CSE 4621/SWE 5620
Software Metrics

Software Metrics Data Collection

Chapter -5-

Khaled Slhoub, PhD

Where Are We?

- Chapter 1 and my introductory material attempted
  - to provide an overview of the field of software metrics a
  - to justify why this is a worthwhile subject
- Chapter 2 presented the theory of measurement including Measurement scales and meaningful operations
- Chapter 3 presented a framework for determining what to measure; specifically, it addressed
  - Goal-Question-Metric approach (GQM)
  - Capability Maturity Model Integration (CMM)
- We skipped Chapter 4 because of the lack of Designed Experiments in the field of software metrics.

Software Metrics Data Collection

- Chapter 5 focuses on assessing the quality of data and on data gathering techniques that will assure us that we can reach valid conclusions based on the collected data.

- It considers what constitutes good data and presents guidelines and examples to show how data collection supports decision making.

Software Metrics Data Collection

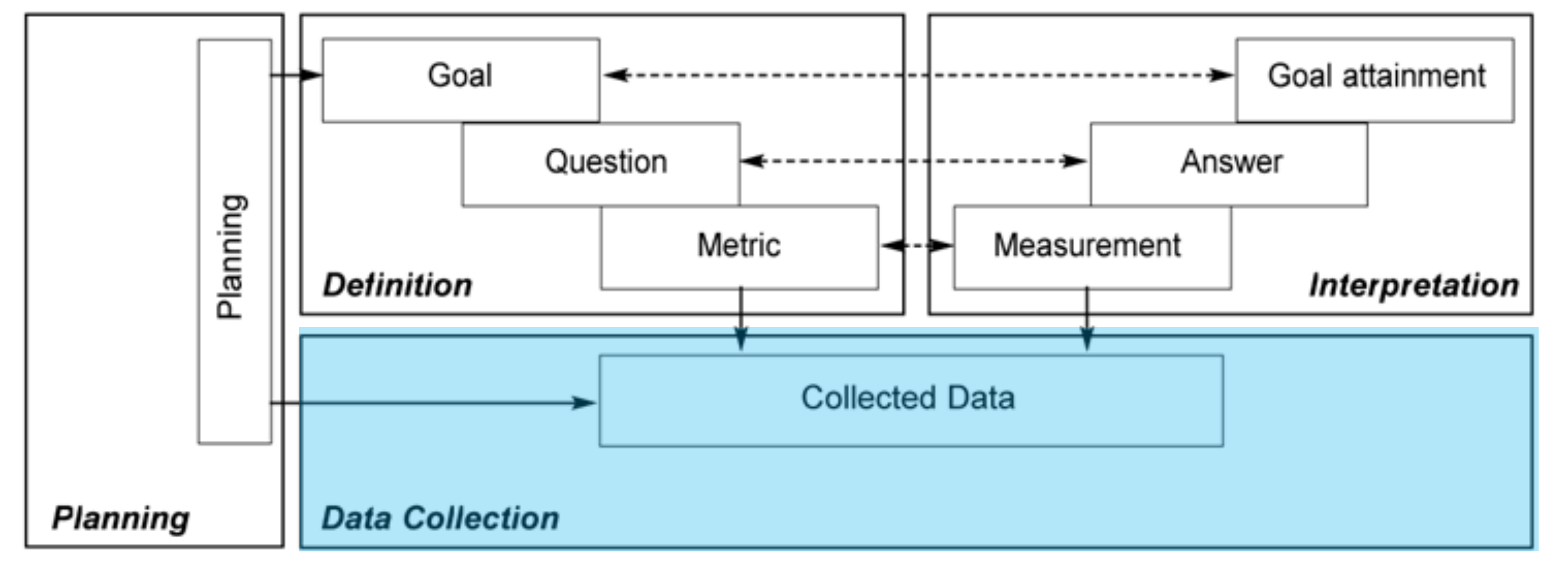
Data should be collected with a clear purpose in mind. Not only a clear purpose but also a clear idea as to the precise way in which they will be analyzed so as to yield the desired information. ...

It is astonishing that men, who in other respects are clear-sighted, will collect absolute hotchpotches of data in the blithe and uncritical belief that analysis can get something out of it.

MORONEY 1962

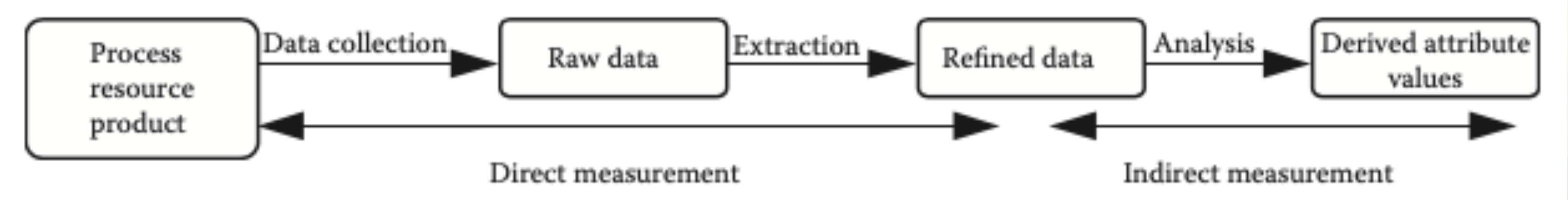
What is Good Data?

- Having the right measures is only part of a measurement program
- Software measurement is only as good as the data that are collected and analyzed
- **We cannot make good decisions with bad data**



Defining Good Data

- It is very important to assess the quality of data and data collection before data collection begins
- Your measurement program must specify not only what metrics to use, but what
  - precision is required,
  - activities and time periods are to be associated with data collection
  - rules govern the data collection

