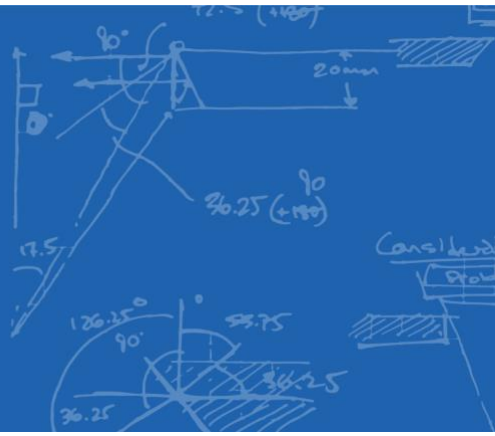
WHITE PAPER | AUGUST 2020

What the Past Teaches us about Optimal Product Development



The Study of Goal Definitions and Managing Hierarchical Systems Trade-offs.

Simplexity® PRODUCT DEVELOPMENT

The Market Leader in Mechatronics and Detailed Design Service | simplexitypd.com

INTRODUCTION

Development of products that involve firmware, electrical and mechanical engineering can get complicated. There can often be seemingly conflicting priorities related to hierarchical product requirements, cost, or development timeframes. This complexity is nothing new. In this paper, we will break down or decompose how to manage the tradeoffs between these systems, but first, we will review this process through the lens of the development process of the DEC-PDP-1 personal computer (circa 1960) as a history lesson in optimal product development.

THE FIRST PERSONAL COMPUTER

In 1960, Digital Equipment Corporation (DEC) released the PDP-1 personal computer. When the machine was released, it was a stellar achievement of cost / performance optimization. This machine was quite small compared with the "mainframes" of the day, behemoths that occupied entire rooms and were serviced by a "priesthood" of operators who fed in punched cards, mounted magnetic tape reels, and tore off reams of paper that came from the printers.

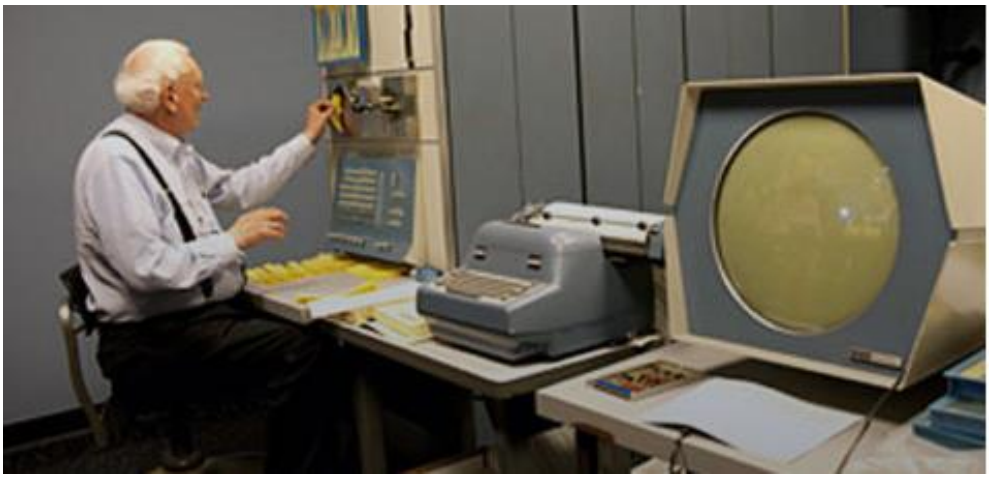
The PDP-1 by contrast was meant to be a "personal computer," operated by technical folks directly interacting with its front panel and other peripherals. Many consider this machine the first personal computer, as it needed no special room, no special air conditioning, no special power wiring, and no specially trained operators.

