

---

# Project Life Cycles in Open-Source Software

---

Anonymous Author(s)

Affiliation

Address

email

## Abstract

Adapting methods previously applied to product life cycles, this paper models developer engagement through the project life cycle for open-source projects, and detects similar dynamics in a cross section of projects. Endogenous growth theory is adapted to model growth dynamics in open-source software engineering, while incorporating the interactions between growth levels and developer activity over time using systems of differential equations. The solution to this model calibrates well to many open-source projects. The model generates an estimate of the lifetime developer engagement and growth, which supports estimating a lifetime production value of open-source projects. These models may be used to drive open-source strategy and value open-source projects for tech firms.

## 1 Introduction

Changes in productivity in modern economies have been greatly ascribed to technological change. In this paper, we model how growth levels (in open-source software) dynamically interact with the number of developers in the project. This interaction is important—as an open-source software (OSS) project grows and finds interest in the user community, a developer community also springs up around it. This accelerates the activity level of the project, and attracts more developers who wish to tweak and enhance the open-source project for their own use. This flywheel persists until the project matures and the level of activity slows down and the number of developers stabilizes (and often declines). These dynamics are similar to those considered in endogenous growth theory developed in Romer (1990). A management framework for this interaction between developers and open-source project growth is highlighted in Dey et al. (2024)—their CROSS model provides a foundation for understanding and enhancing the sustainability of OSS projects.

The term “open source” was coined on February 3, 1998, see Peterson (2018). Open-source software (OSS) has achieved amazing strides and its role in the economy is undervalued and uncounted in GDP as it is free. The Synopsis Report (Bals, 2024) calls OSS everything, everywhere, all at once. It notes that the percentage of commercial code bases containing OSS is huge, on average 96%. And 84% of codebases contain at least one open-source vulnerability.

There are two views on the value of open-source software. (i) The supply-side view that looks at the cost to create OSS, also known as the cost-based valuation. (ii) A demand-side view, i.e., the value created for the users of OSS, who do not have to build the software for themselves or pay for it. Hoffmann et al. (2024) estimate the supply-side value to be \$4.15 BN, and the demand-side value to be \$8.8 TN (which says that demand-side value is 2120x of supply-side value!). Developer time and cost can be measured from activity on OS repositories, but time and cost are extremely variable. Imputing labor costs to rewrite existing OSS arrives at estimates of \$38 BN in 2019 for the US (Robbins et al., 2021) and Euro 1 BN in 2021 (Blind et al., 2021; Blind and Schubert, 2024).

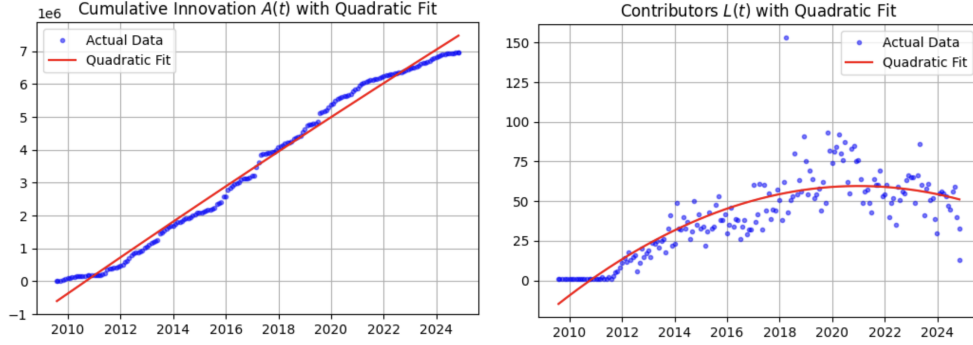


Figure 1: Plot of the cumulative lines of code changed (left plot) and the number of contributors (right plot) since the inception of the `pandas` library in 2009. Lines added and deleted are counted as work done on the repository. The dots represent the effect of each commit. A second-order polynomial trend line was fitted to the code changes data, resulting in the following best-fit equation:  $A(t) = -1.23 \times 10^{-2}t^2 + 1.52 \times 10^3t - 6.03 \times 10^5$ ,  $R^2 = 0.9880$ . The fit of the number of contributors over time is represented as the following best fit polynomial trend of order two:  $L(t) = -4.27 \times 10^{-6}t^2 + 3.56 \times 10^2t - 1.48 \times 10$ ,  $R^2 = 0.7153$ .

Assessments of value can be highly user-specific, and an interesting study reported in Chesbrough (2023) brings out the various features and value of usage of OSS, noting that most firms find that cost savings and faster development are the main drivers of usage<sup>1</sup> (Wladawsky-Berger, 2022, 2024).

We examine each commit to any chosen library over time and count the number of edited lines of code (added and removed) as a measure of technology progress in the repository. Take the example of a popular open-source library, `pandas`, released in 2009. `pandas` is a powerful Python library for data manipulation and analysis. The cumulative growth in the project,  $A(t)$ , is shown in the left plot in Figure 1, and the number of contributors  $L(t)$  per month is shown in the right plot. The models in this paper will calibrate dynamical equations that represent these growth and contributor patterns (not just polynomial fits as shown), which may then be used to model the future path of any repository.

The *key findings* are: (i) Developer engagement follows an S-curve pattern similar to product life cycles. (ii) Project growth elasticity with developer engagement varies between projects (positive or negative). (iii) Project growth is endogenous Romer (1990). (iv) The model forecasts the time to maturation of a project. (v) Life cycle information can help value an OS project. (vi) The ratio of project success to developer effort is highly variable. The study offers long-term implications for planning and developing software projects.

The rest of the paper proceeds as follows. In Section 2, we describe an initial dynamical equation used to model lines of code committed (added and deleted) to a repository. Section 3 models the life cycle of developer engagement. Section 4 calibrates the model to actual data for popular open-source projects and shows how to make lifetime projections (see also Appendix B). Section 5 makes simple suggestions about how the information generated from analyzing the project life cycle may be used to arrive at lifetime value. Section 6 provides closing comments.

## 2 The Endogenous Growth Model

This model presumes that technological progress is endogenous (Romer, 1994), driven by intentional investment decisions of agents (in our case, the developer ecosystem). Technology is treated as a non-rival good. Non-rival means that it can be used by multiple actors simultaneously. Further, in the open-source setting, goods are non-excludable, i.e., the owner does not exclude others from using it. In the open-source world the community owns the software and engagement is seamless. Most technologies tend to be excludable and often rival, which sets open-source software apart from other

<sup>1</sup><https://www.prnewswire.com/news-releases/linux-foundation-research-shows-economic-value-of-open-source-software-rising-in-terms-of.html>

avenues of technological progress. We use these ideas to present a framework for understanding how decentralized growth can drive technological progress.

