Metrics

The term *metrics refers to any measurement related to software development.*

- *Lines of Code*
- *number of defects*
- *defects per thousand lines of code*
- *defects per module*
- *number of global variables*
- *hours to complete a project*

Any measurable aspect of the software development project.

cont.

"Any way of measuring the process is superior to not measuring it at all."

(*Code Complete, A Practical Handbook of Software Construction, by Steve McConnell, Microsoft Press)*

It gives you a handle on your software-development process that you don't have without it.

To argue against metrics is to argue that it's better not to know what's really happening on your project.

Useful Metrics : Size

- Total lines of code written
- Total comment lines
- Total data declarations
- Total blank lines

Useful Metrics: Productivity

- Work-hours spent on the project
- Work-hours spent on each routine
- Number of times each routine changed
- Dollars spent on project
- Dollars spent per line of code
- Dollars spent per defect

Useful Metrics: Defect Tracking

- Severity of each defect
- Location of each defect
- Way in which each defect is corrected
- Person responsible for each defect
- Number of lines affected by each defect correction
- Work hours spent correcting each defect
- Average time required to find a defect
- Average time required to fix a defect
- Number of attempts made to correct each defect.
- Number of new errors resulting from defect correction

Useful Metrics: Overall Quality

- Total number of defects
- Number of defects in each routine
- Average defects per thousand lines of code
- Mean time between failures
- Compiler-detected errors

Useful Metrics: Maintainability

- Number of parameters passed to each routine
- Number of local variables used by each routine
- Number of routines called by each routine
- Number of decision points in each routine
- Control-flow complexity in each routine
- Lines of code in each routine
- Number of data declarations in each routine
- Number of blank lines in each routine
- Number of *gotos* in each routine
- Number of input/output statements in each routine

Collecting Measurements

- Use software tools that are currently available.
- Measurements are useful mainly for identifying routines that are “outliers”; abnormal metrics in a routine are a warning sign that you should re-examine that routine, checking for unusually low quality.
- Don't start by collecting data on all possible metrics! you'll bury yourself in data.

cont.

- Standardize the measurements across your projects, and then refine them and add to them as your understanding of what you want to measure improves.
- Make sure you're collecting data for a reason.
- “You need to define measurement goals before you measure.” (NASA Software Engineering Laboratory, 1989)

Pressman

"Software metrics let you know when to laugh and when to cry " *Tom Gilb*

" Not everything that can be counted counts, and not everything that counts can be counted " Albert Einstein

Software Metrics Etiquette

- Use common sense and organizational sensitivity when interpreting metrics data.
- Provide regular feedback to the individuals and teams who collect measures and metrics.
- Don't use metrics to appraise individuals.
- Work with practitioners and teams to set clear goals and metrics that will be used to achieve them.
- Never use metrics to threaten individuals

Etiquette cont.

- Metrics data that indicate a problem area should not be considered “negative.” These data are merely an indicator for process improvement.
- Don't obsess on a single metric to the exclusion of other important metrics.

