NSWCDD Software Quality Metrics

Questions? Contact:

Candaice Deloach (SSTM A07S)

(540) 284-0047

Candaice.Deloach@navy.mil

RELIABILITY METRICS

We measure reliability to reduce and prevent severe software malfunctions in order to ultimately increase warfighter effectiveness while decreasing the amount of errors the warfighter must contend with. Report out is expected every 30 days with PTEM submission.

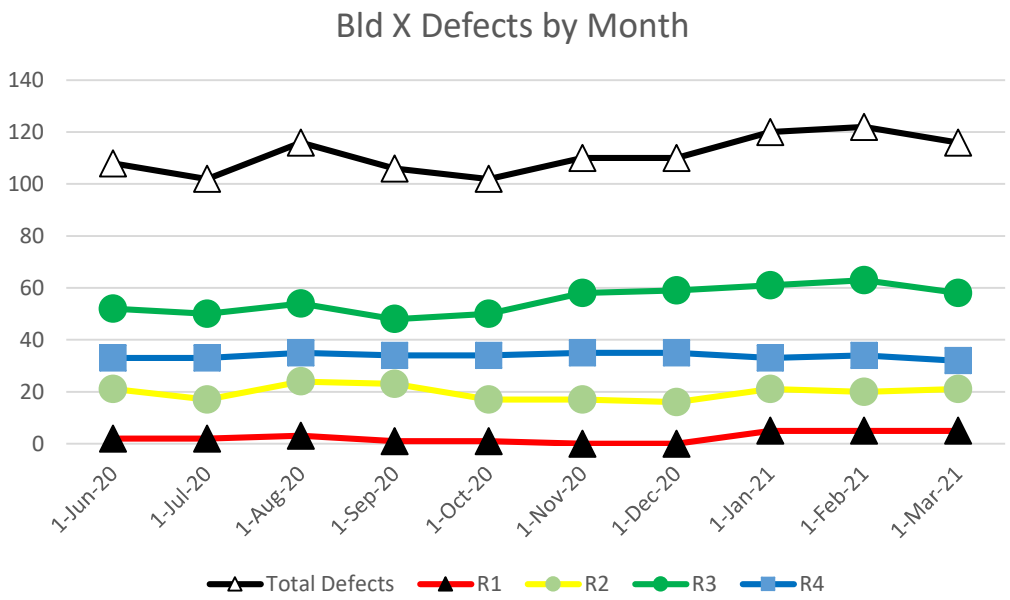
Software Quality Metrics

Metrics that matter	Description	What the metrics tell us
Total Defects Found in the Last 4 Weeks (Internally and Externally)	Looking for 30-day snapshot of new defects introduced over last 30 days. Report out of R1/R2/R3/R4 defects **found** every four weeks. As sprints vary between 2-4 weeks (for Scrum; increments for Kanban), the report out of defect incidents every four weeks will serve as a consistent snapshot in time. Software projects are to utilize **NAVSEA Instruction 9410.2A** regarding the definition of R1-R4.	Allows us to gauge the big picture of the class of defects being introduced every 30 days
Defect Trends	Report out of R1/R2/R3/R4 defects the software is carrying month to month. Software projects are to utilize NAVSEA Instruction 9410.2A regarding the definition of R1-R4.	-How your software quality is trending month to month, release to release. Is your software quality going up or down? -How your software development processes and execution principles are affecting your software quality -Are you trending towards being ready to formally release your software? Is your quality going in the right direction as you get closer to formal release?
Mean Time To Repair (MTTR) in days	Average time it takes to repair defects. Another term is cycle time. It measures the time for a task to go from "started" or "in progress" to "done". Once a baseline value is established, teams can begin measuring variations to detect issues in their process. **Why it is powerful**: A very simple metric that can raise a red flag when items within sprints across your entire system are not moving forward.	Questions that can be answered: -Does your team break down the stories into small enough increments? -Are code reviews picked up quickly enough? -Is there a problem in your pipeline or tooling process? -Are you seeing higher numbers than previous averages? Couple this with rework metrics to further investigate root cause. -What bottlenecks need to be addressed?
Defect Removal Efficiency (of the bugs committed to work)	Gives the measurement of the dev team's ability to remove defects prior to release, and ties to effectiveness of testing vision. **DRE's formula**: DRE= (Number of defects resolved/ Number of defects found internally + Number of defects found externally) × 100.	Questions that can be answered: -How good are we doing at catching defects prior to release? -Do we have gaps in our test infrastructure?