We introduce a basic adaptation of an endogenous growth model here alongside a model for developer engagement (presented in Section 3). As noted previously, we define cumulative growth over time as variable  $A(t)$  and developers (contributors) as labor  $L(t)$ . We use the Cobb-Douglas production functional form for changes in project growth.<sup>2</sup> This version of the model defines the rate of change in OSS growth as a function of current growth levels and the number of developers working on the project. We assume that the rate of change in developers is a function of the current engagement in the project.

Let the dynamics of growth level  $A$  be described by the following function of labor  $L$  (contributors) and cumulative project growth  $A$ . This is known as the “standing on the shoulders of giants” effect (Isaac Newton, 1675), i.e., previous technological advancements drive new ones.

$$\dot{A} = \frac{dA}{dt} = \gamma L^\lambda A^\phi \quad (1)$$

Here,  $\gamma > 0$  is a constant “efficiency” parameter;  $\lambda$  is elasticity of labor; if  $\lambda = 1$  doubling  $L$  doubles the growth rate. If  $\lambda < 1$ , there are diminishing returns to adding more contributors. If  $\lambda > 1$ , there are increasing returns to adding contributors. Also,  $\phi$  is growth spillover. If  $\phi < 0$ , growth makes new growth harder (the “fishing out” effect). If  $\phi = 0$ , new growth is not a function of past growth. If  $0 < \phi < 1$ , there are positive but diminishing returns to existing growth. If  $\phi = 1$ , new growth is linear in existing growth. And if  $\phi > 0$ , growth is explosive. There are scale effects: If  $\lambda + \phi > 1$ , there are increasing returns to scale. If  $\lambda + \phi = 1$ , there are constant returns to scale. If  $\lambda + \phi < 1$ , there are decreasing returns to scale.

The model for developers (labor,  $L$ ) is described next.

### 3 An engagement model for the number of developers

We model the number of developers engaging with an open-source project using the Bass (1969) and Bass et al. (1994) models, which are used for product life cycles. This approach has so far not been applied to software project life cycles. The main idea of the model is that the engagement rate of a project comes from two sources: (1) The propensity of developers to work on the software project because it is apt for their use case or from other motivations, *independent* of the other contributors propensity to engage. (2) The additional propensity to work on the software because others have adopted or engaged with the project. This is social contagion, i.e., the influence of the early contributors becomes sufficiently strong so as to drive others to work on developing the software as well. This is a *network* effect. Once a project is underway and has seen some traction, engagement statistics may be used to develop a forecast of future engagement.

Define the probability of a developer working on an OS project in month  $t$  as  $f(t)$ . Then, the probability of engagement at time  $t$  is the density function  $f(t) = F'(t)$ , where  $F(t)$  is the cumulative probability. We can re-express this as a rate, conditional on no engagement thus far, i.e.,  $\frac{f(t)}{1-F(t)}$ . A functional form proposed for this rate is the diffusion equation

$$\frac{f(t)}{1-F(t)} = p + q \cdot F(t), \text{ or } \frac{dF/dt}{1-F} = p + q \cdot F, \quad F(0) = 0 \quad (2)$$

where  $p$  is the coefficient of independent engagement and  $q$  is the coefficient of imitation (network effect). We can solve for  $F(t)$  (Appendix B):

$$F(t) = \frac{p[e^{(p+q)t} - 1]}{pe^{(p+q)t} + q} \quad (3)$$

Differentiating, we get the engagement probability density at time  $t$ :

$$f(t) = \frac{dF}{dt} = \frac{e^{(p+q)t} p(p+q)^2}{[pe^{(p+q)t} + q]^2} \quad (4)$$

<sup>2</sup>[https://en.wikipedia.org/wiki/Cobb-Douglas\\_production\\_function](https://en.wikipedia.org/wiki/Cobb-Douglas_production_function)

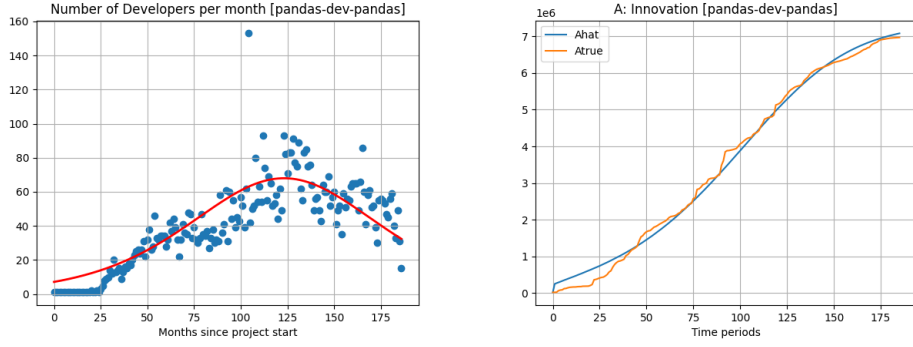


Figure 2: Plot of the developer engagement and project growth since the inception of the `pandas` library in 2009. The fitted line for engagement uses the solution in the equations above where it is determined that  $p = 0.00078$ ,  $q = 0.02833$ , and  $m = 9085$ . The growth plot shows the fit against the raw data after calibrating the ODE of the Cobb-Douglas model to the data. Best-fit parameters:  $\gamma = 143333165.608$ ,  $\lambda = 1.707$ ,  $\phi = -0.977$ . Time periods are months.

Denoting the lifetime engagement as  $m$ , then expected engagement at time  $t$  is  $u(t) = m \cdot f(t)$ . Using data, we want to estimate  $\{p, q, m\}$ . Cumulative engagement up to time  $t$  is  $U(t) = m \cdot F(t)$ . The goal here is to calibrate this equation to software developer engagement data to ascertain  $p, q$  and the lifetime engagement in developer months (denoted  $m$ ). Details are in Appendix B.

Next, we calibrate developer engagement and project growth. Models of growth have been used to value open-source projects (Horowitz et al., 2001; Hoffmann et al., 2024) and obtaining lifetime value begins with estimating lifetime growth.

## 4 Calibrating Developer Engagement and Growth to Data

We download all commits in a project from GitHub and count the number of lines added and deleted per month in each commit, from the inception of the project. We count the number of unique developers contributing to a project in every month. First, we fit the data to the model for developer engagement. Second, we fit the dynamics of the growth model to the data on commits and the data on developer engagement. Details are in Appendix B.

### 4.1 Fitting developer data

To illustrate, the model for developer engagement is fit to the data for the `pandas` library using the closed-form solution in Section 3. Figure 2 shows the monthly engagement data, and the left panel shows the engagement rate. The best-fit parameters are  $p = 0.00078$ ,  $q = 0.02833$ , and  $m = 9085$ . This calibration is straightforward because of the availability of the closed-form solution in Section 3. Estimates for a collection of popular open-source projects are presented in Table 1. We also note here that the parameters  $p, q$  have the same scale as those in the original Bass (1969) paper.

