SOFTWARE MEASUREMENT REPORT

Spring 2021
California State University, Fullerton
CPSC 547 - Software Measurement
Produced By:

Adam Nelson
Zulema Perez
Kenny Chao

TABLE OF CONTENTS

Introduction

“A process can be defined as the logical organization of people, materials, energy, equipment, and procedures into work activities designed to produce a specified end result.” - Gabriel A. Pall

In other words, processes define everything involved in producing work in a system. An end product can be created, but the question of “can we do this better” will always arise. We can answer this question through measuring the process. In software, this can entail the collection of data related to the work of the staff, the lines of code generated, the cost of operations, and more. In this project, we will be examining two processes: the software requirements process and the software maintenance process. In the following pages, we will outline the problems we face, the data we collect, and the investigations performed to determine if the processes are both stable and/or capable.

Process: Software Requirements

Software requirements are the descriptions of the features, needs, functionalities, and constraints of a system or subsystem to achieve an objective. There are two types of software requirements touched upon in *SWEBOK*: functional requirements and non-functional requirements.[1] Functional requirements are the set of basic needs required by users to perform work that can be seen in the final product. For example, a banking system should allow clients to retrieve information about their account(s). Non-functional requirements are the set of constraints on the system dealing with the system’s quality attributes that are not directly seen in the end product like functional requirements. These quality attributes may include modifiability, scalability, and other “ilities”. In a mobile application, for example, it would be vital to highlight and enable the mobility of the system. To develop good software, software engineers need to engineer requirements as communicated to them through use cases depicting the work breakdown structure.

The Problem

We are a team of requirement engineers at a company tasked with analyzing and improving the software requirements process to increase the efficiency of the system’s engineering. We suspect that designing better use cases depicting the work breakdown

structure for our software engineers leads to more efficient engineering that meets estimated time commitments. Simply put, we believe clearly outlining and communicating requirements to the software engineering team for them to engineer leads to less ambiguity and time spent developing the system. The correlation between use cases and time spent engineering the system can be analyzed using collected data, outlined in the next section.

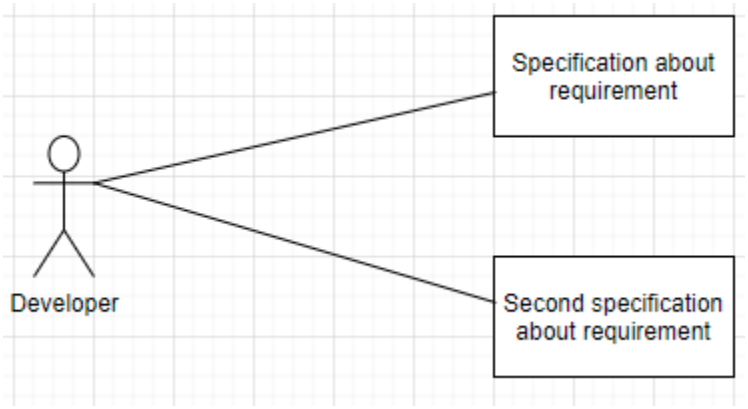


Figure 1: Use case diagram for WBS for developer.

USE CASE #	< the name is the goal as a short active verb phrase>	
Goal in Context	<a longer statement of the goal in context if needed>	
Scope & Level	<what system is being considered black box under design> <one of: Summary, Primary Task, Sub-function>	
Preconditions	<what we expect is already the state of the world>	
Success End Condition	<the state of the world upon successful completion>	
Failed End Condition	<the state of the world if goal abandoned>	
Primary, Secondary Actors	<a role name or description for the primary actor>. <other systems relied upon to accomplish use case>	
Trigger	<the action upon the system that starts the use case>	
Description	**Step**	**Action**
	1	<put here the steps of the scenario from trigger to goal delivery, and any cleanup after>
	2	<...>
	3	
Extensions	**Step**	**Branching Action**
	1a	<condition causing branching> : <action or name of sub-use case>
Sub-Variations		**Branching Action**
	1	<list of variations>

Figure 2: Use case example from https://pja.mykhi.org/mgr/blokowe/INN/sorcersoft.org/io/uml/UML-borland-UCDs.html.

Data collection

We collected the following data relating to development times:

- The number of requirements common to every use case. We were able to identify 20 common requirements.

- The estimated amount of work days allocated to engineer the system requirements as communicated through use cases of the work breakdown structure.
- The actual days spent by developers engineering the system.
- The ratio of the actual number of days spent in development and the estimated number of days.
- We were able to collect 25 subgroups of data for the 20 requirements.

	Requirement 1	Requirement 2	Requirement 3	Requirement 4
Planned 1	10	7	10	7
Actual 1	8	11	8	9
Actual/Planned 1	0.8	1.571428571	0.8	1.285714286
Planned 2	10	7	10	7
Actual 2	9	9	13	8
Actual/Planned 2	0.9	1.285714286	1.3	1.142857143
Planned 3	10	7	10	7
Actual 3	9	7	12	11
Actual/Planned 3	0.9	1	1.2	1.571428571
Planned 4	10	7	10	7
Actual 4	11	10	8	11
Actual/Planned 4	1.1	1.428571429	0.8	1.571428571

Figure 3: Full spreadsheet of data collected available for viewing at https://docs.google.com/spreadsheets/d/11K2-xNQZLmyRvTjHYXQQ2V7XSyoN3TrpyLEtgL3jThU/edit?usp=sharing

Sources of data collection

Information from the following sources were used to get an idea of how long development takes in real world situations:

https://docs.oracle.com/en/cloud/saas/project-management/21a/oapem/project-performance.html#OAPEM1122084
Project Performance (Chapter 8) 21A

https://soltech.net/how-long-does-it-take-to-build-custom-software/

https://explore.easyprojects.net/blog/project-management-101-why-use-estimated-hours

Using this data, we created a python script to generate a .csv with randomly generated data that conforms to real life situations:

```
import csv
import random as rnd

def create_spreadsheet():
    with open('cpsc547_data.csv', mode='w', newline="") as sheet:
        sheet_writer = csv.writer(sheet, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
        requirements = ['']
        planned_rnd = ['7', '10', '14', '21', '30']
        for i in range(20):
            requirements.append("Requirement " + str(i + 1))
        sheet_writer.writerow(requirements)
        #Store a value randomly chosen from our planned_rnd to append to temp_planned.
        temp_planned = []
        for x in range(20):
                x = rnd.randrange(0,5)
                planned_rnd_select = planned_rnd[x]
                temp_planned.append(str(planned_rnd_select))

        #Create actual data based off planned days by randomly generating data
        # with a floor of 2 days before planned_num or twice the value of planned_num.
        for i in range(25):
            temp_actual = []
            temp_rate = []
            for x in range(20):
                planned_num = temp_planned[x]
                actual_rnd_select = rnd.randrange(int(planned_num) - 2, int(planned_num) * 2)
                rate = int(actual_rnd_select)/int(planned_num)
                temp_actual.append(str(actual_rnd_select))
                temp_rate.append(str(rate))
            planned = ["Planned " + str(i +1)] + temp_planned
            actual = ["Actual " + str(i +1)] + temp_actual
            rate = ["Actual/Planned " + str(i +1)] + temp_rate

            sheet_writer.writerow(planned)
            sheet_writer.writerow(actual)
            sheet_writer.writerow(rate)

create_spreadsheet()
```

Figure 4: Script to create random data.

Reviewing and Assessing Collected Data

The data has been collected. Now it is time to check whether the data is credible. In order to be credible, data must possess the following attributes as listed in *Florac*:[2]

- **Verity**
  - Verified data is data that has been collected according to specifications and contains no errors. In other words the data is of the correct type, format, ranges, are complete, and are correct. In our data, we want to make sure the time collected is in the days format, is rounded up to include the whole day, and is numeric. There is a lot of variance in our decimal values, so we round all values to the nearest hundredth to ensure verity.

	Requirement 1	Requirement 2	Requirement 3	Requirement 4
Planned 1	10	30	10	21
Actual 1	18	31	19	34
Actual/Planned 1	1.80	1.03	1.90	1.62
Planned 2	10	30	10	21
Actual 2	17	48	17	35
Actual/Planned 2	1.70	1.60	1.70	1.67
Planned 3	10	30	10	21
Actual 3	17	30	15	26
Actual/Planned 3	1.70	1.00	1.50	1.24
Planned 4	10	30	10	21
Actual 4	16	54	11	23
Actual/Planned 4	1.60	1.80	1.10	1.10

Figure 5: Table data with numbers rounded.

- **Synchronicity**
  - Synchronous data is data that is related to their time of occurrence. After verifying our data again, we notice our data was not synchronous as the actual days of work for requirement 6 and 7 were swapped. We make this correction to ensure the synchronicity of the data.

- **Consistency**
  - Consistent data is data that is within the realm of possibility and leads to data with outlandish and improbable values to be investigated and compared to known sources. We check our data and notice an anomaly concerning the actual amount of work listed in actual work for requirement

8 seems impossibly, extremely inconsistent; a release with an expected development time of 7 days should not have taken 63 days to finish. Upon re-examination we find the data was incorrectly entered as 9 weeks instead of 9 days. We correct the data to ensure consistency.

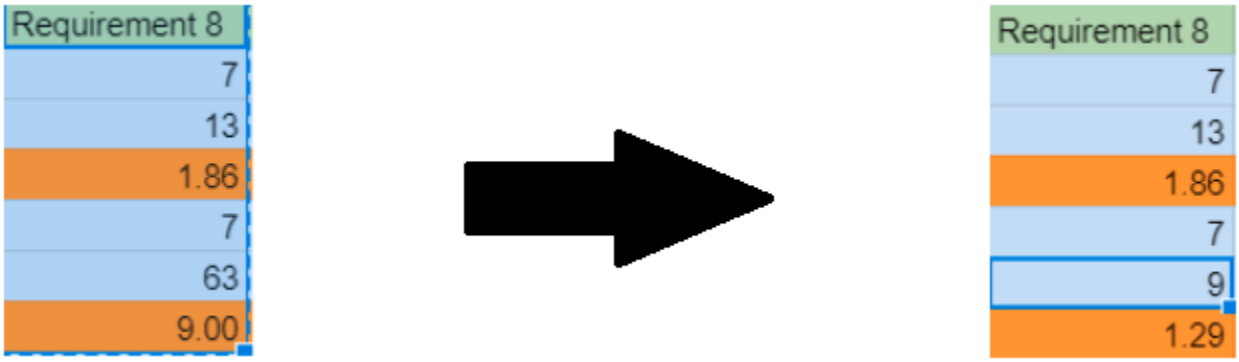


Figure 6: Correction of data to ensure consistency.

