CHAPTER

Platform Governance

6

Details create the big picture.
Sanford Weill

IN THIS CHAPTER

- Platform governance and its functions
- Three facets of platform governance: Decision rights, control, and pricing policies
- Who decides what (decision rights)
- The two functions of control
- The role of platform pricing policies
- Aligning governance with platform architecture, business models, and lifecycle stages
- Aligning decision rights, control portfolios, and pricing policies

6.1 PLATFORM GOVERNANCE AS THE BLUEPRINT FOR ECOSYSTEM ORCHESTRATION

If the metaphor for traditional organizations is an army, the metaphor for platform ecosystems is a symphony. The platform owner is like the conductor and the app developers are like the musicians. Each musician contributes a unique part of the overall musical score. The conductor helps the diverse musicians synchronize their own contributions to help ensure overall coherence in their collective performance. The conductor does not direct but rather orchestrates. The individual musicians *choose* to follow the lead of the conductor, who has limited direct authority or the depth of specialized musical talent of all of the musicians contributing to the performance. A good performance results from each musician being able to independently play her part but in synchrony with the others.

Like in a symphony, orchestration rather than control should be the focus of governance in platform ecosystems. Command-and-control structures work well in traditional organizations because legitimate hierarchical authority of managers over employees is accepted by both sides as a condition for employment. However, no such direct authority exists in a platform ecosystem. App developers are not employees of a platform owner; rather they often are free agents who typically specialize in niche domains outside those of the platform owner's. Performance-based rewards rather than punitive penalties are needed. The goal of good governance by a platform owner must therefore be to *shape and influence* its ecosystem, not to direct it (Williamson and De Meyer, 2012). And the goal of good

☆"To view the full reference list for the book, click here or see page 283."

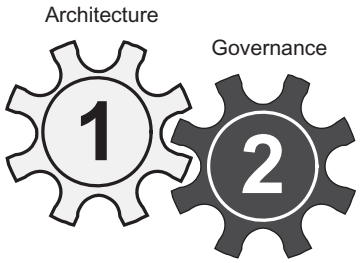

FIGURE 6.1

Governance is the second gear in a platform ecosystem's evolutionary motor.

governance is to respect the autonomy of app developers to do their thing while also being able to integrate their varied contributions into a harmonious whole. This is the essence of platform orchestration. Orchestration therefore entails influencing those whom the platform owner cannot control. The key function of governance is therefore to provide a context in which distributed innovation driven by app developers can emerge around a platform.

Platform governance is the second gear of platform orchestration (Figure 6.1). While architecture can reduce structural complexity, governance can reduce behavioral complexity. Platform governance matters because it determines whether innovation divisibility made possible by modular platform architectures is successfully leveraged (Boudreau, 2010; Rochet and Tirole, 2003; Tiwana et al., 2010). Governance of a platform ecosystem broadly refers to the mechanisms through which a platform owner exerts influence over app developers participating in a platform's ecosystem (Schilling, 2005, p. 159). Governance therefore flows from the platform owner who governs to app developers who are governed by the platform owner.

This chapter provides a foundation for understanding platform governance, beginning with its purpose and importance. It then explains the three dimensions of platform governance. The first is how the authority and responsibility for platform and app decisions are divvied up among app developers and a platform owner (decision rights). The second includes four mechanisms that a platform owner can mix-and-match to ensure goal convergence and coordination with app developers: gatekeeping, metrics, process control, and relational control. The third dimension is five platform pricing policies. It then explains why governance is inseparable from platform architecture and why realizing the potential of modular platform architectures requires aligning them. It finally explains how the three dimensions of governance—decision rights, control, and pricing—can be aligned with a platform's architecture, its business model, and its lifecycle stage. The mirroring principle provides a powerful mechanism for aligning decision rights in an ecosystem. But it also leaves some holes that must be plugged by platform control mechanisms. Aspiring for five simple rules can help design an effective but minimalist platform control system: simple, transparent, realistic, value-based, and fair.

6.2 THREE DIMENSIONS OF PLATFORM GOVERNANCE

Platform governance encompasses three dimensions (Table 6.1 and Figure 6.2): (1) the division of authority and responsibilities between the platform owner and app developers (decision rights partitioning), (2) the collection of mechanisms through which the platform owner exercises control over

Table 6.1 The Four Control Mechanisms and Their Prerequisites

Control Mechanism	Definition	Prerequisites
Gatekeeping	The degree to which the platform owner uses predefined criteria for what apps are allowed into the platform's ecosystem	• Platform owner must be competent to judge • Platform owner must be fair and speedy • App developers must be willing to accept such gatekeeping
Process	The degree to which a platform owner rewards or penalizes app developers based on the degree to which they follow prescribed development methods and procedures that it believes will lead to desirable outcomes	• Platform owner must have the knowledge to mandate methods to app developers • Platform owner should be able to monitor app developers' behaviors or verify compliance
Metrics	The degree to which the platform owner rewards or penalizes app developers based on the degree to which the outcomes of their work achieve performance targets predefined by the platform owner	• Metrics must be set by the platform owner, predefined, and objectively measurable
Relational	The degree to which the platform owner relies on norms and values that it shares with app developers to shape their behaviors	• Existence of shared norms and values between app developers and platform owner • Low app developer churn

app developers (the control portfolio, which includes the authority to accept or reject apps), and (3) decisions about how proceeds will be divvied up between a platform owner and app developers (pricing policies). Before we discuss each of these facets in detail, three important nuances must be emphasized. First, that these three dimensions of governance are interrelated; choices about one dimension can amplify or nullify the choices about the other two. Second, governance is costly. The optimal governance structure is the simplest one that achieves the goals of a platform at the least cost to both app developers and the platform owner. Third, governance structures are strategically inseparable from platform architecture. The same governance structure can be a disaster or a success depending on whether it is aligned well with the platform's architecture, the stage of its lifecycle, and its business model. Misalignment in any one of the three governance dimensions can lead to a collapse of the ecosystem. The role of each dimension of governance is shown in italics in Figure 6.2.

6.2.1 Decision rights partitioning

The first dimension of platform governance is decision-making authority or *decision rights*. A decision right broadly refers to who—the platform owner or app developer—has the primary authority and responsibility for making a specific type of decision—simply put, who makes what decisions

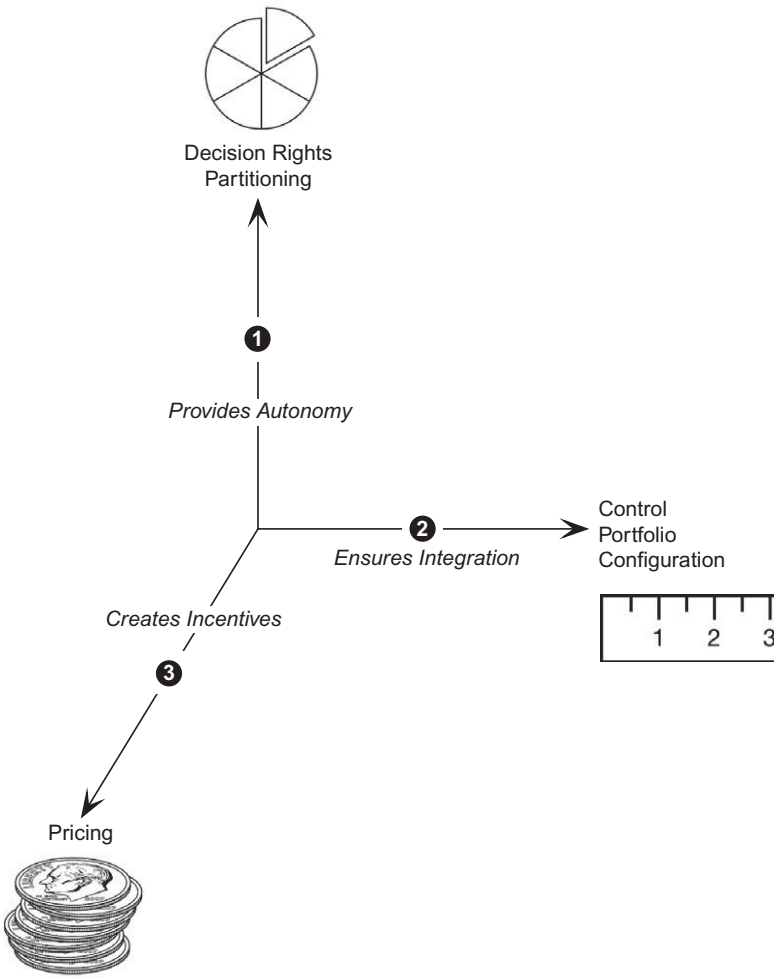

FIGURE 6.2

The three dimensions of platform governance.

(Athey and Roberts, 2001; Vazquez, 2004). A decision right can reside primarily with the platform owner, which represents centralization of a decision right. Or it can reside primarily with an app developer, which represents decentralization of a decision right. Perfect centralization and decentralization exist mostly only in theory and are rarely ever observed in practice. In practice, a decision right can reside anywhere along the continuum of complete centralization and complete decentralization. This means that both the platform owner and app developers have some authority and responsibility for most decisions but it might lean more toward one party. It is therefore useful to think of any decision right as a slider that can be moved along the two extremes of centralization and decentralization (Figure 6.3).

6.2.1.1 Platform versus app decision rights

Thinking of each individual decision in a platform ecosystem can quickly devolve into an overwhelming and meaningless exercise. It is therefore necessary to reduce the broad notion of decision rights into a more comprehensible and tenable subset or classes of decisions. In a platform ecosystem, there are four broad sets of decision rights. The first distinction is whether decision rights pertain to the platform or the platform's apps. *Platform decision rights* refer to whether the platform owner or the app developers have the authority and responsibility for making decisions directly pertaining to the platform. Similarly, *app decision rights* refer to whether the platform owner or the app developers have the authority and responsibility for making decisions directly pertaining to apps. On first glance, you might conclude that platform decisions would be naturally made by the platform owner and app decisions by app developers. But this is a misleading oversimplification for four reasons.

First, decision rights can be placed anywhere along the continuum in Figure 6.3. It is a purposeful choice, not a given. This means that platform and app decision rights do not necessarily have a 1:1 mapping to platform owners and app developers, as the simple representation in Figure 6.4 would lead one to conclude. Instead, we must think of them in terms of their *degree* of decentralization. App decisions, for example, can be decentralized to a greater degree on one platform and less decentralized on another. This means that they are often *shared* to varying degrees between app developers and platform owners; the question is about the degree to which they are shared. We therefore refer to such sharing as partitioning of decision rights. Second, even slight differences in the degree of their centralization and

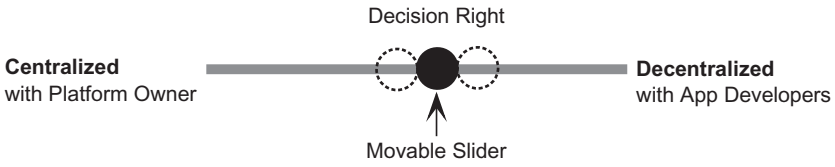

FIGURE 6.3

A decision right can be placed anywhere on the decentralization continuum.