### 4.2 Fitting growth data

The behavior of the endogenous growth system will be a function of the three growth parameters  $\{\gamma, \lambda, \phi\}$  in equation (1), which is an ordinary differential equation (ODE). Using the solution to the ODE with a given set of these parameters enables tracing out the estimated function  $\hat{A}(t)$  over time using the data for commits and developer engagement. While  $\gamma$  is a scaling constant the parameters  $\lambda, \phi$  are elasticities with respect to developer engagement and project growth levels. The sign of these parameters informs us about how the project responds to infusions of each resource.

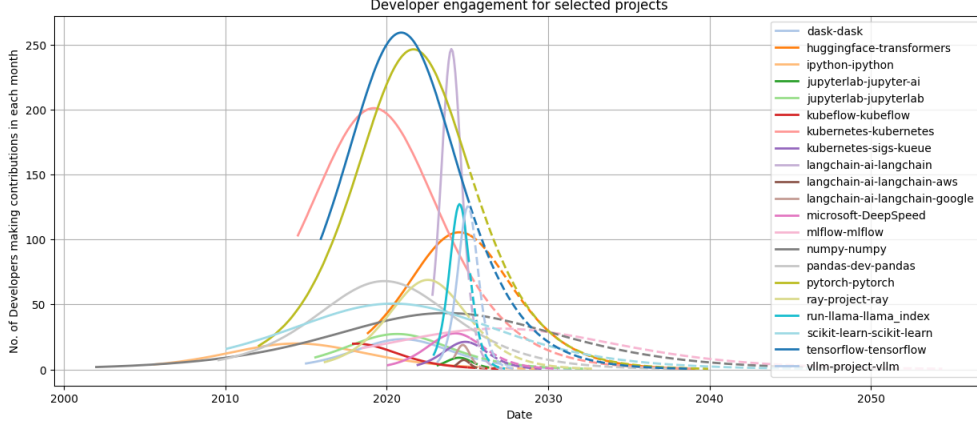


Figure 3: Developer engagement on open-source projects over time, see also Table 1. Engagement is measured as the number of developers who commit code each month. The engagement plots have a dashed line that starts end January 2025 and shows the projected future trajectory of developers engaging until the maturation of the project.

For calibration, we minimize the root mean squared error between the data and estimated functions from solving the ODEs. The calibration objective function is:

$$\min_{\{\gamma, \lambda, \phi\}} \frac{1}{T} \sum_{t=1}^T \left( \hat{A}(t) - A(t) \right)^2 \quad (5)$$

where  $T$  is the number of months in the data.

Numerically, each iteration of the minimizer calls the ODE solver internally to generate an estimated dynamical path for growth  $\hat{A}$ , which is then compared to the true path  $A$  by the optimizer. The ODE parameters are updated and the objective function is refined until parameters are found that minimize it. The solution is obtained in a few seconds and converges to the minimum. Once the optimal parameters are determined, these are used to plot the estimated functions against the actual commits data to assess the fit. For the pandas library, see Figure 2. We see a very good fit. Labor elasticity ( $\lambda$ ) is positive and growth elasticity ( $\phi$ ) is negative. Adding more contributors brings more growth, but growth eventually saturates as seen from the negative elasticity for growth. Additional projects calibrated display similar behavior and are reported in Appendix D and Appendix E.

### 4.3 Extrapolation

In order to assess the future trajectory of the project the ODE solutions may be projected further for additional months. The three projection plots for the pandas project are shown in Figure 4. The phase diagram shows the interaction between  $L$  and  $A$ , and we can see that as the project matures, growth grows without the need for a large number of additional contributors, consistent with the increasing returns and the long-run growth model in Romer (1986).

We can also ask when developer engagement will taper off and stabilize, as it inevitably must for all projects. Bessemer Ventures denoted this point as the “steady-state maturity” of the project (Droesch et al., 2020). We implement this by solving for the value of  $t$  that sets  $f(t) \cdot m = 0.5$  in equation (13). That is, engagement has dropped to a half-developer per month (no full time developer activity). Several project life cycles are shown in Figure 3. Figure 4 (middle plot) shows the start and end of developer engagement in addition to its life cycle forecast for the pandas project.

Table 1 shows the model parameters and life expectancy of each project. We see that the coefficients are similar to that obtained in work on product life cycles. More importantly, the  $p$  and  $q$  parameters are of similar size and scale across all projects, suggesting that the model applies well in the cross-section of open-source repositories. We also conducted a forecast stability analysis to see if the life

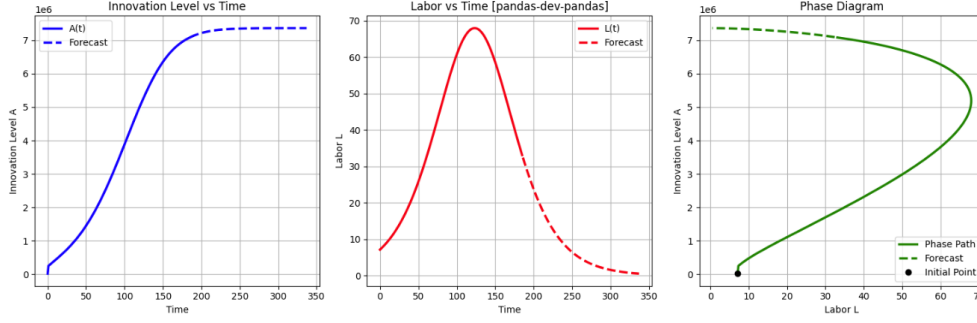


Figure 4: Extrapolation of the solution for the pandas repository till its maturation, which is defined as engagement dropping to half a developer per month. The phase diagram shows the interaction between  $L$  and  $A$ , and we can see that as the project matures, growth grows without the need for a large number of additional contributors.

Table 1: Fitted parameters of the developer model in section 3. The start date and end date of the data used for each project is shown. The columns in the table are as follows: 1.  $p$ : coefficient of independent propensity (probability) to join the project in any month. 2.  $q$ : coefficient of network intensity (probability) to join the project in any month. 3.  $m$ : calibrated developer-months expected over the project’s lifetime. 4.  $t$ : current life of project in months. 5.  $T$ : expected lifetime of project in months. 6.  $yr_s$ : at the current rate, expected remaining time to maturation of a project in years after the End Date of the project.

