Software Platform Ecosystems

INF5750
2017

Unless noted, all tables, citations, and figures are taken from or are facsimiles from: Tiwana, Amrit. *Platform ecosystems: aligning architecture, governance, and strategy*. Newnes, 2013.

From chapters 1, 2 and 5
Contents and learning outcome of the lecture

- What platforms are, and their core components
- Difference between *software platforms* and other types of platforms
- *Drivers* towards software platforms
- Some important *concepts*
- Some important *principles*
- Important aspects of platform *architecture*
- Platform *lifecycles*
- How does all this relate to your group assignments and the DHIS2?
Why software platform ecosystems?

• Software platform ecosystem «logics» increasingly plays a more dominant role in competition in a diverse sets of markets

• Competition migrating to rival platforms
  • potent mix of specialized expertise with the disciplining power of platform markets can foster innovation at a pace that can trump even the mightiest product and service business, e.g. Blackberry vs Apple and Google; Camera produces vs mobile phones.

• Why in the open source development course?
Main components of a software platform

<table>
<thead>
<tr>
<th>Table 1.1 Core Elements of a Platform Ecosystem</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Element</strong></td>
</tr>
<tr>
<td>Platform</td>
</tr>
<tr>
<td>App</td>
</tr>
<tr>
<td>Ecosystem</td>
</tr>
<tr>
<td>Interfaces</td>
</tr>
</tbody>
</table>

![Diagram of a platform ecosystem](image)

**FIGURE 1.1**
Elements of a platform ecosystem.
Evolution of platform ecosystems

• Architecture: Structure

A conceptual blueprint that describes how the ecosystem is partitioned into a relatively stable platform and a complementary set of apps that are encouraged to vary, and the design rules binding on both.

• Governance: Process and rules

Broadly, who decides what in a platform’s ecosystem. This encompasses partitioning of decision-making authority between platform owners and app developers, control mechanisms, and pricing and pie-sharing structures.

• Evolution: «... the interplay between its irreversible architecture and how it is governed.”
Focus in Tiwana (2013): software platforms:

• Platforms where third party complementors add to platform capabilities and functionality
  • Possibilities for hundreds or thousands of actors to add functionality to the same ecosystem

• Upstream value chain the platform itself. Downstream app developers. End users can uniquely mix-and-match downstream complements – making the innovation and adoption in the downstream central for success of failure

• True platforms must be at least two-sided and span at least two distinct groups app developers and end-users that interact through the platform.

• Most successful platforms began as standalone products or services: iOS, Windows, Facebook, Amazon, eBay, Google, Firefox, Salesforce, and Dropbox

• What does that imply and mean?
Drivers towards platformization

<table>
<thead>
<tr>
<th>Driver</th>
<th>Description</th>
<th>Consequences</th>
</tr>
</thead>
</table>
| Deepening specialization    | Increased need for deep expertise due to growing complexity of products and services | • Simultaneously shrinking and expanding firm boundaries  
• Red Queen effect from clockspeed compression  
• Increased interdependence among firms |
| Packetization               | Digitization of “something”—an activity, a process, a product, or a service—that was previously not digitized | • Location-independent distribution ability of work  
• Deepening specialization |
| Software embedding          | Baking a routine business activity into software                           | • Products-to-services transformation  
• Morphing physical–digital boundary  
• Convergence of adjacent industries |
| Internet of Things          | Everyday objects inexpensively gaining the ability to directly talk using an Internet protocol | • Deluge of data streams from networked objects  
• Context awareness |
| Ubiquity                    | The growing omnipresence of cheap and fast wireless Internet data networks | • Loosely coupled networks rival efficiencies of firms  
• Actors who can participate from where  
• Actors where services can be delivered  
• Scale without ownership |

FIGURE 1.3
The five drivers of the migration toward platform-centric business models.
More about drivers

• Deepening specialization:
  • Software code grow larger and become more complex (more functionality) -> more specialization needed for further growth.
  • -> More focus needed for companies
  • -> Need for integration of distributed knowledge from others
  • -> More effort to compete against successful platform owners

• Packetization:
  • Digitalization of an activity, process, product or service
  • -> Enables transportation of information through the Internet – high speed, low cost – Removes location constraints to work -> new possible business models
  • -> Deepening specialization
  •  Example: global radiology service in India (e.g. https://www.outsource2india.com/services/radiology.asp)
More about drivers