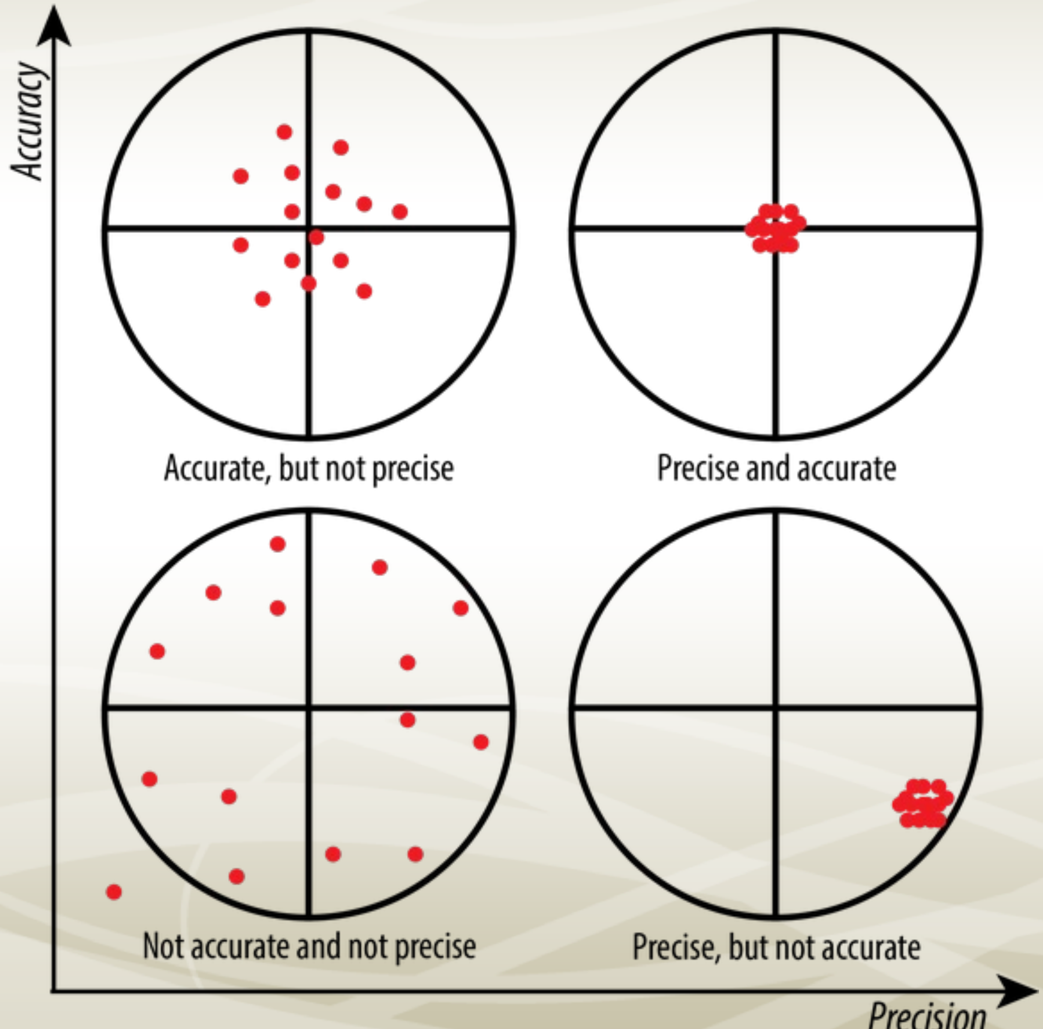


We must specify which direct measures are needed, and also measures that may be derived from the direct ones

Accuracy vs Precision

- Terminology must be clear and detailed, so that all involved understand what the metric is and how to collect it.
- Data should be consistent from one measuring device or person to another, without large differences in value (reproducibility).
- **Correctness** means that the data were collected according to the exact rules of definition of the metric.
- **Accuracy** refers to how close measurements are to the "true" value.
- **Precision** refers to how close measurements are to each other.

Accuracy vs Precision



Because we don't do perfect work

- It is important for developers to measure aspects of software quality. Such information can be useful for determining
  - How many problems have been found with a product?
  - How efficient are the prevention, detection and removal processes?
  - Is the product ready to release to the next development stage or to the customer?
  - How does the current version compare in quality to previous or competing versions?

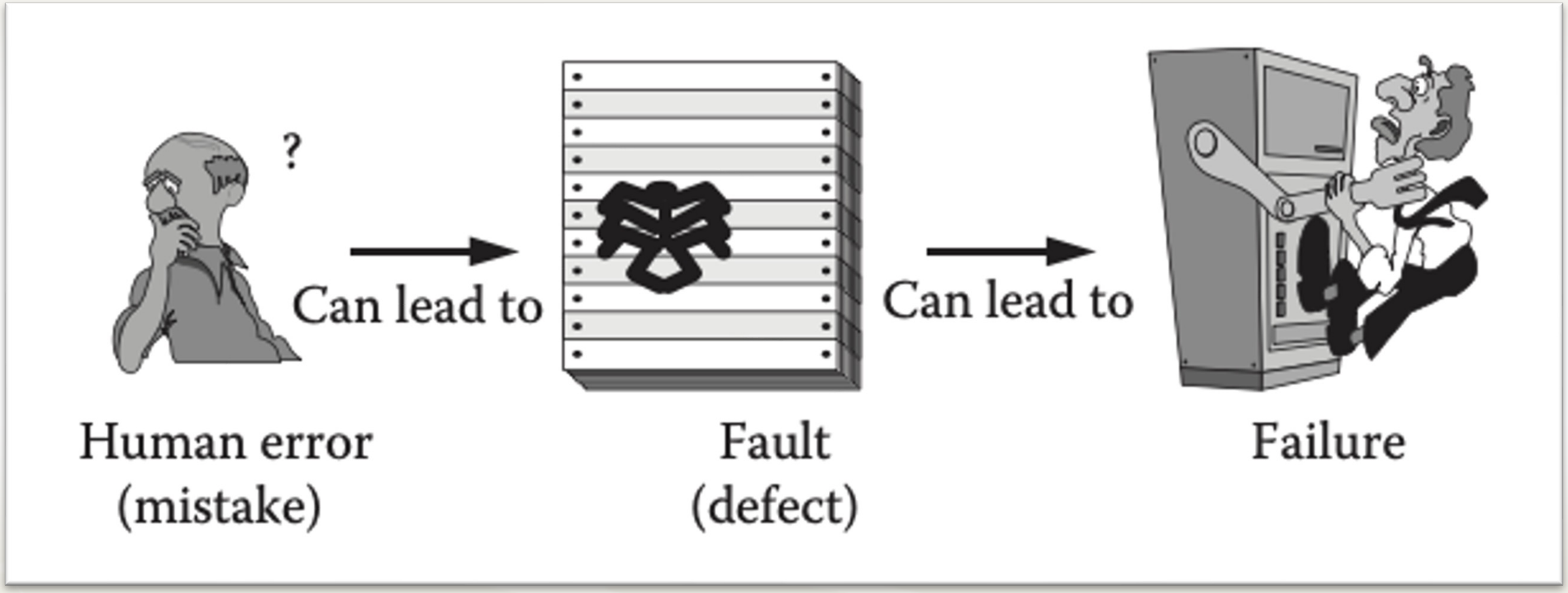
Terminology used to support this investigation and analysis must be precise and accurate

