Engineering Velocity:

Cultivating a sustainable high output team.

DALMO CIRNE
dalmo.cirne@gmail.com

Jul 2019

Abstract

This paper takes a data aware approach to analyzing the impact of interruptions and incidents on the productivity of the engineering team, examines the components of a task and the importance of a minimum layer of processes, and concludes with the proposal of a framework to cultivating a sustainable high output team.

We all know that interruptions have an impact on productivity. Meetings, context switching, bugs from *tech debt*[1], etc. All of those affect, in a negative way, the production capacity of a team. Like everybody else, each of us have their many suspicions about how much or to what extent things are affected. However, those are just our opinions, our gut feeling. Instead, could we measure, quantify, and model the data to have a deeper insight?

This is the motivation behind this paper: A study quantifying the impact of the various interruptions, incidents, context switching, meetings, and other interferences that cause a decay in the productivity of the teams, and a look at what can be done to mitigate the effects of such incidents.

Between Nov 1, 2018, and Jan 30, 2019, I observed several incidents compromising the productivity of engineers, most of which related to service failures in the production systems due to tech debt accumulated over the years and left unaddressed.

There were 270 incidents observed over a period of 91 days; 58 of these (63.74%) experienced incidents, while 33 days (36.26%) did not. During this period there were several different kinds of alerts triggered. They have been compiled in Table 1. (The incident names have been changed to generic names following the pattern: *Incident XX*.)

I ask the reader to keep an open mind to the number of, and different categories of, incidents that can impact the team's productivity. The quantified data presented here comes from production system alerts. However, one can easily imagine that other categories (excessive number of meetings, scope creep[2], constant context switching, etc) can be equally disrupting, and when combined, may have a catastrophic compounding effect.

Table 1: *Incident stats.*

	Number of incidents	Average number of incidents per day ($\overline{I_d}$)	One incident every t hours (I_t)
Incident 1	55	0.6044	39.7091
Incident 2	48	0.5275	45.5000
Incident 3	42	0.4615	52.0000
Incident 4	42	0.4615	52.0000
Incident 5	27	0.2967	80.8889
Incident 6	13	0.1429	168.0000
Incident 7	12	0.1319	182.0000
Incident 8	12	0.1319	182.0000
Incident 9	7	0.0769	312.0000
Incident 10	6	0.0659	364.0000
Incident 11	2	0.0220	1,092.0000
Incident 12	2	0.0220	1,092.0000
Incident 13	1	0.0110	2,184.0000
Incident 14	1	0.0110	2,184.0000
Total	**270**	**2.9670**	**8.0889**

$$I_t = 24 \times (\overline{I_d})^{-1} \tag{1}$$

[1] `https://en.wikipedia.org/wiki/Technical_debt`
[2] `https://en.wikipedia.org/wiki/Scope_creep`

Modeling the data

The data was[3] collected and compiled in tabular format for computation and analysis. However, for the sake of brevity, it has been omitted from this document.

One of the properties of the data points is that the occurrence of an incident is completely independent; that is, it is in no way, shape, or form linked to a previous or future incident. This is similar to the number of people who go to the post office at any given moment: There will be moments when no one is in line, moments when the line may be very long, and moments for every other state in between.

A Poisson Distribution or a Poisson Process is a discrete probability distribution perfectly suited to work on this kind of data, and models the probability of observing a certain number (k) of incidents on a given day.

Table 2 shows the expected number of incidents to be observed per day, based on the historical data observed during the period of the study.

Table 2: *Expected number of incidents per day.*

	1	**2**	**3**	**4**	**5**
Incident 1	0.6044	1.2088	2.4176	4.2308	18.1319
Incident 2	0.5275	1.0549	2.1099	3.6923	15.8242
Incident 3	0.4615	0.9231	1.8462	3.2308	13.8462
Incident 4	0.4615	0.9231	1.8462	3.2308	13.8462
Incident 5	0.2967	0.5934	1.1868	2.0769	8.9011
Incident 6	0.1429	0.2857	0.5714	1.0000	4.2857
Incident 7	0.1319	0.2637	0.5275	0.9231	3.9560
Incident 8	0.1319	0.2637	0.5275	0.9231	3.9560
Incident 9	0.0769	0.1538	0.3077	0.5385	2.3077
Incident 10	0.0659	0.1319	0.2637	0.4615	1.9780
Incident 11	0.0220	0.0440	0.0879	0.1538	0.6593
Incident 12	0.0220	0.0440	0.0879	0.1538	0.6593
Incident 13	0.0110	0.0220	0.0440	0.0769	0.3297
Incident 14	0.0110	0.0220	0.0440	0.0769	0.3297
Total	**2.9670**	**5.9341**	**11.8681**	**20.7692**	**89.0110**

The column headers in table 2 mean: *number of days* (d). Each cell represents the expected number of alerts to be triggered given the number of days listed on the column header.

