Software Engineering

Dokuz Eylül University
Computer Engineering Department

19 February 2001 – 8 June 2001

Akira Imada
imada@cs.deu.edu.tr

□ General Remarks:

1. **Send your e-mail address to imada@cs.deu.edu.tr**

   with the subject being only this course name. and with the text being three lines giving (1) your e-mail address (your most convenient one but preferably the one at this department), (2) your name, and (3) your student-ID.

   as follows:

   To: imada@cs.deu.edu.tr
   Subject: CSE 444
   ———————
   ayhan@hamsi.deu.edu.tr
   Ayhan Akkun Ozturk
   1999510026

⋆ This should be done immediately after the first course.

⋆ This is to communicate you all about the course.

⋆ Important Note: *Do not use any Turkish characters!*

2. **You should basically attend the class every time.**

   Don't be a *never-shown-up-student* when the final exam.

⋆ Although I won't check your presence/absence each time, this does not mean you may skip the class.

⋆ Instead of home works, I give you some tasks during a course and this is to be submitted until the end of the course, not after.

⋆ Since the course materials given in my home-page is not a perfect version like a textbook. but just to compensate my talk in classes, you should take your own note in every classes.

⋆ Though it might be a good strategy to ask me questions before the final exam about the course without attending most of my classes, it will not be accepted.

3. **Final examination.**

⋆ Examination is given only once at the end of the semester where

 – You may open your materials during the exam.
 – Questions are numerical and all fundamental ones that are well explained in the course.

⋆ Under these conditions,

 – do not expect more careful treatment in evaluating your exam. That is, even if
  ∗ the equations you employed are appropriate ones, or
  ∗ your mistakes are just careless

 those will not be evaluated.

 – If you fail to obtain the least point to pass the course, then it means you fail.

□ This course is based on the following three books:

- S. H. Kan:
  - *Metrics and Models in Software Quality Engineering.*
  - Addison-Wesley 1995.
- S. L. Pfleeger:
  - *Software Engineering: Theory and practice.*
  - Prentice Hall, 1999.
- A. von Mayrhauser:
  - *Software Engineering: Methods and Management.*
  - Academic Press, 1990.

Available from, e.g.,

- *http://www.amazon.com*

□ Contents:

1. Introduction
   - ⋆ What is Software Engineering?
   - ⋆ Why Software Engineering?
2. Measurement.
3. Modeling.

Chapter 1

Introduction

Contents:

(1) What is Software Engineering?

(2) Why Software Engineering?

(3) Examples of badly designed/developed system

(4) Two criteria in Software Engineering

- ⋆ Safety critical system
- ⋆ Six sigma criterion

□ What is Software Engineering?

- An engineering that helps us

  to design & develop "*high quality software.*"

- What then is a "*high quality software*?"
  - more efficient;
  - more precise;
  - easier to modify;
  - easier to use;
  - easier to understand.

- How about the quality of our daily life?
  - Where is the button that should be at the sleeve?
  - Why streets are like a river once it rains?
  - Is there any room to improve water drains in our houses?
  - Why should I fixed the washing machine once a year?

- Software Engineering is also a technique to complete
  - *A mission Impossible.*

- The term "*Software Engineering*" was first used in 1968 at a NATO conference.

□ Why Software Engineering?

- Badly developed system

  would cost more than properly developed.
- Improperly designed or programmed software

  would threaten our lives.
- Sometimes it is

  impossible to test.

□ Examples of badly designed/developed system

- A $4-billion fiasco due to inadequate planning.
  - US Internal Revenue Service's project to build an automated federal income tax form processing system proved inadequate to the workload and must be replaced and additionally costed:
    * replacement cost;
    * interest for delayed refunds to taxpayers;
    * employees' overtime wages for that job. (1985)
- THERAC-25 killed patients!
  - A radiation therapy machine malfunctioned and killed several patients by not anticipated the nonstandard usage of arrow keys. (1993)
- A jet fighter targeted a traffic radar gun!
  - One day in Scotland, a radar gun to identify speeding motorists locked up indicating over 300 miles per hour. Second later a low flying Harrier jet flew by whose target seeker had recognized the radar and thought it to belong to an enemy (1996).
- Pilot's computer error caused his plane crash.
  - The code for Cali was the same as the one for Bogoda and this causes the American Airline's jet to hit a mountain in Columbia. 159 people died. (1996)

□ Two Criteria in Software Engineering:

- Safety Critical System
  - A system whose failure[1] poses a threat to life or health should have reliability of at least 10^9, that is, the system can fail no more often than once in 10^9 hours (= 114,000 years) of operation.
- Motorola's Six Sigma Strategy[2]
  - less than 3.4 defective parts per million (10^6) parts.

How can we estimate that a criterion is fulfilled?

$\Downarrow$

By creating a *model*.

$\Downarrow$

Measure and plot the data to ascertain the model.

$\Downarrow$

Measure what/how?

$\Downarrow$

Metrics should be proposed to establish a model.

$\Downarrow$

So, the next topics are:

$\Downarrow$

What metrics can be used and how they should be measured?

[1] A *fault* occurs when a human makes a mistake called an *error* while a failure is a departure from the system's required behavior.

[2] See the next page for more in detail

(But before proceeding)

□ What does it mean by 3.4 ppm?

- If we assume the Gaussian distribution:
  - The probability of defect free parts being distributed within
    * $[-\sigma, \sigma]$ is 68.26 %;
    * $[-2\sigma, 2\sigma]$ is 95.44 %;
    * $[-3\sigma, 3\sigma]$ is 99.73 %;
    * $[-4\sigma, 4\sigma]$ is 99.9937 %;
    * $[-5\sigma, 5\sigma]$ is 99.999943 %;
    * $[-6\sigma, 6\sigma]$ is 99.9999998 %.

  - Hence the proportion of the outside $[-6\sigma, 6\sigma]$ is 0.2 ppm and more precisely speaking 0.1 ppm for the right hand side of the region and 0.1 ppm for the left hand side.

  - But sometimes the distribution shifts due to variations in process execution, and the maximum process shifts by research is 1.5σ.

  - So the interval of $[-6\sigma, 6\sigma]$ sifts between $[-7.5\sigma, 4.5\sigma]$ and $[-4.5\sigma, 7.5\sigma]$

  - It results in 0.34 ppm for the right hand side of the region and practically 0 ppm for the left hand side.

Chapter 2

Measurement

Contents:

□ Terminology in measuring I:

- Ratio:
  - One quantity divided by another;
  - Both quantities are from two distinct populations, and
  - should be *mutually exclusive*
  - Example:

$$\frac{\{\text{number of males}\}}{\{\text{number of males}\}} \tag{1}$$

- Proportion:
  - Numerator should be part of the denominator;
  - Both numerator and denominator should be integer.
  - Example:

$$\frac{a}{a+b+c+d+e} \tag{2}$$

- Fraction
  - The same as proportion but when the measurement unit is not integer.
- Percentage
  - Proportion/Fraction × 100.

- Rate
  - A measure of change in one quantity (y) per unit of another quantity (x) on which the former (y) depends;
  - Both x and y are a function of time, i.e.,
  - rate is *dynamic* while Proportion, Fraction, Percentage are *static*.
  - Example:

$$\{\text{birth rate}\} = \frac{\{\text{total number of birth in a given year}\}}{\{\text{mid-year population}\}} \tag{3}$$

□ Terminology in measuring II:

Reliability and Validity

⋯ Criteria of measurement quality.

- Reliability:
  - Consistency of a number of measurements using the same method on the same subject.
  - A degree of *preciseness*.
  - It can be expressed as:

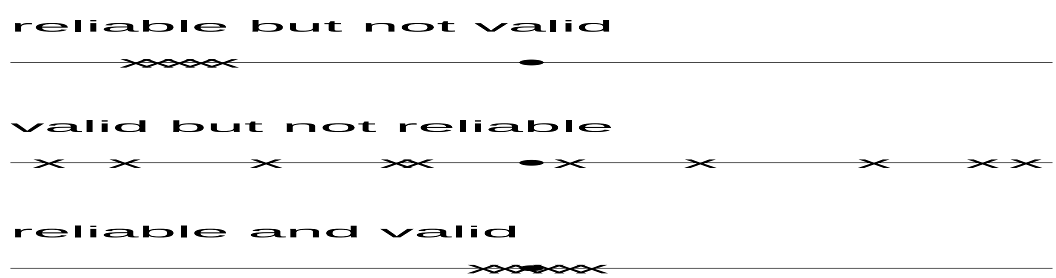
$$\{\text{Index of Variation}\} = \frac{\sigma}{\mu} \tag{4}$$

  - The smaller the index the more reliable.

- Validity:
  - Whether the measurement really measures what we intend it to measure.
  - A degree of *accuracy*.

Note that:

- Reliable measurements may not necessarily be valid.



□ Measurement Errors

$$M = T + s + e. \tag{5}$$

where

$M \cdots$ observed (measured) score;

$T \cdots$ true score;

$s \cdots$ systematic error (associated with validity);

$e \cdots$ random error (associated with reliability).

But under a valid measurement:

$$M = T + e. \tag{6}$$

So,

$$var(M) = var(T) + var(e). \tag{7}$$

where

$$var(\cdot) \text{ represents variance}$$

and

$$\text{Reliability} = \frac{var(T)}{var(M)} = 1 - \frac{var(e)}{var(M)} \tag{8}$$

□ Assessing Reliability

- Retest Method[3]

(1) Measure the subjects twice.

If the measurement is reliable the 1st and 2nd measurement are highly correlated. So,

(2) Compute the correlation[4] between the first and the second measurements.

Then

the correlation coefficient is the reliability.

