Products Over Projects

Software projects are a popular way of funding and organizing software development. They organize work into temporary, build-only teams and are funded with specific benefits projected in a business case. Product-mode instead uses durable, ideate-build-run teams working on a persistent business issue. Product-mode allows teams to reorient quickly, reduces their end-to-end cycle time, and allows validation of actual benefits by using short-cycle iterations while maintaining the architectural integrity of their software to preserve their long-term effectiveness.

20 February 2018

Sriram Narayan

Sriram is a Digital-IT Management Consultant at Thoughtworks. He helps clients improve the performance of their Digital/Product/Engg./IT departments through changes to their ways of working (tech operating model). His book on this topic— Agile IT Org Design, was featured as a must-read for CIOs by Enterprisers Project, a joint initiative of Harvard Business Review, CIO Magazine and RedHat.

ENTERPRISE ARCHITECTURE

TEAM ORGANIZATION

CONTENTS

Software projects are a popular way of funding and organizing software development. Projects are funded on a case-by-case basis on the basis of benefits projected in a business case. They are organized in the form of one or more temporary teams whose members have durable reporting lines outside the project organization. They are

Table of Contents

staffed from a “pool of talent” whose members are considered fungible within lines of specialization. And usually, a software project team’s job is to build or enhance some system or application and move on.

However, projects are not the only way of funding and organizing software development. For instance, many companies that sell software as a product or a service do not fund or organize their core product/platform development in the form of projects. Instead, they run product development and support using near-permanent teams for as long as the product is sold in the market. The budget may vary year on year but it is generally sufficient to fund a durable, core development organization continuously for the life of the product. Teams are funded to work on a particular business problem or offering over a period of time; with the nature work being defined by a business problem to address rather than a set of functions to deliver. We call this way of working as “product-mode” and assert that it is not necessary to be building a software product in order to fund and organize software development like this.

What is Product-Mode?

“Product-mode” is a way of working. It is a way of funding and organizing software development that differs significantly from the projects way of doing it. Although generally applicable to digital-age enterprise IT, this way of working is especially suited to those who aim to drive business through a digital platform. The differences with projects are summarized below and elaborated in the rest of the article.

Projects	*Product-mode*
What gets funded?	
A pre-defined solution or outstanding scope gets funded.	A team gets funded. A product-mode team is funded on a rolling basis (e.g. for a year at a time) with periodic reviews. The team takes on one problem after another, roughly the same space, working through an evolving roadmap aligned with product/business strategy.
For what part of the development lifecycle is funding made available?	
Mainly for building the solution.	For building, running and iterating on the solution or even pivoting to a different solution till the underlying problem is *verifiably* solved.

How are the value stream stages of “ideate”, “build” and “run” organized?	
As separate departments.	As a single department with unified reporting hierarchy.
How long do teams last?	
Until project funding lasts which ideally means until the originally conceived solution is delivered. Typically, several weeks or months.	As long as the roadmap has business relevance. Typically, several years.
Do people stay and work in the same general area for long?	
Not by design. For any given individual, the next project may be in a completely different area.	Yes.
How does prioritization work?	
Projects are prioritized by business along with a project portfolio management (PPM) group. Read more.	Roadmap items are prioritized by product owners and their business counterparts. Cross-cutting initiatives are prioritized by the business or tech leadership. Initiatives don’t get their own team. They are parceled out to pre-existing product-mode teams.
How does benefits validation work?	
There is emphasis on formal assessment of projected benefits prior to project approval. There is typically no mechanism for validation of actual benefits after project delivery.	Product owners prove actual benefits either with data from A/B testing, analytics, user surveys, etc. or with feedback from business. This ability is dependent on good engineering capability to develop and release frequently in small chunks and good analytics capability to determine delta changes in adoption, conversion etc. There is relatively less emphasis on assessing projected benefits upfront, especially amongst the best such teams that execute with short cycle times and can therefore try new ideas without incurring a high cost of failure. The product owner is empowered to approve development of roadmap items as they see fit. By developing in small, end-to-end iterations, product owners are able to detect early any efforts that miss the mark and thereby fail-fast (fail-cheap).
How is success defined?	
Delivery of agreed-upon scope on time and budget.	Improvement of a metric directly related to a business outcome or not more than one or two levels removed from a business outcome. Thus, every product-mode team is ideally a business KPI driven team. For example, Scout24 operates like this with teams responsible for traffic, engagement etc.

Are there any constraints around team organization?

