Software Reliability

Software reliability can be viewed from two different perspectives: (1) Does the software provide a reliable output given a specified input? (2) Does the software run for an operationally useful period of time without "hanging" or "crashing"? The first perspective is considering the functionality of the software and the implementation of the algorithm(s). The second perspective is considering the software architecture, design and coding guidelines used to develop the code. While getting the right answer (perspective 1) is important, making sure that the end user can get an answer at any point in time and especially in a critical situation is equally important. This whitepaper will focus on the second perspective of software reliability. Through the use of modern software analysis tools and subject matter experts, the Navy can potentially save $1M per APB cycle.

Background:

During the decades of the 1970's and 1980's, military systems were designed with an intended life time of 30 years with upgrades every 5 to 7 years. Systems were migrating from primarily hardware implementations to a combination of hardware and software. These "new" software intensive systems had to provide the warfighter with guaranteed performance throughout a mission without "rebooting" or "restarting". As a result, great emphasis was placed on developing "error free" code. Carnegie Mellon University in Pittsburgh, PA initiated the Capability Maturity Model Integration (CMMI) effort to improve the software development process. During this time, defense companies including IBM Federal Systems, Raytheon, Boeing, Lockheed Martin and others developed processes that would implement CMMI guidelines and transition their software development efforts from a typical CMMI level 2-3 where the focus is on program management and engineering processes to a CMMI level 5 where the focus is on continuous improvement and defect prevention. Much of this effort focused on documenting the software design and formalizing the review of the code during development. Although there were no clearly defined rules for estimating Source Lines of Code (SLOC), each company measured their productivity by how many SLOC could be generated per labor month and measured reliability by how many Program Trouble Reports (PTRs) were generated per 1000 SLOC. In the mid 1980's the average number of PTRs per 1000 SLOC was 10-12. The best reported PTR rate was reported by IBM Federal Systems at 0.5 PTRs per 1000 SLOC.

To achieve these low PTR rates companies generally implemented manual code reviews that could take from a few days to weeks to complete. The intent of these reviews was to ensure that:

1. that there were no memory leaks,
2. that the designer had complied with established design standards and incorporated defensive code to prevent erratic behavior if an unexpected input occurred
3. that the designer had implemented error recovery by returning to the proper re-entry point in the code
4. that memory buffers were properly initialized on start-up
5. that memory was being properly locked and unlocked by each process

The impact of implementing these types of code reviews is shown in Figure 1. As a result of the drastic decline in PTR rates over this 14 year period, there was corresponding reduction in the integration effort required to ensure that the system was ready for deployment. As a general rule of thumb, it is at least 2:1 less costly to fix a defect in the development process than to fix a defect once the software is in integration and at least another 2:1 less costly to fix a defect in integration than once the software is deployed.