Project	Start Date	End Date	$p$	$q$	$m$	$t$	$T$	$yr_s$
dask-dask	2014-12-31	2025-01-31	0.00210	0.03841	2179.45720	122	200.48332	6.54028
huggingface-transformers	2018-10-31	2025-01-31	0.00270	0.03506	10381.61793	76	246.36845	14.19737
ipython-ipython	2005-07-31	2025-01-31	0.00139	0.02644	2724.42820	235	287.73291	4.39441
jupyterlab-jupyter-ai	2023-02-28	2025-01-31	0.01390	0.12720	229.97932	24	45.80086	1.81674
jupyterlab-jupyterlab	2015-07-31	2025-01-31	0.00351	0.03381	2648.21695	115	204.79843	7.48320
kubeflow-kubeflow	2017-11-30	2024-11-30	0.02223	0.02489	888.73764	85	109.63188	2.05266
kubernetes-kubernetes	2014-06-30	2025-01-31	0.00462	0.02596	22322.77493	128	297.80900	14.15075
kubernetes-sigs-kueue	2021-11-30	2025-01-31	0.00371	0.08465	920.70601	39	93.37730	4.53144
langchain-ai-langchain	2022-10-31	2025-01-31	0.01197	0.18074	4799.89380	28	53.44874	2.12073
langchain-ai-langchain-aws	2024-03-31	2025-01-31	0.03795	0.28159	80.22511	11	18.87810	0.65651
langchain-ai-langchain-google	2024-02-29	2025-02-28	0.03915	0.23507	235.98312	13	24.78449	0.98204
microsoft-DeepSpeed	2020-01-31	2025-01-31	0.00212	0.06705	1557.26118	61	127.95037	5.57920
mlflow-mlflow	2018-06-30	2025-01-31	0.00258	0.01409	6374.44742	80	432.94123	29.41177
numpy-numpy	2001-12-31	2025-01-31	0.00019	0.01721	9872.12333	278	596.11885	26.50990
pandas-dev-pandas	2009-07-31	2025-01-31	0.00078	0.02833	9085.33978	187	339.36140	12.69678
pytorch-pytorch	2012-01-31	2025-01-31	0.00065	0.03372	28125.42529	157	335.59761	14.88313
ray-project-ray	2016-02-29	2025-01-31	0.00105	0.04912	5377.82270	108	202.34332	7.86194
run-llama-llama_index	2022-11-30	2025-01-31	0.00448	0.19168	2535.01211	27	54.44984	2.28749
scikit-learn-scikit-learn	2010-01-31	2025-01-31	0.00164	0.01746	9697.09792	181	437.88354	21.40696
vllm-project-vllm	2023-02-28	2025-01-31	0.00239	0.19422	2524.36181	24	57.52794	2.79400

cycle projects vary depending on when in the life of the project the forecast is made. The analysis showing the stability of forecasts is in Appendix E.

## 5 Suggested Valuation Approaches

Table 2 reports lifetime developer engagement and growth, as well as illustrative (under assumptions below) supply-side costs for valuation of the ten projects portrayed in Figures 5 and 6. The current life of the project and forecast full life are shown. The project life is assumed to “mature” when developer engagement drops to half a developer a month. For many projects this is shown in Figure 3. We note that the project does not stop being downloaded and used, just that developer maintenance becomes minimal and reaches a low steady-state.

The number of developer months currently invested in the project is shown in Table 2. This is the sum of the number of developers each month for all months. Current cumulative growth is the total number of lines of committed code (additions and deletions) until current time and lifetime growth as forecast by the dynamic model is shown as well. Growth per developer month in the table is a useful

Table 2: lifetime statistics and estimated valuations for ten key open-source projects. Project life (time to maturation) is measured in months. “Dev Months” is developer months. Growth is in millions of lines of code committed. The “Innov/Dev Ratio” is the cumulative lines of code committed (both additions and deletions) divided by the cumulative number of developer months up to current life. The project value is calculated as Growth divided by the Innov/Dev Ratio multiplied by the cost per month per developer, assumed to be \$10,000, multiplied by 0.5, assuming that open-source developers spend 50% of their work time on any project. Valuation here relates to the production cost of the project, also known as its supply-side value.

Project	Current Life (months)	Full Life (months)	Current Cum Developer Months	Current Cum Growth (MM)	Lifetime Growth (MM)	Innov/Dev-Month (Ratio)	Supply-side Valuation Current (\$MM)	Supply-side Valuation To Maturation (\$MM)
dask-dask	122	200	1851	0.91	2.04	493.37	9.26	20.63
huggingface-transformers	76	246	5557	6.30	1774.14	1133.24	27.79	7827.72
jupyterlab-jupyterlab	116	206	2265	9.71	10.22	4285.25	11.32	11.92
kubernetes-kubernetes	128	298	19627	134.30	135.42	6842.38	98.14	98.96
langchain-ai-langchain	28	53	4455	8.77	8.77	1968.49	22.27	22.28
numpy-numpy	278	596	5495	18.31	30.97	3332.54	27.48	46.46
pandas-dev-pandas	187	339	7707	7.10	7.36	921.60	38.53	39.93
pytorch-pytorch	157	336	22518	23.12	61.17	1026.56	112.59	297.92
ray-project-ray	108	202	4369	8.24	10.27	1887.08	21.84	27.22
tensorflow-tensorflow	111	277	22021	451.34	451.64	20495.72	110.11	110.18

statistic that will be used for valuation of the projects and it varies widely across projects. This is calculated based on data up to current time.

## 5.1 Supply-side value example

The valuations shown in the last two columns of Table 2 are calculated as follows. Growth is divided by the Innov/Dev Ratio and then multiplied by 0.5 (by making an assumption that a developer spends about half their time on the project. This quantity is then multiplied by \$10,000 (assuming this to be the global average salary for a developer per month<sup>3</sup>). By using projected lifetime growth, this approach also gives the lifetime builder cost of the project. These costs vary significantly across the selected projects. The analysis here is illustrative of how an understanding of the project life cycle is the first step in assessing lifetime costs and value generation.

## 5.2 Demand-side value examples

Next, we evaluate the “demand-side” value of the project, i.e., what is the project worth to the users of the open-source software? To assess this value, we use downloads as a proxy and if we can determine a value per download, then knowing the lifetime downloads enables an assessment of this value. However, demand-side valuation is complicated and subjective (an approach to understanding the value of OSS is documented in Chesbrough (2023)).

The growth model (lines added and deleted) offers a lifetime forecast, as shown in Table 2. If we have an estimate of the number of downloads per lines edited for each project from existing data, we can estimate lifetime downloads. The ratio of PyPi downloads to lines of code changed varies across projects, with the popular projects having very high values compared to the less popular ones (Appendix F). For example, the ratio for Pandas is 62242. Since this project has a lifetime growth in code of 7.36 million lines edited, the lifetime downloads are expected to be 458 billion. Put a value on each download to determine what the project is worth.