□ Correction for Correlation

$$\{\text{estimated true correlation between } x \text{ and } y\} = \frac{\{\text{observed correlation between } x \text{ and } y\}}{\sqrt{\{\text{reliability of } x\}\ \{\text{reliability of } y\}}} \tag{10}$$

□ Be careful with Correlation

- Plot the data before calculating correlation coefficient!

[3] Carmines. E. G. et al. proposed the following method to assess the reliability of empirical measurements:
(1) Retest, (2) Alternative-form, (3) Split-halves, and (4) Internal consistency
in their book "Reliability and Validity Assessment." Beverly Hills. California: Sage Publications. (1979)

[4] Correlation between x_i and y_i ($i = 1, \cdots, N$) is defined as

$$\frac{\sum_{i=1}^{N}(x-\bar{x})(y-\bar{y})}{\sigma_x \sigma_y} \tag{9}$$

□ Examples of variables in Software Engineering

- Which development process was employed?
  - · waterfall process
  - · spiral process
  - · iterative process
  - · object-oriented programming process
- The quality of software inspection.
  - · very effective
  - · effective
  - · somewhat effective
  - · not effective
  - · poor
- The inspection coverage
  - · argument is percentage
- The total number of defects customers will encounter for a software product over a specified time length
  - · integer argument
- The process rigor scale.
  - · totally adheres to process
  - · somehow adheres to process
  - · does not adhere to process

- Customer satisfaction.
  - completely satisfied
  - somewhat satisfied
  - neutral
  - dissatisfied
  - completely dissatisfied

□ Four Levels of Measurement

- Nominal scale:
  - A classification into
    - *Mutually exclusive*[5] and *jointly exhaustive*[6] classes.
- Ordinal scale:
  - When the classified subjects can be compared in order.
  - No information of magnitude of difference:
    - Not any of *addition*, *subtraction*, *multiplication*, and *division* are possible.
    - Only *greater than* and *less than* are possible.
    - Transitive in that if $A > B$ and $B > C$ then $A > C$.
- Interval scale:
  - When we have exact differences between measurements.
  - *addition* and *subtraction* are possible.[7]
- Ratio scale:
  - When an absolute or non arbitrary zero point[8] exists.
  - *addition*, *subtraction*, *multiplication*, and *division* are possible.[9]

These four scales are hierarchical in that higher level scale has all properties of the lower ones.

[5] A subject can be classified into one and only one category:

[6] All categories together should cover all possible cases.

[7] We can say for example that "quantity A is 1.5 higher than B," or "The difference between A and B is the same as the one between C and D."

[8] We cannot say an average temperature of 40°C in summer is 8 times higher than an average of 5°C in winter.

[9] We can say for example that A is twice as much as B.

□ Examples of Model (Propositions)

- The more rigorously the front end of the software development process is executed, the better the quality at the back end.

- The higher the percentage of the designs and code that are inspected,

  The more effective the design reviews and the code inspections as scored by the inspection team,

  The more thorough the development test (in terms of test coverage) done before integration,

  $\Downarrow$

  the lower the defect rate will also encountered at the formal machine testing phase.

□ Then questions arise are:

- What is the *software development process?*
- What are the *front end* and the *back end.*
- What is quality of the back end?

  $\Downarrow$

  All should be well-defined, for example:

(to continue)

- Software Development Process is:
  - ⋆ Requirement gathering ⇒ Design ⇒ Design reviews & inspection ⇒ Code ⇒ Code inspection ⇒ Debug & development test ⇒ Integration ⇒ Formal machine testing ⇒ early customer program

  - ⋆ front end ··· before the integration process
  - ⋆ back end ··· after the integration process

Or, which measurement indicators are to be used?
For example: Indicators for

- Reviews & inspection
  (1) Inspection coverage expressed in terms of the percentage of the estimated lines of code that are actually inspected.
  (2) Scoring of each inspection by the inspectors at the end of inspection.
- Rigorous execution of the development test:
  (1) The percent coverage in terms of instructions executed.
  (2) The defect rate expressed in terms of number of defects removed per KLOC.

Chapter 3

Software metrics

Contents:

□ Three categories of Software metrics.

- Product metrics
  - describe the characteristics of the product, such as:
    * size
    * complexity
    * design features
    * performance
    * quality level
- Process metrics
  - can be used for improving the software development and maintenance process, e.g.,
    * effectiveness of defect removal during development
    * pattern of testing defect arrival
    * response time of the fix process
- Project metrics
  - describe the project characteristics and execution, e.g.,
    * number of software developers
    * developers' skill level
    * staffing pattern over the life cycle of the software
    * cost
    * schedule
    * productivity

□ Software quality metrics.

A subset of software metrics that relates with the quality of the product, that is:

- ⋆ Product quality metrics
- ⋆ Process quality metrics
- ⋆ Project quality metrics

⇓ Further divided into two categories:

- end-product quality metrics
- in-process quality metrics

⇓ What is Software Engineering (revisit)

Definition 1 (Software Engineering) *Software Engineering is to investigate the relationships between in-process metrics and end-product quality.*

□ Product Quality Metrics.

- Intrinsic product quality

  (1) Mean time to failure (MTTF)

  e.g. $\cdots$ 10^{-9} criteria in safety critical system.

  (2) Defect density

  e.g. $\cdots$ 3.4 defects per million lines in six-sigma strategy.

- Customer satisfaction

  (3) Customer problems

  (4) Customer satisfaction

□ Defect density revisit

- How do we count lines of code (LOC)?
  - only executable lines?
  - plus data definition?
  - further plus comments?
  - physical lines on an input screen?
  - as the number of logical delimiters?

$\Downarrow$ A definition of LOC (Conte et al,1986)

Any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line including all lines containing headers, declarations, executable, and non executable statements.

□ Defect density for the updated version.

- For the first version, it is easy to state, for example,

  "This product has a total of 50 KLOC and the latent defect rate during the next four years is 2 defects/KLOC."

  $\Downarrow$

- What about subsequent release of the product?

  $\Downarrow$

  We need to measure the quality of

  – the entire product for the new release; and

  – the portion of the product that is new.

  (true development quality, i.e.,

  $\Rightarrow$ the defect rate of the new and changed code)

  $\Downarrow$

- Methods of evaluating defect of the *new and changed code.*

  – Change-flagging method

  – Delta-library method

  – etc

□ A method of counting updated LOC.

- change-flagging method
  - When a new function is added or an enhancement is made to an existing function:
    * the new and changed lines of code are flagged with a specific ID number (change-flagging ID) as comments; and
    * the ID is linked with the requirements number (requirements ID) which is usually in the module header.
  - All of these IDs are further linked to the release number

    $\Downarrow$

  - When a defect is reported, the developer can determine
    * in which function or enhancement pertaining to
    * what requirements,
    * at what release,

    the defect was injected.

□ An example of estimating updated release.

- SSI & CSI (IBM Rochester)

  · SSI (Shipped Source Instructions)

  · CSI (new and Changed Source Instructions)

⋆ SSI of current release = SSI of previous release
+ CSI of current release – number of deleted code
– number of changed code
(to avoid double count in both SSI and CSI)

- TVUA & TFVUA

  · APAR (Authorized Program Analysis Reports)

  $\cdots$ defects identified after the release

  · field APAR $\cdots$ opened due to customer problems

  · internal APAR $\cdots$ opened internally

  · TVUA (Total Valid Unique APAR)

  · TFVUA (Total Field Valid Unique APAR)

  $\star\ \dfrac{\text{TVUA}}{\text{KSSI}}$ $\cdots$ total release code quality

  $\star\ \dfrac{\text{TVUA}}{\text{KCSI}}$ $\cdots$ quality of the new and changed code

  (Both are the same for the 1st release)

  These two are *process measure*, while

  $\star\ \dfrac{\text{TFVUA}}{\text{KSSI or KCSI}}$

  is *customer's perspective.*

□ Customer's perspective of defect rate.

- From customer's perspective, *total number* instead of *rate* of defects should decrease from release to release, regardless of *size*, while from developer's point of view *rate* is more important.

- If the size of a new release becomes much larger than the previous release, then defect rate goal should be significantly better than the previous one in order to reduce the total number of defects.

- For example:
  - The initial release:
    - $*$ KCSI = KSSI = 50 KLOC
    - $*$ Defects / KCSI = 2.0
    - $*$ Total defects = ?
  - The second release:
    - $*$ KCSI = 20
    - $*$ KSSI = 50 + 20 – 4 = 66
    - $*$ Defects / KCSI = 1.8 (10 % improvement)
    - $*$ Total defects = 36
  - The third release:
    - $*$ KCSI = 40
    - $*$ KSSI = 66 + 40 – 10 = 96

From developer's point (they care defect rate):

– defects/KSSI in the 3rd release should be less than 1.62 in order to maintain the improve ration of 10 % in the 2nd release.

From customer's point (they care total number of defect):

– The number of defects in the 3rd release should be *at least* the same as the one in the 2nd release, i.e., 36. So, defect/KCSI in the 3rd release should be less than 0.9 % (36/40) which implies 50 % improvement of the 2nd one.

– And hopefully the number of defect in the 3rd release should be less than 13 ($36 \times 36/100 \approx 13$)

□ Customer Problems Metric[10]

— metric which denotes the problems customers encounter when using the product.

- It is made up of (i) non-defect-oriented problems such as:
  - · usability problems;
  - · unclear documentation;
  - · unclear information;
  - · defects that were reported by other customers but the current customers were not aware of;
  - · user errors;

  as well as (ii) defects.

