THE UNIVERSITY OF SYDNEY

HOW OPEN IS ‘OPEN API’? A CRITIQUE OF THE POLITICAL ECONOMY OF OPENNESS IN PROGRAMMING

by

Yuanbo Qiu

A thesis submitted in fulfilment of the requirements for the degree of

Master by Arts (Research)

Faculty of Arts and Social Sciences

The University of Sydney, Australia

August 2015

ABSTRACT

As a way to provide services or data to third-party developers, Open Application Programming Interfaces (Open APIs) have gained popularity among the programming community in recent years. Some advocates regard Open APIs as an alternative to open source as it facilitates collaboration between different software. Today many corporations such as Google, Facebook and Twitter are developing Open APIs for their existing services, and most of them are free of charge. As an API usually takes effort to be developed and exposes proprietary data to public, then why would for-profit companies be interested in investing in these kinds of projects?

Indeed, Open APIs can help external programs integrate with proprietary software. Many developers, however, have found this collaboration risky, and some of them have started to think about the openness of API compared with open source. More importantly, the definition of Open API is rendered ambiguous through the discursive practices that define 'openness' in contemporary digital culture. Drawing on a critical political economy perspective, this study begins with the discussion of theories of openness in programming. It then identifies the 'openness' in Open API when compared with other collaborative ways of software production, including open source. It concludes by discussing how corporations are using Open APIs as a marketing strategy to incorporate external developers into their business.

ACKNOWLEDGEMENTS

There are a number of individuals that helped me and made this dissertation possible. I cannot do justice to all of them, but I would like to thank a few in particular.

I thank my supervisor, Grant Bollmer, who guided me in these two years. Grant was an accommodating, patient supervisor throughout the process, and he has always provided critical feedback to my thesis. I would also like to thank Chris Chesher, for his support and helpful advices. I also appreciate the help from staff and colleagues of Faculty of Arts and Social Science.

Special thanks to the teachers of Writing Hub of the University of Sydney and Claire Reaburn, who helped me correct the grammar and format issues.

Last but certainly not least, I would like to thank my family, especially my parents for their years of support. Also, thank my friends who have been supportive of my research.

TABLE OF CONTENTS

List of Figures

Introduction: The Rise of Open APIs

Open Application Programming Interfaces, or ‘Open APIs’, provide services or data to third-party developers, so these developers can remix and remake the proprietary data owned by corporations such as Google, Facebook and Twitter into new applications and programs. For instance, the Google Maps API launched in 2005, perhaps one of the most well-known APIs today, has been used by map services such as Tripview and Foursquare in the development of new and innovative web platforms. As Google advertised on its website: ‘With the world’s most accurate global coverage and an active mapping community making daily updates, your users will benefit from a continually improving service’ (Google Maps APIs Overview, 2015). The Google Maps API provides geographic data and interactive maps so that programmers can embed map services on their own websites or in their applications. Twitter’s API is described as ‘a fast and easy way to grow your user base for mobile apps… let other users viewing those Tweets to download and launch your app across a number of mobile platforms’ (Twitter Developers, 2014). Twitter’s API helps programmers associate their apps with the Twitter, increasing the apps’ potential market.

Open APIs usually take effort to be developed and maintained, but most of them are free to use. Why do Internet companies like Google commit themselves to develop such freely accessible interfaces? What do they bring to the massive community of programmers and web users? What is the cultural significance of Open APIs? By exploring these questions from both social and technical perspectives, this dissertation argues that Open APIs can be designed for the benefit of the programming community, but profit-seeking vendors restrict the openness of Open APIs to reinforce the commodification of data in contemporary Internet culture. This process reframes the meaning of the word ‘open’ from its historical association with the Free and Open Source Software movements. Most popular Open APIs are vendor-dependant and thus facilitate the domination of Internet companies over the programming community, which could have a negative effect on the computing industry.

History of Open APIs

Open APIs have their origin in private APIs. A private API, or internal API, is an interface to connect the function of different applications within one specific system for internal use, in effect reducing the cost and redundancy of transactions to improve efficiency. Programmers used private APIs long before the popularity of Open APIs, as far back as when Microsoft was developing Windows 3.1x, which was released in 1992 (Microsoft, 2014). The Windows APIs provide access to the fundamental resources or processes such as file systems or devices managed throughout the Windows operating system. Many resources or procedures involved in Windows were standardised through APIs, so programmers could integrate the many dimensions of Windows in order to gain extra functionality for their programs in a more convenient and efficient way

(Microsoft, 2014). For example, for a user to copy a sentence from a TXT document to a Microsoft Word document requires the use of specific APIs to negotiate between different software. In this case, Word is retrieving this sentence from TXT via APIs instead of creating this sentence in Word itself. APIs function as an agent between these two pieces of software. In addition, APIs allow older applications to run on newer versions of Windows via ‘Compatibility Mode’, a process that emulates the functions of older versions of Windows through the implementation of APIs. When users are using an application, it does not mean that only one application is running at this time—many applications could be working together through the mediation of APIs. Like the gears that can combine different parts of a machine, APIs integrate different pieces of software to effectively build a new entity.

Open APIs depart from the history of private APIs by targeting external users, enabling interoperability and the sharing of data between different websites and online services (Bodle, 2011). They are named ‘open’ in opposition with ‘private’, as they are for public use rather than limited by internal intent. Nevertheless, the meaning of ‘open’ has been confused by developers and vendors in pursuit of profit and efficiency. Vendors package parts of their services and open them to the public through Open APIs, in ways that negotiate an apparent ‘openness’ of online information with proprietary control.

The parallel development of the semantic web[1], web services, Web 2.0[2] and the architecture of participation[3] all contribute to the cultural and technical infrastructure of sharing and distribution (Floyd, 2007), which together facilitate the blooming of Open APIs. Via the Internet, Open APIs can mix complex services provided by Internet giants such as Google and Twitter. As Open APIs are often used for fast integration of different pieces of web services (Bodle, 2011), they are also referred to Web APIs or public APIs. Today, Open APIs are so central to many web services that the term 'Open' has been dropped, with the term 'API' assumed to be, in some way, 'open'.

As should be evident, Open APIs are ubiquitous in today's software industry. For instance, when you get a high score in a game online or on your smartphone, you may find buttons like Google Plus' '+1' or Facebook's 'like'. When you click on this button, you share your score with your friends in Google Plus or Facebook without ever leaving the game. This simple act, ubiquitous online and essential for the Web 2.0 logic of 'sharing', depends on Facebook's and Google's use of simple Open APIs. Some

[1] A web-technology make data machine-readable, so that make it easy for the integration and reuse of data across various applications and platforms, see Berners-Lee, T., Hendler, J. and Lassila, O. (2001). "The Semantic Web". *Scientific American*, 284(5). pp 34-43.

[2] A design patterns and business modes that let owners provide their data in a form that allows remixing by others, see O'reilly, T. (2007). What is Web 2.0: Design patterns and business models for the next generation of software. *Communications & strategies*, (1), pp. 17-31.

[3] 'Architecture of participation' is a norm of cooperation, in which the service itself acts as an intelligent broker to connect other services, and harnessing the power of the users themselves. See O'reilly, T. (2007). What is Web 2.0: Design patterns and business models for the next generation of software. *Communications & strategies*, (1), p.22.

location-based services like Foursquare use Open APIs in a more interesting and complex way. They offer location-based data of user-to-social network applications such as Instagram to facilitate user interaction (Wilken, 2013). When maps are embedded into an app through APIs, as they are in the transport system software Tripgo, end users can plan their real-time trip when using public transport. Overall, by using Open APIs, third-party developers can find new ways to package their own content and provide new informational services to end users.

Literature Review

Open APIs and collaboration.

Open APIs may sound like a ‘free lunch’ for web services users, and they remind developers of the ‘open’ standard of Free or Open Source Software. Free and Open Source Software give developers the right to study, use, modify and redistribute the original software without paying, which facilitate collaboration between different softwares. Open APIs also advocate sharing data, but their mechanism are different from Free and Open Source software. Some researchers have been focusing on the differentiation between Open APIs and other collaborative tools. While Free and Open Source Software are non-proprietary and depend on nonmarket production strategies (Benkler, 2006), Open APIs are usually proprietary software and most of them are designed for marketing purposes. In terms of accessibility, Open APIs provide closed and packaged data to users, meaning third-party developers can only retrieve data without any change to the original code. In contrast, open source software provides full access to source code. Bucher (2013) points out that an important dimension of APIs is ‘function without disruption’. The utility of Open APIs is limited, in order to prevent full access to data and keep API under control (Bucher, 2013). He also claims: ‘Situated in between codes belonging to interoperable systems, APIs ensure that changes in the underlying code will not affect the code written to interact with the core system’ (Bucher, 2013). Thus, when cooperating with Open APIs, third-party developers may not enjoy the same freedom as they did with Free or Open Source Software—though this limitation may not hinder external programmers’ enthusiasm for using Open APIs.

The reasons why developers are increasingly interested in APIs have been examined by many studies. As Bucher explains, external developers can build great applications based on Open APIs, as they are standing on the shoulders of giants. In addition, users maintain a degree of independence, as the cost for coordination with Open APIs is low (De Souza & Redmiles, 2009). If users feel it difficult to cooperate with one Open API, they can just switch to another service. For example, Foursquare switched to OpenstreetMap after Google Maps API changed its terms of service and charged programmers (Fox, 2012). Apple developed their own maps service for iOS and cooperated with OpenstreetMap as a response to Google’s changing policy (Openstreetmap, 2012).

Floyd (2007) points out three aspects of Open APIs that facilitate innovation in informational technology, explaining why developers are fond of them:

1. They provide access to highly developed, robust technologies which only a large organization [sic] of expert programmers could create.
2. They provide access to massive amounts of content which no individual could gather on their own or afford to keep and maintain.
3. They lower the barriers to developing creative novel applications with powerful technologies. (p. 2)

For Floyd, developers neither need detailed knowledge of how a particular API works, nor do they need to be proficient in advanced programming skills. What external developers need is merely a creative idea to utilise these services (Chesbrough, 2003; Aitamurto, & Lewis, 2013). This seems to confirm Benkler's argument that human creativity becomes the core resource in the contemporary networked information economy (Benkler, 2006, p. 6). Benkler claims that human creativity and the economics of information structure the new model of economy, which facilitates commons-based peer production and collaborative efforts. In this sense, Open APIs seem like a new paradise for collaboration in addition to Free or Open Source Software among the programming community.

There are significant works about how to design and use APIs (Xie & Pei, 2006; De Souza & Redmiles, 2009; Sullivan, 2010; Jones, 2012; Michel, 2013; Gray, 2015), or how APIs minimise the knowledge required for the construction of software and the facilitation of cooperation (Floyd, 2007; Kansa & Wilde, 2008; Aitamurto & Lewis, 2013; Duvander, 2014). However, there are two key points these works do not cover.

