:

- Design and implement integration specification document to specify all the APIs, contracts and exception scenarios
- Test the interfaces at various loads to understand the performance and scalability
- Handle the exception scenarios such as timeouts, system down issues.

13.5.5 Project Planning

This is part of level 2 “Managed” stage wherein the expectation is to define the accurate project plan for managing the project

Key Focus Areas
Given below are the key focus areas:
- Accurate scope, effort and time estimation
- Accurate risk management.
- Appropriate stakeholder management.

Methods for basic project management
Given below are the main methods and best practices for project planning:

- Use accurate estimates for proper project planning.
- Create a comprehensive project plan to cover the risks, milestones, project metrics, dependencies, resources, acceptance criteria, training plan, escalation matrix, knowledge management plan and others.
- Identify main business stakeholders for each of the departments.
- Develop comprehensive stakeholder management plan covering the following as active engagement of all concerned stakeholders is a key to sell the solution and for the long term success of the program:
    - Map their roles, responsibilities and ownership in the matrix

- o Communicate the key decisions and articulate impact for each of the stakeholders and highlight the impact for their respective departments.
- o Address their concerns in separate spin-off meetings.
- o Take the key stakeholders into confidence in the overall decision making.
- o Organize the demos, PoCs, prototypes, journey map walkthrough, product evaluation sessions based on the roles and responsibilities for each of the stakeholders.

Check Your Progress 2

1. _________ captures the system metrics (CPU, memory), log metrics (system logs, application logs), errors and performance.
2. Splunk is one of the _______ tools
3. ____________ traces the requirements to its corresponding development artifacts
4. _________ specifies all the APIs, contracts and exception scenarios
5. One of the tools to check the web site latency is ________

13.6 SUMMARY

In this unit, we started discussing main features of the SEI CMMi. We looked at various 5 levels of the SEI CMMi methodology. The level 1 is called initial. The level 2 is termed as managed where the main focus area is basic project management. Process standardization is the focus area of level 3 defined. The focus area of level 4 is quantitatively managed and the continuous process improvement is the focus area of level 5. We then discuss the first time right (FTR) framework that discusses the optimizations in the areas of requirements, security, testing, development, DevOps, infrastructure, project management and governance. We also looked at the tools used in the FTR framework. We then had a deep dive about the software process optimization for performance validation framework. We then looked at five examples of SEI CMMi process optimization examples related to requirements development, technical solution, production integration, and project planning and requirements management.

13.7 SOLUTIONS/ANSWERS

Check Your Progress 1

1. Basic project management.
2. Level 3
3. continuous process improvement
4. Requirement traceability matrix
5. Design thinking
6. Proof of concept (PoC)

Check Your Progress 2

1. Monitoring and Notification.
2. log monitoring

3. bi-directional requirement traceability matrix
4. integration specification document
5. Ping Test

13.8 FURTHER READINGS

References

Shivakumar, S. K., Shivakumar (2018). *Complete Guide to Digital Project Management*. Apress.

Team, C. P. (2002). CMMI for Systems Engineering/Software Engineering/Integrated Product and Process Development/Supplier Sourcing, Version 1.1, Continuous Representation. *CMU/SEI*.

Conradi, H., & Fuggetta, A. (2002). Improving software process improvement. *IEEE software*, *19*(4), 92-99.

https://en.wikipedia.org/wiki/Capability_Maturity_Model

https://en.wikipedia.org/wiki/Capability_Maturity_Model_Integration