- It is measured by PUM (problems per user month) as:

$$PUM = \frac{\{\text{Total defect and no-defect problem reported for a period}\}}{\{\text{Total number of license-months of the software during the period}\}} \tag{11}$$

where

$$\begin{aligned}&\{\text{number of license-month}\}\\ &\qquad = \{\text{number of licenses installed}\} \times \{\text{number of month in the period}\}.\end{aligned}$$

[10] Recall we are studying

- Product Quality Metrics
  - MTTF
  - Defect Density
  - Customer Problems
  - Customer Satisfaction

- The other possibilities to represent Customer Problem?

  $\star\ \dfrac{\{\text{defects}\}}{\{\text{source instructions}\}}$

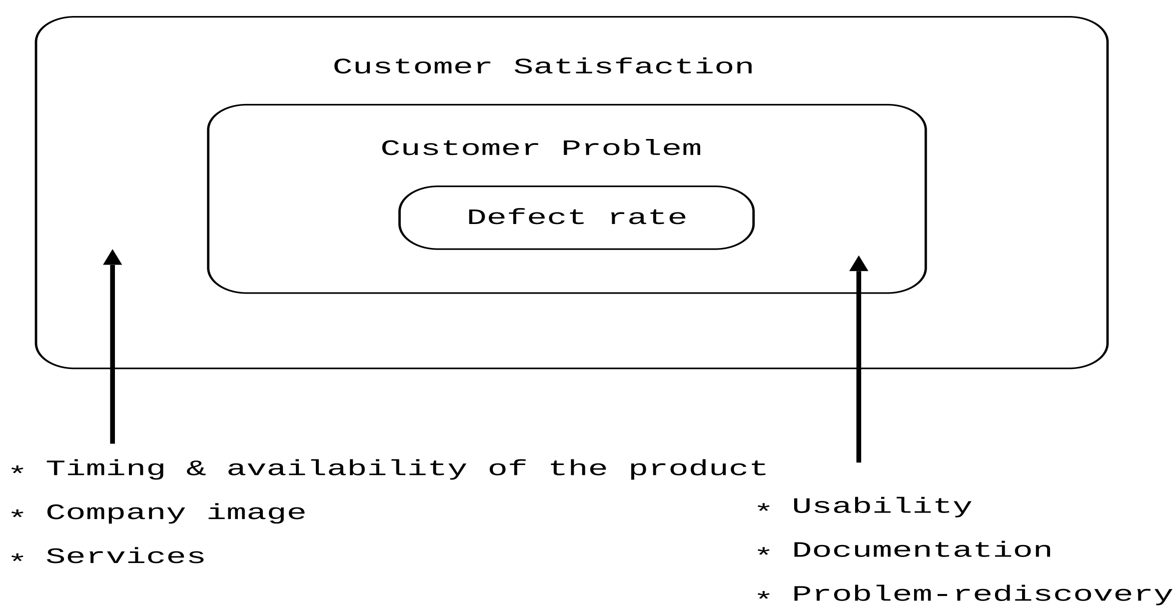
  $\star\ \dfrac{\{\text{problems}\}}{\{\text{usage of the product}\}}$

  these two metrics works as product quality, but

  $\star\ \dfrac{\{\text{total customer problems}\}}{\{\text{size of the product}\}}$

  doesn't work. Why?

- *"Customer Problem Metric"* can be regarded as
  - an intermediate measure between *"defect measurement"* and *"customer satisfaction"*.



□ Alternative measure for LOC

— Function Point Metric

- For example:
  - ⋆ $\{productivity\} = \dfrac{\{\text{function points}\}}{\{\text{person-year}\}}$
  - ⋆ $\{quality\} = \dfrac{\{\text{defects}\}}{\{\text{function points}\}}$

- A function can be defined as
  - a collection of executable statements that performs a certain task

  together with
  - declarations of formal parameters and local variables.

- If defects per unit of functions is low, the quality is better even though the software defects per KLOC are high (when the functions were implemented by fewer lines of code.)

□ How can we obtain Function Point?

I) Divide all the functions in the software we want to asses into 5 categories bellow. Then the complexity of each function of each category is estimated such as low, medium or high complexity. That is to say, we categorize the functions into 15 groups. The total number of the functions in each category is denoted as x_{ij}, and its complexity weighting factor is w_{ij}[11]

i	x_{ij}	complexity w_{ij} low $j=1$	average $j=2$	high $j=3$
1	external input	3	–	6
2	external output	4	–	7
3	logical internal file (files as the user might conceive them)	7	–	15
4	external interface file (files accessed but not maintained by the application)	5	–	10
5	external inquiry (files that are supported on-line)	3	–	6

II) Then Function Count FC is:

$$FC = \sum_{i=1}^{5} \sum_{j=1}^{3} w_{ij} \times x_{ij} \qquad (12)$$

[11] The complexity classification of each component is based on a set of standards that define complexity in terms of objective guidelines.

III) Evaluate the following 14 system characteristics c_i, in terms of their likely effect for the application:

- · Data Communications
- · Distributed functions
- · Performance
- · Heavily used configuration
- · Transaction rate
- · On-line data entry
- · End-user efficiency
- · On-line update
- · Complex processing
- · Reusability
- · Installation ease
- · Operational ease
- · Multiple sites
- · Function of change

IV) Then Value Adjustment Factor VAF is defined as

$$VAF = 0.65 + 0.01 \cdot \sum_{i=1}^{14} c_i. \tag{13}$$

V) Finally we obtain Function Point FP as:

$$FP = FC \times VAF. \tag{14}$$

□ In-process Quality Metrics

- Defect Density during Machine Testing[12](DDMT)
  – measured with $\{\frac{\text{defects}}{\text{KLOC}}\}$

  ⋆ The more DDMT, the more defects will be found later?

  – If *the current* $DDMT \leq$ *the previous release*
    ∗ Ask if the test for current release deteriorate.
      - No ⇒ Quality improved.
      - Yes ⇒ Do extra test such as:
        · test with more cases to increase coverage;
        · blitz test;
        · stress test;
        · test by customer.
  – If *the current* $DDMT >$ *the previous release*
    ∗ Ask if we improved testing effectiveness.
      - Yes ⇒ The quality remains same or improved.
      - No ⇒ Quality perspective negative.

[12]Recall we have the following developing phases:

⋆ Requirement collection → Design →

- Defect Arrival Pattern during Machine Testing
  - The pattern of defect arrivals (reported), or equivalently times between failures, gives more information than the overall defect density.
  - Even with the same overall defect rate, different patterns indicate different quality levels.

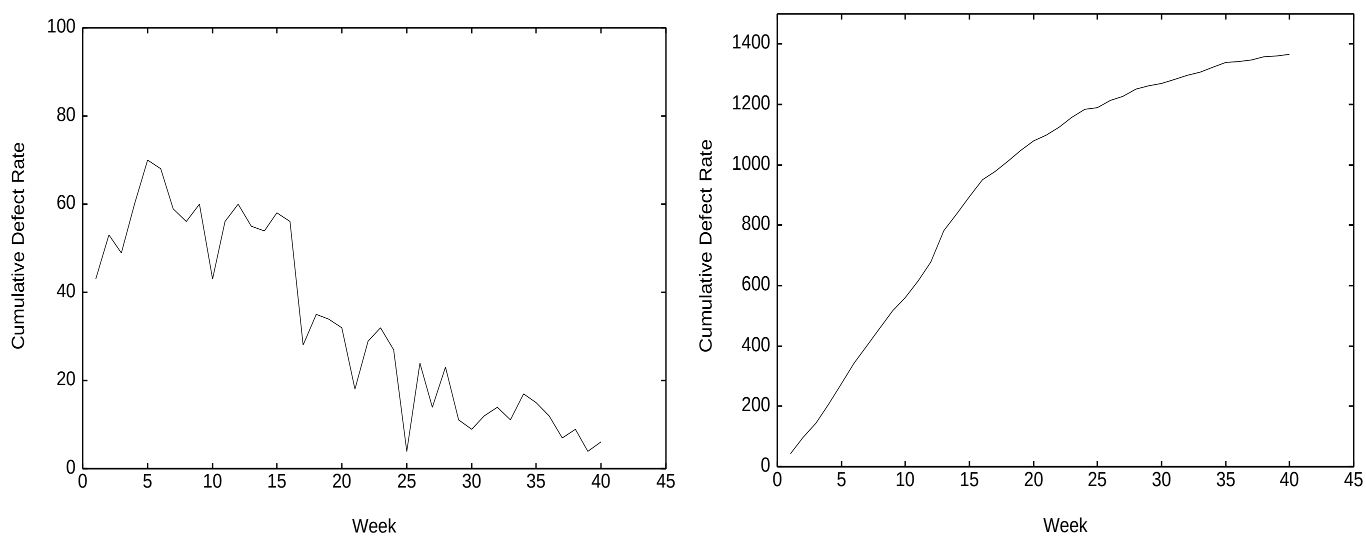


Figure 1: An hypothetical data for DAP and its Cumulative result.