First, although these works do mention the features of Open APIs, few of them go into technical detail about what this ‘open’ means in terms of programming. Without fully examining the technical process of adopting Open APIs, they might misunderstand the point about the meaning of ‘open’ and how economics affect the ‘openness’ of Open APIs. Second, most previous work from practitioners and researchers on Open APIs merely examines how this new technology facilitates the coordination of software. They stress these positive uses of Open APIs while neglecting issues of ownership, control and profit. Some studies point out this taken-for-granted economic dimension should be carefully scrutinised, as it could lead to a potential disadvantage for external software developers whose products depend on Open APIs (Aitamurto & Lewis, 2013; Bodle, 2011), but elaboration on how these power structures are enforced via Open APIs could be done through specific case studies. Drawing from these two key points, this dissertation first explores the meaning of ‘open’ in ‘Open APIs’. It discusses how practitioners conceive of this ‘open’ historically and presently, and then examines the use of this definition of ‘open’ to achieve an economic and political goal of top-down control rather than collective collaboration.

Power of code and Open APIs.

According to Berry (2011), APIs belong to the ‘functions/methods’ category of programming. They are blocks of code that ‘can be used and reused and are usually written in a general fashion to be applicable to standard data types’ (p. 56). APIs, specifically, are a library of handy functions or methods used by programmers and standardised across different platforms (Berry, 2011, p. 56). The underlying technologies and assumptions that guide human collaboration is essential. As Galloway notes, ‘Code is a set of procedures… designed in particular ways to achieve particular ends in particular contexts’ (2004, p. xii). Moreover, as Bucher (2014) notes of the Twitter API, APIs are not only intermediaries between providers and users, but also tools that help providers to control, manage, or take advantage of third-party developers:

> The history and practice of working with the Twitter APIs suggests that APIs are mediatory objects that have enactive powers, structuring both the Twitter platform and its users. By providing an API, the company is able to harness the capacity of the field, of letting third-party developers come up with ideas that they would not have been able to. (Bucher, 2013)

In this sense, when analysing the integration through APIs, the intention and affordance come from APIs providers should be clearly identified. The collaboration is not only caused by technology themselves, but also affected by providers as they decide how Open APIs are developed and being used. The power of API providers is the key point when examining the implementation of power through APIs. As will be elaborated, the purpose of most vendors in distributing Open APIs is to seek profit and build a new infrastructure that depends on them. These purposes influence the technical implementation of Open APIs distributed by commercial organisations, frames these Open APIs as (in spite of their name) Closed Source APIs, and reinforces various forms of control.

Power from legal contract.

These forms of control have two dimensions. First, Open APIs give providers an almost unlimited power through accessibility and the setting of legal contracts such as Terms of Use or User Agreement. Providers can change their rules and limit access whenever they sense a threat from third-party developers. Third-party users are thus rendered passive, as they are not involved in the negotiation of the rules that govern them. Sometimes users' applications or websites are suspended when vendors of APIs limit access. Sometimes they are involved in lawsuits because they offend the constantly changing Terms of Use or User Agreements. Changes in the 2011 Twitter API is a typical example of how this works in practice. Although third-party programmers were obeying the terms of Twitter APIs, from 2010 Twitter enforced increased restrictions and contractual limitations on accessibility. These restrictions caused financial loss for some third-party users and made them constantly worry about offending Twitter's governing principles (Bucher, 2013). Moreover, vendors of Open APIs restrict redistribution of their services, prohibiting peer production and propagating a vendor-dependent infrastructure, which directly contradicts most ideals about 'openness' held throughout the history of the Internet. These arguments will be elaborated in Chapter Five.

Power of Pre-defined Code.

Second, as APIs standardise forms of interaction between applications, their functions are similar to what Alexander Galloway (2004) terms 'protocol', which is a set of communication rules employed for computers to communicate with each other. Galloway divides the history of technology and control into three eras: the classical era, the modern era and the postmodern era. The characteristics of each era are as follows:

Era	Machine	Period	Diagram	Manager
Classical era	Simple mechanical machines	Sovereign societies	Centralisation	Hierarchy
Modern era	Thermodynamic machines	Disciplinary societies	Decentralisation	Bureaucracy

Postmodern era	Cybernetic machines and computer	Control societies	Distributed network	Protocol

Figure 1.1: Galloway’s Historical Periodisation of Technology and Control

Galloway outlines what kind of ‘diagram’ and what kind of force is taking charge in each era. ‘Diagram’, as Galloway defines it, is cartography of organisational designs that coexist with the social field (Galloway, 2004, p. 196). From centralisation to decentralisation and the distributed network, each diagram represents a mode of control maintained by a different techno-cultural assemblage. The distributed network in the postmodern era is managed by protocol, and it extends deeply into every aspect of social field (p. 11). Galloway argues that protocol is not merely a technology that regulates data exchange on a computational level, but also a management style that governs social relations. Protocol carries politics which are similar to bureaucracy and hierarchy, though in a way that is distributed throughout networked machines and perpetuated through the management of information. Protocol seems to shape reality automatically, without direct human intent, making distributed networks more controllable and more regulated than the previous forms of hierarchical and bureaucratic control. From this perspective, Open APIs, which define a way to connect users’ applications and vendors’ applications, also contain political and governmental ‘powers’ that affect social relationships.

Following Galloway, Chun (2011) develops Lessig's (1999) argument that 'Code is law' to emphasise the power of code on the governance of conduct. Chun claims that code 'automatically brings together disciplinary and sovereign power through the production of self-enforcing rules' for governing (Chun, 2011, p. 27). Chun claims that users' actions are limited by code, and as code is developed by its programmers, programmers have the ability to control users as a form of governance. This governance is conducted through self-regulation as users' abilities are limited by predefined code. Since APIs are pre-defined by vendors, the programming processes with APIs have to abide by the protocol that specific Open APIs are using, and external developers have to follow the instructions of specific APIs and alter their own software to work with Open APIs. In short, external programmers might lose their independence and be controlled by forces that come from the inflexible Open APIs developed by vendors.

Indeed, independent power still exists within code itself, in addition to legal contracts like Terms of Service or End-User Agreements. Open APIs can achieve results that follow the purpose of designers or exceed their expectations. This subtle power of code is what concerns critical code scholarship such as the work of Galloway and Chun. Thus, examining the transition of these Open APIs and interpreting the power function from a technical perspective is another way to study the mediation of Open APIs on the interaction between providers and users. Chapter Five will also focus on the force embedded in APIs.

In sum, this dissertation is firstly going to answer: What is the meaning of ‘open’ in Open API? Why do vendors develop Open APIs and open their private data to the public? Discourse analysis will be deployed to answer these questions. This thesis then asks: What politics does this term ‘open’ carry for programmers? What kind of unequal relationship is formed between vendors and users of Open APIs? How are different kinds of power enforced through API usage? To answer these questions this dissertation adopts a perspective derived from Vincent Mosco’s theory of the political economy of communication.

Methodology

A political economy approach to communication.

This dissertation relies heavily on a version of political economy advanced by Vincent Mosco (2009). Political economy is usually deployed to analyse distribution, exchange, power and control in the processes of production and consumption. It can be applied to both physical production and non-physical production, such as that of ‘immaterial’ information in the process of communication. According to Mosco, ‘Political economy is the study of the social relations, particularly the power relations that mutually constitute the production, distribution and consumption of resources, including communication resources’ (Mosco, 2009, p. 2). Based on ‘uncover[ing] connections between ownership, corporate structure, finance capital, and market structures to show how economics affects technologies, politics, cultures, and information' (Meehan, Mosco, and Wasko, 1993, p. 347), the political economy of communication examines how different communication sectors interact with each other along with the power embedded in media within a broader social context. Specifically, Mosco introduces three entry points for the study of political economy of communication: commodofication, spatialisation and structuation. Commodification is the process of transforming use values into exchange values (Mosco, 2008, p. 129). Commodification happens when goods and services, which are valued for their utility, are turned into commodities for exchange in market. Spatialisation is the process of overcoming the constraints of space and time in social life. It is significant for the study of political economy because communication is one of the primary means of bringing about spatialisation throughout society (Mosco, 2008, p. 157). Structuration describes 'a process by which structures are constituted out of human agency, even as they provide the very "medium" of that constitution' (Mosco, 2008, p. 185). Mosco examines structuration of communications in terms of class, gender, race, social movements and hegemony to reveal how power works to crystallise what is popularly called common sense into hegemony. Structuration, as Mosco proposed, is a means of describing the social relations of communication practices (Mosco, 2008, p. 185).

Examining the role of communication technology is another dimension of understanding the political economy of communication. Some scholars might overemphasise the role of communication technology in shaping social relationships. Benkler (2006) proposes an approach to rethink the political economy of new media through the lens of peer-production and sharing. He argues that the economic model of the 19th and 20th centuries emphasises marketing and capital–intensive production. This model results in an increasingly centralised control mode where the capitalist dominates communication and the circulation of information. However, the emergence of computers, networks and increasingly affordable media production outlets has transformed the economy from an Industrial Information Economy to a Networked Information Economy, characterised by non-proprietary forms of production and increasingly decentralised individual actions. In the Networked Information Economy, 'human capacity becomes the primary scarce resource' because of the lower constraints of communication and lower transaction costs (Benkler, 2006, p. 6).

Benkler believes contemporary innovations in informational modes of production mostly come from actors who exist beyond the limits of the capitalist market: 'Advanced economies rely on nonmarket organizations [sic] for information production much more than they do in other sectors' (Benkler, 2006, p. 48). He thereby believes these innovations, which are free of the centralised control of capitalism, reveal the democratic potential of new media.

However, when these 'innovations' that exist 'outside' of the capitalist market are placed into a broader social context, we can see they still have close ties to marketing and other capitalist structures of power. Individuals are not automatically empowered via technological development, and social media firms can take advantage of the increasing 'creative' capability of individuals. Mosco (2009) claims that new media users are being exploited by corporate organisations. Compared with traditional media, new media 'deepen and extend tendencies within earlier forms of capitalism by opening new possibilities to turn media and audiences into saleable commodities' (Mosco, 2009, p. 120). Similarly, drawing on the 'peer production' central for social media, Bauwens identifies how the creative empowerment of users still takes place under the conditions of value extraction that benefit major capitalist organisations:

> Because of the hyperproductive nature of peer production, which allows for broader participation and input, passionate engagement, and universal distribution of its benefits (conditioned by network access), it attracts the participation and engagement of capital through the activities of netarchical capitalists. Netarchical capital is that sector of capital that understands the hyperproductive nature of peer production and therefore enables and empowers social production to occur, but it is conditioned by the possibility of value extraction to the benefit of the holders of capital. (Bauwens, 2012)

Thus, a critical study of the politics of Open APIs should take into account the ownership models implicit in contemporary digital culture, and what kind of value is generated through 'open' software, rather than merely examining how 'openness' is assumed to technically facilitate coordination between peers.