No.	Products-mode works best when teams are organized to be simultaneously aligned with business relevant capabilities and with enterprise architecture boundaries. Without the former, they may lose alignment with business goals. Without the latter, they lose out on autonomy i.e. the ability to evolve their systems relatively independent of other teams.

Product-mode is no longer limited to companies that sell software. It is common among so-called tech businesses enabled by tech platforms that stream content, e-tail, distribute mutual funds, find cabs, accommodation, flights, you name it. It is also catching on in the digital/product/engineering/IT departments at more traditional, old-guard businesses. For instance, Insurance Australia Group (IAG) recently moved away from projects to a more durable platform organization operating in product-mode. ANZ Bank is trying something similar.

There are several, less-than-ideal variations of teams operating in product-mode. Some places use a halfway approach of project-mode funding and product-mode organization. Even the product-mode organization is not always build+run. Or the cross-functional teams consist of people reporting to different function heads.

An ideal product-mode team is an empowered, outcome-oriented, business-capability aligned, long-lived, cross-functional, ideate-build-run team that is able to and expected to solve problems and improve business outcomes, rather than deliver scope on schedule. The problems that these teams take up are usually long-lived e.g. continuously improve the conversion from cart to checkout (reduce cart abandonment rate). Problems also need to be defined sufficiently narrowly so that individual teams can own them. For instance, although "Net Promoter Score" (NPS) is a great holistic metric, it is typically too broad for any particular team to sign up for.

In product-mode, it is not just the team that is long-lived, its association with a problem area is also long-lived. Product-mode teams often run in pull-based development mode with a focus on achieving flow and less emphasis on achieving predictability using story level estimation and release plans.

Benefits of Operating in Product-Mode

In today's (2017) business environment, operating in product-mode has several advantages to operating in projects-mode.

Ability to Reorient Quickly

It used to be sufficient for IT/engineering to be setup to respond within a year of receipt of market intelligence or feedback. With "digital" going mainstream, such is no longer the case. Let's take an online retail example. Broadly, the capabilities of a basic, online retail platform may be classified as catalog, order management, payment and fulfillment.

In projects-mode, it might be the case at a point in time, that owing to priorities, we have active projects affecting order management, payment and fulfillment but not catalog. Now if were to receive unexpected, catalog related feedback from the market or from business stakeholders, we wouldn't have a team in place ready to act on the feedback. Usually, any new feedback would by default be subject to business case and project approval processes followed by prioritization for staffing. Even if we could suspend one of the active projects and redeploy the team to catalog, the team would likely be new to catalog and therefore unable to hit the ground running.

By contrast, in product-mode, we'd have a long-lived team (or teams) dedicated to each of catalog, order management etc. and always ready to act on new feedback. It would only require the decision of the team's product owner and their business counterpart to divert some team capacity to act on new information. This ability to quickly reorient a ready team is the first component of better responsiveness.

Don't run it like a city

Projects-mode is similar to how cities are run. A city has to keep the lights on, so to say, in various departments such as water supply, sanitation, transportation, law and order etc. The people that work in these departments are the equivalent of our ops (run) teams. On the other hand, when new water or sewage treatment plant is to be built or a new metro line is to be introduced, the city funds them as projects and contractors execute them. They are the equivalent of build (change) teams.

In contrast to the "We build it, we run it" ethos of modern software teams, a city's approach is "You build it, we run it" and it works for a city although its citizens might prefer quicker improvement in various areas. On the other hand, unlike the case of software, the skills required for building/changing a city's infrastructure and services are vastly different from the skills for running them. It might not be cost-effective for a city to pay for always-on change teams. The projects model is cost-effective for cities, but they do pay the price for sluggish performance when they lose out on business investment to other cities.

Traditional enterprise IT has followed the two-pronged city model of permanent "run" teams and on-demand "change teams" i.e. the "projects" approach, and have paid the price in terms of responsiveness to the business. Digital-age business can ill afford to be sluggish in responding to feedback from business stakeholders or the market. Business-capability aligned teams need to have the level of funding, staffing and authority to switch focus or pivot their approach to a problem on the fly.

Reduced End-to-End Cycle Time

Cycle time is a common measure of responsiveness. Included inside overall cycle time is the time to reorient and the time to execute. Let's examine the latter component now.

Most enterprise DevOps initiatives don't alter the separation between the "change" and "run" organizations at all. Usually, both organizations adopt some new tooling, some deployment and infrastructure automation, perhaps with some "cloud" thrown in for good measure. They may even realize benefits such as more frequent, automated and reliable deployments.