Software Quality Terminology

- **The use of terms varies widely among software professionals.**
- A **fault** occurs due to a human error or mistake in some software products
- A **failure** is the departure of a system from its required behavior.
- Note that faults are (static) mistakes in the product while failures are operational.
- **An error** as "an incorrect internal state that is the manifestation of some fault" " *Ammann and Offutt, 2018*

The fault starts a chain of software errors that ultimately result in a failure

Software Quality Terminology



Not every fault corresponds to a failure, since the conditions under which a fault results in system failure may never be met

Problem with Quality Terminology

Florida Tech

- Terms are not standardized: **failure**, **fault**, **flaw error, mistake, glitch, defect**, **bug**, **anomaly**, **crash**.
- Unfortunately, the terminology used to describe software problems is **not uniform**
- If an organization measures software quality in terms of faults per thousand lines of code, it may be impossible to compare the result with the competition if the meaning of “fault” is not the same
- The software engineering literature is rife with differing meanings for the same terms.

Terminology Problem - Examples

- Errors can mean faults. This meaning contrasts with the notion of software error as used in the software testing community– an error results from executing faults
- Anomalies usually mean a class of faults that are unlikely to cause failures in themselves (ex. use of nonmeaningful names) – IEEE Standards 610.12-1990 and 1044-2009 use the term “Anomaly” to refer to both faults and failures
- Defects often refer collectively to faults and failures – IEEE Standard 1044-2009 uses the term “defect” to refer only to faults
- Bugs refer to faults occurring in the code, but in some cases are used to describe failures.
- Crashes are a special type of failure, where the system ceases to function.

Eight Problem Attributes Template

Quality problems are described in terms of failures, faults, and changes

1. Location: where did the problem occur?
2. Timing: when did it occur?
3. Symptom: what was observed?
4. End result: what consequences resulted?
5. Mechanism: how did it occur?
6. Cause: why did it occur?
7. Severity: how much was the user affected?
8. Cost: how much did it cost?

IEEE Standard Classification for Software Anomalies (IEEE 1044-2009)

Failures

- A failure report focuses on the external problems of the system (the chain of events leading up to the failure)

<u>*Failure Report*</u>

Location: Such as installation where failure was observed

Timing: CPU time, clock time, or some temporal measure

Symptom: Type of error message or indication of failure

End result: Description of failure, such as "operating system crash," "services degraded," "loss of data," "wrong output," and "no output"

Mechanism: Chain of events, including keyboard commands and state data, leading to failure

Cause: Reference to possible fault(s) leading to failure

Severity: Reference to a well-defined scale, such as "critical," "major," and "minor"

Cost: Cost to fix plus cost of lost potential business

Failures - Example

- Early on December 31, 2008, Microsoft's first-generation Zune portable media players hung.

  - *Location*: Many first-generation Zune 30 media players in use around the world.
  - *Timing*: December 31, 2008, starting early in the morning.
  - *Symptom*: The device froze.
  - *End result*: The device became unusable, even after a restart.
  - *Mechanism*: Upon startup, the loading bar indicates "full," and then the device hangs.
  - *Cause* (1): (Trigger) Starting the device on December 31, 2011.
  - *Cause* (2): (Source type) Coding fault related to date calculation.
  - *Severity*: Serious, as it made the device unusable until the inconvenient workaround was communicated to the large and diverse user community.
  - *Cost*: Effort to diagnose the problem, develop and publicize a work-around, and repair the fault. Perhaps, the greatest cost was damage to the company reputation.

Faults

- A Fault focuses on the internals of the system (seen only by developers)

Fault Report

Location: Within system identifier, such as module or document name

Timing: Phases of development during which fault was created, detected and corrected

Symptom: Type of error message reported, or activity which revealed fault (such as review)

End result: Failure caused by the fault

Mechanism: How source was created, detected, and corrected

Cause: Type of human error that led to fault

Severity: Refer to severity of resulting or potential failure

Cost: Time or effort to locate and correct; can include analysis of cost had fault been identified during an earlier activity

Faults - Example

- The fault with the Zune media player (2008)

- *Location*: Module rtc.c, Convert Days function, lines 249–275.
- *Timing*: Created during coding, detected during operational use.
- *Symptom*: Missing condition test causing a loop to iterate incorrectly (nontermination).
- *End result*: The device froze.
- *Mechanism*: Creation: during code development; Detection: diagnosis of operational failure; Correction: workaround provided, code correction probably done.
- *Cause*: Human mistake in dealing with a special case—leap years.
- *Severity*: Serious, as all of the first-generation Zune devices froze.
- *Cost*: Minimal cost to diagnose, prepare workaround, and repair; however, there was significant cost with respect to the reputation of the company.

Problem Changes Template

- Once a failure is experienced and its cause determined, the problem is fixed through one or more changes
  1. Location: Identifier of document or module changed
  2. Timing: When the change was made
  3. Symptom: Type of change
  4. End result: Success of change, as evidenced by regression or other testing
  5. Mechanism: How and by whom change was performed
  6. Cause: Corrective, adaptive, preventive, or perfective
  7. Severity: Impact on the rest of the system, sometimes as indicated by an ordinal scale
  8. Cost: Time and effort for change implementation and test

Data Collection Tools

What would we need to record to produce this table?

Defect Data Cross-Tabulation — Defect Origin

	Requirements	High-level Design	Low-level Design	Code	Unit Test	Component Test	System Te	Field	Total	Remaining	Defect Removal Effectiveness	Phase Containment Effectiveness
High-level Design Inspection	49	681							730	251	74.4%	79.3%
Low-level Design Inspection	6	42	681						729	461	61.3%	72.5%
Code Inspection	12	28	114	941					1095	903	54.8%	61.2%
Unit Test	21	43	43	223	2				332	573	36.7%	100.0%
Component Test	20	41	61	261	0	4			387	190	67.1%	100.0%
System Test	6	8	24	72	0	0	1		111	80	58.1%	100.0%
Field	8	16	16	40	0	0	0	1	81			
Total	122	859	939	1537	2	4	1	1	3465			

Inspection Effectiveness: 73.9% (730+729+1095)/(122+859+939+1537)

Test Effectiveness: 91.1% (332+387+111)/(332+387+111+81)
OR (332+387+111)/(903+2+4+1+1)

Process Effectiveness 97.7% (1 - 81/3465)

Source: Kan, Stephen H., Metrics and Models in Software Engineering, Addison Wesley, 1995

Data Collection Forms

- Data collection forms and graphs encourage collecting good, useful data
- The form should be self-explanatory and include the data required for analysis and feedback
- The form should record both fixed-format data and free-format comments and descriptions
- Boxes and separators should be used to enforce formats of dates, identifiers and other standard values

Data Collection Forms - Example

Problem report form used for air traffic control support system.