Some theorists have tried to combine different methods to deal with complex economic issues raised by these technological developments. Van Dijck deploys Actor-Network-Theory (ANT) as a complementary method to the political economic study of social media. ANT treats both human subject and non-human elements as equal actors in the construction of 'society', discovering the dynamic interaction between these actors. According to van Dijck, the adoption of ANT is a response to the ignorance of technologies in the mainstream political economic study of media. In particular, van Dijck criticises Manuel Castells. As she contends, Castells mainly focuses on the institutional power, interactions and forms of resistance between individuals. However, Castells' approach overlooks the technologies themselves, especially in their everyday use (van Dijck, 2013, p. 27). Castells' political economy only examines the 'larger power relationship vital to their development' while ignoring technological affordances[4]: 'Political economists, for their part, focus on socioeconomic power struggles between owners and users, but technologies and content are likely to escape their radar' (van Dijck, 2013, p. 27).

Van Dijck thus proposes the use of ANT to emphasize the importance of

[4] The possibilities that technological changes afford for social relations and social structure. See Wellman, B. (2001). Physical place and cyberplace: The rise of personalized networking. *International journal of urban and regional research*, 25(2), pp. 227-252.

technology in relation to political economy. ANT concentrates on coevolving networks of people and technologies, while a political economic approach focuses on the power which promotes such a evolution. These two complementary approaches can help researchers get an overall perspective of a changing ecosystem of connective media. Drawing from ANT and political economy, van Dijck examines popular social platforms and argues that large companies are ‘making sociality technical’ rather than ‘making the web social’ (van Dijck, 2013, p. 12). Social activities of users have been converted into algorithmic concepts by coding technologies. As a result, global corporations are exploiting user connectivity and transmitting ideological principle in a subtle way.

The combination of ANT and political economy considers both social and technical features of digital media. But ANT is not a technologically determinist method—it also stresses the impact of human agents on altering the ways that digital media is used, particularly when these technologies are developed and designed by certain groups for certain purposes. All technologies may have positive potential to advance social and technical welfare, but how they are utilised, and to what ends they are being used, can highly affect the final social impact. Fuchs examines social media revolutions and claims technology itself cannot cause revolution: ‘Only people who live under certain social conditions and organize [sic] collectively can make rebellions and revolutions’ (Fuchs, 2013, p. 102). He identifies the relationship between power and technologies and beliefs:

> Societal phenomena merely express themselves in communicative and technological spaces; technologies do not cause them. Technological determinism inscribes power into technology; it reduces power to a technologically manageable phenomenon and thereby neglects the interaction of technology and society. The Internet is not like the mainstream mass media an ideological actor, but rather an object of ideological signification in moral panics and moral euphoria. (Fuchs, 2013, p. 137)

Thus, the human actors who can alter the usage of these technologies are also the central issues in addition to their technical features.

When applying the political economy of communication to digital media, one cannot see technology itself as the only determining factor. Rather, political economy looks towards the broader structures of power owned by different human agents, and how this power affects the usage of technology. It does not mean that we can ignore the technical affordances of digital technology, but that we have to emphasise the underlying social system within which a technology is situated. As Birkinbine (2014) says, technology has changed, but the underlying class distinctions and social antagonisms have not changed (p. 39). ‘In other words, no matter how purportedly revolutionary or novel the technology, the technology exists within a capitalist system that has certain well-documented tendencies that cause it to remain relatively constant’

(Birkinbine, 2014, p. 39). Drawing from this methodology, this study first examines the particular technical attributes of Open APIs, then looks at how capital takes advantage of these technical attributes.

Ideological criticism and critical discourse study.

Ideology is another dimension that has to be addressed with the political economy of communication. Benkler believes technology is neither deterministic nor wholly malleable: 'It can make some actions, relationships, organizations [sic], and institutions easier to pursue, and others harder' (Benkler, 2006, p. 17). He points out what matters in the usage of technology are the values and motivations for human behaviour. Likewise, Graham (2007) reviews the history of the political economy of communication and criticises contemporary studies. He believes that new media 'inevitably lead to new political economic formations', so scholars need new understandings of political economy to correspond these changes. Graham advocates a new political economy of communication by combining four elements: Smythe's 'consciousness'[5], Marx's 'value'[6], Silverstone's 'mediation'[7], and Lemke's 'meaning'.

[5] The total awareness of life, includes people's understanding of themselves as individuals and their relationships with other individuals and natural environment. See Smythe, D. W. (1981). *Dependency road: Communications, capitalism, consciousness, and Canada*. New York, NY: Ablex Publishing Corporation. P. 81

[6] The ways in which Marx evaluate various aspects of our world from a political economic perspective, See Marx, K. (1867). *Capital. Volume I.* London, UK: Penguin

[7] See Silverstone, R. (1999). *Why study the media?* London, UK: Sage. P. 13

> The new political economies of communication require entirely new understandings that can comprehend the ways in which consciousness is produced; ways in which values are produced; the means by which meanings are moved; and the ways in which these aspects are realised in specific meanings. (Graham, 2007, p. 26)

The synthesis of these four aspects can help shape political economy for the complex communication environment today. Both Benkler and Graham identify purpose and value as significant aspects of human activity and should be considered with political economic study. As discursive practices can shape meaning making and affect human consciousness, ideological criticism on discourse can be complementary to political economic study. Graham and Luke (2011) propose critical discourse analysis and political economy to understand the new corporate order. They define discourse as ‘institutionally and culturally structured patterns of meaning making’ and deploy critical discourse analysis as the study of discourse with particular reference to inequalities of power and formulation of ideology (Graham & Luke, 2011).

Fuchs (2013) outlines a comprehensive way to study social media through ideology critique. He contends: ‘ideologies are practices and modes of thought that present aspects of human existence that are historical and changeable as eternal and unchangeable […] ideologies claim that things cannot be changed, have always been or need to be the way they are now’ (Fuchs, 2013, p. 17). Following Marx, Fuchs believes that everything in society is created by humans in social relationships and can be changed—everything has alternatives (p. 17). Ideology naturalises something as common sense, creates ‘false consciousness’, and makes us believe some claim about reality is undoubtable. Thus, ideology critique is essential to ‘denaturalise’ what is accepted as ‘natural’ relations of existence. Ideology critique is a complementary necessity for any political economy of digital media, as ideology is usually a required component for the maintenance of domination and exploitation. Political economy focuses on the structural features of capitalism, while ideology critique identifies claims about reality and exposes them as false or contingent (Fuchs, 2013, p. 97). This dissertation thus employs critical discourse analysis to discuss the ideology of ‘open’ when employed by Open APIs.

Ideologies are usually naturalised by the use of discourse for specific goals by specific organisations. This discourse claims something is natural and unchangeable. Critical Discourse Analysis (CDA) is one way to reveal ideology, specifically identifying the structural relations of power legitimated by language. Fairclough is a theorist who believes ideologies are significant for various social relations including power differences and inequalities (Fairclough, 2013, p. 26), and he proposes CDA for ideological critique. Ideology is firstly a relation between texts and power, and it is also a relation ‘between orders of discourse and power and between languages and power’. Meanings of texts, in which power is embedded, can in turn affect social structure and power structure and ‘can achieve relative stability and durability in social practices and social structures’ (Fairclough, 2013, pp. 28-29). As Fairclough (2013) contends, contemporary capitalism has the power to ‘prevents or limits, as well as in certain respects facilitating, human wellbeing and flourishing’ (Fairclough, 2013, p. 2), and discourse should account for this power as it ‘brings into the complex relations which constitute social life: meaning, and making meaning’ (Fairclough, 2013, p. 3).

> Specifically, to deploy critical discourse analysis is to ‘explore often opaque relationships of causality and determination between (a) discursive practices, events and texts, and (b) wider social and cultural structure, relations and process; to investigate how such practices, events and text arise out of and are ideologically shaped by relations of power and struggles over power; and to explore how the opacity of these relationships between discourse and society is itself a factor securing power and hegemony. (Fairclough, 2013, p.93)

In this study, the ideology of the seductive word ‘open’ is examined. This dissertation argues how the term ‘open’ conveys an ideology that expresses a strong belief in the power of technology itself, as if Open APIs are effectively independent of society and can automatically facilitate free or open standards among programmers. By denaturalising the taken-for-granted assumption that Open APIs are, in fact, ‘open’, this study puts Open APIs into a broader social context to explore the ideological manipulation inherent in this ‘openness’.

Overview of Dissertation

In sum, how vendors make sense of the ‘open’ of Open APIs, how third-party developers use and understand Open APIs, and what is the implication of this integration in programming are the major questions of this study. Specifically, this study focuses on documentary evidence of corporate operations, FLOSS (Free/Libre and Open Source Software) community and developers’ forums. The structure is as follows: Chapter Two identifies the technical features of Open API and discusses the meaning of the ‘open’. This ‘open’ is compared with different historical meanings given to ‘openness’ in programming. Then it revisits the discussion of the power of code to identify the pre-defined technical features of Open APIs, and concludes with the argument that for-profit corporations alter the technical feature of Open APIs to enforce control, which make the ‘open’ of Open APIs far from the open standards of open source. Chapter Three reviews previous literature about participation in digital technology and identifies new principles for the examination of exploitation. It points out that a new form of surplus value is generated in digital technology use. It also identifies the power that maintains this exploitation as another principle to examine the politics of Open APIs. Drawing from specific examples, Chapter Four explains how the production of surplus value happens in the use of Open APIs from the perspective of political economy. Chapter Five then focuses on how the power of vendors is maintaining this exploitation. Throughout, this dissertation identifies the ideological manipulation implemented by vendors of Open APIs. Vendors are promoting a closed system and use ‘open’ as a marketing strategy to compel developers to join this system, which will be detrimental to the development of software.

Chapter 2: The Openness of Open APIs

This chapter focuses on the specific meaning of ‘open’ in Open APIs. It outlines how different groups understand ‘openness’ and clarifies whether this ‘open’ is aligned with the historical meaning of ‘open’ in the field of programming. As some of the language used by programmers tends to be obscure or ambiguous, this chapter begins by contextualising ‘open’ and associated terms by providing relevant background knowledge essential for understanding debates about the openness of software. Then it gives an introduction to the ‘open’ movement and the rise of Free and Open Source Software (FLOSS) community, and examines their use of openness as a positive value. Finally, this chapter identifies the technical and social characteristics of Open APIs to see how they differ from this historical understanding of ‘open’.

Source Code, Object Code and Software

Source Code is a key element in computing. Berry (2011) identifies two major kinds of code in programming: delegated code and prescriptive code. For Berry, delegated code is what people usually call source code: human-readable text files that are used for editing and programming software (Berry, 2011, p. 52). Source code cannot usually be run and understood by a machine. Instead, it needs to be translated—or compiled—into executable, machine-readable prescriptive code. At a machine level, prescriptive code is usually binary, ‘represented digitally as a stream of 0s and 1s and is very difficult for humans to write or read directly’ (p.53). This prescriptive code is machine-read language that aims to give direct orders to computers.

