Productivity for Software Estimators

Murali Chemuturi & Sarada Kaligotla

1 Introduction

Software estimation, namely, software size, effort, cost and schedule (duration) are often causing animated discussions among the fraternity of software estimators. Normally, it is the senior Project Leaders and project Managers who carry out this activity.

Software development consists of a few disparate activities needing specialized knowledge, namely, in Requirements Gathering, analysis, and Management; Software Design, Coding, Independent Verification and Validation, Rollout / Deployment / Installation & Commissioning. Each of these activities is carried out by a differently skilled person using different tools, having different complexities.

2. Productivity

Productivity is defined as the rate of output for given inputs. Productivity is expressed as "so many units of output per day" or "so many units or output per hour".

Productivity is also defined as the ratio of output to input.

For the context of this paper, Productivity is defined as the rate of producing some output using a set of inputs in a defined time unit.

3. Concerns with Software Size Estimation

The present scenario in the industry is that we have multiple measures, namely,

1. Function Points
2. Use Case Points
3. Object Points
4. Feature Points
5. Internet Points
6. Test Points
7. FPA mark II
8. Lines of Code
9. Etc.

There is no accepted way of converting software size from one measure to another.

One odd aspect of these measures is that the size is adjusted (increase or decrease) due to factors of complexity etc. A size is something that does not change. For example, a pound of cheese does not alter if the person weighing is less/more experienced, or the scale is either a mechanical scale or an electronic one – right?

Or the distance of one mile remains one mile if a young person is walking or an old man is walking or if it is a freeway or if it is a busy city street.

But the rate of achieving changes – an old man completes one mile slower than the younger one you go faster on a freeway than on a busy street.

There is no agreement on how to count Lines of Code – logical statements or physical statements, treatment of inline documentation.

These are some of the issues with size measurement.

4. Concerns with Productivity

The software development world is obsessed with giving one single, empirical, all-activities-encompassing figure for productivity.

Attempts have been made to give productivity figure as – such as 10 person hours per Function Point but with a rider that it could vary from 2 to 135 depending on the product size and other factors.

Some times ranges are given – such as 15 to 30 hours per Use Case Point.

Some times empirical formulae are worked out depending on a set of factors – such as in COCOMO.

Another aspect is that these productivity figures lump all activities – requirements analysis, design, review, testing etc – in one single measure. The skill requirements for these activities are different, the tools used are different, the inputs are different, outputs are different – lumping them all together under the head "Software Development" and giving one single figure of productivity at best can only give a very rough estimate but never an accurate one.

5. The Productivity Path

We have the following activities in software development –

1. Pre-project activities
    a. Feasibility study
    b. Financial budgeting and approvals for the project
    c. Approvals – financial and technical
    d. Project go-ahead decision
2. Project startup activities
    a. Identifying project manager
    b. Allocating project team
    c. Setting up development environment
    d. Project Planning

    e. Setting up various protocols
    f. Service level agreements and progress reporting formalities
    g. Project related training
3. Software engineering activities
    a. User requirements analysis
    b. Software requirements analysis
    c. Software design
    d. Coding and unit testing
    e. Testing – integration, functional, negative, system and acceptance
    f. Preparing the build and documentation
4. Rollout activities
    a. Installing the hardware and system software
    b. Setting up database
    c. Installing the application software
    d. Pilot runs
    e. User training
    f. Parallel runs
    g. Rollover
5. Project cleanup activities
    a. Documenting good practices and bad practices
    b. Project post mortem
    c. Archiving records
    d. Releasing resources
    e. Releasing the project manager
    f. Initiate software maintenance

Now, when we talk of industry thumb rules of productivity, we are not clear as to how many of the above activities are included in the productivity figure.

Interestingly, no one would like to stake his life on the productivity figure – industry thumb rule – that is being floating around!!

Look at the nature of these activities –

1. Requirements analysis – here it is understanding what the user needs, wants and expects and documenting the same so that the software designers understand them and can design a system strictly in conformance with the stated requirements. There is a lot of dependence on external factors.
2. Software design – considering the alternatives of hardware, system software and development platforms, arrive at the optimal one, design an architecture that will meet the stated requirements and fulfill expectations and yet feasible with the current technologies and document the design in such a way that the programmers understand and deliver a product that conforms to the original specifications of the user. There are quite a few alternatives and this is a strategic activity and errors here have strategic consequences.

