Originally published in the PMI World Congress, Prague, Czech Republic, May 2004

Combining Critical Chain Planning and Incremental Development in Software Development Projects

Author Biography

Eduardo Miranda is a Program Director at Ericsson Research Canada and an industrial researcher affiliated with the Research Laboratory in Software Engineering Management at the Université du Québec à Montréal. He is in charge of investigating new management techniques for planning and tracking projects. He has a BSc in System Analysis from the University of Buenos Aires, Argentina, a MEng. from the University of Ottawa, Canada and a MSc. In Project Mangement from the University of Linköping, Sweden. He is a member of the IEEE Computer Society and the ACM. Contact him at Ericsson Research Canada, 8500 Decaire Blvd., Town of Mount Royal, Quebec, H4P 2N2, Canada; eduardo.miranda@ericsson.ca.

Abstract

Cutting content seems to be the prevalent way of meeting deadlines in projects that are running late. But why waste effort and time working on things that most likely, are going to go at the first sign of trouble? Why don't make the decisions about what is important and what is not up-front, and only start work on the latter if we have the necessary time to do it?

By combining critical chain (CC) and Incremental Development (ID), two well known techniques, we can create a new approach to plan and execute projects, which guaranties, with a set probability, the delivery of an agreed subset of the total functionality by a preset date.

Keywords

Project planning, incremental development, critical chain, risk management, project management and control, design-to-schedule.

Introduction

A time-bound project is a project that is constrained by hard deadlines. Hard deadlines are those in which the date of delivery is as important as the delivery itself. If the project delivers after the deadline, the delivery loses much of its value. Examples of hard deadlines are exhibition dates, government regulations, a competitor's announcement and the customer's own business plans.

Most of these projects, start with more requirements that can realistically be handled within the imposed time constraints and consequently, midway through the development, they find necessary to start slashing some of them. These un-planned cuts result in customer frustration and wasted effort. A much better approach would be to define the requirements' priority up-front, allocating their development to successive releases of the project in such way, that we could be almost sure that the project will deliver all the important requirements, that the second less important will still have a fair chance of being delivered, with the gold plated ones only to be done if there is any time left.

While the lack of requirements prioritization, is one of the reasons why most of these projects are late, it is certainly not the only one. The inability of traditional planning methods to deal with the uncertainty present on the estimates on which the plans are based, and the failure to recognize that development work do not progress in linear fashion, the infamous 90% complete syndrome, are also to blame.

As will be explained later, traditional critical path calculations involving uncertainty produce considerably shorter schedules than those that should be realistically expected. With a shorter schedule as starting point, being late is a tautology.

The second problem, assuming that a task progresses at a constant rate, prevents project managers from seeing the early signs of delay in tasks until it is too late to take any other action than trim down features, compromise on quality or re-schedule the project.

The method [1] presented here addresses these problems by combining ideas from critical chain planning [2,3], incremental development [4] and rate monitoring [5] into a practical approach for planning and executing time-bounded projects.

This method is not a one-stop solution for all software development problems. It just focuses on how to best organize a project to guarantee that a working product with an agreed subset of the total functionality could be delivered by a required date.

The sections that follow explain the fundamentals of planning under uncertainty, the use of rate monitoring to track progress and finally the application of these concepts in planning and executing projects.

Uncertainty in the planning and execution of projects

Task statistics

Uncertainty is the reason project management is needed. The estimates on which project schedules and resource allocations are based are never single numbers; whether spoken or not, there are many assumptions behind each of them. Some of these assumptions concern the complexity of the tasks, others our ability to carry them out. Some of them, if true, will contribute to an early completion of a task, others will add to the execution time. Intuitively we could see, that for a task to finish at the earliest possible time, all the “favorable” assumptions must be true and all the “inauspicious” ones false. The probability of this happening is very low. The same could be said for the latest possible date. The most likely date corresponds then to a situation in which the most probable “good” assumptions are true and the most probable “bad” ones are false. Numerically, the situation can be expressed by a triangular probability distribution such as the one shown by Figure 1[1].

[1] Strictly speaking, the caption for the “y” axis in Figure 3 should read *f(x)* since this is a continuous distribution. The term probability is used instead for its intuitive appeal.

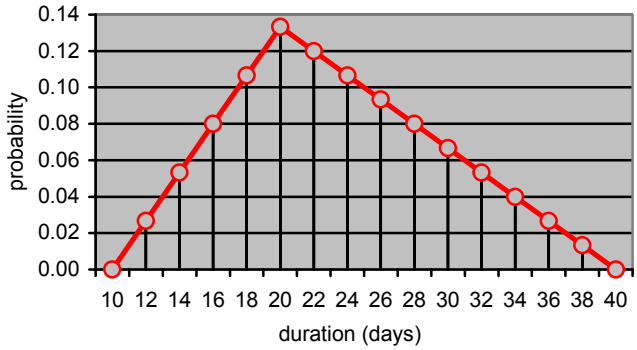


Figure 1. If all the favorable assumptions are true and all the gloomy are false, the task will be completed in 10 days, this is the Earliest Completion Date. The Most Likely duration is 20 days. If everything that can go wrong, short of abandoning the task, goes wrong the task could be completed in 40 days. This is the Latest Completion Date.

