The Practices of Unpaid Third-Party Developers – Implications for API Design

Completed Research Paper

Daniel Rudmark
University of Borås / Viktoria Swedish ICT
daniel.rudmark@hb.se

ABSTRACT

To draw on the innovation capabilities of third-party developers many organizations are currently deploying open application programming interfaces (API's). While third-party services may offer commercial opportunities for independent software firms, a large portion of existing third-party software are undertaken without any financial compensation. Although unpaid developers offers a potential source for innovation of end-user services, the current literature has largely overlooked how these unpaid actors use and appropriate the technology provided by organizations. To this end, this research pays specific attention to the specific practices of unpaid developers. The data used for analysis were collected through a programming contest – a hackathon – where unpaid developers gather to craft end-user services. Through an ethnographic lens we present a number of recurrent activities and patterns of action employed by developers and from this analysis we present implications for API designers seeking to attract unpaid developers.

Keywords (Required)

Third-Party Development, API, Intrinsic motivation, Video observation

INTRODUCTION

Information technology has provided apt opportunities for organizations to interact with their customers. Industries such as banking, travel and music have undergone dramatic transformation in the last decades and the success of actors in these sectors is now at large depending on successful management of the organization's digital presence. However, as users acquire more devices and utilize ICT in less predictable contexts, this development also faces organizations with a dilemma: how can information and services be supplied pervasively given the user diversity and the limited amount of resources available for development? To this end, many organizations open up their products and services to enable outsiders to cater for end-user services. One source of external innovation is *third-party developers*, a term which signifies an independent actor which develops software intersecting between a core service or product and the end-user.

While major platforms may offer new business opportunities for commercial firms (Ceccagnoli et al., 2012), much of the services and applications crafted by third-parties are delivered to end-users without any financial payback to developers. E.g. Boudreau & Jeppesen (2011) reports that shortly after Microsoft released its WindowsPhone platform, more than 70% of the applications were offered at zero cost for the user, and that many of these lacked any other complementary sources of income (such as advertising). Similarly, much of the innovation taking place in the space of open data, i.e. where foremost governmental agencies but also commercial firms provides access to organizational data for third-party developers, is done without any expected monetary returns (Kuk & Davis, 2011).

Given the importance of such unpaid labor, there is surprising dearth of knowledge on how unpaid developers appropriate the resources offered by organizations. Such resources include API's– internet-enabled access to services for developers. Within IS research there is a long tradition of arguments that an incongruence between the practices of a given context and the implementation of a technology will lead to poor adoption of technology, resulting non-use or creative circumvention of designers' intentions (c.f. Grudin, 1991). By using the term 'practice' the emphasis is on what people actually do rather than what they say or what they ought to do (Schultze & Boland, 2000). Since the current literature on unpaid third-party developers have focused on expected effects of engaging unpaid developers (Boudreau & Jeppesen, 2011), the motivational profiles (Bergvall-Kåreborn et al. 2011) and the value chain of developers participating to transform "raw" data to useful end-user services (Kuk & Davis, 2011), there is currently a lack of understanding of how developers actually work when constructing end-user services from API's. To this end, this paper seeks to investigate the following research question: *What are the practices of unpaid developers and how do they appropriate API's?*

The paper is structured as follows: We begin by presenting the existing literature on why software development is performed without monetary compensation. Next we describe how we during a programming contest collected and analyzed data on unpaid developers' work practices and the findings made in this vein. Based on the work practices of unpaid developers we finally move into to present design implications for organizations seeking to tap into these unpaid developers.

WHY DO DEVELOPERS WORK FOR FREE?

Researchers has long acknowledged that a large portion of the motivation to develop software is to be found aside from being paid. To be able to draw in this unpaid labor however, organizations need to find ways to disburse these "invisible wages" of software development. The literature on unpaid development offers several broad categories explaining why such work is undertaken (Boudreau & Jeppesen, 2011).

Just as for commercial third-party developers, *signaling* has been identified as an important incentive. However, when commercial firms whish to signal technology compatibility and platform owner acceptance to prospective customers (Ceccagnoli et al., 2012), developers working without financial remuneration are seeking to signal mastery and expertise to fellow developers within an innovation community (Shah, 2006). E.g. in the open source community, by being promoted to a "committer" – a developer which is allowed to submit code to the main code repository – the reputation and status of that developer within the community is strengthened which in turn further motivates such an individual.

