Platforms as Products:
Serving the customer who serves the customer

Ryan Murray
Director, Platform Strategy
Who am I?

ThoughtWorks is a 20+ year global tech consultancy, 4,000+ people strong across 37 offices in 12 countries. We are a collection of passionate people who care about technology. At our heart, we are a custom software company and a community of designers, strategists and industry experts who solve complex problems.

Realized over time - we’re doing it wrong - fragmented efforts (lack cohesive strategy), reinventing the wheel, not finding balance of autonomy and prescription (unlock solving most interesting problems)

Founded an internal group around platform technology which not develops our offering around platform strategy technology
Your an established leader, or breaking new ground, need to move faster.

Many customers we work with have made significant investments in platforms
lot of big bang investments
- “build it and they will come”, “the devs don’t understand what they need”
Infrastructure driven by efficiency motive and cost takeout.
Often executed out of people, business context , “captive audience” mentality

The real value of technology platform investment is creating opportunities to move faster
Lot of exciting approaches & tools happening - containerization, devops, microservices, API first development
potential for acceleration, but much more so if viewed holistically

Help teams focus **100% on delivering business value** not reinventing operational technology.

Naive cross functional teams and some devops interpretations can make this worse.
“Build it & you run it is great” **build everything you need to run it**, not so much

Many of these platform effort miss the mark, and provide far less lift than they could
They don't focus on their customer
“Platform” is a term that is meaningful at many levels. 
Not talking about platform business models,
talking about foundational technologies.

various types of infrastructure, business capability APIs and common components, that support delivery teams in delivery value to the customer.

“The stuff software delivery teams have to do that the business doesn't care about”
Key areas of technology platform

**Infra** is a traditional platform - today about PaaS, etc.

**Serverless** doesn’t change all this - in some ways it will make it worse

**Delivery Infra** - Auth/c, PaaS, security, monitoring, build and deployment pipelines, even test data

**Business Capability APIs & Architecture** (incl identity / auth, event arch) - particularly internal APIs around core systems

**Data**,  

**Experimentation** (think Optimizely but for your customer touchpoints and core systems)

Google video player story
<table>
<thead>
<tr>
<th>Platform Thinking</th>
<th>Reduce Friction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Enable Internal Ecosystems</td>
</tr>
<tr>
<td></td>
<td>Provide a Foundation for Experimentation</td>
</tr>
</tbody>
</table>

Platform thinking
As my colleagues and I have worked on various forms of platforms, we've found these key outcomes to be valuable as a filtering criteria for Are building the right platform capabilities With the Right Priorities

Reducing friction
Enable Ecosystems
Foundation for Experimentation
Removing friction - Fundamental things platforms do
[Look at diagram on slide]

Customer Value versus operational scaffolding
For any team, what are the most important things your teams are spending time doing and could they get for “free”?
The best abstractions give you some magic essentially for free. Think AWS.
—>> Advance Slide

Types of friction
Delivery friction - from infrastructure to build & deployment pipelines to runtime feature flags
Access friction - to assets & capabilities - Well factored API access to business capabilities (example product & orders)
Cognitive friction - Common style / approach across resources I consume (dev time || architectural style legacy integration, SOAP; docs // operational time :: operation complexity / fault tolerance, problem triage)
Next platform outcome - Enabling ecosystem

Ecosystems are a set of interconnected parts ... providing independence of consumer from producer

many organizations we build a lot of point solutions which have overlapping or related responsibilities.

common mergers & acquisitions

Example

Item legacy + Item master example

Integrating with point solutions pushes complexity to the clients, error prone.

Doesn't provide scale
Well design platforms enable ecosystems & provide scale - scale of consuming capabilities

Back to Item management example,
now multiple teams consume the item capability with minimal friction
And it decouples the evolution of the complexity of legacy remediation from the clients

easy to consume platform component - Effectively Complexity is contained -

This provides “vertical scale” - building one capability on another
Foundation for Experimentation

experiments need to be cheap and easy to run or we won't run them

Platforms also provide horizontal scale

Platforms enable experimentation in several ways:

**Delivery platform** makes it easy to build, deploy and operate new things

**Easy to consume API ecosystem** - experiment quickly with new workflows, reduce risk

**Experiment data with APIs and pre-integrated front end components** - multivariate testing

Example of data collection service ties to collection pipeline tied to dashboarding capability
The goal of a platform should be to provide a compounding effect.