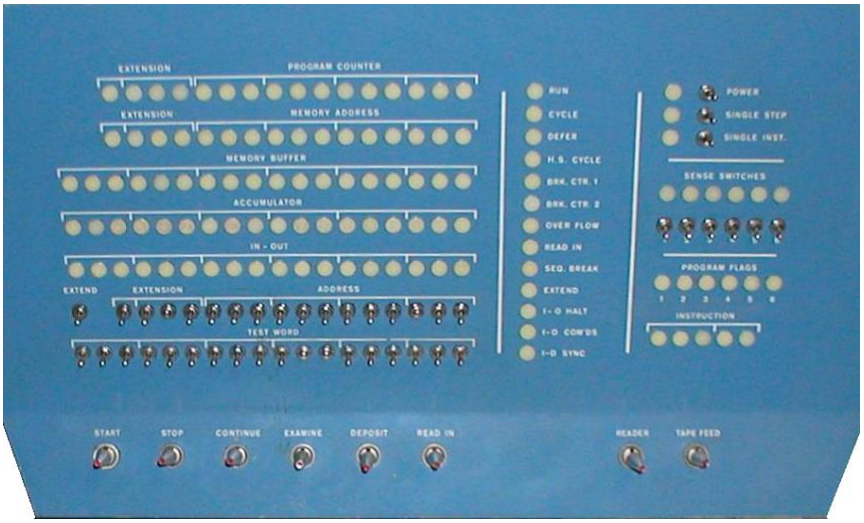
The PDP-1 could add two 18-bit numbers in 5 microseconds. And it had a (core) memory of 4K 18-bit words. It sold for $120,000 (a little over $1M today), which, at the time, was an incredible value.

In order for the designers to achieve this, constrained as they were by cost, size, power, time-to-market, and the requirements of the mechanical, electrical, and the software components, they had to be strictly disciplined to adhere to their design goals and also have an understanding of trade-offs between and among these competing needs. In studying this quintessential first computer, we can learn how breakthrough products are developed.

Many of the same basic truths will apply to your product.

Pictured Left to Right: *The PDP-1 Computer bays, with console display, paper tape reader and punch; and the Type 30 graphics display alongside the console typewriter. (Source: Digital Equipment Corporation)* ***Below:*** *Product Development pioneer Software Wizard Peter Samson at the console, loading paper tape. (Source: Mike Cheponis]*





Pictured: *The front (control) panel of the PDP-1. (Source: Image courtesy of https://www.vintagecomputer.net)*

WHAT?... BY WHEN?... FOR HOW MUCH?

When we examine the PDP-1, or any historically successful product, there are some common lessons we learn:

Every switch, every light, every panel of metal, every color, everything(!) was decided by people with the goals in mind.

The overall goal of the PDP-1 was this:

To produce a personal computer that could be used in scientific, laboratory, and communications sectors, which will take less than 1 year to develop and will sell for $120,000 or less.

In this exact sense, the PDP-1 was the "best that could be done" given the year (1959), the goal statement, and whatever sub-goals may have existed. The machine would go on to become a commercial success, selling 53 units over its lifespan – which provided DEC the resources to produce follow-on machines that eventually lead to Digital Equipment Corporation, at its height, becoming the #2 computer manufacturer in the world (with IBM being #1).

From today's point of view, we can ask: Why didn't they have more memory available? Why wasn't the machine faster? Why were the programming tools rather primitive? Why wasn't it cheaper? To so many other questions like these, the answer is the same: Because it couldn't be, given the goal.

That means that the goal was well specified, and the resultant product met the goal. This is the most important "ingredient" to a successful product: Knowing what it will be, within the constraints. A clear statement of the goal is the first piece of information that you will need on your path to product development success.

It is difficult to over-emphasize the need to clearly, and ideally succinctly, describe your product goal. And there are more questions that must be answered:

1) What do you want? (goal)

2) By when must it be done? (schedule)

3) How much will it cost? (finance)

Once we understand the primary objective, the secondary objectives become functions of well-established product development conventions. That is, we can choose to focus on one (or possibly two) of the "what," the "by when," and the "how much."

WHAT?