CDIS FAULT REPORT S.P0204.6.10.3016

ORIGINATOR:	Joe Bloggs
BRIEF TITLE:	Exception 1 in dps_c.c line 620 raised by NAS

FULL DESCRIPTION Started NAS endurance and allowed it to run for a few minutes. Disabled the active NAS link (emulator switched to standby link), then re-enabled the disabled link and CDIS exceptioned as above. (I think the re-enabling is a red herring.) (during database load)

ASSIGNED FOR EVALUATION TO: DATE:

CATEGORISATION: 0 1 2 3 Design Spec Docn
SEND COPIES FOR INFORMATION TO:
EVALUATOR: DATE: 8/7/92

CONFIGURATION ID	ASSIGNED TO	PART
dpo_s.c		

COMMENTS: dpo_s.c appears to try to use an invalid CID, instead of rejecting the message. AWJ

ITEMS CHANGED

CONFIGURATION ID	IMPLEMENTOR/DATE	REVIEWER/DATE	BUILD/ISSUE NUM	INTEGRATOR/DATE
dpo_s.c v.10	AWJ 8/7/92	MAR 8/7/92	6.120	RA 8-7-92

COMMENTS:

CLOSED

FAULT CONTROLLER: DATE: 9/7/92

Collecting Data to Measure Reliability

Florida Tech

- Comprehensive set of forms for collecting data to measure reliability
- The collection of 10 forms includes all aspects of product fault, failure, and change information

TABLE 5.3 Data Collection Forms for Software Reliability Evaluation

Identifier	Title
PVD	Product version
MOD	Module version
IND	Installation description
IRP	Incident report
FLT	Fault record
SSD	Subsystem version
DOD	Document issue
LGU	Log of product use
IRS	Incident response
CHR	Change record

Applying Ishikawa's Seven Basic Quality Tools in Software Development

Source: Dr Bond's Slides

Review

- Thus far we have
  - examined the general need for software metrics,
  - seen how to use Goal-Question-Metric to focus on the critical metrics to follow.

What’s next?

- Now that we can identify and collect metrics data, what do we do with them?
- Somehow, we must extract the necessary information from the metrics to answer the Questions from GQM.
- In the early stages of a metrics program, we don’t need sophisticated statistical techniques to help us reach conclusions.
- As you collect data, you will often face the need to sort through and understand the information you obtain.
- This involves organizing and summarizing your data and looking for patterns, trends, and relationships.
- Tools can all help you here. These tools are described briefly below and illustrated in greater detail in the SEI report, p 128-p150.

Seven Basic Quality Tools
(Data collection and analysis tools)

- In 1989, Ishikawa identified (but did not invent) 7 fundamental tools:
  1. **The Checklist**
  2. **The Pareto Diagram**
  3. **The Histogram**
  4. **The Run Chart**
  5. **The Scatter Diagram**
  6. **The Control Chart**
  7. **The Cause-and-Effect Diagram**

Reading Assignment

- An excellent discussion of this material can be found in the SEI report: “Practical Software Measurement: Measuring for Process Management and Improvement”
- The PDF version of this report is posted on Canvas.
- Read also Chapters 5 and 6 of the SEI report.

The Checklist/Check Sheet

- A structured, prepared form for collecting, tabulating and analyzing data
- Checklists are used to ensure that the important activities within a process have been performed.
  - Decide what metric will be observed. Develop operational definitions
  - Decide when data will be collected and for how long.
  - Design the form. Set it up so that data can be recorded simply by making check marks
  - check sheet for a short trial period to be sure it collects the appropriate data and is easy to use.
  - This can be done with an Excel spreadsheet so you can analyze the information gathered in a graph.

The Checklist/Check Sheet

- **Example**: a check sheet is just a tabulated list of defects that can be organized by area and by a time-bound aspect such as release, year, quarter, month, week, or day

	Release						
Defect Area	**1**	**2**	**3**	**4**	**5**	6	**Total**
Resume Parser		3	2	2	1	4	**12**
Data Access Layer				1			**1**
Bad Data		4	2		1		**7**
Keyword Analysis		1					**1**
Special Case Rules				1			**1**
Final Calculation						1	**1**
Result Serialization					1		**1**
Total		**8**	**4**	**4**	**3**	**5**	**24**

The Checklist/Check Sheet

Telephone Interruptions

Reason	Day					
	Mon	Tues	Wed	Thurs	Fri	Total
Wrong number	𝍸	||	|	𝍸	𝍸 ||	20
Info request	||	||	||	||	||	10
Boss	𝍸	||	𝍸 ||	|	||||	19
Total	12	6	10	8	13	49

	A	B	C	D	E	F	G	H	I	J
18										
19		Process Name:						Time Period:		
20		Location:						In Person:		
21										
22			Measuring Week : / ~ /							
23		Defect Name	Mon.	Tue.	Wed.	Thu.	Fri.	Sat.	Sun.	Defect Total
24		Day Total:	11	12	14	7	17	14	0	75
25		Defect A		2	3		5			10
26		Defect B	2			1		5		8
27		Defect C	6	2	4	4	1	2		19
28		Defect D		2			1			3
29		Defect E			2					2
30		Defect F				2	3	2		7
31		Defect G		6						6
32		Defect H	3		5		7	5		20

