Programming Optimization - Operation Compression, Part 2

By Joseph Juma

Description

In the my previous writing, "*Programming Optimization - Operation Compression*" I walked through how one can use automation to improve their software throughput, as well as equations for calculating one's improvements and caveats to watch out for. However, I did not cover every possible consideration, and their subsequent calculations, that might arise when considering and measuring optimization via operation compression. I will be going over those here. This paper will make heavy usage of the formula in the previous one, so I'd advise having a copy of that one on hand when reading this one.

However, if you don't have that document on hand here is a brief recollection of formula and variables that will be useful.

1. α is the amount of time a certain programming task takes.
2. ϵ is an estimate of how long it will take once optimized.
3. β is how frequently that task happens within a certain length of time, like a week or year.
4. ζ is the amount of time spent on programming in that same period of time.
5. Δ is the difference in time between the original approach and the optimized approach.
6. δ_t is the amount of time spent to program the same amount once the optimization is in effect.

7. δ_p is what percentage of the original time the new approach will take, which is usable as “the optimized approach takes δ_p percent of the original time.”
8. $\delta_p^{'}$ is usable as, “the optimized approach is $\delta_p^{'}$ percent faster than the original.”

$$\Delta = \beta(\epsilon - \alpha)$$
$$\delta_t = \zeta - \beta(\epsilon - \alpha) = \zeta - \Delta$$
$$\delta_p = \frac{\beta\epsilon}{\beta\alpha}$$
$$\delta_p^{'} = 1 - \delta_p$$

and,

1. Δ_p is usable in the sentence, “with the optimization, we can get the same amount of work done in Δ_p percent of the time.”
2. $\Delta_p^{'}$ is usable in the sentence, “the optimized approach has made our week $\Delta_p^{'}$ percent faster.”

$$\Delta_p = \frac{\Delta}{\zeta}$$
$$\Delta_p^{'} = 1 - \frac{\Delta}{\zeta} = 1 - \Delta_p$$

Errata

I would also like to point out that one of these equations requires a slight change, specifically δ_t. In the prior article I briefly mentioned that when you do the calculation for Δ you will get a negative value, as this is the net change in the time, and negative means saved time. However, if you freely plug in a negative value into the formula $\delta_t = \zeta - \Delta$ then you will be *adding* the time to the original amount, rather than subtracting it. To adjust for this you can either do one of two things, but don’t do both.

1. You can take the absolute value of Δ which I will denote, $\hat{\Delta}$
2. You can turn the formula $\delta_t = \zeta - \Delta$ into the formula $\delta_t = \zeta + \Delta$
3. I will use the $\hat{\Delta}$ approach in the writing below.

Mathematics

In the previous paper I mentioned that many efforts for optimization may see what feel like relatively small numbers such as 2% or 3% gain for each optimization you create, but how these compound to create a greater effect. I didn't mentioned how to calculate this net-change over many optimizations though, and so I will walk through that here.

First, let's consider the original formula for measuring reduction in time (Δ) which uses the variables α, ϵ, β to respectively identify the original length of a task, estimation of the reduced length of the task after automation, and the frequency of the task per measured time. This equation is only useful for a single improvement, but if we want to instead measure the improvement for multiple task optimizations then we will need to use this equation for *each* task, and then combine the calculations together. Do note I will only be covering the case where each task is *completely separate* from one another, as tasks which overlap offer a myriad of complexities and so it is often better to clearly define tasks into non-overlapping divisions.

First, if we want to measure multiple tasks we will need a list of tasks and then a list of the length of time each task takes. We can create an ordered list of the task lengths denoted by A where if we want to refer to the 3rd task's length we would write α_3, or for the ith task's length, α_i. Then we will want similar lists for both the estimated improvements, denoted by E where the ith estimated improved length is denoted by ϵ_i, and for frequency as well, denoted by B where the ith task frequency is denoted by β_i.

We can then say for the i-th task, the value of Δ is denoted by Δ_i and the new equation looks like,

$$\Delta_i = \beta_i(\epsilon_i - \alpha_i)$$

And we can then calculate how much net change all of the improvements creates by just adding together all of the calculated values of Δ_i for the number of tasks that were optimized, which is denoted by n. The equation for adding together all of these just looks like,

$$\Delta_\Sigma = \sum_i^n \Delta_i = \sum_i^n \beta_i(\epsilon_i - \alpha_i)$$

where Δ_Σ is just the sum of the net difference of every optimization. Given Δ_Σ we can then update our value δ_t when stating, “We can get the same amount of work done that used to take us ζ in δ_t now” by swapping the original Δ for the new Δ_Σ, which creates the following equation,

$$\delta_t = \zeta - \hat{\Delta}_\Sigma$$

Keep in mind that $\hat{\Delta}_\Sigma$ just means the absolute value of Δ_Σ. This equation can then be expanded to,

$$\delta_t = \zeta - |\sum_i^n \Delta_i| = \zeta - |\sum_i^n \beta_i(\epsilon_i - \alpha_i)|$$

For example, let’s say we believe we can optimize 3 tasks. The first task takes 3 minutes or 180 seconds, happens 20 times a week and we believe we can reduce it to take 10 seconds.

