The Economic Impact of Software Development Process Choice - Cycle-time Analysis and Monte Carlo Simulation Results

Troy Magennis

troy.magennis@focusedobjective.com

Abstract

IT executives initiate software development process methodology change with faith that it will lower development cost, decrease time-to-market and increase quality. Anecdotes and success stories from agile practitioners and vendors provide evidence that other companies have succeeded following a newly chosen doctrine. Quantitative evidence is scarcer than these stories, and when available, often unverifiable.

This paper introduces a quantitative approach to assess software process methodology change. It proposes working from the perspective of impact on cycle-time performance (the time from the start of individual pieces of work until their completion), before and after a process change.

This paper introduces the history and theoretical basis of this analysis, and then presents a commercial case study. The case study demonstrates how the economic value of a process change initiative was quantified to understand success and payoff.

Cycle-time is a convenient metric for comparing proposed and ongoing process improvement due to its easy capture and applicability to all processes. Poor cycle-time analysis can lead to teams being held to erroneous service level expectations. Properly comparing the impact of proposed process change scenarios, modeled using historical or estimated cycle-time performance helps isolate the bottom line impact of process changes with quantitative rigor.

1. Introduction

Company IT leadership often initiate software development process improvements due to dis-satisfaction with time to market and development cost. Leaders responsible for product direction have many ideas, but feel these ideas take too long to make it into their customers' hands. These leaders get frustrated that competitors are (or appear) faster to market than their organization, and look for ways to compete more equitably.

Process change proponents and vendors offer a variety of the latest tools, books, training, and certification as immediate relief to the vague "too slow" and "too costly" symptoms. XP, Agile, Lean, Scrum, and Kanban are some of the well-known processes that have risen to the top of the popularity charts, each with case studies (often just one) showing great impact when applied correctly by the inventors. The final choice appears to fall on faith based lines, with many organizations moving from one process to the next in search of nirvana. A quantitative framework for estimating and assessing true impact is needed for informed decisions.

Measuring the quantitative impact of a software development process change is hard. Measurable change takes weeks or months to evolve, and there is little in the way of control group – change is implemented and the outcome if that change wasn't performed isn't an interesting or easily discernable metric. This paper presents one technique for quantitatively estimating the potential economic outcomes both before and after a change has been implemented.

The basis for the method described here is probabilistically simulating the impact of changes in cycle-time samples from a prior project to a completed project using new methodology. To estimate the potential payoff for a new process, existing cycle-time samples can be discounted by fixed percentage amount to simulate the financial return for hypothetical reductions (10%, 25%, for example). Once change has occurred, actual results can be compared to the predicted data to validate the difference and improve modeling efforts on future initiatives.

This paper also looks at the impact of process change on cycle-time over the last 40 years to determine if there is a pattern. Understanding the forces at work causing changes in cycle-time distribution data throughout process improvement over this time hints at what management strategies will give the biggest return on investment. This knowledge helps choose the contextually correct process strategy in a compelling and confident way.

The goal of this paper is two-fold:

1. Discuss the cycle-time distribution patterns and parameters for different known software development processes.
2. Demonstrate the use of cycle-time data as a comparative tool for calculating potential and actual economic benefit of process change.

2. Structure of this paper

This paper consists of three major sections –

1. Cycle-time distribution analysis, and how development process can manipulate the shape and uncertainty of this distribution
2. Probabilistic modelling of software processes using cycle-time for delivery estimate and how to formulate an economic model for process change.
3. Case study: Practical application of cycle-time analysis as presented in an economic benefit project for an industry client.

The first part of this paper introduces cycle-time in software development processes and develops the theory behind why cycle-time value distribution consistently follows very similar patterns across teams and organizations.

The second part of this paper shows how through statistical modelling, the likely improvement of process changes can be simulated and an economic argument used to justify the effort of process change.

The third section will demonstrate and show the application of a cycle-time comparative technique used in industry.

3. Cycle-time distribution analysis

3.1. Defining cycle-time

Cycle-time is defined as the time between starting date and completion date of work items. In software development this is most often measured in days, but can be in hours for teams with small work items that are started and completed in the same day. This paper always presents cycle-times in days.

No consistent definition of work item "start" and "complete" has emerged as standard in IT process. Definition often differs between departments and teams within an organization. For the purposes of this paper, any consistently defined start and complete date is acceptable as long as it is consistently applied when capturing cycle-time data. Cycle-time is a good candidate metric for prediction and comparison because it is an elapsed time measurement that is not measured in abstract units (such as function points, story points and velocity – common in agile).

Increasingly, electronic tools are used to manage the software development process and automatic capture of cycle-time data is common-place. The richness of this data hasn't been surfaced beyond scatter plots and for some tools a histogram as eye candy. This paper describes how this data can be leveraged for predictive purposes.