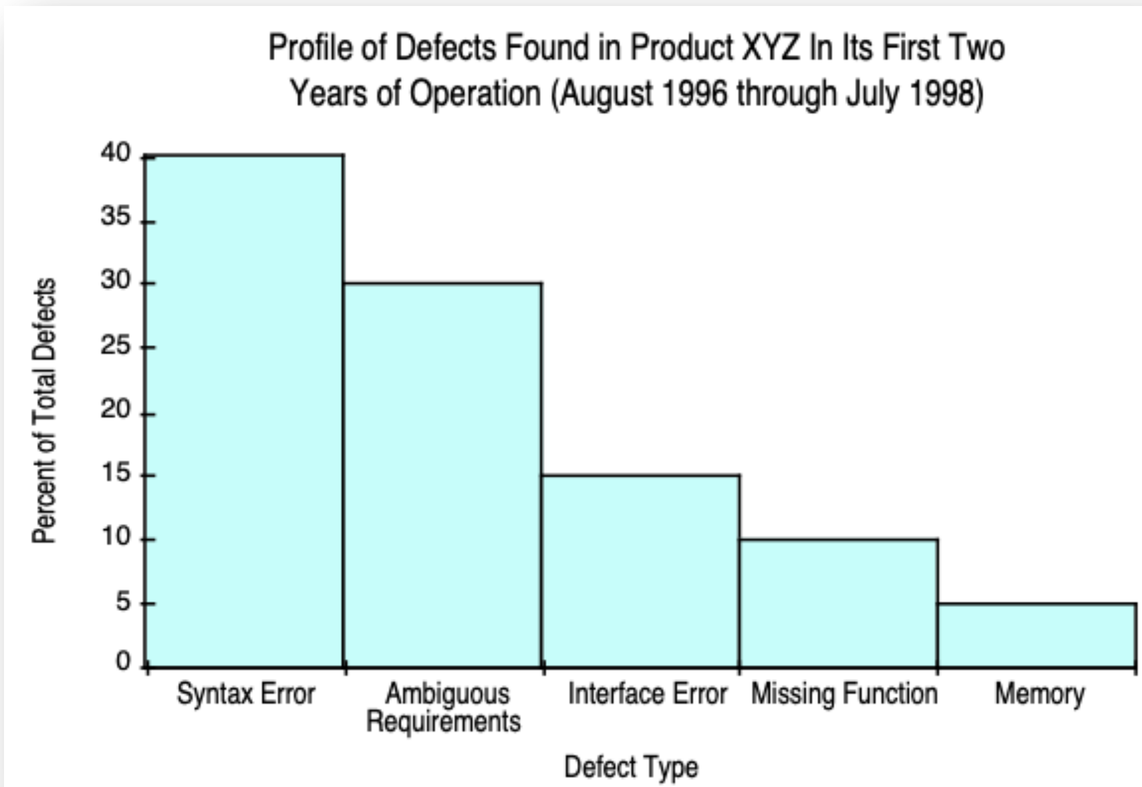
The Pareto Diagram

- A Pareto Diagram is a frequency bar chart displayed in descending order.
- It is very useful in identifying the most frequently occurring software defects.
- It is the source of all of the 80/20 rules.
- Refer to SEI report, page 142-143.

Pareto's Law that states that a relatively small number of causes will typically produce a large majority of the problems or defects.

The Pareto Diagram

- A Pareto diagram helps separate the “vital few” problems from the “trivial many” problems



The Pareto Diagram

Figure 1: Pareto Chart, Customer Complaints

Figure 2: Pareto Chart, Document Complaints

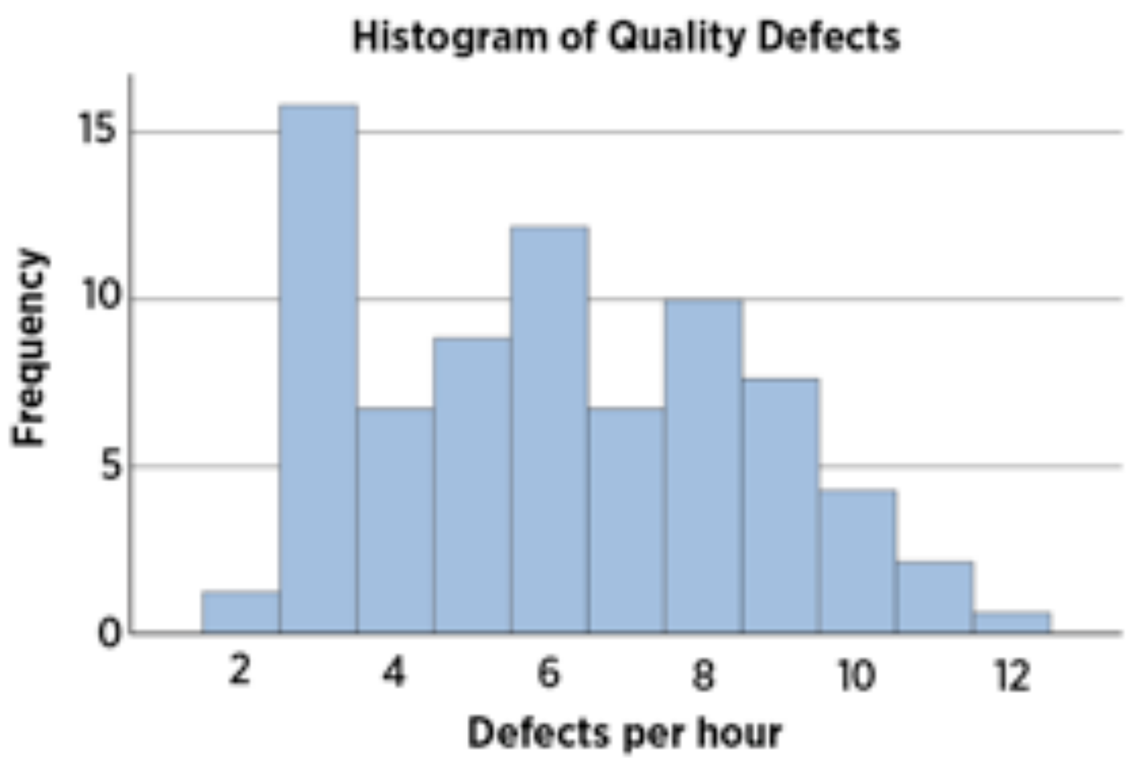
Showing the cumulative percentage line

Histograms

- A Histogram is a graphical representation of the frequency distribution of some dataset of interest.
- A bar graph that uses the height of the bar to convey the frequency of an event occurring.
- They are easy to create using the Data Analysis Add-in in Excel.
- Refer to SEI report, pages 138-140.