Another simpler calculation may be as follows. We know that Pandas has 1,666,660,648 downloads in six months, and we can extrapolate this over the remaining life of the project, which is 158 months, i.e.,  $1666660648 \times 158/6 = 43,888,730,397$  downloads to go. The devil is in the assumptions, and whatever these may be, they should be consistently applied across time and projects.

The number of lines of code committed by a developer per month can vary, based on project complexity, development phase, individual coding style, team size and collaboration, and code review processes. Anecdotal estimates suggest that professional developers tend to commit between 10-50

<sup>3</sup><https://aijobs.net/salaries/developer-salary-in-2025/>

lines of code per day.<sup>4 5</sup> This translates to roughly 200–1000 lines of code per month, assuming 20 working days.<sup>6</sup> Admittedly, lines of code amended is a fuzzy measure of productivity because we could treat hundreds of lines of new code as more productive than debugging or refactoring with minimal new code. Experienced developers often write fewer, more efficient lines of code. Impact may also be measured subjectively as code quality or the impact on project goals.

## 6 Concluding Discussion

The paper adapts the endogenous growth model of Romer (1990) to the dynamic evolution of code growth levels and developer involvement in open-source software projects. Developer engagement is modeled using the Bass (1969) model, which fits the data well, suggesting that a product life cycle model is adaptable to modeling open-source project life cycles. This paper’s system of dual ODEs for growth and developer engagement is solved and calibrated to data on commits from GitHub repositories. Phase diagrams enable an assessment of the maturity of the project, its future trajectory, and the elasticities of growth and contributors on these dynamics.

We are able to speculate when developer engagement will drop below a single developer-month, without any further exogenous changes, signaling a point of low usage and activity. We applied this to some popular projects with interesting results. To the extent that these life cycle projections are within acceptable levels of accuracy, they may be used to forecast how long we may expect growth in and depend upon an open-source project (see Figures 3, 5, and 6). This complements the idea of underproduction risk noted in Champion and Hill (2021), Gaughan et al. (2024).

Experimentation across a few different systems of ODEs suggest some regularities, namely that growth feeds on itself, like a flywheel, and once this is in motion, additional developers do not matter as much and efficiency and downloads per unit of developer effort improves. This suggests why open-source projects are maintained by a small group of core developers, noted in Drosch et al. (2020). As projects mature and taper off in developer engagement as shown in Figure 7, we may detect successful mature projects as those that have a high download to developer effort ratio.

The dynamical equations for developer engagement and cumulative growth support illustrative valuation models for open-source projects. Supply-side valuation for ten sample projects ranges from the tens to hundreds of millions. Demand-side valuation can be imputed from lifetime growth using the ratio of downloads to lines of code changed. Future work can expand the number of projects analyzed. The analysis of strategic investments by tech companies in these projects may be possible with this model and its future extensions.

## 7 Customer Problem Statement

The customer is the leadership team. The dynamical models in this paper answer many questions relevant to formulating open-source strategy: (1) Can we understand developer engagement and is there are pattern across open-source projects? (Yes, the evidence shows the same S-curve as seen with product life cycles). (2) How does developer engagement interact with project growth? (We find that some projects’ growth elasticity with developer engagement is positive and for others it is negative, leading to two different trajectories). (3) Is project growth endogenous and conforming to a version of Romer (1990)’s growth model? (Yes). (4) Can we use the models for developer engagement and growth to project time to maturation of the project when it does not need additional resources than just a developer? (Yes). (5) Can life cycle information help in judging lifetime OS project value? (Yes, though it depends on valuation assumptions). (6) Is there a constant ratio of project success (downloads) to developer effort? (No, there is wide cross-sectional variation in project success relative to developer). This research has long-term impact on how software projects are planned, staffed, and developed over their life cycles.

<sup>4</sup><https://softwareengineering.stackexchange.com/questions/40100/how-many-lines-of-code-can-a-c-developer-produce-per-month>

<sup>5</sup><https://stackoverflow.com/questions/966800/mythical-man-month-10-lines-per-developer-day-how-close-on-large-projects>

<sup>6</sup><https://news.ycombinator.com/item?id=37040552>

## References

- Bals, F. (2024). 2024 Open Source Security and Risk Analysis Report (OSSRA) | Synopsis. Technical report, Blackduck, Inc.
- Bass, F. M. (1969). A New Product Growth for Model Consumer Durables. *Management Science* 15(5), 215–227. Publisher: INFORMS.
- Bass, F. M., T. V. Krishnan, and D. C. Jain (1994). Why the Bass Model Fits without Decision Variables. *Marketing Science* 13(3), 203–223. Publisher: INFORMS.
- Blind, K., S. Pätzsch, S. Muto, M. Böhm, T. Schubert, P. Grzegorzewska, and A. Katz (2021). *The impact of open source software and hardware on technological independence, competitiveness and innovation in the EU economy: final study report*. Publications Office of the European Union.
- Blind, K. and T. Schubert (2024, April). Estimating the GDP effect of Open Source Software and its complementarities with R&D and patents: evidence and policy implications. *The Journal of Technology Transfer* 49(2), 466–491.
- Champion, K. and B. M. Hill (2021, March). Underproduction: An Approach for Measuring Risk in Open Source Software. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 388–399. arXiv:2103.00352 [cs].
- Chesbrough, H. (2023, March). Measuring the Economic Value of Open Source: A Survey and a Preliminary Analysis. Technical report, The Linux Foundation.
- Dey, T., B. Fitzgerald, and S. Daniel (2024, September). CROSS: A Contributor-Project Interaction Lifecycle Model for Open Source Software. arXiv:2409.08267 [cs].
- Droesch, M., A. Karp, A. Sterman, and E. Kurzweil (2020, October). Measuring the engagement of an open source software community, <https://www.bvp.com/atlas/measuring-the-engagement-of-an-open-source-software-community>.
- Gaughan, M., K. Champion, and S. Hwang (2024, April). Engineering Formality and Software Risk in Debian Python Packages. arXiv:2403.05728 [cs].
- Hoffmann, M., F. Nagle, and Y. Zhou (2024). The Value of Open Source Software. *SSRN Electronic Journal*.
- Horowitz, E., R. Madachy, D. Reifer, B. Steece, and B. K. Clark (2001, January). *Software Cost Estimation With Cocomo II* (HAR/CDR edition ed.). Upper Saddle River, NJ: Prentice Hall.
- Peterson, C. (2018). How I coined the term 'open source' | Opensource.com.
- Robbins, C. A., G. Korkmaz, L. Guci, J. B. S. Calderón, and B. L. Kramer (2021). A First Look at Open Source Software Investment in the United States and in Other Countries, 2009-2019. In *IARIW-ESCoE Conference "Measuring Intangible Assets and Their Contribution to Growth"*, London.
- Romer, P. M. (1986). Increasing Returns and Long-Run Growth. *Journal of Political Economy* 94(5), 1002–1037. Publisher: University of Chicago Press.
- Romer, P. M. (1990). Endogenous Technological Change. *Journal of Political Economy* 98(5), S71–S102. Publisher: University of Chicago Press.
- Romer, P. M. (1994). The Origins of Endogenous Growth. *The Journal of Economic Perspectives* 8(1), 3–22. Publisher: American Economic Association.
- Wladawsky-Berger, I. (2022, March). The Impact of Open Source on the EU Economy.
- Wladawsky-Berger, I. (2024, April). What's the Value of Open Source Software Based on Actual Usage Data?

