A NEW PROCESS MODEL FOR EMBEDDED SYSTEMS CONTROL FOR AUTOMOTIVE INDUSTRY

Dr. Adnan Shaout* and Tejas Chhaya*

*The University Of Michigan – Dearborn, 4901 Evergreen Road, Dearborn, Michigan 48128 USA
shaout@umich.edu, chhayat@gmail.com

Key words – Software processes, Embedded systems controls, Automotive, Quality

ABSTRACT

For the major organizations, businesses and government agencies the biggest constraints are cost, schedule, reliability, and quality for a given software product. And hence, more and more emphasis is put on software processes asking software engineers to follow it. The goal of this paper is to present a modified software process model using the Personal Software Process $^{\text{SM}}$ (PSP$^{\text{SM}}$), Team Software Process $^{\text{SM}}$ (TSP$^{\text{SM}}$) and six sigma. The new process model was used for an embedded systems project in automotive industry with ‘Moderately complex and medium size’. The result of using this new process model show 70% improvement in defects/KLOC.

INTRODUCTION

In a big organization for a given product usually there is lots of different people work within a group/team for which an organized effort is required to avoid repetition and get a quality end product. Typically, for big and complex projects there are many teams work with one goal to deliver the FINAL product. Efforts are required to coordinate hardware, software, and systems among these teams as well as for resulting issues at various levels. In order to succeed for such a high degree of complex project a structured process is required that is not too complicated and yet very efficient.

1 QUALITY, RELIABILITY, COST & SOFTWARE PROCESS

1.1 WHAT IS QUALITY?

A quality can be defined as a fuzzy linguistic variable since quality can be very subjective. What is of a high quality to someone might not be a high quality to another. It can be defined with respect to an attribute such as cost or reliability. It is a degree of membership of an attribute or a characteristic that a product or software can or should have. For example, a product should be reliable, or a product should be both reliable and useable, or a product should be reliable or repairable. Similarly, software should be affordable, efficient, and effective [1].

1.2 WHAT IS RELIABILITY?

The IEEE defines reliability as “The ability of a system or component to perform its required functions under stated conditions for a specified period of time.” [2]

1.3 RELIABILITY AS A QUALITY ATTRIBUTE

ISO 9126 defines six quality characteristics, one of which is reliability. IEEE STD. 982.2-1988 states “A software reliability management program requires the establishment of a balanced set of user quality objectives, and identification of intermediate quality objectives that will assist in achieving the user quality objectives” [3]. Since reliability is an attribute of quality, it can be concluded that software reliability depends on high quality software.

1.4 COST AS A QUALITY ATTRIBUTE

Quality is always deemed to have a direct relationship to cost – the higher the quality standards, the higher the cost. Quality may in fact have an inverse relationship with cost in that deciding to meet high quality standards at the beginning of the project/operation may ultimately reduce maintenance and troubleshooting costs in the long term.

1.5 WHY SOFTWARE PROCESSES?

Software processes enable effective communication among users, developers, managers, customers, researchers, and public authority and commission industry. They enhance management’s understanding, provide a precise basis for process automation, and facilitate personnel mobility and Process-reuse.
that yields in reduction of Cost, increase in Reliability, Productivity and Quality.

1.6 WHICH SOFTWARE PROCESS?

There are quite a few software processes. Below is the list of process methods that are either in use or were used in past, for various types of projects in different industries.

1. PSP and TSP
2. Waterfall Model
3. V-Model & V-Model XT
4. Spiral Model
5. Chaos Model
6. Top Down and Bottom Up
7. Six Sigma
8. Model Driven Engineering
9. Iterative Development Process
10. Agile Software Process
11. eXtreme Process (XP)
12. LEAN method (Agile)
13. Wheel and Spoke Model
14. Constructionist Design Methodology

2 SPIRAL MODEL, PSP, TSP, and SIX SIGMA PROCESSES

In the later stage of the paper PSP and TSP are mapped to Spiral Model. And hence, let us go through some basic aspects of these Processes.