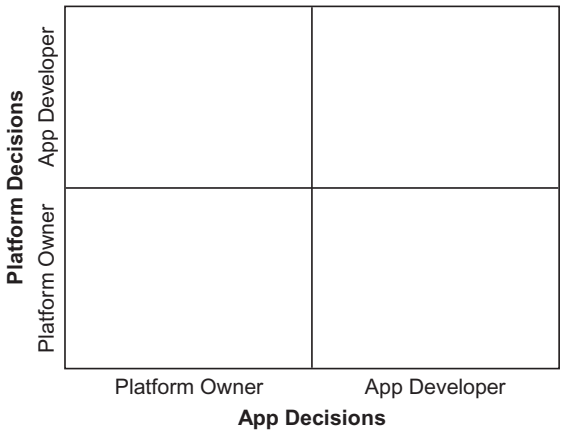

FIGURE 6.4

Platform and app decision rights can be assigned to platform owners or app developers.

decentralization can put two similar platforms on substantively different evolutionary trajectories. Such small differences in the degree of decentralization of platform and app decisions can amplify or mute the benefits of architectural choices made by platform owners and app developers. In other words, they have strong interactions and dependencies with architecture at the platform level and at the app level. A decision rights arrangement that enables efficient specialization among app developers under one architecture can become its Achilles heel under another. Therefore, the partitioning of decision rights must be aligned with the architecture of a platform and the microarchitecture of apps. Fourth, decision rights partitioning between two parties have an assigner and an assignee. The allocation of decision rights to an assignee might have little meaning if the assignee does not—or cannot—accept the responsibility for those types of decisions.

6.2.1.2 *Two classes of decision rights:* strategic *and* implementation

Decision rights for a platform or apps can be broadly split into two broad categories[1]: *strategic* and *implementation*. Strategic decision rights pertain to what a party (app developer or platform owner) should accomplish and implementation decisions are about how it should accomplish it. Strategic decision rights therefore are direction-setting, specification-oriented decisions. Implementation decisions are technical execution decisions that pertain to the choice of features, functionality, design, user interface, and implementation details of a software subsystem. Both classes of decision rights apply to platforms and apps. Therefore, *platform strategic* decision rights represent the authority and responsibility for specifying what the platform must accomplish and *platform implementation* decision rights represent how a platform actually accomplishes those objectives. Similarly, *app strategic* decision rights represent where the authority and responsibility for specifying what an app must accomplish reside and *app implementation* decision rights represent how an app actually accomplishes those objectives.

The division of app decision rights can vary by app, and the same platform can use different combinations of app decision rights partitioning for different apps. Similarly, an app developer with multiple apps on the same platform might see different decision rights structures used for various apps on the same platform. And a multihoming app developer with the same app on multiple platforms will likely encounter different decision rights structures for that app on different platforms. For example, the developer of the Skype Mobile app might see different decision rights structures on iOS, Android, and Windows Mobile platforms. The decision rights structure for an individual app on a given platform can succinctly be represented in the decision rights partitioning framework in Figure 6.5. The sliders are independently movable, must be aligned with platform architecture and app microarchitecture at the outset, and be moved over time to maintain alignment.

6.2.2 Control portfolio design

The second dimension of platform governance is control. Control refers to the means through which the platform owner ensures that the app developers' work is aligned with what is in the best interests of the platform. Control is implemented by the platform owner over app developers using a variety of control *mechanisms*. Therefore control mechanisms are the tools that platform owners use to implement and enforce rules that reward desirable behavior, punish bad behavior, and promulgate standards of behavior among app developers (Evans and Schmalensee, 2007). Control mechanisms can either be formal or

[1]For research on the distinction between these classes of decision rights, see Tiwana and Konsynski (2010) and Jensen and Meckling (1992).

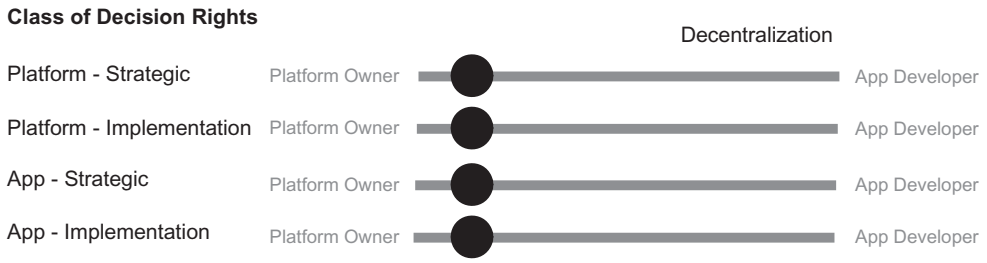

FIGURE 6.5

A decision rights partitioning framework.

informal. Several such mechanisms can be used together and this combination of control mechanisms represents the *control portfolio* used by a platform owner over an app developer. Different apps in the same platform can—but need not—have differently structured control portfolios.

Figure 6.6 summarizes the four control mechanisms that can be used by a platform owner.

A platform owner can use three formal control mechanisms and one informal control mechanism. All formal control mechanisms focus on imposing rules and standards that a platform owner expects app developers to aspire toward meeting (Evans and Schmalensee, 2007, p. 22). The formal control mechanisms include control through gatekeeping, process control, and control using metrics. An informal control mechanism that can be used in addition to these formal mechanisms is relational control. Table 6.1 summarizes these control mechanisms. Each control mechanism also has prerequisites for using it, also summarized in the table.

6.2.2.1 Gatekeeping

The first formal control is via *gatekeeping*. Gatekeeping represents the degree to which the platform owner uses predefined objective acceptance criteria for judging what apps and app developers are allowed into a platform's ecosystem. The platform owner sets this criteria, not just for *what* is allowed in but also *who* is allowed in. This formal control is also known as input control (Cardinal, 2001), much like how organizations select which candidates to hire as employees from a pool of applicants. In platforms, this entails an app developer submitting an app to the platform owner for evaluation for inclusion in the platform's ecosystem.

When gatekeeping is used by a platform owner, the platform owner reserves bouncer rights over apps that can be included as part of the ecosystem (Boudreau, 2010; Evans et al., 2006, p. 254). Apple,

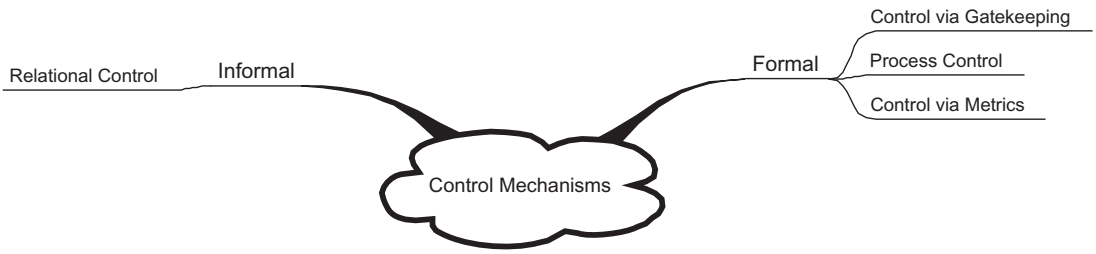

FIGURE 6.6

Control mechanism choices usable by platform owners.

for example, uses gatekeeping heavily as a control mechanism in its iOS platform. However, three important requirements must be met for control via gatekeeping to be viable. First, the platform owner must be sufficiently competent to judge app developers' submissions. Second, the platform owner must be able to do so fairly and speedily. Third, app developers must be willing to be subjected to such gatekeeping. If the platform owner is not perceived as being objective, fair, and speedy in judging inputs into the platform, it exposes app developers to considerable risk and uncertainty about platform-specific work they have already invested in completing. If deployed carelessly, it can discourage app developers from committing to developing for a platform.

6.2.2.2 Process control

The second form of control is *process control*, which refers to the degree to which a platform owner rewards or penalizes app developers based on the degree to which they follow prescribed development methods, rules, and procedures that it believes will lead to outcomes desirable from a platform owner's perspective. Such desirable outcomes in platforms refer to apps that will interoperate well with a platform, not whether they do well on the market. Such rules and procedures are prescribed by the platform owner, who then evaluates the extent to which individual app developers followed them. Compliance with prespecified processes is rewarded and noncompliance is penalized. These processes must be defined upfront and known to app developers for process control to be viable. In many software platforms, platform development and testing tools, simulation environments, and developer toolkits provided by the platform owner to app developers are mechanisms through which a platform owner attempts to implement process control. Process control, however, has two important requirements to be viable. First, the behaviors of app developers should be observable and monitorable by the platform owner. Such behavior observability need not be direct and can be through electronic audit trails and developer logs. Second, the assertion behind process control is that if app developers follow the processes prescribed by the platform owner rather than being left to their devices, the likelihood that they will produce technically better-performing apps is going to be higher. The platform owner must understand the work of the app developers sufficiently well—or better—in order to prescribe processes that will indeed improve the odds of their work being successful (particularly, app integration with the platform).

6.2.2.3 Metrics

A third formal control mechanism is through the use of *metrics*. This formal control refers to the degree to which the platform owner rewards or penalizes app developers based on the degree to which the outcomes of their work achieve predefined target performance metrics. Researchers call this output control because it evaluates the output of app developers' work (Ouchi, 1979). The requirement for metrics-driven control is that such metrics must be: (1) prespecified by the platform owner and (2) objectively measurable. The extent to which an app developer's completed work meets the targets determines the rewards or penalties that the platform owner imposes on the app developer. Metrics-based control comes from the legacy world of traditional software development where an internal manager or client could specify target criteria such as budget, project schedules, and acceptable defect levels to the internal IT department or an outside vendor. However, such metrics have little meaning if the platform owner does not—or cannot—prespecify target metrics such as development budgets or schedules that it expects app developers to meet.

In platforms, performance and survival of an app in the brutal marketplace serves as a powerful metric that eliminates the need for much metrics-based control (Armstrong, 2006; Bester and Krähmer, 2008). In many contemporary platforms, metrics-based control is therefore rarely used

because the end-user market rewards high-quality output with strong sales and penalizes low-quality output with poor sales. In other words, the use of market competition can substitute for control using metrics. Examples of some *weak* app-level metrics in platforms can include performance, memory utilization, and speed at an operational level. They can also include *market-oriented* metrics such as unit sales, downloads, and end-user ratings. Use of market-oriented metrics such as these is also closely tied to the third dimension of governance: pricing structures used by the platform owner to divvy up proceeds with app developers. The sole purpose of market-oriented metrics is therefore simply measurement to implement revenue-splitting agreements.

6.2.2.4 Relational control

The fourth and only informal form of control is *relational control*. This control mechanism refers to the degree to which the platform owner relies on norms and values that it shares with app developers to influence their behavior. This control mechanism thus relies on the platform owner to provide an overarching collective goal for the platform ecosystem; a sort of shared identity that defines the character of the platform ecosystem and rallies app developers around it by harmonizing their own goals with those of the platform. Such system-level goal-setting sets a trajectory to evolve the platform ecosystem and creates unity in effort without micromanaging app developers (Meadows, 2008, p. 165). Therefore, a shared culture, similar set of values, and shared norms provide a common ground that can align the objectives of the platform owner and the work of app developers. Researchers call this clan control because it relies on clannish behavior (Kirsch, 1997; Ouchi, 1980). Relational control is widely used in open-source platforms such as various Linux platform development communities. The upside to relational control is that it is often one of the least costly forms of control since it requires little enforcement and effort from the platform owner. However, the preexistence of shared norms and values in the app developer community is an important precondition for relational control to be viable. This requirement is often not met if an app developer does not share a long history with the platform. It is also unlikely to be met in platforms that have an ongoing inflow of new app developers who join the platform but do not have the same shared history, professional values, and norms as existing developers. High app developer churn therefore reduces the viability of relational control. Therefore, relational control rarely suffices by itself, although it can nicely complement other formal control mechanisms.

The combination of these four control mechanisms at the platform level describes a control portfolio, as illustrated in Figure 6.7. Each control mechanism is an independent slider that can be omitted, used to some degree, or used extensively as part of a control portfolio. The control portfolio used by a platform owner can therefore be succinctly described using this template.

