Programmer Productivity

- A measure of the rate at which individual engineers involved in software development produce software and associated documentation.
- Measure some attribute of the software and divide by the effort required for development. Essentially, we want to measure useful functionality produced per time unit
- Required for project estimation and to assess if improvements are effective.

Programmer Productivity

- Research has found variations up to a factor of ten.
- Is the programmer who produces 10 times the amount of code for a problem more productive?
- Great programmers often write less code.
- If productivity is measured in volume of code generated, some programmers will optimise that measure i.e. by writing more lines of code than are needed.

Productivity Measures

- Size related measures based on some output from the software process. This may be lines of delivered source code, object code instructions, documentation.
- Function-related measures based on an estimate of the functionality of the delivered software. Function-points and object points are the best known of this type of measure.

The Meaning of Productivity

- In a manufacturing environment :
  - Units produced/person hours
- Software engineers define productivity in terms of a measure:
  - Productivity = $\frac{\text{size}}{\text{effort}}$
- Size is usually measured as lines of code and effort is measured in person days or months.

Lines of Code

- Count the number of lines of source code delivered.
- Divide by the number of programmer months required to complete the project.
- Includes time for analysis, design, coding, testing and documentation.

Lines of Code

- What's a line of code?
  - The measure was first proposed when programs were typed on cards with one line per card.
  - How does this correspond to statements as in Java which can span several lines or where there can be several statements on one line.
- What programs should be counted as part of the system?
- Assumes linear relationship between system size and volume of documentation.

Cross-language Comparisons

- Productivity comparisons are impossible unless the same line counting technique is used and the same language.
- The lower level the language, the more productive the programmer
  - The same functionality takes more code to implement in a lower-level language than in a high-level language.
- The more verbose the programmer, the higher the productivity
  - Measures of productivity based on lines of code suggest that programmers who write verbose code are more productive than programmers who write compact code

Productivity Equation Problems

- Productivity = $\frac{\text{lines of code}}{\text{person days or months}}$
- For software development there are many different solutions. One may execute more efficiently, another may be more readable thus maintainable.
- Comparing production rates of different products is meaningless.
- No account is taken of the effective use of labour and resources.
- Effort is hard to measure
  - What is a day ? 8, 12 or 16 hours
  - How productive is a day ? Varies from person to person and day to day.
- Output is measured as size not as value – what value has been delivered.
- Problems with counting line of code

Complexity

- The best programmers are usually given the most difficult/complex sections of code to work on. Their productivity is usually very low compared to programmers who get easier assignments, but other programmers would take twice as long to do the work

Productivity Equation Problems

- This view of productivity treats software development to be much like any traditional manufacturing process. Problems with this are due to the fact that :
  - There is usually little or no replication in software development. When replication is required the cost of each copy is generally minimal in comparison with the cost of building the original. This small replication cost is a key feature that distinguishes software manufacture from more traditional manufacturing.
  - Creativity has an important role in software development. Poet productivity is not measured by the number of lines written divided by time spent writing.

System Development Times

	Analysis	Design	Coding	Testing	Documentation
Assembly code	3 weeks	5 weeks	8 weeks	10 weeks	2 weeks
High-level language	3 weeks	5 weeks	4 weeks	6 weeks	2 weeks

	Size	Effort	Productivity
Assembly code	5000 lines	28 weeks	714 lines/month
High-level language	1500 lines	20 weeks	300 lines/month

- Assembly Programmer has a productivity of 714 lines per month.
- High level Programmer has a productivity of 300 lines per month.
- Development times for high-level language system are lower and it costs less.

Programmer Productivity

- Due to problems with lines of code researchers have suggested :
- Productivity = $\frac{\text{function points implemented}}{\text{person days or months}}$
- Function points are language independent so productivity in different languages an be compared.
- Also an inefficient design/coding style won't generate more function points.

Function Point Analysis

- Based on a combination of system function categories. Estimate the number of the following:
  - external inputs and outputs
  - user interactions
  - external interfaces
  - files used by the system.
- A weight is associated with each of these based on complexity (simple, average, complex).
- The unadjusted function point (UFP) count is computed by multiplying each count by the weight and summing all values.
- UFC = ∑(number of elements of given type) X (weight)
- Function point count further modified by complexity of the project, 14 value adjustment factors (VAFs).