Each cell is computed using equation 2

$$E_d = 24 \times I_t^{-1} \times d \tag{2}$$

The probability of observing an incident, given a length of time, can be computed by utilizing the expected number of observed incidents. The following equations describe the computation of probabilities presented in Table 3.

$$P(k \text{ incidents per time interval}) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{3}$$

Where lambda (λ) is given by equation 4.

$$\lambda = \frac{24}{d} \times \overline{I_d} \tag{4}$$

In Table 3 we see the probability of seeing an incident occurring, given a certain number of days (column header). Each row shows the probability of the incident being of a particular category, and the last row (Total) shows the probability of an incident, from any of the categories, occurring.

[3] "@holizz Singular data annoys the same people that find split infinitives objectionable - pedants with no understanding of linguistics." – `https://www.theguardian.com/news/datablog/2010/jul/16/data-plural-singular`

Table 3: *Probability of observing k incidents in 1 day.*

	1	**2**	**4**	**6**	**8**	**10**
Incident 1	3.302e−1	9.980e−2	3.038e−3	3.699e−5	2.413e−7	9.794e−10
Incident 2	3.113e−1	8.209e−2	1.903e−3	1.765e−5	8.770e−8	2.711e−10
Incident 3	2.909e−1	6.713e−2	1.192e−3	8.462e−6	3.219e−8	7.619e−11
Incident 4	2.909e−1	6.713e−2	1.192e−3	8.462e−6	3.219e−8	7.619e−11
Incident 5	2.205e−1	3.272e−2	2.400e−4	7.043e−7	1.107e−9	1.083e−12
Incident 6	1.238e−1	8.846e−3	1.504e−5	1.023e−8	3.730e−12	8.457e−16
Incident 7	1.156e−1	7.620e−3	1.104e−5	6.401e−9	1.988e−12	3.840e−16
Incident 8	1.156e−1	7.620e−3	1.104e−5	6.401e−9	1.988e−12	3.840e−16
Incident 9	7.123e−2	2.740e−3	1.351e−6	2.664e−10	2.815e−14	1.851e−18
Incident 10	6.173e−2	2.035e−3	7.372e−7	1.068e−10	8.293e−15	4.006e−19
Incident 11	2.150e−2	2.363e−4	9.510e−9	1.531e−13	1.321e−18	7.089e−24
Incident 12	2.150e−2	2.363e−4	9.510e−9	1.531e−13	1.321e−18	7.089e−24
Incident 13	1.087e−2	5.972e−5	6.010e−10	2.419e−15	5.216e−21	6.999e−27
Incident 14	1.087e−2	5.972e−5	6.010e−10	2.419e−15	5.216e−21	6.999e−27
Total	**1.527e−1**	**2.265e−1**	**1.662e−1**	**4.876e−2**	**7.665e−3**	**7.497e−4**

We can see the **Total** row plotted in figure 1. As expected, the distribution is heavily skewed to the left.

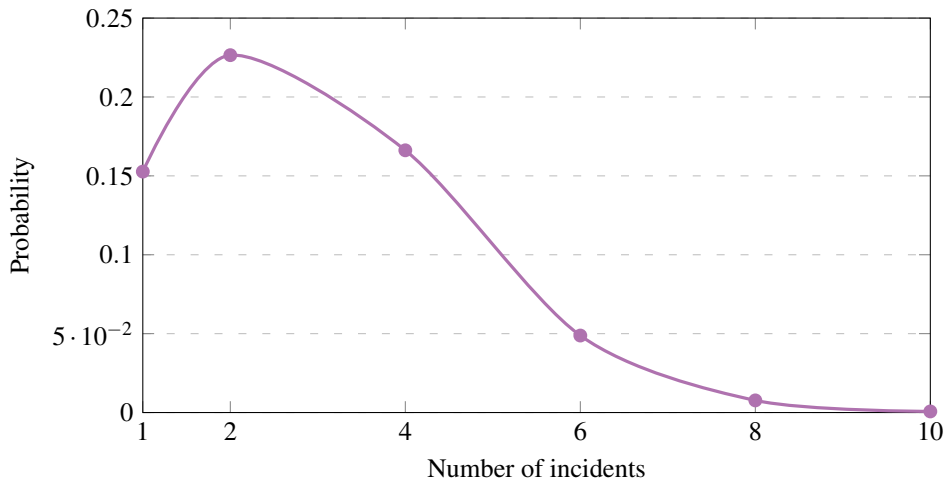

Figure 1: *Probability of observing k incidents in 1 day.*

In addition to computing the probability of an incident occurring, given a certain number of days. The probability of waiting a certain amount of time (fractions of a day), until an incident occurred was also computed. The results were computed using equation 5, and are shown in table 4.

$$P(\text{wait} < t) = e^{-(\overline{I_d})t} \tag{5}$$

In equation 5, t is the variable representing a faction of a day, shown in the header of table 4.

