Developer Experience for Software Platforms: Short Case Studies and A Few Key Lessons

Monojit Basu[1]

TechYugadi IT Solutions & Consulting, Bangalore

Abstract

Software platforms enable developers build end-user applications (both B2B and B2C) through a cohesive ensemble of frameworks, tools, workflows, programmatic artifacts, etc, offered on premise or as Platform-as-a-Service. The success of these platforms in terms of adoption, and direct and indirect revenues, is primarily dependent upon the developer experience, which is a loosely defined concept worthy of more research, from a product management perspective. In this paper, we search for clues on what could be some of the key elements of developer experience that make a platform successful. While not trying to be prescriptive, we first bring out short case studies on four platforms in areas such as Application Programming Interfaces, data analytics, interpretable Artificial Intelligence and Augmented Reality, highlighting product management principles that seem to be working in their favor. Thereafter, we assimilate a few generic strategies, that should help craft and refine the developer experience of a platform. We focus on the facets of ease of development and productivity, and on the ability to offer a multitude of choices in terms of customization and integration with other software, organically and through an ecosystem. We also bring out, how important it is to convey the marketing messages effectively to a developer audience, and design a pricing strategy that leads to greater platform adoption. We believe that this exercise will help platform product managers make a better assessment of the state of developer experience for the platforms they own, and formulate both strategic and tactical initiatives for driving business through developer mindshare.

Keywords: developer experience, ease of development, productivity, customization, ecosystem, marketing messages, pricing strategy, on-premise, Platform-as-a-Service

Introduction

Software Platforms enable us to develop other software products for end users, like Android, on which developers build myriads of mobile apps. Typically such platforms consist of a cohesive bundle of frameworks, tools and workflows that facilitate development of a target set of applications. There could

[1] Email: monojitbasu@yahoo.com

be broader definitions bordering on a catch-all phrase like 'anything that you can build upon', but in this paper, we will focus on the concept of a software platform mentioned above. Product management for software platforms does have a few significant differences compared to end-user products, as highlighted later, in this paper. Platform product managers have to help developers who build on the platform, be successful, without losing sight of end-users for whom they develop. This is where the developer experience comes into focus, in contrast to end-user experience for other software products. Unlike user experience design, which is now based on well-researched methodologies and best practices, not enough standards exist for developer experience. But a strategic approach towards developer experience is needed because of the phenomenal number of platforms being released to market, including those offered on the cloud (known as Platform-as-a-Service, or PaaS). Secondly, most end-user applications too, have platform components targeted at developers who want to build on it. For example, Zoho, primarily a Cloud software suite for business, that includes Customer Relationship Management (CRM), Human Resources Management (HRM) and Accounting, also offers 'Zoho Developer Console', to build applications on top of the Zoho suite.

In this paper, we endeavor to study a few software platforms that have made developer experience a cornerstone of overall product strategy, and edify a few key principles that product managers can keep in mind, in deciding:

- Which are the essential elements of developer experience relevant to the product they own?
- How to measure, improve and differentiate on those parameters of developer experience?
- How to incorporate the above tasks within the overall product management practices?

By no means will we attempt to define all facets of developer experience exhaustively, or an all-encompassing process that applies to every platform. The diversity of platforms and developer communities possibly makes such an exercise intractable. But our research would be valuable in stimulating the right kind of thought process for product managers to offer better developer experience, that contributes significantly to the overall success of their platforms.

In order to motivate our study, we first allude to some of the unique challenges of managing developer experience and expectations compared to end-user experience. Then we present a few short case studies of successful software platforms and analyze how some of these challenges have been addressed. Finally, we present a debriefing based on these studies, on how product managers can bring in a more structured and well-planned approach to creating better developer experiences.

Research Methodology

This paper is primarily based on practical insights gained from developing products and solutions in a wide variety of technology domains and industry verticals. Four platforms were carefully chosen for a more detailed analysis, in the form of short case studies, whereas, several other platforms have been included in the discussion, for illustrating relevant product management principles. The platforms chosen for case studies are as follows:

Platform	**Technology Domain**	**Proprietary / Open Source**	**Free / Paid**	**On-premise / Cloud-based**
Postman	API Management	Proprietary	Paid (Free basic version)	On-premise tool backed by cloud-based repository
Pentaho Data Integration	Data Analytics	Open Source and Proprietary	Paid Enterprise Edition	Both
Captum	Interpretable AI	Open Source	Free	On-premise
Unity	Augmented Reality	Proprietary	Paid (Free basic version)	On-premise

Table 1: List of platforms discussed through case studies in this paper

Figures / diagrams / screenshots have been adduced to these case studies, based on publicly available technical documentation, and from the author's own software development workspace. Some of these artifacts are embedded in-line with the case studies, whereas a few others have been appended at the end of the paper, as Exhibits.

It is clarified that the case studies have been prepared to simulate a cogent managerial approach to solving pertinent challenges in product management, and not to express any opinion on effective or ineffective managerial response to a business situation.

Other platforms mentioned in the rest of the paper include Amazon Web Services Machine Learning platform, Android and iOS platforms, Atlassian Software project management platform, Digital Ocean Cloud Platform, ERPNext, Selenium test framework, SoapUI, Cloud Foundry and Heroku PaaS, PayPal payment SDK, Zoho Developer Console, etc.

Some of the conclusions have been derived from the author's own experience in product management and product marketing.

Different Facets of Developer Experience

In order to appreciate the role of product management in crafting a smooth developer experience, we list below, a few representative examples of what constitutes developer experience.

- *Generic Application Development Framework*: This is a common scenario wherein developer experience encompasses platform capabilities to scaffold a starter application conforming to an underlying framework (e.g., Android), and enhance, build, test and deploy it, on premise or on cloud.
- *Customization*: For example, a general-purpose ERP could be customized for an industry vertical like Retail, or individually for each single enterprise customer.
- *Extension*: This involves developing add-ons, plugins, and utilities. For example, a Business Intelligence platform may offer a plugin mechanism to code up a connectors to data sources that are not already supported out-of-the-box.
- *Integration*: Most platforms provide APIs for integration with other systems, e.g., an IoT platform may be integrated with a Retail ERP for better logistics and inventory management.
- *OEMing*: It may be necessary to embed components into a larger product, like lightweight versions of databases that can be installed as part of a mobile app, preloaded with data.
- *New Framework Development*: In a few cases, the need for a new application framework may be felt, on which the platform can be developed more elegantly. ERPNext, a platform for small and medium businesses, was developed on a purpose-built web application framework called Frappe, written in Python. The framework and the overlying application both evolve over time.

The key to crafting a differentiated developer experience lies not just in the individual features of frameworks and tools. Developers implicitly expect an elegant software development process baked into the platform through the collection of features.

Unique Challenges in Managing Developer Experience

Combinatorial Explosion: Developers are technically advanced with diverse preferences in terms of the programming languages, tools, Integrated Development Environments (IDE), interfaces for integration, etc., some of which change more frequently compared to end-user preferences. A platform product manager with finite resources, has to bet on a small set of developer preferences that can lead to maximum developer adoption in a short time, and may go wrong occasionally.

Inter-organizational Development Resources: Supporting diverse developer preferences could lead to overshooting of development, maintenance and support costs. All platform vendors need to nurture an ecosystem that complements its own development team. However, for product management, this brings in additional complexity in terms of co-ordination across organizational boundaries, synchronizing releases, and in maintaining quality and customer support standards.

Specialized User Interfaces: End-user experiences for business software and consumer applications today are predominantly browser-based or mobile-app based. In contrast, developer experience is often comprised of thick or fat user interfaces like IDEs, debuggers, profilers, test-bench, etc., and command terminals, which are not browser or mobile-friendly. Their design and implementation requires specialized skills in both user experience (UX) and development, that are somewhat scarce compared to web and mobile application development. Developer experience involves programmatic interfaces too.

Redefining User Interfaces: Sometimes the 'right' developer experience for a platform has to be defined ab initio. Initially cloud computing vendors had to design a suitable interface to securely develop, build and run applications on a remote PaaS. The Heroku console defined a new developer experience, emulating the familiar Unix shell in a cloud-native environment, and other vendors followed.

Integrated Marketing Communications (IMC): Marketing communications for a targeted developer audience have to be carefully crafted and backed up with credible and unbiased technical content. Some messages may have to be consciously pivoted on developer productivity instead of business value.

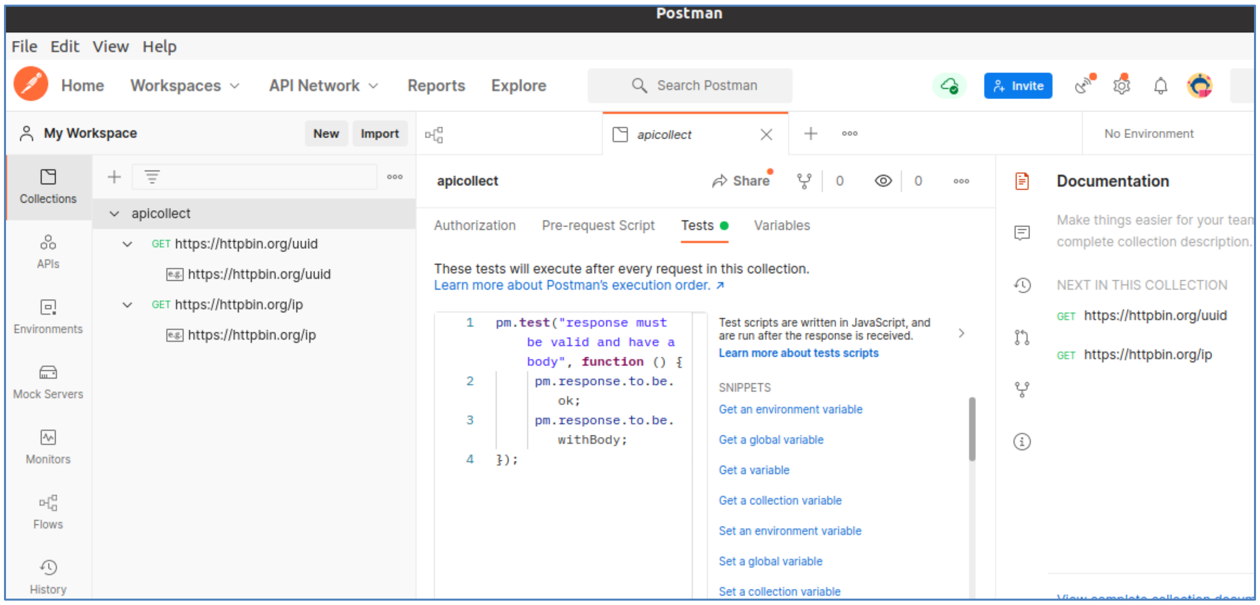
Monetization: The investment made in ensuring a smooth developer experience may or may not be directly monetizable. Most PaaS vendors do monetize directly through pay-as-you-develop pricing strategies. In contrast platforms like Android and Chrome browser are free and indirectly monetized. iOS offers a mix of free and paid components, and also monetizes through the underlying operating system.

Case Study: Postman API Platform

Postman is a one-stop developer platform for designing, documenting and publishing Application Programming Interfaces (APIs), and managing their full life-cycle. API life-cycle includes collaborating with other developers during the process of defining APIs, testing and monitoring, and continuous integration (CI) with a larger software stack.

Unmet Need and Re-imagination: APIs are like contracts between two different software systems or even two sub-systems in an IT system. The contract is realized through permissible requests and expected responses passed between the (sub-)systems over a network protocol (typically HTTP). Today APIs enable not just software systems, but businesses, to work together, like payment APIs and APIs for participants in the travel industry. Mobile apps depend heavily on APIs exposed by backend systems. Besides, modern architectures for large, complex systems, like microservices, are based on internal APIs defined between smaller units. Therefore, these contracts, or APIs are being defined in enormously large numbers across IT projects and enterprises. There is a need to define and evolve these contracts independent of the (sub-)systems themselves, somewhat like data models or user interface wireframes. This is called an API-first design, which allows the API producer and consumer (sub-)systems to be implemented concurrently. Since APIs have broader outreach, collaboration within and across development teams while defining these contracts, is more critical.

But APIs are a more recent phenomenon compared to data models. So there had to be a first mover who would define the process of designing and testing APIs independently, and also build the tools required for this purpose. Postman, started as a hobby project around 2012, went on to become a full-fledged company around 2014, that neatly synthesized the pieces of this puzzle, into a platform of choice for millions of developers.



These tests will execute after every request in this collection.
Learn more about Postman's execution order. ↗

```
pm.test("response must
    be valid and have a
    body", function () {
  pm.response.to.be.
     ok;
  pm.response.to.be.
     withBody;
});
```

Test scripts are written in JavaScript, and are run after the response is received.
Learn more about tests scripts

Make things easier for your team complete collection description.

Figure 1: Postman API Platform at a glance (version 9.6.1)

Postman offers a developer tool to define API specifications and thoroughly test their request-response patterns, standalone, even before the underlying system is fleshed out (refer to Figure 1 above). Once these specifications are stabilized, they are grouped into collections, and published with documentation, on a cloud-based service, from where they can be easily discovered by potential API consumers. It seamlessly spans a wider range of life-cycle tasks than most other comparable tools, including, for example, validating API specifications as part of a continuous integration (CI) pipeline (see Exhibit 1c).