• Software embedding:
  • Making software of business processes and activities
    • Example: credit card, Vipps, cool photo filters
  • -> from products to services – clients to web-based services, software based maps in cars
  • -> physical – digital boundary -
  • -> convergence across industries – gaming consoles and cameras into phones, Amazon kindle

• Internet of things:
  • Cheap sensors online and networked
    • Example: Sensors to monitor patients at home, door sensors telling if you forget to lock your door
  • -> From stock of data to streams of data
  • -> Communication of contextual data
  • Examples: One Tesla car telling about hump in the road – all other cars get the information and adjust car configuration to take less impact when driving through the same place.
  • Optimalization of resources in a hospital, dynamic prize regulations
More about drivers

• Ubiquity:
  • Presence of Internet «everywhere» – lower prices – faster network
  • -> location independence of tasks and services
  • -> networks of firms
  • -> crowdsourcing
    • Example: Google maps traffic information

• The combination of the drivers
  • Pushing innovation ecosystems towards growing number of industries, like:
    • mortgage, finance, drug development, software, automotive, healthcare, banking, food services, and energy
## Platform concepts

- **Platform lifecycle:**

<table>
<thead>
<tr>
<th>Concept</th>
<th>Relevance</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform lifecycle</td>
<td>● ● ●</td>
<td>A multifaceted characterization of whether a technology solution—a platform, an app, or the entire ecosystem—is in its pre- or post-dominant design stage; its current stage along the S-curve; and the proportion of the prospective user base that has already adopted it.</td>
</tr>
<tr>
<td>Dominant design</td>
<td>● ●</td>
<td>A technology solution that implicitly or explicitly becomes the gold standard among competing designs that defines the design attributes that are widely accepted as meeting users’ needs.</td>
</tr>
<tr>
<td>S-curve</td>
<td>● ● ●</td>
<td>A technology’s lifecycle that describes its progression from introduction, ascent, maturity, and decline phases.</td>
</tr>
<tr>
<td>Leapfrogging</td>
<td>● ● ●</td>
<td>Embracing a disruptive technology solution and using it as the foundation for the firm’s market offering in lieu of an incumbent solution in the decline phase of its S-curve.</td>
</tr>
<tr>
<td>Diffusion curve</td>
<td>● ●</td>
<td>A description of whether a technology solution—a platform or an app—is in the stage of having attracted the geeks, early majority, early adopters, late majority, or laggards to its user base.</td>
</tr>
</tbody>
</table>
Lifecycle

**FIGURE 2.2**
Pre- and post-dominant design phases in a software platform.

**FIGURE 2.3**
S-curves in the technology lifecycle.
## Platform concepts

- **Platform properties:**

<table>
<thead>
<tr>
<th>Concept</th>
<th>Relevance</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multisidedness</td>
<td></td>
<td>The need to attract at least two distinct mutually attracted groups (such as app developers and end-users) who can potentially interact more efficiently through a platform than without it</td>
</tr>
<tr>
<td>Network effects</td>
<td>● ●</td>
<td>A property of a technology solution where every additional user makes it more valuable to every other user on the same side (same-side network effects) or the other side (cross-side network effects)</td>
</tr>
<tr>
<td>Multihoming</td>
<td>●</td>
<td>When a participant on either side participates in more than one platform ecosystem</td>
</tr>
<tr>
<td>Architecture</td>
<td>● ●</td>
<td>A conceptual blueprint that describes components of a technology solution, what they do, and how they interact</td>
</tr>
<tr>
<td>Governance</td>
<td></td>
<td>Broadly, <em>who decides what</em> in a platform’s ecosystem. This encompasses partitioning of decision-making authority between platform owners and app developers, control mechanisms, and pricing and pie-sharing structures</td>
</tr>
</tbody>
</table>
Properties

**Figure 2.8**
Two sides in a multisided platform.

**Figure 2.9**
Network effects leverage the number of users that any user can communicate with.

<table>
<thead>
<tr>
<th></th>
<th>Same Side</th>
<th>Cross-side</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Negative</strong></td>
<td>Adding someone decreases appeal to all existing users on the <em>same</em> side</td>
<td>Adding someone decreases appeal to all existing users on the <em>other</em> side</td>
</tr>
<tr>
<td><strong>Positive</strong></td>
<td>Adding someone increases appeal to all existing users on the <em>same</em> side</td>
<td>Adding someone increases appeal to all existing users on the <em>other</em> side</td>
</tr>
</tbody>
</table>
Platform concepts