Another important driver for developing without payment concerns *social norms*. According to these norms, the source code of a software product should always be publically available for installation, inspection and modification and developers thus continues to work from a sense of obligation to the community to sustain such open source alternatives (Shah, 2006).

Finally, the sheer *enjoyment of developing software* has been highlighted as a salient motivator. (Lakhani & Wolf, 2003, Almstrom, 2003, Sharp et al., 2008, Shah, 2006). In particular, overcoming *challenges* during programming is one of the most noted reasons for why people enjoy software development (Sharp et al, 2008, Almstrom, 2003). Being able to apply existing skills or learn new ones to resolve a novel problem has been noted to be sufficiently rewarding for developing software, irrespective of the outcome's utility (Lakhani & Wolf, 2003, Shah, 2006). However, for a developer to appropriate

such an reward the challenge needs to fit the skills of the developer. If the task lies beyond the skills of the developer, the challenge may invoke frustration or anxiety. Conversely, if the problem can be overcome routinely, the reward is not attained but replaced by a sense of boredom.

METHOD

Research Site and Data Collection

The site used for the data collection was called West Coast Travelhack 2013, a development contest spanning 24 hours. The goal of the competition was to generate prototypes for innovative digital services supporting citizens in their everyday travel. More specifically, the prototypes were to support travelers in making more sustainable choices in their everyday travel, e.g. choosing car-sharing over lone driving, public transport over car-sharing, bicycling over public transport etc. The contest was organized as a part of a research project, comprising research institutes, universities, transport authorities and commercial firms. The competition summoned 76 contestants distributed on 20 teams and each prototype was measured towards a defined set of criterion by an expert jury. The contest criterion included e.g. the degree of innovativeness compared to existing services, to what extent the service resolved known issues among existing travelers (such little daily usage of technology, attitudes towards using public transport etc.) as well as a plan explaining how the prototype were to be sustained over time and distributed to users.

A programming contest such as Travelhack provide an excellent venue for studying the work practices of unpaid developers. First, the teams willingly spent a weekend (the total length of the event stretched from Saturday 9 am to Sunday 3:30 pm) developing third-party services without any financial remuneration. Second, as a unpaid third-party developer activities typically takes place in the homes at odd hours these activities are inherently difficult to get close to – a necessity for practice studies. Therefore programming contests where unpaid developers gather to develop software, offers an apt opportunity for conducting such studies.

As suggested for studies in this vein (c.f. Robinson et al, 2007), a multitude of data sources were used to interpret the practices. The foremost source of empirical material was video recordings, in total spanning more some 57 hours. A few weeks prior to the contest the teams were inquired about participating in the study. Of six inquired teams four teams accepted to participate under the conditions given. Of these four, one team was assessed prior to data analysis to fall outside the scope of study as they did not use any external data sources. The video recording further captured 6 informal interviews performed by the authors during the contest. In addition to the videos, the analyzed data included the 17 used data sources , the source code from two of the three teams and log files from 5 frequently used API's.

Data analysis

All recorded video was imported into a computer-based video analysis tool, Transana. Transana allows the analyst to time code the recordings, transcribe the recorded material, sync transcriptions and video content and assign codes to excerpts of the recordings. After importing the videos a first iteration containing a full screening of the 57 hours and transcribing relevant sections of the video recordings were conducted. During this process we also took notes regarding preliminary interpretations concerning patterns found in the developer practices. In the second iteration we inductively coded the transcribed material (Corbin & Strauss, 1990). As a complementary resource, we used the source code, API log files and the data sources used. E.g. inspection of the log files helped the interpretation of what developers was talking about when having trouble retrieving data from an API or if they actually succeeded calling the API. Through constant comparison of events across teams (Klein & Meyers, 1997; Shultze & Boland, 2000) we arrived at concepts closely related to those presented in this paper.

Since avoiding a priori conceptions of patterns is an important aspect of understanding a practice (Hammersley, 2007) we only at this late stage visited the existing literature on why developers work for without being paid. The literature increased the understanding and aided in the refinement of the developed code structure. In what follows we present the patterns that emerged from our analysis.