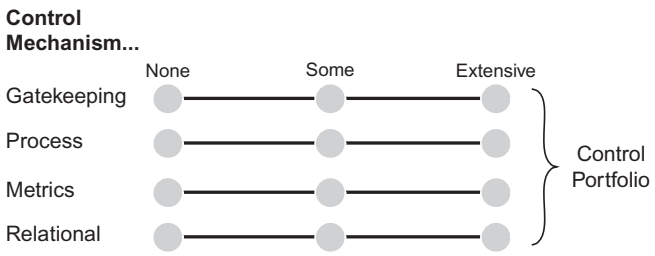

FIGURE 6.7

A control portfolio is a combination of the four control mechanisms used by a platform owner.

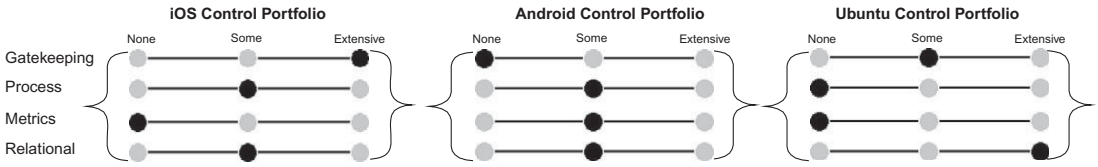

FIGURE 6.8

An illustration of the control portfolios used by three platforms.

Figure 6.8 illustrates the control portfolios used by three different platforms: iOS, Android, and Ubuntu. As this comparison illustrates, some platforms rely on a combination of some degree of diverse control mechanisms (e.g., Android) whereas others lean heavily toward the extensive use of one dominant control mechanism in designing their control portfolio.

6.2.2.5 Attempted versus realized control

Just because a platform owner decides to implement a particular control mechanism over all developers does not necessarily result in the expected behavior. In other words, there is a difference between control attempts by a platform owner and the degree to which the platform owner is actually able to realize it over individual app developers. A control mechanism is effective when the level of attempted control and realized control for that mechanism are not too far apart. This requires two things. First, the control mechanism should be accepted by app developers as being legitimate, fair, and reasonable. The power of a control mechanism requires the consent of the governed. Leadership is possible only when others choose to follow, so some consensus is always needed for any control mechanism to be realizable. Second, the prerequisites for it (Table 6.1) must be satisfied in the relationship between the platform owner and individual app developers. Both of these conditions are more likely to be met with some app developers and less with others. Therefore, there might be little variability in a platform owner's attempted control portfolio but considerable variability in the level of control realized by the platform owner over individual app developers. As illustrated in Figure 6.9, a uniform control portfolio that the platform owner attempts to use over three app developers might be realized differently with each of them.

The optimal control portfolio used in a platform should further multiple, often competing objectives of a platform and should also be aligned with the platform's architecture. We subsequently describe five guiding principles for how platform owners can design an optimal control portfolio in this chapter.

6.2.3 Pricing

The third dimension of platform governance is platform pricing policies. The goal of pricing policies is to create incentives that are compelling enough to encourage app developers to make personal investments to ensure the prosperity of their own app offerings, and in turn the vibrancy of the platform ecosystem as a whole. Pricing policies encompass five choices:

1. Whether pricing should be symmetric or asymmetric for the two sides of the platform
2. If asymmetric, who to subsidize and for how long?
3. Pricing for access versus usage?
4. Pie-splitting using a fixed scale or a sliding scale?
5. App pricing decisions

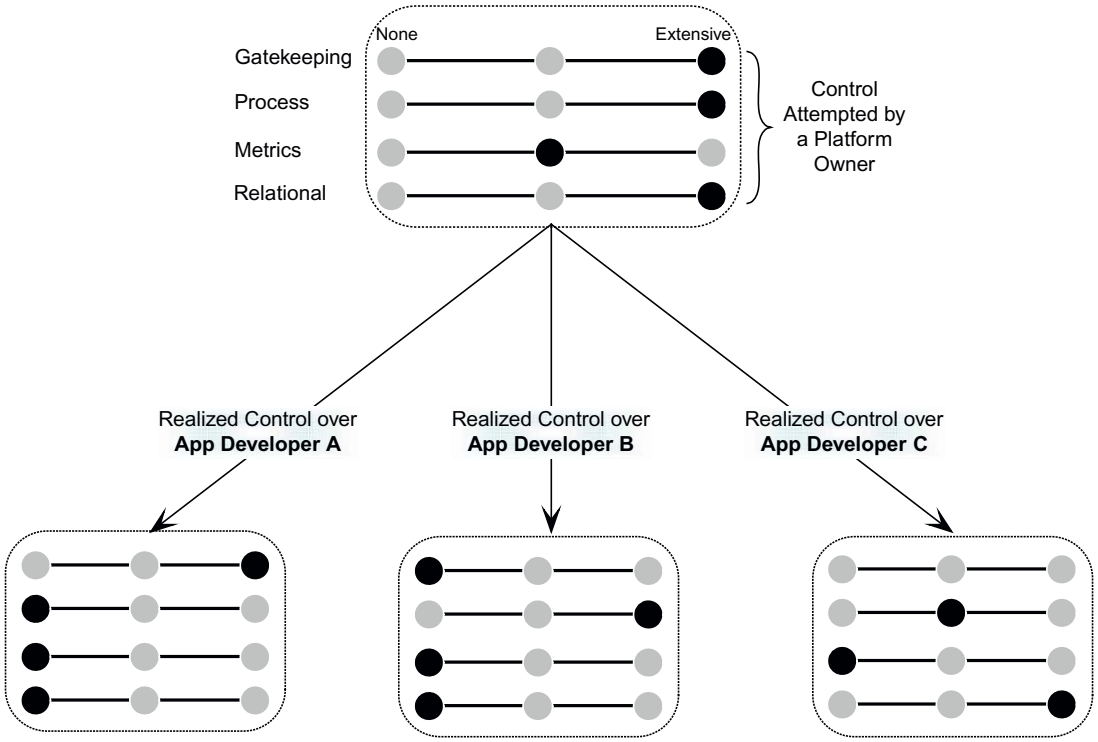

FIGURE 6.9

Control attempts by a platform owner over app developers are often not realized uniformly.

6.2.3.1 Decision 1: symmetric or asymmetric pricing?

Recall that there are at least two sides to a platform: app developers and end-users. The first pricing decision that a platform owner must make is whether you make money on one side and give a break to the other side (asymmetric pricing), or make money on both sides (symmetric pricing). The choices are illustrated in Figure 6.10. If a platform owner chooses to make money on only one side, it usually entails zero pricing or even negative pricing on the other side. For example, some platforms make money from end-users but subsidize app developers. Often, platform owners go even beyond subsidies by providing tools, hardware, and other costly incentives that cause them to lose money on one side. The subsidized side of the platform can often make up the losses in the form of increased profits from the other side. For example, Amazon Kindle and gaming console developers (e.g., Nintendo, Sony Playstation, and Microsoft Xbox) subsidize end-users. In contrast, mobile computing platform owners such as Apple, Google, and Blackberry subsidize app developers. In traditional businesses, this would be considered irrational and a recipe for the business model to collapse. However, in platform markets, strategically subsidizing one side can more than make up the lost money from the other side. It is important to make platform pricing decisions for long-term profitability, which might be at odds with short-term profitability. This is a tricky strategic decision and is tied directly to the stage of the platform's lifecycle, criticality of network effects, and competitive dynamics in the platform's immediate market, as we discuss later in this chapter.

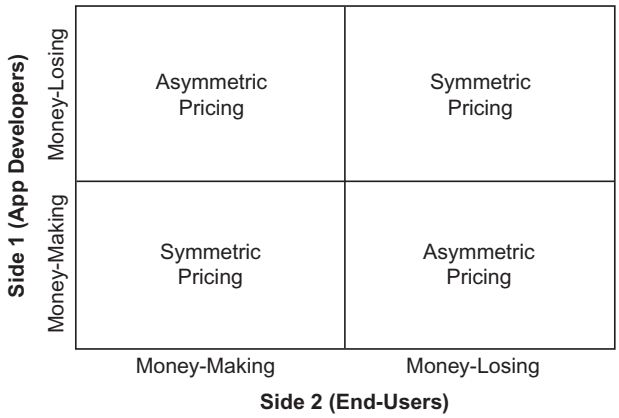

FIGURE 6.10

Asymmetric versus symmetric pricing across two sides of a platform.

6.2.3.2 Decision 2: which side to subsidize and for how long

The second pricing decision platform owners must make is which side to subsidize (if any) and for how long. For example, a platform owner might pay one side to join simply to attract the other side. The unsubsidized side should, however, always be the money-making side. If one side subsidizes the other, the platform owner must also decide how long this subsidy should last. Abruptly attempting to end the subsidy on the subsidized side can be challenging and must be planned in advance. For example, the newspaper publishing industry (e.g., the *New York Times*) subsidized readers by offering them free access to online content from the mid-1990s to the mid-2000s. This successfully attracted advertisers (the other side). But when they tried to end the subsidy to the readers by attempting to charge them access fees, it resulted in mass attrition of readership. If they had the foresight to plan to end the subsidy, a more feasible approach would have been to offer readers expiring monthly credit or points that they could have redeemed to gain free access during the intended subsidy period. We explain later how platform owners can make such subsidy decisions by considering various aspects of their platform and its competitive environment.

6.2.3.3 Decision 3: usage versus access pricing

The third pricing decision platform owners must make is about pricing for access and pricing for usage (Figure 6.11). It is important to set separate prices for access and usage. Access fees are the prices charged to app developers (but rarely to end-users) for gaining access to the platform. Usage fees are prices charged to app developers for actual usage of the platform. It often makes sense to set one of these prices to zero or even negative (e.g., where app developers are paid by the platform owner using side payments) depending on the platform's stage in its lifecycle, whether it is in the pre- or post-dominant design phase, and the accumulation of critical mass on the developer and the end-user sides. We subsequently explain how usage and access pricing policies can be aligned with these properties of a platform.

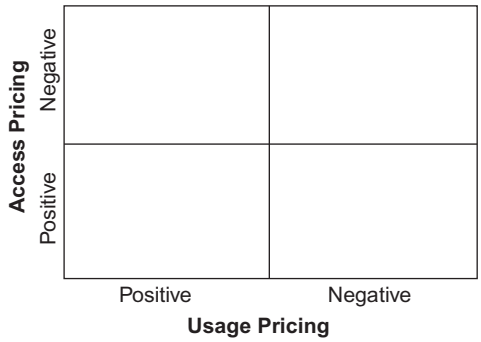

FIGURE 6.11

Access and usage pricing can be negative or positive.

6.2.3.4 Decision 4: pie-splitting using a fixed scale or a moving scale?

The fourth pricing decision that platform owners must make is about the pie-splitting structure. This means that for every dollar of revenue earned from an app, how the revenue will be shared between the app developer and platform owner. The choice is one of adopting a fixed scale or a moving scale. A fixed scale means that the platform owner keeps a predetermined percentage of each dollar of revenue (say, 30%, as in Apple's case). A moving scale means that the scale changes with an increase in the units of the app sold or in its usage. The percentage can either rise (a rising scale) or decrease (a sliding scale) with an increase in sales volume.

These three choices are illustrated in Figure 6.12. Mobile computing platforms most commonly use a fixed scale structure. The traditional book publishing industry, in contrast, frequently uses a rising scale where the first block of copies sold (say the first 5,000) pay the authors a small percentage (e.g., 5%), the next block pays a higher percentage (e.g., 7.5%), and all subsequent copies sold pay a higher ceiling percentage (e.g., 10%). As we subsequently explain, there is no one right model. The choice can have a strong impact on the incentives of app developers, and a model appropriate for one stage of the platform's lifecycle might not be appropriate for another stage.