Since the actual probability distribution function for the duration of the task is unknown, the choice of a simple triangular distribution is a sensible one [5]. Its right skewedness captures the fact that while there is a limited number of things that can be done to shorten the duration of a task, the number of things that can go wrong is virtually unlimited.

From the project management point of view, more important than the probability of finishing on a specific date, is the probability of completing the task on or before a certain date. This probability, called the *on-time probability of the task*, can be derived from the cumulative distribution shown in Figure 2.

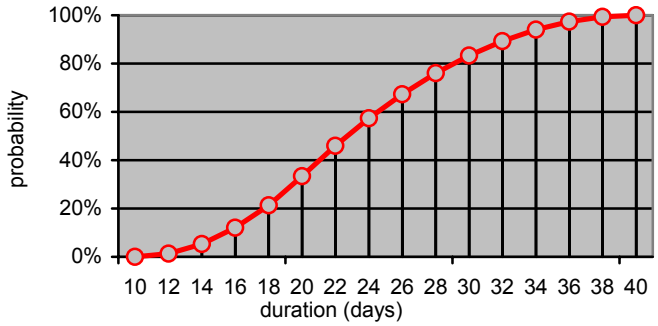


Figure 2. Cumulative probabilities. The Most Likely completion date has an on-time probability of less than 40%. The Expected completion date is of around 23 days. If we want to be 75% sure of completing the task on time we would have to schedule 27 days.

In general, the larger the number of assumptions behind the estimated task duration, the larger the spread between the earliest and the latest completion dates. The effect of such an uncertainty results in very different on-time probabilities, as shown by Figure 3.

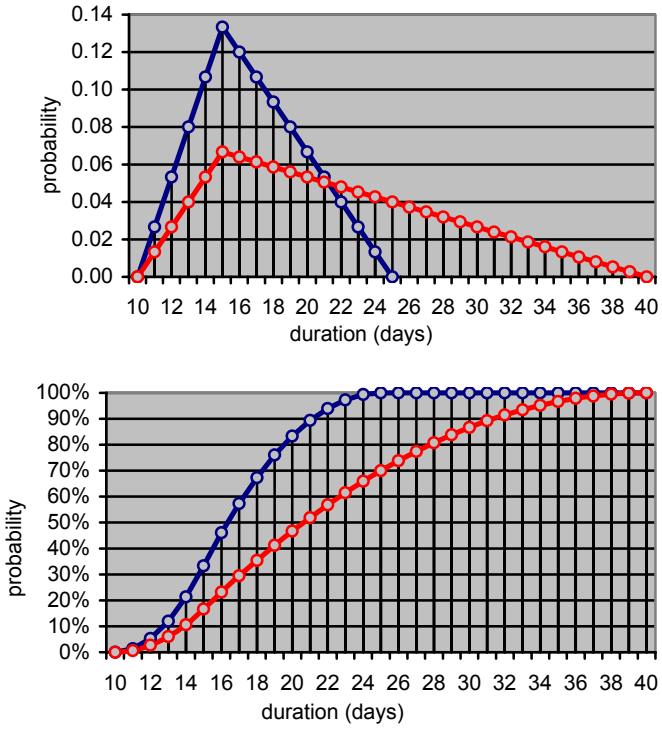


Figure 3. Two tasks with the same Earliest and Most Likely, but different Latest Completion dates have different levels of risk. The Expected completion dates for the less risky task is 17 days, while for the other is 23 days. By the same token, the on-time probability of the Most likely date is around 37% in the first case and under 20% in the second.

From tasks to projects

A common approach used to assess uncertainty in projects, is to calculate the expected duration of the project as the sum of the expected duration of the tasks along the critical path, with an standard deviation equal to the square root of the sum of the squares of the standard deviation of the same tasks, and then to use a normal distribution to calculate the on-time probability for the project. This approach is based on the *central limit theorem*, which states that the distribution of the sum of a number of independent random variables approaches a normal distribution as the number of variables (tasks) grows larger.

Assuming independent tasks duration as required by the central limit theorem, although a very common assumption, is perhaps one of the most dangerous a project manager can make. In practical terms, this assumption expresses the belief that the lateness of some tasks is compensated by the early completion of others and that in the end everything balances out. This may be a valid assumption in the construction industry and when dealing with events such as rain, but not in a software development project where an underestimation of the system's complexity will affect the duration of most tasks in the same direction. Thus, if there is an underlying cause that could shift the duration of several tasks in the same direction, the tasks are not independent but correlated. The practical consequence of dealing with correlated tasks duration is an increase in the project's standard deviation, which translates into higher risks.

Other problem not addressed by traditional critical path calculations, is the problem of merging paths, where the earliest start of the integration task always corresponds to end the latest development path. This results in a mechanism that passes delays, but seldom passes savings! Figure 4, illustrates both cases.