However, due to persistence of the handoff from dev to centralized ops, the end-to-end cycle time of delivering a change doesn't change much. Product-mode teams don't suffer this handoff when they deploy and run what they build. They can therefore achieve the full potential of DevOps adoption. Colocation of application level ops with dev also promotes better end-to-end iteration and better understanding between developers and site reliability engineers.

Note that product-mode teams usually continue to rely on other specialist ops teams that provide a self-service build & deployment infrastructure, manage data centers and/or take care of non-application level security. These teams may also be run in product-mode even though they are not business capability aligned.

Ability to Truly Iterate

COOs often complain about the cost of the tech organization. Yet, they miss one of the biggest opportunities to save money because its root cause is outside tech. Big ticket projects are conceived and funded at one go without an effective mechanism for validating actual benefits. These projects regularly miss the mark in terms of delivering benefits. If organizations had the habit of tracking their benefits realization ratio (actual benefits/projected benefits), they'd be shocked in most cases. A truly iterative development process would be able to fail cheap and save money.

Although the notion of iterative development has been around for a while, in practice, iteration only goes as far as a sprint showcase in most cases. Most Scrum teams are bound to scope-based sprint commitments and are expected to deliver scope on plan rather than solve problems. In order to solve a problem iteratively, teams need to be able to respond to feedback (usage analytics, user surveys, stakeholder testimony) of earlier versions of the solution. Product-mode teams, being stable and long-lived can choose to solve a problem in a truly iterative manner.

Project teams can't easily move out of scope-delivery for two reasons. First, they are usually only meant to "build" the solution, not run it. So, they build a version of the solution over one or more releases, hand it over to a team in the "run" organization and move on to the next project. Second, the way project funding works, funds are made available to sustain a team for a duration of time much shorter than the lifespan of the underlying problem being addressed.

Example: A retirement calculator

For example, in a financial services company, a project was approved to develop a retirement calculator that would nudge prospective customers into buying retirement products or improving contributions in existing plans. The project team was funded only for the time it would take to build the calculator, not address the underlying problem, which was to improve the uptake of retirement products. Another team would embed the calculator on the company website and a third team would set up the necessary digital campaign to drive traffic to the calculator. At least three different managers (supervisors) would closely track the completion of work with a business stakeholder operating at arm's length after having specified what to build or at best, having participated in a day-long inception/hothouse. If the delivered solution missed its mark in the market, it would have to wait for another round of funding and staffing to find its way into an improved solution.

Adapting it to Product-Mode

How would a product-mode org cater to the situation above? Firstly, we wouldn't think of spinning up a new team to develop the calculator. Instead, it would be taken up by a pre-existing, stable, long-lived team closest to the problem area (i.e. product-mode team). What product-mode teams are we likely to have in this case? Given that this line of business sells a bunch of retirement products, could we have one product-mode team per retirement product? Unfortunately, it is not realistic to have a team per retirement product even if they make sense from a customer perspective because of the commonalities in business process & logic, and the shape of existing systems. On the other hand, a single product-mode team for all retirement products would be too big and unmanageable.

Instead, we'd have several business-capability aligned, product-mode teams. Here's one way of partitioning the long-term customer journey with respect to retirement products: on-board companies, enrol employees, save for retirement, withdraw etc. Note that each partition is customer-centric and is a meaty slice with data and business logic distributed across multiple systems. The whole customer journey is too big to be owned by a single team. A product-mode team therefore exists for each partition (business capability). This is somewhat analogous to the earlier online retail platform example where the overall customer journey is partitioned as catalog, order management, payment and fulfillment.

Now, any work on retirement calculators, being an attempt to improve enrolment or savings, would fall into the remit of the enrol or save team. This team, by the very nature of problems it generally addresses, would typically have digital marketing skills and access to marketing systems and assets within the team. The team would now have the responsibility to deliver and prove an improvement in conversion that is attributable to the calculator. As a long-lived team, it would have enough time to iteratively develop and test the effectiveness of the solution whilst attending to other roadmap items during lulls.

In terms of cost and time per percent increase in conversion, a product-mode setup would, in all likelihood, compare favorably to a project-mode setup. The only way to verify this reliably for any given organization is to try product-mode for a couple of problems and assess the results.

Knowledge Retention

The truth about software is that no amount of documentation, handover, and knowledge transfer can make up for a 100% churn in team. Yet this is exactly what a project model brings about. Tech capability does not reside in maturity models, process templates, documentation, code, or the infrastructure. It resides and grows in a team.