6.2.3.5 Decision 5: app licensing decisions

The final pricing decision is app-specific pricing and licensing decisions. Unlike the preceding four pricing policies where the platform owner has the primary say, the app developer usually has considerable leeway in app pricing decisions. However, such decisions are likely to be constrained by the other four platform owner pricing choices. An app developer can use one of three pricing structures for an app:

1. *Single perpetual license*: This sort of license is perpetual in the sense that a one-time payment by an end-user grants nonexpiring rights to use the app. The license can either be an individual license (the buyer can use the app on a limited or unlimited number of her own instantiations of the platform), a machine license (which allows the end-user to use it on a particular machine that it was purchased for but not for any subsequent machine), or a floating license (which allows the user to use it on any one machine at a time and the host machine can change without constraints). This model is the most widespread in the traditional software industry. An app that is given away for

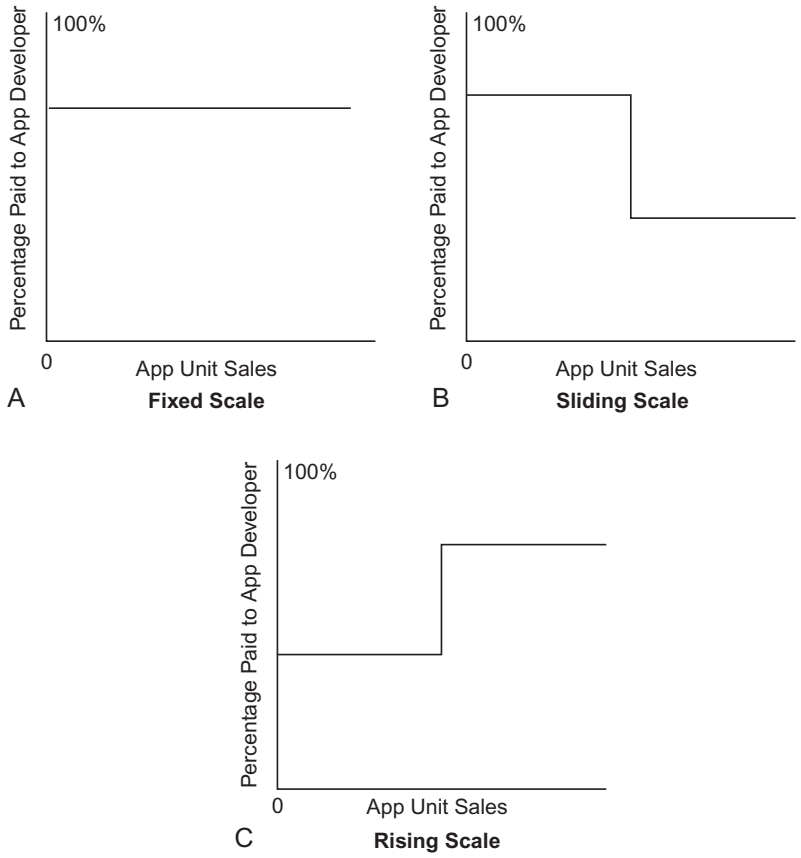

FIGURE 6.12

Pie-splitting between the platform owner and app developers can be on a fixed, sliding, or rising scale.

free is a single perpetual license priced at zero dollars, with revenue generated from a third side (e.g., advertising) or from the sales of a complementary product or service (e.g., department store sales, an airline flight, a banking relationship, or a credit card account).

2. *Subscription-based license*: This sort of license allows the user to use the app for the duration of an active subscription period (e.g., 1 year) and usually includes all future updates to an app during that period. This licensing model is a service-oriented model. The ubiquitous IP connectivity of contemporary software platforms allows this model to now be cost-effectively implemented because instant verification of an active subscription is possible each time the end-user attempts to use an app.
3. *Usage-based license*: This sort of model charges the user based on actual usage and requires some direct measure of usage (such as number of times used or number of hours used). This licensing model is a utility-oriented model, much like you pay for the actual amount of electricity or water consumed. The necessary condition for this licensing model is the ability to precisely and

cost-effectively meter usage. This is increasingly viable in software platforms due to their near-ubiquitous IP connectivity. The actual revenue need not directly be recouped from the end-user; it can be recouped from a third side (e.g., an advertiser) on a multisided platform. These three pricing strategies can directly be linked to free versus paid apps common in contemporary mobile app stores. The free model is a usage-based license subsidized by advertising. It generates an ongoing stream of revenue for the app developer, unlike the paid model that typically generates revenue from a one-time sale.

6.3 ALIGNING GOVERNANCE

Alignment is typically between two things. Table 6.2 summarizes what each of the three dimensions of platform governance must be aligned with. All the dimensions of platform governance must be aligned with its architecture. Innovation at the app level thrives on autonomy—granted by architecture—plus incentives of app developers for risk-taking—granted through governance. The decision rights dimension must additionally be aligned with the platform's business model. The pricing dimension of governance must additionally be aligned with the platform's lifecycle stage and its business model. Platform lifecycle refers to the three dimensions described in Figure 2.1 in Chapter 2: emergence of a dominant design, diffusion among end-users, and phase along the S-curve. We will stay clear of an academic treatise of what business models mean. In this book, it is sufficient to think of the business model simply as how a platform ecosystem's participants hope to pay their bills. It is how app developers and platform owners intend to generate revenue.

6.3.1 Aligning decision rights partitioning

Aligning the distribution of decision rights in a platform ecosystem requires analyzing primarily how they fit with the ecosystem architecture and secondarily with the platform's business model. As Table 6.2 shows, decision rights need to be aligned with ecosystem architecture and with the platform's business model. Two heuristics apply to aligning decision rights with architecture: the mirroring principle and specialized knowledge.

6.3.1.1 Aligning decision rights and architecture using the mirroring principle

The gist of the mirroring principle is that organization of the development teams in a platform ecosystem must mirror its technical architecture (Baldwin and Clark, 2000, p. 47). The division of authority over decisions (decision rights) is a defining property of organizational structure (Nault, 1998).

Table 6.2 Considerations in Aligning Governance Choices

Governance Dimension	Architecture	Lifecycle	Business Model
Decision rights	●		●
Control	●		
Pricing	●	●	●

The architecture of a platform ecosystem determines the structure of dependencies between the task structure of the platform owner's work and the app developers' work. The boundaries of subsystems in an ecosystem determine feasible boundaries between groups that are responsible for each of them. Properties of a platform's architecture therefore determine whether the app development work can be done by outside app developers or is better done inhouse by the platform owner. Therefore, modularization of platform architecture provides a powerful mechanism for the separation of responsibility between the platform owner and app developers. Although a modular platform architecture provides a framework for making innovation effort divisible, the organization of the ecosystem must leverage this property to benefit from modularization.

The technical architecture of an ecosystem is therefore inseparably intertwined with the organizational structure of the ecosystem that is used. Architecture of an ecosystem imposes constraints on app developers who interact with it and build their work on it. It is of little consequence to attempt to resolve the debate about where architecture precedes organization or vice versa. Architecture and decision rights partitioning must be mirror images for them to reinforce each other's potential advantages. A misalignment between the two can create a severe coordination deficit (Sosa et al., 2004). Therefore, the relationships between the platform owner and app developers ought to be analogous to the relationship between the platform and apps. The boundaries of the development teams responsible for apps and the platform therefore usually follow the boundaries between apps and the platform within an ecosystem. Boundaries of individual organizations' responsibilities must align with boundaries of apps vis-à-vis the platform. If the platform's architecture is highly interdependent (monolithic), so should the linkages between app developers and the platform owner.

SHARED VERSUS PROPRIETARY ARCHITECTURE

If the ownership of a platform is shared among multiple owners or it is based on an open standard, it represents a shared rather than a proprietary platform (which belongs to one platform owner). These multiple owners of the platform must cooperate to make any changes to the platform architecture. Such distributed ownership mitigates the hold-up risk faced by app developers but also suffers from coordination challenges. It can result in a gridlock in making platform strategic decisions that can impede the evolution of the platform as well as its ecosystem. In contrast, a single platform owner has more power over the direction of a platform. It is therefore useful to view an increase in the number of platform owners as diffusion of power related to the platform's architecture.

Thus architecture precedes how an ecosystem is organized, and how much of the innovation work is done inhouse vis-à-vis outside developers. While the initial platform designers have knowledge of the entire platform, app developers do not need the same depth of knowledge of the platform if the architecture is sufficiently modularized. Their work can proceed independently, largely in ignorance of the inner workings of the platform. Compliance with the platform's interface specifications ensures that their apps will integrate and interoperate with the platform. This allows greater specialization by app developers in the domain of their apps. Thus technical division of the ecosystem through architecture enables division of cognitive labor.

How development tasks for a platform and its apps in an ecosystem are partitioned must match how authority for them is divvied up between the platform owner and app developers. In other words, the partitioning of decision rights should mirror the technical architecture of the platform ecosystem. Modular ecosystem architectures therefore require modular partitioning of responsibilities and authority

across the ecosystem. A modular organizational structure is where each organization participating in the ecosystem is the owner-operator for their subsystem—autonomous, perhaps even in competition with others within the ecosystem. A modular organizational structure in an ecosystem is therefore one where the authority for platform decisions resides primarily with the platform owner and the authority for app decisions resides primarily with individual app developers. Architectural decoupling between a platform and apps must therefore be mirrored in the division of authority and responsibility across the ecosystem. The more modular the architecture of the platform ecosystem, the more modular should the partitioning of decision rights be for the platform and apps. Modularization of decision rights therefore economizes on the limited coordination capabilities of app developers and the platform owner (Ethiraj and Levinthal, 2004a,b; Simon, 1978). By extension, the more modular the microarchitecture of an app, the more modular should the partitioning of decision rights be for that app. In contrast, the more monolithic a platform's architecture, the more centralized and concentrated should the allocation of decision rights be. Such mirroring creates synergies such that governance amplifies the benefits of modular architectures. The implementation of the mirroring principle is illustrated in Figure 6.13.

6.3.1.2 Aligning decision rights with specialized knowledge

The second criterion for aligning decision rights is to locate the authority for each class of decisions where the specialized knowledge needed to make those decisions is located (Jensen and Meckling, 1992; Macher and Boerner, 2012; Tiwana, 2009). This criterion ensures that decision rights are aligned with the platform's business model. Recall that the four types of decision rights were the *strategic* and *implementation* decisions pertaining to the platform and individual apps. Using this logic, the optimal location for each of the four classes of decision rights can be determined. Decision rights are aligned with the platform business model when each class of decision rights is located with the core knowledge needed to make that class of decisions.

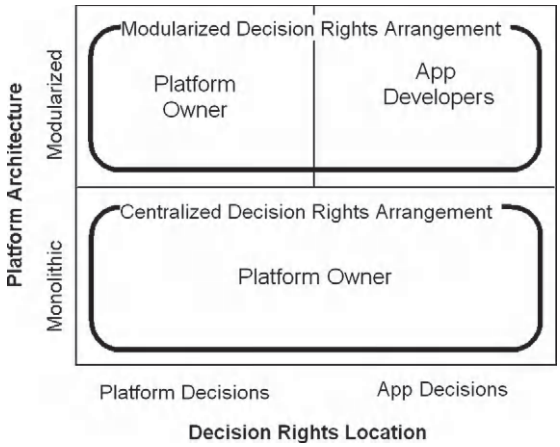

FIGURE 6.13

Using the mirroring principle to align decision rights with architecture.