If no policy and/or process is currently in place for capturing cycle-time data then the following guidance is offered. These are also the definitions used in this paper -

1. Capture the date that work items are first created as an idea. This data is useful for understanding if there is a decrease in "first idea" to customer delivery. This will be referred to as *Created Date* in this paper.
2. Capture the date that work items are prioritized for work to begin. This is the date that a company has committed to delivering the work item, or has to put "effort" into creating the intended result from the work item. This will be referred to as *Started Date* in this paper.
3. Capture the date work is ready for delivery to a customer. This is referred to as *Completed Date* in this paper.
4. Capture the date work items are first exposed to customers in a usable way. This is referred to as *In Production Date* in this paper.

Given these definitions of the times, the following definitions will be defined in this paper as –

(1) Lead-time = In Production Date – Created Date

(2) Cycle-time = Completed Date – Started Date

Lead-time (1) is defined as the time between when work description is created, and that work is delivered to production. Cycle-time (2) is defined as the time between when hands-on work begins on a piece of work until it is considered complete.

This paper focuses on cycle-time in its examples and case study. It is feasible, and likely that the same factors of probability distribution are at play at a lead-time level and this will be the subject of future work.

3.2. Cycle-time probability distribution

3.2.1. Analysis of industry data. Norden [1] in 1963, then Putnam and Myers [2] in 1978 empirically established that software projects follow a Rayleigh density distribution in staffing estimation and effort. Kan [3] in 2002 extended the use of the Rayleigh model to include defect removal rate and remaining defect prediction. The Rayleigh distribution is a specific form of the Weibull distribution (shape parameter of 2.0), and this paper proposes that cycle-

time follows the Weibull probability distribution with lower shape parameters from 1.0 to 1.6 depending on process and inter-team dependency factors.

Cycle-time histograms obtained during many commercial client engagements established the prevalence of left-skewed density distributions. The lack of this pattern in software development cycle-time is often an indicator of poor data capture. Although it is not essential to know the exact distribution curve for modelling an existing process when actual data samples can be used when simulating [13], curiosity drove my initial attempts to determine what distribution and why. Candidate probability distributions are Gamma, Weibull, Rayleigh and Log-Normal as shown in Figure 1 fitted to a typical agile team in a commercial context.

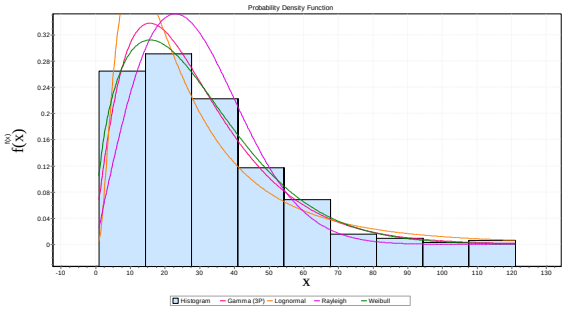


Figure 1 – Potential probability distribution fits against a typical commercial data-set

When faced with multiple potential correct distributions fitting historical data, the "safest" is chosen. For forecasting dates, longer is safer – if a project is valid delivering later, then it is likely still a valid investment delivered early. The Weibull distribution is the wiser choice having a heavier tail, meaning the decay in higher values is at a slower rate (higher values have higher probability) when compared against Log-Normal. It also proved a successful candidate when used as an initial estimate on prior commercial projects.

Written evidence suggesting Weibull is appropriate for modeling similar problems was found by Fente, Schexnayder, and Knutson [5] who describe Weibull as a good candidate for modeling construction and engineering. And McCombs, Elam and Pratt [4] chose Weibull as most appropriate for modeling task durations in PERT analysis.

McCombs et al [4] and Fente et al [5], outlined the desirable attributes of an acceptable distribution for modeling task durations -

1. It is continuous.
2. It has finite endpoints.
3. It has a defined mode between its endpoints.
4. It is capable of describing both skewed and symmetric activity time distributions.

The Weibull distribution is a good choice for all but point 2 - finite endpoints. It could be argued that this isn't a necessarily desirable attribute for a cycle-time distribution; Long delays of very low likelihood occur in the real software development world. Extremely large values have even extremely small probabilities, and when used in a probabilistic prediction, their influence should they occur is minimal. McCombs et al [4] noted that although not a mathematical property of the Weibull distribution, commercial practicalities eliminate (ridiculously long) non-finite endpoints. Tasks taking far longer than anticipated will more often be identified and solved with resourcing and attention of causes.

Fitting actual commercial data and finding confirming prior research increases confidence that Weibull distribution is the most likely candidate. Confirming this with a plausible simulation of a typical real world software development process is valuable to understand limitations, ramifications and potential utilization.

3.2.2. Process factors causing skewed distribution. The software development process involves taking a description of ideas and creating a manifestation of those ideas in working computer code. Large ideas are meticulously broken down into smaller ideas, and implemented incrementally. Each work item builds on the previously completed computer code.

XP, Scrum, Kanban, and Agile, all decompose work items into smaller batches of work for delivery, but differ in approach. These batches of work go by various names: user stories, cards and work items are commonly used (this paper always uses the name work items for no reason other than consistency).