Knowledge grows and stays better within stable, long-lived teams that work in the same general area for several years. This contrasts with project teams that ramp up and down every few months. *Unmaintainable code results—at least partly—from unmaintained teams.*

Although documentation is useful and necessary, in organizations that prefer “working software over comprehensive documentation”, it is no substitute for the sort of knowledge retention that occurs within product-mode teams.

Product-mode teams allow team members to focus on a given area (business-aligned capability) for much longer than a typical project duration. This helps build knowledge, improves understanding of problems and trade-offs, and enriches the quality of interaction with business SMEs and stakeholders.

Do product-mode teams lose knowledge when a team member leaves? Not if they work based on collective ownership of overall scope. Knowledge retention is at risk when team members individually own different features or sub-areas.

Architectural Integrity

The incentives involved in project-mode put pressure on teams to neglect medium-term architectural integrity in favor of (often perceived) short term feature delivery. Since the team doesn't face the consequences of that trade off, they don't benefit from the feedback loop that appears when there is longer term ownership. Project teams are known to contribute (inadvertently) to architectural debt by being unaware of enterprise architecture guidelines or simply ignoring them in the narrow pursuit of timely project delivery. In the medium-term, this phenomenon compromises stability of systems and degrades speed of delivery. A couple of examples:

- They may end up performing a low effort integration with a system meant to be sunsetted instead of taking on a higher effort integrating with an upcoming successor. This compromises the sunsetting roadmap and increases the cost of maintaining an application landscape with multiple systems that do similar things.
- They may end up performing direct database integration with a downstream system instead of integrating after exposing its capabilities via API. In turn, this hurts any evolution of the downstream system that requires a breaking change to its database schema.

Product-mode teams, on the other hand, don’t let other teams integrate inappropriately with the systems they own as they are more keenly aware of the consequences. That’s the advantage of having “build+run” teams own systems as opposed to projects-mode where run-only teams own them.

Ownership of Code and Systems

Team level ownership of code and systems greatly helps a team's ability to operate in ideate-build-run mode. Ownership allows a team evolve systems faster and in a more controlled manner. But what about the ideal of collective ownership? It is great to aim for collective ownership within teams since the alternative of individual ownership is problematic. Between teams though, a clear demarcation of ownership allows for accountable autonomy.

An internal open source model is used at some of the most advanced tech companies. Anyone in any team can propose changes to almost any area of code by sending a pull request to the custodian (similar to core committer team in open source projects) team.

Although this model can co-exist with team level ownership, it tends to create a default expectation that dependent teams will self-service the changes they need. It is a common expectation in open source projects ("Found a bug? Great. Please submit a fix.") but unrealistic at most mainstream enterprise Digital/Engineering/IT departments. For one, there is seldom the level of domain knowledge to effectively make changes to disparate systems. Besides, it requires all code to be maintained at a self-serviceable level of documentation, build (as in, simply checkout and build) and test readiness—a high bar for most organizations. It is one reason why internal open source initiatives have failed to take off in most mainstream enterprises that have tried it. For such organizations, it is pragmatic to keep the default expectation not as one of self-service but rather that of a negotiated dependency. Any change must by default, be agreed with and prioritized and delivered by the owning team.

Team Motivation and Dynamics

Teams take time to work together effectively as a unit. Tuckman's theory of group development asserts that a team goes through a series of suboptimal phases (forming, storming, and norming) before it finds its groove in the performing phase. A project model of staffing runs the risk of disbanding teams just as they enter the norming or performing phase. Product-mode teams, being stable and long-lived can benefit from a long performing phase.

Studies have identified autonomy, mastery, purpose and belongingness as key intrinsic motivators. Product-mode teams tend to have greater autonomy and purpose than typical, scope-delivery project teams. Therefore, the introduction of product-mode teams is usually welcomed by the troops even as management takes its time to adjust to a new normal.

Economies of Flow and Iteration

> *In product development, our greatest waste is not unproductive engineers, but work products sitting idle in process queues.*
>
> *-- Donald G. Reinertsen*

The arguments so far may sound contrary to conventional IT wisdom. But the business landscape has changed dramatically, roughly since the time smartphones hit the market. Although it has never been sensible to treat IT/engineering as a cost center, it is indefensible now. It has never been more counterproductive to manage IT with a primary focus on cost-efficiency. It is not even enough to create a high-speed “digital” front-end organization on top of sluggish back-end IT—Bimodal/Two-speed IT is based on a false premise and its perpetrators have begun acknowledging it.