Prescriptive code is generally referred to as ‘object code’ by programmers; it is usually packaged and sold as product without the underlying source code (Berry, 2011, p. 53). Users cannot edit a product like Microsoft Word, for instance. It is this final product that we refer to as ‘software’. In effect, prescriptive code, object code and software are different names of the same thing (Berry, 2011, p. 32, p. 65). However, source code is different: It is the static textual form of software which consists of human-readable instructions, while software is the processual operating form (Berry, 2011, p.32). When we are using Microsoft Word, it is in the form of object code, while source code is controlled and maintained by the Microsoft Corporation. For skilful programmers, access to source code means they can study and modify software. Moreover, programmers can redistribute new software based on the original source code. As exposing source code might result in infringement of intellectual property, proprietary software vendors usually keep the source code secret. In this regard, whether the source code of a software package is available for external developers is key to a sense of ‘openness’.

The Power of the Human in the transition from Source Code to Object Code

Vaidhyanathan (2001) also identifies the relationship between these two basic forms of code in programming. As he said, source code is a set of instructions that human beings write in order to give commands to programmable computers. Source code can be translated into object code by a 'complier', then object code can give direct orders to the computer (Vaidhyanathan, 2001, p. 154). What should be clarified here is there is a clear line between these two codes. Source code and object code are related but not identical, and the process of 'translation' from source code to object code does not happen automatically. Chun (2011) suggests that this difference has resulted in the positioning of 'source code as fetish':

> To state the obvious, one cannot run source code: it must be compiled or interpreted. This compilation or interpretation—this making executable of code—is not a trivial action; the compilation of code is not the same as translating a decimal number into a binary one. Rather, it involves instruction explosion and the translation of symbolic into real addresses (Chun, 2011, p. 23).

There is a gap between source code and execution. Source code can be rendered into machine-readable commands and be executed, but there is a process pre-defined by human programmers. As Chun (2011, p.53) argues, source code itself is not executable without the intercession of other 'layers', and computers cannot think but merely follow human instructions (Chun, 2011, p. 53). Code has been made to be executable by human, thus the format of source code and its connection with executable object code are controlled by the code creators. Lessig (2006) argues that 'code is law'—code in cyberspace regulates software and hardware like laws do in real space, thus code creators are increasingly lawmakers and are empowered: 'They determine what the defaults of the Internet will be; whether privacy will be protected; the degree to which anonymity will be allowed; the extent to which access will be guaranteed. They are the ones who set its nature' (Lessig, 2006, p. 79). A key point Lessig mentions is: laws penalise people after they violate a rule, but code enforces its control directly; people just cannot offend this new 'law' in the beginning (p. 110). Code changes following the choices and values of the code creators require users of code to adhere to those same values. Chun further argues that code is not law but rather a lawyer's dream of what law should be: automatically enabling and disabling certain actions, functioning at the level of everyday practice (Chun, 2011, p. 27). Code is not inherently neutral and open, it is created and controlled by initial distributors.

In sum, the power of vendors of specific code comes from two dimensions. In addition to the accessibility of source code, another force comes from the design of code. That is, code is executable because it embodies the power of the executive (Chun, 2011, p. 27). Vendors define how source code can be executable, thus enabling and disabling certain usage. This power is given and controlled by designers or owners of specific code. Thus, even if the source code of software is accessible and free to be used by external developers, it does not mean the software is free of corporate power, because source code is already in its pre-defined form to achieve specific goal.

Programmers who wish to modify source code have to obey the pre-defined order of vendors so that they make their program executable. This pre-defined feature of software carries political economic issues as well.

Software without source code cannot be modified or redistributed—this is the first level of control over software employed to manage its political economy. The availability of source code (or lack thereof) and the legality of copyright both ensure this control. In the second and more subtle level, the disciplinary power of code developers is embedded in the source code of software and are ruling the programming process. The following section will introduce the open movement which began in the middle of last century to elaborate the relationship between code and openness, along with the politics of resisting the corporate control of software and programming.

The Open Movement and Disclosure of Software

In the 1960s and 1970s, 'only computer programmers used computers widely' (Vaidhyanathan, 2001, p. 154), and companies often released source code along with their software in the spirit of openness and collaboration among programmers. When the software industry bloomed in the 1980s, companies became aware of the importance of keeping code secret to earn profit (p. 154). UNIX was a popular operating system developed by MIT in the 1980s, and Richard Stallman was one of the developers of UNIX. However, AT&T, as the vendors of UNIX, decided to keep the source code secret and make it commercial in 1982, which made it difficult for programmers to learn and update. Stallman realised the threat to freedom and creativity raised by increasing corporate control, and left MIT to establish the Free Software Foundation (FSF) in 1984. With the FSF, he established the GNU operating system and urged programmers to distribute what they had produced under free software licenses on his new system. Stallman, as the head of the FSF, started promoting software freedom and fighting against the proprietary software since then. This ‘Free’ means freedom to ‘run it, to study and change it, and to redistribute copies with or without changes’ (Stallman, 2014). As it is often described, the ‘free’ of Free Software is not the free of ‘free beer’, as it is not about money but about liberty (Vaidhyanathan, 2001, p. 155).

To avoid free software being used by corporate power, Stallman proposed ‘Copyleft’ licenses. ‘Copyleft licenses require that anyone who copies or alters Free Software agrees to release publicly all changes and improvements. These changes retain the Copyleft license. Thus the license perpetuates itself. It spreads the principle of openness and sharing wherever someone chooses to use it’ (Vaidhyanathan, 2001, p. 155). In short, Copyleft refers to licenses that ‘allow derivative works but require them to use the same license as the original work’ (GNU Operating System, 2014a). The GNU General Public License (GPL) is one typical copyleft license of free software.

In 1998, Eric Raymond and a group of hackers founded Open Source Initiative (OSI). Differentiated from the FSF, OSI focused on the practical value and economic perspective of software. They created the open source operating system ‘Linux’ to promote their works. Unlike Free Software, Open Source Software does not confront the issue of intellectual property directly. Some advocates of Open Source Software believe that Open Source can appeal to corporate entities, as it highlights the practical perspective and admits the commercial value of software. Other supporters of Open Source simply rejected Free Software’s ethical and social values as ideological and radical (Stallman, 2014). For the Open Source movement, the question of whether derivative software should be open source is a practical question, not an ethical or political one. OSI regards non-free software as a suboptimal solution. However, for the FSF, non-free software is a social problem and free software is the solution (Stallman, 2014).

Distinction of Free Software and Open Source

From a practical perspective, all the licenses of open source software and free software allow the source code of the original program to be freely distributed. The main difference is their relationship with proprietary software—with or without restriction on derivative works. Free software emphasises the redistribution of the code, completely opposing any commercial usage of software (West, 2003) which could affect the ownership of derivative works. Programmers of free software have to give up any proprietary control of their works and require any derivative works to follow the principles of free software, too. For instance, if software is released under GNU

General Public License (GPL), all users should share any changes when using it, and any derivative works should also follow the GNU GPL. In this way ‘you can avoid the risk of having to compete with a proprietary modified version of your own work’ by using GNU GPL (GNU Operating System, 2014b), as the FSF suggests.

While the Free Software Foundation recommends such copyleft licenses, and usually use GNU GPL (GNU Operating System, 2015), the Open Source Initiative is more liberal as it maintains a list of approved licenses, which includes GNU GPL, though is not limited to it. Software under any license approved by the Open Source Initiative can pass the review process and label themselves open source (OSI, 2014a). Some licenses do not require the distributed works to open their source code. For example, the MIT License and BSD (Berkeley Software Distribution) License enable programmers to develop proprietary software based on original software. These licenses allow developers to use open code to create a commodity. Thus, Open Source Software are more attractive for external users as users maintain a degree of ownership over their software and can monetise them.

The Openness of FLOSS and its Alteration

Although the Free Software and Open Source Software communities differ in their values, they share the same belief: software should be free for developers to study, modify and distribute. That is why they can be combined together and coined FLOSS

(Birkinbine, 2014, p. 13). According to OSI, to qualify as ‘open’, software needs to use at least one of the approved licenses (OSI, 2014a) and go through a public review process by OSI (OSI, 2014b). All the licenses emphasise the right of redistribution in addition to merely making source code inspectable. The implication of ‘open’ in open source software has been misused by commercial organisations, and OSI has pointed out this phenomenon:

> The term ‘open’ applied to software source code was sometimes used to imply source code being merely inspectable or visible or available, as in the phrases ‘open computing’ and ‘open systems’ that were adopted by proprietary Unix companies’ marketing efforts in the 1980s. OSI’s term ‘open source’, as detailed in the Open Source Definition, has since clarified that open source specifically entails not mere inspection access but also conveying to recipients the perpetual right to fork (develop new software with the source code of original software) covered code and use it without additional fees (OSI, 2014c).

Stallman (2014) also identifies that some companies have taken advantage of the term ‘open’. He points out that IBM and Cygnus were advertising their proprietary programs as ‘Open Source’. The description of their software ‘didn’t actually say that these are “open source software”, they just made use of the term to give careless readers that impression’ (Stallman, 2014). Stallman subsequently pointed out these corporations’ trick: ‘These companies actively invite the public to lump all their activities together; they want us to regard their non-Free Software as favourably as we would regard a real contribution, although it is not one. They present themselves as “open source companies,” hoping that we will get a warm fuzzy feeling about them, and that we will be fuzzy-minded in applying it.’ Stallman also identifies why companies use the term ‘open’ in preference to ‘free’: perhaps the term ‘free’ is loaded with idealism while ‘open’ is more suitable for developers to accept (Stallman, 2014). ‘Open’, to some extent, is deployed by commercial organisations to mislead developers.

What needs to be clarified here is the definition of ‘open’ of FLOSS entails not only inspection or usage, but also the right of modification and distribution without fees (OSI, 2014d). In response to the commercialised misuse of the word ‘open’, programming communities such as Creative Commons gave a clear definition of ‘open’ through principles that define openness in relation to data and content (Creative Commons, 2014; Open Definition, 2014). This definition is according to the principles of OSI and FSF. From the documents that define ‘open’ we can conclude that both free software and open source software meet the same requirements, which are the essence of FLOSS:

1. Both source code and object code should be open to public (OSI, 2014d).
2. There should be no restrictions on use that differentiate between users.
3. Anyone is able to legally modify and redistribute open software.

4. Any implementation of this software should be free of charge.

Software that lacks any one of these conditions is partly closed and cannot be labelled ‘open’ according to FLOSS criteria.

From The Open Software Movement to Open API: Or, The ‘Open’ in Open API

Exploring ‘openness’ with two sets of questions.