- **Validity**
  - Valid data is data that is within values that truly describe the attribute of interest. Upon examination of our data with all fixes performed, we find our data to be valid and credible.

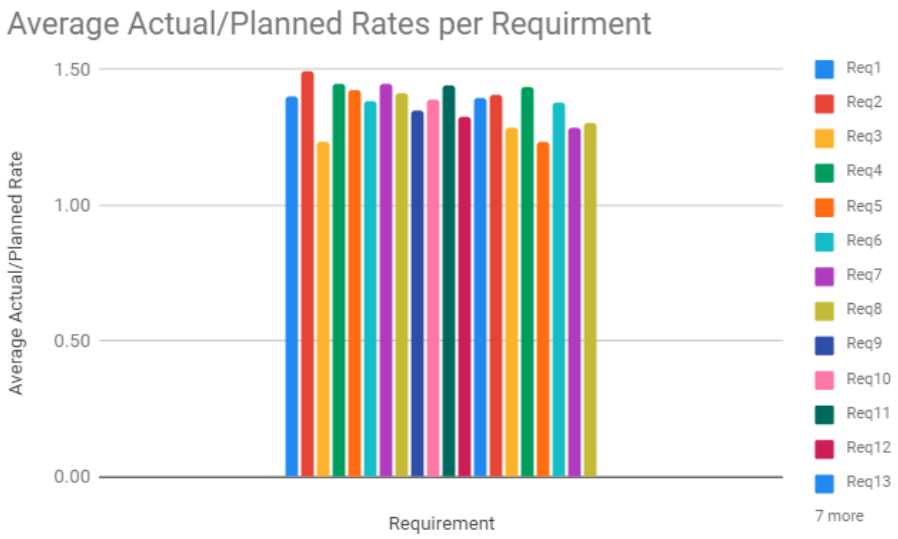


Figure 7: Bar Chart of Average Actual/Planned Rates per Requirement.

Ascertain whether the process is in control

We ascertain whether the process is in control by determining if our data is stable. To do so, we need to decide how to analyze the data. For each of the 20 requirements, we have 25 measurements of actual vs planned data. With 25 subgroups, we can collect the average and standard deviation of the actual vs planned rate for each of our requirements and compare them. Since we are collecting variable data and our subgroup size is larger than 10, we use an $\bar{\mathbf{X}}$ and **S** chart to check for process stability.

The Stability Investigation Process

1. For the $\bar{\mathbf{X}}$ and **S** chart, we need to collect the average actual/planned rate for each requirement. Each requirement is made of 25 subgroups of actual/planned rates, so we average these values for each requirement, then obtain the average of these. The same applies to collecting the standard deviations. Below we have a screenshot of a small subset of our data.

Subgroups Size	25	25	25	25	25
Average	1.40	1.49	1.23	1.45	1.42
Standard Deviation	0.35	0.33	0.35	0.29	0.34

Figure 8: Small set of averages and standard deviations. Full set viewable at https://docs.google.com/spreadsheets/d/11K2-xNQZLmyRvTjHYXQQ2V7XSyoN3TrpyLEtgL3jThU/edit?usp=sharing

2. Now we need to calculate the mean of our averages, the mean of our standard deviations, and our subgroup size using the calculations listed:
   a. Average mean = 1.37

$$\bar{\bar{X}} = \frac{\bar{X}_1 + \bar{X}_2 + \ldots + \bar{X}_k}{k}$$

   b. Average standard deviation = 0.34

$$\bar{S} = \frac{\sum_{i=1}^{k} S_i}{k}$$

   c. Subgroup size = 25

3. Next we look up values from the bias correction table with a subgroup size of 25:
    a. A_3 = 0.606
    b. B_3 = 0.565
    c. B_4 = 1.490

4. Using this collected data, we can make calculations to obtain our control limits with the listed formulas:
    a. $UCL_{\bar{\bar{X}}}$ = 1.58

$$UCL_{\bar{X}} = \bar{\bar{X}} + A_3\bar{S}$$

    b. $CL_{\bar{\bar{X}}}$ = 1.37

$$CL_{\bar{X}} = \bar{\bar{X}}$$

    c. $LCL_{\bar{\bar{X}}}$ = 1.17

$$LCL_{\bar{X}} = \bar{\bar{X}} - A_3\bar{S}$$

    d. $\sigma_{\bar{\bar{X}}}$ = 0.07

$$\sigma_{\bar{X}} = \frac{A_3\bar{S}}{3}$$

    e. UCL(**S** chart) = 0.49

$$UCL_{\bar{S}} = B_4\bar{S}$$

    f. CL(**S** chart) = 0.34

$$CL_{\bar{S}} = \bar{S}$$

    g. LCL (**S** chart) = 0.19

$$LCL_{\bar{S}} = B_3\bar{S}$$

With our data collected and calculations made, we can now plot our $\bar{\bar{X}}$ and **S** charts:

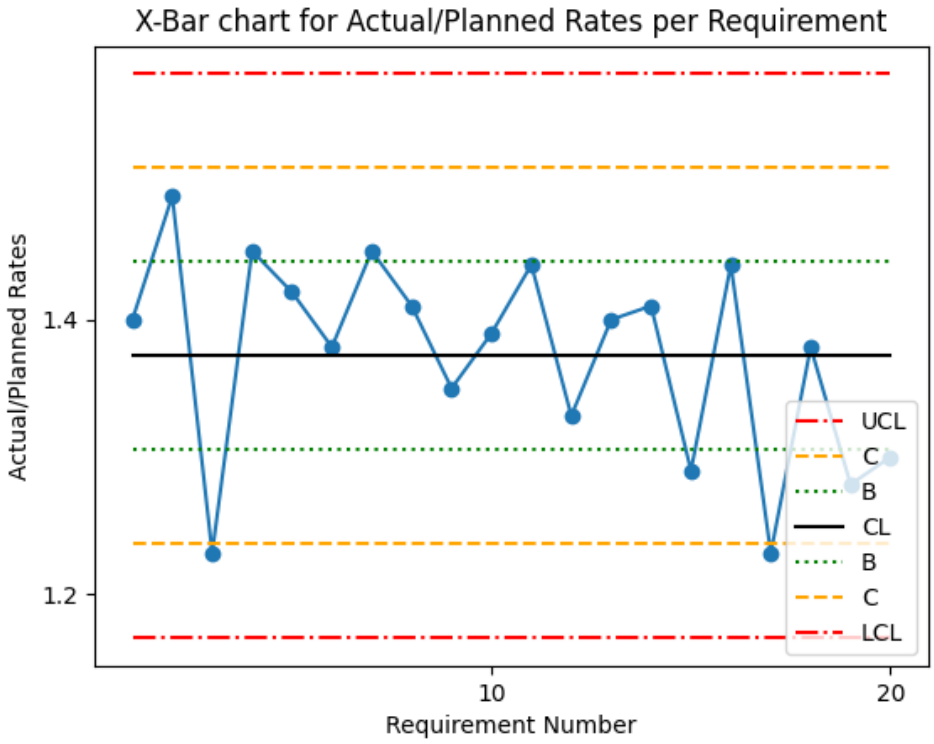


Figure 9: $\bar{X}$ chart for Actual/Planned Rates per Requirement.

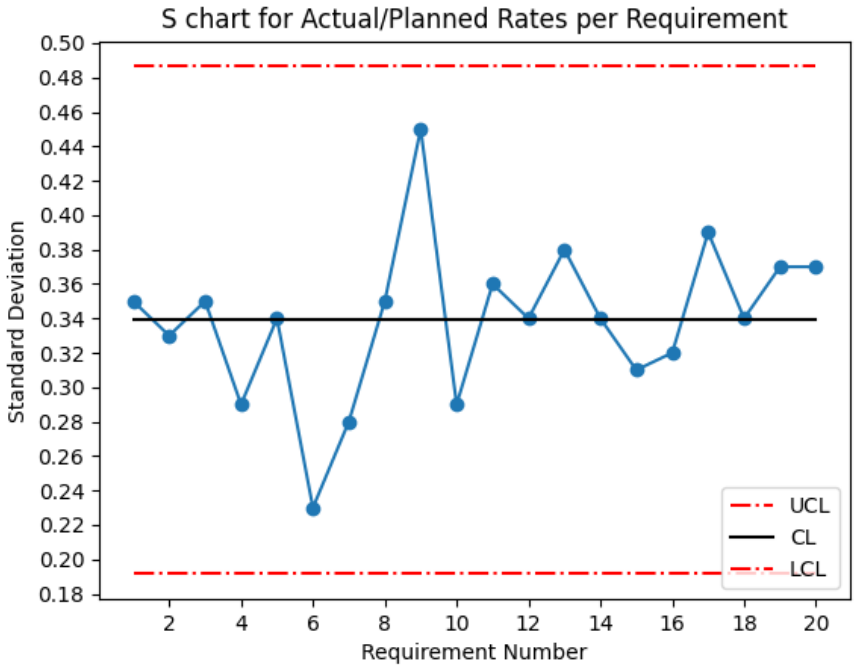


*Figure 10: **S** chart for Actual/Planned Rates per Requirement.*

Test for Stability

Now we can perform tests on our charts to see if the process is stable. The S chart only needs to be checked for the first test:

1. **A single points falls out of the UCL or LCL**
    a. **S chart NONE → ✅**
    b. **$\bar{X}$ chart NONE → ✅**
2. **At least two out of three successive values fall on the same side of the CL, and in Zone C. NONE → ✅**
3. **At least four out of five successive values fall on the same side of the CL, and in Zone B. NONE → ✅**
4. **At least eight successive values fall on the same side of the centerline. NONE → ✅**

According to the tests, our process is stable and in control! ✅

Process Capability

The next step is to check if our process is capable. Per *Florac*, the capability of a process refers to the predictable performance of a process under statistical control.[2] In other words, if we can determine a process is in statistical control, we can predict how the process will perform in the future. A process is capable when all measurements fall within their natural process limits. To be capable, our process needs to meet two criteria:

1. The process must be in statistical control.
2. The capability of the process must meet business and/or customer requirements.

First we can look at the Voice of the Process (VOP) to see if it appears that our process falls within control limits.