2.1 SPIRAL MODEL

The spiral model is shown in Figure 1. It is also known as the spiral lifecycle model. This model of development combines the features of the prototyping model and the waterfall model. The steps in the spiral model can be generalized as follows [4]:

1. The new system requirements are defined in as much detail as possible.
2. A preliminary design is created for the new system.
3. A first prototype of the new system is constructed from the preliminary design.
4. A second prototype is evolved by a fourfold procedure: (1) evaluating the 1st prototype in terms of its strengths, weaknesses, and risks; (2) defining the requirements of the 2nd prototype; (3) planning and designing the second prototype; (4) constructing and testing the second prototype.
5. At the customer's option, the entire project can be aborted if the risk is deemed too great.
6. The existing prototype is evaluated in the same manner as was the previous prototype, and, if necessary; another prototype is developed from it according to the fourfold procedure outlined above.
7. The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.
8. The final system is constructed, based on the refined prototype.
9. The final system is thoroughly evaluated and tested. Routine maintenance is carried out on a continuing basis to prevent large-scale failures and to minimize downtime.

2.2 PSP

The PSP is a personal process for developing software or for doing any other defined activity. The PSP includes
- Defined steps
- Forms
- Standards

PSP PRINCIPLES

The quality of a software system is determined by the quality of its worst components. The individual who developed it governs the quality of the software component. The quality of a software component is governed by the quality of the process used to develop it. The key to quality is the individual developer’s skill, commitment, and personal process discipline [5, 6, and 7]. As software professional, one is responsible for one’s personal process. And should measure, track, and analyze one’s work. Next, from the performance variations then one should incorporate lessons learned into the personal practices [5, 6, and 7].

THE PSP0 PROCESS

The first step in the PSP is to establish a baseline that includes some basic measurements such as
time recording, defects found and its reporting format. If there is no regular process, then PSP0 should be used to design, code, compile, and test phases done in whatever way one feel is most appropriate [5, 6, 7].

**PSP1 – PERSONAL PLANNING PROCESS**
PSP1 adds planning steps to PSP0. The initial increment adds test report, size and resource estimation. In PSP1.1 task and schedule planning are introduced. The intention is to understand the relation between the size of the programs software professional to develop and the required time to develop them. This should help make commitments that the software professional can meet, to give an orderly plan for doing the work, and to give a framework for determining the status of a given software [5, 6, 7]

**PSP2 – PERSONAL QUALITY MANAGEMENT**
PSP2 adds review techniques to PSP1 to help software professional find defects early when they are least expensive to fix. PSP2.1 addresses the design process. Here, PSP does not tell software professional how to design but rather how to complete a design.

**PSP3 – A CYCLIC PERSONAL PROCESS** -
There are times when program gets bigger. e.g. program of 10,000 lines of code (LOC). This type of program is too big to write and debug and code review, using PSP2. The strategy is to subdivide a larger program into PSP2-sized pieces. Each enhancement builds on the previously completed increments, so PSP3 is suitable for programs of up to several thousands LOC (KLOC) [6, 7]. Figure 2 shows the evolution within each of the PSP stage and its final evolution to PSP3.

**TSP – TEAM SOFTWARE PROCESS** -
Using PSP3, programs can be built with over 10 KLOC. However there are two problems:

**First** – as the size grows, so do the time and effort required.

**Second** – most engineers have trouble visualizing all the important facets of even moderately sized programs. There are so many details and interrelationships that they may overlook some logical dependencies, timing interactions, or exception conditions. This will cause missing obvious mistakes because the problem is compounded by habituation, or self-hypnosis [7].

### 2.3 SIX SIGMA

Six Sigma is a lot of things: a methodology, a philosophy, an exercise in statistics, and a way of doing business, a tool for improving quality. Six Sigma places an emphasis on identifying and eliminating defects from one's products - be they power converters, sales quotations or a proposals to a customer [8]. The goal is to improve one's processes by eliminating waste and opportunity for waste so much that mistakes are nearly impossible. The goal of a process that is Six Sigma good which is a defect rate of only a few parts per million. Not 99% good not even 99.9% good, but 99.9997% good [8]! Before diving into the meaning of the Six Sigma terminology, a main enemy threatening any development process should be agreed upon: Variation. Sigma (σ) in the mathematics field is a metric used to represent the standard deviation (variation) from the mean to a specific limit. Six sigma is a representation of 6 standard deviations from the mean. Six Sigma is almost defect-free - Six sigma, the term quantitatively means that the process produces fewer than 3.4 defects per million units (or opportunities). That represents an error rate of 0.0003%; conversely, that is a defect-free rate of 99.9997% [9].