The base tier offers developer workspaces with unlimited design, development and testing of APIs, and is free, though it has reasonable limits on API mock server, monitoring and access to Postman cloud, and limits team collaboration to three developers.

Competitors: API testing per se, is not a unique feature of Postman. Products like SoapUI (supported by SmartBear) offered such capabilities, before Postman came into existence. But Postman probably succeeded in better and more comprehensive conceptualization of what an API life-cycle means, and helped developers appreciate the value of following this life-cycle. Being a new entrant, it could start from a clean slate with the modern RESTful paradigm, which some competitors had to reconstruct over tools originally developed for the bit old-fashioned SOAP-style APIs. Moreover, in the early days, Postman was offered as an add-on to the Google Chrome browser, through the Chrome store, which proved to be a more developer-friendly channel, leading to rapid adoption. Today it is an undisputed leader, and named Visionary in the Gartner Magic Quadrant for Full Lifecycle API Management, 2021.

Developer Adoption: Postman has enjoyed popularity among developers right from inception, but has transitioned from a steady phase to a sudden steep growth over the last two years (see Figure 2).

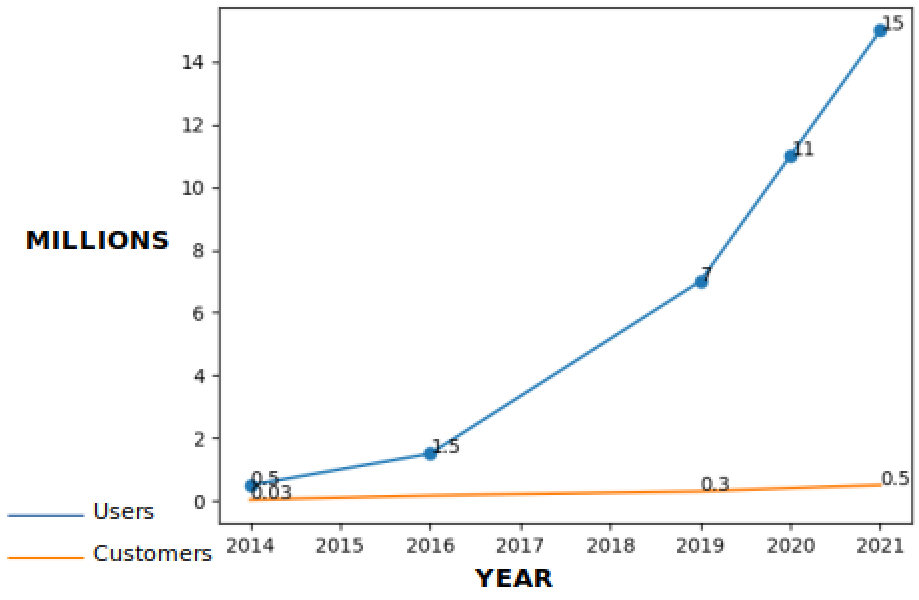


Figure 2: Growth in developer and customer adoption of Postman. Pieced together from data available from different websites like TechCrunch, inc42, VentureBeat, Toplyne, Forbes, and Postman Blogs, listed in References section.

Part of this growth has been fuelled by adoption across large enterprises, who are standardizing on the API life-cycle management process that this tool implicitly defines.

Platform Evolution: Newer capabilities of the platform are geared towards developer productivity and further penetration into the enterprise segment. A new feature being developed, called 'Flows', can significantly reduce code for testing and validation of a chain of APIs, by graphically composing interceptors or processors on the request-response path (see Exhibit 1d), and its beta version is already included in the platform.

An enterprise edition was released in 2020, particularly focused on security (user groups and role-based access control), reporting and integrations, with per-user pricing and any team size. A new feature called 'Partner API Network' is at the concept stage, for which, inputs are being sought from existing customers (see Exhibit 1e). This feature might help partners collaborate and define B2B APIs, in domains such as supply-chain, banking, insurance, travel, etc. Pricing changes introduced in 2020 and 2021 will enable phased upgrade from the free edition to the full-featured enterprise edition, through two added intermediate tiers (Basic and Professional).

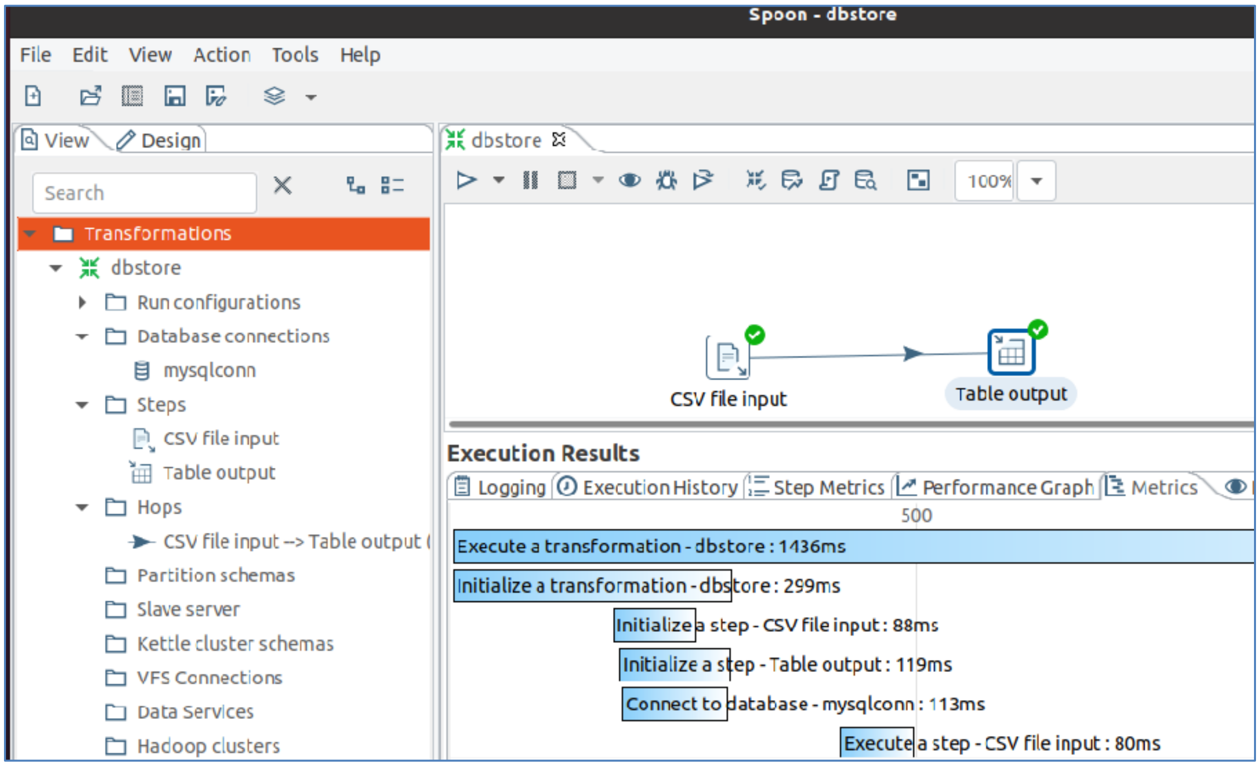
The Postman team is a contributor to domain-agnostic technical standards like OpenAPI, AsyncAPI and JSONSchema, as well as industry-specific initiatives like FHIR (in healthcare industry) and OpenTravel. These involvements will also support it in charting out a right strategy and platform roadmap.

Case Study: Pentaho Data Integration

Pentaho has a long history as one of the earliest and delightfully open source Data Integration and Business Analytics platforms, in existence since 2008. In 2015, it was acquired by Hitachi Data Systems, and is now part of the Hitachi Vantara Data Storage and Analytics suite. In this paper, we will focus on the Pentaho Data Integration tool only, but refer to related products where appropriate.

Data integration is the first step in the process of drawing out actionable insights from data. It involves writing scripts to Extract data from various sources, including databases, web services, messaging servers, social media, in structured and unstructured form (like text, images, videos, etc.). The extracted data is transformed, and loaded into specially architected data repositories for faster analytics (e.g., based on OLAP cubes). The entire process is referred to as ETL. Since this involves lots of repetitive tasks and boilerplate code, several players tapped into the opportunity to offer graphical interfaces to build an ETL job, using lots of pre-canned queries and scripts. Both the set of pre-canned artifacts, and the multitude of data sources, keep growing, especially with new non-relational databases, Big Data and social media platforms, and IoT applications involving machine generated and sensor data, emerging over the past decade.

Core Features and Extensibility: Pentaho Data Integration (nicknamed 'Spoon') is a thick client interface, built on top of the popular Eclipse framework, running on Java, that includes ETL Job design and execution palettes (Figure 3). For a project entirely driven by an open source community in its early days, Pentaho offers an amazingly large set of built-in ETL primitives, data sources and sinks. Exhibit 2a shows just a small set of transformation operations (there are large sets of input, output and other operations). The goal is to support most ETL steps through in-built drag-and-drop operations, while allowing users to patch in hand-coded snippets when unavoidable, following sort of 80:20 rule. Exhibits 2b and 2c illustrate a range of supported relational databases, and Big Data connectors in batch and streaming mode, respectively. That might still not satisfy every conceivable data integration requirement, but a well-designed plug-in framework is in place for churning out custom transformations and database connections. Eclipse is the common substrate on which many Java-based thick client interfaces are developed. Hence the extensibility mechanism for Spoon is quite intuitive for Java developers, as are the widgets and overall look and feel.



```
Execute a transformation - dbstore : 1436ms
Initialize a transformation - dbstore : 299ms
Initialize a step - CSV file input : 88ms
Initialize a step - Table output : 119ms
Connect to database - mysqlconn : 113ms
Execute a step - CSV file input : 80ms
```

Figure 3: Pentaho Data Integration developer environment (community edition, version 9.2)

Related Products: Once an ETL job is designed using Spoon, it can be scheduled, run and monitored through an administrative service (web application) called Pentaho Business Analytics Server. There is also a Report Designer for visualization of data loaded into an OLAP engine. These reports can be combined into dashboards that can also be served by the Business Analytics Server. There are a couple of other utilities: i. Schema Workbench, for designing OLAP cubes (sometimes referred to as Star Schema) before bulk-loading them through an ETL job, and ii. Aggregate Designer, that helps design and tune aggregate operations used in reports, for better performance (refer to Figure 4).

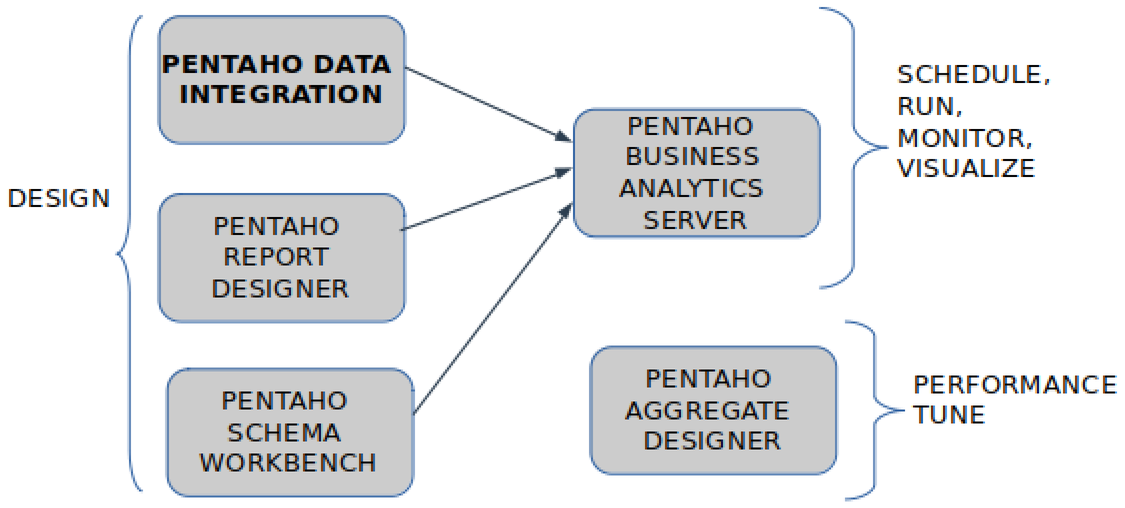


Figure 4: Pentaho Data Integration and Related Data Analytics Products

Most of these tools, and Spoon itself, offered annual subscription-based support for enterprise customers, on top of the open source platform. The nature of monetization has changed after Pentaho was acquired, although the commitment to open source remains.

Acquisition by Hitachi: Hitachi Data Systems has been one of the key players in the enterprise data storage market. Acquisition of Pentaho signaled an intent to capitalize on the Big Data opportunity, that had synergies with its core business.

Firstly, the Enterprise Edition of Pentaho has been revamped. It now supports load-balancing of ETL jobs, and tighter integration with Data Science libraries like R and Weka. It natively supports bulk-loading onto cloud-based OLAP repositories like Amazon Redshift and Snowflake. It comes with 24/7 technical support, mentoring, and an assigned architect. The Enterprise Edition appears to be aggressively promoted.