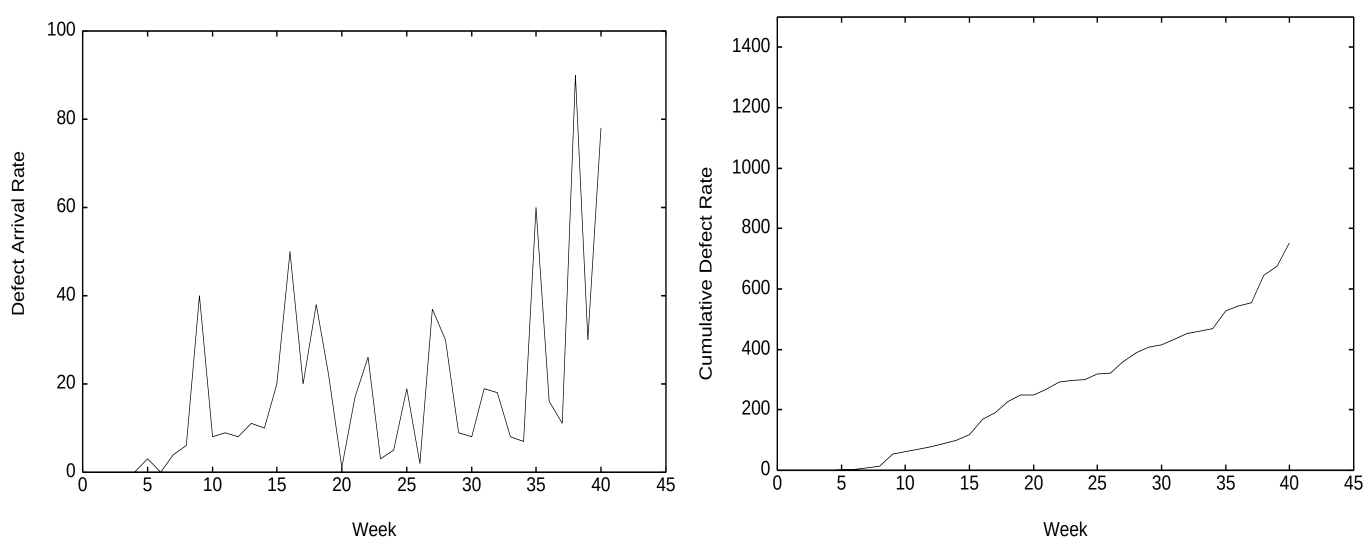


Figure 2: Another hypothetical data for DAP and its Cumulative result.

- The top Figure will give a better quality than the bottom Figure where:
  * Test started late;
  * Test suite was not sufficient; or
  * Test ended prematurely.

- The goal is to make sure that:
  * the defect arrivals stabilizes at a very low level; and
  * the times between the failures becomes long.

- Phase-based Defect Removal Pattern
  - In which phase[13] were defects removed?

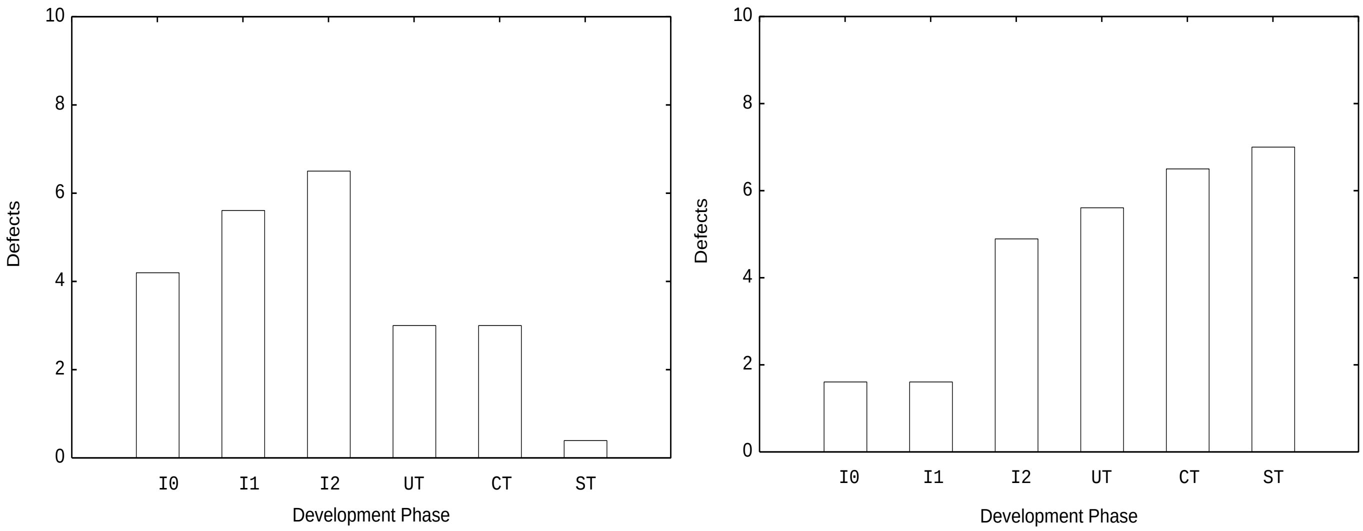


Figure 3: An hypothetical data for PDRP.

[13] Here we have the following six phases:
- I0: high-level design review
- I1: low-level design review
- I2: code inspection
- UT: unit test
- CT: component test
- ST: system test

- Defect Removal Effectiveness (DRE)

$$DRE = \frac{\{\text{defects removed during a development phase}\}}{\{\text{defects latent}\}} \tag{15}$$

- where denominator is usually unknown and evaluated as

$$\{\text{defects removed during the phase}\} \;+\; \{\text{defects found later}\} \tag{16}$$

- We call it

  ⋆ *early defect removal*

  for the phases in the front end; and

  ⋆ *phase effectiveness*

  for each phase.

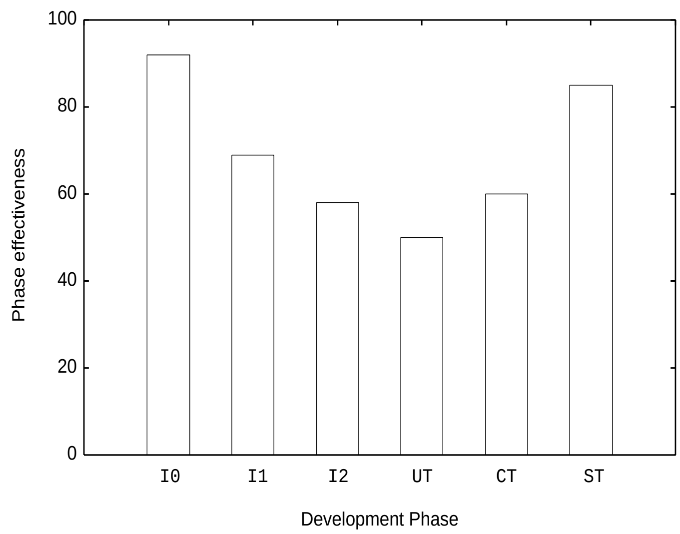


Figure 4: An hypothetical data for DAP and its Cumulative result.

- The higher the value, the more effective the development process; and
- the fewere the defects that escaped to the next phase of to the field.

□ Metrics for Software Maintenance[14]

— Important to improve customer satisfaction (not to improve the defect rate though) such as

· fix the defects claimed as soo as possible

· with excellent fix quality

- Fix Backlog (FB)

$$FB = \{\text{number of reported problems that remain opened at the end of each period (month/week)}\}$$

- Backlog Management Index (BMI)

$$BMI = \frac{\{\text{number of problems closed during a period}\}}{\{\text{number of problems arrived during the period}\}} \times 100\% \tag{17}$$

- If $BMI > 100 \Rightarrow$ backlog is reduced;
- If $BMI < 100 \Rightarrow$ backlog is increases.

---

[14]Main metrics during maintenance are still

· defect arrivals by time interval

· customer problem calls by time interval

but they do not reflect the quality of mintenace.

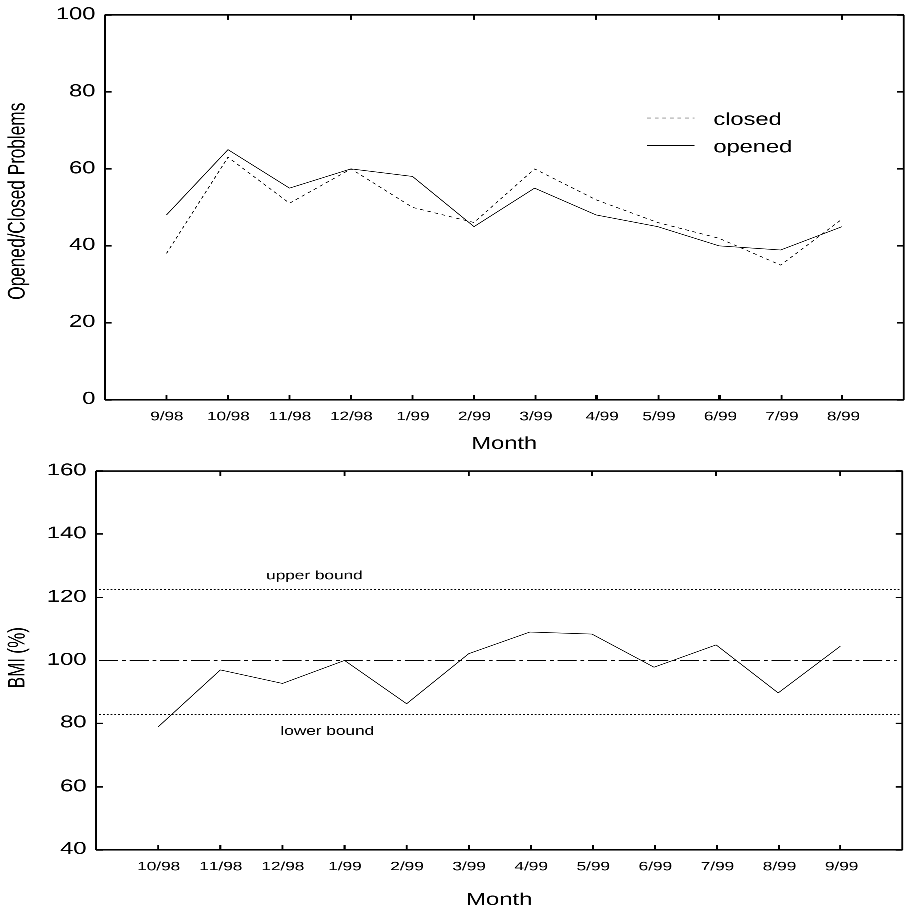


Figure 5: An hypothetical data for opened/closed problems and BMI.