Histograms

- used to show the pattern or the distribution of the data across the categories



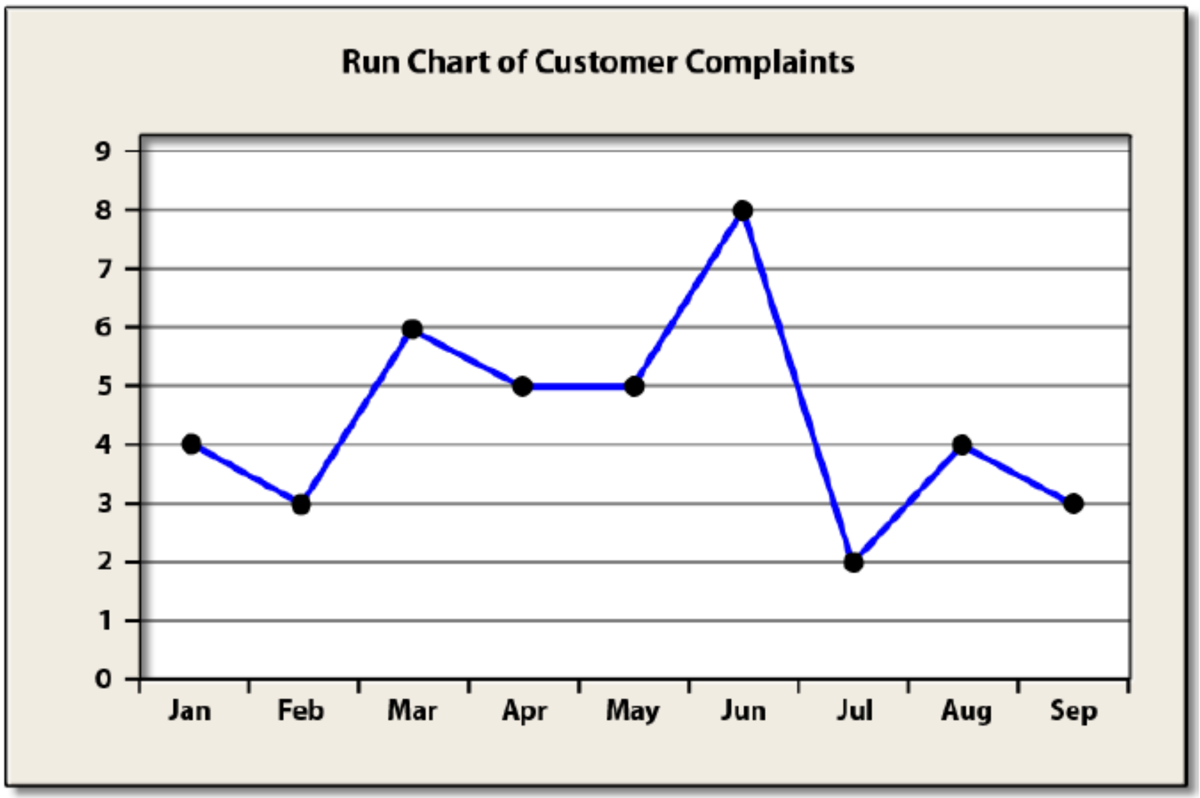
The Run Chart

- A Run Chart is used to track some parameter of interest over time.
- The X-axis is time (or sequence) and the Y-axis is the parameter of interest.
- Used for trend/stability analysis.
- Refer to SEI report, page 132-134.

The Run Chart

Complaints	4	3	6	5	5	8	2	4	3
Month	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep

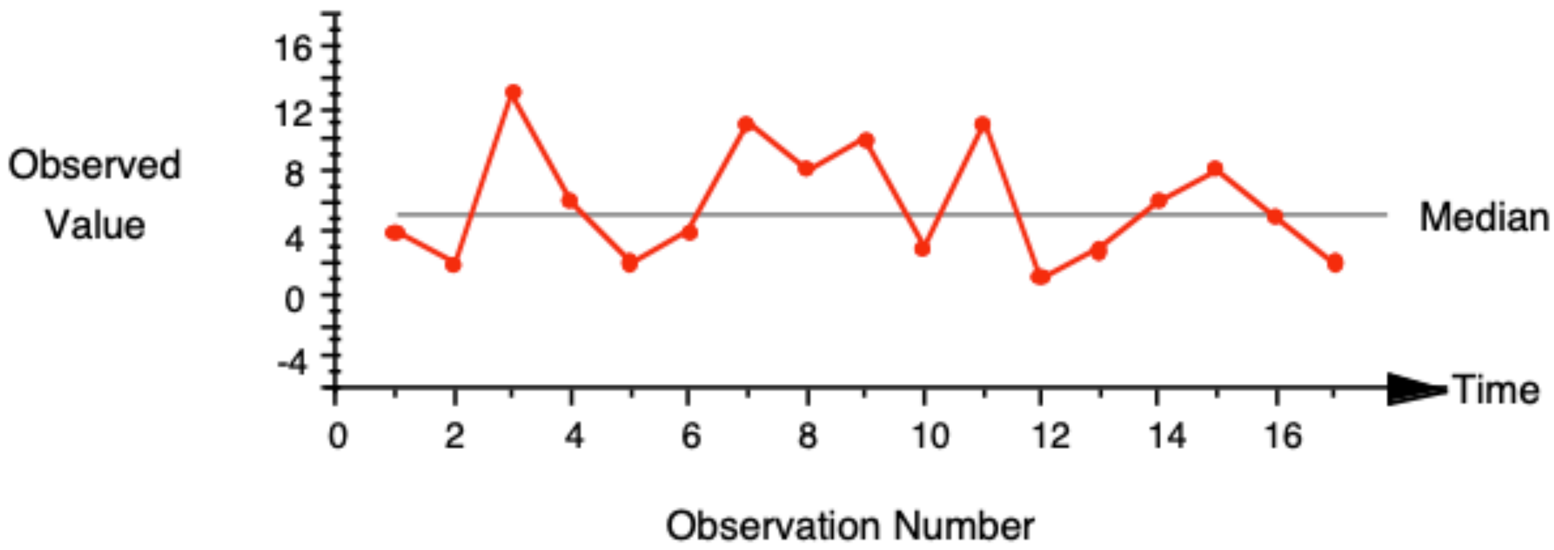
A Run Chart for this data is given below.



