Streamlining the Software Development Lifecycle with Razor Agile

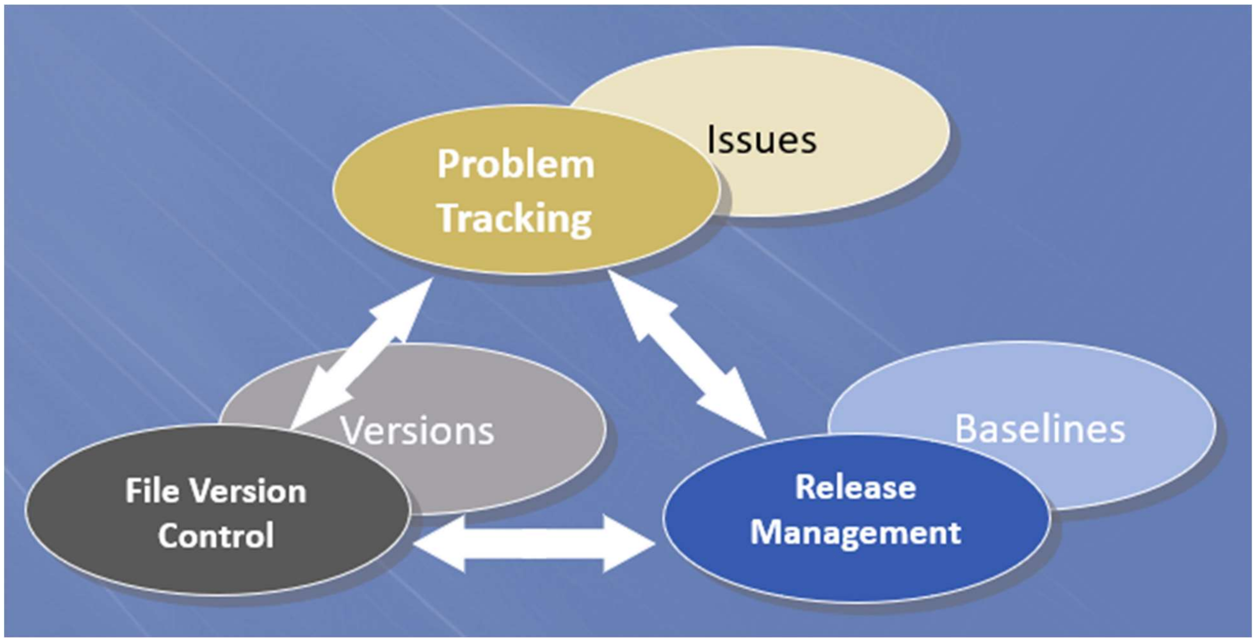


Figure 1: Razor Trilogy

Abstract

The Agile Manifesto (2001) states that in order to follow an agile methodology, one must promote sustainable software development that is attained via early and continuous delivery of valuable software[1]. As such, the tracking and controlling of changes within software is done to promote said continuous delivery. The latter is what is known as software configuration management (SCM) and is a key part of adhering to the notion of having an agile methodology. Razor is an integrated, feature rich suite of tools -- not a conglomeration of purchased products. Razor's core functionality is ideal for software development teams for: issue and problem tracking, version control, and release management. Customers like NASA and NOAA chose Razor for their SCM systems. This white paper will shed further light on how each organization used Razor SCM and the benefits that Razor has been able to provide to each particular use case - NASA and NOAA. Razor got its start as a part of the U.S. Navy's submarine program during the 1990s. Regardless of whether your organization seeks to navigate the depths of the ocean or the vastness of space; Razor and its SCM software suite has a proven track record with decades of success in these spectrums, and many other complex software development initiatives.

Topics:

1. Agile & Software Configuration Management
2. How Razor Enables an Agile Methodology
3. Two Razor Use Cases: NASA and NOAA

[1] *Manifesto for Agile Software Development*, https://agilemanifesto.org/, 2001