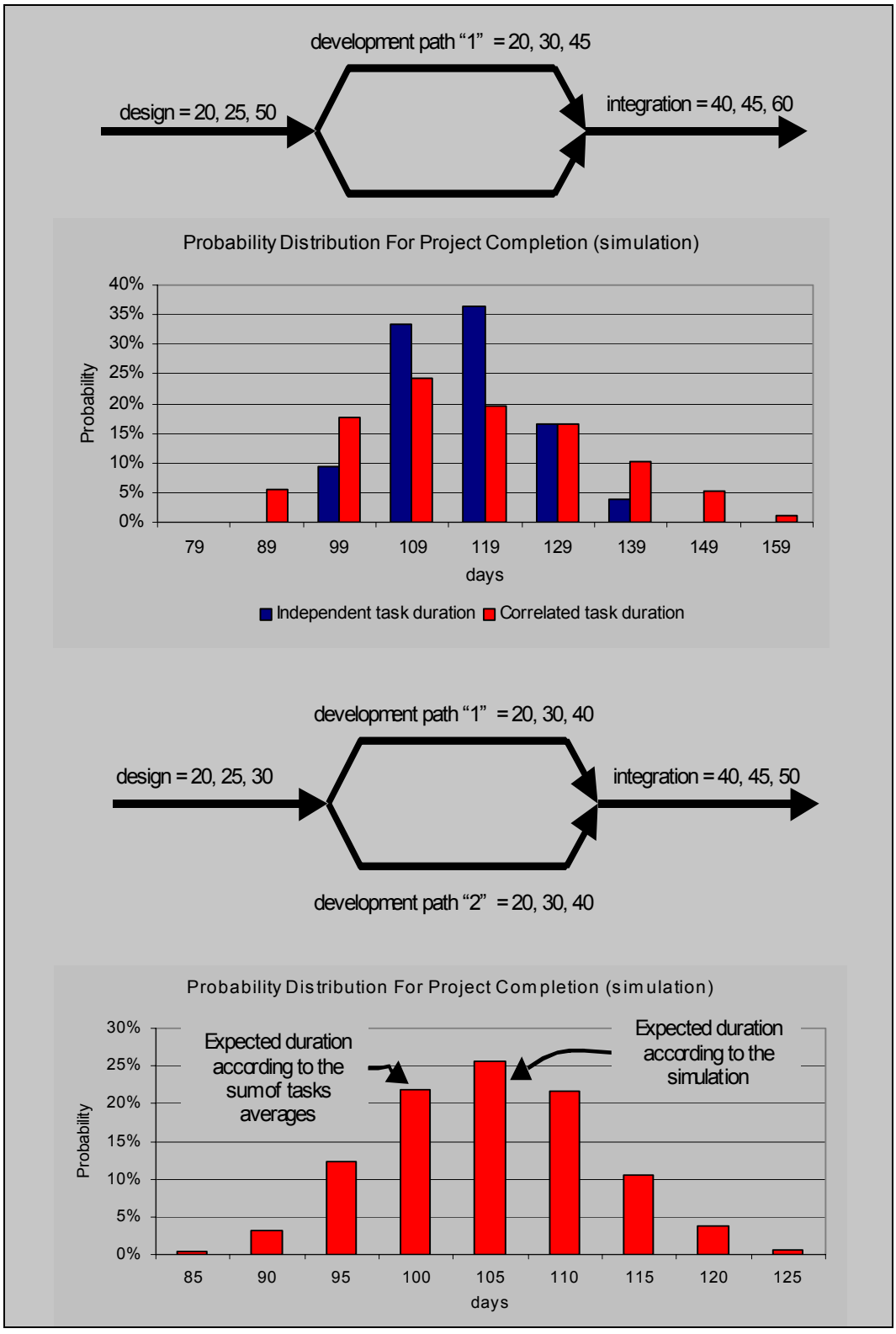


Figure 4 – In the presence of uncertainty, the expected project duration is not equal to the sum of the expected duration of the tasks in the critical path.

Measuring Progress Using Rate of Changes

When measuring the progress of a task in terms of its main output, i.e. requirements defined, LOC, errors found, pages of documentation written, etc, it is possible to observe that the rate of growth of the output is not constant throughout the duration of the task and that it more closely resembles the shape of Figure 5. This "S" pattern [7,8,9,10,11], typical of many intellectual activities could be explained by the existence of a number of actions and thought processes at the beginning and end of the task which, although value adding, do not contribute directly to the quantity being measured. Examples of such actions and thought processes are: learning, team formation and work reviews. Whatever the true reasons for this effect, it is so common and noticeable that has a name of its own: "the 90% complete syndrome".

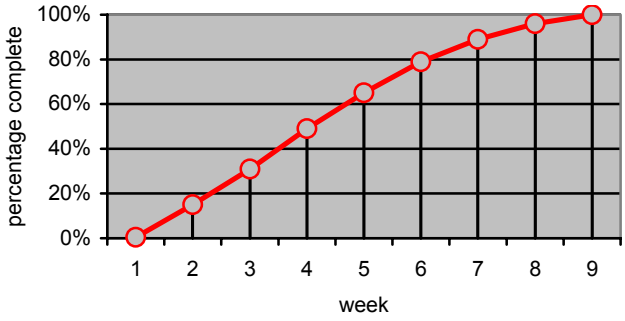


Figure 5. The "S" curve. Production does not grow at a constant rate. At the peak of productivity, between weeks 3 and 5, the percentage complete soars 20% in just one week. Towards the end of the task it takes three more times to go from 80 to 100% complete.