Work items pass through a variety of steps from idea to solution. Although it is simple to think of a software development process like a factory production line where work follows a pre-defined step-by-step process. The key difference is that every work item is different in a functional way from the others. Each work item is solving a different problem. Rather than building many of the same things as in factory production, software development is building consecutive one offs. Efforts to model software development in similar ways to traditional manufacturing production line thinking fails to embrace this key difference.

Traditional repetitive manufacturing process cycle times will lead to a Gaussian response. Work-items flow through fixed sequence of assembly with some

bounded variance. This process yields a Normal distribution due to the Central Limit Theorem.

This expected Gaussian result is engrained in quality management practices. Shewart [7, 8] and Demming [9, 10] make good use of anticipated Normal distribution in manufacturing for statistically managing quality. For example, work output is closely monitored for process deviation, often through policy to keep variability within certain ranges of sigma (standard deviation) through standardization.

Driving for standardization in creative pursuits often leads to erosion of quality and inventiveness as teams strive to meet the pre-set control limits at great effort and at any cost. Acemoglu et al [14] study this tension between standardization and innovation. For work that is constant innovation, cycle-time for each work item will be inherently variable and unlikely to form a Normal distribution pattern.

Software development process has the following traits that set it apart from standard manufacturing process and need to be modelled specifically –

1. Work items follow one of many possible sequential process, not one fixed process for all items. Some work items skip entire steps.
2. The effort for each process step is different for each work item. Some harder, some easier due to the amount of new and known concepts.
3. The delays each work item might encounter are different and often unrelated to the work item itself (key person on vacation, awaiting another item to finish).
4. Completing some work items identifies new work. Missed requirements, un-identified complexity and defects are examples.
5. The uncertainty of effort and delays of each work item means that the system continuously operates in an unstable state, where constraints move, emerge and resolve.

To simulate the simplest process that causes asymmetry in cycle-time, a simple Monte Carlo model can be built. This model explains how initially known work and a combination of potential, but not guaranteed delays probabilistically convolve into a Weibull distribution.

The hypothetical model simulates a fixed amount of work-items being constructed with some uniform variance. Impacting their on-time completion is the possibility of 1 to 5 delays occurring, each with a 25% chance of occurrence. These delays are simulated as an amount of extra work if they occur, or have no impact if they don't occur. Delays in software are a combination of waiting time and discovered work, but this model simply increases work item time by adding work.

Figure 2 shows the result as delays are added in Histogram form. The Y-Axis is count (probability), and the X-Axis is time (earlier to the left, later to the right). Peaks emerge and mix based on the binary permutations of none, some and all delays occurring.

When there are no delays, the result is a Normal distribution (or would be if modelled with more cycles). When the first delay is added, 75% of the outcomes remain un-affected, and 25% incur the delay as our binomial logic would suggest. When two delays occur, there are now 3 possible peaks: no delays occur (left-most), one or the other delay occurs (middle), or both delays occur (rightmost, with a chance of 6.5%). This pattern continues until around 4 delays when it is more likely than not that one delay will impact every work item and it becomes the mode position. By 5 delays, the Weibull shape is clearly forming. The blank spaces between the peaks of this simulation would be populated with the uncertainty of the delay times (not all identical) and the probabilities (not all 25%) in the real world.

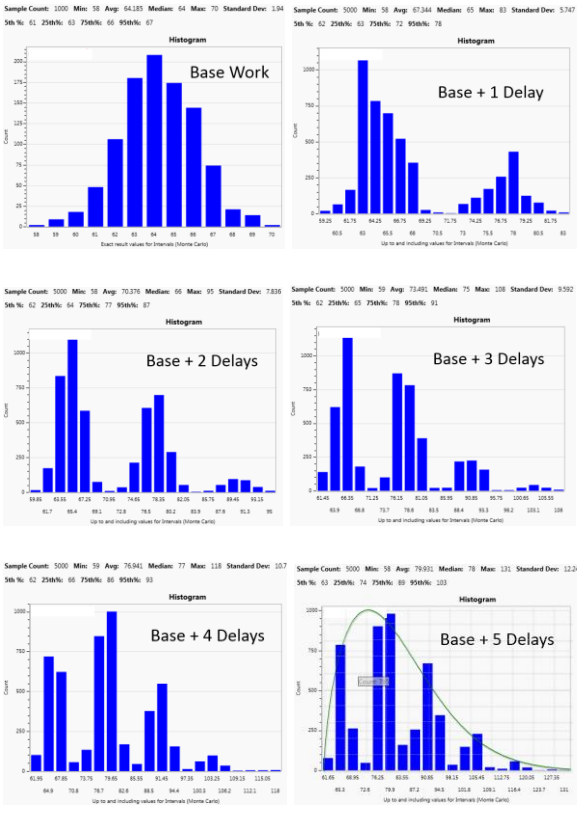


Figure 2 – Simulation of the impact of increasing number of identical delays each with probability of occurring p=0.25

Although this is a simplistic model, it does help explain and visualize the process of uncertain work effort and delays that are un-related to the work item. For example, often these delays are staffing related. Work items stall awaiting a specialist staff skill to become free. Or system related, where work items are stalled waiting for a test environment to free up, or waiting for another team to deliver a dependent piece of work. These delays have little to do with the work item task at hand, they are just bad luck for some work items and not others. The combination of just a few of these delays impacting some work items but not others causes the asymmetry. It's unlikely that every delay factor hits a given work item. It's unlikely no delays hit an individual work item. There will most often be a mix of a few delays.