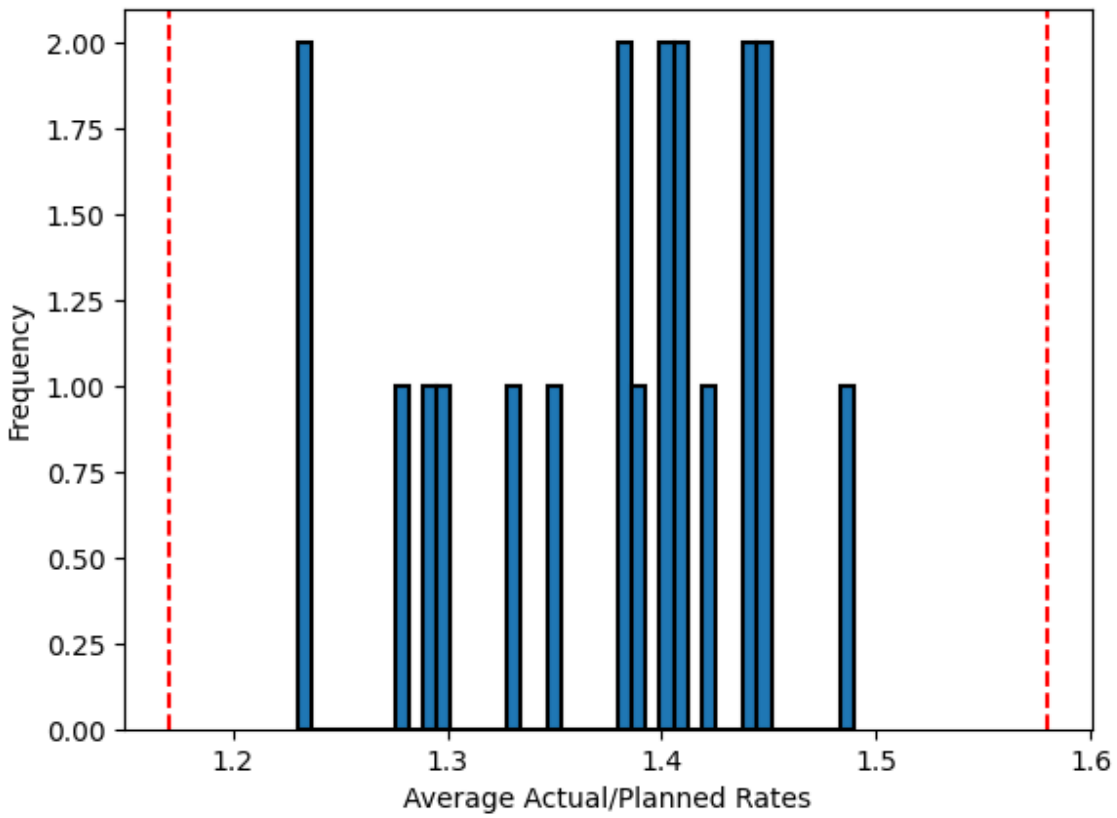


Figure 11: Histogram for the voice of the process.

From the above histogram, it appears our process may be in stable statistical control. All of our measurements fall within the natural process limits so we can move on to more testing.

Before beginning our calculations, we can visually determine if the process appears to be capable by plotting our same histogram, but with our upper and lower specification limits included. Suppose our company initially decides that they want the average work rates to fall within 1.3 and 1.4. Given the lower specification limit and upper specification limit, we can plot our voice of the customer to see if our process is capable.

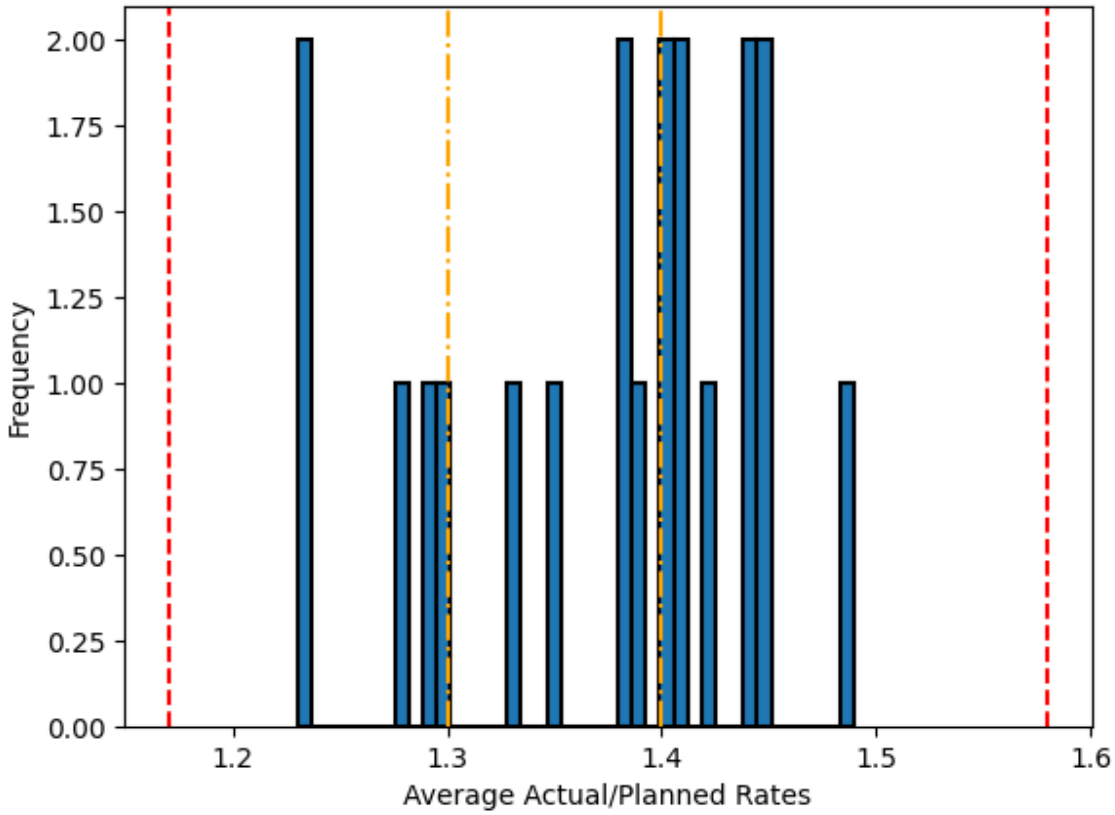


Figure 12: Histogram for the voice of the customer.

Here we can clearly see that our process is NOT meeting the business requirements, and thus the process appears to be NOT capable. ❎

Armed with this information we can verify the process is not capable by examining the specification tolerance and capability indices.

Specification Tolerance and Distance to the Nearest Specification

The specification tolerance (ST) of a process measures if the distance between the specification is wide enough for the natural process variation of a stable process. The distance to the nearest specification (DNS) measures whether the process is sufficiently centered given the specification limits.[3] Given the following data we can calculate our specification tolerance with the listed formula and values from the bias correction table for a subgroup of 25:

USL = 1.4
LSL = 1.3
D_2 = 3.931
R_bar = 0.097

$$ST = USL - LSL$$

ST = 0.1

$$ST_{\sigma_X} = \frac{ST}{\left(\frac{\bar{R}}{d_2}\right)} = ST\left(\frac{d_2}{\bar{R}}\right)$$

ST$\sigma_{\bar{x}}$ = 4.06
ST$\sigma_{\bar{x}}$ < 6

$$Z_u = \frac{USL - \bar{\bar{X}}}{\left(\frac{\bar{R}}{d_2}\right)} = \left(USL - \bar{\bar{X}}\right)\left(\frac{d_2}{\bar{R}}\right)$$

Z_u = 1.048

$$Z_{\text{L}} = \frac{\bar{\bar{X}} - LSL}{\left(\frac{\bar{R}}{d_2}\right)} = \left(\bar{\bar{X}} - LSL\right)\left(\frac{d_2}{\bar{R}}\right)$$

Z_L = 2.84

Here we see our specification tolerance in sigma units is less than 6. This tells us the width of our specification tolerance is not wide enough to contain the natural process variation of our process. Also, the Z_u and Z_L are less than 3. This tells us we have an issue with both our upper and lower control limits. From this we can say that the natural process variation of our process does NOT fall within specification limits. X

Capability Indices

There are two capability indices we can use to help measure the capability of a process. The first, C_P, is the capability ratio, and it measures the ratio of the specification tolerance of the process to six standard deviation units. The second, C_{Pk}, measures the ratio of the distance to the nearest specification to three standard deviation units; with just the C_{Pk} we can determine the process' capability. Given the formulas below we can calculate our capability indices with the following data and our Z_u and Z_L from above:

$$C_p = \frac{USL - LSL}{6\sigma_x}$$

$C_P = 0.238$

$$C_{pk} = \frac{\min(Z_U, Z_L)}{3}$$

$C_{Pk} = 0.349$

With our calculations we can compare our data to the following specifications to determine the capability of the process:

C_p is analyzed like the following:
- $C_p < 1$, the process is not capable at all
- $1 < C_p < 1.3$, the process maybe barely capable, based on centering
- $1.3 < C_p < 2$, the process could be very capable, based on centering

C_{pk} is analyzed like the following:
- $C_{pk} < 1$, the process is not capable at all
- $1 < C_{pk} < 2$, the process is maybe barely capable, but there is a pretty good chance of unpredictable results
- $C_{pk} > 3$, the process is very capable

Figure 13: Values to dictate capability based on Cp and Cpk taken from Week 11 lecture by Arash Sharif.