Table 4: *Probability of waiting more than t days for an incident.*

	0.25	**0.50**	**0.75**	**1.00**	**1.25**	**1.50**	**1.75**	**2.00**
Incident 1	0.8598	0.7392	0.6355	0.5464	0.4698	0.4039	0.3473	0.2986
Incident 2	0.8765	0.7682	0.6733	0.5901	0.5172	0.4533	0.3973	0.3482
Incident 3	0.8910	0.7939	0.7074	0.6303	0.5616	0.5004	0.4459	0.3973
Incident 4	0.8910	0.7939	0.7074	0.6303	0.5616	0.5004	0.4459	0.3973
Incident 5	0.9285	0.8621	0.8005	0.7433	0.6901	0.6408	0.5950	0.5524
Incident 6	0.9649	0.9311	0.8984	0.8669	0.8365	0.8071	0.7788	0.7515
Incident 7	0.9676	0.9362	0.9058	0.8765	0.8480	0.8205	0.7939	0.7682
Incident 8	0.9676	0.9362	0.9058	0.8765	0.8480	0.8205	0.7939	0.7682
Incident 9	0.9810	0.9623	0.9439	0.9260	0.9083	0.8910	0.8741	0.8574
Incident 10	0.9837	0.9676	0.9518	0.9362	0.9209	0.9058	0.8910	0.8765
Incident 11	0.9945	0.9891	0.9837	0.9783	0.9729	0.9676	0.9623	0.9570
Incident 12	0.9945	0.9891	0.9837	0.9783	0.9729	0.9676	0.9623	0.9570
Incident 13	0.9973	0.9945	0.9918	0.9891	0.9864	0.9837	0.9810	0.9783
Incident 14	0.9973	0.9945	0.9918	0.9891	0.9864	0.9837	0.9810	0.9783
Total	**0.4763**	**0.2268**	**0.1080**	**0.0515**	**0.0245**	**0.0117**	**0.0056**	**0.0026**

We can see the **Total** row plotted in figure 2.

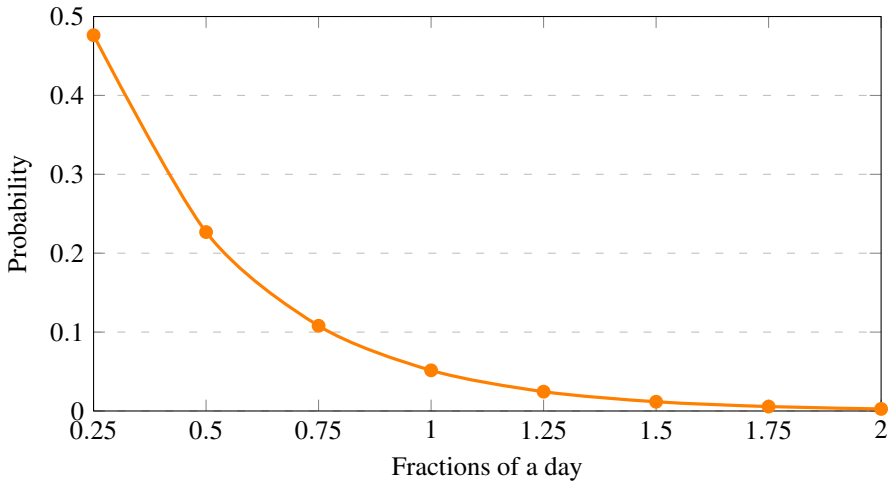

Figure 2: *Probability of waiting more than t days for an incident.*

In this case, as we can see in figure 2, there is a good chance that at least one engineer will have to put out a fire (metaphorically speaking) in the production system before finishing that first cup of coffee. (Whist contemplating whether they'd be better off if instead they were forced to listen to Vogon poetry[1].)

The impact of interferences on productivity

During the observation period, I assumed that a large portion of incidents were triggered due to tech debt embedded in the codebase or implementations that lacked a thorough process (unit and regression tests). Upon looking into several logs from the continuous integration system, many of these assumptions were validated.

Perhaps if more attention had been given to architectural design, algorithms, data structures, design patterns, and unit tests, these incidents could have been significantly reduced in frequency, and in many cases, prevented. (I will talk more about these in the next section.)

Next, I propose three scenarios to measure and analyze the impact that the incidents had on the production capacity of the team.

Imagine a standard Agile[2] 2-week sprint (or 10 working days). It contains epics, tasks, stand-ups, and all else that comes with it.

In table 5 you can see each of the different scenarios. In *Scenario 1* the numbers represent the available number of engineers in the team. Then, on each subsequent scenario, the parameters are relaxed to simulate the different degrees of impact on productivity from having more people, more time, and lower complexity incidents.