Secondly, after the acquisition, Hitachi Data Systems launched a few major products in the Data Analytics space, particularly, a Data Lake and a DataOps platform named Lumada. Both these products are built on top of the Pentaho Data Integration core. This also maps to enterprise customer journey and maturity in the data analytics space, where, they typically start with refining and perfecting their data integration (ETL) projects, and then look forward to greater return on data through advanced architectures and operational innovation. Hitachi is also invested in the Industrial IoT market, which has its own dependencies on data analytics, where the Pentaho tools can be leveraged. There has been some expansion in the storage market too, in terms of Object Storage offerings, and the portfolio was rebranded as 'Hitachi Vantara'.

Developer Community and Marketplace: Hitachi Vantara has an active developer community who contribute ideas, solutions, and add-on components. A market-place has been set up, which also offers a large collection of Pentaho plug-ins and connectors.

The market-place is well structured allowing for two types of contributions: i. 'community lane', for projects that are not meticulously reviewed for architectural consistency with the platform, and managed by the Pentaho Consulting Services, and ii. 'customer lane', for projects that align with the platform architecture and roadmap, and are carefully reviewed by the Pentaho development and product management teams. Contributions in the customer lane follow the same support policies as the platform itself. Both types of contributions are further differentiated based on stage of development, like development release, snapshot release, stable release and mature release, to help customers perform a proper evaluation before relying on them in their projects.

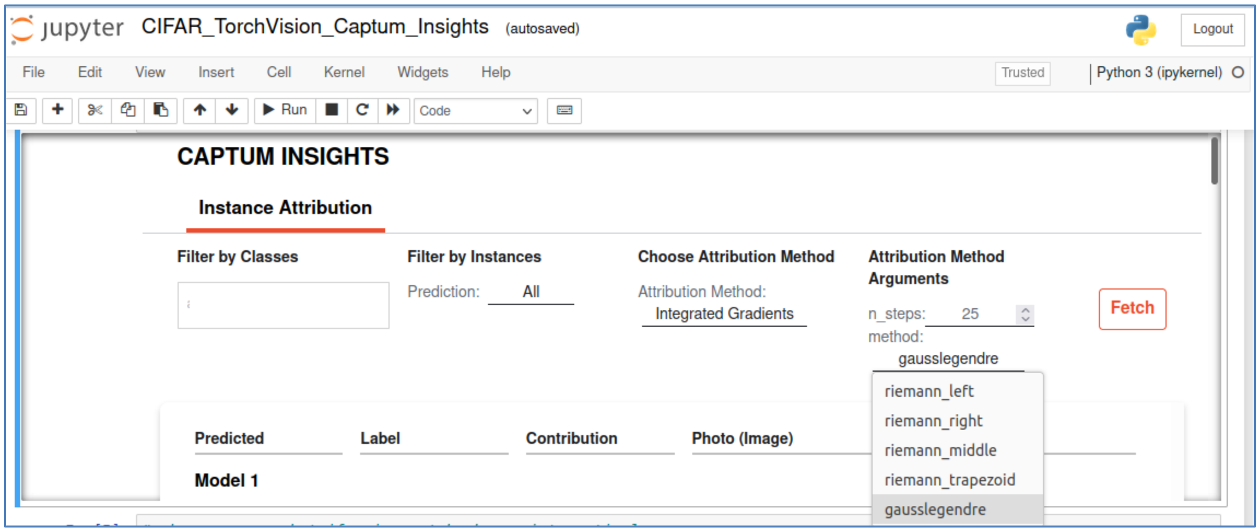
Case Study: Captum for Interpretable Deep Learning on PyTorch

PyTorch, written in Python, is one of the most widely used Deep Learning libraries, and is open source. It is being developed in Facebook AI research Labs since 2016, with contributions from a large community of developers and researchers. Today, we have a fair idea of which deep learning algorithms or architectures can be employed to solve a problem, as AI is being adopted across domains at rapid pace.

However, the algorithms and architectures are getting increasingly complex, and (somewhat akin to the human brain) it may not be crystal clear how a machine arrived at an inference, even if it was correct. This has serious implications in terms of reliability and biases of AI-based systems. The next challenge that the AI community is actively engaged with, is, interpreting or explaining results produced by a deep learning model. Explainable AI might well be the stepping stone to Responsible AI.

Captum (meaning 'comprehension' in Latin), also from Facebook, is a library designed to interpret deep learning models. A common pattern across most AI-based algorithms is that they look at data items as a collection of attributes (or features) with a label. For example, stock price is a label that depends on various attributes of the stock, like last quarter profits, attractiveness of the industry, reputation of the board, etc. Captum attempts to interpret models by applying a method called Integrated Gradients, that estimates how much influence each attribute in the data had on the overall model. For images, attributes consist of pixel values, and the attribution technique is sometimes called a Saliency Map, indicating, pixels from which locations had a major influence. There are a few other approaches to interpretability that Captum supports, which we will skip here for brevity. Mudrakarta et al (2018), assert that "*When a model is accurate but for the wrong reasons, attributions can surface erroneous logic in the model that indicates inadequacies in the test data.*"

In terms of developer experience, Captum is easy to get started with. The existing PyTorch program to generate a model from training data does not change. Instead of using it to make predictions on test or novel data, the model has to be simply passed into a Python function, that prints out attributions. This takes about five lines of code. Instead of printing attributions, a visual interpretation called Captum Insights can be produced (Figure 5), by choosing a slightly different function call. This indicates that the two libraries are tightly coupled, making it easier for developers already familiar with PyTorch.



Prediction: All

Attribution Method: Integrated Gradients

n_steps: 25

method: gausslegendre

Figure 5: Captum Insights for attribution-based interpretation of machine learning models (version 0.4.1)

This is an example of a developer experience built around a programmatic interface, mostly. Captum consists of a rich set of functions that combines well with the base PyTorch library, and supports numeric, image and text data. Considering that in production, most deep learning algorithms are run on GPU or other scalable architectures, Captum also allows scattering and gathering the computation of attributions, over multiple CPU instances.

Exhibits 3a and 3b show how one could make sense of both correct and incorrect image classifications produced by a PyTorch model, using Captum Insights.

Interpretable and Responsible AI is obviously a vast, open-ended problem, likely to need lots of research over many years. It was befittingly open sourced in 2019, after being used for several years internally.

Case Study: Unity Augmented Reality Developer Platform

The Unity Developer Platform is a workbench with a set of tools and workflows to author cross-platform Augmented Reality (AR) applications that can be run in mobile or desktop environment, and in wearables. Augmented Reality applications enable end users to interact with real world objects online, through graphical and audio-visual cues, for an immersive experience. Through persistent efforts to bring AR from research to mainstream, over the last two decades, we now come across wide-ranging applications of this technology in online retail, education, travel, healthcare, remote work, gaming and entertainment, and many other domains. The transition through the 'hype cycle' into a more mature technology suitable for commercial applications, has been remarkably rapid, over the last five years.

AR Application Development Tools: Many of these tools and frameworks are closely coupled with a mobile app development framework or operating system platform. Apple ARKit allows AR applications that receive inputs from camera and motion features of iOS-enabled devices, including iPhones. There are programmatic interfaces like Javascript-based 'ar.js', for developing in-browser AR experiences that are cross-platform, and do not need additional software installations. But there are technical challenges in producing end-effects of the same quality as native applications. A small sample of the variety of AR platforms is captured in Table 2.

Augmented Reality Development Platform	Initial Release
Unity	2017
Apple ARKit	2017
Google Android ARCore	2018
ar.js	2020

Table 2: A small sample of well known Augmented Reality application development tools

Note: Unity started as a gaming platform around 2012, that gradually evolved into a more full-featured platform for AR development (called Unity Hub) around 2017 (corresponding to release 'Unity 2017.x').

Unity offers a cross-platform experience by enabling a one-time development effort followed by multiple builds for different platforms, as needed, for which, platform-specific libraries may be installed (Exhibit 4a) . It encourages a tooling-based drag-and-drop app development, with additional programmatic interfaces (particularly in C#) for further customization (Exhibit 4d).

Unity Features and Differentiation: From the table above, it is understandable that this space was a sort of blue ocean, more so in terms of a cross-platform strategy, when Unity launched its AR workbench.

Unity differentiated itself by creating a generic framework called 'AR Foundation'. Applications developed on this framework can be deployed on various devices (including iOS, Android, and specialized AR/VR devices like HoloLens). The in-built capabilities of this framework include device tracking, light estimation, anchors, face tracking, meshing, and so on, for 2D and 3D applications. It can be credited for crafting an intuitive workbench with the look and feel of a programming IDE, and yet redesigned for geometrical and optical manipulation of 2D or 3D assets in an AR application (Figure 6).

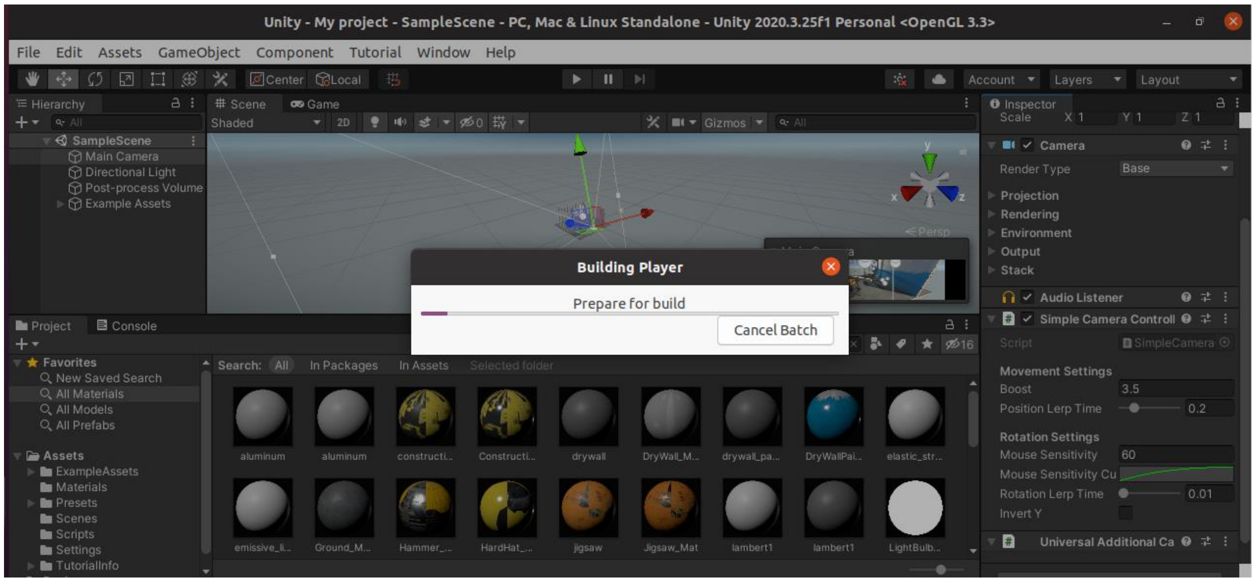


Figure 6: Unity AR Developer Tool (2020.x release) showing an application on 3D objects, freely available with the product

Unity has also charted out a workflow for AR application development, in its early days, called Unity MARS. It simplifies common tasks through a WYSIWYG simulation of the audience view, proxies for unknown elements in dynamic environments, and integrated test environment.

It further supports blending and integrating with other tools for diverse needs of the AR community. For example, advanced computer vision functions offered by Vuforia SDK, can be easily imported into a Unity AR project.

Portfolio and Pricing: The platform is offered free with basic features, for individuals. The paid subscriptions come in three flavors: Unity Plus (for small teams), Unity Pro and Unity Enterprise. Most of the capabilities of the Pro and Enterprise editions are targeted towards team collaboration and faster builds. Given that this technology is very new, quite sensibly, additional priority support and a customer success manager are included in the Enterprise edition (priced more than double the Pro version).

Insights: The Unity platform has been fairly successful so far, by investing in a core framework (AR Foundation) to create a new cross-platform developer experience. It has garnered wider industry acceptance for this framework; for example, Android offers ARCore Extensions for AR Foundations. It offers a balance of tooling an programmatic tasks, and has been able to bundle the right set of product features and services (support / consulting) for this emerging and exciting field. From a strategy perspective, Unity was quick to spot a natural scope for business expansion from gaming to AR application development, and proved to be quite effective in its execution so far.

Long Tail of Developer Choices

Moving on from the case studies now, in this and the next few sections, we collate our key learnings. Developer experience relates to offering more choices for platform users, for example:

- Selenium, a test-bench for automated testing of graphical interfaces, supports at least three recent versions of six different web browsers, some of which have hardly 2% market-share.
- Cloud Foundry enables full-stack cloud-native applications to be developed in about ten frameworks like Java, Python, Ruby, Node.js, etc., and deployed to five different private and public clouds: vSphere, AWS, Azure, GCP and OpenStack. It offers many options for application security, high availability, load-balancing, logging and monitoring.
- Paypal offers software development kits (SDK) for its payment gateway, in around a dozen programming languages, including Javascript, Java, Python, PHP, etc., and for Android and iOS. Both XML-based Simple Object Access Protocol (SOAP) and Representational State Transfer (REST) web services, are supported.