Clearly, the "what" is often a crucial consideration. You'll need to answer the question, "How will I measure that my product has succeeded?" Will you have sales of 10,000 units per month? Will your robot do a particular dull, dirty, and dangerous job well enough to pay for itself in 12 months? Have you limited your product's complexity so that it can be completed quickly enough to have market impact? Do you only need to build 3 to show to potential investors? The more that we both understand about the "what" and how success will be measured, the more we can all bring to bear in realizing the success goal.

BY WHEN?

At times, the "by when" for the product is paramount. If you are launching a robot to Mars, you have a hard deadline to get a feature functional, or it's not going to make it on that planned launch. And that may or may not be a mission scrub. If schedule is the #1 constraint, the other two components ("what" and "how much") suffer. Yes, to some extent, more cash can get pieces of a project to accelerate, but infinite money cannot produce a successful 1-day product development. There are natural limits to how fast a product can be delivered. If we take examples from wartime production during World War 2, we learned that near-impossible and vast projects can be successful – but they still need time.

FOR HOW MUCH?

So if "by when" is the main concern, "how much" can help, but the most common way to achieve the schedule goal is by reducing the project's scope, the "what." In the toy industry, reducing a toy's scope is done all the time, as not-able-to-be-implemented-in-time features get sacrificed to next year's product cycle.

What if you have a product idea, and only have a fixed budget to develop it? This, too, is possible when we are willing to modify the "by when" or "what" parts. To be sure there is a viable product, some schedule relaxation is often helpful. This allows different ways to accomplish the goal to emerge, to use lower-cost materials with longer lead times, use of common parts, semi-custom parts, and more strategies. But the "what" again becomes ultra-important. If the "what" can be fully specified such that there are no changes during development, then yes, fixed price will work. Fixed goal. Fixed price. Flexible time ("by when").

DELIVERING ON THE "WHAT"

In the case of the PDP-1, the engineers had a stated goal: *to produce a personal computer that could be used in scientific, laboratory, and communications sectors, and will sell for $120,000 or less*. They also had a schedule: *less than 1 year to develop*. In theory, they could not control the development costs that might have been necessary to achieve their "what" and "by when." Fortunately for DEC, they had a genius engineer named Ben Gurley heading the project. Ben and DEC management thought about how to reduce schedule. They decided to use a secret weapon, which was to make a computer from parts that DEC (and MIT) had mostly already designed. This, therefore, was an important sub-goal: use existing parts whenever possible. Internal designs were leveraged, magnetic tape drives purchased from other companies and fitted into the PDP-1.

The entire set of details of how to create the computer were up to Ben, and he understood that merely reusing old parts was not going to create a brand-new product category – something that is truly visionary. Recognizing that IBM's mainframes required specially trained personnel to operate, Ben instead wanted to make programming directly accessible to the scientific community. The best way to do this was to create a new computing paradigm, using novel peripherals. What should these peripherals look like? In these high-uncertainty situations, although at first it may seem to take longer and cost more overall, a smaller, intermediate product should probably be produced. Sometimes, this would be an actual stripped-down product; other times, there would be enough developed to answer important "what" questions.

The PDP-1 project did just that.

Pictured: *Mock-up of proposed PDP-1 product In this case, various pieces were physically brought together to help the systems engineers appreciate some of the human factors that would need to be considered in the final design. These types of exercises are useful in getting the "what" to be better understood by all team members[1]. (Source: Digital Equipment Corporation, 1959)*

[1] *I hasten to mention that the only piece of this PDP-1 mock-up that actually shipped with the real product was the chair. Yes, there was a console typewriter, display, light pen, cabinets, and more, but the real product was much more refined.*

INTERRELATED ENGINEERING DISCIPLINES: UNDERSTANDING THE TRADE-OFFS

Now that we have a general idea of how to affirm "what" the product will be, the next part is crucial:

Understanding that trade-offs must be made to establish the division of responsibility for performing system functions between the software/firmware, electrical, and mechanical components.

The task of making these important choices up-front usually falls on a systems engineer, with support from other engineers as needed. The output, the so-called product architecture, will be a roadmap for the entire development effort, for better or worse.