Table 5: *Impact stats*

	Scenario 1	Scenario 2	Scenario 3
Number of engineers (n)	10	12	16
Hours in a day (d)	8	8	8
Time, in hours, to solve an incident (h)	2	1	0.5
Expected number of incidents per day (i)	2.9670	2.9670	2.9670
Expected hours per day spent fixing incidents (s)	5.9341	2.9670	1.4835
Expected proportion of a day spent fixing incidents (w)	0.7418	0.3709	0.1854
Overall impact of incidents on the engineering team (u) (distributed evenly among all engineers)	0.0742	0.0309	0.0116

You can see in equations 6, 7, and 8, how the calculations were performed for table 5.

$$s = h \times i \tag{6}$$

$$w = \frac{s}{d} \tag{7}$$

$$u = \frac{w}{n} \tag{8}$$

After the start of a sprint, we want to measure the fractional decay in productivity caused by an incident. And over time we want to measure the cumulative impact on the productivity of the team as a whole (and to some extent, the whole company).

Let's introduce the concept of *Engineering Velocity* (V). In physics, velocity is measured as a change in distance over a period of time. In the context of engineering, velocity is the change in the development of a product over a period of time.

Equation 9 is the fractional change in engineering velocity. The constant of proportionality u is the factor by which the velocity decays with the passage of time under the interruption of incidents.

$$\frac{dV}{V} = -u\,dt \tag{9}$$

Equation 9 is also an example of a differential equation[3]. It relates the differential dV to V itself and to the differential dt of the elapsed time. Integrating both sides of the equation and developing the solution gives us:

$$\int \frac{1}{V} dV = \int -u\,dt \tag{10}$$

$$\ln\left(\frac{V(t)}{V_0}\right) = -ut \tag{11}$$

$$V(t) = V_0\, e^{-ut} \tag{12}$$

Where V_0 is the initial engineering velocity, which in the case developed in this paper is 1 (or 100% capacity to work). However, if you have an ongoing sprint history and points assigned to each task, then V_0 would be equal to the average of the sum of all the points scheduled for a sprint–as shown in equation 13. (It can't be the points complete per sprint, as those are already subject to impact of incidents.)

$$V_0 = \frac{1}{m}\sum_{1}^{m} S_i \tag{13}$$

Looking at equation 12 we can already see that the impact of incidents on productivity is severe. It causes productivity to decay at exponential speeds.

Table 6 shows the impact of the incidents in each of the scenarios. In scenario 1 we can see that by mid-sprint, the capacity of the engineering team is reduced by almost half.

Table 6: *Capacity to work on a sprint*

Day	Absence of incidents	Scenario 1	Scenario 2	Scenario 3
1	1.0000	1.0000	1.0000	1.0000
2	1.0000	0.9285	0.9696	0.9885
3	1.0000	0.8621	0.9401	0.9771
4	1.0000	0.8005	0.9114	0.9658
5	1.0000	0.7433	0.8837	0.9547
6	1.0000	0.6901	0.8568	0.9437
7	1.0000	0.6408	0.8307	0.9328
8	1.0000	0.5950	0.8055	0.9221
9	1.0000	0.5524	0.7809	0.9114
10	1.0000	0.5129	0.7572	0.9009

In figure 3 we can see each of the scenarios plotted in one graph. It is easy to notice that even though the parameters have been relaxed for scenarios 2 and 3, the impact on productivity is still severe.

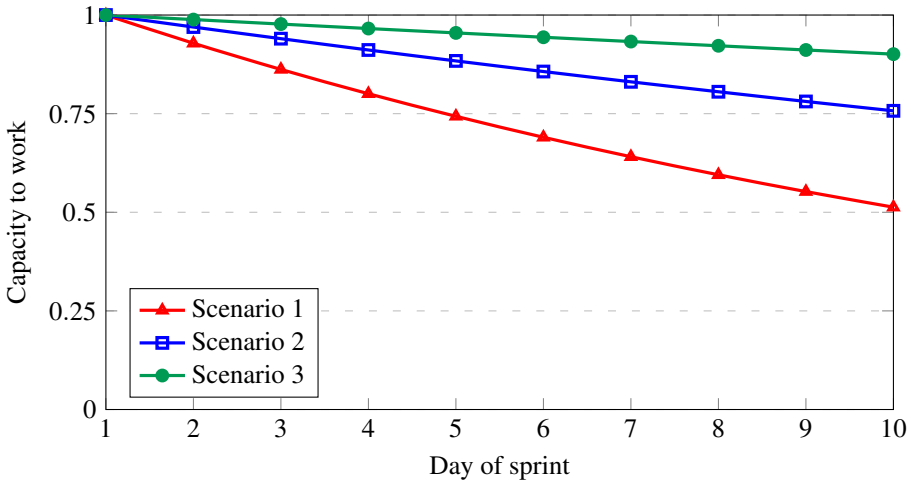

Figure 3: *Capacity to work on a sprint.*

Projects have more than one sprint. If we look at a larger window of time, meanwhile allowing for the impact of incidents to continue, soon enough the productivity of the engineering team will grind to a halt.