<table>
<thead>
<tr>
<th>Sigma</th>
<th>DPMO</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>691,462</td>
<td>30.9</td>
</tr>
<tr>
<td>2</td>
<td>308,538</td>
<td>69.1</td>
</tr>
<tr>
<td>3</td>
<td>66,807</td>
<td>93.3</td>
</tr>
<tr>
<td>4</td>
<td>6,210</td>
<td>99.4</td>
</tr>
<tr>
<td>5</td>
<td>233</td>
<td>99.98</td>
</tr>
<tr>
<td>6</td>
<td>3.4</td>
<td>99.9966</td>
</tr>
</tbody>
</table>

The Table 1 show that how exponential the sigma scale is between level 1 and 6.

**WHAT IS DEFECT?**
The simplest, most global definition of defect is anything that causes customer dissatisfaction.
This may be a product that does not work, an incorrect component inserted on the manufacturing line, a delivery that is not on time, or a quotation with an arithmetic error.

**SIX SIGMA – DMAIC**

Originally, the model was mostly concerned with problem solving to enhance processes by reducing defects and variation that would cause customer dissatisfaction. DMAIC stands for a set of five phases arranged in the following order: Define, Measure, Analyze, Improve and Control. The five phases of the DMAIC methodology are,

1. The Define phase – D
2. The Measurement phase – M
3. The Analysis phase – A
4. The Improvement phase – I
5. The Control phase – C

**SIX SIGMA TOOLS**

Six Sigma provide a set of tools making the process clear and structured and therefore easier to proceed through in order to save both time and effort and get sooner to the final goal. Some of those tools are Kano model, Benchmark, GQ(I)M and Indictor templates, Data Collection Methods, Measurement System Evaluation, Failure Mode Effects Analysis (FMEA), Statistical Interference, Reliability Analysis (RA), Root Cause Analysis, Hypothesis Test, Design of Experiments, Modeling, ANOVA, Decision and Risk analysis, PSM Perform Analysis Model, Control Charts, Pareto Chart, Cause-&-Effect Diagram, Base lining, Surveying methods, and Defect Metrics.

**SIX SIGMA – DFSS**

Although the terminology is misleading, allowing us to assume that DFSS and Six Sigma are interrelated somehow, DFSS is in its roots a distinguishable methodology very different than the Six Sigma DMAIC because it is not intended to improve, but to innovate. Moreover, in opposition to DMAIC, the DFSS spectrum doesn’t have a main methodology to be applied like is the case for Six Sigma, but rather has multiple different models. Subir Chowdhury presents it in a more detailed statement saying “instead of simply plugging leak after leak, the idea is to figure out why it is leaking and where and attack the problem at its source” [10].

**DIFFERENT VARIANTS OF DFSS:**

As mentioned earlier, the DFSS doesn’t have one strictly defined methodology like the DMAIC of Six Sigma, instead multiple models are being employed by different organizations. Some of those most adopted models are the following:

- **DMADV:** Define, Measure, Analyze, Design, and Verify
- **IDDOV (or DIDOV):** Identify, Define, Develop, Optimize, Verify.
- **IDOV:** Identify, Design, Optimize, Validate
- **DMEDI:** Define, Measure, Explore, Develop, Implement
- **DCCDI:** Define, Customer, Concept, Design, Implement

It is up to the organization to decide which implementation of the DFSS will fit best its industry, development environment, and culture. Regardless of where it is utilized, the goal of Six Sigma DFSS is to solve problems by finding and focusing on their root-causes, verifying the ability to turn the problems on and off, and then instituting controls to ensure the problems do not reemerge [11]. In an embedded software context, DFSS can bring Six Sigma rigor to new product development. [11].

3 **APPLYING PSP and TSP IN REAL WORLD**

Here PSP and TSP were mapped to Spiral Model as shown in Figure 3 for an automotive application.

