大内先生　原文

APPLYING THE END-TO-END PRINCIPLE TO PLATFORM DESIGN

As we've seen, adding new features and interactions to a platform can be a powerful way to increase its usefulness and attract more users. But innovation can easily lead to excessive complexity, which makes the platform more difficult for users to navigate. Needless complexity can also create enormous technical problems for the programmers, content developers, and managers who are charged with updating and maintaining the platform. The derisive term *bloatware* has been coined to describe software systems that have become complicated, slow, and inefficient through thoughtless accretion of features.

However, avoiding innovation altogether is no solution. A platform that fails to evolve by adding desirable new features is likely to be abandoned by users who discover a competing platform with more to offer. Instead, a way must be found to strike a balance, changing the core platform only slowly while allowing positive adaptations at the periphery.

This concept is the equivalent, for a platform business, of a long-established computer networking idea known as the *end-to-end principle.* Originally formulated in 1981 by J. H. Saltzer, D. P. Reed, and D. D. Clark, the end-to-end principle states that, in a general-purpose network, application-specific functions ought to reside in the end hosts of a network rather than in intermediary nodes. In other words, activities that are not central to the workings of the network but valuable only to particular users should be located at the edges of the network rather than at its heart. In this way, secondary functions don't interfere with or draw resources away from the core activities of the network, nor do they complicate the task of maintaining or updating the network as a whole. Over time, the end-to-end principle has been expanded from network design to the design of many other complex computing environments.

One of the most storied examples of failure to heed the end-to-end principle concerns Microsoft's 2007 introduction of Vista, the latest version of its Windows operating system. CEO Steven Ballmer trumpeted Vista as "the biggest product launch in Microsoft's history" and backed the launch with a marketing budget of hundreds of millions of dollars.[7]

Yet Vista failed badly. The problem was that Microsoft's design team had sought to retain the software components needed to maintain backwards compatibility with older computer systems while adding features needed by next-generation systems—all within the core platform. As a result,

Vista was less stable and more complex than its predecessor, Windows XP, and outside app developers had difficulty writing code for it.[8]

Critics described Vista as worse than bloatware—in fact, they dubbed it *goatware* because it ate all a system's resources.[9] To this day, millions of Windows users have refused to adopt Vista, clinging to Windows XP despite repeated efforts by Microsoft to retire it. Ironically, while Microsoft stopped retail sales of XP in 2008 and of Vista in 2010, XP's market share in 2015 was above 12 percent, while that of Vista was below 2 percent.[10]

By contrast, when Steve Jobs returned to the leadership of Apple in 1997 after years of developing the ambitious but unsuccessful NeXT computer, he made a crucial decision that honored the end-to-end principle and helped lead to Apple's subsequent success. At NeXT, Jobs and his team had developed an elegant new operating system with a clean, layered architecture and a beautiful graphical interface. Now, planning a successor to Apple's Mac OS 9 operating system, Jobs faced a hard choice: he could merge the NeXT and Mac OS 9 software code, thereby producing an operating system that would be compatible with both systems, or he could jettison Mac OS 9 in favor of NeXT's clean architecture.

Jobs placed a dangerous bet on dumping the old code from OS 9. However, he made one concession: the design team developed a separate "Classic Environment" that would allow consumers to run their old OS 9 applications. This compartmentalized approach satisfied the end-to-end principle. The old code did not slow down or add complexity to the new applications, and new Mac buyers were unburdened by software written to accommodate apps they didn't own. Jobs's choice made innovation on the new Mac OS X easier and more efficient, which enabled Apple to develop new features that made Microsoft's operating systems look dated by comparison.[11] The end-to-end concept can also be applied to the design of a platform. In this case, the principle states that application-specific features should reside in the layer of process at the edge or on top of the platform, rather than at the roots deep within the platform. Only the highest-volume, highest-value features that cut across apps should become part of the core platform.

There are two reasons for this rule. First, when specific new features are incorporated into the core platform rather than attached to the periphery, applications that do not use those features will appear slow and inefficient. By contrast, when application-specific features are run by the app itself rather than by the core platform, the user experience will be much cleaner.