Table 6.3 Aligning Platform and App Decision Rights with Ecosystem Architecture

Decision Right	Core Knowledge Needed	Complementary Knowledge Needed	Locus of Core (Complementary) Knowledge	Optimal Location
Platform: strategic	Platform's market	Some knowledge of user needs	Platform owner (app developer)	*Platform owner*, but with app developer input
Platform: implementation	Platform's technology	–	Platform owner	*Platform owner*
App: strategic	App domain user needs	–	App developer	*App developer*
App: implementation	App domain user needs	Some knowledge of platform technology	App developer (platform owner)	*App developer*, but with platform owner input

Table 6.3 summarizes the knowledge-authority co-location logic for aligning decision rights with the distribution of specialized expertise in a platform ecosystem. Consider each of the four classes of decision rights. Two classes of decisions—the platform's *implementation* decisions and an app's *strategic* decisions—are the most straightforward because they rely primarily on one specific type of core knowledge.

Platform implementation decision rights are operational decisions that pertain to the choice of features, functionality, design, and user interface of the platform (Tiwana, 2009). The core knowledge that they require is a deep understanding of the platform's technology, which is usually the platform owner's knowledge. Platform implementation decisions should therefore ideally be centralized (i.e., held by the platform owner). Centralization of platform implementation decisions also gives a platform owner architectural control over the ecosystem, which subsequently impacts the platform owner's ability to influence its evolutionary trajectory. Centralization of platform implementation decisions also potentially works well for app developers because it allows them to benefit from the scale economies generated by sharing the centrally managed commonalities in functionality that their apps might build on but that do not differentiate them in their own niche markets.

App strategic decision rights pertain to decisions about what an app should do. Recall that the value that end-users attach to a platform is often shaped by capabilities and functionality beyond those natively built into a platform. The source of such capability extension is apps. Apps require a deep understanding and in-depth knowledge of diverse user needs and application domains that might not be obvious to a platform owner but that app developers can bring to the table. These apps therefore target niche market segments of diverse sizes and structures subject to different levels of technical and business uncertainty (van Schewick, 2012, p. 134). These markets might also have different norms for compatibility across rival platforms (i.e., whether an app designed to work on one platform must also work on a rival platform) (Rysman, 2009). For example, social networking app users might expect cross-platform compatibility but instant messaging app users might not expect it. (These markets might also have widely varying norms for app microarchitectures, hence app-level modularity.)

Such sources of knowledge of users' needs in diverse, narrow application is increasingly dispersed (Dhanaraj and Parkhe, 2006). App developers are more likely than platform owners to have deep

knowledge of such user needs in an app's domain. The locus of the core knowledge needed for decisions about what an app should do—app strategic decisions—is therefore app developers. Application strategic decision rights should therefore be decentralized (i.e., held by app developers). However, this is only appropriate when the platform architecture is sufficiently modularized. If the platform architecture were monolithic, app development would either have to be done by the platform owner or a combination of precise metrics and intense monitoring coupled with intense coordination between the platform owner and app developer would be needed.

As we subsequently describe in Part IV of this book, this has significant evolutionary consequences for apps. For example, decentralization of app strategic decisions gives app developers considerable control over cross-platform compatibility, business models used for revenue generation, and the power to adapt as fast as they need to. The diverse markets served by such apps can also vary significantly in how dynamic they are, often requiring different apps to adapt at different rates. Such decentralization therefore increases the prospects for individual app developers to maintain competitive differentiation, rebuff envelopment threats from the platform owner, and leverage network effects that create noncoercive lock-ins. Decentralization of app strategic decision rights also works well from the platform owner's perspective because it fosters "combinatorial" innovation around the ecosystem by mixing ideas from outside and inside its own organization.[2] Decentralization of app strategic decision rights also minimizes the burden on the platform owner for coping with the potentially large variability in evolution rates of the platform's app portfolio.

However, platform strategic decision rights and app implementation decision rights are not as straightforward to align with knowledge because they draw on a distinct core body of knowledge but also require a complementary body of knowledge that is usually not colocated with the same party in an ecosystem (Hoetker, 2005; Kapoor and Adner, 2011; Tiwana and Keil, 2007).

Consider *platform strategic decision rights*. These represent the authority for direction-setting decisions about what a platform should do. The core knowledge needed is an understanding of the platform's target markets, including an appreciation of rival platforms, industry needs, trends, and cost structures. The locus of such core knowledge is likely the platform owner. Therefore, on first glance, it makes sense to centralize platform strategic decision rights. Centralizing platform strategic decision rights also gives the platform owner the power to strategically maintain selective incompatibility that locks out rival platforms and (coercively) locks in app developers to the platform (Rysman, 2009). When the platform owner retains strategic decision rights for a platform, it also retains the power to alter the rights and privileges of app developers and set contractual obligations and rules of participation (Boudreau, 2010). This gives a platform owner the flexibility to tweak the degree of openness of the platform over time.

However, simply centralizing these decision rights with the platform owner runs the risk of overlooking a critical type of complementary knowledge that is likely to be a platform owner's weakness: deep knowledge of user needs. Recall that by definition a platform is at least two-sided. These two sides—app developers and end-users—are both users of the platforms.

[2]When a platform owner holds app strategic decision rights (i.e., they are centralized), the structure begins to resemble traditional software outsourcing. This is a viable decision rights structure only when the platform owner wants to dictate *what* an app developer should accomplish. It also alters the appropriate structure of control portfolios described in the next section, which must then emphasize output metrics set by the platform owner over any other control mechanism.

App developers must have some input in platform strategic decisions because they are likely to be able to contribute two distinct types of knowledge that are needed by the platform owner for making direction-setting decisions. First, app developers are likely to understand their own needs for their app development work around a platform better than the platform owner. External stakeholders such as app developers who were never in the picture in traditional organizations must therefore be given the opportunity to provide real input into platform decisions (de Weck et al., 2011, p. 19). Input from app developers allows a platform owner to appropriately evolve the interfaces to its platform to better meet app developers' evolving needs, while simultaneously protecting and selectively disclosing intellectual property that is core to the platform. By itself, control over the architecture of a platform gained by centralizing platform implementation decision rights does not guarantee that the platform will be able to sustain a win–win, pie-expanding proposition with app developers. A platform will survive only if it helps everyone in its ecosystem do better with it than without it. In order for a platform to thrive, platform owners must not only attract but also retain app developers. A platform owner must therefore continue to contribute unique capabilities valued by app developers that compare favorably with rival platforms, and capabilities that have no direct substitutes. Allowing app developers to have input in platform strategic decisions therefore enables a platform owner to remain sensitized to their emerging needs while also delivering the necessary economies of scale in the functionality shared by their apps.

Second, app developers are also likely to be closer to the pulse of emerging end-user needs from their specialized market segments of their own apps. End-user expectations of tomorrow's mass market end-users often stem from today's leading-edge user needs (Von Hippel, 1986). App developers are more likely to be better tuned to the emerging needs of such leading-edge users, who are typically found in niche segments of the broader end-user communities. In contrast, the platform owner's knowledge of end-user needs is likely to be more mass-market-oriented. Therefore platform strategic decision rights should lean toward centralization with platform owners but with app developers' inputs. In other words, they are optimally placed toward the centralization side of the scale but away from extreme centralization. This is illustrated in the range of possible placements for this class of decision rights in Figure 6.14. Where precisely within this range they are placed depends on the modularity of a platform's architecture. (When a platform owner holds app strategic decision rights, the platform model begins to resemble a traditional outsourcing arrangement.)

Finally, *app implementation decision rights* represent how app developers execute and realize app strategic decisions. The first core knowledge needed for app implementation decisions is likely to be with

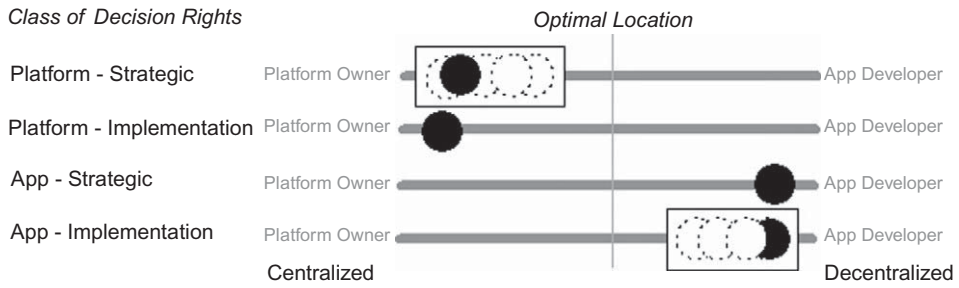

FIGURE 6.14

Optimal decision rights partitioning in modularized platform ecosystems.

app developers. The first advantage of greater decentralization of app implementation decision rights is that it places the authority over app implementation decisions where the core knowledge to make such decisions resides. Second, it frees an app's developer to upgrade and evolve an app unconstrained by the platform. Provided an app developer sufficiently modularizes an app's microarchitecture, this reinforces the potential advantage of the flexibility provided by architectural encapsulation (wherein the app displays no hidden internal details beyond its visible information to the rest of the ecosystem). A third advantage is that control over app implementation decisions provides an app developer the option to multihome (i.e., to potentially make the app portable across multiple competing platforms). Portability means that an app can be readily executed on a different platform from the one for which it was initially designed. For example, an app such as Skype can readily be run on iOS, Android, or Blackberry OS. Portability is attractive to app developers and end-users, but not usually to platform owners.

However, simply decentralizing these decision rights with app developers runs the risk of overlooking a critical type of complementary knowledge that is likely to be app developers' weakness. Implementation of apps requires some knowledge of the platform technologies, especially to ensure that an app leverages the unique capabilities of a platform when appropriate, and to ensure interoperability with the platform. Therefore, such decisions should lean toward decentralization with app developers but with some input from platform owners. In other words, they are optimally placed toward the decentralization side of the scale but away from extreme decentralization. This is illustrated in the range of possible placements for this class of decision rights in Figure 6.14. Where precisely within this range they are placed depends on the modularity of an app's microarchitecture and how the four functional elements of the app are distributed across client- and server-side devices. The decision rights partition framework leaves unaddressed the question of who—platform owner or app developer—actually controls the relationship with an individual app's end-user. The answer depends on the microarchitecture for an individual app chosen by the app developer.

Figure 6.14 summarizes the resulting optimal pattern of decision rights partitioning in a modularized platform ecosystem. The more modular the platform's architecture, the more centralized should platform decisions be and the more decentralized should app decisions be. However, platform strategic decision rights can be anywhere in a range of levels leaning toward centralization but with some app developer say in them. However, app implementation decision rights can be anywhere in a range of levels leaning toward decentralization but with some platform owner say in them.

6.3.2 Aligning control portfolios

6.3.2.1 The dual purpose of control in platforms

The purpose of control mechanisms in an ecosystem is twofold: creating convergent goals and ensuring coordination between the platform owner and app developers.

Creating convergent goals. The first purpose of platform control is to ensure that the work of app developers furthers the interests and objectives of the platform. At the very least, it must not hinder them. App developers and platform owners are usually independent organizations that are likely to pursue their own self-interest, even if it is at the expense of the other party (Eisenhardt, 1989). Goal convergence can partially be accomplished by careful platform pricing choices that align the goals of app developers and platform owners. For example, pricing structures that split revenues create a shared fate that can bind together the interests of platform owners and app developers. But revenue-sharing models are not always viable in platforms (e.g., in open-source, nonprofit platforms). An alternative

way to create convergent goals is through relational controls that rely on a shared sense of purpose, norms, and values that bind together the interests of a platform owner and app developers. Control mechanisms must also vet and weed out apps that are potentially damaging to the ecosystem. These can include outright malicious apps intended to harm the platform's end-users or apps that are designed to compromise or circumvent a platform's business model and rules.