![Figure 3 – Practicing PSP & TSP using the Spiral Model](image)

**‘MODERATELY COMPLEX AND MEDIUM SIZE’ PROJECT**

Spiral model was chosen as a base model over other models because of its effectiveness for embedded projects with prototype iterations. To evaluate these methods moderately complex and Medium (M&M) with size of 10KLOC project was chosen. In this case a ‘Moderately complex and Medium size’ software project in the range of 10KLOC was chosen to understand the
efficacy of PSP and TSP while using spiral model as shown in Figure 3.

3.1 APPLICATION – ELECTRICAL POWER STEERING

REQUIREMENTS

Discussions were held with vehicle system team and steering system team to identify the high level requirements. Next, the system requirements, interface to vehicle, design guide lines, vehicle standards (SAE & ISO), Safety Standards application implementation and integration environment and team interfaces were discussed and agreed upon during this phase. Each of the requirements was discussed thoroughly with internal and external interfaces. The following were the requirements:

1. Electronic Control Unit and Sensor Interfaces:
   This section details requirements related to interfacing of position sensors, torque sensor, temperature sensors and current sensors with electronic control unit. In general type, operating range, sensitivity, error bands, resolution, supply voltages, number of sensors required, interface, placement, and enclosure requirements.

2. Electronic Control Unit (ECU) – Software -
   The detailed software interface requirements document was prepared for software variables related to sensor(s) measurement, resolution, accuracy, error diagnostics, and for local/global information handling. Also, detailed algorithm and controls document was prepared for controls related local and global software variables, error diagnostics, and software interfaces with other software module. The following were the high-level requirements that were further detailed in either the sensor interface or algorithm and controls requirements document.
   - Communication protocols and diagnostics requirements
   - Control Voltage – Low Voltage
   - Motor Power – High Voltage
   - Resolver interface, range and measurement
   - Encoder interface, range, measurement, resolution, and min-max limits
   - Torque sensor interface, measurement, range, resolution, and min-max limits
   - Temperature sensor interface, measurement, range, resolution, and min-max limits
   - Motor frequency-voltage-current range, measurement, resolution, and min-max limits

PLANNING

Understanding the requirements in detail a ‘Program Plan’ consisting of time-line, the deliverables at each milestone, and final buy-off plan were prepared. Before the requirements discussions roughly eight personnel were chosen to handle different task to finish the system in eight weeks based on the engineering judgment because of unavailability of the past data. During the initial stage it was decided to reduce the head count to six since the design was heavily based on the previously exercised concept.

DESIGN

Upon the detailed requirements understanding, the design was based on a previous concept that required adopting the proven architecture to the new vehicle with minimal changes at architecture level. The addition to the previous architecture was adding the Measurement Validity Algorithm to ensure the sensor measurement. As shown in Figure 3 PSP and TSP mapped to Spiral Model was used for this embedded controls project. As per below Figure 4 Encoder, Torque sensor, Resolver, Motor Temperature, and Inverter Temperature were interfaced to ‘Sensor Measurement’ block. Sensor diagnostics block was designed to check power-on sensor health and periodic sensor health check and to report sensor errors upon detection. If sensors were determined to be good, and no hybrid and motor safety interlock fault or ECU health check faults were set then ‘NO FAULT’ flag was SET. Measurement validity algorithm block was designed to determine the validity of the sensor measurement. Vehicle parameters such as torque, RPM, speed, acceleration, deceleration, motor phase R, S, and T voltage and current were fed to Motor Control Algorithm block in addition to the measurements from Sensor Measurement block. Finally this
block determined the required amount of steering angle by determining required motor voltage and current for R, S and T phases of motor.

