Embedded Software Engineering as a Function of Time

Robert Berger
Embedded Software Specialist
Stratigou Rogakou 24, GR-15125 Polydrosso/Maroussi, Athens, Greece
Phone : (+ 30) 697 593 3428, Fax: (+ 30) 210 684 7881
E-mail: robert.berger@ReliableEmbeddedSystems.com

Abstract

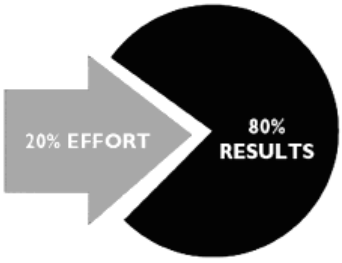
What inspired me to put down a few thoughts about time management and software engineering were the similarities I found while writing a talk about time management[1] and another one about how to avoid bugs and find those you could not avoid[2]. Here I want to give a very quick introduction to the foundations of time management followed by an enumeration of various methods to estimate how long your embedded software project will take. We'll start from a "classic" approach which is taught by the Software Engineering Institute (SEI)[3] followed by "suboptimal" approaches and conclude with agile estimation processes.

1 Introduction

Try to solve the following riddle: "Imagine you need to deliver a simple software project with two tasks. You estimate that each task takes eight hours to complete. How long will the project take?" ... Think about it ... A bit more ... 16 hours? ... What went wrong if it took a week instead? We'll revisit this issue again towards the end of the paper.

2 Time Management Basics

2.1 Prioritize - The Pareto Principle[4]



The Pareto principle (also called 80/20 rule) was suggested by management thinker Joseph M. Juran. It was named after the Italian economist Vilfredo Pareto, who observed that 80% of the income in Italy was received by 20% of the Italian population. Our assumption here is that 20% of all your tasks account for 80% of the value of all the things you do. Which means you need to find those 20% and prioritize accordingly.

2.3 Kaizen - Continuous Improvement[5]

"Things alter for the worse spontaneously, if they be not altered for the better designedly."[6] Sir Francis Bacon (1561-1626). You need to improve continuously if you want to stay competitive on the market.

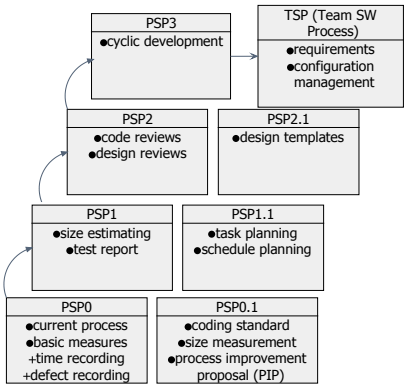
2.4 Change - Try Something Different

"He that will not apply new remedies must expect new evils; for the time is the greatest innovator"[6] Sir Francis Bacon (1561-1626). If you don't get out of your comfort zone and try different approaches don't expect different results.

3 “Classic” Software Engineering Methods - Software Engineering Institute (SEI)

3.1 Capability Maturity Model (CMM)[3]

A top down approach which improves the whole organization starting from management and going down to the developers.



3.2 Personal Software Process (PSP)[3]

A bottom up approach which improves the individual developers.

3.3 Team Software Process (TSP)[3]

Improves teams and guides PSP trained people in developing software intensive products.

4 “Suboptimal” Project time Estimation

4.1 Parkinson Estimation[7]

The Parkinson’s law states that *“Work expands to fill the available volume“.* This means that if the deadline is in twelve months and you have ten people available it will take you roughly 120 person months. Obviously like this you either grossly overestimate, which wastes resources or you grossly underestimate, which will make it very likely for your project to fail.

4.2 Price to Win Estimation[7]

With this approach price and schedule are targeted towards price and/or schedule to win. Usually it’s applied due to a lack of trust or understanding of realistic estimates and leads to a disaster for the customer and high stress for the supplier.

Please note, that countermeasures like a Vickrey auction can be used to avoid the price to win type of estimation. This is a type of sealed-bid auction, where bidders submit written bids without knowing the bid of other people in the auction. The highest bidder wins, but the price paid is the second-highest bid. (Again here is the danger of collusion by losing bidders)

5 “Agile” Planning

Focuses on the planning and not the plan and encourages change by producing plans that can and should easily be changed. Planning is performed throughout the project, not just once at the beginning.

5.1 Variation to Wideband Delphi Estimation[8]

A software team leader calls his team for a kickoff meeting, where he/she dishes out the requirements. The team creates a work breakdown structure (WBS), discusses assumptions and decides on metrics (lines of code, hours,...). This is followed by individual preparations. Everyone can assume that tasks are done in a sequence and that everyone is 100% devoted to each task. Afterwards there are estimation sessions, where everyone proposes his/her estimates until the estimates converge, which leads to discussions and some initial implicit design, Now you have the whole team behind some common and (at least for the team) realistic time plan.