The exact percentage likelihood of each risk cannot be accurately estimated in advance, but can be measured historically. Analysis of these probabilities and impact should drive process selection decisions and budget to solve the most common and impacting delays. Heuristics to isolate the most impacting delays is a vibrant area of commercial interest.

3.2.3. Historical observed cycle-time reduction. A supporting factor that Weibull may be the correct distribution for cycle-time is its consistent prevalence over the last 40 years. The distribution shape common in the 1970's was wider (larger shape parameter), but it is also common in 2014 with a narrower shape. The hypothesis of this paper is that process improvements are correlated to the Weibull distributions shape parameter.

During the 1970's to 90's a general development process called Waterfall was common. The characteristics of this development pattern were up-front design, followed by coding, followed by testing. Projects were inhibited from exiting one phase until the prior phase was completed. This was the era Norden [1], Putnam [2] and Kan [3] documented the Rayleigh distribution. They showed empirical evidence this was the case during projects they oversaw during that era. Figure 3 shows a density function for the Rayleigh distribution. Rayleigh is part of the Weibull family, with its specific characteristic being a shape parameter is 2.0.

Agile methodologies (XP, Scrum, Kanban) became prevalent in the late 1990's to the current time. For these processes the Weibull distribution moves left and narrows. The Weibull shape parameter is decreasing. Figure 4 shows a Weibull distribution with a shape parameter of 1.5, in the center of the 1.3 to 1.6 range seen in available cycle-time analysis by the author. Further research is needed to confirm other sources see similar results.

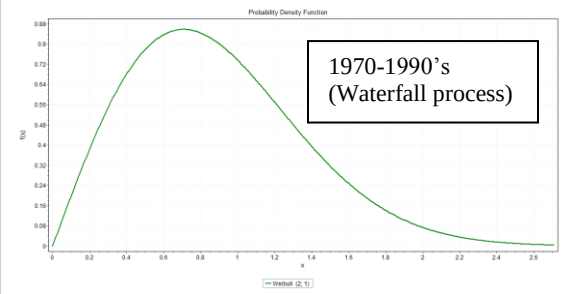


Figure 3 – Rayleigh Distribution (shape = 2.0)

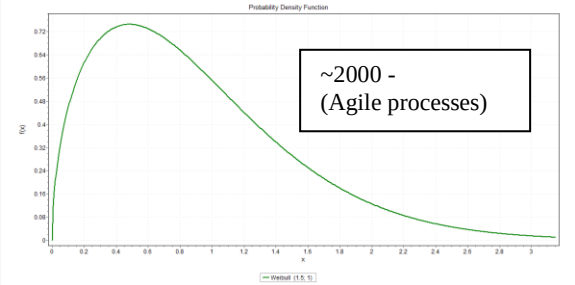


Figure 4 – Weibull Distribution (shape = 1.5)

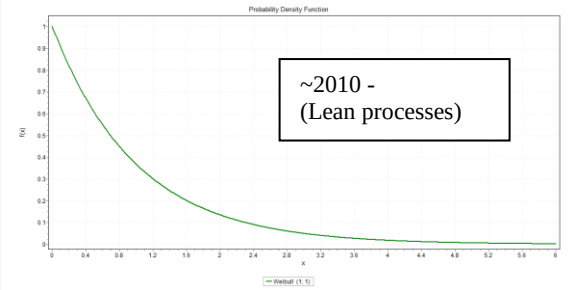


Figure 5 – Weibull Distribution (shape = 1.0)

The lowest limit seen in commercial practice, nears an Exponential distribution as shown in Figure 5. The Exponential distribution is of the Weibull family, its shape parameter being 1.0. It has been seen on teams that have few or no external dependencies and the work performed is mostly repetitive. Operations teams and teams responsible for releasing code to production. The hypothesis is that these teams experience few or no external delays, and work have little inventiveness. This is consistent with the simulation in Figure 2. With none, one or two delays resulting in a distribution with a left mode, and generally more exponentially distributed. It's likely that for these teams measuring in hours or minutes will zoom in on the leading edge of the Weibull that is invisible at the day granularity for cycle-times. Teams in this area have often laser focused on eradicating delays and queuing time using principles from the Lean manufacturing world (reducing WIP, eliminating waste, etc.). Kanban

teams have been observed approaching this ideal when in a predominately operational capacity.

The narrowing of the shape parameter has important impacts on the variability of forecasting using cycle-time data. Teams with a lower shape parameter (narrower curve) have less variance, and will be able to forecast and promise with higher certainty (relative to the unit they estimate in). The hypothesis is that some iterative practices or Agile techniques are reducing the shape parameter due to iterative delivery (cycle-times are lower), and that other Agile practices are decreasing the shape parameter by eliminating uncertainty. The ability to put a dollar figure on this reduction of cycle-time and less uncertainty is paramount to justifying the expense of implementing any change in agile practice.