## Appendices

291 **A** *Solving the initial differential equations for project growth and developer*  
 292 *engagement.*

293 The dynamics of growth level  $A$  are described by equation (1), reproduced here.

$$\dot{A} = \frac{dA}{dt} = \gamma L^\lambda A^\phi \quad (6)$$

294 The equation for developer engagement (labor) is reproduced here as well:

$$\dot{L} = \frac{dL}{dt} = nL \quad (7)$$

With initial conditions:

$$L(0) = L_0, \quad A(0) = A_0$$

295 The solution to this system of equations is as follows:

1. First, let's solve the equation for Labor ( $L$ ) since it's independent of  $A$ :

$$\dot{L} = nL$$

This is a simple separable differential equation:

$$\frac{dL}{dt} = nL$$

$$\frac{dL}{L} = n dt$$

$$\int \frac{dL}{L} = \int n dt$$

$$\ln |L| = n \cdot t + C$$

Using the initial condition  $L(0) = L_0$ :

$$L(t) = L_0 e^{nt}$$

2. Now we can substitute this solution into the equation for  $A$ :

$$\dot{A} = \gamma (L_0 e^{nt})^\lambda A^\phi$$

This is a Bernoulli differential equation in  $A$ :

$$\frac{dA}{dt} = \gamma L_0^\lambda e^{\lambda nt} A^\phi$$

3. To solve this, let's use the substitution  $u = A^{1-\phi}$ :

$$\frac{du}{dt} = (1-\phi) A^{-\phi} \frac{dA}{dt}$$

Substituting:

$$\frac{du}{dt} = (1-\phi) \gamma L_0^\lambda e^{\lambda nt}$$

4. Integrating both sides:

$$u = \frac{(1-\phi) \gamma L_0^\lambda}{\lambda n} e^{\lambda nt} + C$$

5. Substituting back  $u = A^{1-\phi}$ :

$$A^{1-\phi} = \frac{(1-\phi) \gamma L_0^\lambda}{\lambda n} e^{\lambda nt} + C$$

6. Using the initial condition  $A(0) = A_0$ :

$$A_0^{1-\phi} = \frac{(1-\phi)\gamma L_0^\lambda}{\lambda n} + C$$

$$C = A_0^{1-\phi} - \frac{(1-\phi)\gamma L_0^\lambda}{\lambda n}$$

7. Therefore, the complete solution for both processes is:

$$L(t) = L_0 e^{nt} \quad (8)$$

$$A(t) = \left( \frac{(1-\phi)\gamma L_0^\lambda}{\lambda n} e^{\lambda nt} + A_0^{1-\phi} - \frac{(1-\phi)\gamma L_0^\lambda}{\lambda n} \right)^{\frac{1}{1-\phi}} \quad (9)$$

This solution is valid for  $\phi \neq 1$ ,  $n > 0$ . If  $\phi = 1$ , the equation would need to be solved differently. Also, for  $n = 0$ , the solution is shown in equation (23). To check the solution by differentiation, set equation (9) to  $A(t) = u(t)^{1/(1-\phi)}$ . so that  $\frac{dA}{dt} = \frac{1}{1-\phi} \cdot u^{\frac{\phi}{1-\phi}} \cdot \frac{du}{dt}$ , and then proceed to recover equation (1) in a few steps of algebra.

The overall solution shows that: (a) Labor grows exponentially at rate  $n$ . (b) The growth level  $A$  grows as a function of both the initial conditions and the parameters  $\gamma$ ,  $\lambda$ ,  $\phi$ , and  $n$ .

## B Solving and calibrating the differential equation for developer engagement

Define the probability of a developer working on an OS project in month  $t$  as  $f(t)$ . Then, the probability of engagement at time  $t$  is the density function  $f(t) = F'(t)$ , where  $F(t)$  is the cumulative probability. We can re-express this as a rate, conditional on no engagement thus far, i.e.,  $\frac{f(t)}{1-F(t)}$ . A functional form proposed for this rate is the diffusion equation

$$\frac{f(t)}{1-F(t)} = p + q \cdot F(t) \quad (10)$$

where  $p$  is the independent rate of engagement (the coefficient of independent engagement) and  $q$  modulates the network effect (the coefficient of imitation). The idea is that engagement is driven by a developer's independent need for the software package and also driven by awareness from engagement by other developers. The goal here is to fit this equation to software developer engagement data to ascertain  $p$ ,  $q$  and the lifetime engagement in developer months (denoted  $m$ ). We rewrite the equation above as

$$\frac{dF/dt}{1-F} = p + q \cdot F, \quad F(0) = 0 \quad (11)$$

which is a differential equation with an initial condition, which we can solve for  $F(t)$ , provided here:

$$F(t) = \frac{p[e^{(p+q)t} - 1]}{pe^{(p+q)t} + q} \quad (12)$$

Differentiating, we get the engagement probability density at time  $t$ :

$$f(t) = \frac{dF}{dt} = \frac{e^{(p+q)t} p(p+q)^2}{[pe^{(p+q)t} + q]^2} \quad (13)$$

Denoting the lifetime engagement as  $m$ , then expected engagement at time  $t$  is  $u(t) = m \cdot f(t)$ . Using data, we want to estimate  $\{p, q, m\}$ . Cumulative engagement up to time  $t$  is  $U(t) = m \cdot F(t)$ . These are calibrated to the data as explained in Appendix B.

Substituting these into the diffusion equation (10) above we have

$$\frac{u(t)/m}{1-U(t)/m} = p + q \cdot U(t)/m \quad (14)$$

Table 3: Calibrated parameters for the growth model,  $\frac{dA}{dt} = A \cdot L^\lambda \cdot A^\phi$ .  $\gamma$  is a constant,  $\lambda$  is the elasticity of growth with respect to developer engagement, and  $\phi$  is the elasticity of growth with respect to current growth.