Sequential vs. Overlapping Phases of Development

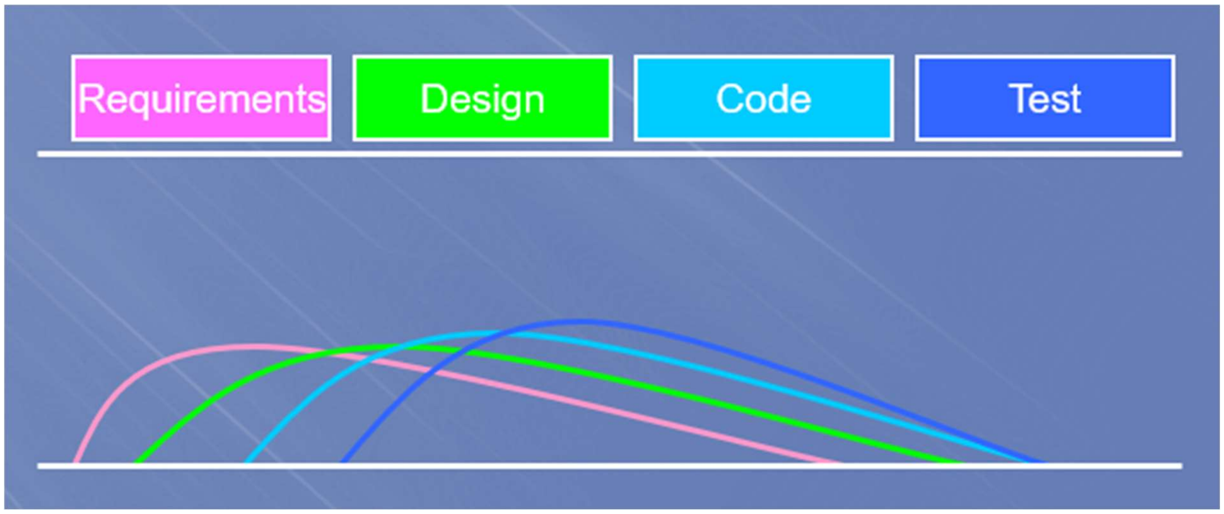


Figure 2: "The New New Product Development Game", Hirotaka Takeuchi and Ikujiro Nonaka, Harvard Business Review, January 1986.

The more things change, the more they stay the same. "The rules of the game in new product development are changing. Many companies have discovered that it takes more than the accepted basics of high quality, low cost, and differentiation to excel in today's competitive market. It also takes speed and flexibility[1]." A white paper from the Harvard Business Review written in 1986 clearly defined the problems and solutions for software development before the agile methodology became well-known.

Being Agile & Having Software Configuration Management

In today's software centric world, deadlines are shorter than ever, and companies are forced to either "sink or swim" when it comes to meeting said deadlines. As such, the companies that follow an 'agile' methodology are the ones who 'swim', and those ones who don't, are the ones who 'sink'. The Agile Manifesto (2001) states that in order to follow an agile methodology, one must promote sustainable software development[3], which is attained via early and continuous delivery of valuable software, commonly done via 'Sprints'. Once more from the Agile Manifesto, an agile methodology is being able to deliver working software frequently, from a

[2] The New New Product Development Game by Hirokata Takeuchi and Ikujiro Nonaka, 1986
[3] *Manifesto for Agile Software Development*, https://agilemanifesto.org/, 2001

couple of weeks to a couple of months, with a preference to the shorter timescale[4]. Hence, Sprints are a short period, often being only 1 or 2 weeks, that teams are assigned to in order to complete a set amount of work.

As such, the tracking and controlling of changes within software is of utmost importance and is done to promote said continuous delivery. The combination of these two is what is known as software configuration management (SCM) and is a key part of adhering to the notion of having an agile methodology.

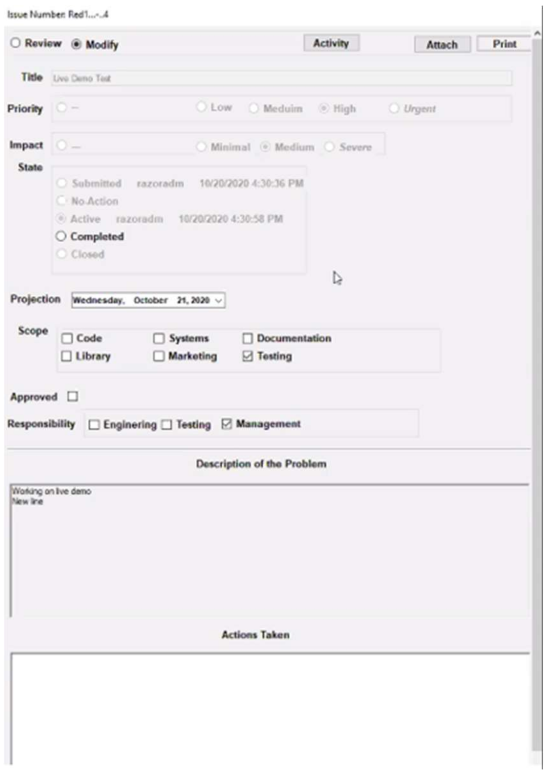
The Razor Configuration Management System

Razor Agile is an SCM system that focuses on three components: Issues, Versions, and Baselines. These components make up the Razor trilogy and provide an additional three key benefits: integrating problem tracking through release management, configuring to your process, and a highly customizable development environment.

Issues

Issues within the Razor configuration management system are a problem tracking system, containing locally defined problem tracking forms containing information relevant to the issue. These forms are completely customizable for roles, approvals, signature chains, and email notifications.

[4] *Manifesto for Agile Software Development*, https://agilemanifesto.org/, 2001


Issue Number: Red1...-..4

Review [✓ Modify]

Title: Live Demo Test

Priority: -- Low Meduim [✓ High] Urgent

Impact: -- Minimal [✓ Medium] Severe

State:
Submitted razoradm 10/20/2020 4:30:36 PM
No Action
[✓ Active razoradm 10/20/2020 4:30:58 PM]
Completed
Closed

Projection: Wednesday, October 21, 2020

Scope: ☐ Code ☐ Systems ☐ Documentation ☐ Library ☐ Marketing ☑ Testing

Approved ☐

Responsibility: ☐ Enginering ☐ Testing ☑ Management

Description of the Problem

Working on live demo
New line

Actions Taken

Figure 3: Razor Issues Form

Versions

Razor provides an intuitive and insightful GUI interface for the standard version control needs; checking files in and out for edit, parallel development, reporting changes, viewing differences, and etc.