There might be added complexity in defining developer experiences attuned to on-premise, cloud-native (PaaS) and hybrid environments.

Some options exist to drastically scale down the cost of managing a long tail, like adhering to industry-standard programmatic approaches in database connectivity and web services, so that a single implementation can support a large number of configurations. But there may still be considerable effort in testing, certification, and possibly tweaking administrative interfaces per configuration. Ultimately, a highly resourceful ecosystem has to be put in place to help support a wide variety of choices.

A few differences are discernible, when compared to a long tail of digital products, or merchandise. Sometimes a disproportionately high business opportunity may emerge even from a rarely needed configuration. For example, for an e-commerce platform, lack of integration with a rarely used payment method, may be a deal-breaker in a specific country where it is essential. Secondly, each item (anywhere on the curve) may go through a life-cycle of maintenance and upgrades, which needs more oversight.

Nevertheless, a few core principles of 'the Long Tail' (Anderson, 2006) are relevant for software platforms too. In order to *democratize production*, the tools for extending and customizing a platform, have to be accessible and well documented at an acceptable cost. Furthermore, product management should play a role in encouraging active collaboration between the internal and external developers through online forums, real-time messaging (e.g., Slack channels), etc. The ecosystem may be architected by creating differentiated roles like technical experts who build general-purpose extensions, professional service partners who aid in supporting one-off customer requirements, consultants and trainers (Figure 7).

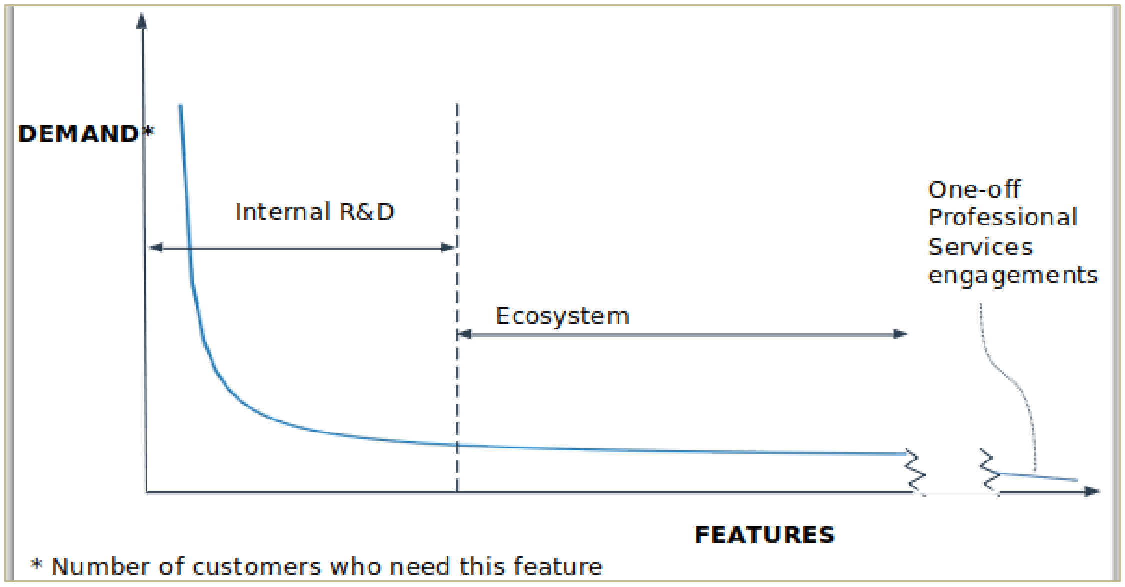

DEMAND*
Internal R&D
Ecosystem
One-off Professional Services engagements
FEATURES
* Number of customers who need this feature

Figure 7: Possible options in managing a long tail of developer requirements

While democratizing, care has to be taken to adequately incentivize the ecosystem, through a two-fold approach: certifications and ratings on each technical contribution, and badges, tier-based promotions, etc., for the ecosystem players themselves.

Democratizing distribution is trickier for platform ecosystems, since the quality of external contributions may affect platform usability and its brand. On one hand, the processes to extend and customize a platform should be simple and well-documented, and the learning curve should not be very steep. On the other hand there should be gate-keeping mechanisms for publishing an extension, like compliance test suites provided by the platform vendor. Unresolved technical issues should be clearly listed against each extension or customization. It might be possible to continuously test some of the extensions contributed by the ecosystem, whenever new versions (including minor versions and patches) of the core platform are released, and notify contributors of failures or breakages. Timelines should be specified for contributors to update their extensions in alignment with new versions of the core platform, and a mechanism to support and de-support (or decertify) an extension should be in place. But there should be some latitude for innovation and market-testing, by allowing some extensions to be published on 'experimental' basis.

Putting up a marketplace for add-on contributions is obviously recommended for *connecting supply and demand*. The marketplace has to be well curated not only in terms of ecosystem contributions, but also in terms of ecosystem players. While the extensions may be rated and marked 'Supported / Certified' or 'Unsupported / Experimental', as the case may be, possibly with reviews and related extensions, the ecosystem participants should be listed by area of expertise, industry verticals, and geographical location. A summary of the types of customer support offered by each ecosystem participant must be published. For an end customer, it should be feasible to use extensions, as well as engage with ecosystem participants. The platform vendor may subtly encourage moderate competition in the marketplace, so that end customers have a few choices for add-on functionalities, and are not affected by business risks faced by individual ecosystem players.

For a vibrant ecosystem, product management should liaise effectively with business development, wherein the business development team should actually be the voice of the external ecosystem, sometimes even prioritizing their interests over those of the platform vendor.

Ease of Development and Productivity

For both existing platforms and new ones in the planning phase, assessment of ease of development and developer productivity should start from the high-level use cases. For each use case, a detail-oriented product manager would first work out the tool-chain that a developer would typically resort to, to get the job done. 'Tool-chain' should be inferred in a broader sense, including not only graphical (thin or fat) interfaces, but also programmatic interfaces for customization, integration and scriptability.

From this analysis, gaps in the tool-chain may be identifiable, which make a use case difficult to carry out. We must clarify that some complex developer tasks could well involve third party tools (e.g., tools for web application development may depend on a third party database schema designer). But the sum total of tools offered by the platform vendor and a few mature, complementary, widely available tools from third parties, should suffice. Then, it behooves us to measure some sort of a kludge index for a use case and its associated tool-chain(s). Effort is needed in making the building blocks of a tool-chain seamless and interoperable. The analysis should then go one level deeper, into how intuitive the features exposed by each of the tools in a tool-chain are. Milestone-based demos for internal and selected external audiences, during a platform development project, can help in this assessment.

To close gaps in a tool-chain, apart from in-house development and tapping into the ecosystem, product managers should be open to acquisitions (even micro-acquisitions) that gel well with the platform. Some acquisitions may have to be forward-looking, i.e., not necessarily to fill in existing gaps, but desirable from the perspective of the platform roadmap. Atlassian offers a comprehensive platform for managing the software development life-cycle. Table 3 below shows a part of its portfolio built out of Jira, Confluence and a few acquired products (shaded).

Plan and Track	**Support and Fix**	**Code, Build and Ship**	**Collaborate**
Jira Align for Enterprise Agile Planning	*Jira* for IT Service Management for	*Jira* for Project and Issue Tracking	*Confluence* for Document Collaboration
Jira Software for Project and Issue Tracking	*Opsgenie* for Incident Response	*Bitbucket* for managing Git repositories	*Trello* for visual dashboard of project tasks
Confluence for Document Collaboration	*Statuspage* for Incident Communication	*Sourcetree* desktop client for code repositories	*Jira* for Business team collaboration

Table 3: Part of Atlassian Product Portfolio developed organically and through acquisitions

For convenience, developer productivity may be separated into features pertaining to graphical interfaces and programmatic interfaces. Ideally, a graphical interface should feel like a single pane of glass on which one can accomplish most tasks including coding, debugging, testing, profiling, team collaboration, etc., end-to-end, without switching. Programmatic interfaces need to be expressive yet concise, and numerous artifices like database object relational mapping, data type and data format inference, fluent APIs, code generation and completion, etc., ultimately contribute to lesser coding. Past efforts have culminated into novel low-code / no-code development on some platforms today. Both types of interface benefit from automation of repeatable tasks, particularly test and deployment. Beyond scriptable test and deployment suites, platforms are benefitting from Robotic Process Automation (RPA) that record workflows navigating through graphical interfaces. Often, a subtle need is right granularity and composability of features for creating flexible developer workflows, like the Unix shell on which we compose myriads of scripts even today, from small bits of functionality.

Indeed, for some platforms, developer experience is the be-all and end-all of product success, whence, it is worth measuring where it stands individually, and vis-à-vis key competitors. The SPACE Framework (Satisfaction and Well-being, Performance, Activity, Communication and Collaboration, Efficiency and Flow) very recently published by Forsgren et al. (2021) offers clues for measuring productivity across a development team.

- A development team may work on multiple platforms in a project. But if one of those platforms is the predominant one (e.g., Android Studio in a mobile app development project), some assessment of the platform based on the SPACE parameters may be feasible, from the overall project assessment.
- Some social network analysis tools may be applicable to online developer forums, in order to gauge overall developer sentiment and even drill down to how specific features are perceived.
- A perceptual map, possibly authored by an independent analyst, on the SPACE parameters and even on other criteria, could be valuable in gaining developer mindshare and market-share.

Overall, ease of development should manifest in on-demand extensibility of the platform, in view of the trend towards shrinking project timelines. An ecosystem player, or in-house client team should be able to build different types of extensions and customizations meeting the goals of business agility.

Marketing Communications

It may be desirable to create two sets of marketing messages that are consistent and in alignment: one for business leaders who invest in a platform, and one for developers who are likely to be key influencers in investment decisions. Here we will focus on messaging targeted towards developers.

Content marketing is key to gaining developer adoption of a platform. It may be worth focusing on how a platform solves interesting technical problems and how it offers a blueprint for better end-user applications. The content should resonate on some of the qualities that developers yearn for, like innovation, elegance and efficiency. There should be persistent communication efforts, for example, through monthly newsletters.

Integrated marketing communications for platforms spans beyond the boundaries of traditional marketing in some ways, two of which are highlighted below:

- Platform documentation, training, knowledgebase, and other technical resources should be aligned with, and reinforce, the high-level marketing messages. This is easier said than done, because of the sheer volume of information and the number of teams involved in the process.
- *Sales enablement* should be the starting point of execution of marketing strategy, especially for enterprise or B2B platforms. Product management must ensure that themes and broad capabilities of each release, down to the intent behind each feature, is well understood by the field, and articulated effectively at each opportunity. Apart from sales playbook, for developer-focused platforms, pre-installed jump-start kits may be helpful.

When targeting a developer audience, marketing messages should rather be unbiased, focusing on the strengths of the platform and the environments or scenarios where it shines. Developers are eclectic and might not appreciate overzealous criticism of competitor platforms; positive points of difference should be presented in a nuanced manner.

To complement marketing efforts of the sales and marketing organizations, the internal development team should be encouraged to play the role of evangelists. Some of the technical leaders who build the platform, could contribute content, and address ecosystem players or the larger developer community through online forums. This establishes a strong developer to developer connect, that is effective in selling the strengths of the platform. Some openness in communication, for example, by publicly sharing a few upcoming features, helps build enthusiasm among developers. Another message that has to reach the community effectively, is, available options for developer support (contrasted with customer support). This includes timely response to not just bugs reported, but questions on how to solve a problem, which requires higher bandwidth interaction (e.g., through a Slack channel).

The external developer community should also be encouraged to create a strong network effect, by participating in discussions, sharing technical expertise and earning reputation. Digital Ocean prides itself as the developer-friendly cloud platform, and has a community of a million-plus developers. For several years, it has encouraged community contributions for technical content, to be published on its own website. Recently, it announced the Digital Ocean Navigator program for the community to contribute content, code, and engagements, and be recognized. Through such initiatives, platform vendors disseminate independent, credible opinion through the larger developer network, amplify the key messages and turn them viral. (Google search for technical tutorials on a large gamut of topics, leads to Digital Ocean blogs within the first few results.) Local meet-ups (that are also highly cost-effective compared to promotional events) can help internal teams and external developer community to share knowledge, leading to cross-pollination of ideas.

It may be a good idea to reconstruct some of the marketing messages based on feedback from the developer community. A platform may be made use of in more ways than what the vendor itself envisaged. The community brings in their own ideas and can have interesting perspectives. It is important to ask what capabilities they like most, and tweak sales playbooks accordingly.

Pathways to Monetization

Firstly, for any platform, depending on deployment options and pricing models, a matrix of pricing strategies takes shape, which is holistic, taking into account the core platform and support. The one shown below (Table 4) is illustrative and not exhaustive.

	On-Premise		**Cloud (PaaS)**	
	Core Platform	*Support / Services*	*Core Platform*	*Support / Services*
Free				
Freemium				
Paid				

Table 4: A suggested matrix of platform pricing strategies

The 'Paid' model above is simply the one that is not free or freemium, though there could be a trial.

A platform product manager may have to strike a fine balance between developer adoption and revenues earned. For the sake of developer adoption, pricing should be as simple as possible, with a low barrier to entry. In addition, there should be a few straightforward bundles, since, a platform is often a bundle of tools to build a solution.