Here we see our C_P and C_{Pk} are under 1. C_P tells us our specification width is NOT capable, and C_{Pk} tells us our process is not capable to meet the specification. As we expected, our process is now confirmed to be NOT capable. ❌

Make Capable

We make some adjustments to our process to try to make it capable. Our team rallies the company to expand their specification limits. Given the following data, we plot our new VOC histogram:
USL = 1.6
LSL = 1.1

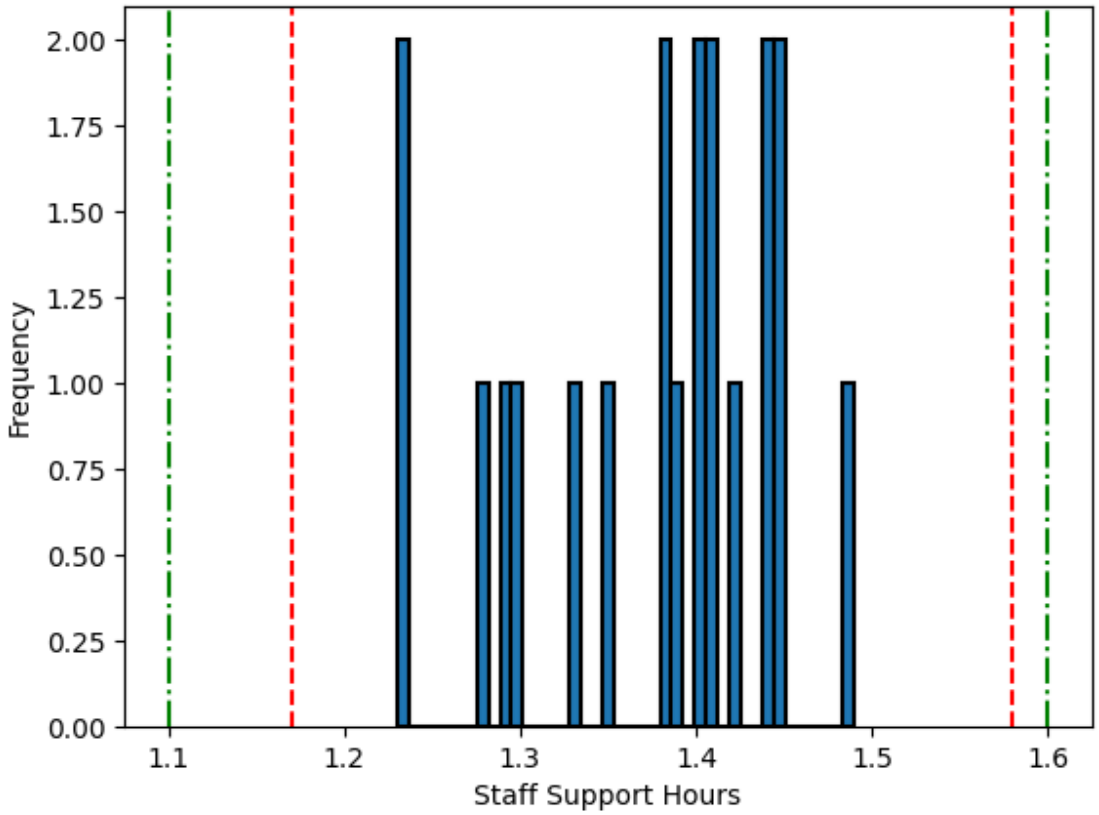


Figure 13: Histogram of the voice of the customer.

Our process appears to fall within specification limits, but we will need to make some calculations to verify.

Specification Tolerance and Distance to the Nearest Specification

We use the same formulas but with our new specification limits to calculate the values below:
ST = 0.5
ST$\sigma_{\bar{x}}$ = 20.30
Z_u = 9.43
Z_L = 10.96

Here our specification tolerance is greater than 6, and our Z_L and Z_U are both over 3. This tells us our specification limits are wide enough to contain the natural process variations and that the process' specification is centered. Our process falls within specification limits.✅

Capability Indices

Again, we check our capability indices with our updated specification limits:

$C_P = 1.19$
$C_{Pk} = 3.14$

From the calculated capability indices, C_P tells us the process is likely capable. The C_{Pk} value tells us the process is very capable. Therefore, we can confirm that our process has indeed been made capable. Although not initially, we have changed and measured our process to confirm it to be stable and capable. ✅

Conclusion

The software requirements process is vital to help teams meet business and customer requirements. Designing clear and unambiguous use cases for work breakdown structures helps lower ambiguity among engineers and helps engineers meet estimated work time frames established by the company. We initially showed that our process was stable but incapable. By widening the specification limits imposed by the business, we were able to make our process capable. In future iterations of our process, we would like to further improve the quality of our use cases of work breakdown structures and measure new data gathered regarding Actual/Planned work days per requirement to see if our process can remain stable and capable while meeting more stringent specification limits.

Process: Software Maintenance

Every software has a life cycle and that life cycle can be divided into two major parts: the first one is the software development part and second is the software maintenance part. The software maintenance part consists of user services and software system responsibilities. With priorities being shifted around and work requests of software system corrections often taking precedence over other work already in progress.

There are five major features that software maintenance focuses on which are:

- Maintaining control of software systems daily functions.
- Maintaining control of software systems modifications.
- Perfecting already existing accepted functions.
- Preventing degradation of software system performance to unacceptable levels.
- Identifying security threats and fixing vulnerabilities.

It is important to understand the range of maintenance activities and the factors in which the software maintenance work is done on a daily basis. There are many parts to the software maintenance process. In the figure 1A below, borrowed from the SWEBOK Guide V3.0[3] under the Software Maintenance process, we can see just how large the software maintenance process is.

With the software maintenance process being the last stage of the software life cycle it is also the longest stage. Due to these reasons the cost for software maintenance is high and is largely impacted by the following factors: hardware and software compatibility, policies, competitors,other processes, other products, and personnel.

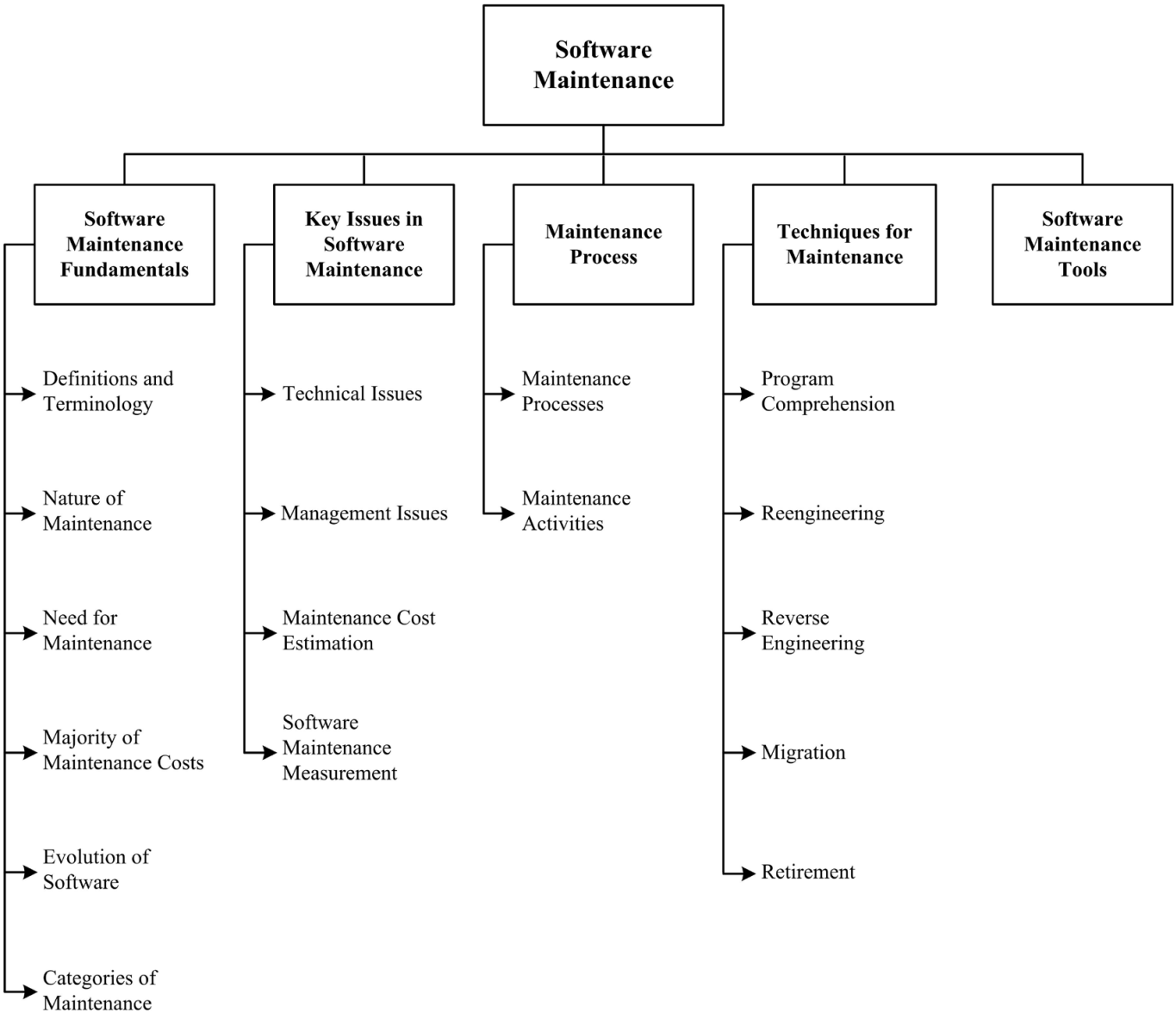


Figure 1A. Is Figure 5.1 Breakdown of topics for the Software maintenance KA
From SWEBOK Guide V3.0

Problem we are trying to solve

We are a team of software maintenance engineers that are responsible for the software maintenance at our software company. Our company is a small start up that has been steadily acquiring clients for over a year. Recently our company has acquired, under contract, two more clients. These two clients are larger companies and have more employees, which will actually double the customer base we will have to provide software maintenance and support services to. Due to the fact that we are a small start up company, our software maintenance team provides support for all aspects of the software maintenance process. Because the workload for the software maintenance team is increasing we believe that additional software maintenance engineers will need to be hired in order to have a consistent and under control maintenance process at our company. In order to justify the need for additional software maintenance engineers and to understand how many additional software maintenance engineers will need to be hired, we will analyze data collected over the last four months.

Data: The details

We have received some data regarding the software maintenance team:

- ❏ Staff hours per day for the last sixteen weeks.
- ❏ Software maintenance support tickets per day for all staff.
- ❏ We are rationally subgrouping the data by the number of days in each week.
- ❏ The data was manufactured using the textbook Measuring the Software Process[4] as a reference.

Staff Hours Per Day					
Week	Monday	Tuesday	Wednesday	Thursday	Friday
1	70.6	69.9	69.9	69.9	64.6
2	68.9	66.8	65.8	64.8	60.6
3	69.9	70.1	69.9	69.6	70.1
4	66.6	65.6	61.6	67.7	64.8
5	69.8	65.5	70.6	68.8	69.7
6	68.8	67.6	67.7	64.6	60.7
7	69.4	64.8	63.7	60.4	67.9
8	70.1	69.7	70.1	69.6	69.4

Figure 2A: snapshot of staff hours per day dataset

Reviewing and Assessing Collected Data

Now that the data has been collected, we need to check the credibility of the data. The criteria for data credibility is:

1. Verity
2. Synchronicity
3. Consistency
4. Validity

First we check if the data is the correct type and format. We also want to make sure that the data is within the specified ranges, is arithmetically correct and that it is complete. Below are snapshots of our datasets of staff hours per day in Figure 2A and software maintenance support tickets per day in Figure 3A. For each dataset the data is the correct type and format. The data is also within the specific range of hours and tickets expected based upon the knowledge of the long hours the software maintenance team puts in each day. The data is arithmetically correct and complete based off of what the software maintenance team had previously reported. Following the criteria for data credibility we find that the data has been collected according to specifications and contain no errors.