Tkacz (2014) traces the proliferation of the word ‘openness’ and considers how it functions in contemporary culture. As he claims, the ‘spirit of open’ is closely articulated with collaboration and participation with reference to the debates about open systems and the emergence of the FLOSS community (Tkacz, 2014, pp. 7-8). ‘Translated from the world of software (but not reducible to it), openness must therefore be understood as a powerful new form of political desire in network cultures’ (Tkacz, 2014, p. 28). And network cultures, as Tkacz states, refer to ‘groups who organize [sic], communicate, and produce primarily through the web and other networked technologies’ (Tkacz, 2014, p. 28). As he identifies, today openness passes through technocultures, undergoes transformations, flows into every aspect of contemporary culture and triggers politics. For example, discourses of ‘open’ in open access, the Open Everything movement, and the ‘openness’ of organisations all carry different meanings. ‘Open’ in open access describes the making available of published content such as scholarly, educational or scientific materials. This ‘open’ refers to making knowledge accessible and visible. The strongest expression of a new form of openness comes from the Open Everything movement from 2010. The ‘open’ of the Open Everything movement covers different dimensions of openness but mainly focuses on a new mode of collaboration, which gathers people to use openness to create and improve software, education, media, architecture, etc. (Tkacz, 2014, pp. 26-28). The meaning of open also varies in different organisations. For example, Tkacz identifies the openness of Wikipedia as allowing for creation and modification under commons-based licenses. But sometimes openness is fraught with ambiguity, when used, for instance, by companies like Google for ‘an innovative and competitive production method’ (Tkacz, 2014, p. 29).

Thus, it is important to differentiate these forms of ‘openness’ and ask two questions: What is actually open? To what extent is it open? In regards to programming, we have to ask specific questions in relation to its technical features. This dissertation proposes two set of key questions when examining the openness of one software. The first set of questions is: With a specific piece of software, what is actually open? Source code? Or object code? Or both? The second set of questions are concerning the right of external developers: Open to whom? To what degree it is open—is it only visible for watching and learning, or can users use and modify it? Can users actually redistribute it and gain profit? Clarifying these questions is key to identifying the ‘openness’ in programming.

Two levels of interaction in Open APIs.

After identifying the criteria of the openness, we can examine the openness of Open API from a technical perspective in this section. One API alone does not compose an entire software platform. De Souza and Redmiles (2009) point out the importance of information hiding in software engineering, and the relevance of this hidden software with the development of Open APIs. They identify APIs as a way of hiding functionality of software, as APIs support the separation of an interface from its implementation. For vendors, the main advantages of deploying APIs come from 'the possibility of separating modules into public (the API itself) and private (the implementation of the API) parts' (De Souza & Redmiles, 2009). In this sense, adopting an API contains two level of interaction, and external users are dealing with two objects: modules (which this dissertation also refers to as 'interface') and implementation (which this dissertation also refers to as 'data' or 'functionalities' from underlying software).

Technically speaking, one Open API is just an interface to specify a way for integration and it cannot function without the underlying software or functionality (e.g., Google Maps is the underlying software and data set of Google Maps API). However, the meaning of 'Open API' has been conflated with the underlying functionalities in recent years. According to Stanton (2015), the term API was initially referred to a precise specification that helps developers to connect software components: 'It describes what functionality is available, how it must be used and what formats it will accept as input or return as output.' However, now API is colloquially used to describe both the interfaces and the underlying service by many programmers (Patterson, 2015). As a result, what is actually open has been confused, and to what extent it is open becomes controversial. Thus, we need to begin with answering the two sets of questions about openness in terms of two levels of coordination in API use.

The integration between the programs of external developers and API providers specifically contains two levels of interaction: external programs and Open APIs themselves; external programs and underlying software or implementation that APIs link to. These two interactions are deeply intertwined and cannot be completely separated. Ideally, the openness of Open API should draw from both of these two levels. Figure 2.1 outlines this openness:

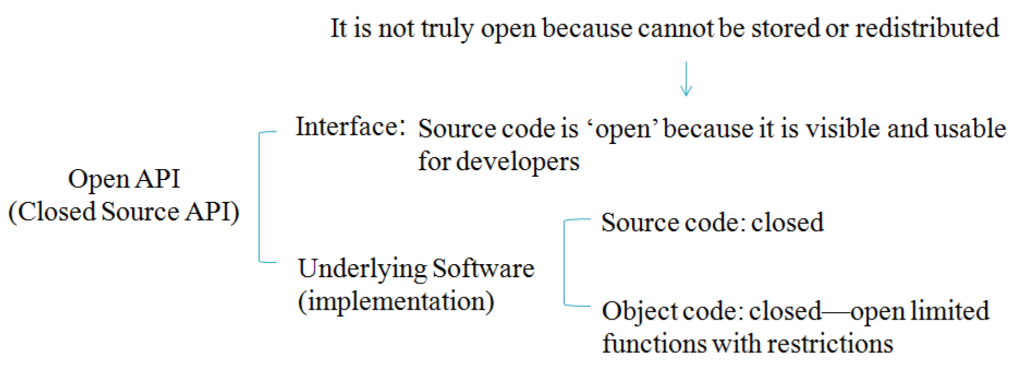

It is not truly open because cannot be stored or redistributed
Interface:
Source code is 'open' because it is visible and usable for developers
Open API (Closed Source API)
Source code: closed
Underlying Software (implementation)
Object code: closed—open limited functions with restrictions

Figure 2.1: Openness of Open API

In the level of interface, the source code of Open APIs is 'open' because it is visible for public and developers can use these codes to retrieve data from underlying software. However, sometimes these codes have restrictions on redistribution and some of them are not free to use. For example, the Google Maps Terms of Service sections 10.1.1, 10.1.3 and 10.2 all specify limitations on the redistribution of users' software after integrating with Google Maps API (Google Maps/Google Earth APIs TOS, 2013). In addition, Google Maps API has a request limit per day that prohibits commercial users from using it freely (Google Maps API FAQ, 2015). Thus, in the first level, the openness of interface refers to merely visible and limited usable.

In the second level, the object code of underlying software is available, running as popular platforms such as Google Maps and Twitter. However, their source code is not available for programmers. Corporations tend to keep the source code of underlying software secret, as they previously did with other examples of proprietary software. The only way to interact with underlying software is to use a vendor's API without examining the source code. Its object code is also closed as platforms usually just provide limited function to different users and sometimes charge fee for premium features.

Power of the Pre-defined 'openness' of Open APIs

Some articles encourage developers to embrace Open APIs, and yet do not make a clear differentiation between Open APIs and Open Source (Lyman, 2012;

Carrillo-Tripp et al., 2009; Aitamurto, & Lewis, 2013). These two different kinds of technologies contain different affordances. The following chapters will elaborate unique features of Open API with specific examples about these affordances from the perspective of political economy, while the point here is to clarify a misunderstanding: the source code of interfaces must be differentiated from the source code of underlying software when analysing the openness of Open APIs. The source code of APIs may be open for developers to study and use, while the source code of underlying software is always closed for most vendors' owned APIs. In fact, a key problem for vendors has been solved by APIs: How can we let external programmers integrate and build upon our software while minimising the potential threats to our intellectual property? The solution is to obscure any underlying software and provide APIs. By means of integrating APIs, external programmers can 'use' the underlying software without the vendors' source code, but this implementation is limited in 'use' and never 'owned' by anyone other than the vendor. Michel (2013) introduces a metaphor used by David Orenstein to explain how APIs works:

> Let's imagine that you are building a deck in your backyard and you realize [sic] that you don't have a hammer. You have three neighbours who you know have a hammer. One neighbour never allows anyone to borrow anything or use any of his stuff without paying up. This neighbour represents a closed or proprietary system. Another neighbour leaves his garage open and allows you to take anything you

need without any rules or guidelines. This is your bearded open-source neighbor. The third neighbour represents an API. You can utilize [sic] the services of her hammer as long as you ask her in the proper way. Furthermore, she doesn't actually give it to you; she just allows you to use it (Michel, 2013, p. xi).

API providers define the 'proper way' to use, and the code of APIs which designed by vendors implicates how 'open' Open APIs are.

Example of using Google Maps API.

Analysing the code of Google Maps API is an entry point to explain the openness of APIs. Creating a digital map in one application from scratch is complicated, but applying APIs to retrieve an existing map from vendors is much easier. Here is an example of creating a digital map centred on the University of New South Wales via the Google Maps API (W3school, 2014):

```
<!DOCTYPE html>
<html>
<head>
<script
src="http://maps.googleapis.com/maps/api/js?key=A
IzaSyDY0kkJiTPVd2U7aTOA
whc9ySH6oHxOIYM&sensor=false">
</script>
<script>
function initialize()
```

```
{varmapProp = {

center:newgoogle.maps.LatLng(-33.91722,
151.23064),

zoom:16,

mapTypeId:google.maps.MapTypeId.ROADMAP };

var map=new
google.maps.Map(document.getElementById("googleMa
p"),mapProp);}

google.maps.event.addDomListener(window, 'load',
initialize);

</script>

</head>

<body>

<div id="googleMap"
style="width:500px;height:380px;"></div>

</body>

</html>
```

Figure 2.2: Creates a digital map centred on the UNSW via Google Maps API

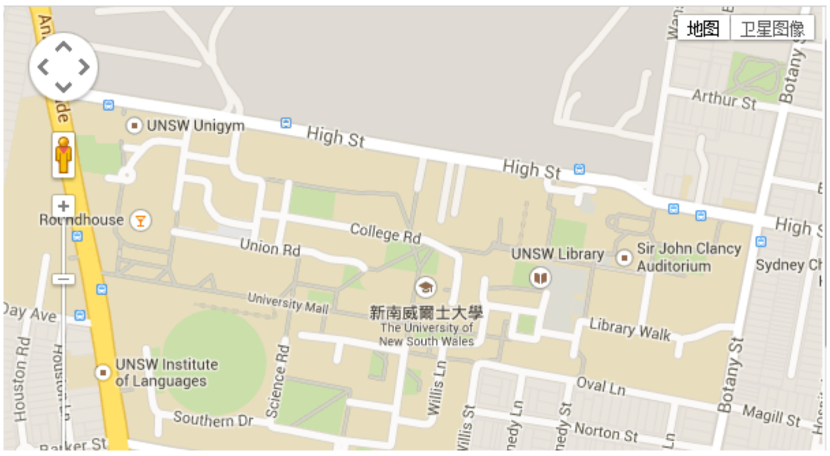


Figure 2.3: Result of digital map centred on the UNSW via Google Maps API

These lines of code are a basic example of using the Google Maps API. When external developers put this code, or a version similar to this into their websites, they can retrieve a map (Figure 2.3) and embed it into their web pages. In addition, external developers can also customise the maps with their own information. For example, they can change the colour of the map, add markers such as parking areas or add a custom legend following the using Google Maps API tutorial (Google Maps Tutorials, 2014). The result of customised map can be seen in Figure 2.4:

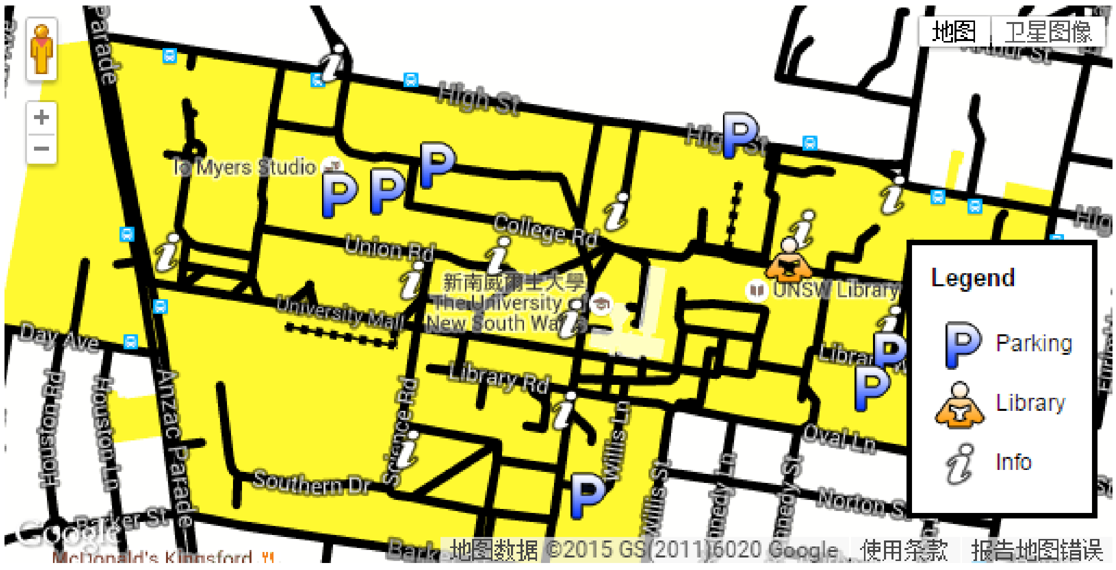


Figure 2.4: Result of customised digital map

Figure 2.4 is the customised digital map based on Figure 2.3. To be clear, the maps created by Open APIs are not static pictures, but interactive. End users can zoom in or explore street view on the embedded maps, in the same way they can with Google Maps. As we can see here, Open APIs in these cases are just several lines of code, but the result shows the whole function of interactive Google Maps centred on the UNSW. Two things should be mentioned here: First, the APIs connect to Google Maps and authorise users' API key in the beginning of the codes via:

-
  ```
  <script
  src="http://maps.googleapis.com/maps/api/js?key=AIz
  aSyDY0kkJiTPVd2U7aTOAwhc9ySH6oHxOIYM&sensor=false">
  ```

The API key authorises API users and gives them access. Second, there is no source code or object code about original Google Maps; all codes for creating the digital map are the source code of an API—interface that specifies how to retrieve data from the underlying Google Maps. Without the key data of original Google Maps, these codes only contain some parameters of how to display Google Maps such as the coordinates: -33.91722, 151.23064, or the zoom scope: 15. In short, APIs just define a way to retrieve certain functions of Google Maps and customise a map after the authorisation of users' unique API key. While the source codes of interfaces are available and editable, the source codes of Google Maps remain secret by vendors. Every time users need a digital map from Google Maps, they request it via their unique API key allocated by Google and use the code of interfaces defined by Google. Making underlying data closed source and using API key are two ways for vendors to control external developers.

Building asymmetrical relationships between vendors and developers—power from closed source feature.

With Open APIs, vendors find a way to provide their proprietary data to external developers while keeping the source code of their software secret—providing Open APIs to their closed/proprietary software. Keeping code secret protects the software of vendors from being exposed, modified or reduplicated. To some extent, it also reduces the workload of external developers, as will be examined in Chapter Three. While increasing numbers of developers are deploying Open APIs, there has been controversy among the programming community about the usage of Open APIs and their advantages, especially when compared with Open Source. Some developers think source code availability of underlying software does not matter, because executable functions that APIs retrieved are enough for software integration (Soulskill, 2011), thus Open APIs could be a replacement for Open Source (Lyman, 2012). However, other users point out the potential risk when Open Source Software is becoming increasingly replaced by Open APIs:

> Seriously though, the source availability may not be of use to you now, directly but, if you don't think it is of use to you, then realize [sic] that if the service with the published API goes away, you don't have a starting point to replace it, other than the API docs. (TheCarp, 2011)

Once users embrace an API without source code of underlying implementation, they are bound to an overpowering partner. The modification and availability of the underlying implementation of an API could be fatal to external developers' application. As source code is totally owned and controlled by vendors, external developers would lose control over their own product while they become dependent on APIs.

For example, Google Reader was a platform for serving news and information. It aggregated web content such as news, blog posts and video together in one place for easy reading. Google Reader provided an API for external websites to integrate with it for significant data. However, as the usage of Google Reader declined, Google decided to close it in July 2013 even though there were still users using it (Google, 2013). As many users found Google Reader valuable, they launched a petition to urge Google to keep Google Reader (Albanesius, 2013). Realising the difficulty in persuading Google to keep running Google Reader, some users asked Google to open source the code and let the fans take control: 'We will find a way to monetise it, we will find a way to keep it going' (Albanesius, 2013). Yet, Google did not respond to users' requests. As Google only gave a notice of three months before deleting all the data, many users failed to continue their applications or websites. Thus, users have to save their data and find an alternative to store it as soon as possible. Many people asked Superfeedr (another feed API which aggregates web content) if it could help them save the data from Google Reader API (Julien, 2013). However, developers of Superfeedr Blog claimed it was difficult to find the right tool to store all this data and they could only try to ‘back up as much of the data’ as possible. Moreover, to continue their websites means to find an alternative service to store significant data, which is very difficult in a short time (Julien, 2013). In order to maximise profit, Google and other commercial organisations sometimes shut down their services and their APIs when usage declines, regardless of the developers who are reliant on these services. This is one risk of using API without underlying source code of implementation.

Another concern argues the potential risk of engaging with closed/proprietary software via API is they might contain unwanted functions or resources that developers do not know. A developer wrote a post to state his concerns about using API: ‘You don’t know what the code’s going to do. Specifically, what will it do to you or your business? How will it use or abuse your data?’ (MBR1, 2012). Developers only get an executable file, not the source code, which makes it hard to detect what the program really does, and hard to change it. Services or data may contain malicious content or bugs which is detrimental to end users of external programs—those who API vendors do not or cannot care for. APIs themselves can be used by hackers to deliver malicious content. As the source code is not available, it is difficult for developers to detect the malicious features embedded in Open APIs. For example, security researcher David Sopas has discovered a bug in the Instagram API that could be exploited by hackers to post a link to a web resource they manage (Fisher, 2015). Via this bug, hackers can post a link which points to a malicious file for end users to download which appears to come from a legitimate Instagram domain. 'The attacker can host any malicious file he chooses at the target location, including malware'. Sopas points out that this kind of bug is 'very dangerous and combined with other attacks like phishing or spam it can lead to massive damage' (Fisher, 2015). Although developers discovered this bug and find it dangerous to their services, they cannot fix it by themselves without underlying source code. All developers can do is ask vendors to fix the problem they have identified. As Sopas said, he tried to convince Instagram to solve this problem as soon as possible, but the security engineers of Instagram told him the issue was not a priority.

While the safety and privacy of developers' labour cannot be guaranteed, vendors usually deploy detailed clauses to avoid their programs being harmed while letting users bear the risk alone. For example, Google clarifies its innocence when developers' software is damaged:

> 13.2 You expressly understand and agree that your use of the service and the content is at your sole risk and that the service and the content are provided 'as is' and 'as available'. […] 13.3 Any content obtained through the use of the Google services is done at your own discretion and risk and you will be solely responsible for any damage to your computer system or other device, loss of data, or any other damage or injury that results from the download or use of any such content. (Google Developers, 2013)

By hiding their source code, vendors ensure the ownership of their software and avoid the potential risk when integrating with external developers' software. Without the source code, external developers are rendered passive and dependent at the same time. This asymmetrical relationship is due to the closed source feature of Open APIs. Open APIs can be open source, but vendors choose not to do so. For Galloway, code is designed in particular ways to achieve particular ends in particular contexts (2004, p. xii); for vendors, designing APIs in this closed source way aims at achieving their domination in programme development.

Management of API key as a form of control.

The API key plays an important role for vendors' control of Open APIs. For most Open APIs, the first time an external programmer intends to use them, he needs to sign up and apply a 'key' from the providers. 'A key is a string of numbers and letters that identifies you as the source of an API request' (Yee, 2008, p. 122).

```
<!DOCTYPE html>

<html>

<head>

<script

src="http://maps.googleapis.com/maps/api/js?key=A
IzaSyDY0kkJiTPVd2U7aTOA

whc9ySH6oHxOIYM&sensor=false">

</script>
```

```
<script>
[…]
<html>
```

Figure 2.5: Key of users in API

In Figure 2.5, 'IzaSyDY0kkJiTPVd2U7aTOAwhc9ySH6oHxOIYM' is the key of current user, and an API usually authorises a user's identity with the API key in the beginning of the code. Each time users intend to use an API they need their unique key to grant them access, just like some websites require users to log onto their personal account before entering. This key not only grants a user access to the API, but it also identifies the user, thus enabling vendors to track and manage that user's usage of the API. Yee identifies the power of the key:

> Through keys, the API provider knows something about the identity of an API user (typically at least the API key holder's email address if nothing else) and monitors the manner in which a user is accessing the API (such as the rate and volume of calls and the specific requests made). Though such tracking, the API provider might also choose to enforce the terms of use for the API—from contacting the user by email to shutting down access by that key…to suing the user in extreme cases! (Yee, 2008, p. 122)

Keys are required for entrance to most Open APIs today. But once there is a key, it means there is a lock—vendors keep track of users, controlling their use of the API, and reserve the right to suspend the external developers' usage. The API key, as a pre-defined requirement for developers, entails a sort of protocological control that limits developers in the beginning, and it is almost impossible for external developers to escape this pre-defined power of code.

As an API key grants developers access, controlling usage of APIs can also be implemented by revoking or suspending an API key. What should be mentioned here is: an API key does not inherently exist in API code but is designed by vendors as a way of control. Some APIs do not require a key for access in the beginning, but most of them deploy an API key as a management style currently. For example, Twitter API did not require all developers to be authorised in the beginning, but the vendors increased restriction and deployed an API key (known as OAuth) to manage accessibility after a few years (Bucher, 2013). When Twitter figured out some audiences of Twitter were going to third-party applications based on Twitter API, it felt unprecedented threat. From 2010, Twitter enforced increasing restriction to limit third-party developers. On March 11, 2011, Twitter's platform manager Ryan Sarver even posted a message to urge developers to stop making applications which provide similar user experience to Twitter. Finally, in late 2012, Twitter updated its APIs versions to 1.1, which has several significant changes including the requirement of an API key.