5.2 Planning Poker[9]

Planning Poker is another variation of the Wideband Delphi method and most commonly used in agile software development, in particular in Extreme Programming. Planning Poker is based on a list of features to be delivered and a deck of cards. The feature list describes some software that needs to be developed. The deck contains the following cards: 0, ½, 1, 2, 3, 5, 8, 13, 20, 40, 100.

The following steps need to be executed:

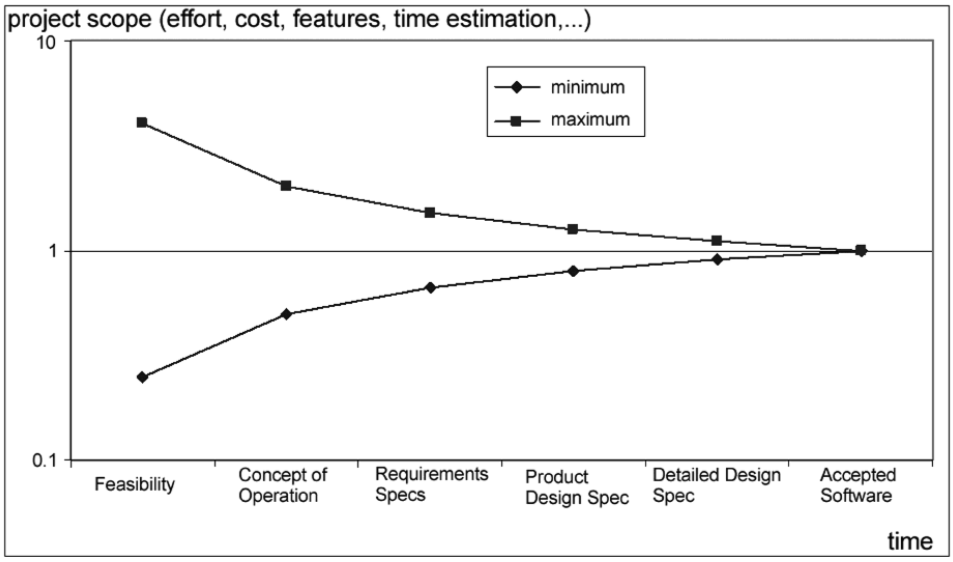
1. Customer explains story
2. Team discusses work involved
3. Each estimator picks a card representing estimates
4. Everybody reveals estimate simultaneously
5. Lowest and highest estimator justifies
6. Team discusses the estimates
7. Repeat from 3. until estimates converge
8. Team decides on collective estimate

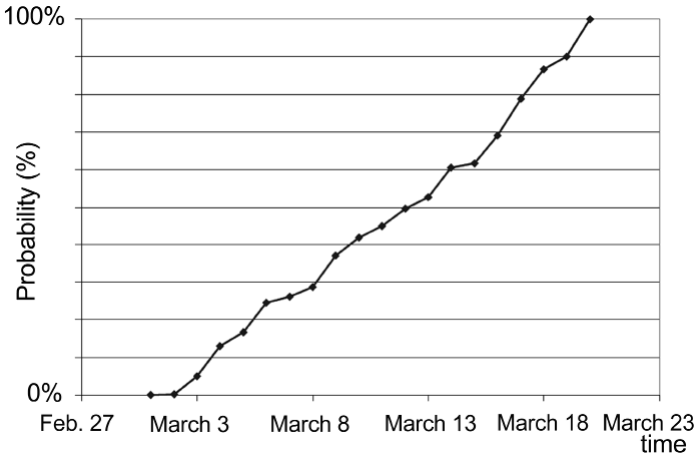
The Moderator or the Project Manager may at any point turn over an egg timer and when it runs out all discussion must cease and the round of poker is played.

The numbers on the cards account for the fact that the longer an estimate is, the more uncertainty it contains. Thus, if a developer wants to play a 6 he/she is forced to reconsider and either decide that some of the perceived uncertainty does not exist and play a 5, or accept a conservative estimate accounting for the uncertainty and play an 8.

Before we discuss the next methodology let's have a second look at our initial riddle: "Imagine you need to deliver a simple software project with two tasks. You estimate that each task takes eight hours to complete. How long will the project take?" If you answered 16 hours and it indeed took 16 hours you might have discovered the holy grail of software time estimation. Please let me know how it works and let's make lots of money.

In case your answer was: "It's impossible to make precise estimates" you are more likely in line with mere mortals like me. The cone of uncertainty[10] (see below) shows that the further down we move the time line in our software project the better the estimation gets.