3.3. Implications of Weibull Distributed Cycle-times

3.3.1. Errors in control chart control limits. The most visible implication of non-Gaussian cycle-time distribution is the setting of service level agreements (the time development organizations promise to deliver within) or other control limits. Shewart [7, 8] and Deming [9, 10] introduce and discuss the use of control charts to maintain quality and determine when intervention in a process is needed (beyond normal variation). One popular chart is Shewart's control chart, a scatterplot with horizontal upper and lower bounds statistically calculated. When samples from production move beyond these limits, action is warranted. Traditional calculations for these limits assume normality which is correct for general repetitive manufacturing, but not software development cycle-times as shown. The computed control limits shown in graphs within almost all commercial task management systems is incorrect for cycle-time. Teams are being held to a service levels that are erroneous.

To demonstrate the impact of this error on team expectations, random numbers were generated for a Weibull distribution with a shape parameter (α) of 1.5 and a scale parameter (β) of 30. Table 1 shows the various results of target intervals computed three different ways. The Actual Weibull calculation (1) is based on the Weibull formula and is the correct value if the data truly conforms to the specified Weibull parameters (these would be close in our example but not exact – generating perfect random numbers is difficult). The next row (2) shows the erroneous Standard Deviation calculation (when applied to Weibull data) that adds the Standard Deviation (σ) the mean (μ), 1, 2 and 3 times. The lower limit calculations below the mean (subtracting σ from μ) have been omitted because they go below zero. Some major commercial tool vendors have plotted the lower control limit below zero without noticing this is invalid. Cycle-times cannot go below zero, until time can be reversed.

Table 1 – Incorrect x(P) on Weibull (shape k = ~1.5, scale γ = ~30)

	μ + *1σ*	μ + 2σ	μ + 3σ
Target p	0.683	0.954	0.997
(1) Weibull formula $f(x) = \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}$	31.342	61.217	94.194
(2) Using StdDev in Excel	43.686	61.567	79.447
(3) Using Percentile in Excel	30.819	60.677	96.155

The formula using Standard Deviation (row 2) is widely incorrect at +1 and +3 sigma (*σ)*. If a team was expected to perform within a μ + 3σ upper limit, typical off-the-shelf charting controls (and Microsoft® Excel) would calculate 79 days as the target. The actual performance solved using the Weibull PDF is 94 days. The team would be at a 15 day disadvantage due to poor mathematics.

The alternative is to compute the percentiles directly, the results shown in row (3). This is a simple remedy for the issue and acceptable with the computing power we have today (but didn't back in Shewart's [7,8] time circa 1939 where Standard Deviation and percentile values were computed using printed tables, not computers). The formulas used in the closer approximations seen in Table 1 in Microsoft Excel are -

=PERCENTILE.INC([cycle-time data], 0.683)
=PERCENTILE.INC([cycle-time data], 0.954)
=PERCENTILE.INC([cycle-time data], 0.997)

3.3.2. Process decision implications. Figure 2 presented a simple model showing the impact of delays on delivery time. The uncertainty of time it would take to create the work items with no delays quickly shifts to the right. Depending on the final width of cycle-time Weibull distribution (its shape parameter) and its stretch (its scale parameter), different process decisions might be appropriate.

The first step required to give this advice is determining what the actual shape and scale parameter are given historical cycle-time data. Many commercial packages exist to estimate the Weibull shape and scale parameters from a set of samples (Easyfit, CrystalBall, @Risk are common). R [6] is a

freely available statistical application and has libraries that will estimate the best fit parameters as shown in Listing 1.

```
R code:
require(MASS)
fit<-fitdistr([data frame],"weibull")$estimate
fit

Output:
  shape    scale
1.243665 9.828095
```

Listing 1 – R Code to estimate Weibull parameters from historical samples

Having determined the shape and scale parameters, likely traits and process advice can be broadly stated. For example, higher shape parameters are indicative of teams being impacted by more delays, and for longer (as seen in Figure 2, more delays the more the fatter the distribution). External team dependencies have been commonly seen as a major root cause for these teams. Teams with low shape parameter values approaching 1.0 with low scale parameters do many smaller items fast. This is typical of operations and support teams who undertake critical, but repetitive work. Automating the repetitive and critical tasks is prudent for these teams. Figure 7 shows some high level traits and process advice based on a matrix of shape and scale parameters. Advice is very contextual, and this isn't an exhaustive list. Further research and confirmation is needed.