Frequently, engineers set goals of making products that push the boundaries of performance in several directions. Without a clear understanding of the trade-offs required, the product development effort can get bogged down, leading to disastrous results. Rather that pushing all the boundaries, note that there is often a market opportunity for a product that just focuses on being smaller or faster or easier to use. DEC struggled to compete with large computer systems dominated by IBM until it decided to create the smallest, cheapest, most user-friendly machines possible, starting with the PDP-1.

How then can we arrange mechanical, electrical, and firmware components to give us a reasonable chance to produce an elegant, simple product? We consider a hierarchy, with mechanical elements at the bottom, electrical next, and firmware at the top. The three rules of system decomposition guide us to push complexity upwards from mechanical pieces to firmware.

The First Rule of System Decomposition states that, if possible, push the complexity into the microprocessor's firmware. The cost of a microprocessor to do a task will only continue to decrease. The on-chip storage capability for firmware also continues to grow, and manipulating firmware is the easiest system way to modify or add functionality. In contrast, the costs of many mechanical parts, and certainly electro-mechanical assemblies (motors, solenoids) are already at rock-bottom prices, and unlikely to go much lower.

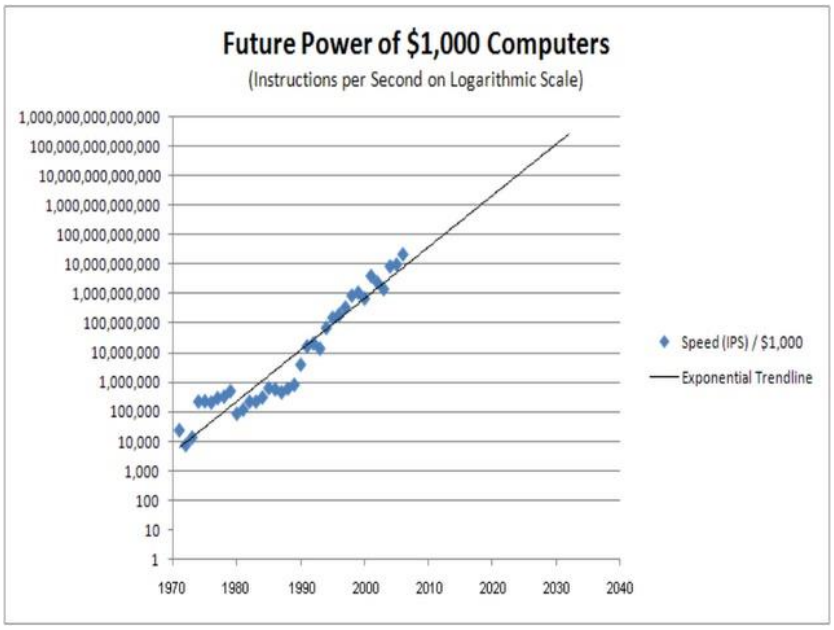
The Second Rule of System Decomposition states that, if possible, push functionality into electronics, once the First Rule is applied. Electronics (besides microprocessors) continue to decrease in cost. This is especially true of the analog-meets-digital A/D and D/A converters that are most often right on the microcontroller chip itself. Interfacing to the real world via analog voltages and currents has become fairly easy and inexpensive. Yes, high-speed and high-precision applications will still require more expensive components, but as a rule, any physical value or process that can be represented by a voltage or current can be easily handled by today's electronics.

Another basic building block that continues to increase in capacity and decrease in cost is memory. The PDP-1 had 9,216 bytes of memory[2]. Not Kbytes. Not Mbytes. Not Gbytes. A modern smart phone's computer has 4GB and it fits in your pocket with a battery that lasts all day. Today, a chip that has all of the PDP-1's logic and memory would be less than 0.1 mm x 0.1 mm in size.