In many organizations, we see only point solutions or unfocused attempts at “platform” building. They don’t focus on the key outcomes we just discussed.

The execution capacity at these organizations tend to stagnate - stuck in the mud.

Example - infrastructure platforms that provide virtualized capacity but still have high-friction manual processes or artificially impose resources limits.

→ ADVANCE SLIDE - compounding effects

The leading digital organizations have done platforms this very well (examples Google, Amazon).

  - **Amazon** - Bezos mandating all teams communicate via services, AWS
  - **Google** - Borg, Gaia

So how do we ensure our platforms actually deliver the compounding effects?
key success factor -> thinking of your platform capability as products,
Products are are built iteratively with intention - understanding their market and customer needs
When we look at building platforms, we turn the lens inward
The customers of platform capabilities are the delivery teams building valuable products for the organization
What does it mean to manage platforms a products?
there are several areas of platform technology -> delivery infrastructure, APIs, data - each organization will have different platform priorities
Product owner dedicated to each capability to manage the desired outcomes for consumers
Essential aspects of product ownership that are essential for building platform capability are:

Need to solving for customer needs == Customer-centricity
Successful products solve for real market needs
Engage regularly with our customers

Need a clear owner that can drive strategy and prioritization
Group accountability or accountability without execution capacity don't work

Adoption and scaling - Unless you are a build and make them come shop, you need strategy
Balancing feature development with operating and support
Evangelizing the product
Key thing Product teams do - solve for customer needs
Need to Connect with your customer & Understand the domain
Market research - understand the needs of development teams that consume the capability
And spend time with the business that “own” the capability
Your capability is a product - the business is your essential stakeholder
Delicate balance of doing what customers want (convenience) and representing what the business “is”

Surveys - Voice of the customer - Ongoing surveys of development teams - Nike example

Get the High level vision - Business model canvas
A well known template in the lean startup space. 

You can't build an API without knowing what your “business model” is - **who does this API serve, what can it enable?**

Focus on the **API value proposition, customer “segments”, revenue stream, cost to operate**

A useful and familiar tool for getting **clarity and building alignment with the business stakeholders.**
To build a great platform product - Enable composability

What enables composability? How do you enable multiple customers in the ecosystem?

Domain-centricity (not technical or system view, DDD -> Bounded Context)
Balancing what one customer wants versus what many customers will need

The best way to prevent composability is to treat the API as an implementation detail of a desired workflow

Coarser grained - API is for “order management” - focus key assets and operations not the “getIncompleteOrders” API

Emit domain events (decouples from communicating with the capability teams as much)
Product teams provide Longterm Ownership, Strategy and Prioritization

Balancing iteratively (over time) what customers think they need with what the capability “needs to be” (business domain, not atrophied business systems)

Product owner - Someone is accountable for RIGHT thing, to do, WHEN versus the Fastest thing?
Contrast to project mentality - API development is just typically a “feature”

Fight the "can you just add a some fields to that payload" -- messing up the domain model

With POs, Cross-product feature arbitration is natural - You need less Governance (control) -> more about cross-platform team interactions focus on strategy (opportunity)

Creates incentive for maintenance, evolution (technical debt)

Example: Design challenge: Catalog service taking on inventory data - easy for web site
Ownership, Strategy & Prioritization

1. Long Term Ownership

2. Priority: Balancing Costs & Customization

Specific challenge of the Product Owner - Balance Cost & Customization
Complexity Tradeoffs - don't solve all possible cases - Pushing composition to up to consumer
Easy for enterprise project to hyper-customize for a small user base
Prioritization is a Balancing act: Metrics & KPIs - lean approach to evolving capabilities
All KPI goals are imperfect but
There are useful KPIS to drive platform outcomes
Time to Hello World
NPS (surveys)
# of consuming applications (incl prospectives) / traffic volumes
Need to change API on new request / Parallel API changes -> refactor capability
### Adoption & Scaling Strategy