- Handles ASCII and binary files
- Provides full history and accountability
- Concurrent development supported through branching/merging
- Easily importing existing Git investments or any type of file

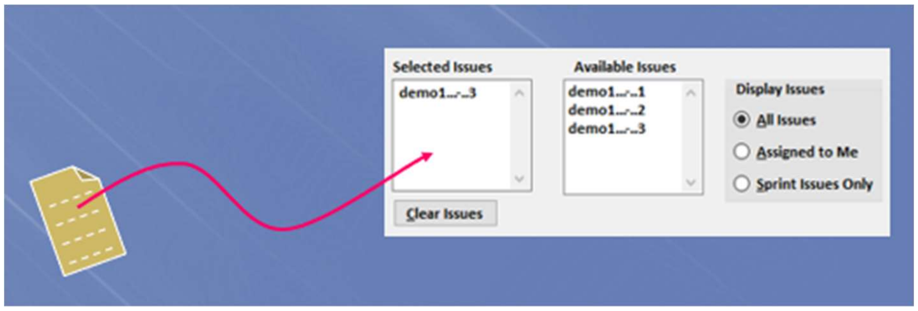


Figure 4: Issue Based Version Control

Razor Agile's version control is capable of handling ASCII and binary files, and with the version control it facilitates; it provides full history and accountability with regards to changes that have been made. A key element of version control is the concurrent development supported through the allowance of branching and then merging said branches. Branches themselves are deviations from the main development hierarchy of a project. Branching is beneficial because it permits more than one person to edit a project simultaneously i.e., parallel development.

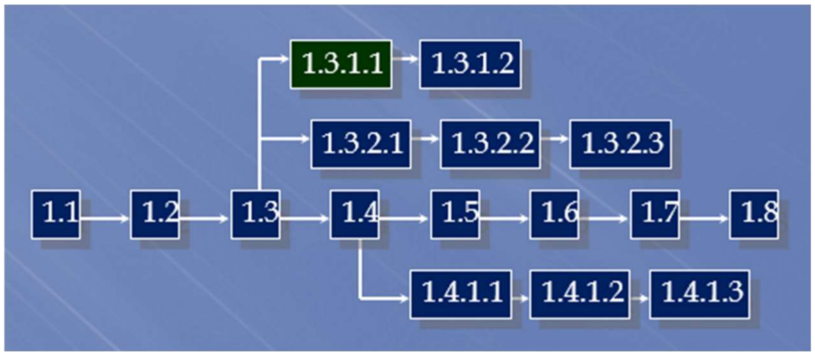


Figure 5: Parallel Development in Razor

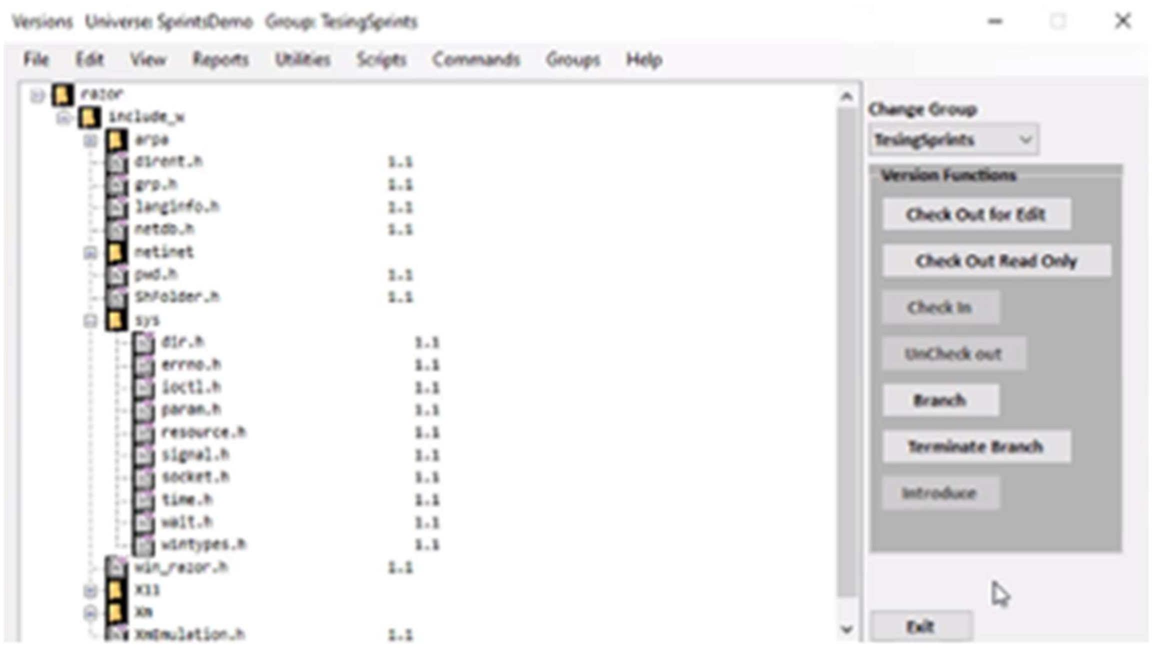
Razor provides full accountability and enforcement of your development process. Triggers may be attached to nearly every Razor action. All user activity is logged for audit traceability. You can customize your state transition model to your needs. Some customers have 10-15 unique steps in the resolution of an issue; others have 4-5, it's up to you. More importantly, Razor conforms to YOUR process, not the other way around. And, Razor can instantly recreate any release ever created for product testing, verification, auditing, or warranty purposes.