We delve into two important questions on pricing, for platform product managers:

- If the platform is *not* free or *not* intentionally low-priced, what are the core principles of right-pricing it?

- If the platform is intentionally free or low-priced, how can it be indirectly monetized?

Obviously, all opportunities for indirect monetization should be exploited for paid platforms too, but the question is more critical for free or low-priced ones.

The right price may be estimated by clearing the total cost of development on the platform against the value it offers not only to individual developers but also to teams. This immediately indicates that it may be worth supporting flexible team licenses. Reducing the total cost of development in a scalable manner could lead to a consumer surplus, that can be partially monetized by the platform vendor. Built-in templates, workflows, starter applications, pre-configured integrations, etc., smoothen the learning curve or training costs, thereby reducing cost of development. PaaS brings down the cost of development drastically by offering an assembled set of tested and proven tools and underlying computing infrastructure, for the developer, configured to work seamlessly (Weinman, 2012). Amazon Web Services (AWS) brings together machine learning algorithms and an MLOps framework on the same cloud platform, enabling developers to build reliable machine learning applications, and also deploy and execute them efficiently.

For new platforms that are offered free in order to accelerate developer adoption, there is an implicit expectation, that once a critical mass is reached, there will be a sudden disproportionate increase in developers, leading to opportunities for direct and indirect monetization. The moot questions in such cases are a) how reasonable this expectation is, and b) whether there would be backing from investors until the critical mass is reached.

A few patterns are discernible in monetizing free platforms, although this is not meant to be exhaustive:

1. Monetize through paid enterprise-grade features that enable scalability, reliability, monitoring, administration and automation, better integration with other enterprise systems, and security.
2. Monetize from support, professional services, consulting, and customized solutions. For a platform vendor, especially with limited resources, it is imperative that the monetization path be scalable. Annual Support tiers may be relatively scalable compared to services and solutions that need more techno-managerial resources, but a solution catalogue may be more scalable.
3. Offer the platform on the cloud at a price, though this requires experience in cloud-hosting. Alternatively, other vendors may be allowed to deploy the platform on cloud, or OEM it. If the platform is open source (apart from being free), a GPL license enables monetization through OEMs, whereas an AGPL license enables charging for cloud-hosting.

4. Innovate and develop a new paid platform building on the strengths of the current core platform, that maps the customer journey through technology adoption. Customers who use the core platform currently become prospective opportunities for the new platform.
5. Generate a strong brand, even from a free platform, like Android, which extends beyond mobile apps, into smart wear, personal health devices, TV, Android Things, etc. (On a related note, Google indirectly monetizes through search and ad revenues from Android phones, but this is somewhat unique, and may not be a pattern that other platforms can easily emulate.)

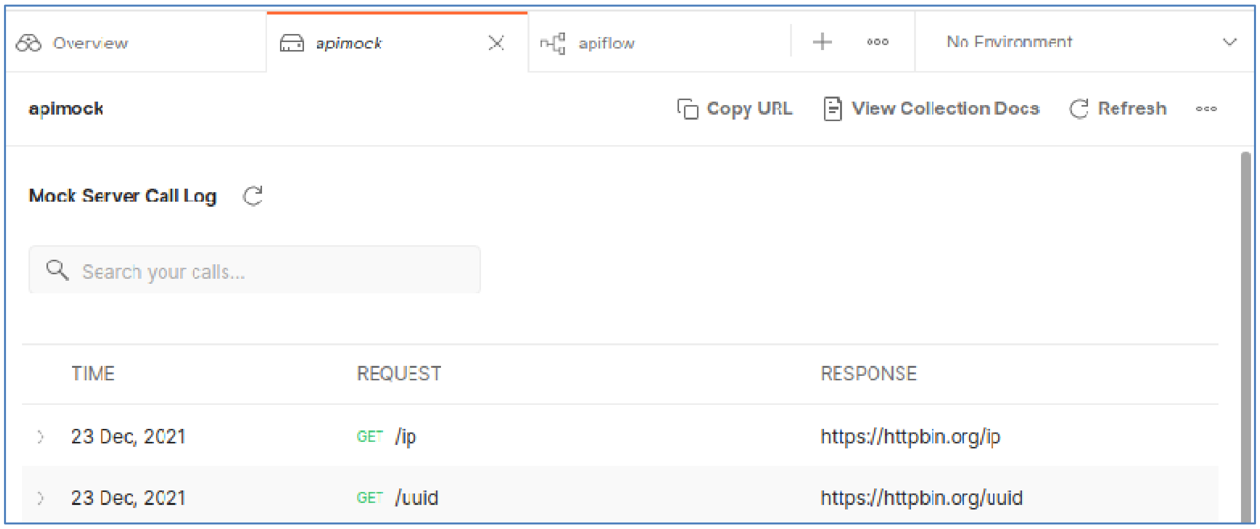
Monetizing through ecosystem players has to be trodden with caution, especially for developer-focused platforms. Rather a conscious approach is needed for them to reasonably monetize from the platform.

Conclusion and Future Work

In this paper we reviewed best practices followed by a few software platforms in crafting a superior developer experience, which leads to wider adoption and usually greater revenues. Our effort has been to analyze these best practices through the prism of established product management and strategic marketing principles, so that platform product managers may be able to apply or adapt some of them, with more confidence. We also show that though developer experience is often considered to be a matter of perception, it is possible to carry out a more structured, measurable and objective analysis of where a platform stands in this respect, that is needed in a competitive environment. The paper is intended to stimulate a cogent thought process rather than be prescriptive. Indeed, there are many specialized kinds of developer experiences, which we have not explored here, like graphic design, multidisciplinary development (e.g., computer aided design and drafting platforms), phygital products, hardware-software co-design, etc. It would be interesting to bring out case studies on developer experience, on diverse platforms, in future. But, for most forms of developer experience in general, it may be worth rephrasing what Kotler et al. (2021) propound, on customer experiences in the era of Marketing 5.0. That is, a well-crafted developer experience may further mechanize structured, pattern-oriented, programmable logic, and yet leave space for human insights, divergent thinking and contextual understanding.

Exhibits

Exhibits on Postman API Platform



TIME	REQUEST	RESPONSE
23 Dec, 2021	GET /ip	https://httpbin.org/ip
23 Dec, 2021	GET /uuid	https://httpbin.org/uuid

Exhibit 1a: Mock Server Feature in Postman

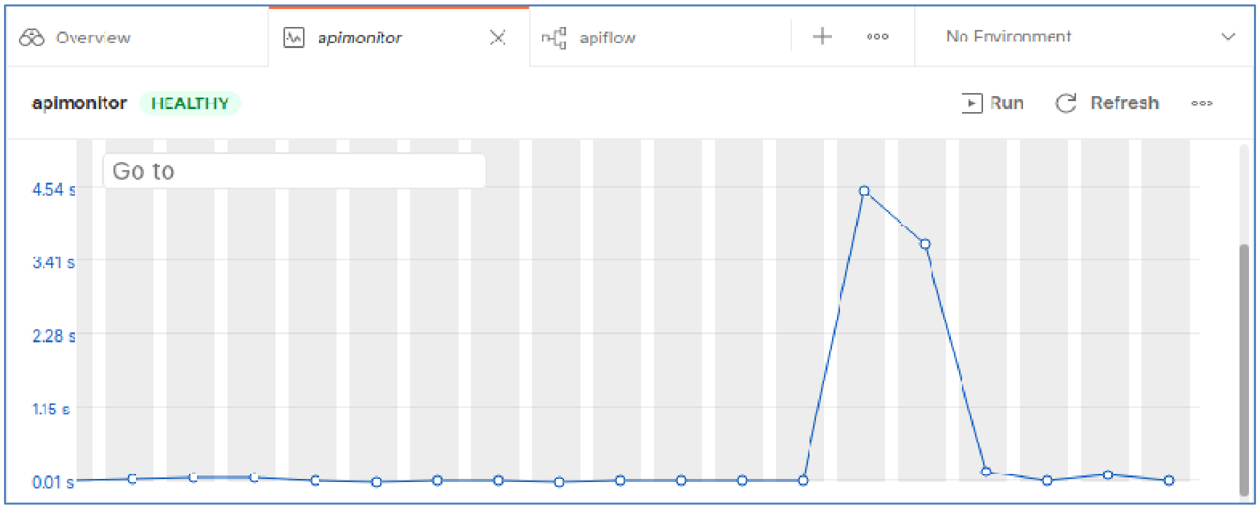
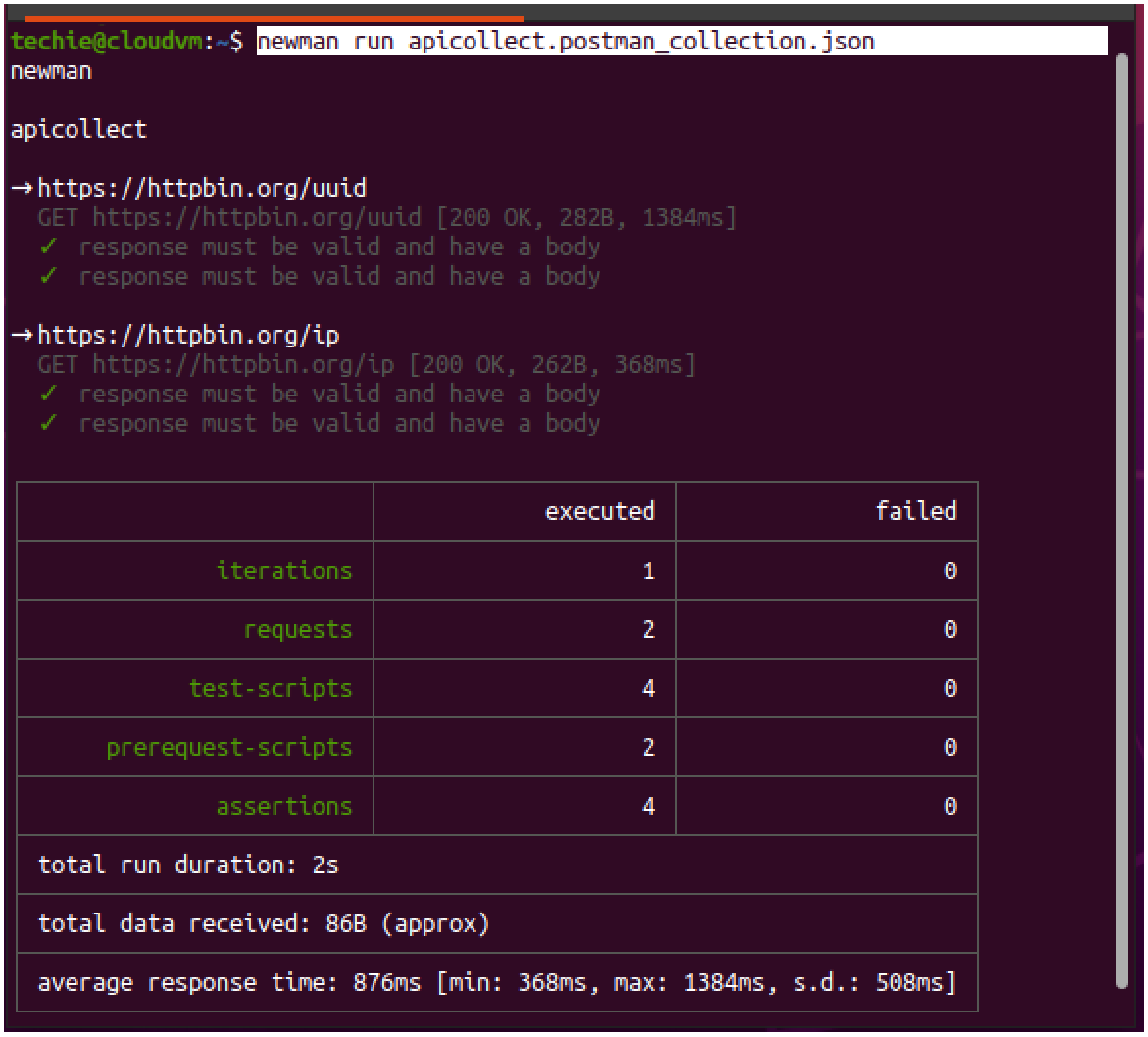


Exhibit 1b: API Monitoring in Postman



```
techie@cloudvm:~$ newman run apicollect.postman_collection.json
newman

apicollect

→https://httpbin.org/uuid
  GET https://httpbin.org/uuid [200 OK, 282B, 1384ms]
  ✓  response must be valid and have a body
  ✓  response must be valid and have a body

→https://httpbin.org/ip
  GET https://httpbin.org/ip [200 OK, 262B, 368ms]
  ✓  response must be valid and have a body
  ✓  response must be valid and have a body

┌─────────────────────────┬──────────────────────┬──────────────────────┐
│                         │             executed │               failed │
├─────────────────────────┼──────────────────────┼──────────────────────┤
│              iterations │                    1 │                    0 │
├─────────────────────────┼──────────────────────┼──────────────────────┤
│                requests │                    2 │                    0 │
├─────────────────────────┼──────────────────────┼──────────────────────┤
│            test-scripts │                    4 │                    0 │
├─────────────────────────┼──────────────────────┼──────────────────────┤
│      prerequest-scripts │                    2 │                    0 │
├─────────────────────────┼──────────────────────┼──────────────────────┤
│              assertions │                    4 │                    0 │
├─────────────────────────┴──────────────────────┴──────────────────────┤
│ total run duration: 2s                                                │
├───────────────────────────────────────────────────────────────────────┤
│ total data received: 86B (approx)                                     │
├───────────────────────────────────────────────────────────────────────┤
│ average response time: 876ms [min: 368ms, max: 1384ms, s.d.: 508ms]   │
└───────────────────────────────────────────────────────────────────────┘
```