Facilitating coordination. The second purpose of platform control is to facilitate coordination between the platform owner and app developers, primarily ensuring integration between the platform and apps. The potential power of platform ecosystems comes from leveraging the unique expertise of many, diverse independent app developers on a scale irreplicable within a single organization. However, this diversity can also be the root of utter chaos. Apps must seamlessly interoperate and integrate with the platform to ensure a cohesive platform ecosystem. This property—composability—is the degree to which an app can be readily integrated with the platform. After all, ecosystem-level innovation does not arise from the mere gathering—but from new *combinations*—of many app developers' capabilities (Dhanaraj and Parkhe, 2006).

We have already described two strategies in platforms that help facilitate such coordination: (1) an appropriate choice of architecture by the platform owner and app developers and (2) mirroring decision rights partitioning and architecture. However, even these two devices taken together leave one hole in ensuring coordination across the ecosystem, which a control portfolio must plug: variance in interface standards compliance. The implicit coordination capability of architecture works only if every developer is following agreed-upon standards for connecting their apps to the platform. Compliance by app developers to a modularized platform's interface specifications ensures integration of apps with the platform.

The challenge comes from variance in the extent to which different app developers comply with the interface standards. A roadblock in ensuring systems integration is the *repeated* need to ensure it. Different apps often evolve at different rates, and often evolve faster than the platform does. So, the need for ensuring seamless integration of apps with the platform is hardly a predictable, one-shot activity but rather an ongoing one. As an app evolves, it must maintain interoperability across each iterative release. Therefore, it is useful to think in terms of systems *re*integration rather than one-shot systems integration between a platform and apps. Mirroring modularized architecture with modularized decision rights does not eliminate the need for integration processes.

Paradoxically, this means that modular ecosystems organized around modular architectures require monolithic—not modular—*integration* processes (Brusoni and Prencipe, 2006). This is illustrated in Figure 6.15. By integration processes here we are referring to the processes used to integrate the outputs of app developers' work, not how they develop apps. (The latter are app implementation decision rights.) This ensures that app developers can be locally optimizing but globally aware. Freedom in app developers' work must therefore be coupled with strict specifications of contributions, expectations of performance, and appropriate rewards and penalties. This requires some degree of formal control by the platform owner over app developers. Control mechanisms can help address this challenge by ensuring better compliance to a platform's interface specifications and design rules.

This brings us back to the Goldilocks principle: Platform owners must get the control portfolio *just* right. The challenge for platform owners is to manage the delicate tension between developer autonomy and ecosystem-wide integration. Control portfolios must simultaneously support the entrepreneurially oriented independence of app developers and seamless integration of their output into the platform's ecosystem. Too much control can stifle innovation by app developers. Too little control can compromise ecosystem-wide coherence (Boudreau, 2010). Granting access by opening a platform

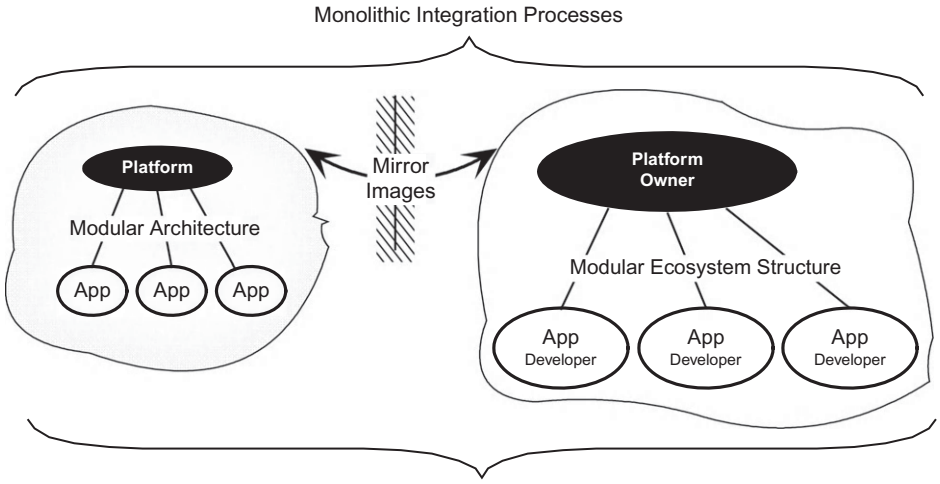

FIGURE 6.15

Modular ecosystems organized around modular architectures still require monolithic integration processes.

therefore does not entail giving up control. In going overboard with giving up control, a platform owner can position itself out of the ecosystem altogether. (Think of IBM and the PC business that it literally created.) A 16-year study of platforms found evidence that giving up control did not increase innovation in platforms beyond an incrementally trivial level; instead it put them on a devolving path to failure (Boudreau, 2010). Therefore, a platform ecosystem that thrives will neither be like a democracy nor a centrally planned regime. Instead, it must be like a benevolent dictatorship (Tiwana et al., 2010).

6.3.2.2 Rules of thumb for designing a platform control portfolio

Platform-based businesses represent a shift from closed innovation models to open innovation models. The closed innovation model—the mainstay of traditional organizations—was based on the philosophy that *innovation requires control*: Companies generated their own ideas, developed and implemented them, and then took them to market. Self-reliance was the prized virtue, and the key task of managers was to screen out bad ideas. This model dominated most of the past century. This internally focused approach to innovation is increasingly becoming obsolete as traditional organizational structures are being supplanted or altogether replaced by distributed innovation networks such as platform ecosystems. The platform model is based on the philosophy that *innovation requires giving up some control*. Five rules of thumb should guide the design of a platform's control portfolio to accomplish these goals.

1. *Simple*. The first rule of thumb is that a control portfolio should be as simple as possible in its structure. Simply put, less is more. Controls impose costs on the platform owner who must design, implement, and most importantly enforce control mechanisms (Anderson and Dekker, 2005). These costs must be commensurate with their realized benefits. A control mechanism that cannot be reliably enforced is likely to deliver little benefit but will nevertheless impose costs. Controls

also impose compliance costs on app developers. These costs are not only direct financial costs but also indirect costs such as discouraging frequent releases of apps. This can slow down—even cripple—the evolution of individual apps, create a destructive pattern of missed windows of opportunity, and increase time lag between updates. Such costs are borne immediately by app developers, but their damage to the vibrancy of a platform can easily exceed the damage they can do to individual app developers. Rapid iterative refinement is usually critical in highly competitive app market segments. If controls stymie the speed of such refinements in apps, they can amplify the destructive power of the Red Queen effect that we discussed in Chapter 2. A useful heuristic is therefore to structure a control portfolio that is least costly for both app developers and for the platform owner. Remember that architecture itself is a form of control. By defining the architectural rules for app developers, a platform owner invokes a source of real power in a platform ecosystem (Meadows, 2008, p. 158). Control though architecture can substitute for many of the costlier, overt control mechanisms. But overt control is effective only in extreme moderation. Therefore, the minimal subset that gets the job done (i.e., ensures that the app developers are working in the best interests of the platform and ensures easy integration of their work into the ecosystem) is optimal. Instead of asking themselves how much of each control mechanism to use, it is more fruitful to ask the question of whether a specific control mechanism is even necessary. If it is, can it be realized in practice? Can it be enforced? If a control mechanism is absolutely necessary, it is the platform owner's responsibility of minimizing the cost imposed by it on app developers.

2. *Transparent*. The second rule of thumb is that a control portfolio should be transparent to app developers. Compliance with a set of controls imposed by the platform owner is more likely to happen if app developers clearly understand precisely how their work will be evaluated. Ambiguity about this (as was the case with Apple's iOS) can make compliance so cryptic that it can discourage existing app developers from sticking with a platform and dissuade new app developers from joining the platform. To have high transparency, a platform owner can take two steps. First, be explicit about expectations and how performance on specific criteria related to how those expectations will be measured. Second, make the process of evaluation on such criteria visible to app developers.
3. *Realistic*. The third rule of thumb is that a control portfolio should be based on a good understanding of how the app developers' work is done. The platform owner must therefore consider whether each control mechanism chosen for the control portfolio is a realistic reflection of app developers' day-to-day routines and practices. To the extent possible, a platform owner must seek to establish guidelines—not rulebooks—for app developers and also make a few nonnegotiable principles explicit.
4. *Shared values*. The fourth rule of thumb is that a control portfolio must conform to the platform owner's philosophy about the platform. This philosophy is effective only if it is also shared by other members of the ecosystem, particularly app developers. A well-designed control portfolio should reinforce this philosophy and at the very minimum not contradict it. (An important control mechanism that can promulgate shared values, norms, and a shared culture among app developers and the platform owner is relational control.)
5. *Fair*. The final rule of thumb is that a control portfolio should be fair. This means that it should be consistently applied to all app developers, it should have no contradicting rules, and must be fairly applied and interpreted by the platform owner. Fairness also includes fairness to the

platform's other sides such as the end-users. An important role that controls must therefore also serve effectively is to prevent members of either side to take unfair advantage of the other (Evans and Schmalensee, 2007, p. 35).

6.3.2.3 Aligning individual control mechanism choices

Platform owners must ask three questions to pick which of the four control mechanisms to deploy and to what degree:

1. Is it needed? If not, skip it.
2. Whether something else can substitute for it? If yes, use the substitute instead.
3. Is it viable? Viability requires that the preconditions in Table 6.1 for using it are met. If not, skip it.

The forthcoming logic is summarized in Table 6.4.

Table 6.4 Evaluating Individual Control Mechanisms for Inclusion and Exclusion

Control Mechanism	Needed?	Substitute?	Viable if...
Gatekeeping	When creating performance metrics or monitoring processes is not possible	None; but prescreening app developers helps	• App developers accept a platform owner's authority to play gatekeeper • Compliance criteria are known and considered fair by app developers • Platform owner can cost-effectively verify compliance
Process	Not needed if performance metrics are used	• Gatekeeping • Use of metrics-based control • Allocation of app implementation decision rights to app developers	• Platform owner has credible expertise to dictate methods • Platform owner can verify process compliance by app developers
Metrics	Not needed if app developers retain app strategic decision rights	• Use of process control • If market determines winners and losers among app developers	Metrics are objectively measurable
Relational	• Fills gaps left by formal controls • Lower cost than formal controls	None, but prescreening app developers helps	• App developer churn is low • App developers and platform owners are bound by clan-like shared values

6.3.2.4 Aligning gatekeeping with platform architecture

Gatekeeping is required when a platform owner can neither reliably use predefined performance metrics nor cost-effectively impose or monitor process compliance by app developers. Both of these challenges often exist in platforms, leaving platform owners with no good substitutes for gatekeeping. But gatekeeping requires that criteria for their app being allowed into the platform ecosystem is explicitly known and considered fair by app developers and that the platform owner can cost-effectively verify compliance. The more visible—but less important—role of gatekeeping is to keep ill-intentioned apps out. The more important role of gatekeeping is to ensure the integrity of apps and their compliance with the design rules, constraints, and interface standards to proactively ensure interoperability with the platform. Recall that modularization of platform architecture facilitates app integration with the platform only if individual apps comply with the platform's interface specifications and design rules. Therefore, the primary purpose of gatekeeping is ensuring app developer compliance.