Complexity guidelines

		Weighting Factor			
measurement parameter count		**simple**	**average**	**complex**	
number of external inputs	x	3	4	6	=
number of external outputs	x	3	5	7	=
number of external inquiries	x	3	4	6	=
number of internal logical files	x	7	10	15	=
number of external interfaces	x	3	4	6	=

Unadjusted FPcount = total $\longrightarrow$ $\boxed{}$

FPA Example

External inputs

2 simple X 3 = 6
4 average X 4 = 16
1 complex X 6 = 6

External outputs

6 average X 5 = 30
2 complex X 7 = 14

External inquiries

8 average X 4 = 32

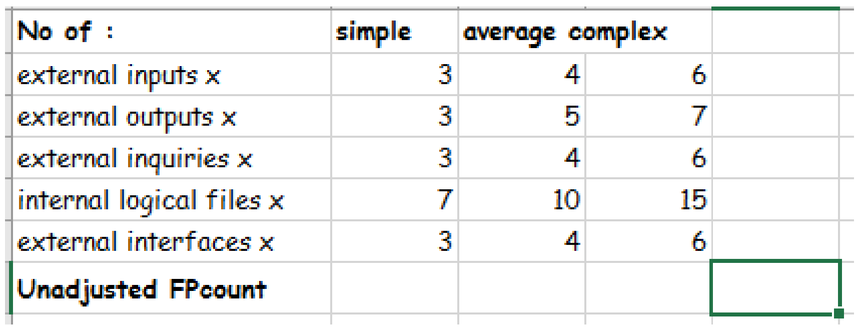
Internal logical files

5 complex X 15 = 75

External interfaces

7 simple X 3 = 21
10 average X 4 = 40

Unadjusted function points 240



No of :	**simple**	**average**	**complex**	
external inputs x	3	4	6	
external outputs x	3	5	7	
external inquiries x	3	4	6	
internal logical files x	7	10	15	
external interfaces x	3	4	6	
Unadjusted FPcount				

Value Adjustment Factors

In addition to these individually weighted function points, there are factors that affect the project and/or system as a whole. Rate each factor on a scale of 0 to 5 : 0 no influence, not present to 5 strong influence, essential

F_i

1. Does the system require reliable backup and recovery ?
2. Are data communications required ?
3. Are there distributed processing functions ?
4. Is performance critical ?
5. Will the system run in an existing, heavily utilised operational environment ?
6. Does the system require on-line data entry ?

Factors contd.

7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?
8. Are the master files updated on-line ?
9. Are the inputs, outputs, files or inquiries complex?
10. Is the internal processing complex ?
11. Is the code designed to be reusable ?
12. Are conversion and installation included in the design ?
13. Is the system designed for multiple installations in different organisations ?
14. Is the application designed to facilitate change and ease of use by the user ?

Function Point Analysis

- When the values of these 14 factors are summed a Total Degree of Influence (TDI) is derived.
- TDI can vary from 0 (when all are low) to 70 (when all are high) i.e. $0 \leq TDI \leq 70$.
- Value Adjustment Factor
  - **VAF =0.65 + (0.01 x TDI)**
- The constant values in the equation and the weighting factors that are applied to information domain counts are determined empirically.
  - Adjusted FP Count = Unadjusted FP Count * VAF

FPA Example

Continuing our example . . .

Complex internal processing	= 3
Code to be reusable	= 2
High performance	= 4
Multiple sites	= 3
Distributed processing	= $\underline{5}$
Project adjustment factor	= 17

Adjustment calculation:

Adjusted FP = Unadjusted FP X [0.65 + (adjustment factor X 0.01)]
= 240 X [0.65 + (17 X 0.01)]
= 240 X [0.65 + 0.17]
= 240 X [0.82]
= 197 Adjusted function points

Function Point Analysis

But how long will the project take and how much will it cost?