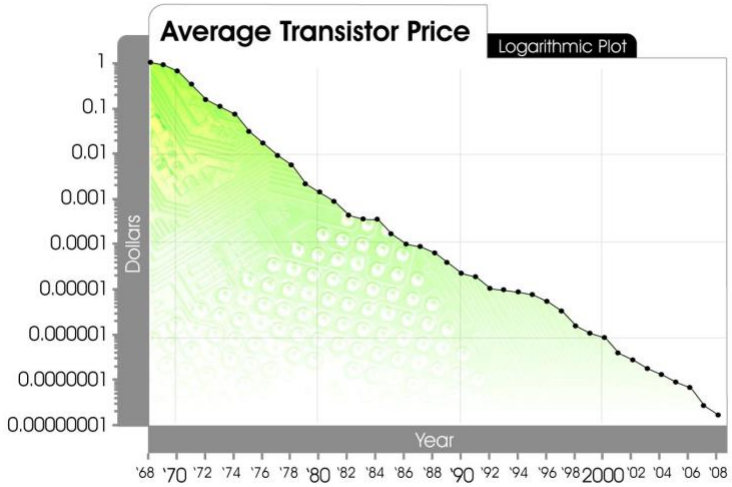
[2] This funny number is because the pdp-1 had a 4K-word memory of 18-bit words. The concept of 'byte' had not been invented! If we take the total number of bits 73,728 and divide by 8 bits per byte, we get 9,216 bytes.

There has never been such a rapid increase in performance and simultaneous decrease in cost of anything else besides computation. Yes, it is Moore's Law but also better architectures and better software.

This graph shows what a constant $1,000 will buy in computation. If it had gone back to 1960, the pdp-1 ran at 200,000 instructions per second and cost $120,000 (1960 dollars). The consequence of this trend is (and this is very important): **The cost of computation for a fixed task is approaching $0.**



Lest it seem like this is hyperbole, last year Jay Carlson wrote about the 3-cent microcontroller: *(Source: https://jaycarlson.net/2019/09/06/whats-up-with-these-3-cent-microcontrollers/.)*



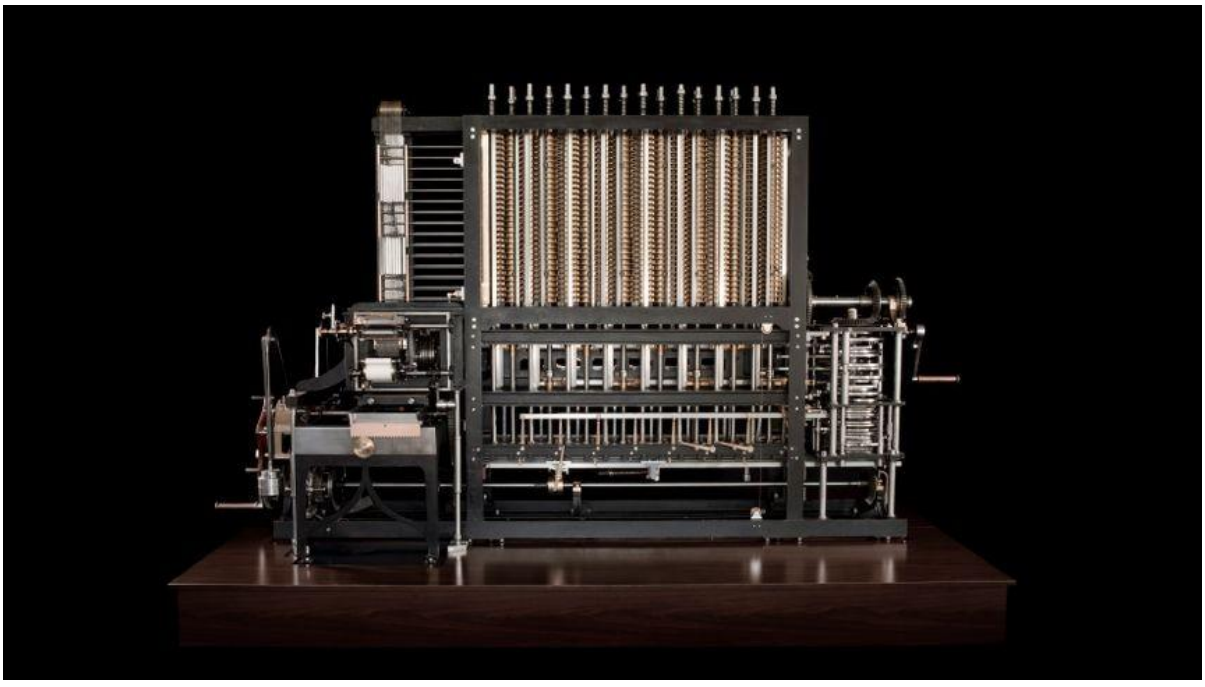
(Source: https://www.kurzweilai.net/average-transistor-price#!prettyPhoto)