Conventional IT wisdom is outdated on several fronts. One such front is the norm of organizing IT teams for economies of scale to maximize utilization of IT specialists. In some ways, a project-based operating model reflects this norm by, one, the use of temporary change teams and two, the use of shared services for design, testing, deployment etc. On the other hand, economies of flow and iteration matter more to the goal of responsiveness.

Economies of scale come about when the unit cost of processing reduces with an increase in scale of processing. On the other hand, economies of flow come about when the unit time of processing (end to end cycle time for a batch size of one – single piece flow) reduces with an improvement in flow of processing. Build-only project teams supported by lots of shared services may help achieve economies of scale whereas ideate-build-run product-mode teams help achieve economies of flow.

Similarly, economies of iteration come about when the effort need to achieve business benefit reduces with an improvement in the ability to iterate with small changes. Build-only project teams suffer too many handoffs to iterate effectively. Besides, they don’t live long enough to carry out a sustained series of real feedback based iterations. Product-mode teams, by design, don’t suffer these disadvantages.

Challenges of Operating in Product-Mode

Staff Utilization

A common concern goes like this: “What happens when a product-mode team runs out of work?”. This never happens if the team is responsible for a big enough area like catalog or fulfilment. Besides, team sizes are reviewed periodically (e.g. once a year) to adjust for changing, relative priorities. Some places use a core-plus-flex model of a core team of employees and a flex top-up with contractors. Still, if a given business capability area does not have a robust or important enough roadmap, it is folded (along with a core team) into a neighboring capability until the next review.

Sometimes, the underlying question is: “What if we (the PMO or equivalent) don’t prioritize anything in a given team’s area of work?” And the answer is that a product-mode team is meant to be an empowered, semi-autonomous unit and therefore, not entirely dependent on prioritization from outside. Only cross-cutting initiatives are prioritized externally. For roadmap work, team level product owners are empowered to prioritize in concert with business stakeholders. A rule of thumb is to aim for twothirds roadmap work and no more than one-third initiative work. Portfolio management thus undergoes radical change in product-mode.

What about utilization of different roles in a cross-functional team? What happens when there isn't enough work to keep all the different roles busy? Well, people take on work that is adjacent to their skills, e.g. a business analyst may perform exploratory testing, a QA might help improve documentation, etc. In the beginning, they may have to shadow others to gain an initial level of skill but soon enough they go from being a specialist to being a generalizing specialist.

While on the topic of utilization, it is important to note that optimizing for utilization is detrimental to reducing end-to-end cycle time. You don't improve the flow of traffic on a highway by improving its utilization (i.e. introducing more vehicles). A predominant focus on utilization is a relic of "IT as cost-center" mindset. It is out-of-place for a digital business.

Insularity

Aren't stable, long-lived teams a recipe for insular attitudes and general stagnation? It is a risk but usually, there is always some churn in tech teams and this provides some injection of new thought. Besides, many organizations sponsor communities of practice—informal, internal networks aligned to specific competencies such as customer experience, A/B testing, etc.—that bring people across teams together.

New silos

In projects-mode, we encounter silos such as product management, architecture, design, development, testing, operations and support. In product-mode, will we end up with new, business-capability aligned silos such as catalog, order management, payment and fulfillment? Perhaps, but they are nowhere as bad as activity-oriented silos. Product-mode teams, being outcome-oriented, may at worst result in silos that still work towards a business outcome. Silos aligned to business capabilities are not as bad as silos aligned to stages of a product development value stream.

That said, we can mitigate the risk of new silos somewhat by setting up product-mode teams along business aligned capability boundaries defined by service experiences such as "issue to resolution" or business value streams such as "order to cash". If this gets too big for a single team, then we segment into multiple teams (squads within a tribe, in terms of the naming schema made popular by Spotify) but try to keep them co-located and appoint a service experience champion or business value stream owner to work with segment level product owners.

As a countermeasure, for initiatives that cut across multiple product-mode teams, it is recommended to appoint silo-penetrating, priority-negotiating, dependency-managing solution champions who work with product owners and business stakeholders. Another good practice is to offer people an option to change teams after a longish stint, say 18 to 24 months.

Ultimately, silo-busting in product-mode is about ensuring alignment along with autonomy. The old school approach achieves alignment by sacrifing autonomy. Instead, we use techniques such as alignment maps to get the best of both worlds.