- The low *BMI* of the 1-*st* month is due to the new release.
- The low *BMI* of July 1999 is due to the vacations around Independence day.

- Fix Response Time (FRT)
  - Usually we have guidelines on the time limit within which the fix should be available for the reported defects depending on severity from critical cases (where work should be around the clock to fix it) to less severe defects for which circumventions are available (where works are somehow relax).
  - Defined as[15] :

$$FRT = \{\text{meantime of all problems from open to closed}\} \tag{18}$$

[15] if there exist extreme values, use median.

- Percent Delinquent Fixes (PDF)

  - Defined as:

$$PDF = \frac{\text{number of fixes that exceeded the fix response time criteria by severity level}}{\text{total number of fixes delivered in a specified time}} \qquad (19)$$

  - This is not a real time metric, because only for closed problems
  - Still open problems must be considered to be a real time one

$$\Downarrow$$

  - A new proposal:
    - ⋆ Percent delinquent of all problems in the active backlog
      - · where *active backlog* is all open problems for a period, that is,

$$\{\text{existing backlog at the beginning of the period}\} + \{\text{new problem arrivals during the period}\}$$

      - · which includes the information about the total number of problems to be processed for the period; and
      - · the number of delinquent poblems is cheked at the end of the week.

- Percent Defective Fixes (DF)

  - Defined as:

$$DF = \frac{\text{fixes that are defective}}{\text{all fixes in a time interval}} \tag{20}$$

  - *Fix Quality* is estimated by this metric;

    and

  - A fix is defective if
    - · it did not fix the problem that was reported; or
    - · it fixed the original problem but injected a new defect.

  - An argument against using percentage, where
    - · the absolute value of the nuber of defective fixes is to be used and goal shoul be zero defective fixes without delinquency.

  - Two ways in recordin a defective fix:
    - · Record it when it was discovered (customer's measure);
    - · Record it when the fix was delivered (process measure).

$\Downarrow$

Time diference between two dates = latent period of the defective fix.

The longer the latency, the more customers are affected.

□ Other Metrics Used in the Real World

Motorola

- To improve Project Planning:
  - What was the accuracy of estimating the actual value or project schedule?
    - ⋆ Schedule Estimation Accuracy (SEA)

$$SEA = \frac{\text{actual project duration}}{\text{estimated project duration}} \tag{21}$$

  - What was the accuracy of estimating the actual value of project effort?
    - ⋆ Effort Estimation Accuracy (EEA)

$$EEA = \frac{\text{actual project effort}}{\text{estimated project effort}} \tag{22}$$

- To increase defect containment:
  - What is the currently known effectiveness of the defect detection process prior to release?
    - ⋆ Total Defect Containment Effectiveness (TDCE)

$$TDCE = \frac{\text{number of prerelease defects}}{\text{number of prerelease defects} + \text{number of postrelease defects}} \tag{23}$$

  - What is the currently known containment effectiveness of faults introduced during each constructive phase of software development for a particular software product?
    - ⋆ Phase Containment Effectiveness for phase (PCEi)

$$PCEi = \frac{\text{number of phase } i \text{ errors}}{\text{number of phase } i \text{ errors} + \text{number of phase } i \text{ defects}} \tag{24}$$

- To increase Software Reliability
  - What is the rate of software failures, and how does it change over time?
    - ⋆ Failure Rate (FR)

$$FR = \frac{\text{number of failures}}{\text{execution time}} \tag{25}$$

- To decrease Software Defect Density
  - What is the normalized number of in-process faults, and how does it compare with the number of in-process defects?
    - ⋆ In-process Faults (IPF)

$$IPF = \frac{\text{in-process faults caused by incremental software development}}{\text{assembly-equivalent delta source size}} \tag{26}$$

    - ⋆ In-process Defects (IPD)

$$IPD = \frac{\text{in-process defects caused by incremental software development}}{\text{assembly-equivalent delta source size}} \tag{27}$$

- What is the currently known defect content of software delivered to customers, normalized by assembly-equivalent size?
  - ⋆ Total Released Defects Total (TRDT)

$$TRDT = \frac{\text{number of released defects}}{\text{assembly-equivalent total source size}} \tag{28}$$

  - ⋆ Total Released Defects Delta (TRDD)

$$TRDD = \frac{\text{number of released defects caused by incremental software development}}{\text{assembly-equivalent total source size}} \tag{29}$$

- What is the currently known customer-found defect content of software delivered to customers, normalized by assembly-equivalent source size?
  - ⋆ Customer-Found Defects Total (CFDT)

$$CFDT = \frac{\text{number of customer-found defects}}{\text{assembly-equivalent total source size}} \tag{30}$$

  - ⋆ Customer-Found Defects Delta (CFDD)

$$CFDT = \frac{\text{number of customer-found defects caused by incremental software development}}{\text{assembly-equivalent total source size}} \tag{31}$$

- To improve Customer Service
  - What is the number of new problems that were opened during the month?
    - ⋆ New Open Problems (NOP)

$$NOP = \text{total new post-release problems opened during the month} \tag{32}$$

  - What is the total number of open problems at the end of the month?
    - ⋆ Total Open Problems (TOP)

$$TOP = \text{total new post-release problems that remain open at the end of the month} \tag{33}$$

  - What is the mean age of open problems at the end of the month?
    - ⋆ Mean Age of Open Problems (AOP)

$$AOP = \frac{\text{total open time of post-release problems remaining open at the end of the month}}{\text{number of open post-release problems remaining open at the end of the month}} \tag{34}$$

  - What is the mean age of the problems that were closed during the month?
    - ⋆ Mean Age of Closed Problems (ACP)

$$ACP = \frac{\text{total time post-release problems closed within the month were open}}{\text{number of open post-release problems closed within the month}} \tag{35}$$

- To reduce the cost of nonconformance
  - What was the cost to fix postrelease problems during the month?
    - ⋆ Cost of Fixing Problems (CFP)

$$CFP = \text{dollar cost associated with fixing post-release problems with the month} \tag{36}$$

- To increase software productivity
  - What was the productivity of software development projects?
    - ⋆ Software Productivity Total (SPT)

$$SPT = \frac{\text{assembly-equivalent total source size}}{\text{software development effort}} \tag{37}$$

    - ⋆ Software Productivity Delta (SPD)

$$SPD = \frac{\text{assembly-equivalent delta source size}}{\text{software development effort}} \tag{38}$$

Hewlett-Packard
(under preparation)

IBM Rochester
(under preparation)

Chapter 4

A Model of Software Reliability — The Rayleigh Model

Contents:

□ Software Reliability Model

- *Software Reliability Models* are used to estimate
  - the reliability
  - the number of latent defects

  of a software.

- They are important
  - as an objective statement of the quality; and
  - for a resource planning.

- Two categories:
  - *static models*
  - *dynamic models*

  exist.

□ Static Model and Dynamic Model

- Static Model:
  - It uses attributes of the project or program modules to estimate the number of defects.
  - It has the form:

$$y = f(x_1, x_2, \cdots, x_n) + e$$

    where $x_1, x_2, \cdots, x_n$ are attributes of the product (e.g., size, complexity, $\cdots$) the project (e.g., number/skill of programmers, $\cdots$) or the process (e.g., period cost, $\cdots$) during the development, y is defect rate or the number of defects, and e is an error term.
  - Static models are static in the sense that contribution of attributes are estimated based on previous projects.

- Dynamic Model:
  - It uses current development defect patterns to estimate end-product reliability.

    (to be continued)

- It has two categories:
  - · those that model entire development process
  - · those that model the back-end formal testing phase,

  the representative of which is:
  - · Rayleigh Model
  - · Exponential Model

  respectively.

- The denominator is always a function of time.

- Which model for what?
  - Dynamic model gives better estimation of the global reliability, in general; but
  - static model is better for *hypothesis testing* (to show, for example, that certain project attributes are related to better quality) rather than for reliability; or
  - when the unit is more granular like program module level the static model can be more powerful to provide clues to the software engineers on how to improve the quality of their design and implementation.

□ The Rayleigh Model

- The Weibull distribution:

$$f(t) = (m/t)(t/c)^m \exp\big(-(t/c)^m\big) \qquad (39)$$

where

· m is shape parameter;

· c is scale parameter;

· t is time,

has long been used for *reliability analysis*, such as

· lifetime of ball bearings;

· electron-tube failures;

· overflow incidence of rivers.

- Tail approaches zero asymptotically, but never reaches it.

- When applied to software $f(t)$ might represent

· defect density over time;

· defect arrival pattern,

and its cumulative density function (cdf) form:

$$F(t) = 1 - \exp\big(-(t/c)^m\big) \qquad (40)$$

means

· cumulative defect arrival pattern

- To model software reliability $m = 1$ or 2, usually.