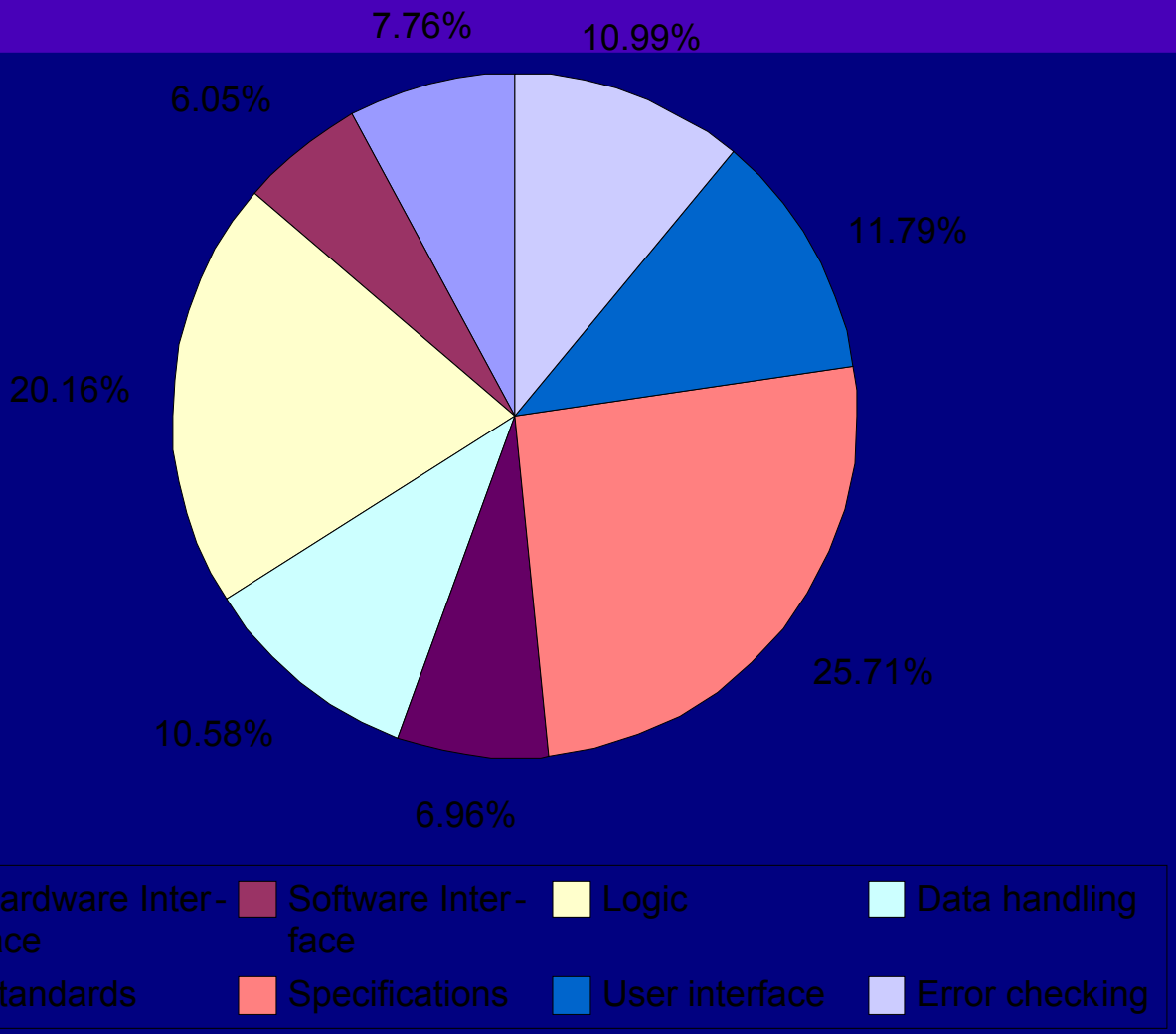
SSPI
Statistical Software Process Improvement

- Failure analysis works in the following manner.
  - All errors and defects are categorized by origin (e.g. flaw in specification, flaw in logic, nonconformance to standards.)
  - The cost to correct each error and defect is recorded.
  - The number of errors and defects in each category is counted and ranked in descending order.
  - The overall cost of errors and defects in each category is computed.

cont.

- Resultant data are analyzed to uncover the categories that result in highest cost to the organization.
- Plans are developed to modify the process with the intent of eliminating (or reducing the frequency of) the class of errors and defects that is most costly.

Causes of defects and their origin for four software projects



Project Metrics

Project metrics and the indicators derived from them are used by a project manager and a software team to adapt project work flow and technical activities.

- Estimation
- Production rates
  - pages of documentation
  - review hours
  - function points
  - delivered source lines

Project Metrics Intent

- Used to minimize the development schedule by making the adjustments necessary to avoid delays and mitigate potential problems and risks.
- Used to assess product quality on an ongoing basis and, when necessary, modify the technical approach to improve quality.

cont.

- Every project should measure:
  - Inputs – measures of the resources (e.g. people, environment) required to do the work.
  - Outputs – measures of the deliverables or work products created during the software engineering process.
  - Results – measures that indicate the effectiveness of the deliverables.

Software Measurement

- Direct Measures
  - process
    - cost
    - effort applied
  - product
    - loc produced
    - execution speed
    - memory size
    - defects reported over some set period of time

Software Measurement cont.

- Indirect Measures
  - product
    - functionality
    - quality
    - complexity
    - efficiency
    - reliability
    - maintainability
    - ...

These indirect measures are harder to collect.

Size-Oriented Metrics

- Errors per KLOC (thousand lines of code)
- Defects per KLOC
- $ per LOC
- Page of documentation per KLOC
- Errors per person-month
- LOC per person-month
- $ per page of documentation

Function Oriented Metrics

- Number of user inputs.
- Number of user outputs.
- Number of user inquiries.
- Number of files.
- Number of external interfaces.