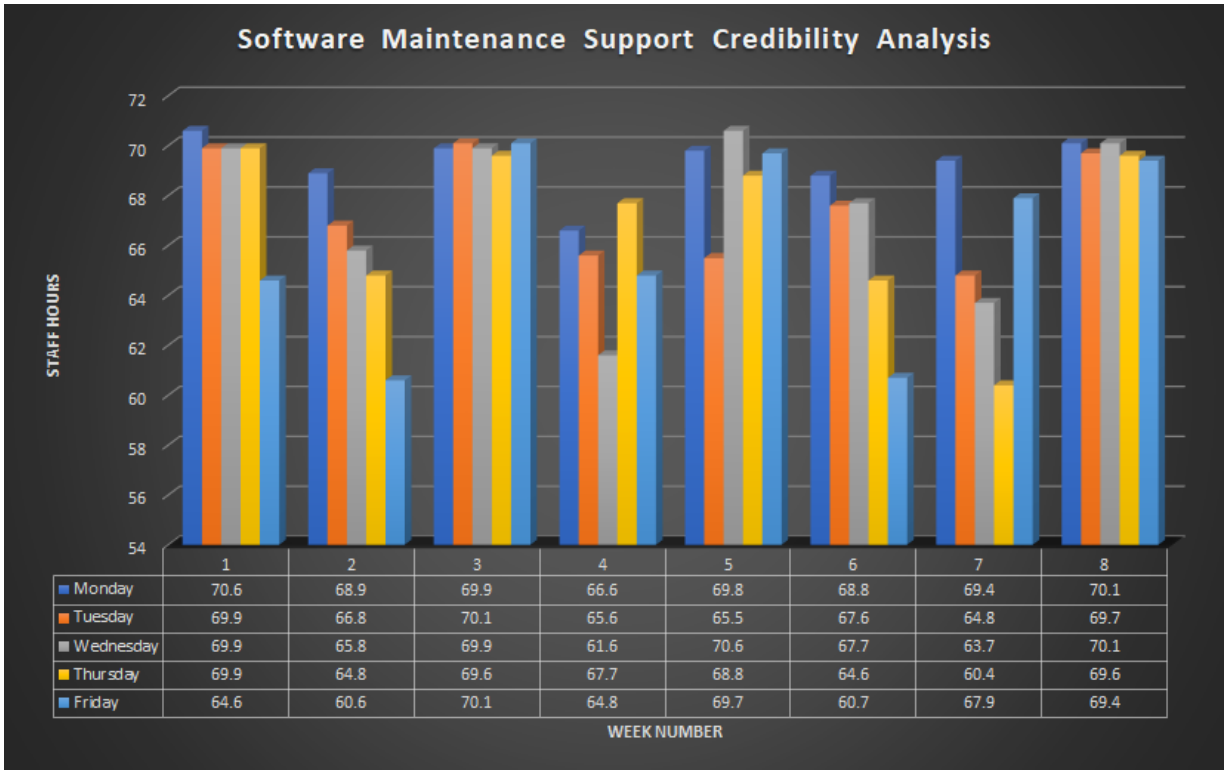
Staff Hours Per Day					
Week	Monday	Tuesday	Wednesday	Thursday	Friday
1	70.6	69.9	69.9	69.9	64.6
2	68.9	66.8	65.8	64.8	60.6
3	69.9	70.1	69.9	69.6	70.1
4	66.6	65.6	61.6	67.7	64.8
5	69.8	65.5	70.6	68.8	69.7
6	68.8	67.6	67.7	64.6	60.7
7	69.4	64.8	63.7	60.4	67.9
8	70.1	69.7	70.1	69.6	69.4

Figure 3A. snapshot of staff hours per day.

Software Maintenance Support Tickets Per Day					
Week	Monday	Tuesday	Wednesday	Thursday	Friday
1	141	140	140	140	129
2	138	134	132	130	121
3	140	140	140	140	140
4	133	131	123	136	130
5	140	131	141	138	140
6	138	135	135	129	121
7	140	130	127	121	136
8	140	140	140	140	139

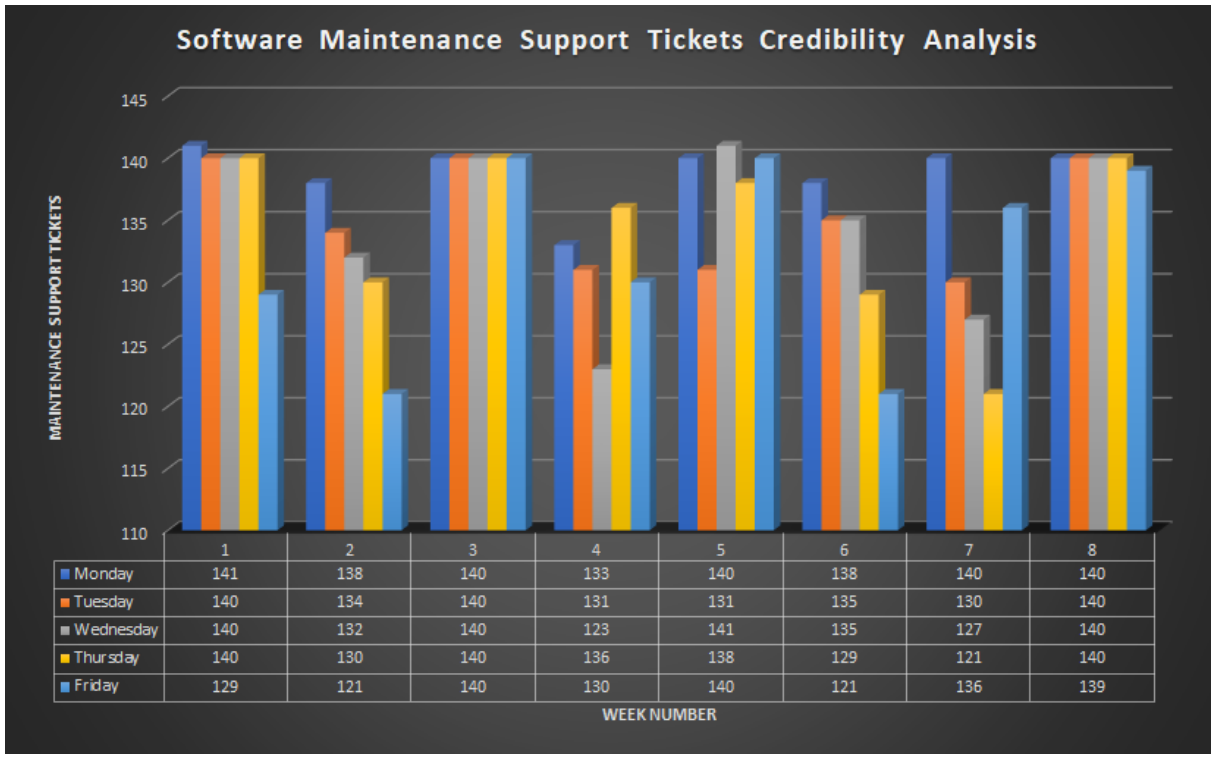
Figure 4A. snapshot of maintenance support tickets per day.

To further solidify our understanding of the data, we require visualization of the data. We use bar charts in our data analysis to show how the datasets relate to each other, which provides credibility and consistency of the data.



	1	2	3	4	5	6	7	8
Monday	70.6	68.9	69.9	66.6	69.8	68.8	69.4	70.1
Tuesday	69.9	66.8	70.1	65.6	65.5	67.6	64.8	69.7
Wednesday	69.9	65.8	69.9	61.6	70.6	67.7	63.7	70.1
Thursday	69.9	64.8	69.6	67.7	68.8	64.6	60.4	69.6
Friday	64.6	60.6	70.1	64.8	69.7	60.7	67.9	69.4

Figure 5A. Credibility Analysis of staff hours



	1	2	3	4	5	6	7	8
Monday	141	138	140	133	140	138	140	140
Tuesday	140	134	140	131	131	135	130	140
Wednesday	140	132	140	123	141	135	127	140
Thursday	140	130	140	136	138	129	121	140
Friday	129	121	140	130	140	121	136	139

Figure 6A. Credibility Analysis of maintenance support tickets

The above graphs show that there is a strong correlation between the data staff hours per day and maintenance support tickets per day. We can see that the amount of staff hours worked is driven by the amount of tickets being addressed per day. The data analysis verifies the consistency and the credibility of the data and now can be used for stability and capability analysis.

We now use the staff hours per day data and we divide the staff hours per day by the total number of the current software maintenance engineers on the software maintenance team. Currently our software maintenance team consists of 7 software maintenance engineers. Below is the Figure 6A which provides a snapshot of the data. We do this in order to calculate the average number of hours each software maintenance engineer works per day of the week.

Staff Hours Per Day / Number of Staff = 7					
Week	Monday	Tuesday	Wednesday	Thursday	Friday
1	10.09	9.99	9.99	9.99	9.23
2	9.84	9.54	9.40	9.26	8.66
3	9.99	10.01	9.99	9.94	10.01
4	9.51	9.37	8.80	9.67	9.26
5	9.97	9.36	10.09	9.83	9.96
6	9.83	9.66	9.67	9.23	8.67
7	9.91	9.26	9.10	8.63	9.70
8	10.01	9.96	10.01	9.94	9.91

Figure 7A. Staff hours per day / number of staff = 7

After examining the data with corrections and modifications made we conclude that our data is valid and credible.

Ascertain whether the process is in control

Now we have data for each of the 16 weeks, which is 5 days of the average hours worked per day. With the subgroup size of 5, we can collect the average and range for each week of the average hours worked per day. Since the subgroup size is 5, we can use the X bar and R chart to check for stability.