Collaborative Change Management

Razor supports parallel development, including: multiple projects or features at the same time. This includes: multiple teams and parallel releases.

Razor's version control system enables parallel development and the merging of source code for any programming language. It activates file locks on individual files or specific file types. When agile teams collaborate through parallel development (multiple engineers are working on the same code base at the same time) which is the default way of working in Razor. Razor will notify specified functionaries when check-in conflicts occur that cannot be merged.

Additionally, the ability to either 'check-in' or 'check-out' a file is another key element within version control. This is because, 'checking-out' a file is the process of obtaining a version of the file so as to work on it, that will then be a new version when 'checked-in'.



```
razor
  include_w
    arpa
    dirent.h          1.1
    grp.h             1.1
    langinfo.h        1.1
    netdb.h           1.1
    netinet
    pwd.h             1.1
    ShFolder.h        1.1
    sys
      dir.h             1.1
      errno.h           1.1
      ioctl.h           1.1
      param.h           1.1
      resource.h        1.1
      signal.h          1.1
      socket.h          1.1
      time.h            1.1
      wait.h            1.1
      wintypes.h        1.1
    win_razor.h       1.1
    X11
    Xm
    XmEmulation.h     1.1
```

Figure 6: Files and revisions are managed in Razor using Git Version Control

Integration with Git Version Control

Files and revisions are managed in Git. Changes are associated with approved Issues/Sprints. Changes that diverge or are temporary are stored as branches. Razor has built features into **Git allowing users to selectively work with a file or group of files and then merge them back in.** Opposed to being required to blanket copy and paste each time a user needs to work with a file or group of files. Any files can be dragged and dropped into Razor and all history will be maintained from that time forward.

In order to associate changes to issues, Razor enforces the relationship of issues to file operations. A trigger action is associated with the introduction and check-in of files to enforce the relationship. This means that only issues which are in the Active state and assigned to the user are allowed to be associated.

Baselines

Within Razor Agile, a baseline represents a collection of files that belong together. Baselines themselves allow file groupings to be made based upon the state of the file; open, inspected, integrated, tested, and/or released. A ‘baseline’ defines which combination of files and versions “belong together” It’s easy to recreate any release or staging area you need. What are baselines? Release management as opposed to individual file management. There is a direct correlation between baselines and objects under version control.

Baselines are not just for members of the SCM team. All team members can/should use baselines to define related collections of items that they are testing. These ‘developmental’ baselines can serve as the basis for formal baselines that ultimately form the basis of a product delivery. Baselines provide an import, a copy, and a compare mechanism for managing this approach. A baseline is defined on a Razor group basis. The level of granularity is up to you. You could have multiple baselines representing small pieces of a group or one baseline encompassing all elements of a group. When multiple Razor groups exist, you need a mechanism to pull all of the pieces together. A project baseline does just that. It is a baseline of baselines.

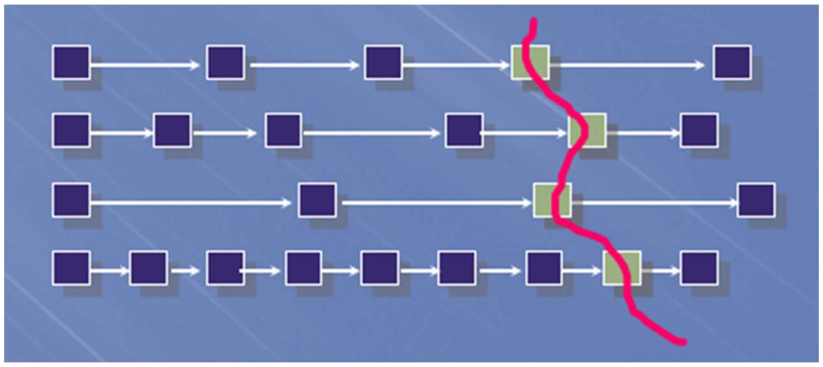


Figure 7: Parallel Development

The Formation of a Razor Agile Project

The Razor Agile Project may be comprised of two teams - a Peer Review Team which consists of people who are interested but who will not be doing the work, and a Development Team. The people who are interested and will be doing the work on the project.

A team typically has no more than 6-9 working members.

If there are more members than manageable, the project should be broken into multiple “groups” within Razor, each focusing on one, self-contained area of work, for example one for quality assurance (QA), one for documentation, etc.

There should be Roles defined in Razor for people to act as bridges - that is, to attend the meetings of more than one team.

Members of teams that are closely related/involved with each other should sit in on the other teams' meetings.