The Run Chart

U.S. Equity Markets

The Run Chart

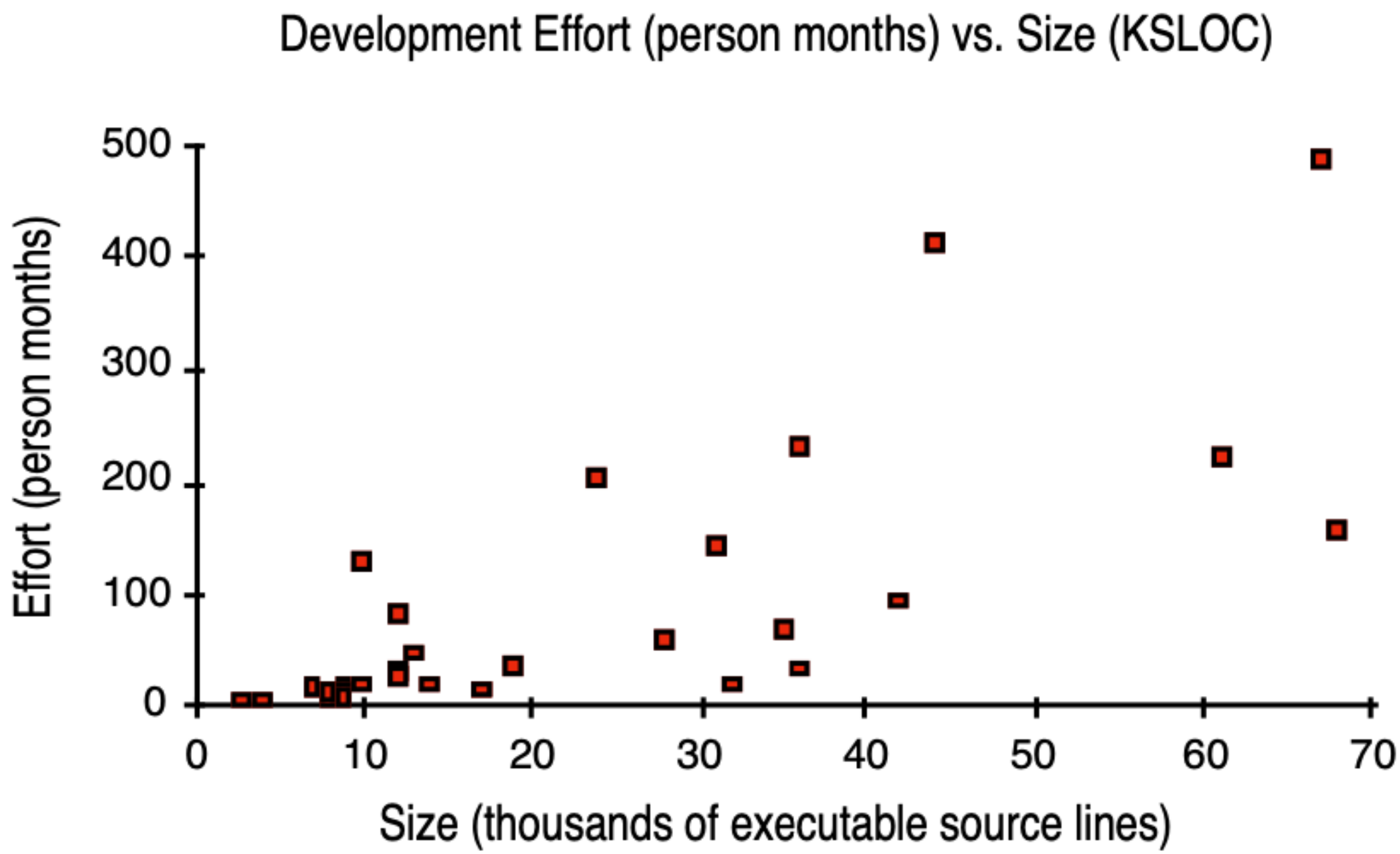


Example of a Run Chart with Level Performance

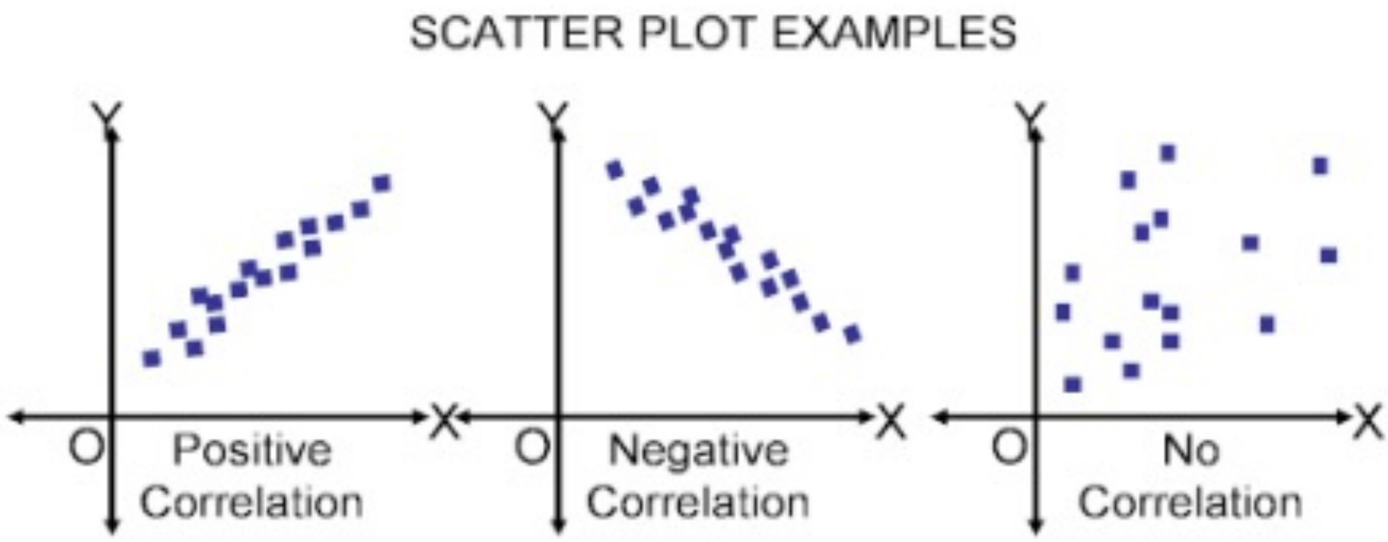
The Scatter Diagram

- The Scatter Diagram is simply a Cartesian plot used to identify a relationship between two parameters of interest.
- It helps to detect and analyze a pattern relationships between two quality and compliance variables (as an independent variable and a dependent variable)
- Excel or other tools may be used to plot the points and to add trend lines.
- Refer to SEI report, page 131.