The Stability Investigation Process

For the X bar and R chart, we need to compute the average and range for the daily staff hours/number of staff for each week. For our data the subgroup size is 5.

$$\bar{X} = \frac{X_1 + X_2 + ... + X_n}{n}$$

$$R_k = |X_{max} - X_{min}|$$

Staff Hours Per Day / Number of Staff = 7									
Week	Monday	Tuesday	Wednesday	Thursday	Friday	Average	Min	Max	Range
1	10.09	9.99	9.99	9.99	9.23	9.85	9.23	10.09	0.86
2	9.84	9.54	9.40	9.26	8.66	9.34	8.66	9.84	1.19
3	9.99	10.01	9.99	9.94	10.01	9.99	9.94	10.01	0.07
4	9.51	9.37	8.80	9.67	9.26	9.32	8.80	9.67	0.87
5	9.97	9.36	10.09	9.83	9.96	9.84	9.36	10.09	0.73
6	9.83	9.66	9.67	9.23	8.67	9.41	8.67	9.83	1.16
7	9.91	9.26	9.10	8.63	9.70	9.32	8.63	9.91	1.29
8	10.01	9.96	10.01	9.94	9.91	9.97	9.91	10.01	0.10

Figure 8A. snapshot of averages and ranges.

We then compute the grand average and the average range.

- The equation for grand average by averaging each of the k subgroups, which for this case is the number of weeks:

$$\bar{\bar{X}} = \frac{\bar{X}_1 + \bar{X}_2 + ... + \bar{X}_k}{k}$$

$\bar{\bar{X}}$ = 9.61

- The equation for the average range by averaging each of the k weeks:

$$\bar{R} = \frac{R_1 + R_2 + ... + R_k}{k}$$

$\overline{R}$ = 0.86

- Bias correction data for X bar and R chart, will be used to calculate UCL, CL and LCL:

  n = 5
  A_2 = 0.577
  D_3 = 0 (Undefined)
  D_4 = 2.114

- Equations for X-Bar Chart:

  - UCL = Upper Control Limit
  - CL = Control Limit
  - LCL = Lower Control Limit

$$UCL_{\bar{X}} = \bar{\bar{X}} + A_2\bar{R}$$

$$CL_{\bar{X}} = \bar{\bar{X}}$$

$$LCL_{\bar{X}} = \bar{\bar{X}} - A_2\bar{R}$$

$UCL_{\bar{X}}$ = 9.61 + 0.577(0.86) = 10.11109643
$CL_{\bar{X}}$ = 9.61
$LCL_{\bar{X}}$ = 9.61 - 0.577(0.86) = 9.11

- Equations for R Chart:

$$UCL_{\bar{R}} = D_4\bar{R}$$

$$CL_{\bar{R}} = \bar{R}$$

$$LCL_{\bar{R}} = D_3\bar{R}$$

$UCL_{\bar{R}}$ = 2.114(0.86) = 1.81955
$CL_{\bar{R}}$ = 0.86
$LCL_{\bar{R}}$ = 0

- Equation for Sigma for X-Bar Chart Zones.

$$\sigma_{\bar{X}} = \frac{A_2\bar{R}}{3}$$

$\boldsymbol{\sigma}_{\bar{X}}$ = $0.577(0.86) \div 3$ = 0.165544048

Now we plot the measurement data on our X-Bar and R charts.
X-Bar Chart:

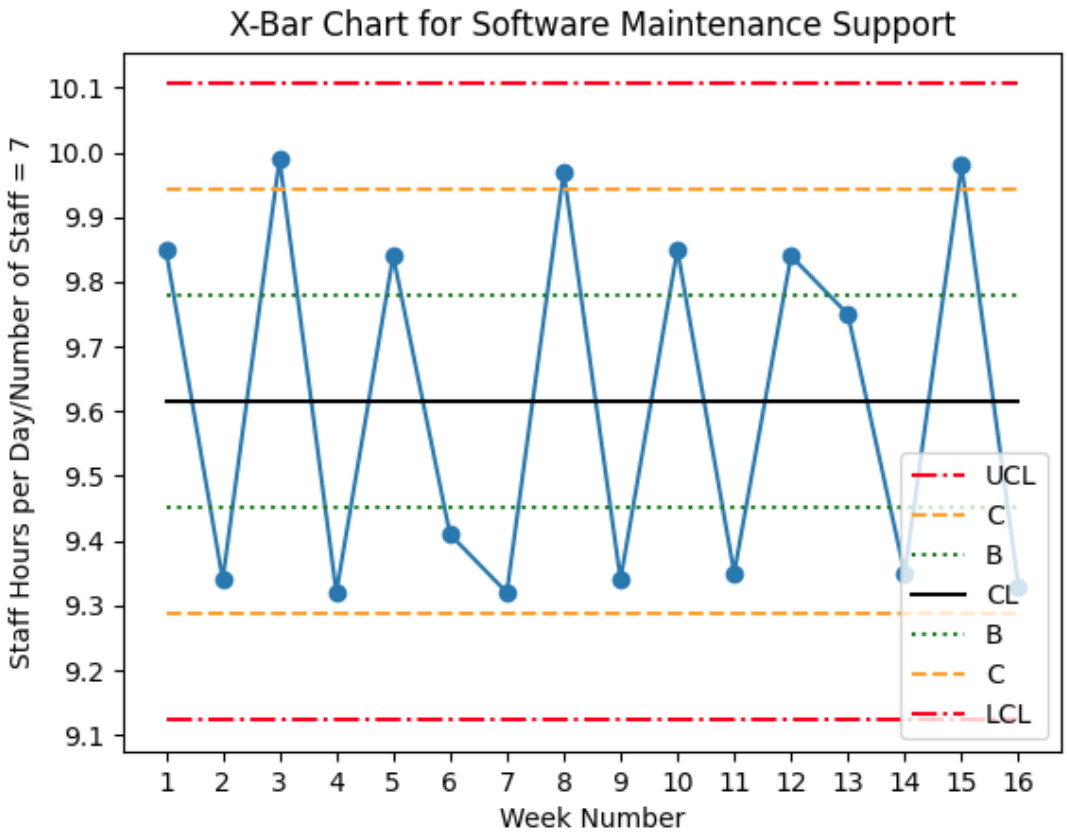


Figure 9A. X-Bar Chart for software maintenance support

R Chart:

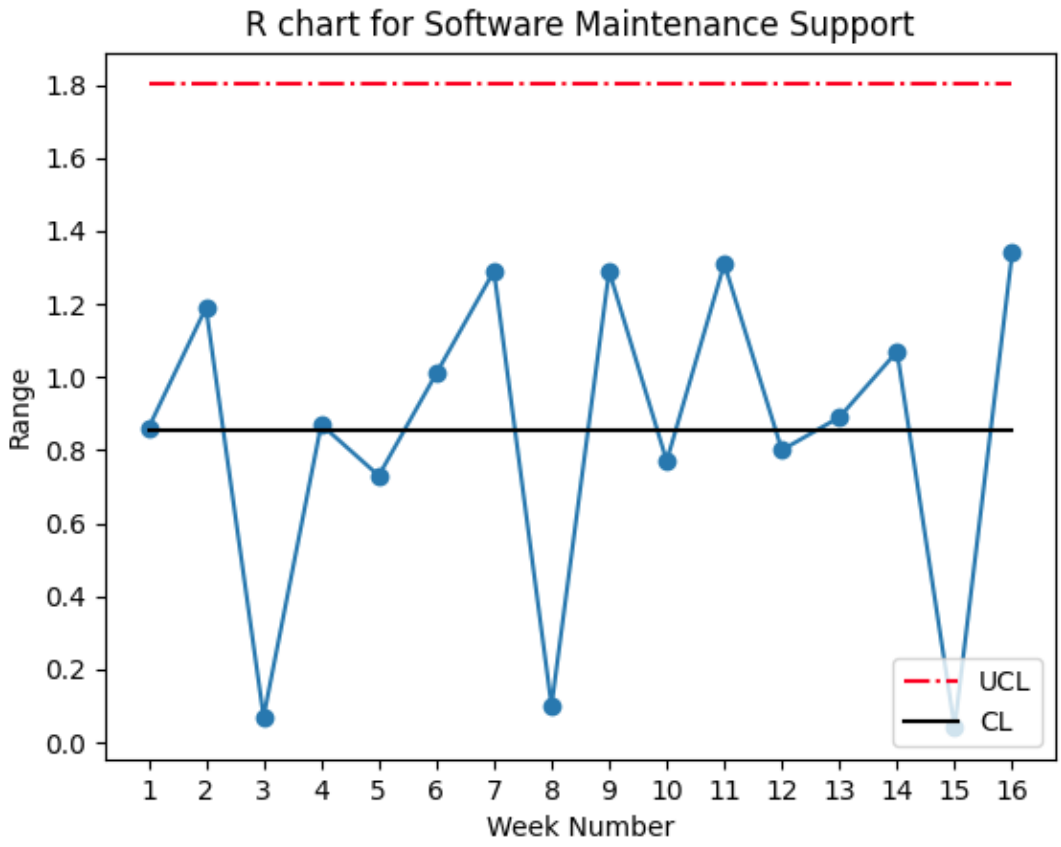


Figure 10A. R Chart for software maintenance support

Test for Stability

Now we can analyze our control charts to see if the process is stable. The R chart needs to be checked first. Only if the R chart passes do we continue to the X-bar Chart. Test 1 applies to both X-Bar and R charts, tests 2 - 4 apply only to the X-Bar chart:

1. **A single points falls out of the UCL or LCL**
    a. **R chart NONE → ✅**
    b. **$\bar{X}$ chart NONE → ✅**
2. **At least two out of three successive values fall on the same side of the CL, and in Zone C. NONE → ✅**
3. **At least four out of five successive values fall on the same side of the CL, and in Zone B. NONE → ✅**
4. **At least eight successive values fall on the same side of the centerline. NONE → ✅**

According to the tests, our process is stable and in control!✅

Process Capability Analysis

Now that we have verified that our process is stable and in control we can perform the capability analysis for our process. The capability of a process is when the predictable performance of a process is under statistical control. Which means that if the process is determined to be under statistical control, we can then predict the future performance of the process. A process is capable when it continues to remain under statistical control, all measurements fall within their natural process limits, and the capability of the process meets business and/or customer requirements.
We use the UCL, CL and LCL that we previously calculated and plot the values onto a histogram for capability analysis.

First take a look at the Voice of the Process (VOP) and see if the process falls within the natural process limits.