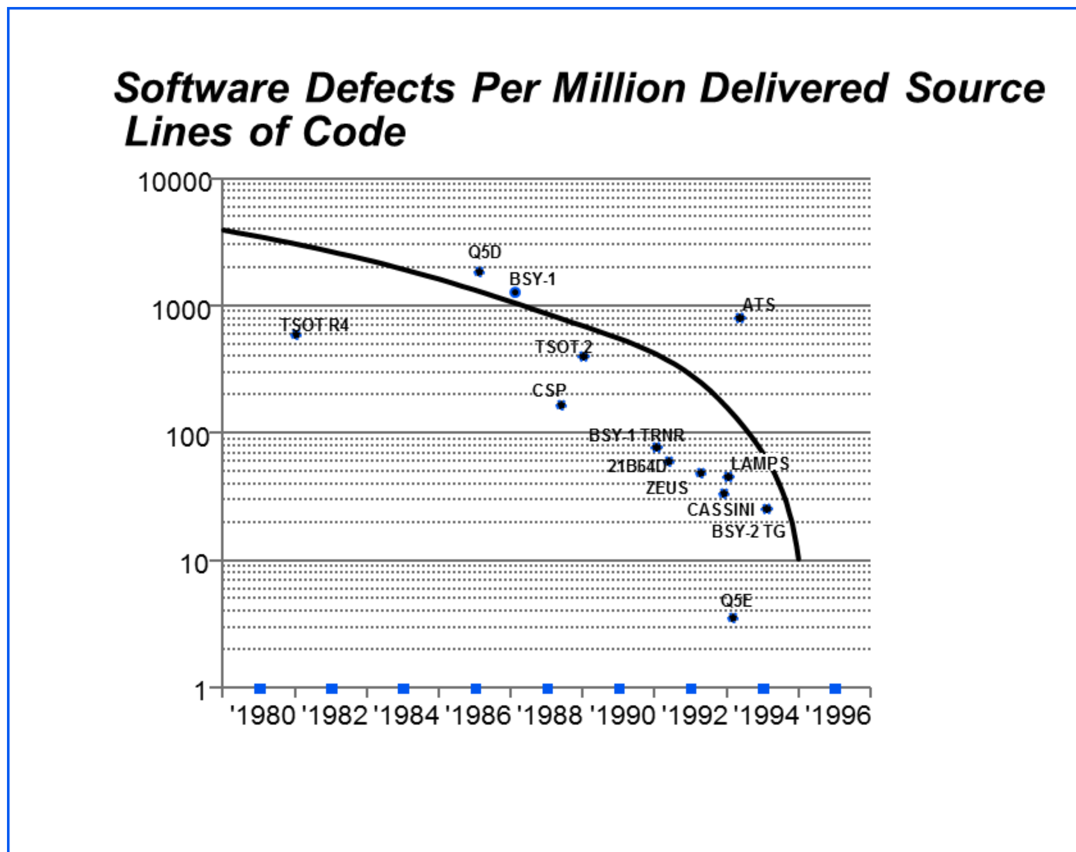

Figure 1 - Software Defect Rates Driven by Improvements the S/W Development Process[12]

It was therefore shown that detecting and fixing software defects early in the development process made a significant cost reduction in the overall system development and deployment.

While, today's high level languages and modern software development techniques have made significant strides in reducing the potential for errors, because humans are involved in the process, software defects still find their way into the code. Because there are software defects present in the code software reliability becomes an issue. Not all defects impact reliability but those that do can have a major impact on end user acceptance, mission success, system performance, ship safety, and cost due to the time to identify, isolate and repair the defect.

---

[1] IBM Federal Systems Division software development statistics, 1998

Process to Improve BYG-1 Reliability

This whitepaper proposes a two-step process to improve software reliability for the AN/BYG-1 system.

Step 1: In parallel with current on-going development, analyze a subset of the code that is currently deployed for the AN/BYG-1 system to determine if software architecture and coding defects are prevalent. If defects are prevalent then extend the process to analyze APB13 software updates prior to APB step 4. If software architecture and coding do not appear to be a major contributor to the BYG-1 reliability issue then discontinue the process.

Step 2: Given that step 1 on indicates software and coding defects are contributors to the BYG-1 reliability issue, establish a collaboration team between industry and the Navy that will implement the process applied during step 1 in a more formal basis. There are multiple options ranging from updating the current APB process to include this analysis as an IWS 5 activity to involving an independent third party to perform the analysis and make recommendations.

Example of the software analysis process:

Figure 2 below depicts the software analysis process as implemented for the ISIS effort (PMS 435 2007 thru 2010) and the RMS effort (PMS 403 2011 thru 2012). The process is intended to provide the same insight into the software architecture and code that was gleaned through manual software peer reviews. However, instead of consuming days and weeks to analyze the code, the use of modern tools such as PDM and Parasoft, allows this analysis to be done in a matter of hours once the code is available and the tools are installed and operational. While the tools provide a comprehensive view of the software architecture and code, the value add comes from: (1) properly interpreting the output of the tool(s), (2) being able to identify the impact to the operational system and (3) prioritize which defects need to be addressed before deployment and which can be delayed and those that require no further attention.