FINDINGS

In this section we present two major areas of unpaid third-party developer practices. The first, *autonomous innovation*, denotes the observed rationales and discourses shaping the overall design of the service. The second, *appropriating datasets*, describes the practices employed when working the data sets during the contest.

Autonomous innovation

As the teams arrived to the contest quite they had an idea on what to develop and after listening to the introductory contest presentations, a process of cross-checking their initial ideas towards the information given in the presentations followed. While Team A and B pursued their original ideas after the cross-checking, Team C completely changed the course of their development efforts during the first hours of the competition. The reason for this shift was found in perceived similarities with an existing service. The discarded idea concerned ride-sharing and how employers through their employee directories were able to cater for more efficient ride-sharing among their staff as they possessed information of the home locations of each employee and thereby were able to identify suggestions for ride-sharing which may not be obvious to individual workers. Further, the idea rested on an assumption that since trust is considered a major obstacle for travelers to engage in ride-sharing, the shared identity from being employed by the same organization would work as a trust builder. As the team during the presentation became aware that a commercial firm sponsoring the contest based its business on a related idea, the team assessed that they would not be able to develop a more innovative service within the contest timeframe. Since novelty relative to existing services was a contest criteria, the team instead chose to implement a web-based widget that organizers of music festivals, sports events and the likes easily could incorporate into their web sites. The widget presented different travel options for potential spectators on how to get to the event by car, by sharing a ride with other spectators or by using public transportation. Also, an estimation of each travel option's CO2 emissions was presented.

Team A developed a web-based service for users who were about to relocate. By the entering the addresses for the future work location and potential residencies, various social and commercial services in each surroundings were presented. The data included information about nearby entities of interest such as schools, healthcare institutions, public transport infrastructure, parking opportunities, retail and recreational activities. Team B wanted to turn around the typical behavior of public transport travel planners. The user would enter a preferred departure time, the location from which they were leaving and where they whished to travel. As everyday travel often includes a well established pattern of route selection and departure time, the idea was that the service instead only would signal in the case of traffic disturbances. In the event that the typical route would not get the user to work on time, the service would inform the user about alternative ways of travelling to the workplace.

During the contest all teams discussed the commercial potential of their services. These discussions originated from one of the contest criterion, to present a business model connected to the service. The business model followed the structure of Osterwalder (2004) where key business characteristics such as revenue streams, distribution channels and key partners are to be defined. While the contestants acknowledged the business model as an important part of the contest submission, the activity of finishing the model was carried out opportunistically, and no substantial interplay between the service design and the business model was observed. The activity was instead discussed as e.g. "mumbo-jumbo" "business bullshit" amongst team members across groups. One of the teams had the following discussion about the commercial potential, some 7 hours into the competition:

> *D1: We will not make any money from this right? It will be totally free?*
>
> *D2: Mm*
>
> *(Laughter by team members)*
>
> *D3: I mean this thing is free to use, but when an organization uses it they have shown awareness of environmental issues so that way you might be able to...but why would you pay for it?*
>
> *D1: It doesn't really feel like there is an incentive to pay*
>
> *D2: We always start from the market, that's the best thing about us, that sort of appears afterwards*
>
> *(Laughter by team members)*
>
> *D1: But what about [revenue streams in the business model canvas]? What should I put there?*
>
> *(Laughter by team members)*

A similar pattern was noted when teams discussed a contest resource representing user needs – personas. The personas was grounded in substantial research efforts comprising several years work of quantitative and qualitative inquiries of users' travel habits and perceptions about travel options. The reason for the contest organizers to incorporate personas into the contest was to highlight relevant issues that a diverse set of travelers face when choosing the means of transportation in their everyday travel. By introducing personas the idea was to align the service development with known user issues and thereby avoid contest submissions to be based on participants' own preferences and needs. Despite these efforts by the organizers, the

teams used different strategies in their reasoning to stick with their original idea, even when it did not fit the needs of the personas. One team introduced a whole new persona whose needs would fit the service being developed and further broadly assessed that the service would fit all other personas. Another team cross-checked the existing personas and assessed that the service would match three of the given personas and stuck with the original plan. The third group dismissed the specific characteristics of the personas as a whole and legitimized their decision through that the service instead would help to change the personas' current attitudes to e.g. public transport.

