

# Effort Estimation in Agile Software Development

Fernando Machado  
Universidad Pontificia de Salamanca  
campus Madrid  
fmachado@ucu.edu.uy

Luis Joyanes  
Universidad Pontificia de Salamanca  
campus Madrid  
ljoyanes@fpablovi.org

## 1. Introduction

Effort estimation in software development is a difficult task and difficulty is inherent to software development. To build software, if you always do the same tasks in the same way, you can know how much effort takes each task and subsequently the entire software. But more often than not, you have to build something that is different from any other thing that you have ever built before.

Traditional estimation techniques include COCOMO [4] and Function Points [1]. These estimation techniques are fine for entire application lifecycles in the so called “traditional methods” [6], but they are not adequate in more lightweight processes, like XP [3], FDD [5], Scrum [8], and others, because:

- They don’t work without huge amounts of data collected from similar projects. Project and process diversity in agile development makes data gathering tough.
- They aren’t so fine for estimating iterations in iterative and incremental development, where you don’t need to know the entire lifecycle time span in advance.
- The estimation process might be overwhelming compared with construction process. Agile practitioners might see this like bureaucracy.

Agile development promotes iterative lifecycles and incremental development. In small organizations, with relatively permanent member teams, developing software with quite similar requirements and problem domains, enough useful data can be gathered somewhat quickly.

In XP for example, the plan for an Iteration is done during the Planning Game. The Customer provides prioritized Stories -functional requirements- and developers estimate each Story in units of effort instead of in units of time. A number called Project Velocity is used to calculate how many units of effort per week can really be done. An iteration contains only the most valuable stories that can be done in a fixed time frame, let’s say, four weeks [7].

An open question with this approach is how you determine these units of effort. Here we propose using a simple, but effective approach for effort estimation in this context.

## 2. Estimation procedure

Instead of estimating effort required by each task in terms of hours per developer, we take an indirect measure of effort and then we use historical data to convert this measure in development days.

The process is as follows:

- We estimate size and complexity for each task. There are simple guidelines that allow us to easily and quickly estimate each task.
- Then we classify tasks simultaneously by size and complexity in three categories.
- Each category has a rate to calculate how many hours will take to a developer to complete a task in this category. Rates changes as time goes by and estimation will be better iteration by iteration.
- Finally we sum up hours for all tasks and divide by the number of developers assigned to the iteration to compute duration of the whole iteration.

We use a previous iteration to calculate the initial rate: we asked developers to classify tasks in one of the three categories for the last finished iteration and to recall how much time each task has taken. As we know the complete development time spent, we adjusted the time provided by developers to match the sum of time for all tasks with the total time spent in the iteration.

Size and effort criteria to classify tasks are as follows:

**Table 1. Size and effort criteria**

Size	
Small	You must do one thing in one place once.
Medium	You must do a couple of things in a couple of places or a couple of times.
Large	You must do many things in many places many times.

Complexity	
Low	You know how to accomplish the task and have done similar things before.
Medium	You haven't done any similar task before but you can figure out how to do it.
High	You -and nobody- have no idea about how to do this task.

The matrix in figure 1 is used to determine the category for each and every task from the previously appraised size and complexity. Categories are easy -light gray-, medium -dark gray-, and hard -black-.

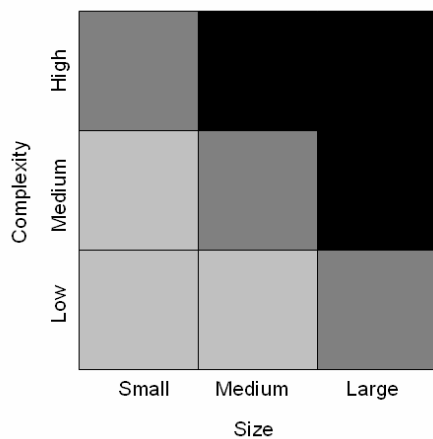


Figure 1. Task categories

We have been using this estimation method in practice in different agile projects from a local independent software vendor developing enterprise resource management software for small and medium business<sup>1</sup>.

Here we report results for a sample iteration estimation using this method. Estimation from the previous iterations gives us ratios like 0.82, 3.05, and 8.60 for easy, medium and hard tasks, respectively. Classification for current iteration give us 8 easy, 14 medium, and 11 hard task, resulting in 143.83 hours or 20.5 days, with days of seven working hours and one developer. The project really takes 19 working days to complete. The cumulative flow diagram [2] in figure 2 shows project progress.

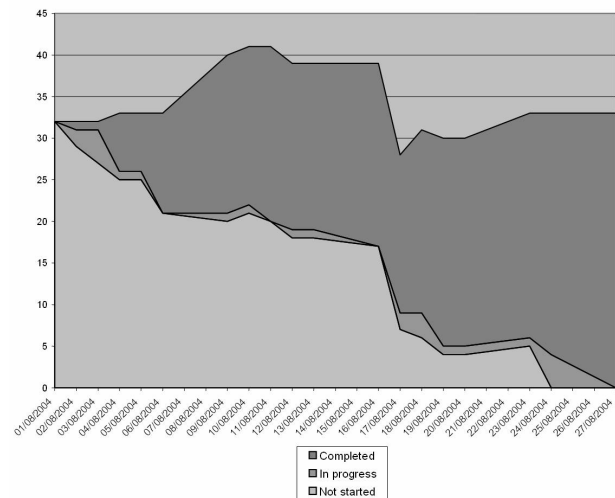


Figure 2. Cumulative flow diagram for test project

Delivery date is a commitment that we try to respect, that's why the total number of tasks varies during project; we can add new tasks to the current iteration, although in progress, if we can remove existing tasks in order to keep delivery date unchanged.

### 3. Conclusions

Results from applying of our estimation method are very a encouraging. We dramatically reduce estimation time, because developers do not spend a lot of time trying to figure out how much time each task will take; they simply classify tasks, which is simpler and quicker. And better yet, estimation resulted exact enough.

### References

- [1] Albrecht, A. J., "Measuring application development" *Proceedings of IBM Applications Development Joint SHARE/GUIDE Symposium*, Monterey, California.
- [2] D. Anderson and E. Schragenheim. *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results*, Prentice Hall PTR, 2004.
- [3] K. Beck. *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999.
- [4] B. Boehm. *Software Engineering Economics*, Prentice Hall PTR, 1981.
- [5] P. Coad, E. Lefebvre, and J. De Luca. *Java Modeling in Color with UML: Enterprise Components and Process*, Prentice Hall PTR, 1999.
- [6] M. Fowler. *The New Methodology*. 2003.
- [7] R. Jeffries, A. Anderson, and C. Hendrickson. *Extreme Programming Installed*, Addison-Wesley, 2001.
- [8] K. Schwaber. *Agile Project Management with Scrum*, Microsoft Press, 2004.

<sup>1</sup> <http://www.memorycomputacion.com>