Part 1: The Basics of Production
Chapters 1 to 2

Vladimir Feinberg

February 24, 2019

Notes on Chapters 1 to 2 of Andy Grove's *High Output Management* [1].
Note that the sections are structured according to topics, not the chapters themselves

1 Creating Production Requirements

Work backwards from what the market needs. What do customers desire?

1. Timely delivery
2. High quality
3. Low cost

Then the question is, how do you trade off each of the above? At Intel, the trade-off seemed to be

$$\begin{aligned} \min\ & \texttt{cost} \\ \text{s.t. } & \texttt{quality} \geq \text{some acceptable threshold} \\ & \texttt{delivery time} \leq \text{sufficiently timely} \end{aligned}$$

But how should you set up the optimization problem in the context of your business? Andy does not answer this question for you. This is usually implied by your business model. For instance, for Amazon prime, delivery time is much more important, so Bezos might be interested in minimizing delivery time while keeping costs smaller than competitor's and quality at least as good as theirs.

Thus, the question of determining this aspect of your business model could in part be answered by considering the following.

1. Profile your customer to see what they care about more. For instance, in some industries, like e-commerce, cost is actually not as important as fast delivery turnaround (maybe, just guessing).
2. It just comes from the industry, market, and your business model or unique approach. This is what makes your business special: it's your business model

2 Production planning

The key insight is to *consider planning tasks with critical-path planning in the workflow DAG*. Work backwards from delivery time and requirements to construct the DAG of tasks to perform, and track the critical path progress over time.

Dont forget to incorporate non-productive tasks into your planning.

1. Assembly
2. Testing
3. Inspection
4. Queue wait time in components that have limited capacity

The above planning informs inventory purchases. For example, in a development environment, if your Travis Continuous Integration only handles 1 job at a time then you cant merge in more than one PR every 30 minutes or however long your build time is. In turn, you should never bottleneck yourself on CI by buying more concurrent builds as you get more developers.

3 Indicators

Indicators enable appropriate and accurate forecasting for decision making, such as:

1. Sales forecasts
2. Manpower quantity and quality
3. Equipment status and output rate
4. Raw materials

Taken on their own, indicators can result in a negative feedback loop. Paired indicators resolve Goodharts law-like bias from metrics.

- For instance, suppose youre measuring custodial staff by square feet cleaned alone. This may cause rooms to be poorly cleaned. By adding a complementary indicator, like the number of reports of unclean rooms, you can properly account for quality.
- Analogously, if you only record the number of shortages, then you may over-supply your inventory to avoid shortages. But if you also record end-of-day remaining inventory then you know not to over-invest.
- In software, consider using both the number of PRs (“diffs” or “branches”) merged as well as per-PR SLOC as a measure of developer productivity (neither is perfectly accurate, but both should be considered).

Provide variance estimates on measurements. Not every number should be taken under equal consideration. With variance estimates, you can add slack to scale with desired confidence in allocations. Critical processes may need more slack.

In trend prediction,[1] make sure that you use:

- Historical data
- Capture seasonality
- Autoregressive trends
- Nonlinearity (make sure you know what kind you need to have in your models)

Properly taking advantage of trends can lead to a lot of value creation (see the Bridgewater example).

4 De-risking Parallel Flows

If you have two dependent flows that are both deep and need to converge to create value, then you're undertaking a lot of risk. For instance, consider some manufacturing process that transforms raw material into goods and an analogous sales process that are both required to start early (for long sales cycles) that result into a successful deal (Fig. 1).

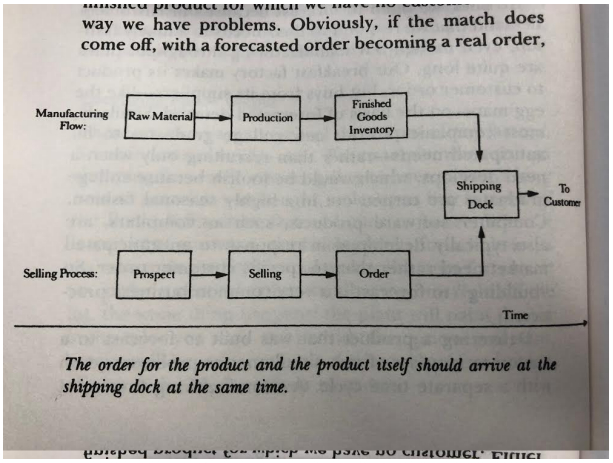

Figure 1

Fail early. Capture failures at the lowest value stage (earlier) that is possible to detect them (needs to be traded off with detection costs). For instance, in candidate recruitment: assess candidate interest early in the pipeline (i.e., during initial phone screens), and if theyre just shopping around then theyre not worth the interview time investment.

Introduce synchronization points as much as possible between parallel flows. Two parallel but dependent features that two teams are working should be incrementalized and should be interacting at as early stages

[1] This section is pretty weird, kind of like Andy just teaching you forecasting.

as possible (rather than building two things separately then connect). An example in code development would be to rebase often.

Program slack in volumes of output so that failures downstream can be compensated for. Deadlines for a feature from, say, a customer request, should be made with slack.

Introduce additional quality assurance to take into account the additional risk that the parallel flows incur whenever cheap, continuous tests are amenable (need to trade off w/ quality assurance risk).

5 Quality Assurance

5.1 Gated Tests

Gated tests, which hold-the-line until the completely check out a batch of a product, or take a while to decisively complete, but dont allow further changes to be made. In software:

- Diff review
- Pen test
- CI is a very cheap gated test

5.2 Sampling-based Tests

Sampling-based tests probabilistically evaluate for potential failures by taking a look at only parts of the actual total amount of product youre producing. If they fail then they indicate theres a broader problem

- Random selection diff re-review
- Fuzzy testing and nightly large builds or end-to-end tests, can be coupled with bisection to get gated-level test quality (bisection on commits, for instance).

6 Work simplification

If you list out the steps for production with fine detail, you can optimize away frequent unnecessary steps.

1. List out the steps rigorously
2. Set a goal for step reduction
3. Call each step into question

References

[1] Andrew S. Grove. *High Output Management*. Vintage, 1983.