The dynamics of loss and restoration of productivity can be seen represented in figure 4. Assuming productivity at its peak at time points 1 and 2. We can imagine that is an incident occurs at point 2, losing all the velocity. Time between time points 3 and 4 is used addressing the incident.

When resuming work, productivity is not gained immediately, it takes some time for one to get back to the mental state prior to the incident and restore the same level of productivity. This *restoration period* is represented by the Sigmoid function plotted between time points 4 and 14.

For the sake of simplicity of the model, and to better explain and visualize the negative effects of interruptions in the velocity of the engineering team, this dynamics of loss and restoration of productivity, although acknowledged and demonstrated here, has been left out.

Including this dynamics to the model would probably require a separate paper by itself. Having said that, excluding it from the model presented here does not diminish neither the value nor the degree to which the impact of interruptions, incidents, meetings, tech debt, and more, affect productivity.

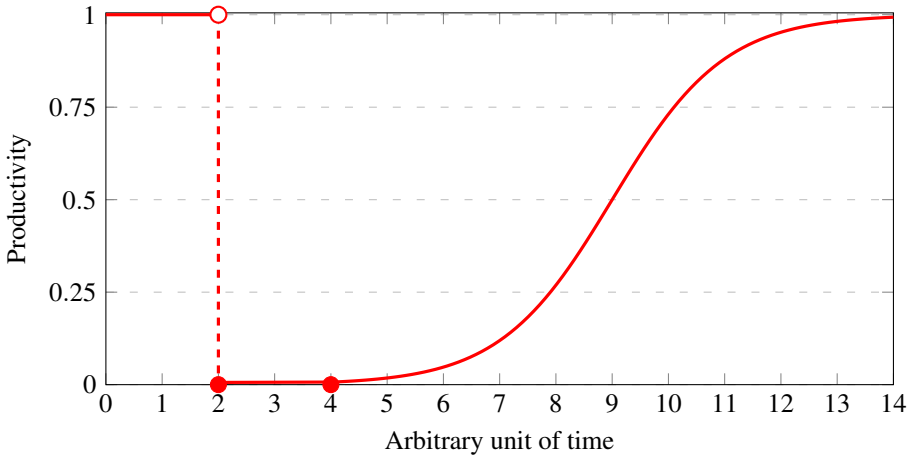

Figure 4: *Restoring productivity after incident.*

$$R(t) = \frac{1}{1+e^{-t+c}} \tag{14}$$

Equation 14 models the behavior of restoring productivity after an incident, where t represents time and c is a constant representing the time point at which productivity would be half-way restored.

Going back to the argument we were making about the long term sustainability of velocity and the capacity to work, productively, sprint after sprint. Table 7 shows 10 weeks, or 5 sprints, of the compounding effect of incidents on the capacity to get work done.

Table 7: *Capacity to work over multiple sprints*

Day	Absence of incidents	Scenario 1	Scenario 2	Scenario 3
1	1.0000	1.0000	1.0000	1.0000
6	1.0000	0.6901	0.8568	0.9437
11	1.0000	0.4763	0.7341	0.8906
16	1.0000	0.3287	0.6290	0.8404
21	1.0000	0.2268	0.5390	0.7931
26	1.0000	0.1565	0.4618	0.7485
31	1.0000	0.1080	0.3957	0.7063
36	1.0000	0.0746	0.3390	0.6665
41	1.0000	0.0515	0.2905	0.6290
46	1.0000	0.0355	0.2489	0.5936

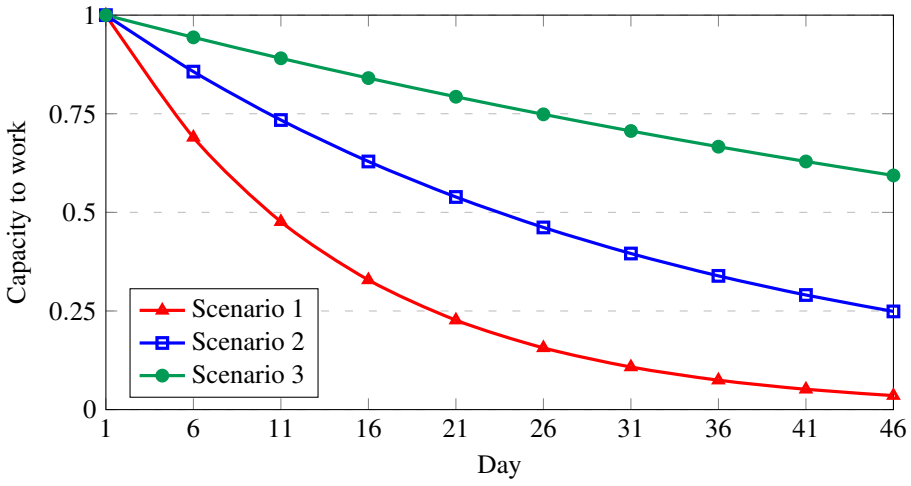

Figure 5: *Capacity to work over multiple sprints.*

Observation of the impact on productivity are not as severe in practice as predicted by the model and shown in table 7 and figure 5. Mercifully, in real life, incidents eventually get resolved, meetings come to an end, and product scope has to draw a line in the sand, and that time becomes productive time once again.