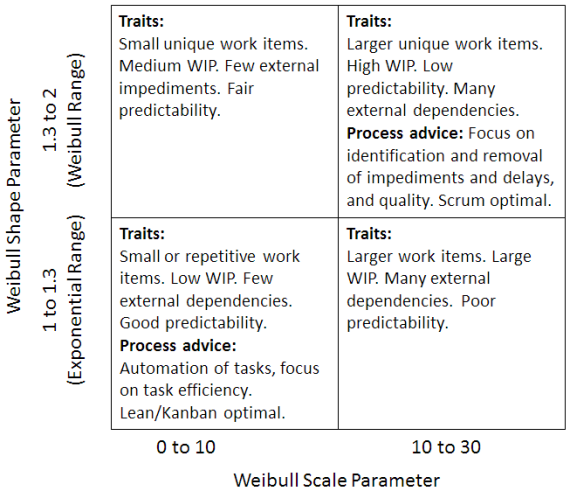


Figure 7 – Suggesting process advice based on Weibull Scale and Shape parameters.

In theory, the shape parameter of the Weibull should not be < 1.0 in the software development estimation context. In software development, there seems to be good logic expressing that the longer a work item remains incomplete the more likely they are to stay that way. Once a work item has been impeded and the team takes on other work whilst waiting, pressure and focus is on the new work. Based on the Hazard function of the Weibull distribution (related to Survival analysis), shape parameters < 1.0 would be at increasing Hazard (reverse logic for completion, hazard in this case is work finishing). Work items would have a higher chance of being completed the longer they are impeded. Stories of this occurring in real teams seem rare. Not impossible and future work will attempt to investigate this issue further to see what process circumstances might cause this.

4. Software process impact modelling

Having established that software development practices shape the variability of cycle-time, computing a likely impact of this cycle-time in financial terms can provide evidence when process change is prudent. The first step is to model software projects using cycle-time to forecast a baseline delivery date of a prior project that matches what occurred (actual delivery).

Modeling a prior software project using cycle time requires the following data –

1. The known start date of the project.
2. The known end-date of the project.
3. The number of work-items delivered.
4. The cycle-time values for each work-item.

Inputs 1 and 2 are known based on team knowledge, and inputs 3 and 4 are obtained from the electronic tracking system used by the organization. The cycle-times samples are checked for obvious erroneous outliers, and that they follow a Weibull distribution for data quality confidence.

Figure 8 shows the basic process for generating a probabilistic forecast based on the historical cycle times. The goal is given the known start date, tune the model so that a given confidence level (0.85 is common) forecasted release date matches the known actual release. The forecast we produce will be elapsed days, and this target is calculated as the number of days between actual release and actual start using the logic –

(3) Target forecast days =
Known actual release date – Known start release date

Cycle-time samples are combined with the known amount of work completed using the Monte Carlo process described in Figure 8. Random samples of

historical cycle-time are bootstrapped (sampled with replacement) as described by Efron and Tibshirani, 1994 [11]. Sets of these samples the same size as the completed work-item count are summed. This is repeated many times, often thousands of trials. Some trials happen to randomly choose all lower numbers from the samples, other trials get all higher values, but mostly a mix of values. By analyzing the results, probability of any one outcome can be computed by the percentage of trials that were equal to or less the desired value.

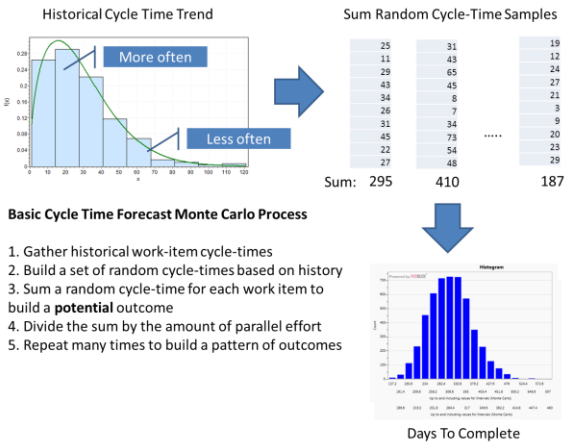
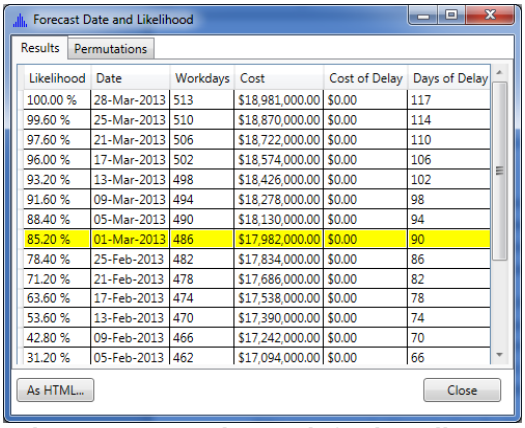


Figure 8 – Cycle-time Monte Carlo process for forecasting project completion

More than one work item was in-progress at a time, and this is modeled by dividing the total summed cycle-times in each trial by an average amount of parallel effort. Average parallel work effort is computed experimentally by increasing or decreasing an initial guess until the forecasted number of days to complete matches that of formula (3) Target forecast days at a given certainty (0.85 common). The certainty is the value at the 85th percentile value of all trials after being divided by the parallel effort.