3. Coding – developing software code that conforms to the design and is as failure-free as possible – it is so easy to leave bugs inside!!
4. Code review – walking thru code written by another programmer and deciphering the functionality and try to guess the possible errors
5. Testing – trying to unearth all the defects that could be left in the software – it is an accepted fact that 100% testing is impossible!

Now with such variance in the nature of activities, it is obvious that the productivity of all these activities is not uniform. The pace of working differs for each of these activities.

These activities do not depend on the amount of software code produced but on other factors – such as –

1. Requirements depend on the efficiency and clarity of the source of requirements – users or documentation
2. Design depends on the complexity of processing, alternatives available and constraints within which the functionality is to be realized
3. Code review depends on the style of coding
4. Testing depends on how well the code is written – more errors are left, it takes more time to test and re-test
5. Coding itself depends on the quality of design

Therefore, we need to have separate productivity figures for each of these activities.

Drawing a parallel from the manufacturing industry, for punching hole in a sheet –

i. Machine setup
ii. Tool setup
iii. Load job
iv. Punch hole
v. De-burr hole
vi. Clean up
vii. Deliver the sheet for next operation

If multiple holes are punched, "per hole" time comes down, as setup activities are one-time activities.

If we look at "coding a unit" – the activities could be –

i. Receive instructions
ii. Study the design document
iii. Code the unit
iv. Test & debug the unit for functionality
v. Test & debug the unit for unintended usage
vi. Delete trash code from the unit
vii. Regression test the unit
viii. Release it for next step

Similarly, we can come up with micro activities for each software development phase.

5.1 Empirical or study-based Productivity figures?

Each of these activities has a different rate of achievement. We have to establish standard times for each of these activities. Then using the Work Study techniques like Synthesis or Analytical Estimation, we need to arrive at the over all time to complete the job.

Whether we use time study techniques to arrive at individual productivity studies or gather empirical data – to answer this query, we have to acknowledge that software development is not totally mechanical in nature nor is it totally creative in nature. Work Study acknowledges that it is not practical to time activities that have a creative component. Lots of work is being undertaken in the matter of "white-collar-productivity" and perhaps future may provide some methods to "time" software development productivity figures. As of present, empirical data seems to be the solution.

Where do we get data for this? One way is the Time Study using the Industrial Engineering techniques. Second and more easier, as well as reliable, is from historic data from the timesheets.

Most timesheet software available and is being used by the industry are oriented towards payroll and billing rather than capturing data at micro level so that it can be used for arriving at the productivity data. Most timesheets capture data at two, or three levels (project is always the first level, second and third can be module & component or component & activity or a similar combination) in addition to date and time. The timesheet needs to capture at five levels, namely, project, module, component, development phase, and the task accomplished – in addition to date and time for each employee. Thus data would be available to establish productivity data empirically in a realistic manner.

The present focus is on macro productivity – for all activities of software development. This needs to change and we need to shift our focus from macro to micro – productivity for all activities. The way to achieve is to modify our timesheet.

Benefits of productivity at micro level are –

i. Better predictability of software development
ii. Better quality estimates for pricing assistance during project acquisition/sanction stage
iii. More precise target setting while assigning work, which leads to better morale in the software developers
iv. More accurate cost estimation

6. Conclusion

The conclusions are that we need to shift focus from macro productivity to micro productivity; empirical data gathering is preferred for arriving at productivity figures and that improvement of timesheet is the way forward for computing micro level productivity figures.

****

About the Authors:

Murali Chemuturi is a Fellow of Indian Institution of Industrial Engineering and a Senior Member of Computer society of India. He is a veteran of software development industry and is presently leading Chemuturi Consultants, which provides software products for software estimation and software project management.

Sarada Kaligotla has completed her Master's in Computer Applications and is a certified PMP (PMI) and CSQA (QAI). She presently works for Blue Cross Blue Shield of MA. She has around 6 years experience in software industry with development and project management experience.

****