Complexity Adjustment Values

- Does the system require reliable backup and recovery
- Are data communications required?
- Are there distributed processing functions?
- Is performance critical?
- Will the system run in an existing, heavily utilized operational environment?
- Does the system require on-line data entry?
- Does the on-line data entry require the input transaction to be built over multiple screens or operations?
- Are the master files updated on-line?
- Are the inputs , outputs, files, or inquiries complex?

cont.

- Is the internal processing complex?
- Is the code designed to be reusable?
- Are conversion and installation included in the design?
- Is the system designed for multiple installations in different organizations?
- Is the application designed to facilitate change and ease of use by the user?

... / function point

- Errors per FP
- Defects per FP
- $ per FP
- Pages of documentation per FP
- FP per person-month

LOC/FP (average)

Assembly language	320
C	128
COBOL	106
FORTRAN	106
Pascal	90
C++	64
Ada95	53
Visual Basic	32
Smalltalk	22
Powerbuilder	16
SQL	12

Measuring Quality

- **Correctness**
  measure: defects/kloc
- **Maintainability**
  measure: mean-time-to-change (MTTC)
  spoilage – the cost to correct defects
- **Integrity**
  threat : probability that an attack of a specific type will occur within a given time.
  security: probability that the attack of a specific type will be repelled.
  integrity = summation[(1-threat) x (1-security)]

Measuring Quality cont.

- Usability
  - the physical and or intellectual skill required to learn the system.
  - the time required to become moderately efficient in the use of the system
  - the net increase in productivity measured when the system is used by someone who is moderately efficient
  - a subjective assessment of users attitudes toward the system.

Defect Removal Efficiency

DRE = E/(E+D)

E = Number of errors found before delivery
D = Number of errors found after delivery

Establishing A Software Metrics Program

- Identify your business goals.
- Identify what you want to know or learn.
- Identify your subgoals.
- Identify the entities and attributes related to yoru subgoals.
- Formalize your measurement goals.
- Identify quantifiable questions and the related indicators that you will use to help you achieve your measurement goals.

cont.

- Identify the data elements that you will collect to construct the indicators that help answer your questions.
- Define the measures to be used, and make these definitions operational.
- Identify the actions that you will take to implement the measures.
- Prepare a plan for implementing the measures.

More lists of quality measures

Correctness – satisfies its specification
Reliability – perform with required precision
Efficiency – Amount of computing resources required.
Integrity – Extent to which access by unauthorized persons can be controlled.
Usability – Effort required to learn, operate, prepare input, and interpret output.
Maintainability – Effort required to locate and fix an error.

cont.

Flexibility – Effort required to modify an operational program.

Testability – Effort required to test a program and ensure that it performs its intended function.

Portability – Effort required to transfer the program from one system to another.

Reusability – Extent to which a program can be reused in other applications.

Interoperability – Effort required to couple one system to another.

McCall checklist 0(low) – 10(high)

- Auditability – ease with which conformance to standards can be checked.
- Accuracy – precision of computations and control.
- Communication commonality – degree to which standard interfaces, protocols, and bandwidth are used.
- Completeness – degree to which full implementation of required function has been achieved.

cont.

- Conciseness – compactness of the program in terms of loc
- Consistency – use of uniform design and documentation techniques throughout the software development project.
- Data commonality – use of standard data structures and types throughout the program.
- Error tolerance – damage that occurs when the program encounters an error.

cont.

- Execution efficiency – the run-time performance of a program.
- Expandability – the degree to which architectural, data, or procedural design can be extended.
- Generality – breadth of potential application of program components.
- Hardware independence – degree to which the software is decoupled from the hardware on which it operates.

cont.

- Instrumentation – degree to which the program monitors its own operation and identifies errors that do occur.
- Modularity – the functional independence of program components.
- Operability – ease of operation.
- Security – availability of mechanisms that control or protect programs and data.
- Self-Documentation – degree to which the source code provides meaningful documentation.

cont.

- Simplicity – degree to which a program can be understood without difficulty.
- Software system independence – degree to which the program is independent of nonstandard programming language features, operating system characteristics, and other environmental constraints.
- Traceability – ability to trace a design representation or actual program component back to requirements.

cont.

- Training – degree to which the software assists in enabling new users to apply the system.