The results shown in this paper were computed using a commercial Monte Carlo simulation tool, KanbanSim and ScrumSim by Focused Objective [12]. Figure 9 shows a typical result table. The advantage of this tool is it correctly computes calendar dates from elapsed days, and can compute the cost of delay based on missing target dates. The process it employs is no more complex that described in this paper.



Likelihood	Date	Workdays	Cost	Cost of Delay	Days of Delay
100.00 %	28-Mar-2013	513	$18,981,000.00	$0.00	117
99.60 %	25-Mar-2013	510	$18,870,000.00	$0.00	114
97.60 %	21-Mar-2013	506	$18,722,000.00	$0.00	110
96.00 %	17-Mar-2013	502	$18,574,000.00	$0.00	106
93.20 %	13-Mar-2013	498	$18,426,000.00	$0.00	102
91.60 %	09-Mar-2013	494	$18,278,000.00	$0.00	98
88.40 %	05-Mar-2013	490	$18,130,000.00	$0.00	94
85.20 %	01-Mar-2013	486	$17,982,000.00	$0.00	90
78.40 %	25-Feb-2013	482	$17,834,000.00	$0.00	86
71.20 %	21-Feb-2013	478	$17,686,000.00	$0.00	82
63.60 %	17-Feb-2013	474	$17,538,000.00	$0.00	78
53.60 %	13-Feb-2013	470	$17,390,000.00	$0.00	74
42.80 %	09-Feb-2013	466	$17,242,000.00	$0.00	70
31.20 %	05-Feb-2013	462	$17,094,000.00	$0.00	66

Figure 9 – Sample result for baseline Monte Carlo simulation using cycle-time

4.1. Hypothetical cycle-time reduction impact

Calculating the impact of reducing cycle-time by a fixed percentage is easy once a baseline model has been produced. Cycle-time samples are discounted at fixed percentage when randomly sampled in a trial. For example, multiplying each sample by 0.9 simulates a 10% decrease in cycle-time across the board. Forecasts using the discounted cycle-time show an earlier delivery date and the value of this earlier release calculated (lower development cost plus extra revenue).

Table 2 shows the cash flow impact when cycle-time samples were reduced by 10% and 20%. An updated forecast release date was used to calculate a cost saving (development expenses) and additional revenue forecast based on being earlier to market. The total cash-flow benefit shown is based on financial revenue predictions obtained from the financial team. The baseline shows some revenue was made but the project missed a seasonal window and paid dearly. Even small percentage decreases in cycle-time can make a big cash-flow impact.

Table 2 – Example cash flow Improvement

Cycle Time	Forecast Date	Forecast Cost	Cash flow Benefit (cost savings + FY revenue)	
Current	15-Jul	1000K	$0 + $60K = $60K	0%
10% Decrease	27-May	912K	$87K+ $90K = $177K	296% Better
20% Decrease	4-Apr	820K	$120K + $145K = $265K	442% Better

5. Case study: Process Economic Impact

A large commercial client (approximately 200 staff in the division analyzed) wanted to calculate the impact of a recently introduced process change. Although the IT department felt that a newly introduced process considerably improved completed work throughput, there was some healthy skepticism within the financial controllers who glibly stated "show us the money."

The initial question to answer is "did the process change make any impact at all?" If so, how much? To answer these questions, a baseline model for a prior project (called prior in this paper) and test project (called test in this paper) were built using the process described in section 4. These models were tuned as described in section 4 to accurately match the known outcomes.

Having a baseline for the prior and test project with forecasts, the cycle-time samples were swapped between the models. Prior samples were modeled in the test project, and the test project's cycle-times were used in the prior project. New forecasts were produced giving new hypothetical forecasts if the new process was used in the prior, or the old process was used in the test. Figure 10 shows the general process for tuning against actual dates then swapping cycle-time histories to get the modeled outcomes.

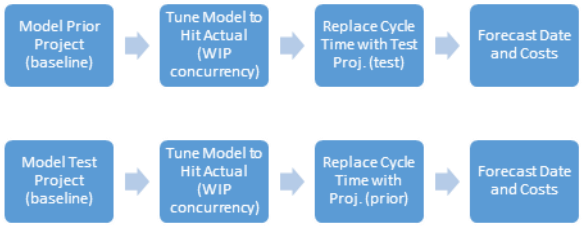


Figure 10 – Process for modeling known and modeled (hypothetical) forecasts

This straight swap was appropriate in this context because teams were stable and the prior and test projects were similar in complexity. This was confirmed by looking at the work-item count and effort for features delivered in both. If the teams or projects were significantly different, then this analysis would be erroneous.

Testing the impact of the new process on the prior project demonstrated the results in Table 2. With the new cycle-time history bought about by process change, there would have been an almost 4 month (110 day) improvement in delivery time, with the reduced staff carrying cost alone saving $4M. It was decided that the impact of being on the market earlier in gained revenue was hard to estimate, so it was omitted. This means that the outcomes seen understate the losses. They do not account for cost of delay on the projects delivered late, or the cost of delay for the next project that was delayed. Staff carrying cost was calculated as the per day cost of the teams implementing the project. The calculation was simply the yearly IT budget divided by 365 days.