Exhibit 1c: Testing a Postman API collection using newman tool which can be integrated with a CI pipeline

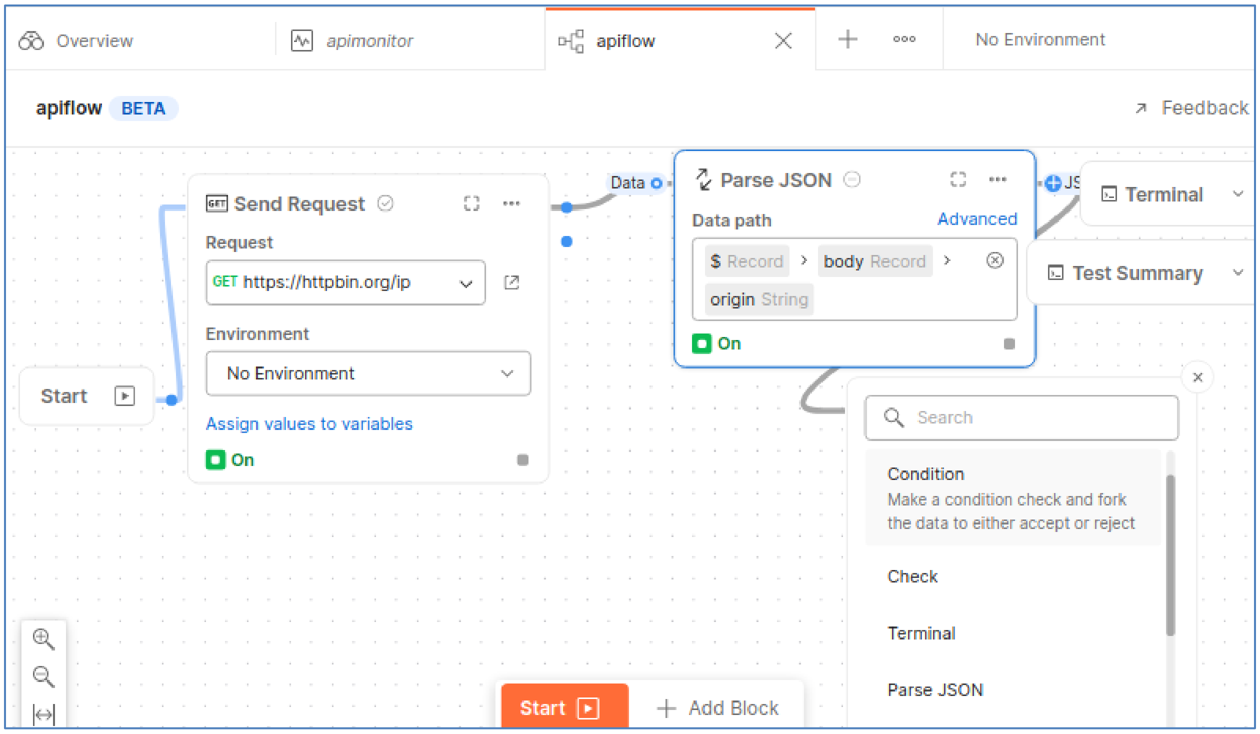
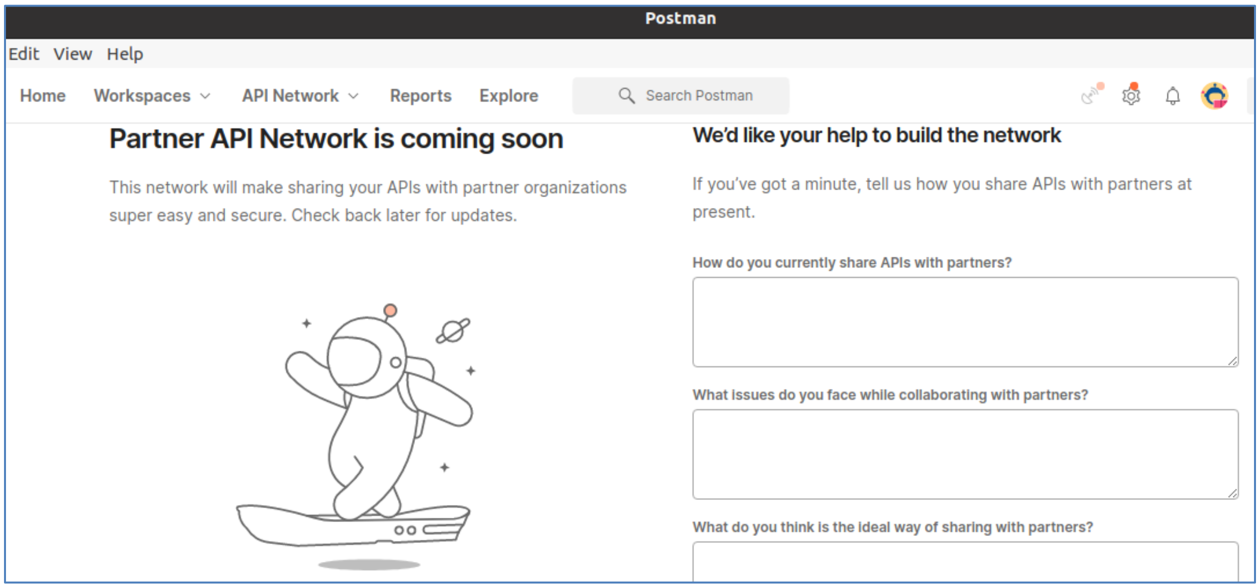


Exhibit 1d: Postman API Flow (currently in beta)



Partner API Network is coming soon

This network will make sharing your APIs with partner organizations super easy and secure. Check back later for updates.

We'd like your help to build the network

If you've got a minute, tell us how you share APIs with partners at present.

How do you currently share APIs with partners?

What issues do you face while collaborating with partners?

What do you think is the ideal way of sharing with partners?

Exhibit 1e: Upcoming Feature - Partner API Network

Exhibits on Pentaho Data Integration

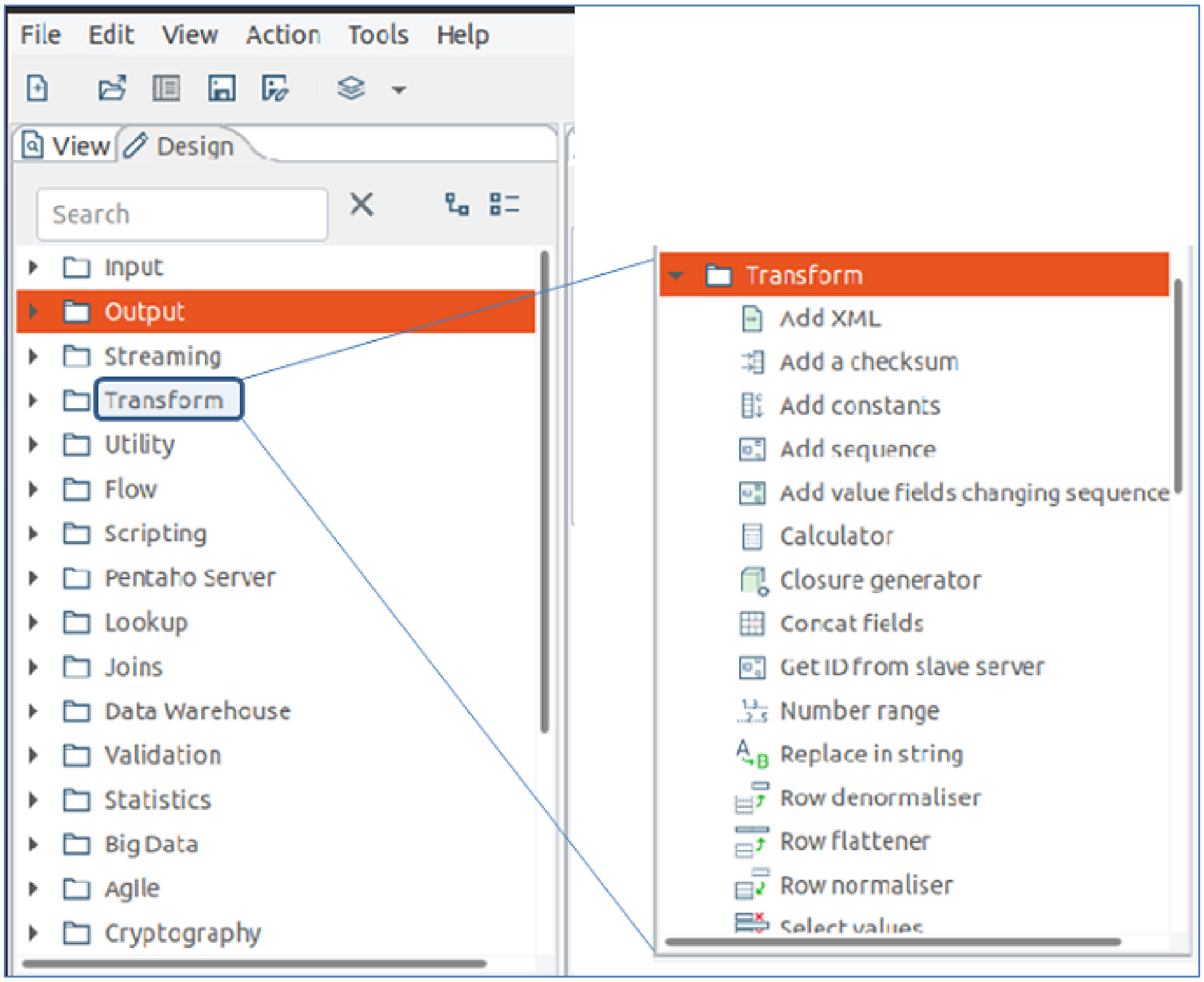
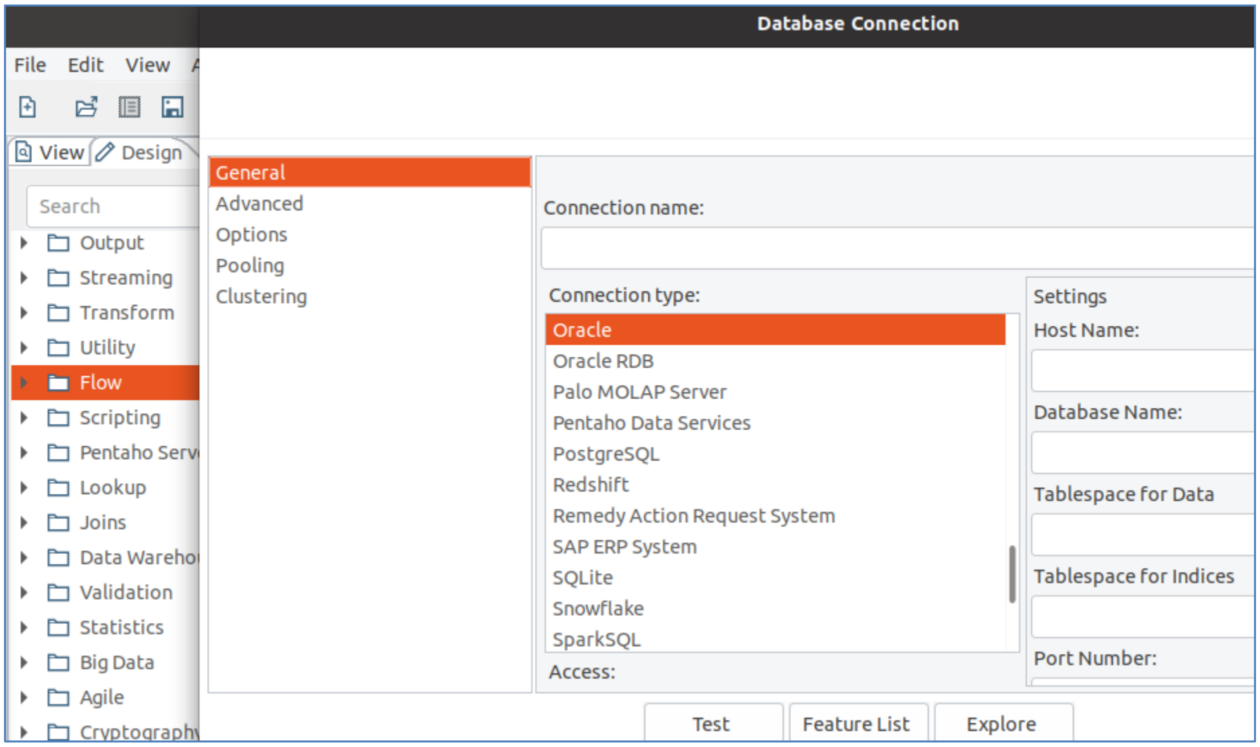