Michel Slippey, as the vice president of product and design at Twitter, outlined and explained these changes in a developer blog about Twitter API v1.1. According to him, the new Twitter API requires ‘every request to the API to be authenticated’, because Twitter found many applications were pulling data from Twitter at very high rates and some of them might have malicious use. It also sets per-endpoint rate limiting on the API to limit the traffic. Moreover, new rules require developers to work with Twitter APIs directly if they have large amount of users. Developers’ applications which have more than 100,000 user tokens, from then on, have to gain permission from Twitter directly and are subject to additional terms (Twitter, 2015). Twitter said by direct cooperation they could guide developers ‘toward areas of value for users and their businesses’ (Slippey, 2012), but the implication of these rules are more than just a guide: it lets Twitter enforce control and surveillance on third-party applications especially those which have high volume of API use. In addition, new rules also prohibit creating certain kind of applications which are similar to Twitter’s own function, and existing applications which offend these rules are being shut down. Tweetbot and Echofon are two applications that have been forced to close. According to Slippey, Twitter’s dominance over third-party applications must be ensured: ‘Nearly eighteen months ago, we gave developers guidance that they should not build client apps that mimic or reproduce the mainstream Twitter consumer client experience […] And to reiterate what I wrote in my last post, that guidance continues to apply today’ (Slippey, 2012). For the vendors of Twitter, using an API key to monitor and control developers can significantly cut down the threat of competitive third-party applications and ensure vendors’ dominance.

The control from an API key also happens in the *Guardian*'s APIs, which provide access to news articles, images, audio or videos. 'If a partner is in breach of our terms we will terminate their key. We will also terminate keys when partners abuse our service, use it to compete with us unfairly, or damage our brand' (The Guardian, 2014a). In addition, an API key also plays an important role in differentiating different levels of developers of *Guardian* APIs, which is a kind of discrimination that offends the 'openness' of FLOSS.

As we examined before, Open APIs do not open all functionalities to all users, vendors can open specific, limited function to specific developers. This discrimination can be implemented via an API key to classify developers, so vendors can gain more profit by targeting specific developers. *Guardian* API is a typical instance of this discrimination. There are three levels of access for *Guardian* API users: Keyless, Approved, and Bespoke (The Guardian Open Platform, 2014a). Keyless tier is the entry level. 'Keyless' here might mean an API Key is not even needed, as developers do not need to be authorised when access the data through Keyless tier. However, it only offers developers the right to access the headlines of news instead of the full text. In addition, developers' IP addresses will be subject to limitation on the times of requests per day. The second level—Approved level—provides full articles to developers, but developers have to apply an API key for this level. Moreover, in this level, the *Guardian* keeps the right to 'sell the ad inventory embedded within *Guardian* articles wherever they appear' (The Guardian Open Platform, 2014a). Thus, external developers who access *Guardian* API through Approved level have to display ads and let vendors profit from advertising. The third one—Bespoke is a high level of access and requires unique API key. It offers custom ways for software integration and provides premium services. In this level, *Guardian* API grants users unrestricted access and ad-free services, thus third-party developers do not need to worry about the interruption of advertisements and limitation of content. For external developers, different kinds of API keys determine how they can integrate with *Guardian* API, these integrations define their programming process in the beginning.

In short, vendors make source code of implementation closed and deploy API keys for controlling external developers, so these two technical transformations become pre-defined features of most Open APIs today. As Chun claims, code enforces power through the production of 'self-enforcing rules' for governing. These two features of Open APIs entail irresistible forces that coerce developers to act in particular ways. Developers' independence and freedom are limited by the code of APIs, and as vendors define the code, developers are under the governance of vendors via inflexible code. The codes of APIs become a way for vendors to utilise surveillance, control and discrimination on developers. As APIs are proprietary, vendors can alter the code to fit their purpose as a new form of management, thus power of protocol constitutes a coercive means of control designed by vendors.

The ‘openness’ of Open APIs.

According to the Open Source Initiative (OSI, 2014), open source should have no discrimination and restriction on users. However, most Open APIs have many limits on use such as request time and ways of using, and some Open APIs set different levels of access to differentiate users (Google Developers, 2014; Bucher, 2013; The Guardian Open Platform, 2014a) as we discussed above. As companies usually own Open APIs, they clearly belong to the category of proprietary rather than ‘open’. They usually have copyright protection and keep source code secret (Free Software Foundation, 2014). For Kelty, proprietary is opposed to open and free: ‘the opposite of an “open system” was not a “closed system” but a “proprietary system”’ (Kelty, 2008, p. 149).

Thus, the ‘open’ of Open API is, at best, a discursive trick. Many people automatically associate ‘open’ in the world of software and programming with the openness of open source. These APIs are often thought to be a new means for ‘open’ and ‘free’ collaboration and peer production. However, most Open APIs (except for some APIs that are actually open source) are not open source software and the ‘open’ only means that an API is open for public use rather than restricted to private use within one organisation. This ‘open’ means the certain kind of services that vendors provide are visible for public consumption, and external users can implement these services via API without direct contact with vendors and their underlying, proprietary software and data. However, visibility and limited access are far from the essence of the openness advocated by the standards developed within the FLOSS movement. Compared with the criteria of open advanced by FLOSS, Open API is not open at all. In the level of interface, the source code and executable file of APIs are exposed to users, but limitation on accessibility are enforced by vendors. Moreover, as some companies own the copyright of API, redistribution without permission or any imitation might result in legal issues. In the second level, the underlying software is no different from the previous closed or proprietary software—users only get executable files without the source code. In fact, Open APIs only give users the right to retrieve a specific, complied, executable form of underlying software, while the source code of both APIs and the software behind them differ from the openness of FLOSS.

As Kelty (2008, p. 303) claims, a great deal of social software today merely borrows the idea of Open Software, but is, in fact, not open at all. This software merely intends to make users into consumers (Kelty, 2008, p. 303). In essence, Open API is an affiliated tool of proprietary software, which borrows the name of open and is used by vendors to pursue profit through the language of the movements that previously shaped the communal and political use of software as a form of anti-corporate activism.

Chapter 3: Profiting from Closed Source Open APIs—How Vendors Exploit Developers

As examined in Chapter Two, despite their name, Open APIs are neither open through their interfaces nor through their underlying software. Technically, these Open APIs are Closed Source APIs, and are usually owned and managed by commercial organisations. However, another set of Open APIs—Open Source APIs—does exist, which follows the criteria of the ‘open’ standards of FLOSS. For example, OpenStack is a set of open source software for building and managing cloud computing platforms. It aims to build an open source community with researchers, developers and enterprises to create a scalable cloud computing system (Wen, 2012). OpenStack APIs let developers access all the components of OpenStack and use predefined functions (OpenStack, 2014). As they are developed under open processes, followed by open licenses with open governance, OpenStack APIs are Open Source APIs. Although Open APIs contain both Open Source APIs and Closed Source APIs, Open Source APIs are often ignored by developers, and their numbers and popularity do not rival Closed Source APIs. As shown on *Programmableweb.com*, a popular website for APIs developers to communicate and share ideas, most popular APIs discussed, including those operated by Facebook, Google Maps, Twitter and YouTube, are all owned by commercial organisations (Duvander, 2014). When programmers hired by these corporations develop their APIs, they do not follow the model of those from the FLOSS community who aim for sharing and peer-production, and are instead beholden to their employer. They are employees serving the companies they work for, which use APIs to seek financial profit. As a result, the 'open' of their final programming product is different from the 'open' of FLOSS community. As Closed Source APIs account for a major proportion of popular Open APIs today, they in effect force some programmers to overlook the differences between the 'open' of 'Open Source' and 'Open APIs'.

In this regard, this chapter will focus on how these prevalent Closed Source APIs, as particular kinds of Open APIs, have become dominant in contemporary Internet culture and facilitate a new form of participation. It first identifies the convenience of adopting Open APIs to advance external programs, and explores how enthusiastic external developers are in using these APIs. It then examines this enthusiastic participation in programming in terms of critical concepts such as participatory democracy, peer production and recursive publics, and points out these modes of participation constructed by Open APIs are exploitative and can be utilised by companies. After that, this chapter introduces the theories of exploitation developed by Marx, Fuchs, Andrejevic and others, and then discusses how to apply their concepts of exploitation to understand the power of digital technology; specifically how to examine the collaboration of Open APIs in terms of a critical political economy of digital media.

The Convenience of Using Closed Source APIs

How Open APIs reduce the cost of collaboration and facilitate the integration of software has been identified by many studies. Rather than discuss the impact of Open APIs on the whole software industry, this section argues Open APIs, at least, are a fast and easy way for individuals to integrate with vendors' software, and this convenience is a major advantage of Open APIs over Open Source, which ensures the domination of this new 'openness'.

Once programmers find a useful API, they can learn to use it and embed them into their application in a very short time. Even for nonprofessional programmers, to begin with an unfamiliar API is not a difficult task if they have basic programming knowledge (Gray, 2015). There are three reasons why Open APIs are seductive for external programmers: they are easy to access, easy to use and they provide substantial infrastructures for programming.

First, gaining access to an Open API is much easier than obtaining and making use of the source code of concrete software. The underlying proprietary software contains substantial resources for external developers, but it usually takes time and effort to get direct access from vendors if developers need an entire source code of software. Using Open APIs is another way to collaborate with proprietary software rather than negotiate with vendors, signing a contract after negotiation, and obtaining direct access to source code. Although developers have to accept the TOS of APIs and get an API key before using, this process usually just takes a few minutes and users do not need to consciously engage in the establishment of these legal contracts. For instance, if developers intend to build an application based on the Twitter API, they just need to do three things:

1. Log on to their Twitter account;
2. Go to the ‘create an application’ page and fill in a simple form—which only requires the application’s name, description and URL;
3. Read the Developers Agreement (which many individual users do not actually read, anyway) and agree to it (Twitter, 2014).

After these three steps, Twitter grants developers an API Key, and API users gain access to the Twitter API. Obviously, it is much easier to begin with an Open API than through direct contact with vendors to ask for access to original source code.