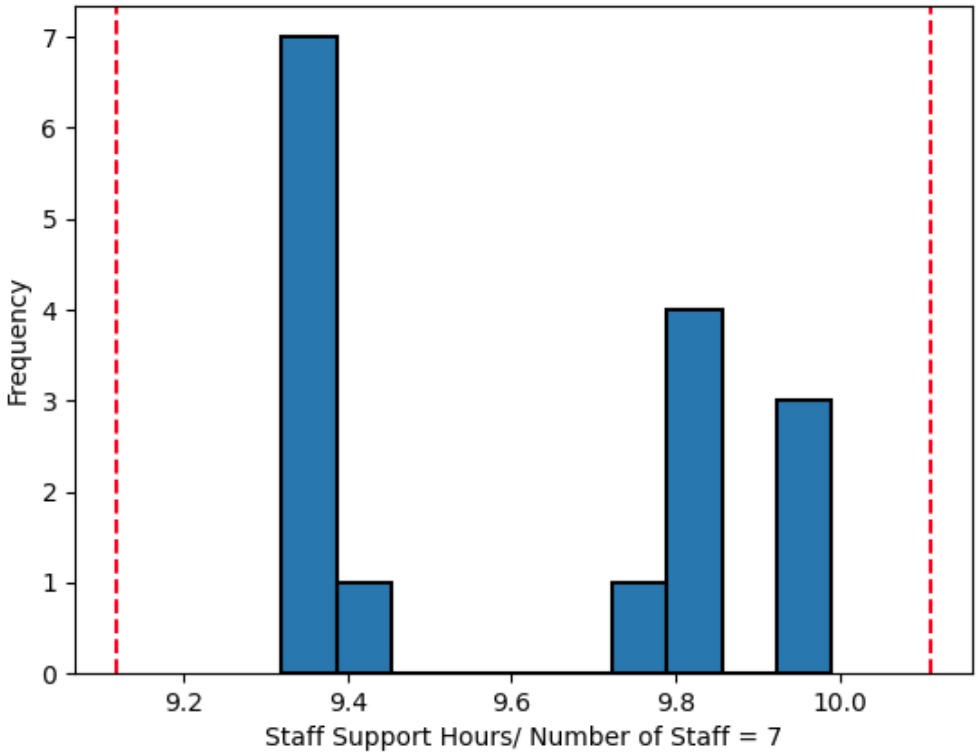


Figure 11A. Histogram for the voice of the process.

According to the histogram above, the process appears to be under stable statistical control and all of the measurements fall within the natural process limits.
To determine if our process is capable, we now need to plot the specification limits onto the voice of the customer (VOC) histogram. To avoid burnout and discontent with the company the software maintenance team requires the lower specification limit (LSL) to be the average of 7 hours worked per day per person and the upper specification limit (USL) to be the average of no more than 8 hours worked per day per person.

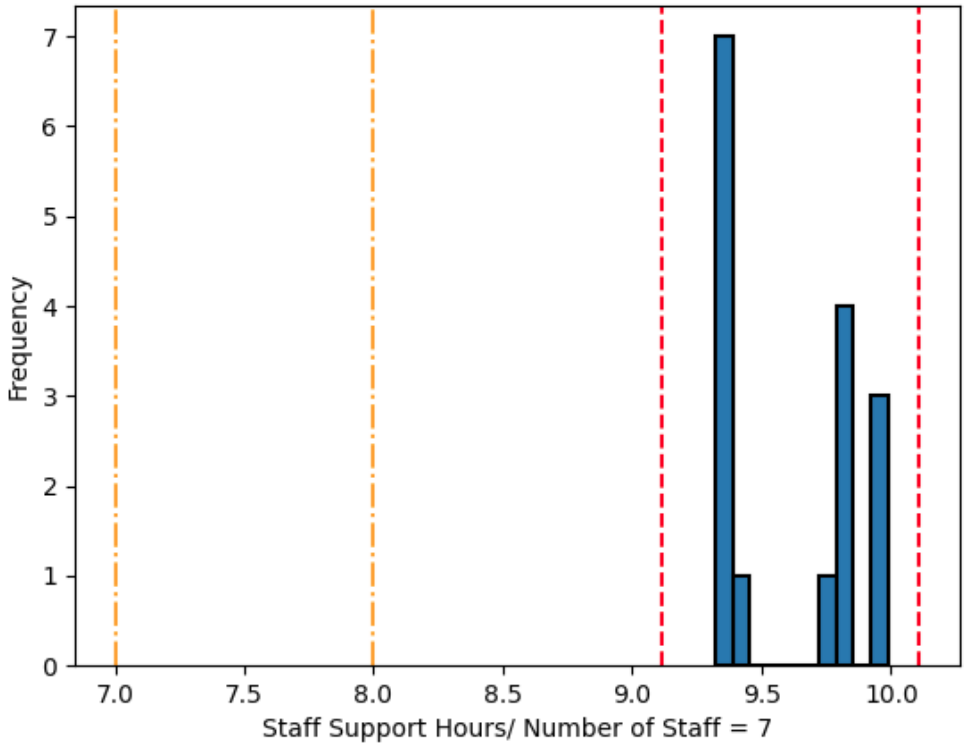


Figure 12A. Histogram for the voice of the customer.

It can be seen from the above histogram that our LCL and UCL are at 9.11 and 10.11, which means that on average each person on the software maintenance team has worked 9 to 10 hours per day for the last 4 months. It is not even close to the desired range.

It can be seen from the above histogram that the process is not meeting the current business requirements and therefore does not appear to be capable. ❎

Not Capable ❎

Make Capable

So what can we do to make our process capable. Let us start by adjusting the number of software maintenance engineers to 9. Below is a snapshot of the modified data with the adjusted values.

Staff Hours Per Day / Number of Staff = 9									
Week	Monday	Tuesday	Wednesday	Thursday	Friday	Average	Min	Max	Range
1	7.84	7.77	7.77	7.77	7.18	7.66	7.18	7.84	0.67
2	7.66	7.42	7.31	7.20	6.73	7.26	6.73	7.66	0.92
3	7.77	7.79	7.77	7.73	7.79	7.77	7.73	7.79	0.06
4	7.40	7.29	6.84	7.52	7.20	7.25	6.84	7.52	0.68
5	7.76	7.28	7.84	7.64	7.74	7.65	7.28	7.84	0.57
6	7.64	7.51	7.52	7.18	6.74	7.32	6.74	7.64	0.90
7	7.71	7.20	7.08	6.71	7.54	7.25	6.71	7.71	1.00
8	7.79	7.74	7.79	7.73	7.71	7.75	7.71	7.79	0.08

Figure 13A. Staff hours per day / number of staff = 9

We once again calculate the grand average and range:

$\overline{\overline{X}}$ = 7.48
$\overline{R}$ = 0.67

Bias correction data for X bar and R chart, will be used to calculate UCL, CL and LCL:

n = 5
A_2 = 0.577
D_3 = 0 (Undefined)
D_4 = 2.114

We calculate the UCL, CL and LCL for X-Bar and R chart:

$UCL_{\overline{X}}$ = 7.48 + 0.577(0.67) = 7.864186111
$CL_{\overline{X}}$ = 7.48
$LCL_{\overline{X}}$ = 7.48 - 0.577(0.67) = 7.091647222

$UCL_{\overline{R}}$ = 2.114(0.67) = 1.415205556

$CL_{\bar{R}}$ = 0.67
$LCL_{\bar{R}}$ = 0

Equation for Sigma for X-Bar Chart Zones:

$\sigma_{\bar{X}} = 0.577(0.86) \div 3 = 0.128756481$

X-Bar Chart:

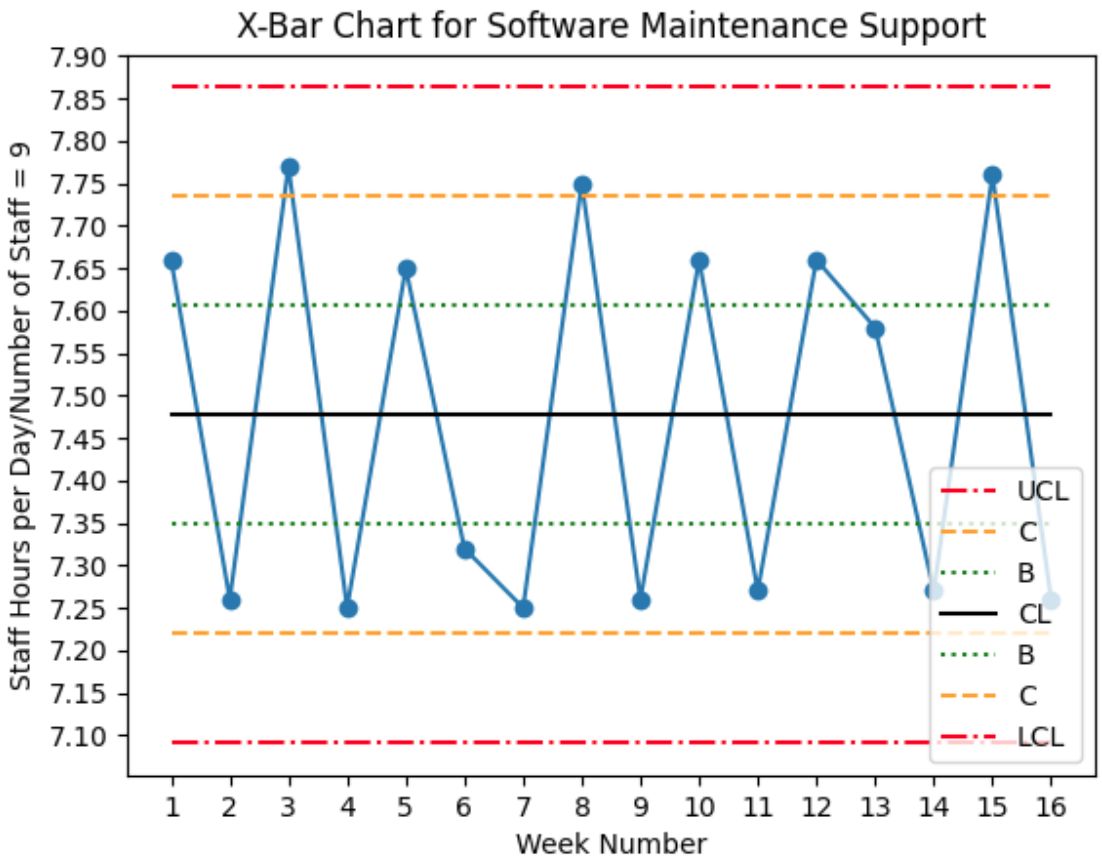


Figure 14A. X-Bar Chart for software maintenance support

R Chart:

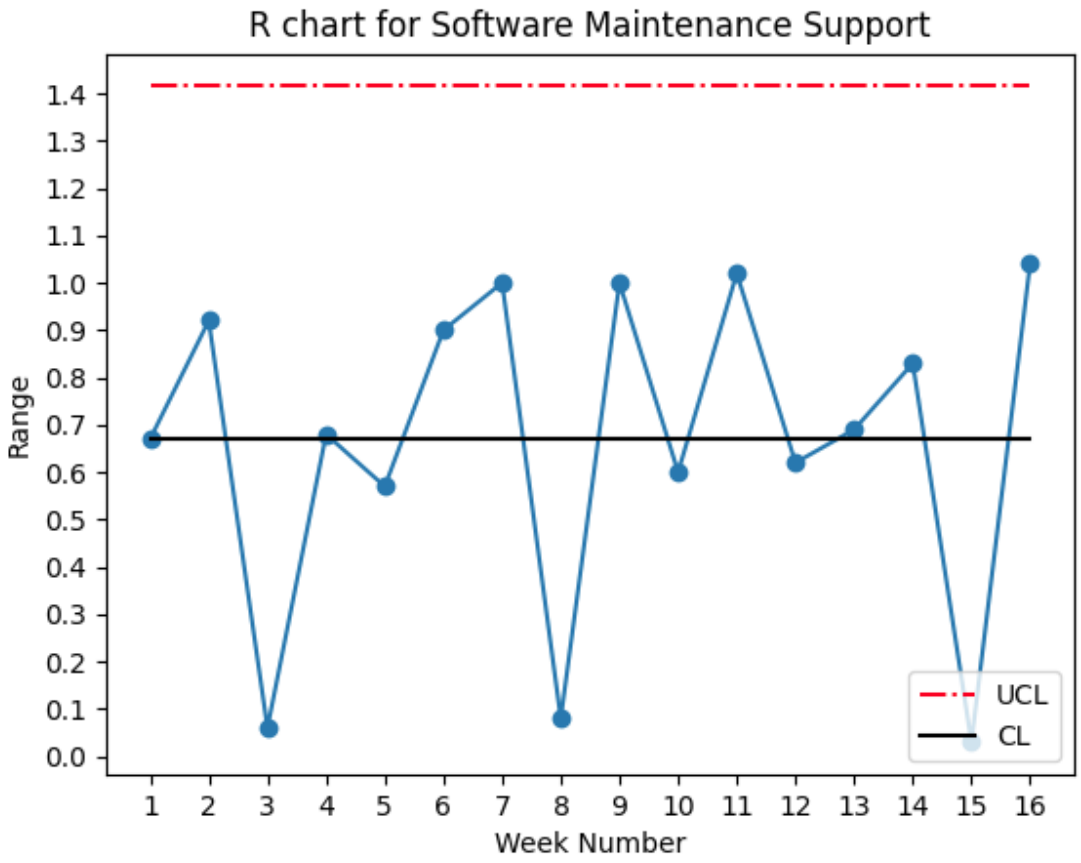


Figure 15A. R Chart for software maintenance support

Test for Stability

Now we can analyze our control charts to see if the process is stable. The R chart needs to be checked first. Only if the R chart passes do we continue to the X-bar Chart. Test 1 applies to both X-Bar and R charts, tests 2 - 4 apply only to the X-Bar chart:

5. **A single points falls out of the UCL or LCL**
    a. **R chart NONE → ✅**
    b. **$\bar{X}$ chart NONE → ✅**
6. **At least two out of three successive values fall on the same side of the CL, and in Zone C. NONE → ✅**
7. **At least four out of five successive values fall on the same side of the CL, and in Zone B. NONE → ✅**
8. **At least eight successive values fall on the same side of the centerline. NONE → ✅**

According to the tests, our process is stable and in control!✅

Now that we have verified, once again, that our process is still stable and in control we can perform the capability analysis for our process.
We use the UCL, CL and LCL that we previously calculated the second time and plot the values onto a histogram for capability analysis.
We take a look at the Voice of the Process (VOP) and see if the process falls within the natural process limits.

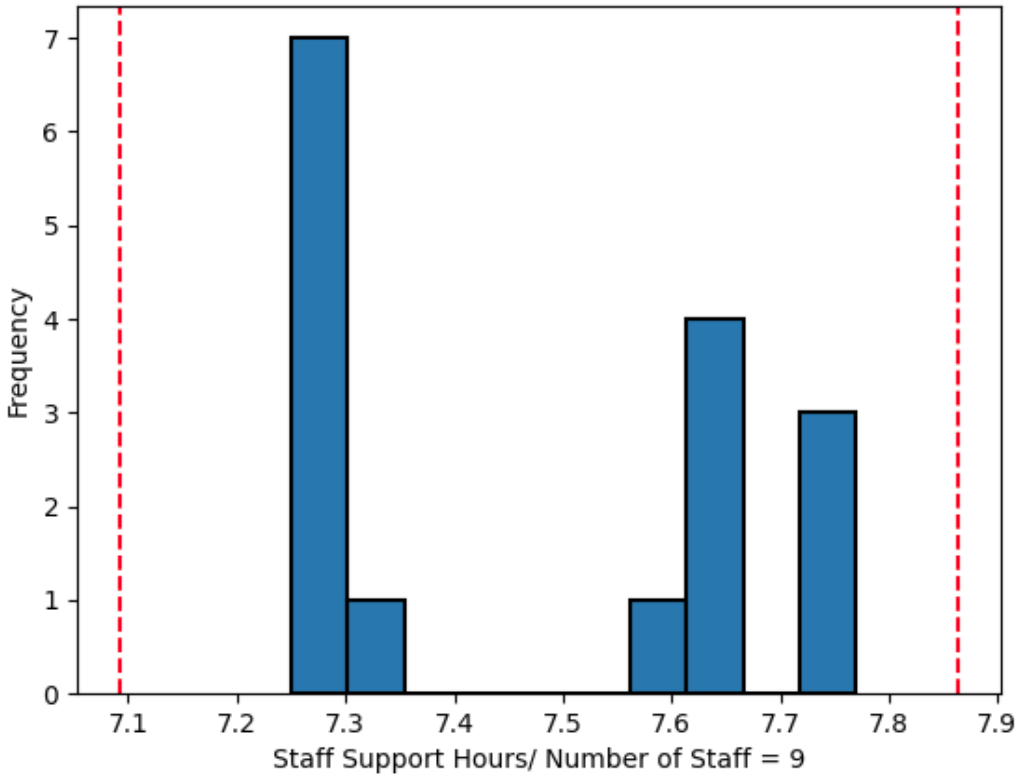


Figure 16A. Histogram for the voice of the process.

According to the histogram above, the process appears to be under stable statistical control and all of the measurements fall within the natural process limits.

To determine if our process is capable, we now need to again plot the specification limits onto the voice of the customer (VOC) histogram. To avoid burnout and discontent with the company the software maintenance team requires the lower specification limit (LSL) to be the average of 7 hours worked per day per person and the upper specification limit (USL) to be the average of no more than 8 hours worked per day per person.

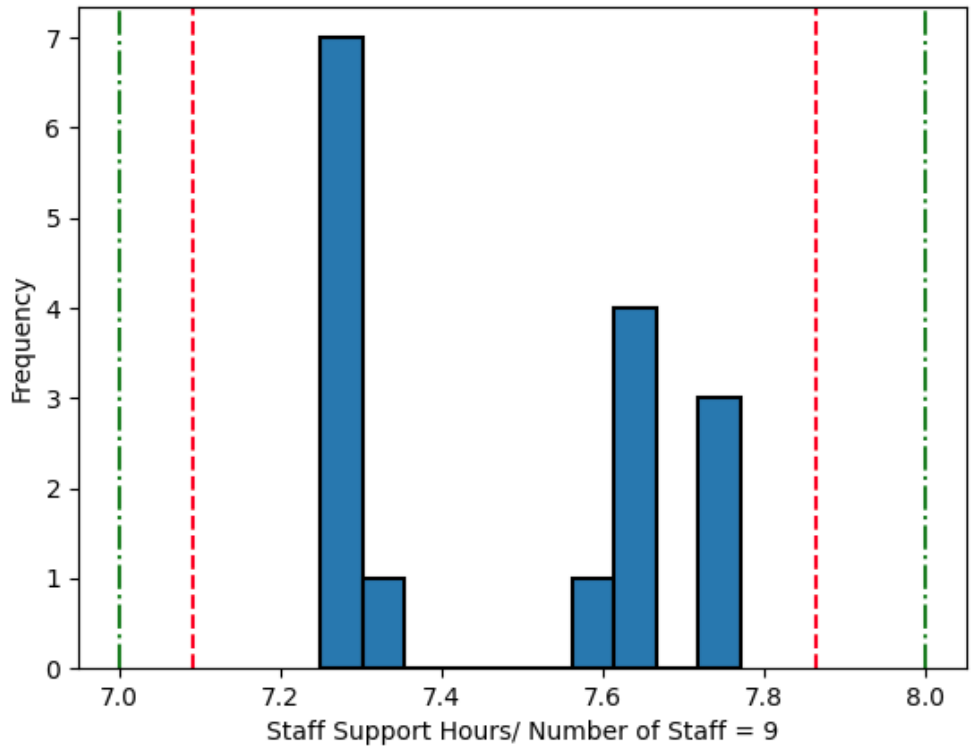


Figure 17A. Histogram for the voice of the customer.

It can be seen from the above histogram that our LCL and UCL are now at 7.09 and 7.86. The NPL is within the SPL so the process is now meeting the current business requirements and therefore is now capable.

Capable ✅

Conclusions and recommendations

Based on our analysis we recommend that our company hires 2 more software maintenance engineers in order to make the current software maintenance process capable. The 2 additional staff on the software maintenance team will ensure that the current staff avoids burnout and/or discontent with the company which can create a toxic work environment and decrease productivity.

Moving on to the two new clients we recently acquired. The amount of clients to support will be doubled and based on the current daily work flow for the last 16 weeks we expect that to double as well. We highly recommend that a total of 11 additional software maintenance engineers be hired in order to be able to maintain a stable and capable software maintenance process at our company. In return our company will be able to provide quality software with quality software maintenance support.

REFERENCES

[1] Bourque, P., and R. E. Fairley. *Guide to the Software Engineering Body of Knowledge, Version 3.0*. IEEE Computer Society, 2014. *www.swebok.org*, www.swebok.org. Accessed 01 04 2021.

[2] Florac, W. and Carleton, A., 1999. *Measuring the software process*. Boston: Addison-Wesley.

[3] Sharif, Arash. "Paths To Process Improvement". 2021.

[4] Institute of Quality and Reliability, *Control Chart Constants and Formulae,* https://web.mit.edu/2.810/www/files/readings/ControlChartConstantsAndFormulae.pdf. Accessed 01 04 20201.