Exhibit 2a: A subset of the wide range of drag-n-drop data transformations built into the Pentaho Data Integration tool



Connection name:

Connection type:

Oracle
Oracle RDB
Palo MOLAP Server
Pentaho Data Services
PostgreSQL
Redshift
Remedy Action Request System
SAP ERP System
SQLite
Snowflake
SparkSQL

Access:

Settings

Host Name:

Database Name:

Tablespace for Data

Tablespace for Indices

Port Number:

Exhibit 2b: A subset of the relational databases supported as Data Sources or Sinks in Pentaho Data Integration tool

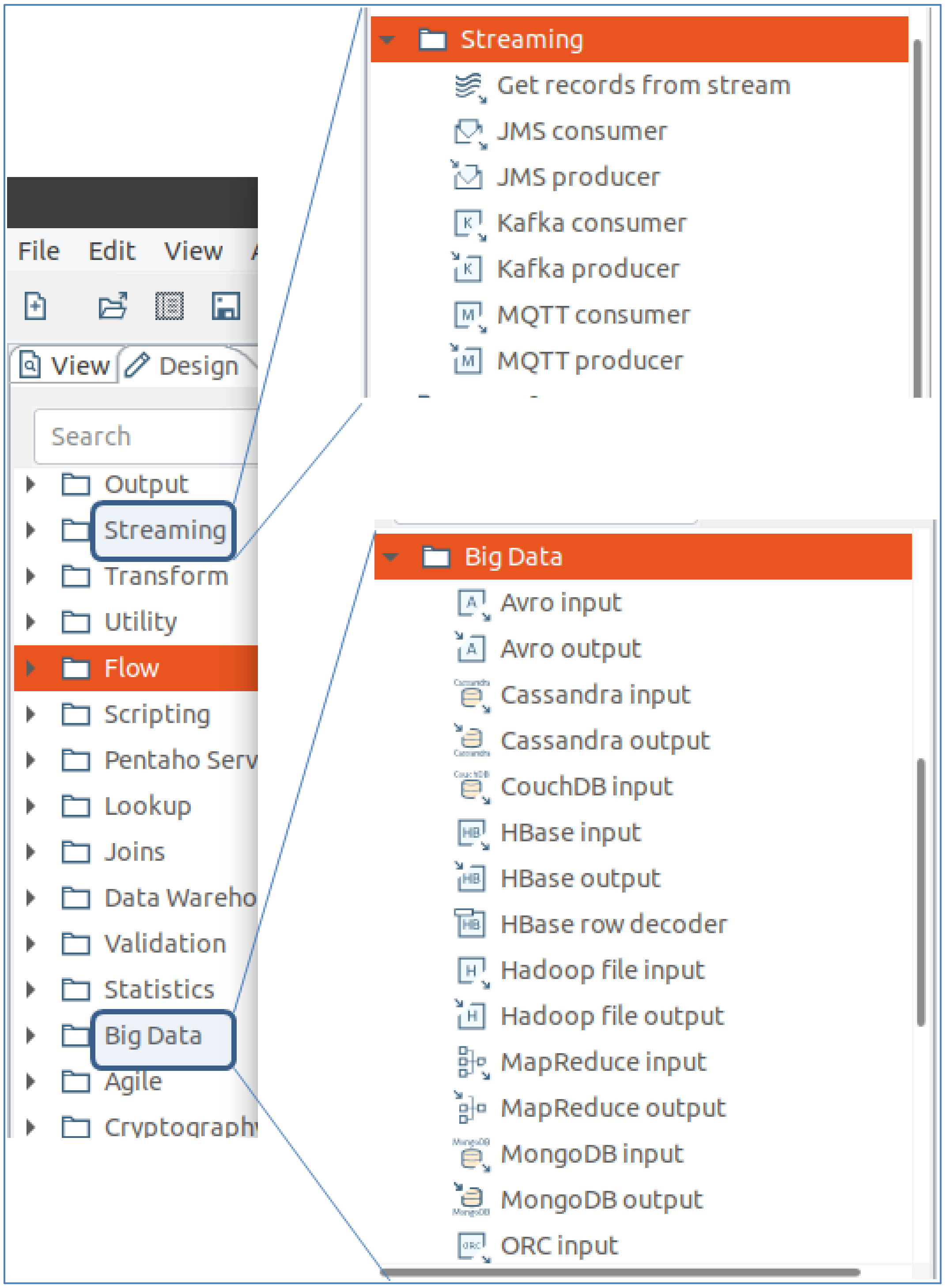


Exhibit 2c: A subset of big data platforms and data streaming platforms for which connectors are built into the Pentaho Data Integration tool

Exhibits on Captum Interpretable AI

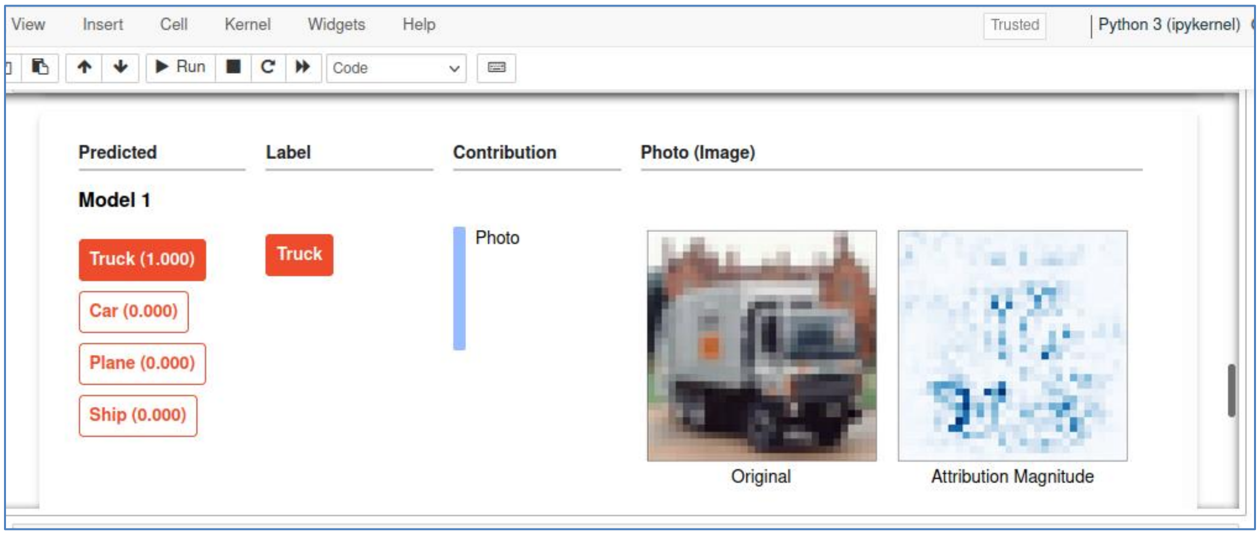


Exhibit 3a: Example of a correct prediction and its attribution using Captum Insights

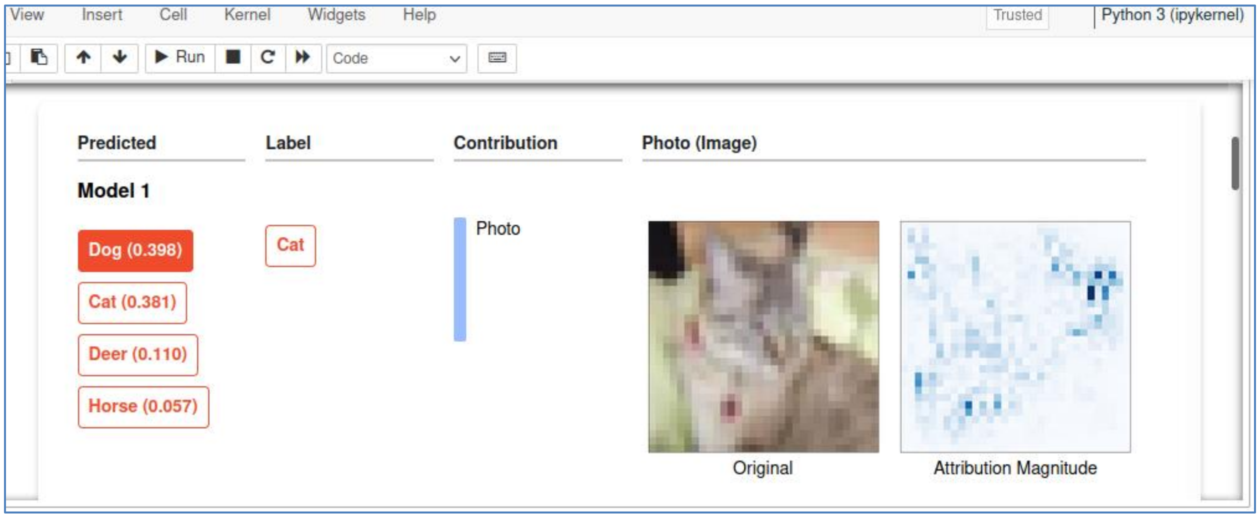
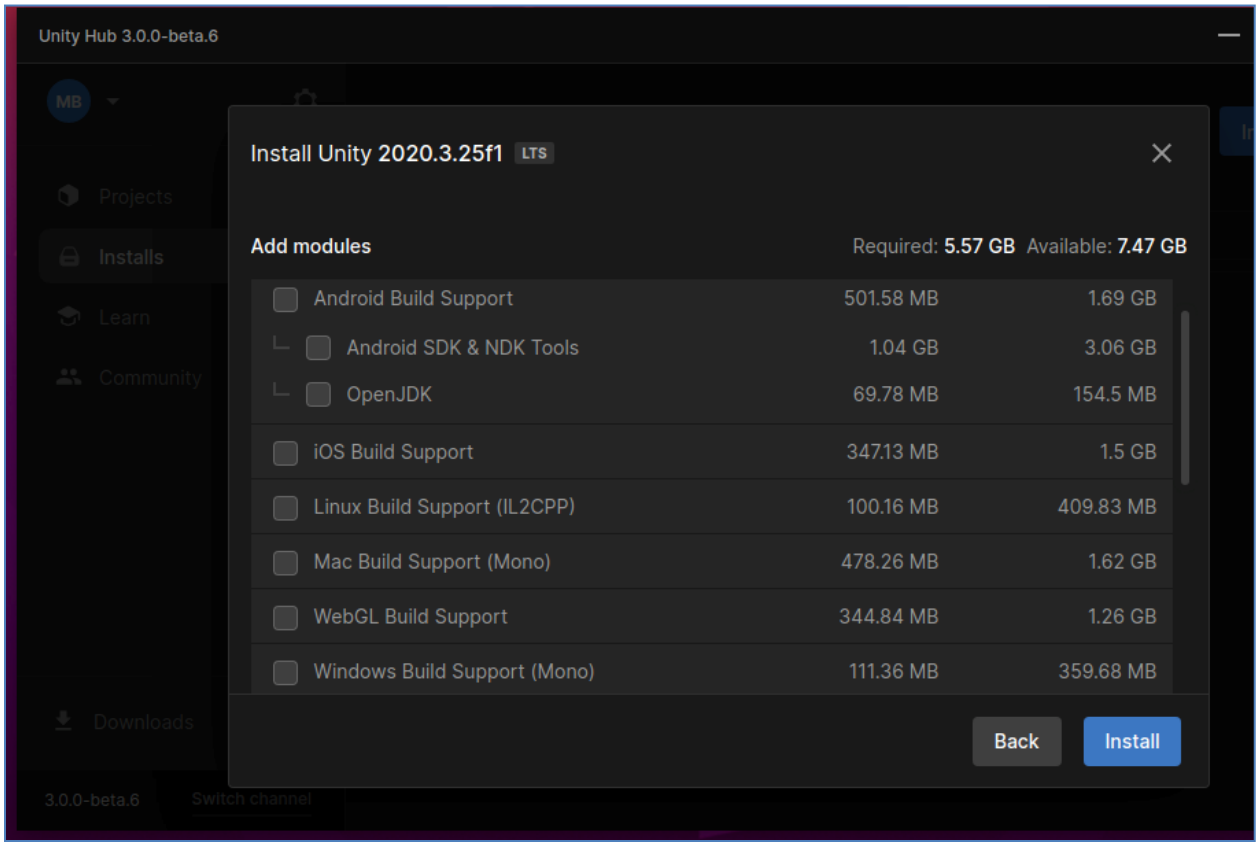


Exhibit 3b: Example of an incorrect prediction and its attribution using Captum Insights