The result of extrapolating completion dates from the rates of progress observed through the half-life of the task using a straight line, is the announcement of optimistic completion dates that are never met. Figure 6 shows the error incurred by using a linear forecast instead of the "S" curve paradigm

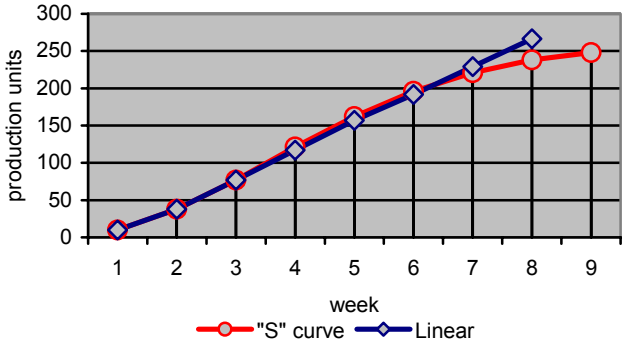


Figure 6. Assuming that the task output is 250 units of production (Requirements, FP, Errors detected, etc) a linear projection would forecast its completion by week 7.5 while the "S" curve will put it at week 9. Assuming the task duration was originally estimated to be 7 weeks, according to the linear projection it will be completed almost on time, but according to the "S" curve it will be 2 weeks late.

Combining Critical Chain and Incremental Development

Figure 7 illustrates the proposed project model. The *Increment Planning* task uses statistical techniques to break down the project scope into a series of *Development Increments* in such a way that it is almost certain that all requirements allocated to the first increment will be implemented on time; that there is a fair chance to implement those allocated to the second increment and so on. *System Engineering* encompasses requirements, value and trade-off analysis from a user perspective; this is the activity where the prioritization takes place. *System Architecting* is responsible for the general form of the solution, interface definitions and the analysis of dependencies between

requirements. The system architecting activity shall take and all encompassing view in order to prevent the surfacing of inconsistencies later in the development process. All three activities take place concurrently as there is a need to balance what needs to be done from the user perspective with what could be done from a technical perspective. Each *Increment Development* is a self-contained mini-project. We do not assume or impose any particular approach beneath this level, so development could be organized according to a waterfall or an iterative life cycle as deemed appropriate. All increments, but the last, are isolated from the project delivery date by a buffer whose purpose is to absorb any overrun in their execution.

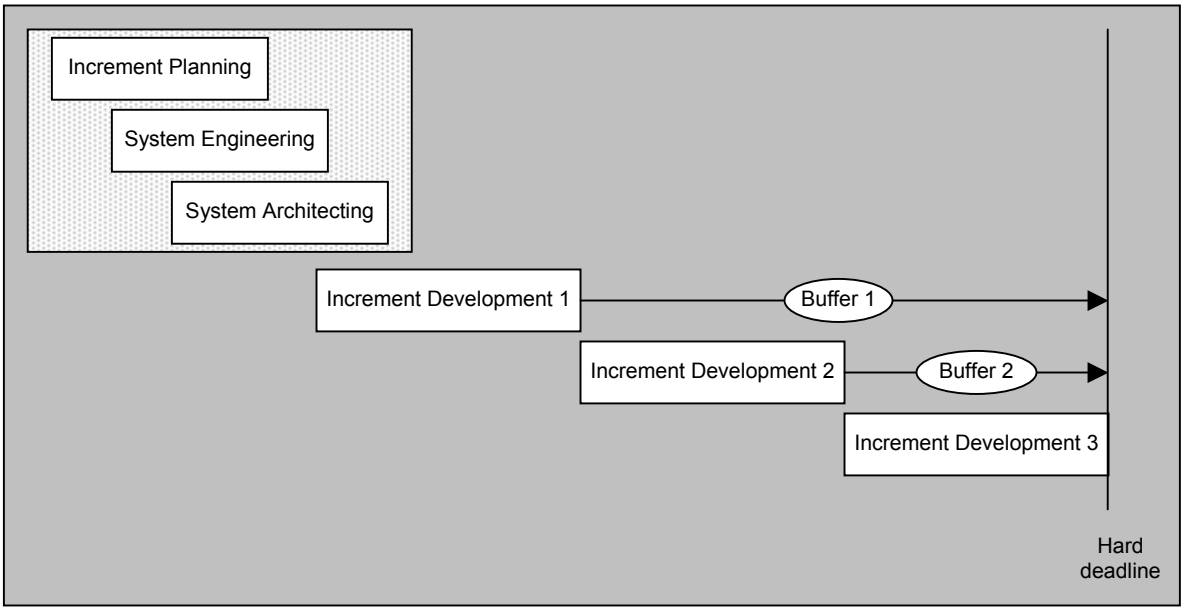


Figure 7. Combining CC and IC in a single project Model

During execution, work progress is forecasted using models that more closely resemble the way people work than a simple extrapolation of last week's results. As shown by Figure 8, the output from the models is used to forecast the activities' completion dates and to take corrective actions. Work in one increment does not start until the previous one is completed. This prevents people from wasting time developing things that might never be finished anyway