Razor’s Scrum Agile Development Process breaks down into three phases:

1. Planning
2. Sprint Cycle
3. Closure

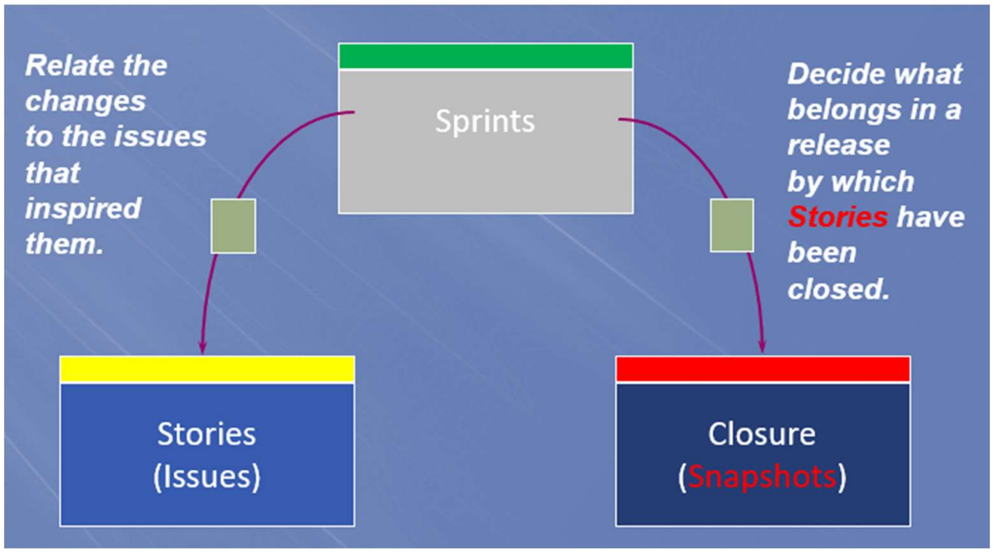
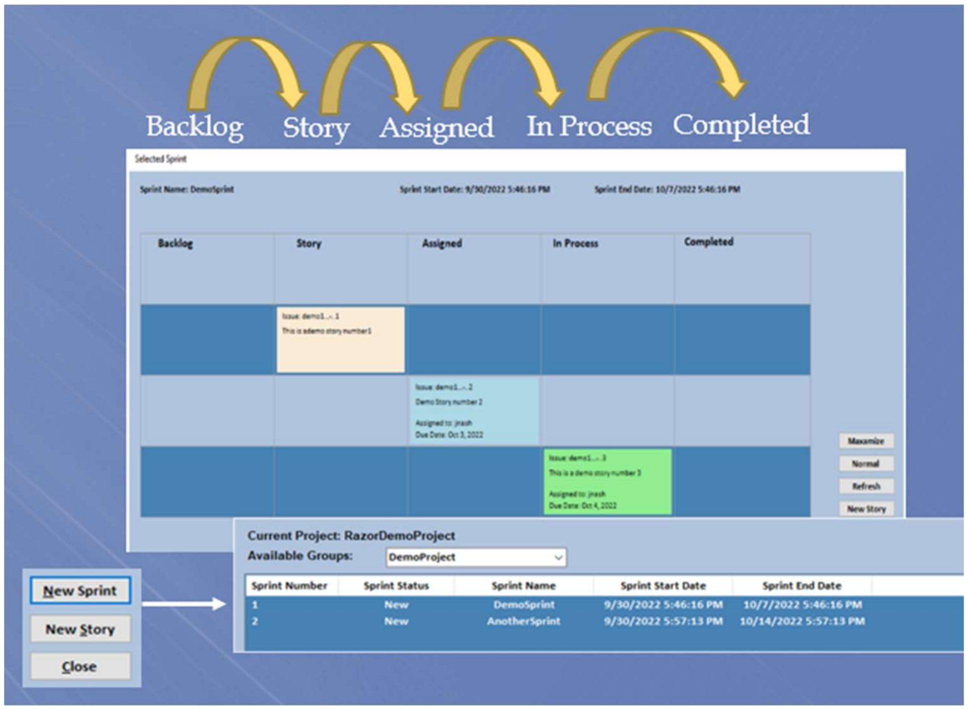

Relate the changes to the issues that inspired them.
Sprints
Decide what belongs in a release by which Stories have been closed.
Stories (Issues)
Closure (Snapshots)

Figure 8: The Razor Agile Framework

Being Agile with Razor

While Razor focuses on providing a well-rounded software configuration suite via its Issues, Versions, and Baselines. Razor promotes an agile environment by having a built-in scene for managing sprints. Having this dedicated sprint scene is pivotal when seeking early and continuous delivery of valuable software. The sprint scene itself is made up of five parts: the backlog, story, assigned, in process, and completed. When used correctly, these parts add up to an additional level of release management that is attributed to the sprint in all the phases of its lifecycle. To elaborate, a story would be the inception portion of the sprint life cycle, whereas the story is defining the task that shall be completed during the sprint. After the sprint has been created, it will be assigned to the relevant person or persons. Now that the relevant party(s) have been assigned, the story can progress to being in progress. Coming to the end of a sprint's lifecycle, a story can be marked as completed and any relevant information should be included in the backlog portion as well. This straightforward breakup of a sprint into the aforementioned parts, is of great value to any sprint lead(er) wanting to have release management at a 'sprint' level.