The data seen so far is from a certain category of interruption, but it is easy for us to realize that other categories of interruption would have similar impacts on productivity (e.g. excessive meetings, re-work of features).

Framework for a high output team

Creative people have a natural aversion to the act of planning. This is the kind of activity reserved for bureaucrats and grammarians of prescription labels. The impression that creatives get is that planning stops the flow of creativity from the mind; a ball and chain tethered to their ability to make progress.

However, planning is an essential instrument in our toolbox. It allows us to grasp an entire project in our heads. We can see the *big picture* and break it into its many parts, like pieces of a gigantic puzzle, then work on each of these bite-size activities, reassembling the puzzle step-by-step, as each of these tasks are complete.

Planning serves at least 2 purposes:

1. View of the whole, which also allows us to have a sense of progress.
2. Workable tasks that are feasible to work on within a reasonable amount of time. This not only gives us a sense of progress, but can also be a powerful motivator, reminding us of where we want to go.

In other words: planning helps us tame the beast of complexity. This is a fundamental condition in order to achieve and maintain the velocity that will make a difference in bringing products to market and avoiding burn outs: A sustainable velocity, if you will.

The act of planning, executing, and all other related activities, is not to hamper velocity. On the contrary, it is a necessary condition in order to move faster. No one wants keep working and re-working the same feature over and over again. Everyone wants to get things done and move on to the next chapter.

One of the biggest challenges in working on a project is to manage the expectations of the different teams involved: Engineering, Project Management, Marketing, Sales, Legal, Finance, HR, and more. Each have potentially different deliverables, scopes, and deadlines. Divergence in expectations may inevitably lead to attrition among the teams, to blaming, and to finger pointing.

The actual speed of implementing and releasing a feature may be deceiving. Coding and shipping alone may not be a good metric, at least not one thorough or exhaustive, especially if engineers need to often go back to the same code to fix, refactor, or patch a bug. This additional work has a measurable impact on the productivity (as shown in the previous sections) of engineers and the other teams.

Planning gives a degree of transparency to the process and makes possible the management of expectations among different teams, as each team may look into the scopes and timelines of other teams, talk among themselves, and negotiate deliverables.

We all try to find ways to implement things faster. However, it is often the case that the mental shortcuts sound better in our heads, compared to saying them out loud. In the end we are all aiming for the same thing: the system should move faster, in a sustainable way, not just as a one-off feature. Thus, before we proceed, we need to ask ourselves two questions: *What is a task? And what does it mean to complete a task?*

It is often the case we find ourselves running on the treadmill of the effective production line, and think of tasks as just their core component, forgetting that there are peripheral activities related to them that sometimes are of equal importance to the core task itself. (You want the brakes to be tested after servicing the brake fluid and pads, right?)

Here are the fundamental components of a task[4]. After, and only after, these have been addressed, can we consider a task to be complete.

The six components of a task

1. **Architectural design**: Think before acting. Before rolling up your sleeves and starting to code, draw activity diagrams, use case diagrams, class diagrams, sketches, anything that will allow you to visualize all the moving parts and how they interact with the ecosystem.

Talk your design over with other engineers. They may have a different perspective on some of the aspects your approach, and a fresh set of eyes will help to strengthen your work.

This also is of great importance as the foundation of the documentation. You won't remember all the details in your head. Even if you did, when another person needs to work on the product, they will be able to become acquainted with the system, and productive, without requiring your assistance.

2. **Documentation**: Express and explain what the product, feature, or subsystem does. Sometimes documentation comes prior to implementation, other times implementation comes first. However, irrespective of which one comes first, documentation is an essential part of the process.

   Furthermore, documentation is not meant to be a *fire-and-forget* solution: it should be a living piece of work that evolves along with what it describes.

   Attributes of good documentation[4]:

   (a) *Requirements*: what is needed in order for the system to work?
   (b) *Setup of the environment*: how to configure the dials and knobs?
   (c) *Diagrams*: a picture is worth a thousand words
   (d) *Interaction*: how does it interact with other subsystems and clients?
   (e) *How-to*: steps to effectively integrate and use it

3. **Implementation**: Craftsmanship is the name of the game. Is it built to just work, or was it built properly? Are there design patterns[5] applied to it? Which algorithms[6] have been used? What data structures[7] is it using? Is the code decoupled? Can it be reused in other parts of the system?