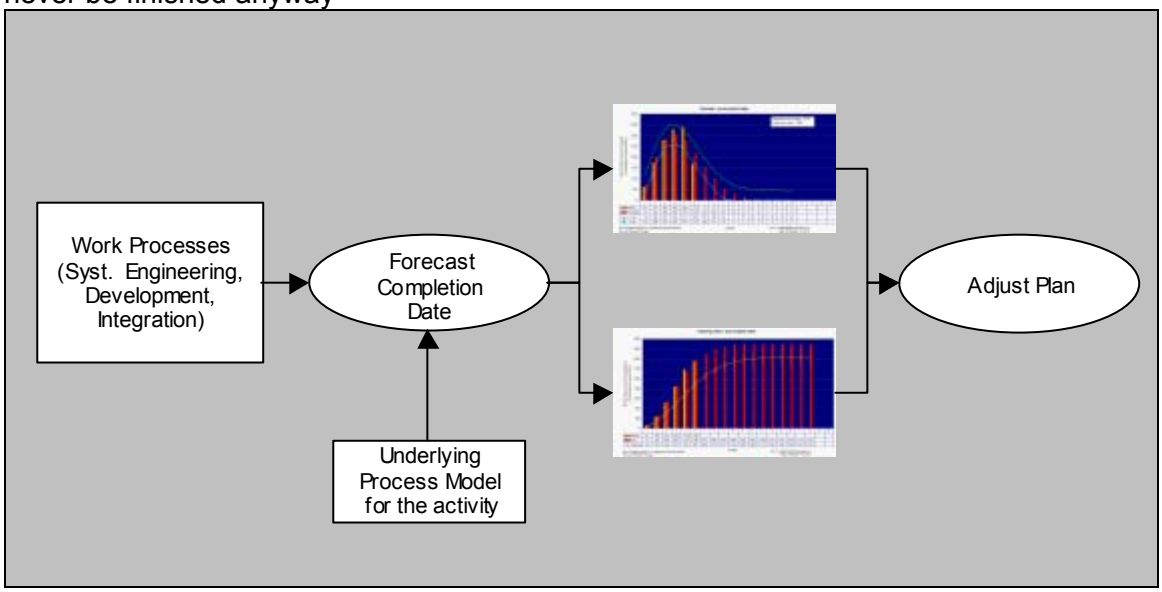


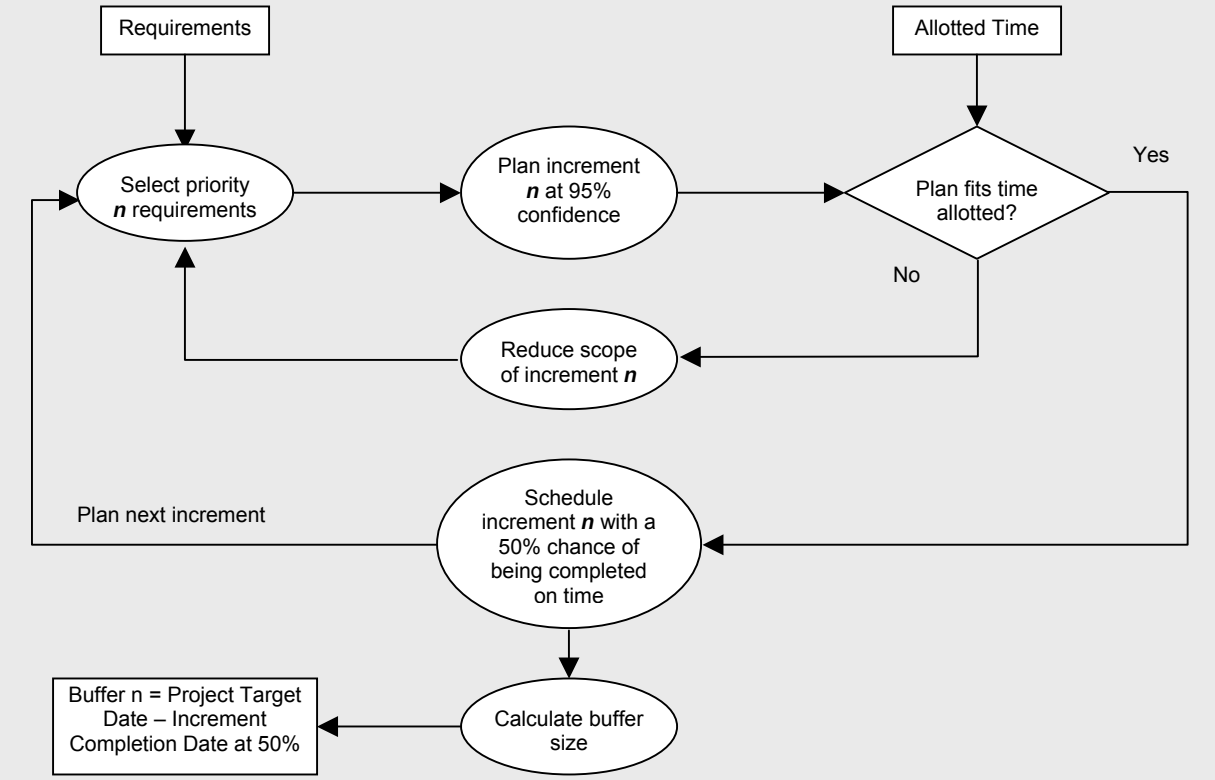
Figure 8. Project tracking

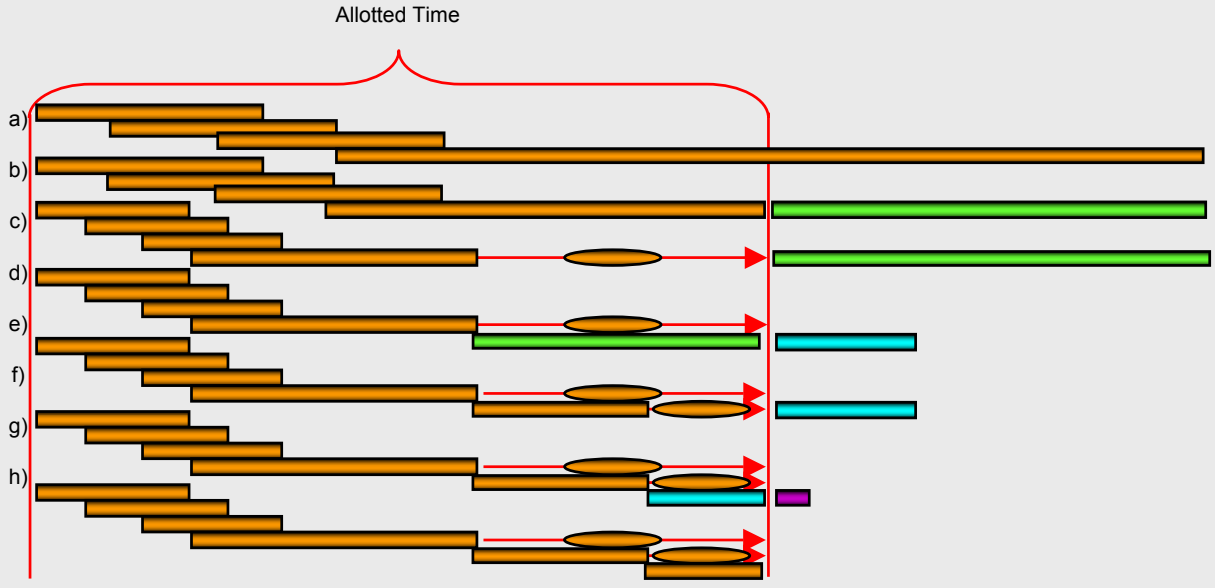
Project Planning

Once the feasibility of the project has been established, the next step is to define the duration of the development tasks in terms of its Best, Most Likely and Worst case scenarios as functions of the increment's scope. Second, the content of the increment is adjusted so it will have a high probability, i.e. 95%, of being completed in the allotted time. Third, the tasks are re-scheduled using the duration that corresponds to a 50% on-time probability, allocating the difference between the high and the lower confidence dates to a buffer. The next increment is then planned using the length of the buffer as the time allotted.

Two aspects that need to be considered in the selection of the requirements to be developed in each increment are: the technical dependencies that might exist between them and the need to provide functionally complete subsets to the user.

Figure 9 illustrates the overall process and the boxed note at the end of the paper, the probability calculations.





a) Total project duration. b) First priority requirements planned at 95% certainty. c) First priority requirements scheduled at 50% probability of being on time. d) Requirements that did not fit into Increment 1 are moved into increment 2. e, f, g & h) The process is repeated.

Figure 9 Increments are planned to fit within the allotted time

Table 1 shows the approximate[2] probabilities of delivering the content of each increment when planned according to the proposed approach. Compare this to a conventional plan, in which every requirement has the same probability, let's say 50% irrespective of its importance to the user.

Table 1 - Success Probabilities

Increment	Calculation	On-time probability
1	As planned	95%
1 + 2	0.50 * 0.95	≈ 47.5%
1 + 2 + 3	0.50 * 0.50 * 0.50	≈ 12.5%

Estimating the Minimum, Most Likely and Maximum durations

Although the specific techniques for estimating the minimum, most likely and maximum duration of the tasks will depend on whether the estimation is done using a cost model, an expert approach or a Delphi process, it is crucial to the success of the method, that all completion dates that could reasonably be expected, be included between the minimum and the maximum duration.