**CODE, COMPILe AND TESTING**

Here the scope is not to discuss the software implementation since the intention is to evaluate the software process and its effectiveness on the software product quality and reliability. As decided during the design phase, with portability of available code and modification of a new functionality, coding phase was concluded. The changes in the Application Layer were hand coded in C++. It was decided that on later stage it would be transferred to Matlab environment. Modular coding approach was taken and each module was checked with its corresponding functional requirements by a coder. After approximately four weeks, core modules were made available and integration phase was started. Test cases for white box testing and black box testing with hardware-in-loop were written jointly by test engineer and coder and reviewed with different teams. Time Recording Log, Table 2 – PSP Project Plan Summary

<table>
<thead>
<tr>
<th>Block (B)</th>
<th>Actual (A)</th>
<th>Estimated (E)</th>
<th>Percent of Estimated (P/E)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>7200</td>
<td>7200</td>
<td>7200</td>
</tr>
<tr>
<td>Code</td>
<td>9000</td>
<td>9000</td>
<td>9000</td>
</tr>
<tr>
<td>Test</td>
<td>8000</td>
<td>8000</td>
<td>8000</td>
</tr>
<tr>
<td>Total</td>
<td>24200</td>
<td>24200</td>
<td>24200</td>
</tr>
</tbody>
</table>

Defect Recording Log, PSP Project Plan Summary were used to determine Planned, Actual, To Date and To Date% PSP process parameters during this project. In this case PSP processes results for six persons who have worked for eight weeks and their combined efforts in terms of time, defects injected and defects removed were logged per Table 2. Also, defects were identified and removed related to Code errors, Compile errors, Testing errors as detailed in the Table 3 and fixed before final delivery of software product for vehicle level systems’ integration and testing.

**POSTMORTEM**

An error was corrected caused by communication issue that was found, identified, notified and resolved during the test phase. Also, there were approximately four changes that were required in the diagnostics and interfaces to match the vehicle requirements because of new safety standards adoption by vehicle architecture after lengthy discussions with different program teams working for the same vehicle. Overall the project was integrated successfully with the result of the vehicle systems.

Table 3 – Project Result

<table>
<thead>
<tr>
<th>Project</th>
<th>Plan</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (LOC)</td>
<td>10000</td>
<td>9500</td>
</tr>
<tr>
<td>Effort (People)</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>Schedule (Wk)</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project Quality (Defect/KLOC removed in phase)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration</td>
</tr>
<tr>
<td>System Test</td>
</tr>
<tr>
<td>Field Trial</td>
</tr>
<tr>
<td>Operation</td>
</tr>
</tbody>
</table>

As shown in Table 3 the results were near to the estimated but not that encouraging while comparing with Six Sigma. Looking at these results and system performance issues, it was determined that the current design and its implementation doesn’t provide industry required reliability and quality and thus more efforts were asked to put in by management.

**4 THE MSPTS SOFTWARE PROCESS FOR EMBEDDED SYSTEMS CONTROL**

**4.1 WHAT IS NEEDED?**

The need of a process which is efficient, lean, easy to implement, less complex with minimal resource loading that could be used for any combination of small, simple, medium, moderately complex, large, and or very complex embedded hardware, software and systems control projects.

**4.2 WHY ENHANCE PSP AND TSP?**

Below are some excerpts various people sought with PSP/TSP, CMMI, Six Sigma as well as Embedded Systems Control’s basic nature of
operation and its interwoven complexity affecting reliability and safety, human errors, changing regulatory and public views of safety. As Jim McHale and Dan Wall reported in “Mapping TSP to CMMI,” while PSP/TSP cover the engineering and project management process areas generally well; they do not adequately cover all process management and support process areas of CMMI [12]. As Gary Gack reported in “Combining CMMI®, PSP, TSP, and Six Sigma for Software” although a few elements of the Six Sigma for Software toolkit are invoked within the PSP/TSP framework (e.g. regression analysis for development of estimating models), there are many other tools available in the Six Sigma for Software toolkit that are not suggested or incorporated in PSP/TSP [13]. While PSP/TSP refers to and may employ some statistical techniques, specific training in statistical thinking and methods generally is not a part of PSP/TSP, whereas that is a central feature of Six Sigma for Software [13]. Whereas Six Sigma for Software incorporates the DFSS approach to improving the feature/function/cost trade-off in definition and design of the software product, this aspect is not addressed by CMMI/PSP/TSP [13]. CMMI/PSP/TSP is among the several potential choices of software development process definition that can lead to improved software project performance. The full potential of the data produced by these processes cannot be fully leveraged without applying the more comprehensive Six Sigma for Software toolkit [13]. The relation of Six Sigma for Software to CMMI/PSP/TSP might also be characterized as a difference in goals, in which the goals of CMMI/PSP/TSP may be a subset of those associated with Six Sigma for Software [13].