Project	$\gamma$	$\lambda$	$\phi$
dask-dask	666550.65	-0.5419	-0.2432
huggingface-transformers	19799.16	-0.9491	0.3716
jupyterlab-jupyterlab	73686.07	1.5335	-0.2960
kubernetes-kubernetes	17172.65	2.7311	-0.5453
langchain-ai-langchain	0.05	6.4637	-1.2194
numpy-numpy	219002.65	0.5762	-0.1793
pandas-dev-pandas	143333165.61	1.7070	-0.9769
pytorch-pytorch	12.86	0.2813	0.5119
ray-project-ray	1665.11	0.9161	0.0301
tensorflow-tensorflow	0.01	7.0426	-0.3981

Re-arranging we have

$$\begin{aligned} u(t) &= [p + q U(t)/m][m - U(t)] \\ &= \beta_0 + \beta_1 U(t) + \beta_2 U(t)^2 \end{aligned} \quad (15)$$

$$\beta_0 = p m \quad (16)$$

$$\beta_1 = q - p \quad (17)$$

$$\beta_2 = -q/m \quad (18)$$

If we have data on periodic engagement  $u(t)$  and cumulative engagement  $U(t)$  we can fit a regression to the data to get coefficients  $\beta_0, \beta_1, \beta_2$ . We can then solve for  $p, q, m$  as follows. Given that

$$\beta_1 = q - p = -m \beta_2 - \beta_0/m, \quad (19)$$

we re-arrange to get a quadratic equation

$$\beta_2 m^2 + \beta_1 m + \beta_0 = 0 \quad (20)$$

with solution for lifetime engagement  $m$ :

$$m = \frac{-\beta_1 \pm \sqrt{\beta_1^2 - 4\beta_0\beta_2}}{2\beta_2} \quad (21)$$

The positive root of  $m$  may then be used to solve for

$$p = \beta_0/m, \quad q = -m\beta_2 \quad (22)$$

## C Adaptation to Closed Source Software

These interactions in the open-source ecosystem may be used to estimate headcount needed to develop closed-source software using a simpler model with only equation (1), given that developer labor will be set to a constant  $L$ . The solution is

$$A(t) = \left[ (1 - \phi) \left( \gamma L^\lambda \cdot t + \frac{A_0^{1-\phi}}{1 - \phi} \right) \right]^{\frac{1}{1-\phi}} \quad (23)$$

This equation can be calibrated to the data for growth by modulating the parameters  $\gamma, \lambda$ , and  $\phi$  for a fixed level of  $L$  using the same solution approach from the previous subsection. This is simpler than modeling the dynamics of open-source software projects. Extensions to the model for how developers are added to the closed-source project would be internal to and vary by organization.

## D Calibration of ten open-source projects

For ten projects shown in Table 3 we see that most of the projects have a positive elasticity with developer engagement, where  $\lambda > 0$  and a negative one with growth level,  $\phi < 0$ . This means that, over time, projects get a stable pool of contributors who improve their skills in the project and are able to accelerate growth ( $\lambda > 0$ ). This is good because over time, developer engagement peaks and then declines. Because  $\phi < 0$ , as the project matures and cumulative growth level  $A$  grows, the pace at which lines of code are added or deleted slows down. A similar pattern has been noticed by venture capitalists (Droesch et al., 2020).

## E Forecast Stability

Since the model may be calibrated to data at different points in the life of the project, and then used to forecast the future life of the project till its demise, it is useful to examine how stable these forecasts are for the time of maturation. Of course, as the time moves on, we gather more information about a project, which will mean making adjustments to the forecast of both developer engagement and growth levels. In order to assess how these forecasts change, we examined ten well-known open-source projects as follows. First, we calibrated the model using all the data about the project from its inception until the current time (end January 2025 at the time of data collection). Second, we plotted the fitted developer engagement and fitted cumulative growth and compared these to the actual data. As shown in Figures 5 and 6, the fit of both the developer engagement model and the cumulative growth model is very good across most projects.

Third, we then used only data for the first 75% of the project's current life. For a project like `pandas`, which began in 2009, this would mean dropping the most recent four years to get estimates. In the second row of Figure 6, we see that the fitted developer engagement model (left plot) is almost the same when calibration uses full data (red line) versus when only the first 75% of the project's lifetime data is used (green line). When considering the cumulative growth fit (right plot) the two lines are almost identical. Examining both figures 5 and 6 reveals that the future projections for the project are robust to the timing of calibration. Even for projects with a short life so far, this does not matter for estimating growth dynamics, as they seem to fit very well regardless of when the calibration is done. However, for estimating developer engagement, sometimes the calibrations do lead to different fitted dynamics as in the case of `pytorch` and `tensorflow`, though for most of the other projects, even for developer engagement the calibrations at different times lead to similar developer engagement plots. Researchers may use these forecasts to support valuation of these projects, so the fact that robust estimates of lifetime dynamics are available early in a project's life is encouraging.

In Section 5 we will use these dynamic models for developer engagement and growth to suggest methods for valuation of OSS projects, though we surmise that these would vary based on assumptions made by the valuer.

## F Project popularity and developer effort

If we have an estimate of the number of downloads per lines edited for each project from existing data, we can estimate lifetime downloads. The ratio of PyPi downloads to lines of code changed varies across projects, with the popular projects having very high values compared to the less popular ones as shown in Figure 7.