<table>
<thead>
<tr>
<th></th>
<th>Social Strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Developer Experience</td>
</tr>
<tr>
<td>3</td>
<td>Self Service</td>
</tr>
<tr>
<td>4</td>
<td>Affordances</td>
</tr>
</tbody>
</table>

As much as you need to build a great product - you also need to get it adopted

**Adoption Strategies**

**Social**

**API evangelization** - “platform evangelists” - “go to people”, hackathons

May be product owners but **should always have engineers who do some evangelization**

Participate in CoPs

**Not just a “push” channel - also learn about your user needs and frustrations**
User Experience == Developer Experience (DX)
Making our product intuitive and easy to use drives adoption
Can end up spending a lot of time supporting the API rather than building new value
Really second order concern - must have a core value proposition that works;
Great experience design can make a big difference to adoption
DOTNET User testing / Usability analysis
Self-service - Critical area of DX
This is an umbrella concept - How do we reduce predictable communication needs?
Most organization do not do this well for internal APIs, to their detriment - NorthStar for funding conversations
Your goal is to reduce the “Time to Hello World” and reduce support requests (“How do I?”).
AWS is the great example of self-service infrastructure.
Some of these cross-cutting concerns are hard to weave in later

Self-signup - self-sign-up for api keys (not as common in the enterprise) ...
Adoption & Scaling Strategy - Documentation

Documentation - number of good documentation tool (an outcome - not a tools argument)
"Try it out" direct from docs is valuable - very powerful form of self-service
Format - Stripe example - 3 pained approach - navigation on left, docs center and language specific example
Up-to-date Docs with examples can greatly reduce support calls
Good docs are a DOUBLE WIN - reduce drag on the API dev team as well as friction on the consuming delivery team
Put a feedback link on the docs page or host them in github
Take some UX to your documentation, e.g. personas of documentation users (Brad Fults):
- **Newcomer**: Developer looking to get started
- **Debugger**: Dev troubleshooting a specific issue
- **Decision Maker**: CTO/Architect evaluating competing APIs
- **Searcher**: Product manager figuring out if X is possible with the API
SDKs & Code samples - Time to Hello World, Intellisense in IDE, Less need for RTFM
Affordances
Donald Norman (Design of Everyday Things) defined affordances as properties that make objects easy to use and hard to misuse, all without the need for an instruction manual.
His famous example of putting pull bars on both sides of a door is a great example of making something commonplace confusing.

→ [Cut to video]
Why pull bars when you need to push

Ask your consumers what are the equivalent for your APIs / capabilities?

**strongly consider an API style guide to establish patterns**
Adoption & Scaling Strategy - Affordances

```
{
   "status": 400,
   "message": "No to number is specified",
   "code": 21201,
   "more_info": "http://www.twilio.com/docs/errors/21201"
}
```

**Some examples of Affordances -> Hysterix default behavior**

- **APIs** - Human friendly payloads
- **Documentation URLs** in response
- **Returning null values** for missing keys
- **Providing breadcrumb trails for troubleshooting, e.g.**
  - Stripe adds a Request-Id response header that keys into their logs for support requests
  - Twill Request-Duration header to tell you how long Twilio thinks the call took
- **Passing back full urls instead of relative paths**, so that the client dev code “doesn’t have to think”
Platform thinking and key outcomes (long term acceleration)
remove friction, enable ecosystems, enable experimentation
At any scale of platform teams, this is a key thought model

Treat your platform capabilities as products
Customer-centric, Ownership, KPI, Focus on adoption & scaling

The one thing I'd like to leave you with -> The case for overarching platform strategy (PlatformMANIFESTO)

We start with a 5-6 week discovery phase to build alignment and momentum

Get senior leadership to agree on the value of platform building and the KPIs that will guide it

This is not just an technology unlock, moving to a product mentality this requires organizational change, skills development

A strategic approach will help determine: How to align your IT and product organizations around the mission

Platform building is not easy - you need to have a comprehensive and evolving strategy for platform execution.