In the case of a parametric cost model like CoCoMo, this could be done for example, by changing the value of key cost drivers such as SLOC, PCAP or CPLX[3] and in the case of the Delphi process by recording, not only the converging value, but the optimistic and pessimistic estimates as well.

Project Control

In a time-bound project there is very little room for recovery, so once a problem manifests itself, it is almost too late. Controlling a project under these circumstances requires a mechanism that:

1. Identifies the early the signs of a delay;
2. Minimizes false alarms;
3. Minimizes disturbances to ongoing work;
4. Provides a clear definition of what will be delivered and by when.

While the first three properties are important to the people working and managing the project, the fourth is of utmost importance to the customer who depends on the project's deliverables to execute his own business plan.

The early identification of a delay is achieved by updating the buffers, not with the actuals but with the estimates at completion (EAC) of the individual tasks. The estimates are computed by fitting a Rayleigh curve to the progress reported, and then projecting it into the future.

[2] These calculations assume that the times it takes to develop each increment are independent from one another. As was said earlier this is seldom the case, nonetheless these numbers offer a reasonable approximation.

[3] These are cost drivers defined in the CoCoMo II model. SLOC stands for Source Lines of Code, PCAP for Programmer Capability and CPLX for Software Complexity.

False alarms and disturbances to on-going work are prevented by the use of buffers, which isolate workers from overreactions to small variations, by absorbing up to a 25% variance before sending a signal.

Figure 10 describes the control approach. Depending on the specific task being monitored, the units in which the work performed is measured will be Requirements Defined, LOC produced per week, number of errors detected, etc.

The re-planning of the next increment, if necessary, should take into consideration whether the factors that affected the development of the current increment will also have an effect on it, and the duration an effort adjusted accordingly[4].

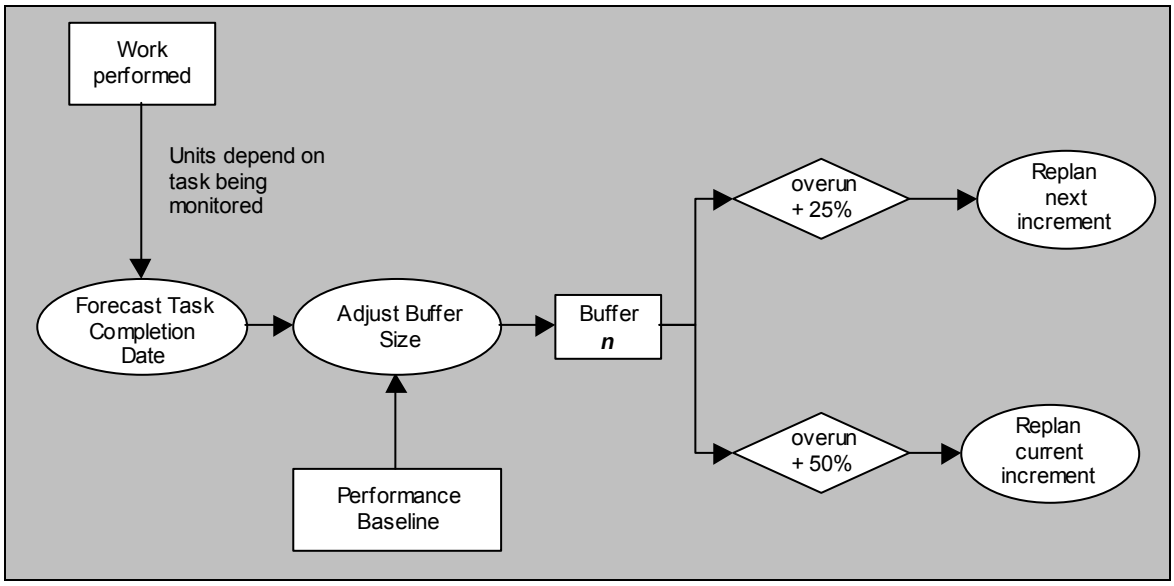


Figure 10 Monitoring progress and triggering of re-planning

Rewards, recognition and price incentives

How can all project stakeholders be sure that the best effort will be applied towards implementing all requirements and that people will not just get by implementing those in the first increment? The answer could be found in the reward and recognition system.

Whether employee's rewards or price incentives in contracts, the incremental model provides a clear criterion by which performance can be evaluated and rewarded. The delivery of the first increment has no reward associated with it: everybody is just doing their job; subsequent increments result in increased recognition of the extra effort put into the task.

The On-time probabilities shown in Table 1 can be used to calculate the expected value of the reward. This calculation is important because a large amount, with a very small probability will result in a low expected value and could be perceived as a lottery by the employees, thus failing to act as motivator.

[4] This valuable idea was suggested by one of the reviewers.

As an example, a $5,000 reward for “Increment 2” has an expected value of $2,375. The same amount applied to “Increment 3” has an expected value of $625. Clearly, the motivational value of the reward is not the same in both cases.