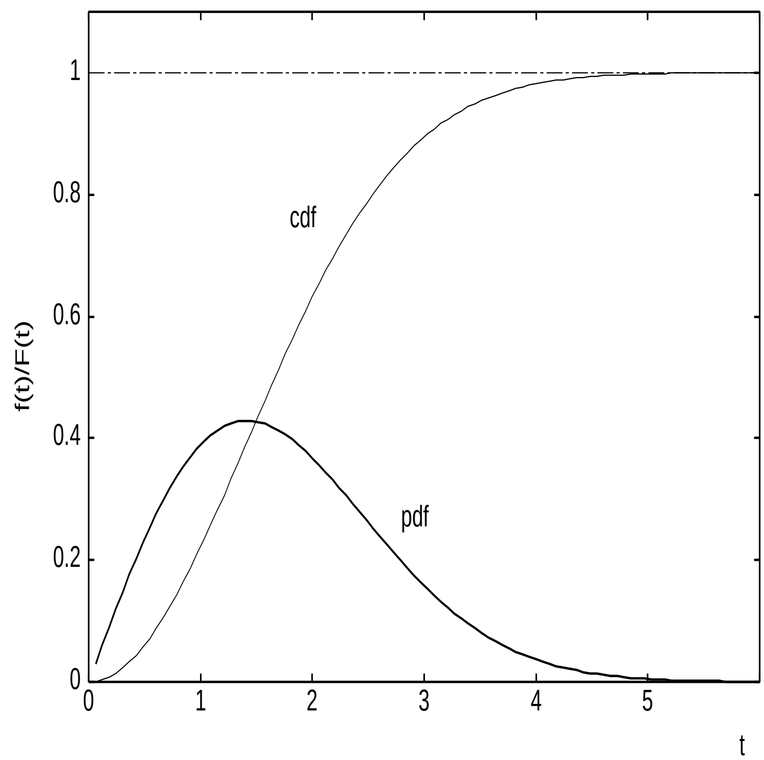


Figure 6: An example of Rayleigh pdf and cdf when $m = 2$.

- The Rayleigh model is a special case when $m = 2$:

$$f(t) = (2/t)(t/c)^2 \exp\big(-(t/c)^2\big) \tag{41}$$

- where c determines the time t_m at which the curve reaches its peak the curve

$$t_m = c/\sqrt{2} \tag{42}$$

- The area below the curve up to t_m is 39.35% of the total area.

- In actual case when K is the total number of defects

$$f(t) = K(2/t)(t/c)^2 \exp\big(-(t/c)^2\big) \qquad (43)$$

- Examples of applications to software:

  - Software projects follow a pattern described by the Rayleigh density curve (Norden, 1963).

  - The defect removal pattern follows the Rayleigh pattern (Gaffney, 1984).

  - The month-by-month error pattern of software projects resembled a Rayleigh-like curve (Trachtenberg, 1982).

- Defect counts at six phases followed a Rayleigh curve (Gaffney, 1984).

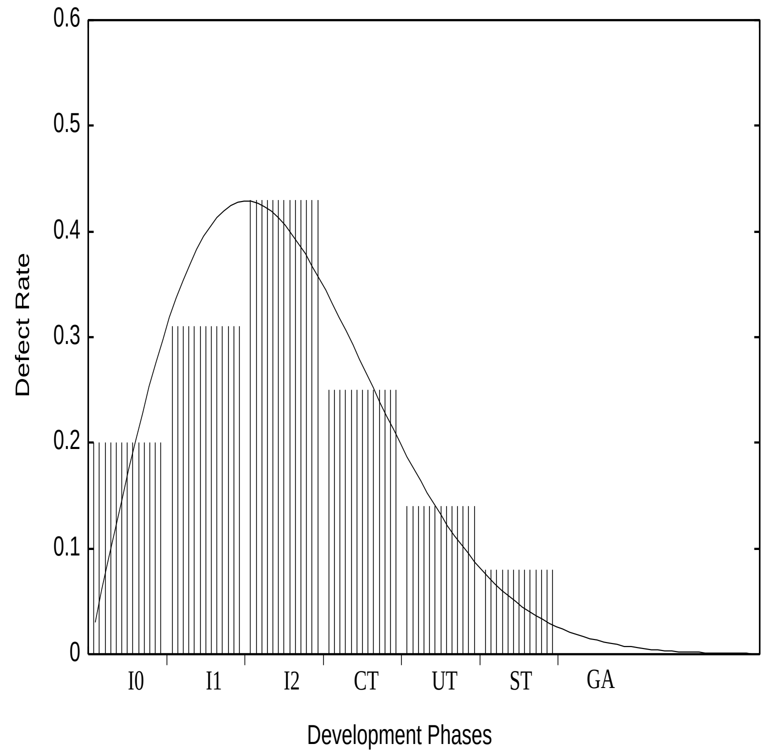


Figure 7: An example of application of the model to software development.

Given the defect removal pattern, the goal is to estimate the defect rate when the product is shipped, which is called here *general-availability phase (GA)*.

□ Basic Two Assumptions of the Rayleigh Model [16]

1. Defect rate observed during the development is positively correlated with the defect rate after release

   $$\Downarrow$$

   The higher the curve (the more area under it) the higher defect rate after release and vice versa, assuming the defect removal effectiveness remains unchanged.

   Exercise 2.

   > *Draw the two curves of the Equation (43) with different K values to show this assumption 1. Note that it is important to show the difference of the tail of the two curves.*

2. Given the same error injection rate, if more defects are discovered and removed earlier, fewer will remain in later stages.

   Exercise 3.

   > *Draw two curves of the Equation (43) the area under which are the same but peak at varying locations. to show the curve that peaks earlier has smaller area at the tail.*

[16]Both assumptions give us the *Do-it-right-the-first-time* principle, that is, if errors are to be injected, they should be removed as early as possible.

Chapter 5

Reliability Growth Model

Contents:

□ Reliability Growth Model

- While *Rayleig Model* models defects during entire process, *Reliability Growth Models* are based on data from formal testing phses.
- Defect arrival or failure patterns during these test phases is a good indicator of reliability of the product after release.
- when fails occur and defects are identified and fixed the product becomes more stable and reliability **grows** over time, which is the reason of the name of the model.

□ The Exponential Model

— Weibull model with $m = 1$.

- Its pdf form:

$$f(t) = \frac{1}{c} \exp\big(- (t/c)\big). \tag{44}$$

- cdf form:

$$F(t) = 1 - \exp\big(- (t/c)\big). \tag{45}$$

- Graph of its pdf and cdf:

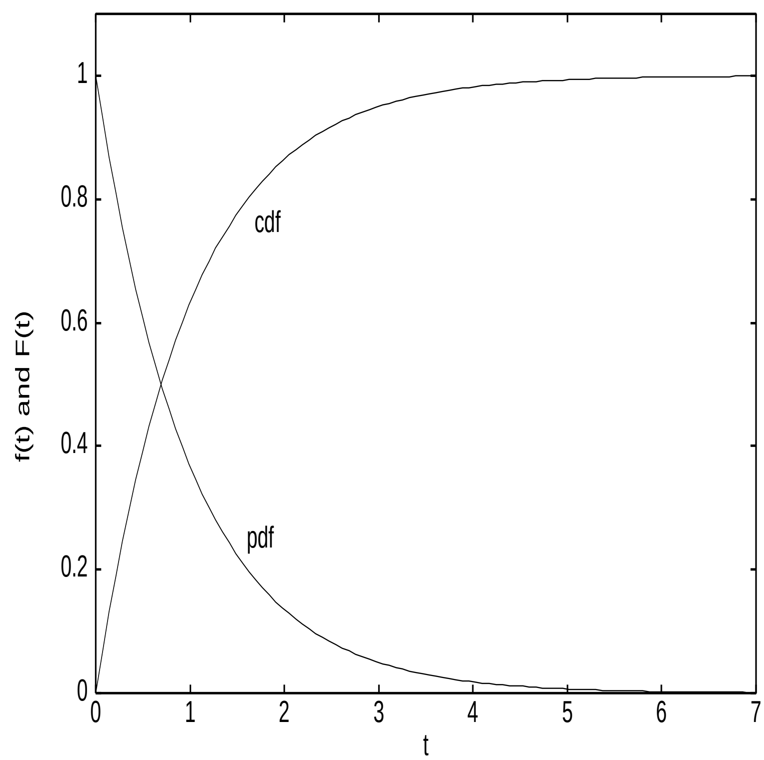


Figure 8: Exponential pdf and cdf.

- $1/c$ is referred to as the *error detection rate*, *instantaneous failure rate*, or *hazard rate.*

- Decline nonmonotonically to an asymptote, (but never reach it.)

- The area under the curve is 1, i.e.,

$$\int_{-\infty}^{\infty} f(t)dt = 1. \tag{46}$$

- It has long been successfully used for the analysis of failure data, such as:
    - · bank statement and ledger errors;
    - · payroll check errors;
    - · light bulb failure;
    - · automatic calculating machine failure;
    - · radar failure; and so on.

- The importance of the exponential distribution in reliability studies is analogous to the Gaussian pdf in Statistics.

- The model in Software Engineering:
  - Misra (1983) used the model to estimate the defect-arrival rates for Shuttle Ground System software of NASA.

  - Other examples:
    - · Defect-arrival data since the start of system test versus week when the development work was virtually complete.

    - · The pattern of defect arrivals during the test
    — indicative of the *latent-defect rate after release.*

  - When apply the Model:
    - · The more accurate the input data, the better the outcome.
    - · The more data points available, the better the model fit to the data.

- The basic assumption:

  The testing effort (e.g., person-hour) is homogeneous throughout the testing phase.

Normalize if not!

$$\Downarrow$$

1. Accumulate the total person-hours in testing for the entire testing phase, and calculate the average number of person-hours in testing per week $\Rightarrow n$.

2. Starting from the beginning of testing, calculate the defect rates for each n person-hour units.

3. Allocate the defect rate observed for the first n person-hours of testing to the first week and so on.

□ The Other Reliability Growth Models[17]

- Two major classes (according to the dependent variables)

  1. The time between failures model:

     * The parameters of the distribution of time between consecutive failures should be related to the number of latent defects remaining.

     * The goal is to estimate meantime to next failure.

  2. The fault count model:

     * The number of faults in a specified time interval (CPU time, hours, days, week, or month.)