• Platform dynamics:

<table>
<thead>
<tr>
<th>Concept</th>
<th>Relevance</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tipping</td>
<td>●</td>
<td>The point at which a critical mass of adopters makes positive network effects take off</td>
</tr>
<tr>
<td>Lock-in</td>
<td>●</td>
<td>The ways in which a platform can make it more desirable for existing adopters to not jump ship to a rival</td>
</tr>
<tr>
<td>Competitive durability</td>
<td>●</td>
<td>The degree to which the adopters of a technology solution continue to regularly use it long after its initial adoption</td>
</tr>
<tr>
<td>Envelopment</td>
<td>●</td>
<td>When a platform swallows the market of another platform in an adjacent market by adding its functionality to its existing bundle of functionality</td>
</tr>
</tbody>
</table>
Platform guiding principles

- Platform startup principles:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chicken-or-egg problem</td>
<td>The dilemma that neither side will find a two-sided technology solution with potential network effects attractive enough to join without a large presence of the other side</td>
</tr>
<tr>
<td>The penguin problem</td>
<td>When potential adopters of a platform with potentially strong network effects stall in adopting it because they are unsure whether others will adopt it as well</td>
</tr>
</tbody>
</table>
Platform guiding principles

- Platform design principles:

<table>
<thead>
<tr>
<th>Seesaw problem</th>
<th>The challenge of managing the delicate balance between app developers’ autonomy to freely innovate and ensuring that apps seamlessly interoperate with the platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Humpty Dumpty problem</td>
<td>When separating an app from the platform makes it difficult to subsequently reintegrate them</td>
</tr>
<tr>
<td>Mirroring principle</td>
<td>The organizational structure of a platform’s ecosystem must mirror its architecture</td>
</tr>
</tbody>
</table>
# Platform guiding principles

- **Platform evolution principles**

<table>
<thead>
<tr>
<th>Emergence</th>
<th>Properties of a platform that arise spontaneously as its participants pursue their own interests based on their own expertise but adapt to what other ecosystem participants are doing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coevolution</td>
<td>Simultaneously adjusting architecture and governance of a platform or an app to maintain alignment between them</td>
</tr>
<tr>
<td>Goldilocks rule</td>
<td>Humans gravitate toward the middle over the two extreme choices given any three ordered choices</td>
</tr>
<tr>
<td>Red Queen effect</td>
<td>The increased pressure to adapt faster just to survive is driven by an increase in the evolutionary pace of rival technology solutions</td>
</tr>
</tbody>
</table>
Some key points

• The lifecycle of a technology solution has three-dimensions:
  • pre- or post-dominant design stage (from many to one)
  • maturity trajectory (the S-curve)
  • proportion of the total prospective user base adoption

• Multisidedness offers: same-side and cross-side network effects, lock-ins (coercive and value-driven), prospects of swallowing or be swallowed

• Architectures provide blueprint for mass coordination. Conventional coordination and control mechanisms costly and implausible in large ecosystems

• Governance can amplify or diminish the advantages of good architecture. Governance and architecture must be co-designed and coevolved

• Evolutionary pace of a platform is relative to its rivals (the Red Queen effect).

• Emergent innovation can only be facilitated, not planned by a platform owner.
  • Spontaneously arise from the selfish pursuit of self-interest by individual ecosystem participants.