Sprint Name: DemoSprint   Sprint Start Date: 9/30/2022 5:46:16 PM   Sprint End Date: 10/7/2022 5:46:16 PM

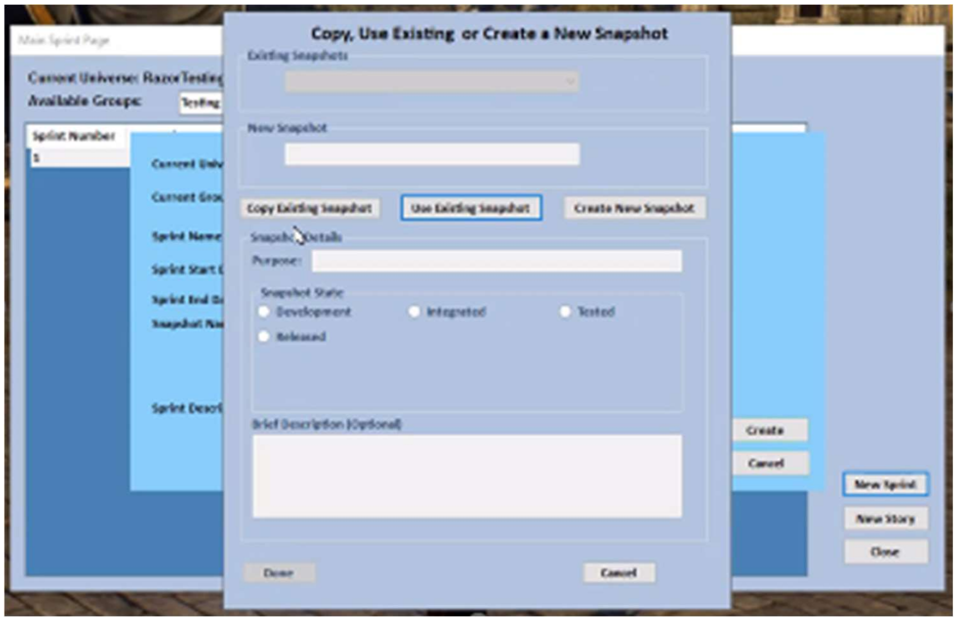
Backlog	Story	Assigned	In Process	Completed
	Issue: demo1...-. 1 This is ademo story number1			
		Issue: demo1...-. 2 Demo Story number 2 Assigned to: jnash Due Date: Oct 3, 2022		
			Issue: demo1...-. 3 This is a demo story number 3 Assigned to: jnash Due Date: Oct 4, 2022	

Current Project: RazorDemoProject
Available Groups: DemoProject

Sprint Number	Sprint Status	Sprint Name	Sprint Start Date	Sprint End Date
1	New	DemoSprint	9/30/2022 5:46:16 PM	10/7/2022 5:46:16 PM
2	New	AnotherSprint	9/30/2022 5:57:13 PM	10/14/2022 5:57:13 PM

Figure 9: Razor Agile Dashboard

Since sprints are known to be short periods and efficiency is key when meeting any deadline, Razor also allows for the use of scripting so that users can be agile.



Current Universe: RazorTesting
Available Groups: Testing

Figure 10: Razor Snapshot

Scripting is Agile

Razor can support any scripting language supported by the Windows or Linux operating systems. As such, many of Razor's users have used scripting to manage complex processes and the automation of the software development lifecycle. A common example of this would be using a script to generate automatic baselines. Automating this particular process within the software development lifecycle allows for a much more streamlined method of tracking reference points within the software development lifecycle. Another common type of scripting that is utilized would be a quality-of-life benefit, which involves having a script that auto fills user info with regards to creating and/or editing issues.

Altogether, the power of Razor Agile is limited only by whatever the user makes of it, and the use of scripting only adds to the capabilities that provide the desired process.

In Razor, you can create new scripts or customize available scripts to reflect your current configuration. You can baseline scripts after successful configuration. And, as your software engineering process changes, you can adapt the definition and customization of your workflow through Razor's rule base and unique scripting capabilities. Razor's functionality is fully extensible through callable external system actions for which script routines can be developed in the programming language of your preference. As your software engineering process changes, Razor can adapt the definition and customization of your workflows through a rule base and unique scripting capabilities.

Providing DFARS/NIST and CMMC Compliance

Razor Agile can help with your DFARS compliance for Configuration Management with respect to NIST SP 800-171 and CMMC for cyber security. With regards to DFARS compliance for Configuration Management, Razor Agile supports the specifications that there are established baseline configurations and inventories of organizational information systems (including hardware, software, firmware, and documentation) throughout the respective system development lifecycles. Razor Agile also promotes the establishment and enforcement of security configuration settings (via encryption in Razor Agile) for information technology products that are employed in organizational information systems.

The specific encryption that Razor employs is between the socket layer and the Razor server; three levels of encryption are available: Standard Encryption, Enhanced Encryption, or Ultimate Encryption, up to 256 bit encryption are supported.

Razor Success Stories

To further showcase how Razor Agile could help your organization, the following stories are from organizations that have used Razor.

The insider story at NASA

Razor has been used by NASA at the Kennedy Space Center to write space shuttle launch processing software. NASA liked using Razor because it allowed them to take their internal development process and implement it within Razor. They also liked Razor's open architecture, which made it easy to administer and configure. Moreover, they liked its simplicity and flexibility as they were adapting Razor to their internal process. Ultimately, implementing Razor allowed for a shorter learning curve, leaving more time available on what was important – that being sustainable software development!

NASA primarily used Razor for the Space Shuttle's Launch Control System to design consoles to communicate with ground support for measuring 40,000 temperatures, measurements, flow rates, turbine speeds, voltages, valves, switches, etc., with their development team of 200 software engineers.