$$\alpha_1 = 180s, \epsilon_1 = 10s, \beta_1 = 20$$

The second task takes 2 minutes or 120 seconds, happens 10 times a week and we believe can be reduced to take 15 seconds.

$$\alpha_2 = 120s, \epsilon_2 = 15s, \beta_2 = 10$$

And the third task takes 5 minutes or 300 seconds, happens 30 times a week and we believe it can be made to take only 5 seconds.

$$\alpha_3 = 300s, \epsilon_3 = 5s, \beta_3 = 30$$

We know that over a week we spend 20 hours, or 72,000 seconds programming. So the new time is calculated in the following,

$$given\ \zeta = 72000s$$
$$given\ \Delta_i = \beta_i(\epsilon_i - \alpha_i)$$
$$\Delta_1 = 20(10s - 180s) = -3400$$
$$\Delta_2 = 10(15s - 120s) = -1050s$$
$$\Delta_3 = 30(5s - 300s) = -8850s$$
$$\hat{\Delta}_\Sigma = |\sum_i^n \Delta_i| = |-3400s - 1050s - 8850s| = 13300s$$
$$\delta_t = \zeta - \hat{\Delta}_\Sigma = 72000s - 13300s = 58700s$$

So it will now take 58700 seconds, or 978 minutes and 20 seconds, or 16 hours 18 minutes and 20 seconds per week to complete what used to take 20 hours per week. This saves 3 hours 41 minutes and 40 seconds per-week. Over a year (52 weeks) this will save a total of 192 hours 6 minutes and 40 seconds. That's over a month of full-time work.

If we want to then measure the percentage change (δ_p) we can take a similar approach, where the percentage change for a given task is denoted by $\delta_{(p,i)}$,

$$\delta_{p,i} = \frac{\beta_i \epsilon_i}{\beta_i \alpha_i}$$

and the speed up can then be calculated by,

$$\delta^{'}_{p,i} = 1 - \delta_{p,i} = 1 - \frac{\beta_i \epsilon_i}{\beta_i \alpha_i}$$

Given the prior example, we can then calculate this out,

$$\delta_{p,1} = \frac{20 * 10}{20 * 180} = 0.0555...556 = 5.56\%$$
$$\delta_{p,2} = \frac{10 * 15}{10 * 120} = 0.125 = 12.5\%$$
$$\delta_{p,3} = \frac{30 * 5}{30 * 300} = 0.01666...667 = 16.67\%$$

So we can say, "Task 1 will take 5.56% of the original time, task 2 only 12.5% and task 3 16.67%", and as for how much faster things are,

.

$$\delta_{p,1}^{'} = 1 - \delta_{p,1} = 1 - 0.0555...556 = 0.9444...444 = 94.44\%$$
$$\delta_{p,2}^{'} = 1 - \delta_{p,2} = 1 - 0.125 = 0.875 = 87.5\%$$
$$\delta_{p,3}^{'} = 1 - \delta_{p,3} = 1 - 0.01666...667 = 0.98333...3333 = 98.33\%$$

We can say, “Task 1 will be 94.44% faster, task 2 87.5% faster and task 3 98.33% faster.” But, what if we want to answer the question, “How much faster overall is our process given these three optimizations together?”

We will denote the net percentage changes as $\delta_{p,\Sigma}$ and can use the following formula to calculate it,

$$\delta_{p,\Sigma} = \frac{\delta_t}{\zeta} = \frac{\zeta - \hat{\Delta}_\Sigma}{\zeta}$$

Which expands out to,

$$\delta_{p,\Sigma} = \frac{\zeta - |\sum_i^n \Delta_i|}{\zeta} = \frac{\zeta - |\sum_i^n \beta_i(\epsilon_i - \alpha_i)|}{\zeta}$$

And if we want to calculate the increased in speed, denoted by $\delta_{p,\Sigma}^{'}$ we can use the formula,

$$\delta_{p,\Sigma} = 1 - \delta_{p,\Sigma}$$

In the prior example we know that $\hat{\Delta}_\Sigma = 13300s$ so we can calculate out the percentage easily,

$$\delta_{p,\Sigma} = \frac{72000s - 13300s}{72000s} = \frac{58700s}{72000s} = 0.8152777...778 = 81.53\%$$
$$\delta_{p,\Sigma}^{'} = 1 - 0.8152777...778 = 0.1847222...222 = 18.47\%$$

So we can say, “With optimizations we can do the same work in 81.53% of the time, and overall we’re now working 18.47% faster.” As you can see, with these three optimizations alone we’re able to reduce a developer’s time spent by almost 20%, or a fifth per week. If we were to extrapolate this from a 20 hour work week, to a 40 hour one and project over 52-weeks (1 year) we would save a total of 384 hours 13 minutes and roughly 20 seconds. That’s over two months of full-time work for a single-developer.

Conclusion

In review,

1. You can now calculate the net improvement in time and as a percentage of multiple tasks.
2. You can also measure how much your process has improved overall.
3. You should use these to analyze how much your programmer practices overall can be improved, by creating estimates for each step.
4. Using the calculations of time-to-benefit from the original article, you should be able to vastly improve your programming through-put.
5. Remember to project over a team, if you are working with multiple people.

Hopefully this has been helpful to you. Thank you for your time and have a nice day.