Our contribution

As mentioned at the beginning of the paper, the proposed approach brings together several existing techniques. Its value resides precisely in this. Specifically we combine a general project management approach like Critical Chain with a well-known software development method, the incremental model, to realize a new approach specially conceived to deal with time-bounded projects. We also provide a decision rule to calculate the size of the increments to be developed, a reward model based on the expected value of the increments and a recommendation to track the project based on forecasts rather than in actual progress. Furthermore, we do not presume independent tasks' duration, which leads to significant differences in the size of the buffers and addresses one of the main issues raised by the critics of the Critical Chain approach.

Summary

The premise in which the method is based, is that businesses are better off when they know what could, realistically, be expected than when they are promised the moon, but no assurances are given with respect as to when they could get it.

By taking a probabilistic, rather than a deterministic approach, the method recognizes that in any development project there are hundreds of things that can go right and thousands that can go wrong and makes them an intrinsic part of the planning and control processes.

Although still in an experimental stage, the method proposed in this paper has received a warm welcome when presented both, within and outside Ericsson.

Up to today, the main obstacles found to the wider acceptance of the techniques proposed, has nothing to do with the validity of the arguments cited or the rationale behind the method, but rather with a “can do attitude” that rejects the existence of things over which we have limited control and the prevalence of a business culture which seems to reward wild promises over a bounded rationality.

Acknowledgements

Thanks to Tamara Keating, Ericsson Research Canada; Alain Abran, Université du Québec à Montréal; and Raul Martinez, RMyA, and the IEEE reviewers for their comments and insight.

References

1. Planning Time Bounded Projects, IEEE Computer, March 2002,Volume 35, Number 3
2. Critical Chain, E. Goldratt, The North River Press, 1997
3. Project Management in the Fast Lane, R. Newbold, St. Lucie Press, 1998
4. Rapid Development, Taming Wild Software Schedules, S. McConnell, Microsoft Press, 1996

5. Technical Performance Measurement, Earned Value and Risk Management: An Integrated Diagnostic Tool for Program Management, N. Pisano, http://www.acq.osd.mil/pm/paperpres/nickp/nickpaso.htm
6. Practical Risk Assessment for Project Management, S. Grey, John Wiley & Sons, 1995
7. A Model for Software Development Effort and Cost Estimation, K. Pillai and S. Nair, IEEE Transactions on Software Engineering, Vol. 23, No.8, 1997
8. Measures for Excellence – Reliable Software On Time, Within Budget, Prentice-Hall, 1992
9. Technological Forecasting for Decision Making, J. Martino, McGraw-Hill, 1993
10. The Use of Reliability Growth Models in Project Management, E. Miranda, 9th International Symposium in Software Reliability, IEEE, 1998
11. On Predicting Software Related Performance of Large-Scale Systems, J. Gaffney, CMG XV, San Francisco 1984

Calculating Project Probabilities

The most common way of calculating project probabilities is using the PERT approach:

1. For each activity *i*, produce best, most likely and worst case estimates.
2. Compute the mean, d_i and standard deviation, s_i of each task using the following formulas[5]:

$$d_i = \frac{Best + MostLikely + Worst}{3}$$

$$s_i^2 = \frac{Best(Best - MostLikely) + Worst(Worst - Best) + MostLikely(MostLikely - Worst)}{18}$$

3. Determine the critical path based on the d_I , $i = 1,2,\ldots,n$
4. Once the critical activities are identified, sum their means and variances to find the mean and standard deviation of the project length using the following formulas:

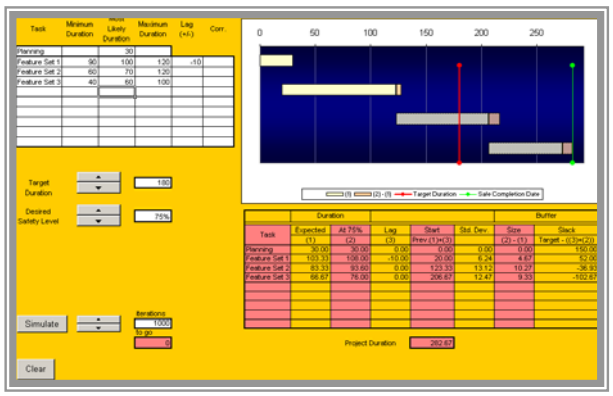
$$ProjectLength = \sum_i^p d_i$$

$$ProjectStdDev = \sqrt{s_1^2 + s_2^2 + \ldots + s_p^2}$$

5. Calculate the probability of finishing before a given date *T* using the formula below and a table of normal probabilities:

$$P(t \le T) = P\left(t \le \frac{T - ProjectLength}{ProjectStdDev}\right)$$

The approach described above works well as long as the duration of the tasks is independent, however as mentioned before this is hardly the case in most software development projects. The calculation of the variance of a project on the presence of correlated variables is a complicated process, which requires the calculation of the task's covariance, so a better approach is to use a method called Monte Carlo simulation. At Ericsson a homegrown tool called MinimumTime, see below, implements all necessary calculations. More sophisticated packages could be obtained from specialized vendors such as Primavera, Palisade or Crystal Ball.



[5] These formulas assume a triangular distribution. Other approximations, such as the traditional PERT calculations based on the beta distribution, could be also used.