Other Considerations

So far, we've looked at product-mode through the lens of benefits and challenges. This final section address a few other common considerations.

Tiered Teams

The online retail example described earlier suggested product-mode teams such as catalog, order-management, checkout and fulfillment. The retirement products example suggested product-mode teams such as on-board, enrol, save and withdraw. This brings up the question whether these teams have end to end responsibility with respect to the enterprise application stack, i.e. from market-facing front-end systems all the way to back-end systems that support back-office functions (Tiers 3 and 4 in the picture below). The answer is "not always" for reasons below.

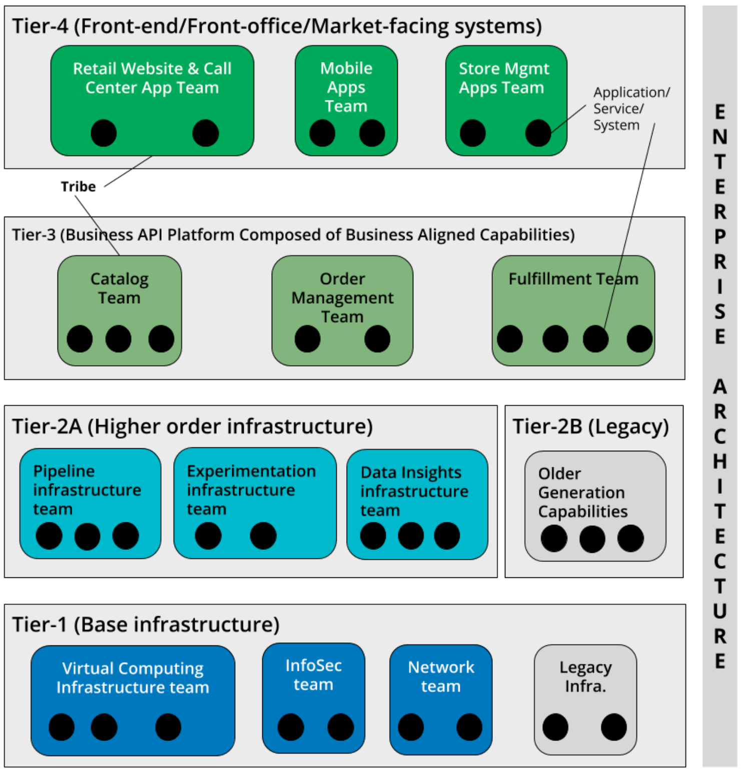
Tier-4 (Front-end/Front-office/Market-facing systems)
Retail Website & Call Center App Team
Mobile Apps Team
Store Mgmt Apps Team
Application/ Service/ System
Tribe
Tier-3 (Business API Platform Composed of Business Aligned Capabilities)
Catalog Team
Order Management Team
Fulfillment Team
Tier-2A (Higher order infrastructure)
Pipeline infrastructure team
Experimentation infrastructure team
Data Insights infrastructure team
Tier-2B (Legacy)
Older Generation Capabilities
Tier-1 (Base infrastructure)
Virtual Computing Infrastructure team
InfoSec team
Network team
Legacy Infra.
ENTERPRISE ARCHITECTURE

Figure 1: Simplified example of tiered, product-mode teams in online retail.

Even though end-to-end responsibility helps the cause of outcome-orientation, it is often unrealistic given the shape of systems. For instance, online retail platforms typically have, at a minimum, a webapp and mobile app as customer shopfronts, a call center app as a support front-end and a store-management app as another front-end for store managers. Typically, each of these front-end systems has it own dependency footprint on underlying capabilities such as catalog, order-management, checkout and fulfillment. Usually, there are no clear, front-end system boundaries for an end-to-end catalog (or any other capability) team. For instance, it might be possible to avoid a tier-4 store-management apps team if it is only dependent on a single tier-3 capability such as catalog. The picture shown assumes that store-management has dependencies on more than just catalog. Therefore, we resort to separate product-mode teams in tiers 3 and 4 as shown. Higher level tiers are consumers of lower level tiers.

A new trend is to decompose front-end systems into micro-frontends that line up with tier-3 capabilities. That makes it easier to have end-to-end teams that cut across tiers 3 and 4. But most places that have moved/are moving to microservices aren't yet planning a move to micro-frontends just yet.