4.3 ENHANCEMENT TO PSP AND TSP

The unique difference to the processes discussed in Section 3 as well as PSP and TSP is that the use of analysis tools like Risk Analysis (RA), Functional Hazard Analysis (FHA), Fault Tree Analysis (FTA) and Failure Mode Effects Analysis (FMEA) that should be started early during the requirements phase through the various stages of the software life-cycle to help determine quantitative and qualitative aspects of Functional, Availability, Reliability, Redundancy, Security, Safety, and Quality. It essentially forces engineer(s) and or team(s) of engineers, specialists and a leader to apply logical and best trade-off approach through an analysis for a given technology with good understanding of time and resources required. And hence, the attempt is made to modify previously mapped stages of PSP and TSP to Spiral Model while applying Six Sigma analysis tools such as Risk Analysis, FTA, FMEA, RA as shown in Figure 5. One must note that in some cases it may reduce the time and or improve functionality, availability, reliability, redundancy, safety, quality, and thus in turn a cumulative effect of reducing the cost in long run and improving the System Quality!

![Figure 5 – Modified Spiral Model using PSP, TSP, and Six Sigma (MSPTS)](image)

4.4 IMPROVED VERSION OF ‘MODERATELY COMPLEX AND MEDIUM’ SIZE PROJECT

REQUIREMENT

While conducting the functional hazard analysis for the requirement phase it raised few questions and surfaced flaws in understanding the requirements. The major issues were to identify and define that what kind of security if any required for data integrity and controls and if the sub-system was a safety critical system from hardware and software aspects. Next to define which module or function in the system was related to it. Obviously if this was not defined at initial stage like “Requirements” the design is not going to reflect it. In some cases the redundant measurements were part of the requirements but there were bottlenecks. In general functional and quality requirements were not changed, however because of going in details in the case of reliability, operational, redundancy, and safety requirements following changes were required in the requirements:

- Redundant Sensor Requirement
- Redundant Sense Requirement
- Interface (e.g. sensor) EMC requirements
- Interface (e.g. sensor) error diagnostics
- Two processors in separate modules
- Two process threads
- Identified software and hardware variables needed to be secured for reliable operation
- 2nd communication channel
- Discrete control signal in case of communication failure.
- Identified Safety requirements

**DESIGN**

Here attempts were made in parallel by separate group of hardware and software engineers to go through fault tree analysis and design failure mode effects analysis. While conducting this analysis for the design phase, the following were required to modify from reliability, redundancy and safety aspects to improve the overall Systems Quality:

- Effect of failure of a hardware component on functionality, reliability and safety and redesigning of critical paths.
- Dual sensors for critical sensing.
- Separate hardware measurement paths for critical signal.
- Redundant Sensing - if not possible on a different module, a 2nd channel on the same module with separate hardware path.
- At least two communication nodes with redundant third discrete node.
- Secured rationality checks on measured values
- Addition of checksum on the crucial data.
- Processor, ALU, register, memory integrity checks at power-on.
- Two processors running two threads - 2nd processor (would be better if lies in different module) in different module may not be a feasible solution because of commercial feasibility, system boundary, etc., then use of 2nd processor or core in the same module.

Here because the hardware design was not frozen most of the changes were carried out and in some cases alternative paths were chosen from the available hardware to fulfill the design requirements as shown in Figure 6. And thus attempts were made to safeguard against safety critical cases keeping the balanced trade-off between time, cost, reliability and safety. The modified design had redundant messages added

**CODE, COMPILCE AND TESTING**

Obviously, all these changes in the requirement and design did impact the coding, implementation and finally the testing of it. The

<table>
<thead>
<tr>
<th>Table 4 – PSP Project Plan Summary – (Modified)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Size (LOC)</td>
</tr>
<tr>
<td>Base (D)</td>
</tr>
<tr>
<td>Defined (D)</td>
</tr>
<tr>
<td>Modified (M)</td>
</tr>
<tr>
<td>Achieved (A)</td>
</tr>
<tr>
<td>Revised (R)</td>
</tr>
<tr>
<td>Total New &amp; Changed (N)</td>
</tr>
<tr>
<td>Total LOC (T)</td>
</tr>
</tbody>
</table>