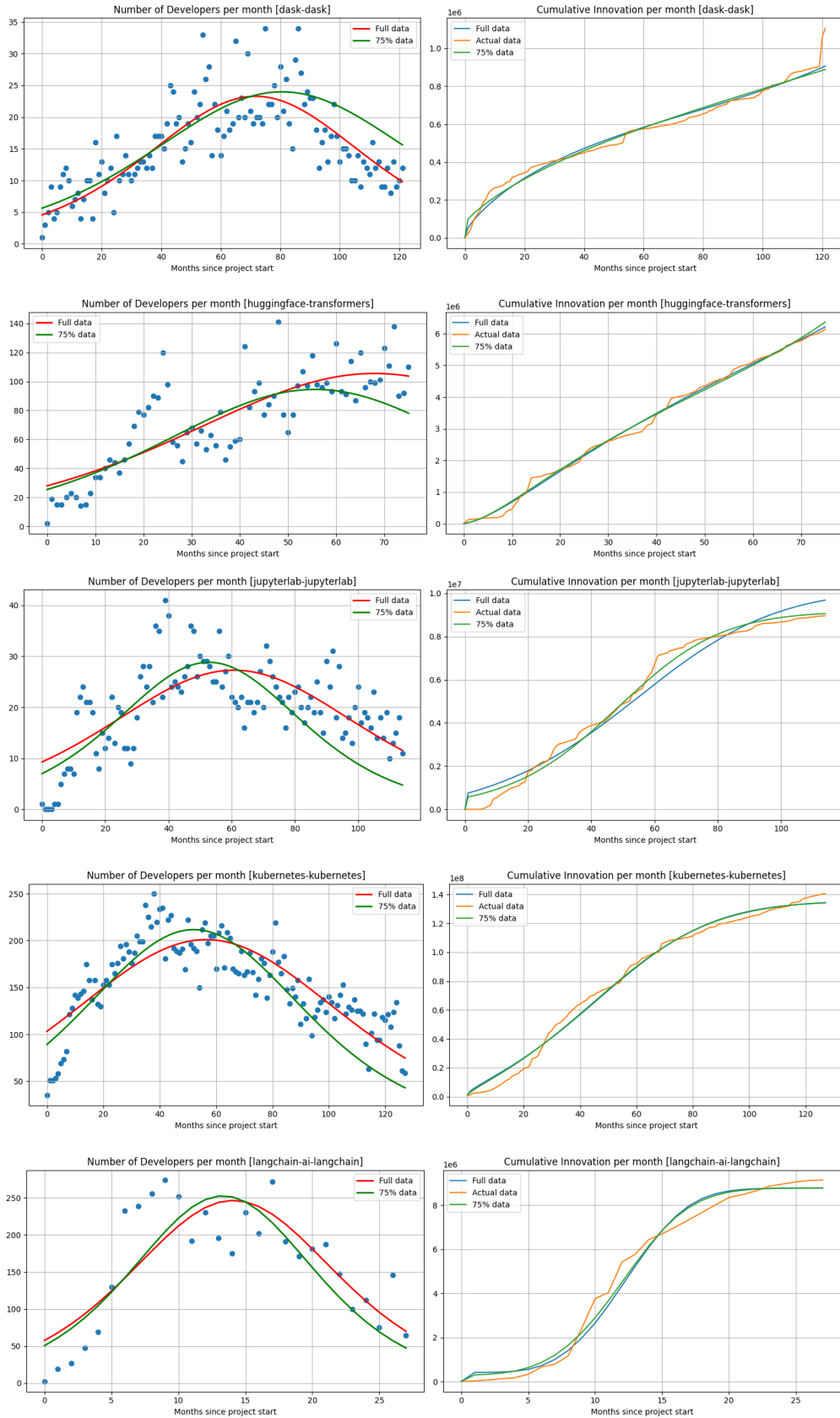


Figure 5: Model calibration I: The plots show the fitted values to developer engagement data (left) and cumulative growth (right). The actual data is shown alongside the fitted data when the fit is undertaken using both, the entire data sample and the first 75% of the data available for each project.

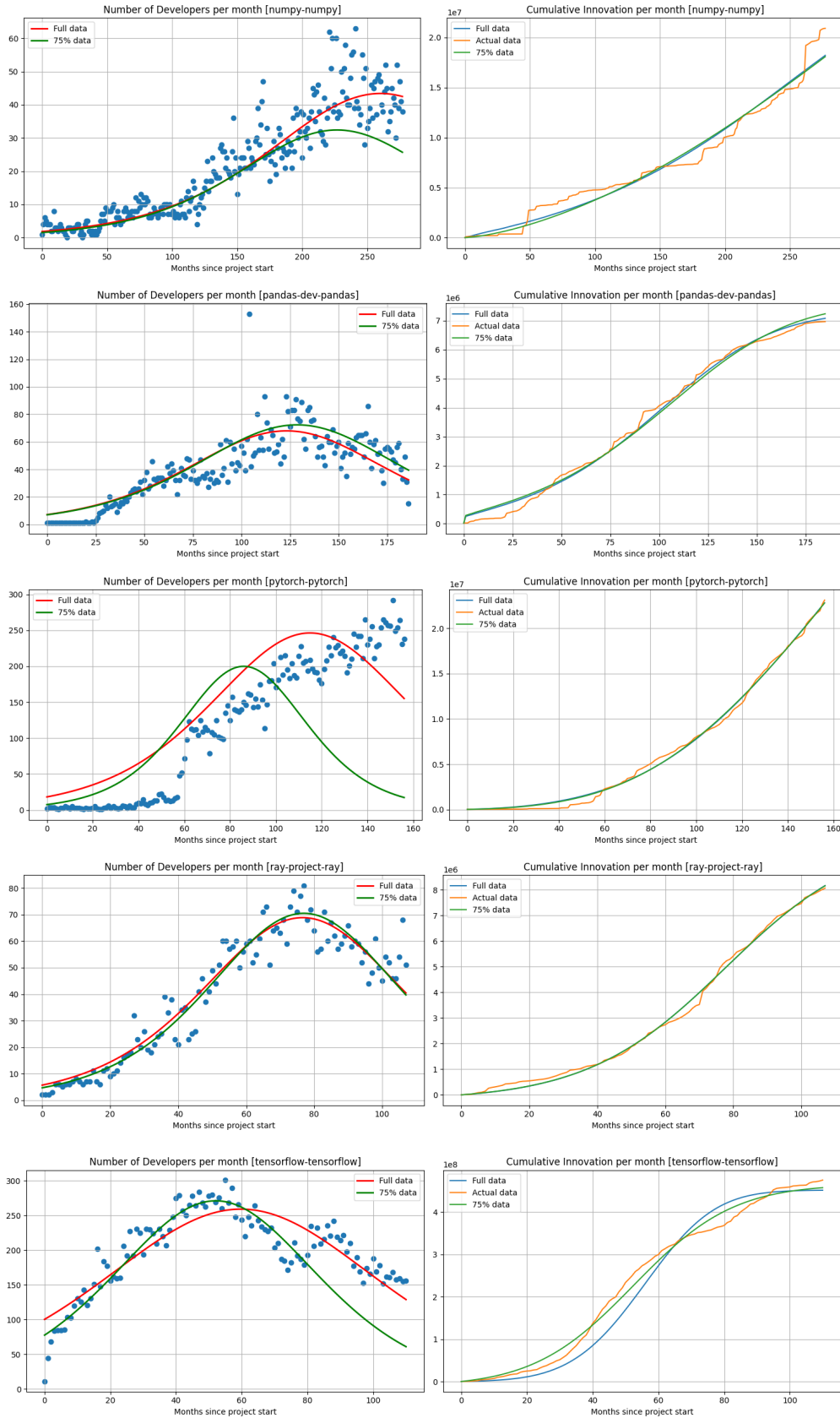


Figure 6: Model calibration II: The plots show the fitted values to developer engagement data (left) and cumulative growth (right). The actual data is shown alongside the fitted data when the fit is undertaken using both, the entire data sample and the first 75% of the data available for each project.

Figure 7: The ratio of PyPi downloads to lines of code changed (additions and deletions). These ratios are determined by dividing the total downloads over the past 6 months by the total lines of code added and deleted over the same time frame. These ratios offer insight into the usefulness of projects to the community around them for the degree of effort invested by developers.