The vast majority of the contest however, developers spent their time programming, including accessing and processing a wide array of datasets. Given the centrality of this activity and the purpose of this paper, the patterns found in this vein are presented below.

Appropriating datasets

Data was retrieved via two techniques, either through dedicated web-based API's or by scraping. Scraping is a technique to extract data from an open data source which is not intended for programmatic access. We observed how scraping typically was performed towards a web page, but also other sources such as publically available XML-files were used.

From our analysis we derived three distinct recurring interlocking activities, whose outcome significantly affected how services were developed and how different API's was perceived and adopted by developers. The first stage concerned *gaining access* to the data source. In the case of API's this typically meant acquiring an API key (for authenticating requests to an API) and retrieving the "base URL" of the API. Further, *assessing API functionality* was observed. Here developers sought answers around issues such as an overview of what an API afforded, specification of function calls and responses and the used formats. Finally, when *working with datasets* developers invoked method calls, processed the response and integrated the processed data into the service.

An often recurrent phenomena in our data set was the outburst of emotions when developers either failed or succeeded to complete a task. E.g. when developers succeeded to make a first live quest, using the base URL and the API key, they typically displayed clear signs of positive emotions:

> *D4: Damn that's great, now GET is working! Cheers!*

Conversely, when developers were unable to complete a task they expressed dislike with the API's design. E.g. in our analyzed material we noticed that a critical passage point often was team members' development environments. As all actions connected to the datasets had to be marshaled through the programming tools used by contestants, both frustrations and excitement were often connected to the level of compatibility between the data sources and the development environment. In following excerpt, developers struggle to integrate a result set into the service, based on the incompatibility between the dataset and the libraries used by developers:

> *D1: Our parking services, do they work now?*
>
> *D2: Well, this "handicap parking"…data is returned but it's in a freaking bad format. There's no quotations around their [float numbers] and that screws everything up.*
>
> *D3: Have we only gotten developer-preview-alpha versions of the API's or what? [Another team developer] sounded a little bit frustrated with the API's he was messing with. It's bloody difficult to build services then…*
>
> *D2: I mean, sure, you can always make a hack and bypass [the integration problem] but then you need to do it on every single service and they all work in quite different ways.*
>
> *D3: It's kind of the same as in [another hackathon] where we ended up cleaning the document structure instead of [building the actual service]*

Although different teams worked with the same API's, they assessed such problems differently. Where one team got stuck and dropped the API from the service, another team could fairly easy circumvent the hurdle and thus did not experience the API in the same negative way. In other words, a barrier was not given but rather a result of previous developer experiences, but when they occurred they invoked similar negative responses across teams. Moreover, we observed how developers when faced what was perceived as a mundane task, worked to push that activity to the next level of expertise, even though it was not required to finish the task at hand. E.g. one developer managed to quickly get the results he wanted from a public transport API (which a developer in another team unsuccessfully struggled with) but then pursued integration with an external web service to induce a "cleaner look" of the code (even though he admitted to the other team members that this added little value to the service).

The second pattern of data set-related action that surfaced in our analysis was *delegation*. All teams strived for minimizing their own code base in favor of delegating a task to an external code library (such as the JavaScript library JQuery) or an existing web-based service (like YQL, a freely available data transformation service from Yahoo!). Delegation was used in several ways. Some usages included imposing needed yet currently missing technical functionality onto the API's (such as adding asynchronous functionality to a synchronous API). However, delegation was also used for core, value-adding features of the service, and thereby seeking responsibility over as little functionality as possible. One developer used the following illustrative comment to signify the nature of their service:

> *D1: Really smart! Super-user friendly! Very easy to code! No responsibility, adds a value! Damn, I like this!*

The final theme found was *immediate task accomplishment*. This pattern surfaced in teams especially when retrieving and processing data from the API's. The datasets were quite differently designed in regard to the configurability of the request and verbosity of the response. We found that discussions among teams strongly favored a less complex interface both in terms of request and response, even if such simplification would limit the functionality of the dataset. In following excerpt two developers are inquired about these issues in an on-site interview concerning different electronic payment API's:

> *D1: Me and [another team member] has worked a lot with [credit card payments] and ploughed through what's available.*
>
> *D2: [One credit card payment company] is totally different – they skip documentation and only shows examples*
>
> *I1: And that work fine?*
>
> *D2: Well, all API's work fine but when you're working with any other API you typically have to guess before you know what parameters you should pass in and then we're not talking about three or four parameters but maybe 30. […] And it's so annoying if all you should want to do is to enter a credit card number but you sit there and test…you end up entering your credit card number so many times.*

DISCUSSION

The purpose of this paper is twofold. First, as existing theorizing on unpaid third-party enrollment has overlooked much of the 'actual doing' of unpaid programmers, we sought to develop a deeper understanding of the developer practices of unpaid third-parties. Second, for organizations seeking to tap into these unpaid third-party developers, we wanted to investigate how the resources mediating such innovation can be crafted to support the work practices of these actors. In what follows, we seek to answer these questions.

According to the extant literature on commercial third-parties, the combination of favoring delegation over self-execution (even of core, value-adding features), favoring own preferences over known user needs and a lack of interest in capturing monetary value from the innovation would make the innovation a bad fit for a commercial service (Ceccagnoli et al., 2012). However, while the developers disregarded many of the core requirements for building a commercially viable service, they went to great lengths to ensure that the service should be novel. In this sense, unpaid developers offers a unique opportunity for adding innovative, complementary services which may offer significant user value but are commercially unsustainable due to ease of replication, merely target niche users or where there exist a general unwillingness to pay for the service for by users.

The current literature offers three broad areas for why developers work without being financially compensated: social norms, signaling and enjoyment (Boudreau & Jeppessen, 2011; Shah, 2006; Lakhani & Wolf, 2003). In our analysis we saw how an air of openness were pervasive among teams. E.g. two of the teams published their source code for anyone to download and modify (also including the API keys) on GitHub, a web site for publishing code openly. Moreover, all teams considered and one team pursued scraping data when an official API was not available. Hence, they perceived any data available on the web as "public goods" and did not consider the potential infringement they were committing by re-using such data. Given this social orientation towards openness, one could expect that developers might e.g. dismiss data sources whose license terms were too restrictive. However, the social norms connected to the open movement was not observed to affect the adoption of data source among the developers. One explanatory speculation would be that the license terms were simply ignored.

The second area found in existing theory for spurring motivation concerns signaling to fellow developers. In our material however, we were unable to find support of signaling having affected adoption of any data source, positively or negatively.

Finally the sheer enjoyment derived from developing software has been brought forward. When motivated by enjoyment there is a need for progress in the activity to derive amusement from the practice of programming (Sharp et al., 2008, Lakhani & Wolf, 2003). In our analysis, we found this stream of theorizing to refine our interpretation of why developers chose to

work with a certain API: The clear signs of emotions expressed when developers either failed or succeeded in their undertaking of tasks highlights such progress as important for API adoption (Alstrum, 2003). Further, more experienced developers seemed to inject non-value adding complexity to mundane problems, which is consistent with theorizing on intrinsic developer motivation (Lakhani & Wolf, 2003). Finally, developers frequently rejected data sources for "over-delivering" configuration opportunities given the task at hand. Instead they sought an API which they immediately could comprehend and use to solve their problem.

As more organizations seek leverage from unpaid third-party services, there is necessity to recognize the crucial integration point – the API. As we found in our analysis - a rewarding developer experience is contingent on a frictionless interaction between the development environment and the datasets. Based on the findings in this research we next present some implications for API design which organizations may draw upon in their efforts of attracting unpaid third-party developers.

IMPLICATIONS FOR API DESIGN

In our study we found three activities which needed to be successfully completed for developers to integrate a data source into the service, *getting access*, *assessment of functionality* and *working with datasets*.

The core of getting access typically consisted of two pieces of information which needed to be retrieved by developers, the API key and the base URL. Given that developers are unable to proceed to both aspects of assessment of functionality and any activity connected to working with data sets, it is thus crucial for organizations to cater for an as quick access procedure as possible. Since an API key typically requires some sort of developer authentication, our suggestions are select a path of requiring as little information as possible for issuing a key and clearly highlight how the retrieval of such information can be performed.