   Avoid creating tech debt at almost all costs[5]. If you have to create tech debt, it must be discussed with and approved by your manager. The intent here is not to ask for permission, but to raise awareness.

   Priorities to consider when implementing (in this specific order of importance):

   (a) *Security*: the data should be safe
   (b) *Stability*: things should run when expected, and no one should need to wake up in the middle of the night to patch something
   (c) *Scalability*: does it work for 1 customer? How about 1,000? 1,000,000?
   (d) *Architecture and design*: if done properly, tech debt and refactoring should be rare and far in between
   (e) *Maintainability*: implementation should be modular, clear, decoupled, robust, tolerant in what it receives as parameters, strict in what it sends as response, and extensible[6]
   (f) *Runtime speed of the program*: yes, things have to run fast, but only after all the prior items have been satisfied

4. **Quality control**: Trust, but verify. Test everything. Furthermore, the tests should go beyond verifying the functionality. Implement tests with the intention of breaking things. Test for unlikely scenarios, edge cases, wrong data types, and whatever else comes to mind. Last, but not least, if a bug is reported, immediately write a test case to reproduce it and ensure it won't happen again in the future.

5. **Release process**: Establish a protocol for the release process, including a checklist with the key steps. Furthermore, is this the release of a feature? A bug fix? A new product?

   It is almost always the case that Product, Marketing, and other teams need to be involved. Bring them into the conversation at an early stage. Promotional material may need to be produced, training of the sales team, or even a notice of deprecation may be necessary to give to existing customers. The release process is how customers become aware of how you're adding value to their business and to their lives.

[4] The more of these items, the better. Not all are necessary in all cases. Use your best judgement.

[5] Tech debt can sometimes be an ally. On early stages of development of a new product or feature, it may be more important to have something functional than a system that is complete and is able to land the lunar module on the moon.

[6] Always have the future in mind, it will be here sooner than you think. Try to avoid having the feature come home to roost.

6. **Monitoring of execution**: Measure (almost) everything. Usability analytics contain valuable information about how the system is performing, being used, and whether it is producing the intended results.

   Last, but not least, have an alert system to notify you if things are not functioning as intended. It is important to be aware of failures and to take restorative actions as soon as possible.

Communication

Earlier we mentioned how excessive meetings can bring havoc to productivity. The other side of the coin is that without meetings there is no efficient communication and circumstances are equally derailed.

Scheduled and Ad Hoc meetings should happen. They are paramount to communication, planning, monitoring, and alignment of the team. Without these elements there is no path to success.

Having said that, there is a tendency to think that everything needs a meeting, that all things need to be discussed with everyone, *ad infinitum*. That is when the tables turn and productivity begin suffering. The challenge is to strike balance.

Here are my suggestion for categories and number of meetings (in no particular order):

- Sprint planning: Assuming a 2-week sprint cycle, there should be 1 meeting to both give closure to a previous sprint, and begin a new sprint. The first part should address dynamics of the previous sprint, and the second part should be about what comes next and kickoff.

  Sprint planning meetings should happen on the first business day of the week, or whenever it makes you happy. Just be consistent.

- Stand up: Following the kickoff of a sprint, there should be 2 weekly meetings to monitor progress, share roadblocks, and brief discussion in general. The meeting should happen on the first business day of the week, and the second to last business day of the week. On a regular week the meetings take place on Mondays and Thursdays.

  Keep in mind that stand up meetings are synchronous by nature. They happen either in person, or opportunistically on the internal communications app. In both cases people are either attending the same meeting (in person or via conference call) or are posting updates at about the same time. It is better for everyone to acknowledge that these are indeed synchronous in nature, to schedule as few as possible (but not fewer), and to get everyone to make their participation count.

  A final note on stand up meetings. Since every two weeks there will be an overlap between a sprint planning and a stand up meeting, any remaining conversation in the context of a stand up can be addressed prior to closing the active sprint.

- Architectural design: These are meetings where diversity of ideas are particularly welcome. They also allow for a broader understanding of the system as a whole. Whenever possible, invite people to participate and criticize ideas, constructively.

- Ad Hoc: Life happens, so do impromptu meetings. Although they have to happen, try to keep them to a minimum. If left unchecked they can cause a real dent in productivity. Only the people who need to be there should be invited to attend.

A newly assembled team or a team with several new members may need more frequent meetings than what is suggested here. However, progressive adjustments should be made to reduce the number and cadence of the meetings as the team dynamics mature.

Deadlines and tech debt

In the book: *Every Tool's a Hammer: Life Is What You Make It*[8], Adam Savage gives the best description I ever encountered to how deadlines can work in your favor. He says: *"They are the chain saw that prunes decision trees. They create limits, refine intention, and focus effort."*

Deadlines can be a powerful ally to set scope of a project, force you to focus on getting things done, and know when and why to make compromises in order to deliver a feature or a product.