However, testing costs are the Achilles heel of modular architectures (Baldwin and Clark, 2000, p. 272). Gatekeeping can become very costly and time-consuming for a platform owner as the frequency of updates to individual apps increases and the size of the app developer pool grows. Over time, this can become enough of a bottleneck that a platform can fall behind in the competitive race against rival platforms. Therefore, a platform owner has a strong incentive to invest in tools and mechanisms that lower app developers' costs and its own costs of verifying compliance of apps with the platform's design rules and conformity with its critical interface specifications. Developer toolkits, reference models, integrated development environments, and app testing tools are examples of such tools (Evans et al., 2006, p. 413; Parker and Van Alstyne, 2005). Two additional ways to reduce gatekeeping costs upfront include (1) prescreening who is allowed to join the platform (a form of relational control) and (2) being clear about constraints on what apps are *not* allowed to do. An example of the latter is Apple's prohibition on apps duplicating the native functionality of the iOS platform. In summary, gatekeeping has almost no direct substitutes and is essential for fully extracting value out of modular designs.

6.3.2.5 Aligning process control with platform architecture

Process control is required when the platform owner either (a) cannot prespecify objective metrics to evaluate app developers' outputs or (2) has a better understanding than app developers of how to successfully develop apps. The first condition is often met in platform settings but the second one holds true only occasionally. The platform owner can rarely dictate to the developers how to do their day-to-day app development work because it lacks the legitimate authority to issue commands and independent app developers are not obligated to obey (Dhanaraj and Parkhe, 2006). Nor can it inexpensively monitor or micromanage how app developers do their own development work. Decentralization of app implementation decision rights to app developers also contradicts the use of process control because it translates into granting autonomy to app developers for how to implement their apps.

However, a platform owner often *can* credibly prescribe development and testing procedures that will ensure app interoperability with its platform. Modular platform interfaces themselves provide some app integration mechanisms that reduce the need for ongoing coordination between a platform owner and app developers. Furthermore, use of gatekeeping by the platform owner also reduces the need for extensive process control. This leaves only one potential use for process control: to help app developers pass gatekeeping checks. Therefore, *some* process control can increase the value of extensive gatekeeping in ensuring apps' compliance with a platform's interface specifications and design rules.

A platform owner can invest in four things to facilitate such compliance using noncoercive, facilitative process control:

- *Programming resources*. Examples of such resources include documentation such as technical specification, manuals, programmers' guides, and programming tools used to write apps. Apple's investments in such resources for its iOS developers have paid back handsomely (Burrows, 2011).
- *Integrated development environments* (IDEs), software development toolkits (SDKs), and reference models are other common ways for platform owners to assist app developers with compliance to a platform's interfaces (Evans et al., 2006, p. 79; Parker and Van Alstyne, 2005). They automate rote tasks such as tracking changes, managing source code versions, and testing of apps using simulators. Providing such tools can help improve app developer productivity and better manage complexity associated with developing for a platform. The platform owner should rarely charge for such app developer tools. The focus should be on enabling everyday development processes and helping improve app developer productivity. The objective is to reduce the cost and effort incurred by app developers in developing around the platform.
- *Mock-up and prototyping tools*. These are tools that allow app developers to inexpensively create prototypes of what a finished app would look like before having to implement it in code. The purpose of such tools is what Michael Schrage (2000, p. 126) calls demand articulation: End-users cannot readily tell app developers what app features that they need but recognize them when they see them. This allows app developers to quickly refine app design concepts before they make irreversible investments in implementing apps that eventually turn out to be duds. A platform owner providing such tools also reinforces the decentralization of app strategic decision rights to app developers in modularized platform ecosystems. This subsequently has much larger payoffs in a platform's evolution and survival in Red Queen competition against rival platforms.
- *Integration protocols and testing standards*. These are procedures that allow the app developers to proactively assess how well an app conforms to a platform's rules and interface specifications (Baldwin and Clark, 2000, p. 77). The earlier in the app development process an app developer can test standalone apps for compliance, the cheaper it will be to fix potential problems (Hibbs et al., 2009).

6.3.2.6 Aligning metrics-based control with platform architecture

Control via metrics is rarely required in platform ecosystems for two reasons. First, when app strategic decision rights are decentralized, app developers—rather than platform owners—set most criteria and metrics for judging the performance of their apps. However, metrics-based control requires platform owners to set predetermined metrics for evaluating the app developers' performance. Objective measurement of app developers' performance using metrics, even if possible, is meaningless when the platform owner does not set them. (Recall from Section 6.3.2.3 that this information-intensive, costly control mechanism is a legacy from the traditional software development model.) Second, competitive markets determine winners and losers among apps in platform ecosystems. End-user markets therefore provide high-powered incentives for app developers to create apps that do well in the marketplace. These incentives, coupled with their own investments in their apps, suffice to motivate app developers to use their distinctive capabilities and to leverage their unique expertise to ensure that their apps compete well in meeting the needs of their end-users. Third, process control and metrics-based control do not mix well (Tiwana and Keil, 2007). If process control is used by a platform owner—as is often done

in platform ecosystems—it provides a direct substitute for metrics-driven control. Therefore, control via metrics is rarely needed[3] or viable, and often has less costly substitutes in modular[4] platform markets. Only a small set of objective performance metrics such as sales or usage is sufficient to implement the revenue-sharing agreement between the platform owner and individual app developers.

6.3.2.7 *Aligning relational control with platform architecture*

Relational control fills gaps left by formal controls, especially in dealing with outside-the-contract situations not covered by the formal agreements between the platform owner and an app developer (Bernheim and Whinston, 1998; Tadelis, 2002). Relational control is among the least costly control mechanisms. However, it is difficult to implement right off the bat because shared values, norms of behavior, peer pressure, and the shared mindset among the app developer community—especially a globally distributed one with markedly different values and norms—that it relies on take time to emerge. The app developer community must also be stable enough over time for it to be viable. Platforms often have developer churn, with new developers lacking the shared history of existing developers. The potential diversity of app developers, the churn in membership, and the youth of the platform can therefore decrease the viability of relational control.

Relational control will therefore rarely suffice by itself but can be a useful complement to the other formal control mechanisms. The platform owner can help foster a clan-like culture by promulgating a sense of shared values, norms, sense of purpose, and mindset among its developer community by (1) setting examples through its own actions (e.g., fairness, impartiality, design ethos), (2) reinforcing a common identity among members of the ecosystem (Dhanaraj and Parkhe, 2006), and (3) organizing socialization opportunities among its app developers (e.g., developer conferences). Prescreening who is allowed to join the app developer community is another way to accelerate the development of shared norms. Prescreening might use criteria such as prior history of the developer on other platforms to ensure that members of the app developer community meaningfully complement the platform owner's own capabilities.

The litmus test for an ideal control portfolio is one that is simple, transparent, fair, and realistic. The ideal mix of controls is to use one formal control mechanism as a dominant form of control and supplement it with relational control if possible. When relational control is not viable, use gatekeeping with either process control or metrics-based control.

6.3.3 Aligning platform pricing policies

Pricing decisions by platform owners are key to creating incentives that can encourage a critical mass of app developers to create platform complements that can keep a platform ahead of rival platforms in Red Queen competition. The five pricing decisions in platforms must be aligned with the architecture, platform lifecycle stage, and the platform's business model, as summarized in Table 6.5.

[3]The only rare exception is when app strategic decision rights are centralized (i.e., the platform owner rather than app developer makes direction-setting decisions for an app). In this case, the platform model begins to resemble a traditional outsourcing arrangement where metrics-based control should largely replace process control and gatekeeping.
[4]In contrast, if the platform architecture were monolithic and output metrics were unavailable, development of the app would have to be done by the platform owner.

Table 6.5 Considerations in Aligning Pricing Policies

Pricing Decision	Business Model	Lifecycle	Architecture
Symmetry	●	●	
Subsidy-side	●	●	
Access/usage fees		●	●
Sliding scale?	●	●	●
App pricing model			◓

6.3.3.1 Aligning the pricing symmetry decision

The first criterion in deciding whether to price the two sides asymmetrically is the platform's business model. If a platform is two-sided *from the beginning*, getting both sides on board is critical to getting it off the ground. Recall that most successful multisided platforms started out as one-sided services that added a second side only after a critical mass of adopters was on board on the first side. For example, Dropbox, the popular file-sharing service, added app developers (the second side) only after it had a large end-user base (the first side; initially the only side).

In two-sided platforms, one side is often the loss leader and the other side is the profit center. Money lost on the money-losing side is usually made up on the money-making side. Only in rare cases do platforms make money on both sides; this is often when it starts off as a dominant early mover that started out as a successful product (e.g., iPhone) or service (e.g., Dropbox and YouTube) with a lot of adopters. Platform pricing therefore often must be asymmetric such that either the app developers *or* the end-users are the side that pays a lot less than the other side.

A second consideration is whether the platform's business model depends on cross-side network effects for its success. If so, asymmetric pricing can accelerate the creation of cross-side network effects in the early stages of its lifecycle; if enough subsidy-side users are attracted to a platform, the money-side users will pay a premium to reach them (Eisenmann et al., 2006). An example of this is Amazon's Kindle platform. Amazon subsidized end-users (by selling devices based on the platform at a loss) to create a large potential reader base. This attracted major book publishers to the Kindle platform, jumpstarting cross-side network effects. The initial subsidies can then be reduced, possibly even eliminated. This approach, however, can fall flat if there are no early-mover advantages in the platform's market.

Being a first mover to sell in a product or service category does not necessarily guarantee success. A first mover can be riddled by an underdeveloped or immature pipeline of adopters on the subsidized side and unclear market requirements, which can make being an *early follower* a more attractive proposition for entering a potential platform market. The ability to create switching costs[5] among the subsidized-side adopters, exploiting scale and increasing returns advantages and possibly network effects is necessary for first-mover advantage to be plausible. Dropbox is an example of a platform

[5]Switching costs arise when the initial investment in complementary products such as purchased apps, going up the learning curve for a platform, and the whole hassle of replicating her setup on a rival platform can discourage the subsidized-side user from leaving the platform for another one. Learning costs and familiarity with the QWERTY keyboard, for example, kept users from switching to a technically superior Dvorak keyboard. The same logic often applies in platforms.

that inspired many copycats attempting to replicate its offering. Such copycats who enter the market after the product or service has begun to penetrate the mass market are known as late entrants. They were able to replicate Dropbox's offering but were unable to overcome the strong same-side network effects that Dropbox had used to create a first-mover advantage.

6.3.3.2 Aligning the choice and duration of the subsidized side

If a platform owner decides to subsidize one side, which one should it be and for how long? The first criterion in deciding which side to subsidize is the platform's business model, particularly if it depends on cross-side network effects. The trick is to subsidize the more price-sensitive side and charge the side whose demand increases more strongly with growth on the subsidized side. A platform owner should charge the lowest prices to the side it needs most to get cross-side network effects started. Put another way, charge more to the side that derives more value from the presence of the other side. Even if a platform owner makes a loss on one side, it can recoup the losses from the other side provided the demand on the other side is sufficiently strong.

Consider two examples. Adobe's portable document format (PDF) did not catch on until Adobe priced the PDF reader at zero. Subsidizing end-users substantially increased sales of PDF writers, from which Adobe earned all of its revenues. Now, think of a trade press magazine as another example. If readers of a magazine value the number of ads less than advertisers value the number of readers, then magazine publishers should do better to subsidize readers relative to the advertisers (Armstrong and Wright, 2007). This approach works well for many trade magazines that are free to readers but that charge hefty fees to advertisers. Therefore, as a general rule, use prices that lead to zero (or negative profits) from the side that is more valuable to the other and has greater ambiguity about the value they will derive from adopting the platform (Evans and Schmalensee, 2007, p. 84). However, such subsidies are needed only until critical mass is achieved by a platform on both sides. Achieving critical mass triggers a self-reinforcing bandwagon effect because the value of a platform to either side increases approximately proportional to the square of the number of users (Rohfls, 2003, p. 55). Subsidies can be reduced after critical mass is reached, although it is unadvisable to completely eliminate them.