[17] Though more than a hundred models have been proposed, they are not so satisfactory, like

- the cost of gathering data is expensive;
- some are not understandable;
- some are not tested in practical environment;
- some do not work when tested.

- Examples of each classes:

1. Time between failure models:

   Estimation of the time between $(i-1)$ and i-th failure, assuming N defects at the beginning of testing.

∗ Jelinski-Moranda (J-M) Model (1972):

$$\{\text{time between failure}\} = K\{N-(i-1)\} \tag{47}$$

- under the assumptions:
  · failures occur purely at random
  · all faults contribute equally to cause a failure during the test
  · fix time is negligible
  · the fix is perfect.

∗ *Imperfect Debugging Model* (Göel-Okumoto, 1978)

$$\{\text{time between failure}\} = \{N-p(i-1)\}\lambda \tag{48}$$

- The assumption here is:
  · fix is not always perfect but new defects may be injected when fixing.

where p is the probability of imperfect debugging and λ is the failure rate per fault.

2. The fault count model:

∗ *Non-homogeneous Poisson Process Model*[18]

(Göel-Okumoto, 1979)

This model is:

· To model the number of failures observed in given testing intervals, i.e.,

· the cumulative number of failures observed at time t, $N(t)$, can be modeled as a Non-homogeneous Poisson Process, i.e.,

· the time dependent failure rate follows a exponential distribution.

The model is given as:

$$prob\{N(t) = y\} = \frac{\{m(t)\}^y}{y!} \exp\{-m(t)\} \qquad y = 0, 1, 2, \cdots \tag{50}$$

where

$$m(t) = a(1 - \exp(-bt)) \tag{51}$$

(to be continued)

[18] Poisson pdf gives a probability that an event whose average occurence is *lambda* per unit time occurs k times during the unit time, being given as:

$$prob(X = k) = \exp(-\lambda)\frac{\lambda^l}{k!} \tag{49}$$

is expected number of failures observed by time t in which a is the expected number of failures to be observed eventually, b is the fault detection rate per fault, and

$$\lambda(t) \equiv m'(t) = ab\exp(-bt) \tag{52}$$

means failure density.

It is easy to check $m(t)$ and $\lambda(t)$ is a *pdf* and *cdf* form of the exponential model, respectively, with $a = K$ and $b = \lambda$.

Göel extend this into more general form

$$\Downarrow$$

Generalized Non-homogeneous Poisson Process Model

$$m(t) = a(1 - \exp(-bt^c)) \tag{53}$$

$$\lambda(t) = abc\exp(-bt^c)t^{c-1} \tag{54}$$

where a is the expected number of faults to be eventually detected, and b and c are constants that reflect the quality of testing.

(2. The fault count model — continued:)

$*$ *Logarithmic Poisson Execution Time Model*
(Musa-Okumoto, 1983)

The model denotes:

- observed number of failures by a certain time τ;
- also a Non-homogeneous Poisson Process, but
- $m(t)$ has a different form.

$$m(t) = \frac{1}{\theta} \ln(\lambda\theta + 1) \tag{55}$$

where λ is the initial failure, and θ is rate of reduction in the normalized failure, i.e., intensity per failure.

- Under the assumption that fixes found later have a smaller effect on the software's reliability than the earlier ones.
- The logarithmic Model is superior when some functions are executed much more frequently than others.

(2. The fault count model — continued:)

∗ *Delayed S Model* (Yamada et al., 1983)

The assumption:

- Testing process consists of not only a defect detection but a defect isolation.
- Because of the time needed for failure analysis, delay can occur between the time of the 1st failure observation and a time of reporting.

Here, $m(t)$ is defined as:

$$m(t) = k\big(1 - (1 + \lambda t)\exp(-\lambda t)\big) \tag{56}$$

where λ is error detection rate, and k is the total number of defects.

∗ *Inflection S Model* (Ohba, 1984)

This models:

- a software failure detection phenomenon with a mutual dependence of detected defects.

(to be continued)

- The more failures we detect, the more undetected failures become detectable.

Here, $m(t)$ is defined as:

$$m(t) = k\frac{1 - \exp(-\lambda t)}{1 + i\exp(-\lambda t)} \tag{57}$$

where λ is error detection rate, k is the total number of defects, and i is inflection factor.

□ **Homework Assignment**

- For the three models we have learned so far, that is,
  - Exponential Model,
  - Delayed S Model,
  - Inflection S Model,

  compare the degree of fitness of them to a set of real data by the following way.

  ⋆ For each of the three models:

  (1) Fill the blanks of the table below. In calculating the column (C), you must adjust each of the parameters in the Model such that the model best fits the data of the column (B).[19]

Table 1: A Comparison of a Model to a Cumulative Data of a Weekly Defect Arrivals

week (x)	observed defect ratio (A)	observed defect cumulative (B)	model defect cumulative (C)	normalized data cumulative $F^*(x)$	normalized model cumulative $F(x)$	absolute difference between data and model $\lvert F^*(x) - F(x)\rvert$
1	.353	.353				
2	.436	.789				
3	.415					
4	.351					
5	.380					
6	.366					
7	.308					
8	.254					
9	.192					
10	.219					
11	.202					
12	.180					
13	.182					
14	.110					
15	.155					
16	.145					
17	.221					
18	.095					
19	.140					
20	.126			1.000	1.000	0

  (2) Draw the histogram of column(B) as well as the curve by the model[20] on one coordinate.

  (3) Judge the fitness by the *Kolmogorov-Smirnov* goodness of fit test, that is,

$$\text{if} \quad \max |F^*(x) - F(x)| < .294 \quad \text{then model fits the data.} \tag{58}$$

  The answer hopefully should be positive.[21]

---

[19] The judgment of the fitness depends on your eyes and therefore this column should be unique one. Do not use a set of results by others. I won't believe any coincidence.

[20] You should draw the graph of the equation (not by giving the points from column (C).

[21] This means the answer is something like "Yes, the model fits the data because $\max |F^*(x) - F(x)| = .262 < .294$."

Chapter 6

Complexity vs Quality

Contents:

□ Complexity vs Quality

This chapter studies relationships between metrics about the design and code implementation and the software quality.

- The simplest Model:
  - Assuming LOC reflects complexity.
    - ⋆ The larger the size, the more defects?

$$\Downarrow$$

No!

- · Defects were mostly in the small module (Basili et al, 1984).
- · The larger module contains small defect rate, probably because of interface error which are more or less constant regardless of size (Shen et al., 1985).

  $$\downarrow$$

  The smaller modules has higher error density due to small denominator.
- · Defect density decreases with size and then up again at the tail (Withrow, 1990).

  $$\downarrow$$

  There might be an optimal program size.

- Structural Metrix and Models

We use:

– fan-in: # of modules that call a given module.
– fan-out: # of modules that are called by a given module.

⋆ The number of modules with a large fan-out have a positive correlation with the number of defect, while the number of modules with a large fan-in have a negative or insignificant correlation?

(1) Henry et al.'s Structure Complexity (1981):

$$\{\text{structure complexity}\} = (\{\text{fan-in}\} \times \{\text{fan-out}\})^2 \tag{59}$$

(2) Card and Glass's System Complexity (1990)

$$\{\text{system complexity}\} = (\{\text{structural intermodule complexity}\} + (\{\text{data intramodule complexity}\} \tag{60}$$

Here, each term of the RHS is defined as:

$$\text{structural intermodule complexity} = \frac{\Sigma\, D(i)}{n} \tag{61}$$

and

$$\text{data intramodule complexity} = \frac{\Sigma\, f^2(i)}{n} \tag{62}$$

where

- $D(i)$ is data complexity of module i defined as:

$$D(i) = \frac{V(i)}{f(i)+1} \tag{63}$$

- $V(i)$ is I/O variables in module i,
- $f(i)$ is fan-out of module i,
- n is the number of (new) modules in system.

Then model says:

- Fan-in is not an important complexity.
- Complexity increases as the square of fan-out.
- The more I/O variables in a modules the more functionality is needed, hence the higher internal complexity.
- "more fan-out" means that functionality is deferred to modules or lower levels, hence the less internal complexity.

And finally the Model give an equation by regression formula:

$$\{\text{error rate}\} = -5.2 + 0.4 \cdot \{\text{system complexity}\} \tag{64}$$

Chapter 7

Customer's Satisfaction

— Measurement and Analysis

Contents:

□ Product Quality + Customer Satisfaction = Total Quality

- Studies show:
  - · it is five times more costly to recruit a new customer;
  - · dissatisfied customers tell 7 to 20 people about their experience, while satisfied customers tell only 3 to 5.

□ How can We Measure the Customer Satisfaction?

- By interview, phone, or mail?

- The five-point satisfaction scale:

  (1) very satisfied
  (2) satisfied
  (3) neutral
  (4) dissatisfied
  (5) very dissatisfied

- To obtain representative samples:
  - Simple Random Sampling:
    * Every possible sample of size n has the same chance of being selected, say, by using random numbers.

  - Systematic Sampling:
    * Simply goes down a list taking every k-th individual where

$$k = \frac{\{\text{total number of items in the list}\}}{\{\text{number to be drawn}\}} \tag{65}$$

(to continue)

- Stratified Sampling
  - First classify individuals into non-overlapping groups *(strata)*, and then select simple random sample from each stratum [22].
  - It can be designed to yield greater accuracy for the same cost or for the same accuracy with less cost.
  - Proportionnal vs Dispropotional:
    - The former being the sampling frachions within each stratum is equal, while the latter being different when compose different subpopulations that differ in size.
  - To yield the maximum efficiency:
    - Decide sampling fraction for each stratum proportional to the standard deviation within the stratum and inversely proportional to the square root of the cost of each case within the stratum.

(to continue)

[22] E.g., network-users and stand-alone-usrs.