The Scatter Diagram



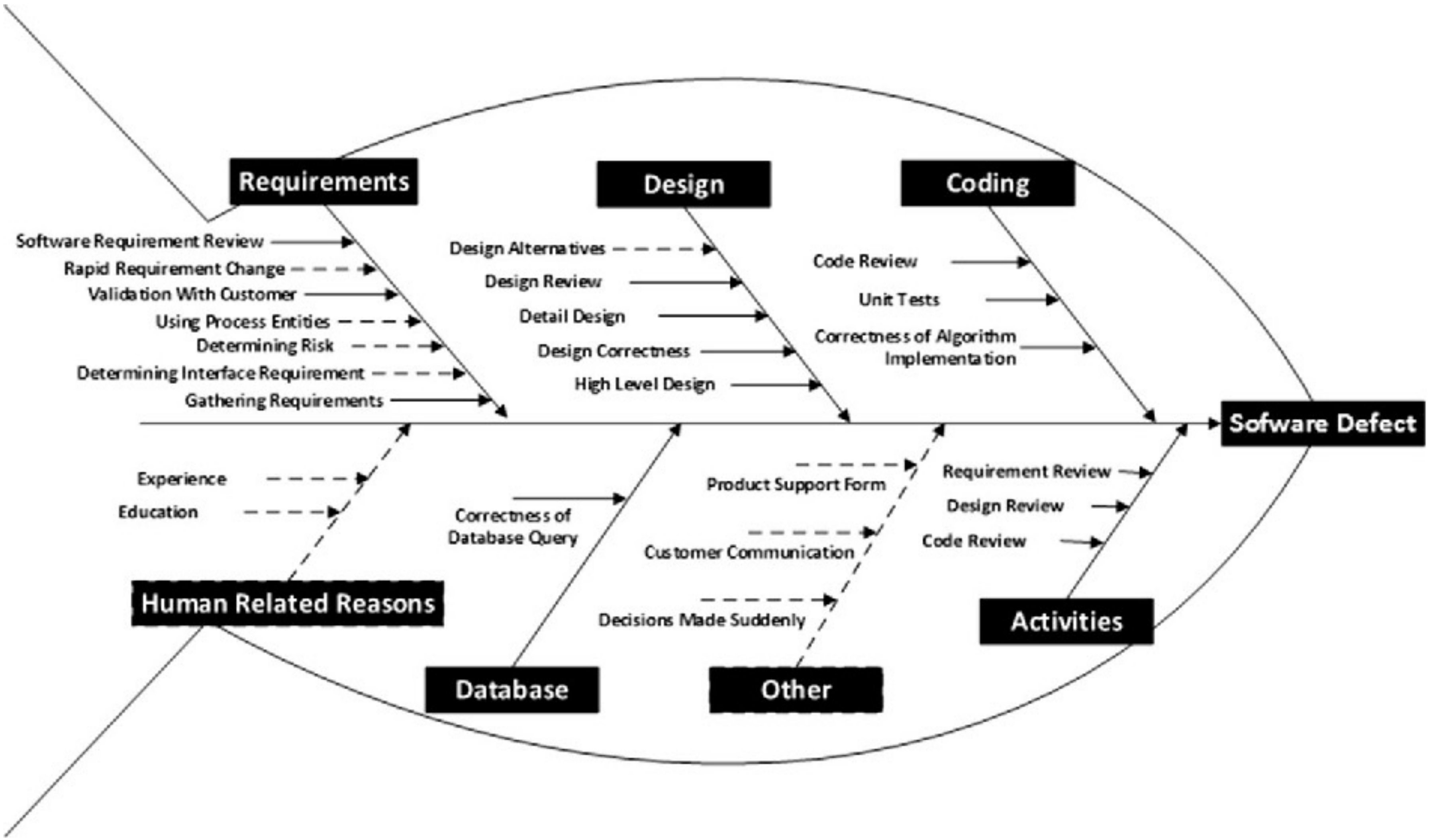
The Scatter Diagram



The Cause-And-Effect Diagram

- Once you determine that a parameter of interest is out of control, how do you find out why?
- The Cause-And-Effect (or Fishbone) Diagram creation process is a **brainstorming process** that may lead you to a conclusion.
- Helps to identify the various factors (or causes) leading to an effect, usually depicted as a problem to be solved.
- Refer to SEI report, page 135-137.

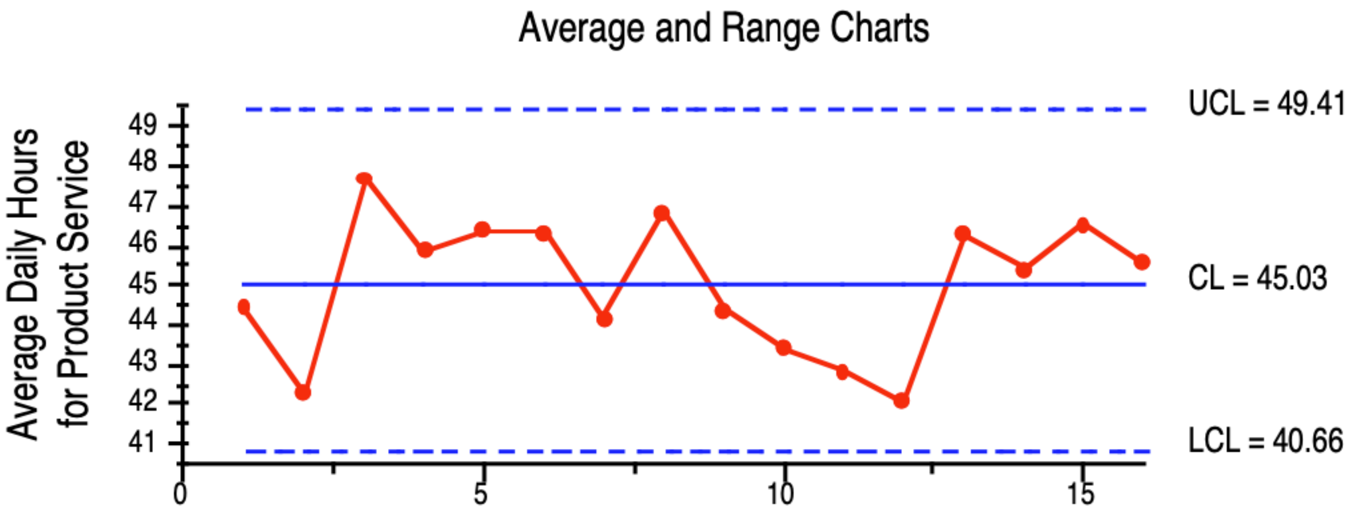
The Cause-And-Effect Diagram



The Control Chart

- The Control Chart is the most complex of the 7 tools and is the major tool used in Statistical Process Control.
- The Control Chart is a graphical representation of the stability and capability of a process.
- The Control chart is a graph used to study how a process changes over time
- The X axis represents a time sequence, while the Y axis represents the parameter of interest, usually calculated from a sample of the output of a process.
- Refer to SEI report, Chapter 5.

The Control Chart



You need to find out how to calculate CL, UCL and LCL.

The Control Chart