The question for platform owners to ask themselves is whether app developers value access to end-users more, or end-users value apps more. Will subsidizing app developers increase end-user demand for the platform by increasing the number of apps they can use, particularly in comparison to rival platforms? If so, subsidize the app-developer side. This is the case with many smartphone platforms. In such markets, availability and variety of apps is decisive in end-user platform choice. Superstar apps and so-called killer apps that end-users value highly can have a disproportionately large effect on end-user platform adoption; their developers are often prime candidates for subsidies. Conversely, will subsidizing end-users increase the demand from app developers by increasing the prospective pool of willing buyers of their apps for that platform? If so, subsidize the end-user side. This is the case with many gaming console platforms. The same logic can be used by app developers when an app attempts to grow into a nested, mini platform (see Chapter 11).

The second criterion is the platform's stage in its lifecycle. Two considerations enter this decision: (1) whether a platform is in its pre- or post-dominant design stage and (2) its diffusion among end-users. In the pre-dominant design phase, app developers are usually stronger candidates for subsidizing because a winning industry-wide design is yet to emerge. At this stage, the platform's market can

be a winner-takes-all kind of competition. Therefore, rapid adoption of a platform by app developers can boost the rate of innovation, increase end-users' perceptions of its usefulness, and accelerate its mass adoption. Together, this can increase the likelihood of a platform becoming *the* dominant design that rivals will eventually be forced to follow. Negative pricing can therefore become optimal when a platform owner's viability (and profits) are contingent on promoting network effects (Parker and Van Alstyne, 2005). Second, same-side network effects are difficult to initiate in the early stages of diffusion among end-users but are self-reinforcing once they take off. Having app developers on board shapes end-users' expectations about the platform's future; such expectations heavily influence their present adoption decisions (Rysman, 2009). A useful metric for tracking progress in diffusion among end-users is a platform's installed base (Rochet and Tirole, 2003). Real-time information about this metric can readily be collected in ubiquitously networked software-based platforms. Subsidies might be reducible for the subsidized side once a platform reaches the early-majority stage in its diffusion among end-users.

6.3.3.3 Aligning usage and access pricing

The first criterion in deciding whether to charge access and usage fees is the platform's stage in its lifecycle. A platform owner can theoretically charge all sides (e.g., app developers, end-users, advertisers) platform access fees (typically fixed upfront fees) and usage fees (typically variable fees), or both (Hagui, 2006). Or platform owners can choose to charge neither to two of their sides (e.g., app developers and end-users) and instead make up for the loss from a third side (e.g., advertisers). The correct choice of access and usage policies and fees depends on the platform's business model, its industry, and the norms among direct and indirect rivals. Indirect rivals can be particularly tricky to recognize for platforms and apps that are attempting to create new "blue ocean" markets. For example, iOS's competitors were dumb phones and Blackberries but Android OS's competitor was iOS. Similarly, Dropbox's competition was the inexpensive USB thumb drive, not other file-sharing services. Therefore, the choice of the primary fee model is one that does not have easy answers. However, two general rules usually should be followed. As a first general rule, platform owners must keep access fees low to encourage prospective adopters on all sides to try the platform. But access fees for the subsidized side should *not* be zero for an established platform but can be zero or negative in fledgling platforms. Willingness to pay a token access fee can be just enough to signal credible commitment and seriousness from app developers. Apple, for example, charges its iOS app developers a token $99 annual fee for access to the platform. For fledgling platforms yet to attract a critical mass of app developers, access fees can even be negative. Blackberry, for example, guaranteed to underwrite a minimum revenue for $10,000 for new apps on its BBOS 10 platform, effectively *paying* app developers to join. Poor pricing decisions about access fees for upstart platforms can keep them from building the right mix or critical mass of participants from one or both sides, leading to failure even before they've had a chance to take off. As a second general rule, a platform should charge either access fees or usage fees to individual sides, but not both. However, access and usage fees can be mixed within a platform but not on the same side of the platform.[6]

[6]For example, it is possible for a platform to charge access fees to app developers, nothing to end-users, and usage fees to advertisers.

The second alignment criterion is app microarchitecture. A platform owner should charge usage fees to at least one side—either end-users or app developers—if (1) an app's microarchitecture *heavily* uses native services of the platform and (2) such services lack scalability. This permits recouping expenses from the usage of apps that hog platform resources and scale-limited services. But this imposes a considerable overhead of metering usage by individual apps on the platform owner.[7] However, having to resort to such pricing policies is often symptomatic of two larger problems. First, it might be a red flag that the platform's architecture itself lacks sufficient scalability, often a sign of larger looming problems as the platform grows. Second, app microarchitecture is a decision largely made by app developers but it is heavily influenced by platform architecture. Having to charge fees for platform services could *potentially* signify weaknesses in modularization of the platform's architecture.

6.3.3.4 Aligning the pie-splitting scale choice?

The first criterion in deciding how to split the revenue pie with app developers is the platform's business model. To remain attractive, a platform must allow app developers to profit sufficiently from their work (Gawer and Cusumano, 2008). Pie-splitting choices by a platform signal are also an important credible signal to app developers that a platform owner will not abuse its power and that they share a common destiny. Although fixed scales are most common in contemporary platforms (e.g., Apple splits revenues using a 30–70% fixed scale), moving scales can serve a useful role in various stages of a platform's lifecycle. A rising scale model is fairly common for author royalties in the traditional book publishing industry. If used in a platform ecosystem, rising scales create stronger rewards for apps that perform better in the marketplace. This can create strong incentives for app developers to study their own niche markets more intensively, invest in understanding user needs more closely, and evolve their apps. When app developers maintain presence on more than one rival platform and when platform owners cannot dissuade or restrict such multihoming, rising scales can also encourage them to invest more heavily in developing their app for that platform. This subtle strategy to deter multihoming is known as *steering* (Rochet and Tirole, 2003).

The platform's lifecycle stage provides a second criterion for aligning pie-splitting choices. The earlier a platform is in its lifecycle, the more intense is the competition for app developers. Incentives matter even more in such early stages because app developers are potentially more mobile and can move to a competing platform. The vibrancy of a platform ecosystem then critically hinges on attracting new app developers and retaining existing ones. By using a rising scale in a platform's early lifecycle stages, a platform owner can offer stronger incentives than rivals to app developers. In the long run, app developers are more likely to opt for a smaller piece of a bigger pie than a smaller pie.

Architecture provides a third criterion for aligning pie-splitting choices. If an app intensively uses the platform's native services in its microarchitecture and the platform has scalability limitations (or scaling is costly for the platform owner), a sliding scale is a viable option. Platform owners should generally refrain from using sliding scales.

[7]An alternative is to insist that app developers use multi-tiered app microarchitecture as an alternative capacity scaling strategy.

6.3.3.5 Decision 5: app licensing decisions

The choice of perpetual, subscription-based, or usage-based licensing by an app developer for an app largely depends on the *app developer's* business model. A platform's architecture and portfolio of software services, however, can constrain viable business models that app developers can implement. For example, integration of advertising services into iOS and Android platforms makes it more viable for an app developer to create advertising-supported apps rather than being limited to revenue-generating apps. Furthermore, app microarchitectures play a direct role in how easy it is to create variants of a single app. These variants are often priced differently. This strategy (called app *versioning*) means different customers pay different prices for more or less capable variants of an app. However, the possible choices of app microarchitecture available to app developers are also constrained by platform architecture. Therefore, the connection between app licensing and platform architectures is at best indirect. The app licensing decision has consequences for versioning of apps by app developers. We describe in detail several strategies for versioning in our discussion of app evolution in Part IV.

Table 6.6 summarizes the foregoing discussion of how each of the five pricing choices can be better aligned with a platform's business model, its lifecycle, and its architecture.

Platform governance decisions by platform owners impact several of the ecosystem evolution principles over the life of a platform. Table 6.7 provides a preview of these. The next section explains these ideas in depth.

Table 6.6 Summary of Considerations in Aligning Platform Pricing

Pricing Decision	Business Model	Lifecycle	Architecture
Pricing symmetry?	Asymmetric if two-sided from outset and dependent on cross-side network effects	Asymmetric if first mover advantage can be secured through network effects or switching costs	–
Subsidized-side?	• If one side values the other more • "Superstar" apps	• Dominant design emerged? • Diffusion among end-users?	–
Access fees?	• Generally avoid • Token access fee from app developers to signal credible commitment	Negative or zero access in early but nonzero in later lifecycle stages	Usage fees for native services-intensive app microarchitectures
Moving pie-splitting scale?	Rising scale if app developers multihome rival platforms	Rising scale if intense cross-platform rivalry	Sliding scale if low platform scalability
App pricing model?	–	–	Viable app licensing models are constrained by platform architecture

Table 6.7 Consequences of Governance Choices in Platform Ecosystems

Principle Affected	Governance Dimension		
	Decision Rights	**Control**	**Pricing**
Red Queen effect	●	◓	◓
Chicken-or-egg problem			●
Penguin problem			●
Emergence	●	●	
Seesaw problem		●	
Humpty Dumpty problem		●	
Mirroring principle	●		
Coevolution	●	●	
Goldilocks rule			●

CHAPTER SUMMARY

- *Governance is how a platform owner influences its ecosystem.* App developers are not soldiers in an army but rather like musicians in a symphony. The role of governance is to coherently orchestrate the integration of their unique contributions into a platform's ecosystem. Good platform governance must respect app developers' autonomy while ensuring ecosystem-wide integration.
- *Governance complements architecture.* Platform governance determines whether thoughtful architecture pays off. The two must be aligned.
- *Governance has three dimensions.* These are (1) who decides what (decision rights), (2) how a platform owner controls app developers (control mechanisms), and (3) pricing policies.
- *Governance must aspire to be simple and cheap.* The optimal governance structure is the simplest one that achieves the goals of a platform at the least cost.
- *Decision rights are division of authority among a platform owner and app developers.* The authority and responsibility for four classes of decisions can be split any way between a platform owner and app developers: strategic and implementation decisions about the platform, and strategic and implementation decisions about individual apps. Strategic decisions are decisions about what it should do and implementation decisions are about how it should do it. Centralization and decentralization of decision rights refer to whether they lean toward the platform owner or app developers.
- *Control is how a platform owner creates goal convergence and facilitates coordination with app developers.* A portfolio of control mechanisms used by a platform owner can mix-and-match different levels of formal (gatekeeping, metrics, and process control) and informal (relational) control mechanisms.
- *Pricing policies involve five decisions.* These involve decisions about whether app developers or end-users are subsidized by the other side, for how long, whether a platform owner charges access or usage fees, the pie-splitting structure, and app licensing choices.

- *Aligning decision rights follows the mirroring principle*. The partitioning of decision rights among a platform owner and app developers must mirror the platform's architecture. Decentralization of design in platform architecture must therefore be mirrored in the decentralization of authority. It must also be aligned with who has the knowledge to make each class of decisions. The optimal structure requires some sharing of decision rights between a platform owner and app developers.
- *Control complements decision rights.* Modular ecosystems organized around modular architectures require monolithic integration processes, which a platform's control mechanisms provide.
- *Aligning a control portfolio uses five simple rules*. It should be simple, transparent, realistic, reflect shared values, and fair.
- *Platform pricing policies*. Pricing policies must be aligned with the platform's business model, its stage in its lifecycle, and its architecture.

Chapters in the next section delve into the evolution of ecosystems, platforms, and apps.