- How large is a sample sufficient?
  - It depends on sampling method, the level of confidence and margin of error.
  - A proposition for the simple random sampling,

$$n = \frac{NZ^2p(1-p)}{NB^2 + z^2p(1-p)} \tag{66}$$

where N is population, p is estimated satisfaction level[23], B is margin of error, and

$$\begin{aligned} z &= 1.28 \quad \text{for 80\% confidence level} \\ z &= 1.45 \quad \text{for 85\% confidence level} \\ z &= 1.65 \quad \text{for 90\% confidence level} \\ z &= 1.96 \quad \text{for 95\% confidence level} \end{aligned}$$

  - The required sample size decreases as customer satisfaction increases, i.e., the larger the p the smaller its variance $p(1-p)$.

[23] when no estimate of satisfaction available use $p = 0.5$ which require the largest sample because pq takes the largest value.

□ Specific Attributes vs Overall Satisfaction

CUPRIMDA[24] vs Overall Satisfaction:

- A Survey[25]

  1. Degree of Satisfaction

Overall	U	P	R	I	M	D	A
.855	.841	.911	.938	.753	.829	.733	.945

Table 2: An example of Degree of Satisfaction

  2. Correlation

	Overall	U	P	R	I	M	D	A
U	.61							
P	.43	.46						
R	.63	.56	.42					
I	.51	.57	.39	.47				
M	.40	.39	.31	.40	.38			
D	.45	.51	.34	.44	.45	.35		
A	.38	.39	.52	.46	.32	.28	.31	

  · Those who are happy with R, U, or I are happy with the overall product.

  · Those who are happy with R, I, or D are happy with U.

  · Those who are happy with A are happy with P.

[24] Each capital is abbreviation from Capability, Usability, Performance, Reliability, Installability, Maintainability, Documentation, and Availability, respectably. These eight attributes are often used in software engineering.

[25] Data are from IBM Rochester whose sample size is 3,658.

3. Coefficient of Least Squares Regression[26]

R	U	I	P	M	D	A
.391	.247	.091	.070	.067	.056	.022

- Each of the x_i is the expression of RUIPMDA expressed by the 5-point scale. The coefficients are re-arranged according to the order of magnitude and mean its importance.

[26]Coefficients of the following equation

$$y = a_0 + a_1x_1 + a_2x_2 + \cdots + +a_kx_k + \epsilon \tag{67}$$

where ϵ is error term are determined such that the total sum of the distance from data points to the hyper-line (67) minimizes.

4. Logistic Regression

x: dichotomous variable, i.e.,

- satisfied: {very-satisfied}+{satisfied} $\Rightarrow 1$
- dissatisfied: {neutral} + {very-satisfied} + {satisfied} $\Rightarrow 0$

and

$$\log\left(\frac{\{\text{satisfied}\}}{\{\text{disstisfied}\}}\right) = b_0 + b_1x_1 + b_2x_2 + \cdots + b_kx_k + \epsilon \tag{68}$$

- *Odds ratio* of satisfied and dissatisfied:

How much more likely customers will be positive in overall satisfaction if they were satisfied with specific UPRIMDA vs if they were not.

	R	U	A	I	M	P	D
b_k	1.216	.701	.481	.410	.376	.321	.164
Odds ratio	11.4	4.1	2.6	2.3	2.1	1.9	1.4

Table 3: Logistic Regression Coefficient and Odds

For example we can read this table that the overall satisfaction occurs 11.4[27] times as often among those satisfied than those dissatisfied.

[27] The value greater than 1 has a positive effect.

5. The probability for customers being satisfied depending on whether or not they are satisfied with CPRIMDA:

$p\{y=1\|x\}$	U	P	R	I	M	D	A
.9632	1	1	1	1	1	1	1
.9187	1	1	1	1	1	1	0
.9552	1	1	1	1	1	0	1
.9331	1	1	1	1	0	1	1
.9287	1	1	1	0	1	1	1
.7223	1	1	0	1	1	1	1
.9397	1	0	1	1	1	1	1
.8792	0	1	1	1	1	1	1
.0189	0	0	0	0	0	0	0
.0480	0	0	0	0	0	0	1
.0260	0	0	0	0	0	1	0
.0392	0	0	0	0	1	0	0
.0132	0	0	0	1	0	0	0
.1796	0	0	1	0	0	0	0
.0353	0	1	0	0	0	0	0
...				...			

6. Regression coefficient vs Satisfaction

- To determine priority of improvement:

(1) Plot each CPRIMDA on the coordinate coefficient vs satisfaction level.

(2) Pick up item by going from top to bottom.

(3) If coefficients have the same level do it by going left to right.

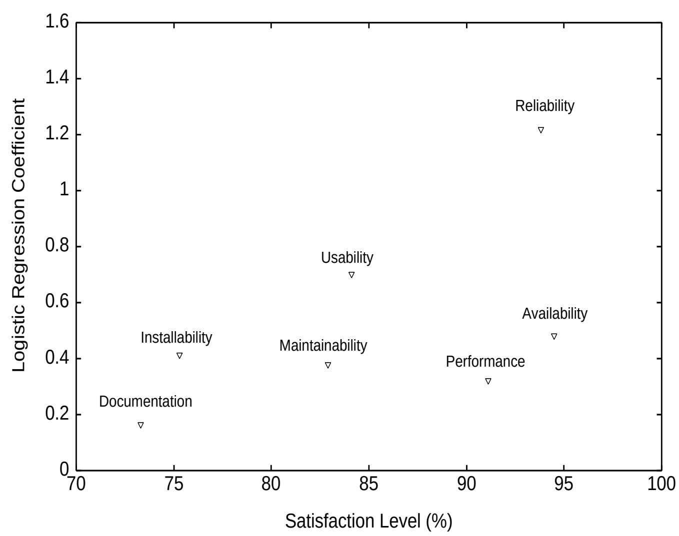


Figure 9: An example of Regression coefficient vs Satisfaction. The data in Table 2 and 3. are used.

□ How good is Good Enough?[28]

- Though the long term goal should be of course 100 %, there might be better strategy, like:
  - Should we invest \$2,000,000 to improve satisfaction from 85% to 90 %?
  - When our customer's satisfaction is 95 %, should we invest another million dollar or should we do it later?

- A model for prediction of future market share:

$$A_{t+1} = A_t(1-x) + B_t y \frac{A_t}{A_t + C_t} + C_t z \frac{A_t}{A_t + B_t} + G \frac{A_t}{A_t + B_t + C_t} \tag{69}$$

where $A_t(B_t,C_t)$ is the number of A(B, C)'s customers, at time t, G is the number of new customers to market, and $x(y,z)$ is dissatisfaction level with A(B,C)'s products.

B_t and C_t is also defined in the similar way.

An example of the plot of A_t, B_t and C_t with different satisfaction level is those in the next page.

Hence

You must be better than your competitor!

[28] This title is from the book by S.H.Kan who cited Bebich (1992).

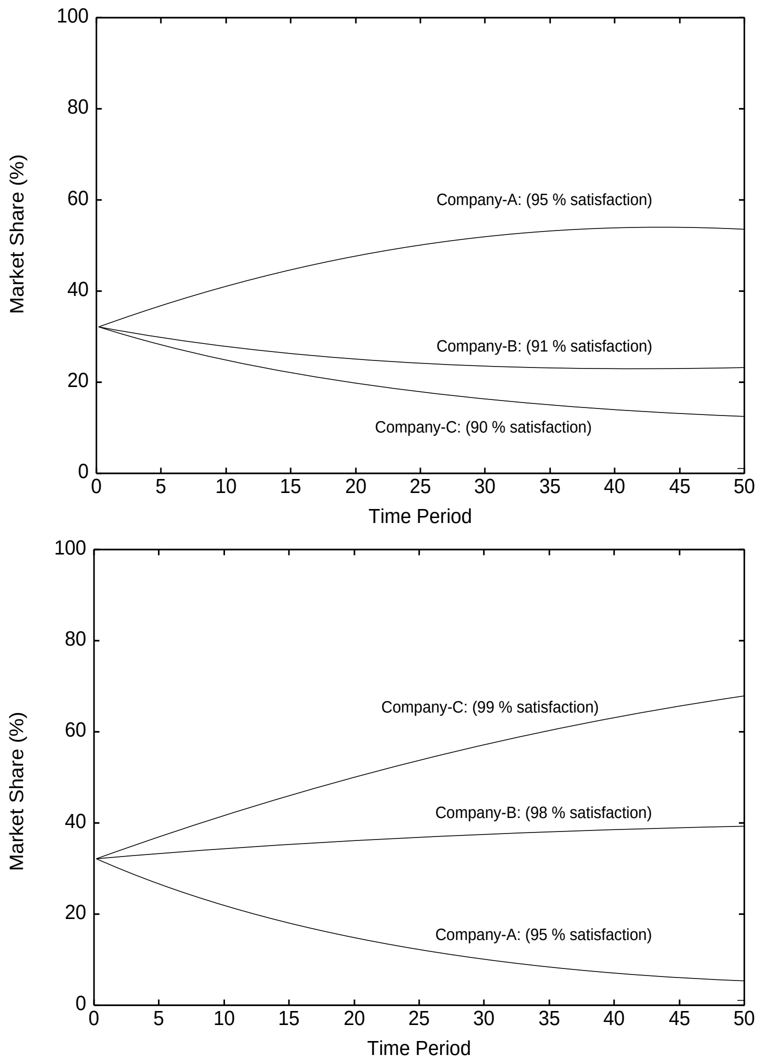


Figure 10: How the customer's satisfaction affects market share?

$\star$ This concluses the course of this semester.
$\star$ Thank you very much for your interest and attendance.
$\star$ See you.