Software Quality Metrics

Metrics that matter	Description	What the metrics tell us
Number of failed fix attempts (external rework)	Defects that failed test and required rework. External rework represents defects that failed test after the software is released (informally/formally), which were originally marked as fixed/verified.	High external failed rework could indicate: -Deficiencies in local test environments or test procedures -Deficiencies in communication internal and external to the dev team (e.g., requirements to code understanding)
Number of failed fix attempts (internal rework)	Defects that failed test and required rework. This is internal rework before the software is released, and represents when developers mark an item as fixed and it fails internal test.	High internal rework could indicate: - Was the ticket poorly defined? - Was there too much scope to accomplish? Do we need to split the effort into smaller chunks? - Is there a disconnect within the dev team on understanding the capability being implemented? - Could this have been detected by an automated test? Should an automated test have been written to catch this earlier in the sprint cycle?
Change Failure Rate w/in Release Cycle	The number of new defects introduced by a fix in your development cycle (sprint cycle); Regression related.	High change failure rate could indicate: -A need to refactor your code -Code that is too tightly coupled -Maintainability issues -Issues in test coverage, whether automated, unit, or regression
Automated code coverage (percentage)	Automated developmental tests provide a means of ensuring that updates to the code do not break previous functionality and that new functionality works as expected. Ideally, for each function that is implemented, a set of automated tests are constructed that cover both the specification for what the performance should achieve as well as the code that is used to implement that function.	The percentage of specifications tested by the automated test suite provides rapid confidence that a software change has not caused some specification to fail, as well as confidence that the software does what it is supposed to do. **Target percentage is 80%.**

Defect Trends Example



Click the image below if you would like to use the excel template to build your trends.

	Bld X.X	Enter data below				
ID	MONTH	Total Defects	R1	R2	R3	R4
1	2-Dec-19	122	7	26	54	35
2	17-Jan-20	121	5	24	60	32
3	14-Feb-20	125	6	27	61	31
4	13-Mar-20	134	4	28	72	30
5	17-Apr-20	114	2	23	55	34
6	14-May-20	113	2	22	55	34
7	11-Jun-20	108	2	21	52	33
8	15-Jul-20	105	2	19	50	34

Reliability Reporting Example

	R1	R2	R3	R4	Comments
Total Defects Found in the Last 4 Weeks (Internally and Externally)	1	4	7	11	
Mean Time To Repair (MTTR) in days	5	10	15		
Defect Removal Efficiency (of the bugs committed to work)	50%	70%	60%		
Number of failed fix attempts (external rework)	0	0	0		
Number of failed fix attempts (internal rework)	1	1	1		
Change Failure Rate w/in Release Cycle	1	1	0		
Total open as of today	4	15	45		
Test Code Coverage Percentage					List percentage here and any applicable comments

The goal is for projects to export these metrics from their respective toolsets. As such, if the data is present, the design of the template does not have to be strictly adhered to. The importance is placed on the metrics provided and ensuring ease of readability by the consumer. If you can export, then export.

MAINTAINABILITY

We measure maintainability to understand how quickly software can adapt to required changes, and/or its ability to be ported or reused for other applications. Maintainability relates to the size, consistency, structure, and complexity of the codebase. Measuring maintainability isn't one equation, but instead a cumulative look at testability (e.g., how well the software supports testing efforts), understandability (e.g., how well you can decipher what the code is doing upon review; readability of the code), portability (e.g., how usable the same software is in different environments), modifiability (e.g., how easy it is to change the code), and reusability (e.g., whether existing assets such as code can be used again). Report out is expected every 30 days with PTEM submission.

Key Quality Indicators

- Software test strategy implemented through areas such as Unit Testing and Test-Driven Development:
  - What is your percentage of code coverage?
- Do you identify areas of Technical Debt and how do you prioritize and plan to address it?
  - "If you are able to estimate roughly the time needed for fixing what is not right into your code, the principal of your debt, you could compare this information to other project data, like remaining days before release date." (www.agilealliance.org)
- Does your project have a style guide and coding standards that are adhered to and enforced during peer/code reviews?
  - Do these guides/standards account for code commenting expectations?
- What software documentation do you develop that supports maintainability?
- Does your current software architecture support reusability?
- How often are you factoring modernization of your code into your sprints?
- Do you have explicit architecture owners?
- Have you considered utilization of code metrics such as Maintainability Index, Cyclomatic Complexity, Depth of Inheritance, and Class Coupling?
- How do you maintain your safety critical code?
  - Do you tag it?
  - Do you map the safety critical code to safety critical requirements?

Software projects are expected to account and answer for maintainability, and utilize the question set above to support this focus area.

SOFTWARE ASSURANCE

Software projects are expected to identify and track their vulnerabilities. Both static and dynamic code analysis tools are to be utilized to identify vulnerabilities, and subsequently tracked using their applicable defect management tool. Integrating these tools whether upon check-in of source code or prior to creating a software build will bolster your software security posture and provide for a stronger code base. Report out is expected every 30 days with PTEM submission.

Software Assurance

- List the secure code analysis tools utilized
- Describe how those tools are integrated into your software development process
- Provide timeline if the tools are not integrated
  - Planned timeline for integration
  - Current status of integration