Secondly, assessment of functionality were an on-going process which often started before developers attempted to get access to the data source. Since the work practices of unpaid developers favors an on-going uninterrupted process, organizations should thus allow such assessments without forcing developers to e.g. complete the getting access activity. In other words, based on findings in this paper, unpaid developers would prefer to explore the functionality such as documentation and sample requests without any formal registration.

Finally, developers spent most of their time working with the datasets. Based on the analysis in this paper, it is paramount to support a process marked by continuous progress in the programming activities. Grounded in the practices found, we hence suggest that designers ensure that API's integrates painlessly into development environments and carefully verifies the compatibility with commonly used software libraries and frameworks. Further, we suggest designers to allow a minimum of configuration opportunities in the request and response of the API to support immediate task accomplishment. However, as a final note, API's are seldom developed from a clean slate but are tied to technological trajectories where the ideals presented in this paper may be unfeasible to implement. In such case, we suggest that API designers employ the pattern of delegation and clearly describe how developers can bypass the potential hurdles in the API. This way organizations lock into an established work practice in its user audience while at the same time are catering for a continuous developer experience.

CONCLUSIONS AND LIMITATIONS (200)

In this paper we set out to develop a better understanding of the practices of unpaid developers and grounded in the findings, develop implications for API designers. Through an ethnographic observation and analysis we arrived at three major interlinked activities and two recurrent patterns of action. The study has clear limitations. While a group contest offers apt opportunities for studying unpaid developers as well as constitutes an important setting for this type of development the specific circumstances of a contest may not apply on other unpaid development settings. Group dynamics (vs. sole development efforts), contest criterion and tightly bounded development effort length may have influenced the results.

While unpaid developers offer an enticing opportunity for organizations seeking to leverage innovative end-user services, recruiting and sustaining unpaid third-parties is a difficult task. Our hope with this research has been to add to the existing literature with insights into the development practices of unpaid third-parties as well aid organizations in their quest for a larger supply of innovative end-user services.

REFERENCES

1. Almstrum, V. L. 2003. "What is the attraction to computing?," *Communications of the ACM* (46:9), pp. 51-55.
2. Bergvall-Kareborn, B., Bjorn, M., and Chincholle, D. 2011. "Motivational profiles of toolkit users iPhone and Android developers," *International Journal of Technology Marketing* (6:1), pp. 36–56.
3. Boudreau, K. J., and Jeppesen, L. B. (2011). Unpaid Complementors and Platform Network Effects? Evidence from On-Line Multi-Player Games. *SSRN eLibrary*.
4. Ceccagnoli, M., Forman, C., Huang, P., and Wu, D. J. 2012. "Cocreation of Value in a Platform Ecosystem: The Case of Enterprise Software," *MIS Quarterly* (36:1), pp. 263–290.
5. Grudin, J. 1991. "Interactive systems: bridging the gaps between developers and users," *Computer* (24:4), pp. 59–69.
6. Hammersley, M. 2007. "Ethnography: principles in practice", London : Routledge.
7. Klein, H. K., and Myers, M. D. 1999. "A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems," *MIS Quarterly* (23:1), pp. 67–93.
8. Kuk, G., and Davies, T. 2011. "The Roles of Agency and Artifacts in Assembling Open Data Complementarities," In *ICIS 2011 Proceedings*
9. Lakhani, K., and Wolf, R. G. 2003. "Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects," *SSRN Electronic Journal*.
10. Osterwalder, A. 2004. "The business model ontology: A proposition in a design science approach," *Academic Dissertation, Universite de Lausanne*
11. Robinson, H., Segal, J., and Sharp, H. 2007. "Ethnographically-informed empirical studies of software practice," *Information and Software Technology* (2007:49), pp. 540–551.
12. Shah, S. K. 2006. "Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development," *Management Science* (52:7), pp. 1000–1014.
13. Schultze, U., and R. J. Jr Boland. (2000). "Knowledge management technology and the reproduction of knowledge work practices," *Journal of Strategic Information Systems* (2000:9).
14. Sharp, H., Baddoo, N., Beecham, S., Hall, T., and Robinson, H. 2009. "Models of motivation in software engineering," *Information and Software Technology* (51:1), pp. 219–233.
15. Strauss, A., and Corbin, J. 1990. "Basics of Qualitative Research: Grounded Theory Procedures and Techniques"., Newbury Park: Sage.