• Chicken-or-egg problem and the penguin problem to get off the ground - unattractive for either side to join unless there is a critical mass on the other side. Uncertainty about whether others will join the platform ecosystem can stall initial adoption, creating the penguin problem

• Balance autonomy with integration (the seesaw problem) separable but re-integratable (Humpty Dumpty). Organized to mirror the architecture and the “microarchitecture” (the mirroring principle).
Platform architectures

• The architecture enable (or not) participation among potential and actual third party innovators

• Third party innovators must be able and motivated to participate
  • Ability through architecture
  • Motivation through governance

• Main architecture parts (and their interconnectedness)
  • Platform core
  • Platform interfaces
  • «Apps»
Managing complexity

• What is complexity?
  • A function of the number of parts, types of parts, and number and types of connections between the parts.
    • Structural (difficult to describe)
    • Behaviorally (difficult to control and predict)
  • Too high complexity will lead to at least
    • Incomprehensibility
    • Gridlock
  
  • -> loss of predictable output from input – ripple effects
  • -> co-innovation risk (80%×80%×80%=51%) – need to reduce dependencies at the right place
Managing complexity

• In a platform ecosystem with numerous actors, complexity must be controlled somehow to reduce risk of gridlocks, unpredictable ripple effects and co-innovation problems

• -> Architecture
  • Balancing between control and autonomy
  • Keeping transaction costs and coordination cost as low as possible
Architecture solutions to orchestrate

• Partitioning (modularization) – core <-> apps - degrees
  • Creating «autonomous» subsystems
    • To cognitively manageable parts
  • Blackboxing
    • Visible information: what they do and how to interact with them
    • Hidden information: how they work

• Systems integration
  • Development activities coordination between platform owner and app developers
    • Managing dependencies
    • Minimizing need for coordination
  • Apps must be integrated to the platform to enable value to end-users
    • Platform – app integration – uneven development, platform changes – ongoing effort
    • App – app integration
Architecture solutions

• Relatively stable core
  • Platform architecture
  • Visible part: Shared sets of assets through defined interfaces
  • Hidden: inner functions of the platform core to make interfaces work and behave as they do

• Dynamics and variability in apps -> innovation
  • Microarchitecture
App architecture (microarchitecture)

1. Presentation Logic
2. Application Logic
3. Data Access Logic
4. Data Storage
Possible partitioning of layers

**Figure 5.10**
Each of the four functional elements of an app can be flexibly partitioned between an app and the platform.
Many possibilities for partitioning the app

**FIGURE 5.11**
All four functional elements reside on the client device in the standalone app microarchitecture.

**FIGURE 5.12**
All four functional elements reside on the client device in the cloud app microarchitecture.

**FIGURE 5.13**
Only data storage resides on the server side in client-based app microarchitecture.

**FIGURE 5.14**
Client–server app microarchitectures evenly split application functionality among clients and servers.
Many possibilities for partitioning the app

**FIGURE 5.15**
Peer-to-peer app microarchitecture.

**FIGURE 5.16**
Tiering.
App architecture choices have consequences

• Hard, or, impossible to maximize all positive consequences; always trade-offs between partitioning inside the platform and across the Internet

• Early architecture choices are hard to change later
  • -> creating path dependencies in architectures

• Some characteristics show up immediately:
  • speed, security, reliability, scalability, testability, and usability

• Some at later stages:
  • maintainability, extensibility, evolvability, and the capacity to mutate and envelop adjacent app market segments

• Developers need knowledge about which types of app architectures gives which types of trade-offs and advantages
  • -> design, not experience too late
Platform architecture

• In practice, irreversible
  • -> have to stick with early choices and their consequences

• Desirable properties
  • Simple; defined interfaces, functionality etc.
  • Resilient; not breaking the ecosystem upon app failure
  • Maintainable; minimizing consequences of local changes
  • Evolvable; balancing between stability/control of interfaces and autonomy of innovation

• But also here, trade-offs.
More on modularization and amount of modularity

• Monolithic versus modular
• Not either or – rather a continuum between the two extremes, where most lies in between

• Some important aspects:
  • Division of work among several organizations/actors
    • Emergent properties
  • Dependencies among modules is restricted to defined interfaces
  • Need to be compliant only to interface specifications
  • Possible performance sacrifices
## Balancing needs and implications

### Table 5.2 Upsides of Modularizing a Platform for Platform Owners and App Developers

<table>
<thead>
<tr>
<th>Platform Owner</th>
<th>App Developer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Massively distributed innovation</td>
<td>Less reinvention, more specialization</td>
</tr>
<tr>
<td>Increased variety of apps</td>
<td>Valuable ignorance</td>
</tr>
<tr>
<td>Greater volume of incremental innovation</td>
<td>Greater app evolvability</td>
</tr>
<tr>
<td>Control via architecture rather than ownership</td>
<td>Multihoming in rival platforms more feasible</td>
</tr>
</tbody>
</table>