Second, a platform ecosystem can evolve faster when the core platform is a clean, simple system rather than a tangle of numerous features. For this reason, S. Y. Baldwin and K. B. Clark of Harvard Business School describe a well-designed platform as consisting of a stable core layer that restricts variety, sitting underneath an evolving layer that enables variety.[12]

Today's best-designed platforms incorporate this structural principle. For example, Amazon Web Services (AWS), the most successful platform for providing cloud-based information storage and management, focuses on optimizing a handful of basic operations, including data storage, computation, and messaging.[13] Other services, which are used by just a fraction of AWS customers, are restricted to the periphery of the platform and provided through purpose-built apps.

THE POWER OF MODULARITY

There are advantages to an integral approach where the system is developed as quickly as possible to serve a single purpose, especially in the early days of a platform. However, in the long run, a successful platform must have a more modular approach. A full discussion of this trade-off is well beyond the scope of this chapter, but we will cover some of the important ideas. We begin with a definition provided by Baldwin and Clark (1996):

Modularity is a strategy for organizing complex products and processes efficiently. A modular system is composed of units (or modules) that are designed independently but still function as an integrated whole. Designers achieve modularity by partitioning information into visible design rules and hidden design parameters. Modularity is beneficial only if the partition is precise, unambiguous, and complete. The visible design rules (also called visible information) are decisions that affect subsequent design decisions. Ideally, the visible design rules are established early in a design process and communicated broadly to those involved.[14]

In a 2008 paper, Carliss Young Baldwin and C. Jason Woodard provided a useful and succinct definition of a stable system core:

We argue that the fundamental architecture behind all platforms is essentially the same: namely, the system is partitioned into a set of "core" components with low variety and a complementary set

of “peripheral” components with high variety. The low-variety components constitute the platform. They are the long-lived elements of the system and thus implicitly or explicitly establish the system’s interfaces, [and] the rules governing interactions among the different parts.[15]

A critical factor that makes modularity so effective is that when systems are cleanly partitioned into subsystems, they can work as a whole by connecting and communicating through well-defined interfaces. The implication is that subsystems can be designed independently so long as they adhere to overall design rules and connect to the rest of the system only through standard interfaces. Readers will likely have heard the term *application programming interfaces*, or APIs. These are the standard interfaces that systems such as Google Maps, the New York Stock Exchange, Salesforce, Thomson Reuters Eikon, Twitter, and many more use to facilitate access by external clients to core resources.[16]

Amazon has been especially effective at opening APIs to its modular services. Figure 3.1 compares the range of APIs made available by Amazon and by the leading traditional retailer, Walmart, which is making a strong effort to become a significant platform competitor. As you can see, Amazon has by far outstripped Walmart in the number and variety of APIs provided.

The power of modularity is one of the reasons that the personal computer industry grew so quickly in the 1990s. The key components of PC systems were central processing units (CPUs) that provided the computation, graphical processing units (GPUs) that created rich images on the screen, random access memory (RAM) that provided working storage, and a spinning hard drive (HD) that provided large amounts of long-term storage. Each of these subsystems communi-

(図)

FIGURE 3.1. *Amazon has far more remixes or “mashups” of APIs than Walmart. These span payments, e-commerce, cloud services, messaging, task allocation, and more. While Walmart optimizes logistics, Amazon also allows third parties to build value on its modular services. Source: Evans and Basole using ProgrammableWeb data.17 Reprinted by permission.*

cated with the others using well-defined interfaces that allowed for tremendous innovation, as firms such as Intel (CPUs), ATI and Nvidia (GPUs), Kingston (RAM), and Seagate (HD) all worked independently to improve the performance of their products.