Sometimes, the existence of legacy systems supporting multiple tier-2B capabilities may lead to a situation where a tier-3 team also depends on a tier-2B team with corresponding skills. In theory, it is possible to embed tier-2B people into corresponding tier-3 teams. In practice, however, this may be constrained by scarce tier-2B capacity or by practical challenges of shared configuration management, test environments, etc.

Tiers 1 and 2A are enablers for continuous delivery, DevOps, and Lean product development. They are more applicable across business domains. For example, the pipeline infrastructure team is responsible for maintaining Jenkins/GoCD/TeamCity... infrastructure. They are not responsible for attending to broken builds of higher tier teams. Thus, the higher tiers may have one or more dependencies on lower tiers. Irrespective of tier, each team is product-mode in the sense that they are stable, long-lived, cross-functional, outcome-oriented, ideate-build-run teams. The lower tier teams treat the higher tier teams as their (internal) customers in a problem-solving (not scope-delivery) manner.

Squads and Tribes

Often, a single area like catalog may be too big for a single team and it’s okay to break it up into multiple product-mode teams. In this case, the catalog tribe has multiple squads, each with its own outcomes, roadmap and systems under custody. This structure also allows local sharing (internal to tribe) of roles that may be hard to embed within each squad. Note that squads are meant to be much longer lived than typical project teams although squad members may be deliberately asked to rotate within the tribe at say, 18 to 24-month intervals in order to gain breadth of knowledge and avoid becoming single points of failure.

But where is the Product?

To reiterate a point made in the beginning, it is not necessary to be building products to operate in product-mode. That said, if we go beyond the conventional definition of a product as something that's sold in the market, then it possible to look at things like catalog as if it were a product. Once a capability like catalog is contained within a set of systems owned by a team and exposed via internal applications or well-documented API, it becomes a reusable and self-serviceable, product-like capability catering to internal customers. The same applies to lower tier product-mode teams.

Conclusion

In sum, for greater responsiveness and a higher benefits realization ratio, "product-mode" is a more effective way of working than projects.

For mainstream enterprise Digital/Product/Engg./IT departments, it may feel like radical change and it is to some extent. However, it only feels radical because we are so steeped in the "projects" way of doing things. Amazon CTO Werner Vogels, described a similar approach when they adopted in 2006:

> *In the fine-grained services approach that we use at Amazon, services do not only represent a software structure but also the organizational structure. The services have a strong ownership model, which combined with the small team size is intended to make it very easy to innovate. In some sense you can see these services as small startups within the walls of a bigger company. Each of these services require a strong focus on who their customers are, regardless whether they are externally or internally.*
>
> *-- Werner Vogels*

To migrate to product-mode, it is best to adopt an iterative and fail cheap approach. Start with a pilot or two, learn and adapt. Although it may feel unsound to those who are used to approving big change programs with detailed roadmaps, it is the essence of a Lean-Agile mindset to avoid overinvesting before validating actual (not projected) benefits.

Acknowledgements

A big thanks to all reviewers including Martin Fowler, Evan Bottcher, Duncan Stephens, Brandon Byars and Kevin Yeung. Additional, special thanks to Martin for his help and guidance with editing and publishing.

FAQ

How do we address security and regulatory compliance in product-mode?

By itself, product-mode does not require a very different approach to these concerns as compared to projects. That said, product-mode discourages a complete separation of these concerns into specialist shared services. Through an internal consulting approach, security and compliance specialists enable squads and tribes to gain a measure of competence in these areas while continuing to provide an overarching review/audit/control function. Although not directly required by product-mode, the demands of responsiveness encourage a move to automated audits. Please refer to this deck for details.

In keeping with Bimodal/Two-speed IT, can’t we run the digital org in product-mode and the IT org in project-mode?

The Bimodal/Two-speed prescription is based on a false premise that you can either build software in an agile manner or in a reliable manner, not both. It misses the point that agile engineering techniques achieve speed through reliability.

Why talk about “Products over Projects” in an age of Platforms?

Again, this article is not about building products. It is about organizations and teams working in product-mode rather than project-mode. Through this way of working, they often create platforms, internal or external, where it makes sense for the business. For example, the illustration of tiered teams has the third tier developing an internal business API platform for cross-channel consumption at tier four.

Shouldn’t we be organizing for customer-centricity rather than for product/project centricity?

By setting up teams to solve problems rather than just delivering scope, product-mode enables greater customer/market/business centricity than projects.

How does product-mode work with outsourced teams?

In product-mode, there are no projects to be outsourced to vendors. Instead, vendors may be asked to supply a stable, long-lived, ideate-build-run team with some key roles staffed with in-house talent.