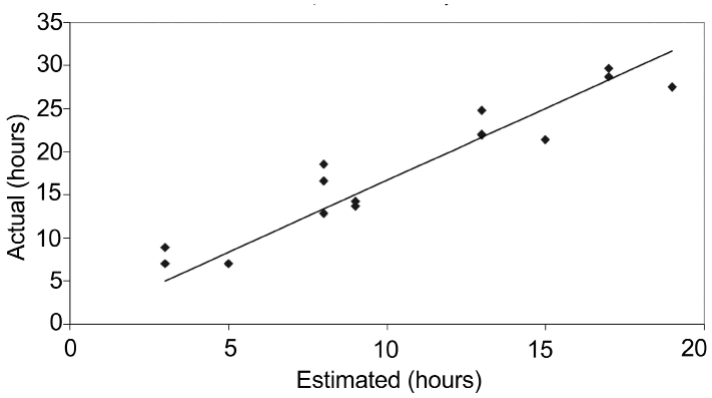
The estimated completion date is not a point on a graph, but a probability distribution. The steeper the slope (see top of page), the more reliable your estimates are.

5.3 Evidence Based Software Scheduling[11]

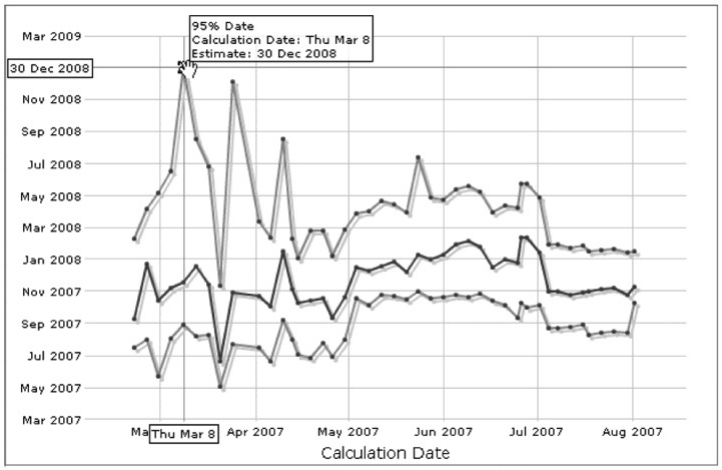
The following steps need to be executed:

1. Break down the work
   If you can not break down the work in units smaller than 16 hours you did not think hard enough about the design.
2. Track elapsed time

Calculate the "velocity" (progress), which is the ratio between actual and estimate



3. Simulate the future



With Monte Carlo technique[12] you can simulate the completion of each task many times based on velocities (past performance) and like this the probability of completion on a certain date can be calculated.

4. Manage the project actively
   e.g. Prioritize Features

6 Discussion

It depends on the domain you are working in, but many times the good old waterfall model just does not cut it anymore in case of software development e.g. due to requirement changes. A top down approach like CMM is costly and lengthy compared to bottom up techniques like PSP, TSP which show better results much quicker. Try out something new, but before changing anything in your current way of working make a baseline measurement of your current "status quo" and keep doing the metrics while making changes and also when you apply the new process so you can see if things improved or worsened. Don't give up immediately if you see things worsening since new technologies usually take some time to pick up. Next time your boss asks you for the delivery date give him a probability distribution and when he yells at you refer him to me, if you like.

7 References

[01] Robert Berger: IEEE Professional Activities: Time Management, http://ieeesb.elec.qmul.ac.uk/sbc2008/index.php/programme/fullprog/workshop8/
[02] Robert Berger: Busting bugs from birth to death of an embedded system running an RTOS ISBN: 978-3-7723-3961-5, Proceedings & Conference materials embedded world 2008
[03] http://www.sei.cmu.edu/
[04] http://en.wikipedia.org/wiki/Pareto_principle
[05] http://en.wikipedia.org/wiki/Kaizen
[06] http://www.brainyquote.com/quotes/authors/f/francis_bacon.html
[07] Estimating Software Projects - http://tarpit.rmc.ca/leblanc/492/2008/EEE/lectures/
[08] Andrew Stellman, Jennifer Greene: Applied Software Project Management, ISBN: 978-0-5960-0948-9, O'Reilly, 2005
[09] http://en.wikipedia.org/wiki/Planning_poker
[10] http://www.construx.com/Page.aspx?hid=1648
[11] http://www.joelonsoftware.com/items/2007/10/26.html
[12] http://en.wikipedia.org/wiki/Monte_Carlo_method

8 Author

After working for multinational companies in Middle Europe, Robert Berger is currently an embedded systems software manager for one of the largest multinational groups in Southeast Europe. He is part of the team that received from TMC the 2006 product of the year award in the USA for their communications solution and deployed 2007 the first commercial HD (high definition) H.264 IPTV solution in the USA and 2008 the first commercial HD H.264 IPTV solution in Canada. He has always seen himself as self employed and consults, trains and engineers projects all over the globe. His true passion is to mentor and coach, so he works also as a teaching assistant for the embedded course of a private university in partnership with Carnegie Mellon. He enjoys helping companies improve their development process and writing embedded software. He is a project manager for the Linux Driver Project, IEEE lecturer for professional activities in the EMEA Region, speaker at the Embedded World Conference and moderates various Embedded Systems Professionals forums all over the world wide web.