Table 2 – Impact if new process WAS used in the prior project

Process	Target	Delivered	Cost
Old (actual)	1-Dec	1-Mar +90 days	$18M
New (modeled)	1-Dec	11-Nov -20 days	$14M -$4M

The second part of the analysis was to estimate the outcome if the new process wasn't implemented for the test project. The cycle-times for the prior project were used in the model for the test project. The outcome is shown in Table 3. The new process saved at least $1.4M, again only accounting for staff carrying cost.

Table 3 – Impact if new process WASN'T used in the test project

Process	Target	Delivered	Cost
Old (modeled)	31-Dec	5-Feb +36 days	$8M +$1.4M
New (actual)	31-Dec	31-Dec On-time	$6.6M

The combined message to the executive teams was that if the new process was in place for the prior project -

a. Delivered on-time rather than 3 months late
b. $4M decrease in development budget

And if the new process wasn't implemented for the test project -

a. Test project delivered 1 Month late
b. Increase of $1.4M development budget

Due to the compelling nature of these forecasts even if halved, the decision to fund and roll-out the new process to the entire organization was happily accepted by all financial and IT staff.

The ability to quickly model the financial impact using the cycle-time data on-hand was central to gaining executive sponsorship in the process change initiatives. Three years after this analysis, the teams in question still model, assess viability and measure resulting impact of all process changes.

Confidential data available on request.

6. Conclusion

This paper has outlined a method for calculating the financial impact of software development process change. Practitioners can choose to predict likely impact before a change is implemented, and/or assess its impact post change process implementation.

The software industry has a history of documented use of Rayleigh and Weibull distributions for various problems. Prior examples exist for predicting remaining defects in the field and staff capacity planning. This paper continues the Weibull tradition with a theory of why cycle-time for software development follows this distribution within a narrow range of parameter values. This paper provides new sources of evidence in support of this claim –

1. Theory of why Weibull distribution is the outcome of the typical process constraints seen in software development.
2. The Weibull shape parameter can be seen to correlate with progressive process changes over forty years.

Knowing cycle-time follows the Weibull distribution allows informed decisions to be made in risk mitigation and process change decisions. This paper makes the following advice –

1. Calculations on cycle-time data should not assume a Normal Distribution. This is evident in commercial tooling vendors who show erroneous confidence level limits on control charts.
2. Agile processes has compressed the shape and scale of the cycle-time distribution, decreasing its variability.
3. Computing the shape and scale parameters of the cycle-time distribution allows informed process recommendations.

Probabilistic modelling allows financial impact of process change to be estimated. Monte Carlo modeling using captured cycle-time samples and known project outcomes allows simple what-if experiments based on hypothetical cycle-time improvements. This is a rapid way to quantitatively determine economic outcome.

The commercial case study shown in this paper demonstrates how quickly a financial argument can be made for the continually challenging the current process and investing in improvements. The payback is rapid.

7. References

[1] Norden, P V, Useful Tools for Project Management Research, Operations Research in Research and Development, John Wiley and Sons, NY, 1963

[2] Putnam, L H, and W. Myers, Measures for Excellence: Reliable Software on Time, Within Budget, Yourdon Press, Englewood Cliffs, NJ, 1992

[3] Kan, Stephen H, Metrics and Models in Software Quality Engineering – 2nd Edition, Addison Wesley, Upper Saddle River, NJ, 2002

[4] McCombs, Edward L.; Elam, Matthew E.; and Pratt, David B. (2009) "Estimating Task Duration in PERT using the Weibull Probability Distribution,"Journal of Modern Applied Statistical Methods: Vol. 8: Iss. 1, Article 26. Available at:http://digitalcommons.wayne.edu/jmasm/vol8/iss1/26

[5] Fente, J., Schexnayder, C., & Knutson, K. (2000). Defining a probability distribution function for construction simulation. Journal of Construction Engineering and Management, 123(3), 234-241.

[6] R Core Team (2014). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

[7] Shewhart, W. A. (1931). Economic Control of Quality of Manufactured Product.

[8] Shewhart, W. A. (1939). Statistical Method from the Viewpoint of Quality Control

[9] Deming, W. E. (1975). "On probability as a basis for action". The American Statistician 29 (4): 146–152.

[10] Deming, W. E. (1982). Out of the Crisis: Quality, Productivity and Competitive Position.

[11] Efron, B., & Tibshirani, R. J. (1994). An introduction to the bootstrap (Vol. 57). CRC press.

[12] KanbanSim and ScrumSim, A software environment for modeling and forecasting agile software development projects. Focused Objective LLC. Seattle, USA. URL http://www.focusedobjective.com

[13] Magennis, Troy (2012) "Managing Software Development Risk using Modeling and Monte Carlo Simulation". Lean Software and Systems 2012 Proceedings: 32–52. url: http://leanssc.org/files/2012-LSSC-Proceedings.pdf

[14] Acemoglu, Daron & Gancia , Gino & Zilibotti , Fabrizio, "Competing engines of growth: Innovation and standardization." Journal of Economic Theory 147 (2012) 570–601.