Results:

Before this process was applied to the ISIS system at the TI-04 level, the average time between re-boots was 1.8 hours. After applying this process the average time between reboots increased to over 75 hours. The RMS failed its reliability KPP during Operational Assessment. Our analysis of the RMS system to date has shown a high degree of dependency between software modules making it difficult to update. Secondly, the RMS documentation was shown to be back level and not reflect the current software design making it difficult to maintain. Thirdly, the detailed code analysis indicated that there were three threads that could lock memory but four could unlock memory. While this may be acceptable, it may also represent an opportunity for memory to be corrupted by a user.

On both programs, after analysis, the developer instituted additional steps in software development processes to correct defects like these before the software is deployed.

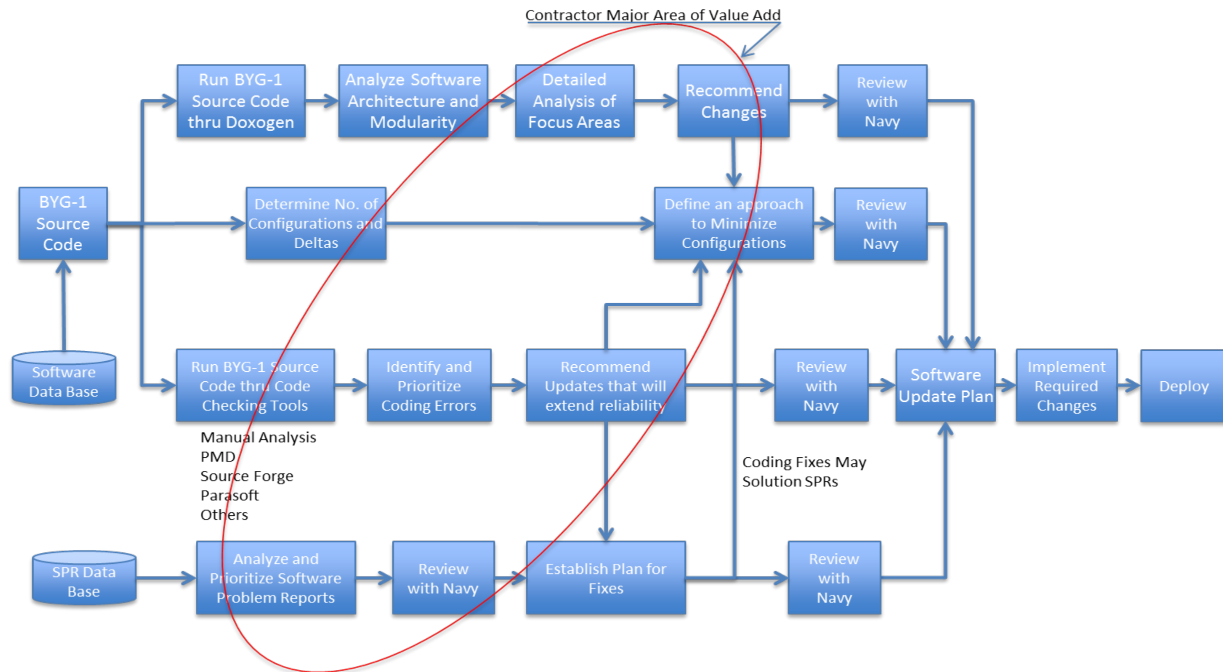

Figure 2 - Software Analysis Process

Cost:

Because we no longer measure code in terms of source lines, it is difficult to address the rate of defects within software. However, if we take a representative example such as RMS, the software package resident on the vehicle has approximately 600 open PTRs. If we assume that BYG-1 is similar we can apply the following approximation:

600 PTRs total

60% Low priority or can be ignored

240 PTRs that should be addressed prior to deployment

0.5 labor months per PTR required for debug and fix

0.25 labor months for retest

0.75 labor months per PTR for find, fix, and retest

Therefore 0.75 x 240 = 180 labor months or 15 labor years @250K per labor year = $3.75M

If applying the process outlined in the previous section can eliminate 50% of the defects that require fixing prior to deployment then there would be a savings of $1.8M

For the RMS project, the cost of setting up and executing this process is estimated to be $750K per year or approximately a 2.5 Full Time Equivalent effort. This would leave a net savings to the Navy of $1M and a highly reliable system to supports the warfighter.