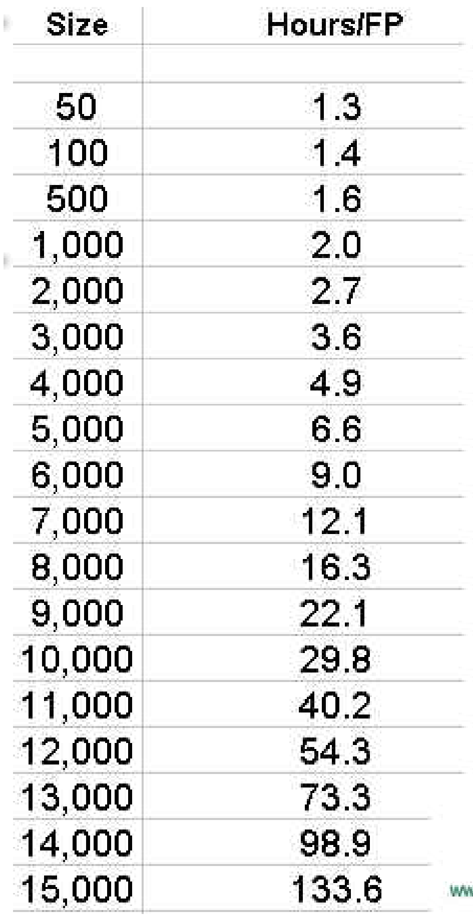
- If programmers average 18 function points per month.

  197 FP /18 = 11 person-months

- If the average programmer is paid €5,200 per month (including benefits), then the [labour] cost of the project will be . . .

  11 person-months X €5,200 = €57,200

Productivity



Size	Hours/FP
50	1.3
100	1.4
500	1.6
1,000	2.0
2,000	2.7
3,000	3.6
4,000	4.9
5,000	6.6
6,000	9.0
7,000	12.1
8,000	16.3
9,000	22.1
10,000	29.8
11,000	40.2
12,000	54.3
13,000	73.3
14,000	98.9
15,000	133.6

www.SoftwareMetrics.Com

Organisations should collect their own data, in it's absence this table can be used. The item that impacts productivity the most is the size of the project. It is interesting to note that nearly 75% of projects over 8,000 function points will fail and/or be cancelled.

Programmer productivity

- If function points do capture functionality, this measure is better because :
  - The function-points-based measure more accurately reflects the value of output.
  - It can be used to assess the productivity of software-development staff at any stage in the life cycle, from requirements capture onwards.
  - We can measure progress by comparing completed function points with incomplete ones.

Problems with Function Points

- FPA is subjective
  - When calculating UFP count.
  - When assessing technology factors.
- There are problems with double counting. It is possible to account for internal complexity twice in weighting the inputs for the UFP and again in the VAFs.
- FPA applies well only to systems, which are dominated by input-output and filing functions.
- Because function points largely ignore internal processing complexity, care must be taken in applying FPA to complex applications, even in the business area.
- Different companies will calculate function points slightly different, making comparisons questionable.
- Hard to map function points to individual work packages for individual programmer productivity.

Productivity Equation Problems

- A measure of productivity must have a more direct relationship with quality or utility and the input must be measured by more than effort or time as other resources are consumed.
- Ideally we want to relate measures of productivity to the quality of products.

Story-based Planning

- The planning game is based on a set of user stories that cover the functionality of the final system.
- The project team read and discuss the stories and rank them in order of the amount of time they think it will take to implement the story.
- Stories are assigned 'effort points' reflecting their size and difficulty of implementation.
- The number of effort points implemented per day is measured giving an estimate of the team's 'velocity'.
- This allows the total effort required to implement the system to be estimated.

Factors Affecting Productivity

Factor	**Description**
Application domain experience	Knowledge of the application domain is essential for effective software development. Engineers who understand a domain are likely to be the most productive.
Process quality	The development process used can have a significant effect on productivity.
Project size	The larger a project, the more time required for team communication. Less time is available for development so individual productivity is reduced.
Technology support	CASE tools etc. can improve productivity.
Working environment	A quiet working area with private work areas contributes to improved productivity.

Factors Affecting Productivity

- Large teams are likely to have a mix of abilities and experience resulting in average productivity.
- In small teams overall productivity is mostly dependent on individual aptitudes and abilities.
- Productivity varies dramatically across application domains:
  - Large, complex, embedded systems 30 LOC/pm
  - Straightforward, well understood systems, written in e.g. Java 900 LOC/pm

Quality and Productivity

- All metrics based on volume/unit time are flawed because they do not take quality characteristics into account e.g. reliability and maintainability.
- More means better !
- Productivity may generally be increased at the cost of quality.
- Measures do not take into account the possibility of reusing software or using code generators and other tools.

Exercise

- Should measured productivity be used by managers during staff appraisal ?
- If so what safeguards are necessary ?