The insider story at NOAA

NOAA, the National Oceanographic and Atmospheric Administration, uses Razor to measure weather around the globe. More specifically, NOAA makes use of Razor's scripting capabilities to do overnight software extractions and builds. Since the weather is always changing, NOAA pulls in new data every day, afterwards, making use of scripting, they update the baseline of their data with whatever new files have been added or modified. Luckily for them, Razor was highly configurable, allowing them to do this process via a command line interface, then have their scripts automate the updating of their baselines within the Razor client.

NOAA takes advantage of scripting to do overnight software extractions and builds. This results in a build every morning. What the developers put in a check-in, gets built overnight. When Razor does the extraction, they do the modifications to the baselines. The developers then go in and see if there are any new files. If there are, they update the baseline, followed by an extraction and a build of the software. They rely very heavily on the build management part of Razor. The baselines are very important to them. Scripting is used a lot for their interface. NOAA's developers use mostly C, C++ as well as the older Fortran language.

Regardless of the type of development being done, it is important that whatever work needs to be done can be streamlined into an agile methodology. Razor can easily be integrated into any software development process and manage the software for the life of your project(s).

APPENDIX

HARDWARE/SOFTWARE ENVIRONMENT AND VARIABLES

RAZOR'S ARCHITECTURE

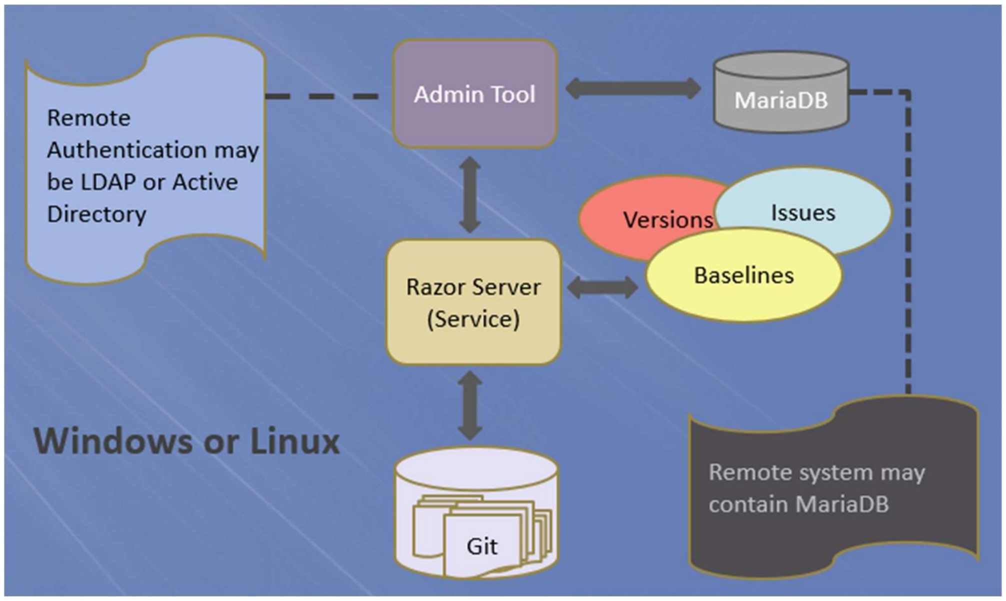

Remote Authentication may be LDAP or Active Directory
Admin Tool
MariaDB
Versions
Issues
Baselines
Razor Server (Service)
Windows or Linux
Git
Remote system may contain MariaDB

Figure 11: Razor's Architecture

COMPONENTS

- WINDOWS
  - RUNS ON WINDOWS 10, 2016, 2019 (64BIT)
  - USES .NET 4.X FRAMEWORK
  - RUNS IN A VMWARE VIRTUAL MACHINE
- MARIADB
  - USED TO STORE SETTINGS, CONFIGURATION, USER INFORMATION, ENCRYPTION PROFILE, LICENSE, ETC.
  - CONTAINS THE GIT FILE AND BASELINE HISTORY
  - OPEN SOURCE AND SPAWNED FROM MYSQL
- GIT
  - USED TO STORE REVISIONS OF ISSUES, FILES, AND BASELINES
  - OPEN-SOURCE FILE REVISION ENGINE AVAILABLE ON WINDOWS AND LINUX
- LINUX
  - RUNS ON UBUNTU 20, RHEL 7/8, CENTOS 7/8 (E.G., 3.X, 4.X, OR 5.X 64BIT KERNELS) USING MONO
  - RUNS IN A VMWARE VIRTUAL MACHINE

- MONO
    - OPEN SOURCE, OWNED BY MICROSOFT, USED TO RUN .NET FRAMEWORK APPLICATIONS ON LINUX

CAPABILITIES

- RAZOR5 DATABASE (I.E. UNIVERSE)
    - CAN BE ACCESSED DIRECTLY FROM THE RAZOR6 SERVER AND CLIENT
    - MAY BE USED AS-IS REMOTELY WITH NO CHANGES
        - OLD GROUPS CONTINUE TO USE RCS OR SCCS
        - NEW GROUPS USE GIT