Tech debt can also be an instrumental tool at our disposal. Imagine for a moment that you are building a road, and along the way you and the construction crew encounter a mountain. You may chose to surgically bore a tunnel through the mountain and continue the road, or you may chose to use explosives and blast it to smithereens. Just like as if handling explosives, handle tech debt with care.

One may argue that re-architecting a system, or refactoring it for a different scale, can be categorized as tech debt. I'd like to offer a different perspective, and a distinction between those. Tech debt is like making a balloon out of lead. At some point it may need to be rebuilt with a lighter canvas. Re-architecting or refactoring a system, however, would be more like driving a two-seater sports car when you're single, or you and your spouse don't have kids. That is an appropriate setup for the circumstances.

Imagine now that you have a family and kids. The two-seater sports car is no longer adequate. A sedan or an SUV may be a better fit–given that the constraints and scale of your *"system"* have changed. It is perfectly acceptable to architect and build systems compatible with the constraints and requirements of each of the growth stages, that is not tech debt.

If and when rethinking parts of system is needed, due to a change of architecture or scale, that is alright, too. We will cross that bridge when we get there.

The importance of sharing knowledge

"Jack of all trades, master of none." This figure of speech refers to a person who has experimented with many skills, but has not focused nor gained expertise in any of them. Often although this is true, it doesn't mean we should not share what we know with our and/or other teams.

It serves multiple purposes: awareness of products and features, review of concepts you may already know, learning new technologies or processes, and expanding one's horizon in general.

Learning helps keep everyone's skills sharp. Try starting and maintaining an ongoing study group in your team or company. Both presenter and attendees will grow a great deal in each of the sessions.

Conclusion

The conclusion of this study, and I hope that by now you share the same or a similar opinion, is that to be truly fast in delivering products, one needs to achieve and maintain a sustainable engineering velocity. It comes with planning and processes, but those are kept to a minimum overhead, and benefit the entire group.

The absence of such processes defining expectations end up imposing a semi-invisible cost (at least not immediately and directly visible) on productivity.

Among the imposed costs is the reduction in the production capacity of the engineering team. Such reduction in capacity has a direct impact on the deliverables of a sprint and may lead to claims of teams being under staffed, where staffing is at an appropriate level to meet the required output.

We could clearly see that even if more engineers were added to the team, given enough time, the impact on productivity would converge to the same low levels.

Investing in establishing a working framework as suggested here may be the best and cheapest way of improving a long term sustainable high output team.

Building things with craftsmanship and a thorough process may be our most powerful and efficient instruments to keep a sustainable velocity, and an engaged team. If circumstances asks from us to make sub-optimal choices, at least now we are better prepared to deal with its consequences.

As the old proverb goes: *"Measure twice, cut once."* Careful first steps may avoid a lot of additional work later.

References

[1] D. Adams, *The Hitchhiker's Guide to the Galaxy*. Harmony, 1979.

[2] A. Stellman and J. Greene, *Learning Agile: Understanding Scrum, XP, Lean, and Kanban*. O'Reilly Media, 2013.

[3] W. E. Boyce and R. C. DiPrima, *Elementary Differential Equations*, 7th ed. John Wiley and Sons, 2001.

[4] D. Cirne. (2019, Jan) Engineering culture manifesto. [Online]. Available: https://dcirne.github.io/engineering_culture_manifesto.html

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[6] J. Erickson. (2018) Algorithms. [Online]. Available: http://jeffe.cs.illinois.edu/teaching/algorithms/

[7] P. Morin. Open data structures: An introduction (open paths to enriched learning). [Online]. Available: http://opendatastructures.org

[8] A. Savage, *Every Tool's a Hammer: Life Is What You Make It*. Atria Books, 2019.

[9] R. V. Hogg and E. A. Tanis, *Probability and Statistical Inference*, 7th ed. Prentice Hall, 2005.

[10] N. N. Taleb, *The Black Swan: The Impact of the Highly Improbable*. Nassim Nicholas Taleb, 2010.

[11] W. E. Deming, *The Essential Deming: Leadership Principles from the Father of Quality*. McGraw-Hill Education, 2012.

[12] E. S. Raymond, *The Art of UNIX Programming*. Addison-Wesley, 2003.

A black swan

On Nov 21, 2018 there were 30 incidents. The probability of having that many incidents in one day is: 0.00000000000000000286%, in other words, it should never happen; yet, it did. A structured process and quality control could perhaps have prevented this *"Fat Tail"*[10] day.

Other outliers were: Nov 7, 2018 (10 incidents), Nov 29, 2018 (19 incidents), Dec 11, 2018 (13 incidents), Jan 8, 2019 (16 incidents), and Jan 15, 2018 (17 incidents).