Exhibits on Unity Augmented Reality Platform



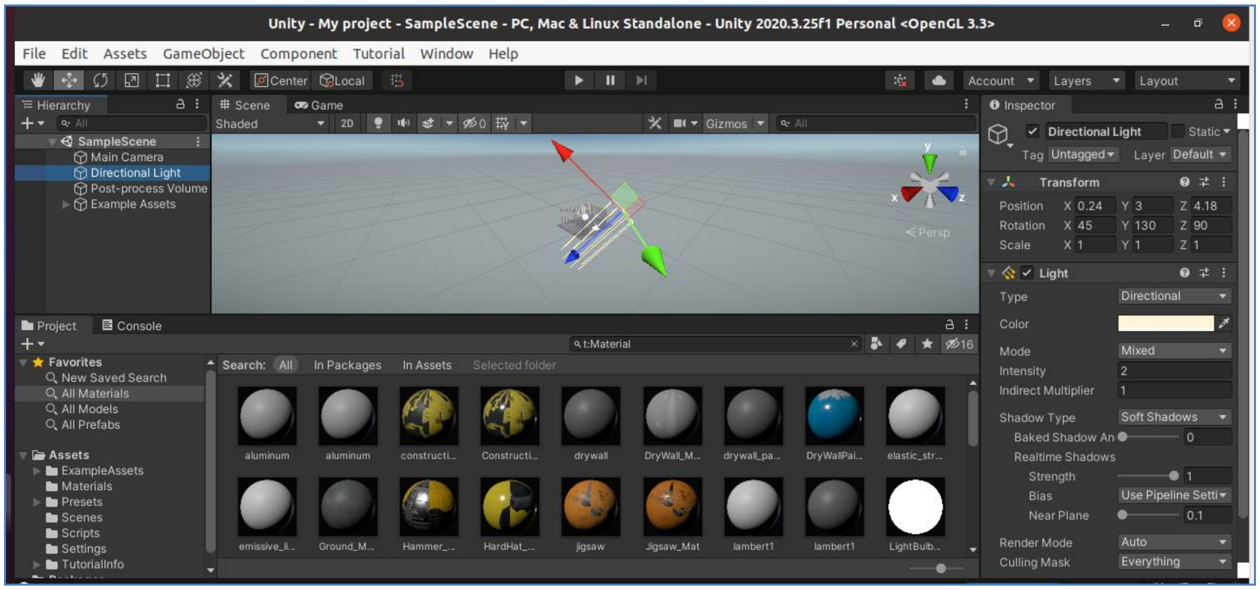
Install Unity 2020.3.25f1 LTS

Add modules — Required: **5.57 GB** Available: **7.47 GB**

Android Build Support	501.58 MB	1.69 GB
└ Android SDK & NDK Tools	1.04 GB	3.06 GB
└ OpenJDK	69.78 MB	154.5 MB
iOS Build Support	347.13 MB	1.5 GB
Linux Build Support (IL2CPP)	100.16 MB	409.83 MB
Mac Build Support (Mono)	478.26 MB	1.62 GB
WebGL Build Support	344.84 MB	1.26 GB
Windows Build Support (Mono)	111.36 MB	359.68 MB

Back Install

Exhibit 4a: Unity AR Platform - Options for installing modules for cross-platform support



Directional Light Static
Tag Untagged Layer Default

Transform

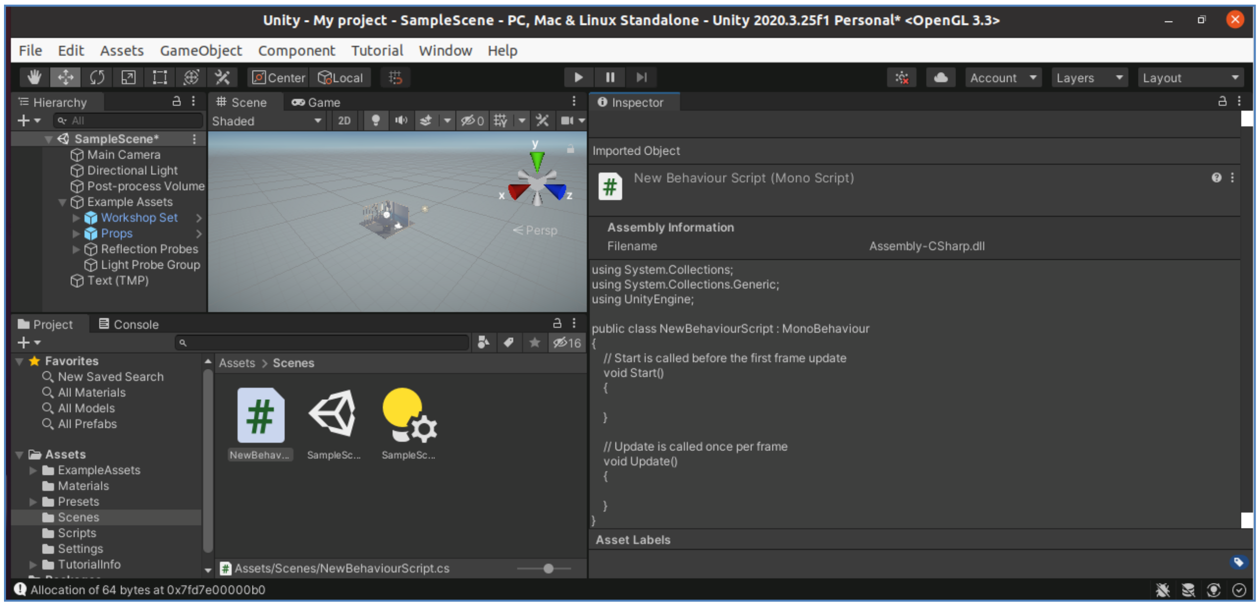
Position	X 0.24	Y 3	Z 4.18
Rotation	X 45	Y 130	Z 90
Scale	X 1	Y 1	Z 1

Light

Type	Directional
Color	
Mode	Mixed
Intensity	2
Indirect Multiplier	1
Shadow Type	Soft Shadows
Baked Shadow An	0
Realtime Shadows	
Strength	1
Bias	Use Pipeline Setti
Near Plane	0.1
Render Mode	Auto
Culling Mask	Everything

Exhibit 4b: Another screenshot of Unity AR platform

Exhibit 4c: A 3D scene generated through Unity AR tool after completing a build



Unity - My project - SampleScene - PC, Mac & Linux Standalone - Unity 2020.3.25f1 Personal* <OpenGL 3.3>

Imported Object

New Behaviour Script (Mono Script)

Assembly Information

Filename Assembly-CSharp.dll

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NewBehaviourScript : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

Assets/Scenes/NewBehaviourScript.cs

Allocation of 64 bytes at 0x7fd7e00000b0

Exhibit 4d: C# Code editor integrated with Unity AR platform

References

Anderson, C. (2006). The Long Tail. Hyperion.

API development platform Postman nabs $225M (2021). VentureBeat. Retrieved from: https://venturebeat.com/2021/08/18/api-development-platform-postman-nabs-225m/

API platform Postman delivers $150M Series C on $2B valuation (2020). TechCrunch. Retrieved from: https://techcrunch.com/2020/06/11/api-platform-postman-nabs-150m-series-c-on-2b-valuation/

Apple ARkit Framework (2021). Apple. Website: https://developer.apple.com/documentation/arkit

AR.js - Augmented Reality on the Web (2021). Ar.js. Retrieved from: https://github.com/AR-js-org/AR.js/

ARCore - Getting started with AR Foundation (2021). Google Webpage. Retrieved from: https://developers.google.com/ar/develop/unity-arf/getting-started-ar-foundation

AR Foundation - A framework purpose-built for augmented reality development (2021). Unity webpage. Retrieved from: https://unity.com/unity/features/arfoundation

Atlassian Solutions (2021). Atlassian. Website: https://www.atlassian.com/

Augmented Reality - An end-to-end creation platform (2021). Unity webpage. Retrieved from: https://unity.com/unity/features/ar

Augmented Reality Disappeared From Gartner's Hype Cycle – What's Next? (2020). ARPost. Retrieved from: https://arpost.co/2020/09/25/augmented-reality-gartners-hype-cycle/

AWS MLOps Framework (2021). Amazon Web Services Webpage. Retrieved from: https://aws.amazon.com/solutions/implementations/aws-mlops-framework/

Browser Marketshare worldwide (2021). StatsCounter. Retrieved from: https://gs.statcounter.com/browser-market-share

Captum Model Interpretability for PyTorch (2021). Captum. Website: https://captum.ai/

CloudFoundry Overview (2021). CloudFoundry Webpage. Retrieved from: https://docs.cloudfoundry.org/concepts/overview.html

Create Database Plugins (2021). Pentaho Documentation. Retrieved from: https://help.hitachivantara.com/Documentation/Pentaho/9.0/Developer_center/Create_database_plugins

Digital Ocean's Navigator Program (2021). Digital Ocean Webpage. Retrieved from: https://www.digitalocean.com/community/pages/digitalocean-navigators

Forsgren, N., Margaret-Anne, S., Maddila, C., Zimmermann, T., Houck, B., Butler, J. (2021). The SPACE of Developer Productivity. *ACMqueue*, March 2021.

Frappe Application Development Framework (2021). Frappe. Website: https://frappe.io/

Free Machine Learning Services on AWS (2021). Amazon Web Services Webpage. Retrieved from: https://aws.amazon.com/free/machine-learning

Get a Faster Return on Data with DataOps and Lumada (2021). Hitachi Vantara webpage. Retrieved from: https://www.hitachivantara.com/en-in/insights/dataops-insights/dataops.html

Get Paid to write tutorials (2013). Digital Ocean blog. Retrieved from: https://www.digitalocean.com/blog/get-paid-to-write-tutorials/

Heroku Console (2021). Heroku. Website: https://devcenter.heroku.com/categories/command-line

How Postman built a $5.6B developer community (2021). Toplyne. Retrieved from: https://www.blog.toplyne.io/how-postman-built-a-2b-community/

Install Browser Drivers (2021). Selenium. Retrieved from: https://www.selenium.dev/documentation/webdriver/getting_started/install_drivers/

Kotler, P., Kartajaya, H., Setiawan, I., (2021). Marketing 5.0 Technology for Humanity. Wiley.

Leverage Open-Source Benefits With the Assurance of Hitachi Vantara (2021). Hitachi Vantara Product Brochure. Retrieved from: https://www.hitachivantara.com/en-us/pdf/brochure/leverage-open-source-benefits-with-assurance-of-hitachi-overview.pdf

Lumada Data Integration (2021). Hitachi Vantara webpage. Retrieved from: https://www.hitachivantara.com/en-in/products/data-management-analytics/lumada-data-integration.html

Mudrakarta, P. K., Taly, A., Sundararajan, M., Dhamdhere, K. (2018). Did the model understand the question? *arxiv*, May 2018.

Overview of ARCore and supported development environments (2021). Google. Retrieved from: https://developers.google.com/ar/develop

PayPal SDKs (2021). PayPal. Website: https://github.com/paypal

Pentaho Marketplace (2021). Hitachi Vantara webpage. Retrieved from: https://marketplace.hitachivantara.com/pentaho/

Postman API Collaboration Platform Delivers For 15 Million Software Developers (2021). Forbes. Retrieved from: https://www.forbes.com/sites/brucerogers/2021/07/23/postman-api-collaboration-platform-delivers-for-10-million-software-developers/

Postman API Platform (2021). Postman. Retrieved from: https://www.postman.com/downloads/

Postman Raises $1 Mn From Nexus Venture Partners (2015). inc42. Retrieved from: https://inc42.com/buzz/postman-raises-1-mn/

Postman raises $50 million to grow its API development platform (2019). TechCrunch. Retrieved from: https://techcrunch.com/2019/06/19/postman-series-b/

Postman Raises $7 Mn Series A Funding From Nexus Venture Partners (2016). inc42. Retrieved from: https://inc42.com/flash-feed/postman-7-mn-series-a/

Postman recognized as a Visionary by Gartner (2021). Retrieved from: https://www.postman.com/product/gartner-magic-quadrant/

Postman's Series C Funding and the Future of APIs (2020). Postman Blog. Retrieved from: https://blog.postman.com/postmans-series-c-funding-and-the-future-of-apis/

PyTorch From Research to Production (2021). PyTorch. Website: https://pytorch.org/

SoapUI (2021). SoapUI. Website: https://www.soapui.org/

Unity MARS - Create augmented reality (AR) apps with better workflows and purpose built authoring tools (2021). Unity webpage. Retrieved from: https://unity.com/products/unity-mars

Unity Plans and Pricing (2021). Unity Webpage. Retrieved from: https://store.unity.com/#plans-business

Unity Releases (2021). Unity webpage. Retrieved from: https://unity.com/releases/release-overview

Vuforia Engine in Unity (2021). Vuforia Webpage. Retrieved from: https://library.vuforia.com/articles/Training/getting-started-with-vuforia-in-unity.html#intro

Weinman J (2012). Cloudonomics: The Business Value of Cloud Computing. Wiley.

What's The Difference Between A Software Product And A Platform? (2015). Forbes. Retrieved from: https://www.forbes.com/sites/adrianbridgwater/2015/03/17/whats-the-difference-between-a-software-product-and-a-platform

Which Pentaho Experience is Right for You? (2021). Hitachi Vantara webpage. Retrieved from: https://www.hitachivantara.com/en-in/products/data-management-analytics/pentaho/download-pentaho.html

Why Lumada Software for IIoT (2021). Hitachi Vantara webpage. Retrieved from: https://www.hitachivantara.com/en-in/products/iot-software-solutions/lumada-software-for-iiot.html

Zoho Developer Platform (2021). Zoho. Website: https://www.zoho.com/developer/