Measurement & metrics

S/W Project

S/W project
- size
- technology
- delivery deadlines
- costs
- user requirements
- environment
- people

* Risk is inversely proportional to technology maturity.
* use metrics to improve S/W quality.

Measurement & Metrics ⇒ Total Quality Management (TQM) connection

- TQM → continuous process improvement
- "Kaisen" → the use of measurement isolate common defects and remove them
- S/W metrics → a basis for Kaisen

A good manager measures:

focus on process → measurement; " " product → measurement

measurement → ~~(Hard) data~~ project data for productivity measurement

measurement → technical data for quality measurement

measurement → normalized data → normalize using:

what do we use as a basis? size? function?
- size → (defects/KLOC)
- function → function point

* Staff experience has a major impact.
* Irrational schedules force mistakes and lower quality
* unstable requirements

Metrics for the project:

normalize:
- use LOC or function points (FP) as a basis for ~~exp~~ normalization
- examine effort applied (in work units)
- examine deliverables produced
- examine after-the-fact quality

collecting project (Hard) data:
- number of people assigned to a project
- effort (person-months) expended on project tasks
- duration (calender units) for each project tasks
- volume of documentation, source code, test cases
- number of defects found and reported

Hard Technical metric example: – cyclomatic complexity or McCabe complexity metric
- (measures algo. complexity of a program component.)
- (correlates to errorproneness of a module)

* dedicate testing resources to those modules with high cyclomatic complexity.

collecting ~~support~~ support data (Soft):

collecting support (soft) data: (why?)

- business constraints on the project
- skill of the project team
- experience of the project manager
- stability of requirement
- difficulty of the problem to be solved
- adequacy of the software engineering environment (methods and tools)
- user satisfaction with end-product
- value of the software to the business

Normalization: Size Vs Function:

* LOC "rewards" verbose programs
* LOC does not accomodate 4GLs particularly well.
* LOC is difficult to use in mixed language applications.
* LOC can present problems when reuse is applied.

(Size)
* the LOC measure ← most widely used
* a physical artifact of the software engineering process

(the function point measure)
* independent of programming language.
* uses readily countable characteristics.
* does not 'penalize' short programs.
* makes it easier to accommodate reuse.

Function points Computation: (FP):

- develop a count of information domain values
- assess complexity of the counts
- compute raw FP number
- determine overall complexity of application
- adjust raw FP number to compute function point value

$$FP = \text{raw FP count} \times \left[0.65 + \left(0.01 \times \sum_i CF_i\right)\right]$$

~~value~~ value

Information domain values

- output (OT) – data items produced for the external world.
- input (IT) – data items from external world to software
- inquiries (QT) – inputs that cause some database lookup and response
- files (FT) – externally observable data stores
- interfaces (EI) – connections to other systems or databases

domain value complexity

	Simple	Average	Complex
output	4	5	7
input	3	4	6
inquiry	4	5	7
files	7	10	15
interface	5	7	10

complexity multipliers

* A variation for FP: "feature point"
- it adds new count one "algorithm"
- it is computed the same way as FP

Computing raw values:

raw FP values =

$$(OT_{simple} \times 4) + (OT_{avg.} \times 5) + (OT_{complex} \times 7) +$$
$$(IT_{simple} \times 3) + (IT_{avg.} \times 4) + (IT_{complex} \times 6) +$$
$$(QT_{simple} \times 4) + (QT_{avg.} \times 5) + (QT_{complex} \times 7) +$$
$$(FT_{simple} \times 7) + (FT_{avg.} \times 10) + (FT_{complex} \times 15) +$$
$$(EI_{simple} \times 5) + (EI_{avg.} \times 7) + (EI_{complex} \times 10)$$

Taking algorithms into account: Feature Points:

		weight	
number of user inputs	—	× 4	= —
no of " outputs	—	× 5	= —
" " " inquiries	—	× 4	= —
" " files	—	× 7	= —
" " interface	—	× 7	= —
algorithms	—	× 3	= —

— count-total

— complexity multiplier

FP (feature points)

~~Algorithm: any specific method for solving a certain kind of problem~~

For the purposes of feature point, algorithm
- is a set of logical and computational steps that solves a specific, bounded problem or sub-problem.
- has input (or a start value): produces output.
- often corresponds to a cohesive module in a well designed program.

Taking complexity into account:

(Factors are rated on a scale of 0 (not important) to 5 (very important)):

data communications	on-line update
distributed functions	complex processing
heavily used	installation ease
transaction rate	operational ease
on-line data entry	multiple sites
end user efficiency	facilitate change

Complexity factors, CF_i

Computing Function Points by "Backfiring":

	statements per FP
Assembly language	→ 320
C	128
Fortran	105
COBOL	105
Pascal	91
Ada	71
LISP	64
O-O language	29
4GL	20

ex.
34 000 lines of FORTRAN
3650 lines " assembly
8400 " " C

$$\text{total function points} = \frac{34000}{105} + \frac{3650}{320} + \frac{8400}{128}$$

adjust total by complexity factor 0.70 (very low) and 1.30 (very high)

Productivity metrics:

- effort (person-months)/nv* → *nv = KLOC or FP
- cost/nv
- calender months/nv
- product attribute/nv
  - * pages of documentation/nv
  - * test cases/nv
  - * classes/nv
- people/nv

Factors impacting productivity & quality:

1. inexperienced staff
2. irrational schedules
3. inexperienced managers
4. unstable requirements
5. poor S/w engg. methods
6. no inspections
7. perfunctory testing
8. no measurement
9. low design reuse
10. low code reuse

} Process factors

Quality Metrics:

* the quality of product:
  - after-the-fact quality metrics focus on reported defects.
  - technical metrics focus on S/w design attributes.
* the quality of process:
  - process maturity — computed using a process assessment method.
  - defect removal metrics — computed using information about errors and defects.

ex. (A quality metrics)

E = total errors found before delivery
D = " defects " after "

$$\text{defect removal efficiency, } DRE = \frac{E}{E+D}$$

Balancing Agility and discipline

hybrid agile and plan-driven methods provide different but highly ~~ef~~ effective balances of agility and discipline to fit their unique situations.

A way, to plan your program and incorporate both agility and discipline in proportion to your project's needs,

The criteria, we develop are based on your project's particular risks with respect to the use of agile or plan-driven methods.

Risk-based method (5-steps method)

1. Risk analysis: (Rate project's environmental, agile, and plan-driven risks. If uncertain about ratings, buy information via prototyping, data collection, and analysis.)
2. Risk comparision: (compare risks (agility risks and plan-driven risks) and go with lower risks method.
3. Architecture analysis: (if confusion, architect application to encapsulate agile parts. ~~and go~~ Go risk-based agile in agile parts and risk-based plan-driven elsewhere.)
4. Tailor life cycle: (establish an overall project ~~dev~~ strategy by integrating individual risk mitigation plans.)
5. execute and monitor: (monitor progress and risks/opportunities, readjust balance and process as appropriate.)

B/P six conclusions:

1. neither agile nor plan-driven methods ~~pro~~ provide a silver bullet.
2. Agile and plan-driven methods have home grounds where one clearly dominates the other.
3. Future trends are towards application developments that need both agility and discipline.
4. Some balance methods are emerging.
5. It is better to build your method up ~~that~~ than to tailor it down. (starting with minimum sets of practices and only adding extras where it can be clearly justified by cost-benefit.)
6. methods are important, but potential silver bullets are more likely to be found in areas dealing with people, values, communications, expectations management.

value-proposition about proposed S/W system.

(value-based S/W engg.)