Transistors make up electronic computers; the PDP-1 used 2,700 of them. Extrapolating to 2010 prices in the graph, those would cost $0.000027. And this downward trend continues unabated to this day.

Lastly, what about those mechanical pieces? We live in the material world, so understanding materials and their characteristics, sensing of all types, and actuation (movement) of mechanical pieces are all things mechatronic systems require. **And so the Third Rule of System Decomposition is to use the minimal amount of mechanism in any design, use the minimal number of parts, even use existing parts, if possible.** (Example: Ben's use of existing parts in the PDP-1 example). Of course, it is impossible to eliminate all mechanical parts, especially for those products that must perform an action, like a robot versus a computer. However, it is generally best to actuate electrically, control electronically, and use microcontrollers to handle the vast amount of product complexity.

MANAGING COMPLEXITY

While every product has a certain amount of inherent complexity, by using these rules you can architect your product to keep unintended additional complexity from creeping in, and you can distribute the complexity according to the technology's ability to successfully control and minimize it overall. As an example, let's say you wanted to build a machine to calculate polynomials via the difference method, and you wanted the results to be printed out into a book-like format. If you were Charles Babbage, you'd invent this:



Pictured: *A modern implementation of Babbage's Differential Engine 2, all done the mechanical engineering way, with whirring gears, cams, levers, and some rotary input, as from a steam engine. Output is printed pages. Today, we would probably write some Excel macros and call it a day. (Source:* https://www.computerhistory.org/babbage/)

CONCLUSION

Optimal product development requires a clear statement of the goal and a solid architecture to provide a roadmap to achieving the objective. A conscious decision needs to be made to control one or two of the "what", the "by when" and the "how much" of the product.

Additionally, there needs to be flexibility in the breakdown into mechanical, electrical, and software/firmware pieces to achieve your product goal as quickly and as cost-effectively as possible. The three rules of system decomposition are applicable to a wide variety of situations, for example when the division between firmware and hardware (electrical and mechanical) is made to minimize hardware costs or reduce product complexity.

DEC, led by the visionary engineer Ben Gurley, used these principles to create the first personal computer. The PDP-1 represented the best in the world that could be produced in technical computers at the time for the market it served. In studying the PDP-1, we have learned how breakthrough products are designed that meet time-to-market, budget, and performance constraints. With a little luck, these principles might also help you yield the first of a class of product[3].

ABOUT THE AUTHOR

Mike Cheponis is a Senior Systems Engineer in the Bay Area Office of Simplexity Product Development. Mike graduated from Culver Military Academy with honors in Science, Math, and Russian. He holds a bachelor's degree in Electrical Engineering and Computer Science with emphasis in abstract mathematics. He has done graduate work in Computer Science at Carnegie-Mellon University. Mike was the project lead, along with Joe Fredrick and Eric Smith, of the Computer History Museum's PDP-1 Restoration Project. Please stop by the Computer History Museum when it reopens, and play the original Spacewar! Game on the original machine where it all started. https://www.computerhistory.org/pdp-1/restoration/

[3] DEC later produced many advanced computers, and one of them, the PDP-11, was the base upon which UNIX was developed, and simultaneously birthed the C programming language, the oldest systems programming language still in common use.