        - ADMINISTRATION
            - MANAGES ENCRYPTION STRENGTH, USERS, AUTHENTICATION, LICENSING, PERMISSIONS, CONFIGURATION, UNIVERSE, GROUPS
            - MANAGES BACKUP, RESTORE, IMPORT (E.G. MIGRATION FROM RAZOR5)
            - COMMUNICATES WITH THE RAZOR SERVER AND THE MARIADB DATABASE
- SECURITY
    - COMMUNICATION BETWEEN COMPONENTS IS SECURE AND ENCRYPTED
    - CHOOSE BETWEEN THREE LEVELS OF ENCRYPTION (STANDARD=128 BIT, ENHANCED=192 BIT, ULTIMATE=256 BIT)
- AUTHENTICATION
    - MAY USE A LOCAL DATABASE OF USERS
    - MAY USE REMOTE LDAP DIRECTORY SERVER
    - MAY USE REMOTE MICROSOFT ACTIVE DIRECTORY
- CHANGE PROCESS
    - TRIGGER POINTS MAY BE CONFIGURED (E.G. DURING UPDATE OF AN ISSUE, FILE, OR BASELINE)
    - MAY USE SCRIPTS TO CONFIGURE BUSINESS RULES AND GUIDE DEVELOPMENT ACTIVITIES
    - SCRIPTS MAY BE WRITTEN IN BASH SHELL
    - EXISTING SCRIPTS MAY BE REUSED WITH SMALL MODIFICATIONS (I.E. ENVIRONMENT RELATED)

SIMPLE CONFIGURATION

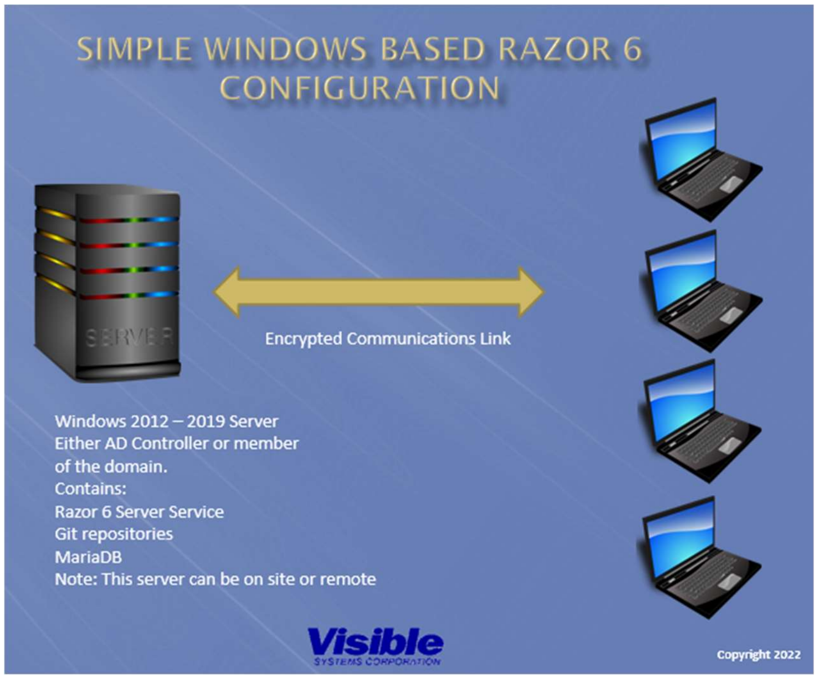

SIMPLE WINDOWS BASED RAZOR 6 CONFIGURATION
SERVER
Encrypted Communications Link
Windows 2012 – 2019 Server
Either AD Controller or member
of the domain.
Contains:
Razor 6 Server Service
Git repositories
MariaDB
Note: This server can be on site or remote
Visible SYSTEMS CORPORATION
Copyright 2022

Figure 12: Simple Configuration

ENTERPRISE CONFIGURATION

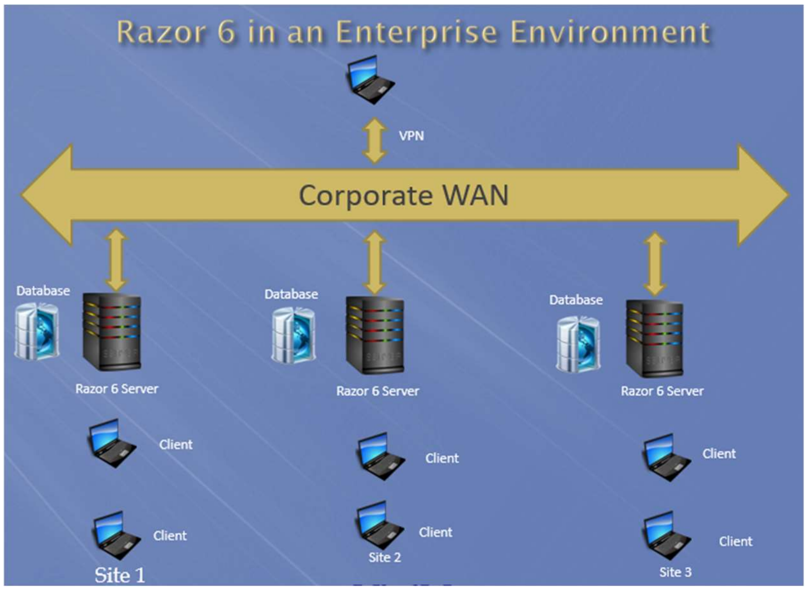


Figure 13: Enterprise Configuration