The reason that most platforms launch with a tightly integrated architectural design is that there is significant work involved in carefully specifying subsystem interfaces—and even in simply documenting them. When firms are pursuing narrow market windows with limited engineering resources, they can easily be tempted to skip the hard work of decomposing systems into clean modules and instead proceed as quickly as possible to a viable solution. Over time, however, this approach makes it much more difficult to mobilize an external ecosystem of developers who can

build on top of the core platform and extend its offerings into new markets.[18] Thus, a firm that has an integral architecture will likely have to invest in remaking its core technology.[19]

RE-ARCHITECTING THE PLATFORM

It is possible to pull off the trick of re-architecting a system toward a modular design. The first step is to analyze the degree of modularity the system has already achieved. Fortunately, a number of tools have been developed to accomplish this goal. Key among these are "design structure matrices" that allow a visual examination of the dependencies in complex systems.[20]

In a 2006 article in *Management Science*, Alan MacCormack and Carliss Baldwin document an example of a product that successfully evolved from an integral to a modular architecture.[21] When the software was put into the public domain as open source, the commercial firm that owned the copyright invested significant resources to make the transition. This was critical because the software could not have been maintained by distributed teams of volunteer developers if it had not been broken into smaller subsystems.

The need to re-architect a complex system is not unique to software. In the early 1990s, Intel faced a major challenge in growing its market. The performance of Intel's CPU chips was doubling every eighteen to twenty-four months.[22] Similar performance improvements were occurring in the other key PC subsystems: GPU, RAM, and hard drives. However, the information connections between the subsystems were still defined by an old standard called the Industry Standard Architecture (ISA). As a result, consumers saw little improvement in PC performance and thus had little reason to buy new machines. In a 2002 paper, Michael A. Cusumano and Annabelle Gawer document how Intel took the lead by investing in a new Peripheral Component Interconnect (PCI) to better connect the main subsystems, and the universal serial bus (USB) standard which fostered tremendous amounts of innovation in connected devices such as computer mice, cameras, microphones, keyboards, printers, scanners, external hard drives, and much more.[23]

ITERATIVE IMPROVEMENT: THE ANTI-DESIGN PRINCIPLE

When you're launching a new platform—or seeking to enhance and grow an existing platform—thoughtful attention to the principles of platform design will maximize your chances of value creation.[24] But as we've seen, platforms cannot be entirely planned; they also emerge. Remember that one of the key characteristics that distinguishes a platform from a traditional business is that

most of the activity is controlled by users, not by the owners or managers of the platform. It's inevitable that participants will use the platform in ways you never anticipated or planned.

Twitter was never meant to have a discovery mechanism. It originated as simply a reverse-chronological stream of feeds. There was no way to seek out tweets on particular topics other than by scrolling through pages of unrelated and irrelevant content. Chris Messina, an engineer at Google, originally suggested the use of hashtags to annotate and discover similar tweets. Today, the hashtag has become a mainstay of Twitter.

Platform designers should always leave room for serendipitous discoveries, as users often lead the way to where the design should evolve. Close monitoring of user behavior on the platform is almost certain to reveal unexpected patterns—some of which may suggest fruitful new areas for value creation. The best platforms allow room for user quirks, and they are open enough to gradually incorporate such quirks into the design of the platform.

Smart design is essential to building and maintaining a successful platform. But sometimes the best design is anti-design, which makes space for the accidental, the spontaneous, and even the bizarre.[25]

TAKEAWAYS FROM CHAPTER THREE

- The design of a platform should begin with its core interaction—one kind of interaction that is at the heart of the platform's value-creation mission.
- Three key elements define the core interaction: the participants, the value unit, and the filter. Of these, the value unit is the most crucial, and often the most difficult to control.
- In order to make the core interaction easy and even inevitable, a platform must perform three crucial functions: pull, facilitate, and match. All three are essential, and each has its special challenges.
- As a platform grows, it often finds ways to expand beyond the core interaction. New kinds of interactions may be layered on top of the core interaction, often attracting new participants in the process.
- It's important to design a platform thoughtfully to make mutually satisfying interactions easy for large numbers of users. But it's also important to leave room for serendipity and the unexpected, since users themselves will find new ways to create value on the platform.