Table 4 – PSP Project Plan Summary – (Modified) coding methods were not changed but because the design was changed, different test cases were required to design and code in addition to testing the base unchanged code. As carried out
previously, Time Recording Log, Defect Recording Log, and PSP Project Plan Summary were used to determine various PSP process parameters. In this case PSP processes were used to estimate time, defects, etc. for 4 persons for 5 weeks, while in actual the project was finished in 4.5 weeks. Also, defects were identified and removed related to code errors, compile errors, and testing errors. All these details were logged as detailed in Table 4. In addition to PSP processes, PROBE method was used to estimate size and time of the project.

It was observed that estimated LOC from PROBE method calculation was quite close to actual time compared to PSP calculation as detailed in the Table 5. The engineering judgment for time for the first phase was estimated to 153,600 minutes while in actual it had took 115,200 minutes. In 2nd phase, using estimated and actual ‘Time’, ‘Defects Injected’, and ‘Defects Removed’ differences using PSP and PROBE method for different phases within the software process.

| Table 5 – Estimating Time and Size based on PROBE method |
|----------------|----------------|
| β_P = 250 and β_T = 1.3 | Projected size P = 250 + 1.3 * 10595 = 14023 LOC |
| Total size T = 14023 + 8600 - 600 - 670 = 22693 LOC |

| Development Time = β_P * β_T |
| Development time = 2400 * 2.05 * 14023 = 729.5 Hours |
| - 18.35 weeks |

5 RESULT ANALYSIS

The effects and improvements of modified software process and analyzing results from Table 2 and Table 3 that were resulted using unchanged software process model and comparing it against the relative Table 4 and Table 5 that were resulted using MSPTS software process for ‘Moderately complex and Medium size’ project the calculated % improvement in defects/KLOC are as below.

**Before modification** - 0.0110 defects/KLOC

**After Modification** – 0.0040 defects/KLOC

110 –40/110 = 70/100 * 100 %

= 70% improvement in defects/KLOC

**Other improvements or effects:**
- Change in code required = 10032 LOC
- Additional Time required = 720 hours
- Increase in embedded systems redundancy
- Improved embedded systems safety reducing single point of failure.
- Improved design with added complexity
- Additional time required for various analysis and the result evaluation
- Increase in embedded systems availability, reliability, and quality
- Overall reduction in test phase time
- Increase in project development cost with decrease in product maintenance cost

6 CONCLUSION

During the first phase with Modified Spiral Model using PSP and TSP, The results for software errors in terms of quality were encouraging but little less than Six Sigma – This could be because of newness of PSP and TSP for the users as well as the some changes in the requirements. The PROBE Method to estimate size and time was extremely useful and seemed an easy but strong statistical approach to planning phase. The fact that Embedded Control System, which typically consists of hardware, software, and systems control leads to an understanding that Final Quality depends on the combined quality of software, hardware and system controls and their design – And thus just emphasizing on software quality the overall quality of the system cannot be achieved. As said before “when design itself is defective, properly functioning software results in poor system operation.” While using various analysis methods, it was found that some crucial requirements were not in place from the start and hence it caused ripple effects that were observed in the design, coding, implementation, and testing. Not to mention that when the hardware design is frozen it becomes almost impossible to achieve these forgotten or ambiguous requirement. For these reasons, the Spiral Model using PSP and TSP processes was further modified to ‘Modified Spiral model using PSP, TSP, and Six Sigma (MSPTS)” process to cover all aspects of software, hardware and systems control for embedded systems control. Going further into the details, here the major difference to most of the process models is that “the process approach that was taken to use various analysis tools at the early stage to define various aspects of requirements, and then subsequently during the design, coding, implementation, and testing phases”. When used these analysis tools, it surfaced the flaws in the thought process during the requirement and planning phase as well as during the design phase while addressing reliability, security, safety, and quality.

REFERENCES