How does this compare with feature teams?

Feature teams typically aren’t ideate-build-run teams. They are build-only. They build one feature after another without too many build-time handoffs/dependencies with other teams. The features may not be in the same sub-area of business domain. Knowledge preservation is at risk because although the team itself may be long-lived, their association with an area of business domain is not.

How does product-mode sit with Conway's Law?

To the extent that the communication structure of an organization influences system design, we might begin to see architectural boundaries develop/harden along the lines depicted in the tiered-teams illustration, even if such boundaries didn’t already exist. Operating in product-mode might therefore facilitate a move away from a monolithic enterprise architecture.

What do product-mode teams do once they develop a product/application and it's mainly maintenance after that?

“Build once and maintain thereafter” is old school. It results in products that miss the mark i.e. wasted investment. Product-mode is best suited for ever-evolving products/applications/ capabilities/platforms where development and maintenance go hand-in-hand until it is marked for end-of-life. That said, if we ask the same question at “feature” granularity, then the answer is that they take up other problems (to be addressed via new/enhanced features/experiences).

Can product-mode work where ownership of systems is constrained by large, monolithic systems?

Monoliths may have to be owned collectively (for build and run) by a number of squads or even tribes. Sometimes, it is possible to assign temporary ownership (e.g. for a year) if the overall pipeline of work indicates that a particular squad or tribe is going to have to make changes to the monolith a lot more than the others. During this period the owning team will service incoming dependencies. In some cases, shared ownership of “run” may be problematic and we may have to assign some “run” functions to a separate, dedicated ops team until the monolith is decomposed into smaller systems.

How do we address technical debt in product-mode? What if empowered product owners choose to prioritise functionality all the time?

Product owners are expected to take the advice of tech leads and enterprise architects to prioritize work on technical debt. They may choose to temporarily delay tech debt work in favor of functionality but ultimately, they (and the team) are responsible for solving problems in a stable manner. If tech debt is unaddressed, problems are bound to resurface in some shape or form. Organizations that have a history of difficulty in striking a balance could use decision records to bring some rigor to the process.

Does product-mode require a different style of governance?

A transition from projects to product-mode is also meant to be a transition from scope-delivery teams to problem-solving teams. Correspondingly, we govern for value over predictability. The ratio of people adding value to those tracking value goes up in product-mode. Pull-based delivery techniques that emphasize flow over utilization are more relevant for product-mode.

Does the need for a unified reporting hierarchy mean that product-mode teams ultimately report into the business?

Not necessarily. Think of tribe level product owners as software product owners who work with one or more business product owners/managers (who own P&L or report to those who do). Software product owners ultimately report into a CTO, CPO, CDO or CIO.

Does the ability to capitalize software development costs diminish by moving to ever evolving software development in product-mode?

This is a common misconception and the answer is quite the opposite. The early and constant focus on business-value allows for more effort to be capitalized than a traditional, linear approach. For example, please refer to this post.

Do teams still enter timesheets?

Although product-mode is about funding stable, long-lived teams rather than projects, we may still want to track the effort expended towards solving a given problem. Therefore, time tracking may still be necessary, but timesheets are not the only way to do it. For example, it could be done using agile project management tools by setting them up to report story transition times. Please refer section 9.4 of the book, Agile IT Org design for details.

What are the exact steps to make the change?

That's beyond the scope of this article and FAQ but at a high-level it starts with getting top executive buy-in in tech and business. It also helps to understand that despite some initial planning and guidance from experts, real change progresses in a learn-as-you-go manner.

Are ideate-build-run teams a departure from Google's model of dedicated SRE teams?

Yes and no. Ideate-build-run teams can co-exist with tier-2A SRE infrastructure teams that also provide SRE consultancy on demand. Björn Rabenstein and Matthias Rampke describe their experience at SoundCloud in the book, Seeking SRE. The takeaway is that at less than Google/Netflix scale, it can be cost-prohibitive to staff dedicated SRE team(s) that can effectively cope with the variety of domain-specific alerts from the application landscape.

Don't we need to focus on IT effectiveness before adopting Product-Mode in order to steer clear of the alignment trap?

IT effectiveness is a must but it is also a moving target. If, as of 2018, you still have the luxury of doing it in sequence, then do so by all means. Another approach is to go all out on the IT effectiveness front and go depth-first (vertical slice) with Product-Mode.

▶ Significant Revisions