### Table 5.3 Downsides of Modularizing a Platform for Platform Owners and App Developers

<table>
<thead>
<tr>
<th>Platform Owner</th>
<th>App Developer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modularity is not free</td>
<td>Modularity imposes additional costs</td>
</tr>
<tr>
<td>Technical performance takes a hit</td>
<td>App performance takes a hit</td>
</tr>
<tr>
<td>Modularization forecloses architectural innovation</td>
<td>Modularity constrains experimentation</td>
</tr>
<tr>
<td>Increased risk of imitation by rivals</td>
<td>Leveraging the platform risks getting locked into it</td>
</tr>
</tbody>
</table>
Balancing needs and implications

**Figure 5.18**
Tradeoffs between modular and monolithic platform architectures.
What is in, what is out?

• High-reusability functionality
• Generic functionality
• Stable functionality
• Interfaces integral parts of the platform

• High uncertainty functionality – out

• But also in:
  • For attractiveness
  • Expectation from end-users
The interfaces

- Standardization
- Stability
- Versatility
  - flexibility in standards
  - highly dependent functionality stays in the platform
- Openness
  - who can participate
DHIS2 as a platform ecosystem?

• How do your developed apps relate to platform architectures as described?

• Do the architectural choices in your app (together with the DHIS2) imply anything for further development and evolvement of your app, and in relation to the DHIS2 core
  • Dependencies – loose coupling
  • Modularization
  • Usage of APIs
  • Placement of functionality and layers
## Platform vs application vs Information infrastructure

<table>
<thead>
<tr>
<th>Property/Type of IT system</th>
<th>Application</th>
<th>Platform</th>
<th>Information Infrastructure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Emergent properties</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shared</td>
<td>Yes, locally and through specified functions</td>
<td>Yes, across involved user communities and across a set of IT capabilities (Star and Ruhleder, 1996; Porra, 1999)</td>
<td>Yes, universally and across multiple IT capabilities (Star and Ruhleder, 1996; Porra, 1999)</td>
</tr>
<tr>
<td>Open</td>
<td>No, closed by user group and functionality</td>
<td>Partially, depends on design choices and managerial policies</td>
<td>Yes, universally allowing unlimited connections to user communities and new IT capabilities (Weill and Broadbent, 1998; Kayworth and Sambamurthy, 2000; Freeman, 2007)</td>
</tr>
<tr>
<td>Homogeneous</td>
<td>Yes, partially and mainly by involved social groups</td>
<td>Partially, mainly by social groups but also by technical connections</td>
<td>Yes, increasingly heterogeneous both technically and socially (Kling and Scacchi, 1982; Hughes, 1987; Kling, 1992; Edwards et al., 2007)</td>
</tr>
<tr>
<td>Scaling</td>
<td>Yes, but limited by time horizon and user community.</td>
<td>Yes, and limited by architectural choices and functional closure</td>
<td>Yes, unlimited by time or user community (Star and Ruhleder, 1996; Freeman, 2007; Zimmerman, 2007)</td>
</tr>
<tr>
<td>Linear growth</td>
<td>Mostly linear growth</td>
<td></td>
<td>Both linear and nonlinear growth (Hughes, 1987)</td>
</tr>
<tr>
<td>Evolution bounded and context free</td>
<td>Evolution path dependent</td>
<td>Evolution path dependent (Star and Ruhleder, 1996; Porra, 1999; Edwards et al., 2007)</td>
<td></td>
</tr>
<tr>
<td><strong>Structural properties</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Organizing principle</td>
<td>Direct composition of IT capabilities within a homogeneous platform</td>
<td>Direct composition of a set of horizontal IT capabilities within a set of homogeneous platforms</td>
<td>Recursive composition of IT capabilities, platforms and infrastructures over time (Star and Ruhleder, 1996; Edwards et al., 2007)</td>
</tr>
<tr>
<td>Control</td>
<td>Centralized</td>
<td>Centralized</td>
<td>Distributed and dynamically negotiated (Weill and Broadbent, 1998) Can involve only basisorganizing principles (standards) and rely on installed base inertia (Star and Ruhleder, 1996; Edwards et al., 2007).</td>
</tr>
</tbody>
</table>