Second, without the process to study the underlying software, developers can improve their own services and bypass any effort to study complicated source code. Unlike dealing with software from companies or the FLOSS community, developers just need to examine and use the source code that APIs provide from the level of interface instead of the complex underlying software. Although different APIs vary in their source code, and it still takes time for developers to learn how to program with a specific API, they are all simpler than dealing with the source code of an entire corporate platform. In addition, companies are constantly updating their APIs, not only to provide new functions, but also to simplify code to maintain ease of use. Twitter's new REST APIs (a new style of API developed under Representational State Transfer—a new and light web service architecture), for instance, provide easy programmatic access for external developers to read and write Twitter data. If developers need a collection of the most recent tweets posted by specific Twitter users, they just need to log on their account, authenticate their API key, and simply enter the API request URL into their browser:

-
  ```
  GET
  https://api.twitter.com/1.1/statuses/user_timeline.
  json?screen_name=twitterapi&count=2
  ```

Twitter will then send back the result code for developers for copy. This request comparatively easy to learn: 'GET' means retrieve information; 'https://api.twitter.com/1.1/statuses/user_timeline' is the resource URL that Twitter has defined for this request; 'json' is the format of result code, can be replaced by XML, RSS or any other format that Twitter supports; 'screen_name' is the name of a user, which, in this example, is 'twitterapi'; and 'count' is the number of tweets developers intend to retrieve, in this case is two ('2') (Twitter Developers, 2013). There are a number of other parameters that can be added to the request, which can be found in the tutorial (Twitter Developers, 2013). But the point of this example is that a simple request can let developers get a set amount of information through the new Twitter API.

Vendors usually specify categories for their APIs, and have detailed tutorials for them (e.g. Twitter Developers, 2015b; Google Maps Tutorials, 2014). Some API providers archive code samples so users do not need to remember the actual code of each API. For example, Google archives sample code of Google Maps JavaScript API v3 for reference at https://developers.google.com/maps/documentation/javascript/-examples/. Moreover, some APIs have a system that lets programmers customise how they access the API via a simple form, which then automatically generates source code for programmers. Programmers just need to copy and paste rather than write the code themselves. Twitter's new OAuth Tool, for instance, can generate part of the final source code that developers need to interact with Twitter (Twitter Developers, 2015a). Some API providers like Autonavi even provide users with visual tools, and amateur programmers with little coding knowledge can simply change the parameters in a form (Autonavi LBS Open Platform, 2014a), or edit digital maps to depict intended results (Autonavi LBS Open Platform, 2014b). These visual tools can then generate source code to be used to interface with these platforms. Programmers merely need to copy and paste API code into their software.

Third, some Open APIs provide programmers with a solid infrastructure so they can build new applications that then augment a specific platform. It takes a huge amount of effort for an individual to launch a location-based service from scratch. However, building these applications on the infrastructure afforded by, say, the APIs of Google Maps or Foursquare is significantly easier. Innovative developers can also integrate several APIs and build a new application without any first hand resources. For example, AA Route Planner uses the Google Maps API for driving directions, and retrieves real time traffic conditions from the TomTom Geocoding API, all to create a comprehensive route plan for drivers (AA, 2014). Location Based Instagram Photos is another interesting application built by third-party programmers. It is based on two different APIs: the developer uses the Google Maps API and the Instagram API to display Instagram photos based on your location (MobileMerit, 2014).

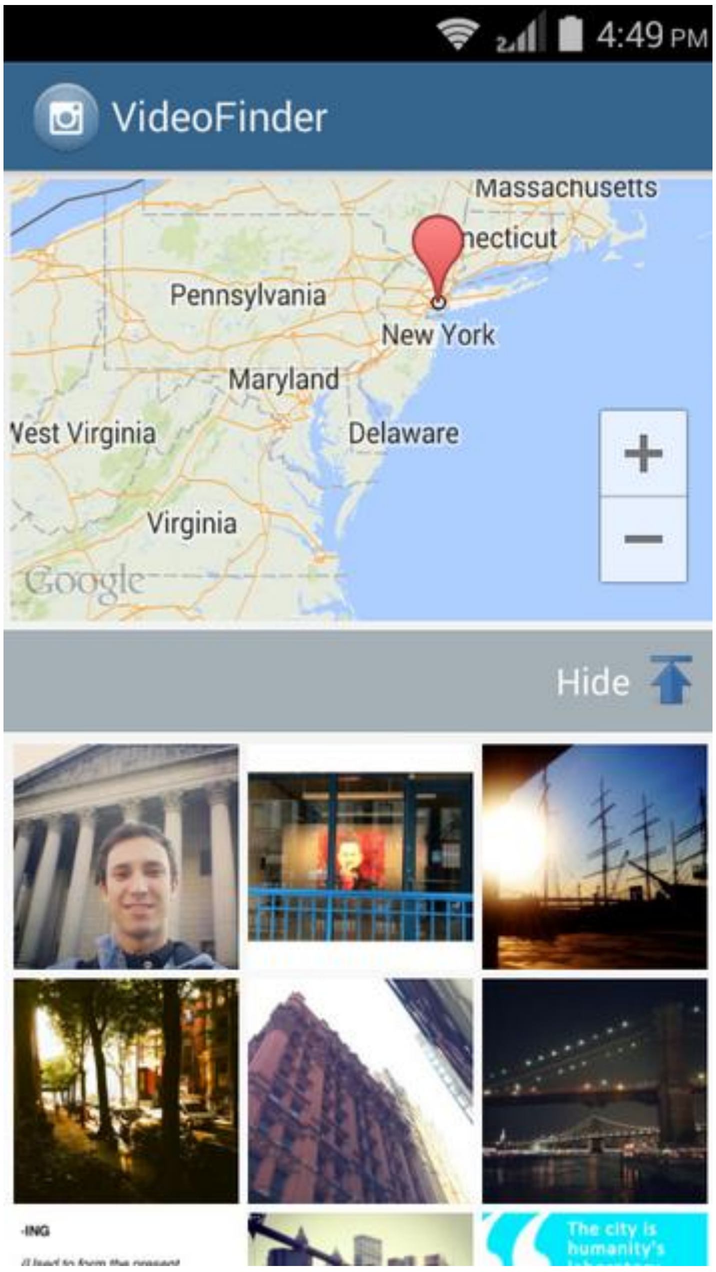


Figure 3.1: Location Based Instagram Photo

The convenience of integrating Open APIs gives programmers an unprecedented ability to improve existing applications or build new applications without too much effort, and nonprofessional programmers who are interested in software can build their own application through APIs. Rather than handling complicated copyright issues and dealing with the complexities of the source code of an entire platform, programmers implementing APIs only need to sign a user agreement and learn simple programming interfaces. The latter choice also offers substantial resources and functionalities for external programmers.

Participating in Using APIs and the Feeling of 'Fun'

As discussed in the previous section, APIs seem like a terrific opportunity for programmers to enhance their applications, which is primarily why they are welcomed. Indeed, studies show that developers are often even thought to be addicted to 'playing' with Open APIs. Bucher (2013) examines developers' addiction to APIs through an investigation of individual responses to the Twitter API. As Bucher shows, '[u]tterances like "engineer by day, Twitter hacker by night", or "making apps for fun after work"' is common parlance among Twitter developers forums. Some developers' vocations are completely irrelevant to Twitter API, but they still use it in their spare time as it is 'a super fun way to get into coding' and some of them hope to create the 'next killer app' (Bucher, 2013).

These examples show the fact that, while some have to deal with APIs in their daily work, programmers still spend plenty of their spare time playing with APIs in their daily life as a gratification. Developers are enthusiastic in participating in programming with Open APIs, and their enthusiasm is similar to what they had on FLOSS. However, as this 'openness' is different, as discussed in Chapter Two, engaging with APIs and FLOSS can carry different politics for programmers.

Participatory democracy in digital media.

Fuchs' (2013) discussion of participation and exploitation in social media is an entry point to identify the external developers' usage of Open APIs. While Jenkins (2008), Benkler (2006), Tapscott and Williams (2007) argue that contemporary social media and Web 2.0 technologies require active 'participation' and emphasise the positive aspects of this 'participation', Fuchs proposes suspending this judgement and understanding participation in terms of participatory democracy. Drawing from Staughton Lynd (1965), participatory democracy means all members in a society have the right of decision-making, and all members control the organisations that affect them. It not only refers to a political democracy that all people have control over society, but also refers to an economic democracy in which producers control the production process and the means and outcomes of production (Fuchs, 2013, p. 14).

In this regard, only having the right to create and share content in digital media does not constitute real participation: 'Participation means that humans have the right and reality to be part of decisions and to govern and control the structures that affect them' (Fuchs, 2013, p. 57). Real participation in an Internet platform relies on its participatory ownership structure. Questions about the ownership of platforms, collective decision-making and profit should be concerned when exploring participation in digital media. In this sense, corporate social media does not constitute real ‘participation’ as it lacks a participatory ownership structure. To apply this logic to software industry, we can see software from FLOSS usually does not have copyright protection, so programmers can use, modify and redistribute them in the ways they want. Programming with FLOSS, to some extent, fits the meaning of ‘participatory democracy’ as developers have the right to govern and control both the programming process and final software product. However, as Closed Source APIs are proprietary and belong to profit-seeking vendors, forces and restrictions can be embedded in the programming process and final programmes. This reduces developers’ control over software and eliminates the potential of participatory democracy.

Market-based Capitalist Production Beyond Peer Production

The implication of participation can be further elaborated through the discussion of coordination and peer production. As Kelty (2008) points out, for many participants and observers, coordination is the central innovation and essential significance of Open Source: ‘the possibility of enticing potentially huge numbers of volunteers to work freely on a software project, leveraging the law of large numbers, “peer production”, “gift economies,” and “self-organizing [sic] social economies”’ (Kelty, 2008, p. 210). In software development, peer production means a group of developers working together freely in a shared software product, which is a typical feature of

FLOSS. Bauwens (2012) points out the importance of peer production:

> Peer production is both immanent and transcendent vis- à-vis capitalism, because it has features that strongly decommodify both labor and immaterial value and institute a field of action based on peer-to-peer dynamics and a peer-to-peer value system. Peer production functions within the cycle of accumulation of capital but also within the new cycle of the creation and accumulation of the commons. (Bauwens, 2012)

Peer production, in this sense, is not only a method of coordination, but also a way to avoid commodification in the cycle of accumulation of capital. However, this peer production is unlikely to happen in programming through Open APIs. While Open APIs attract many developers to participate in programming, they do not facilitate actual peer-production because there is no shared product for developers to modify and redistribute. There are two kinds of products generated in the use of Open APIs. The first kind of product is the API itself, which is proprietary and maintained by vendors. As will be examined later in this chapter, vendors update APIs based on third-party users' efforts, or develop new products based on the information gathered from external developers. As all these products are owned by vendors instead of opening among 'peers', they lack a 'peer-production' element. The second kind of product is the developers' program, which has been advanced by Open APIs. As developers' programs contain proprietary APIs of vendors after integration, their ownership might be affected by commercial organisations. As will be examined in Chapter Five, the redistribution and modification of developers' programs are limited according to the TOS of APIs, which make developers' programs become dependent on vendors. This means external developers who used Open APIs as components can no longer share their program among the community.

As these two kinds of products cannot be freely shared in communities for developers to modify, peer-production can hardly happen in the usage of Open APIs. Unlike FLOSS, on which programmers can collaboratively develop derivative works, Open APIs only let programmers develop their own software without the right making it public for modification and redistribution. Thus, there is no coordination between external developers on a shared product because the vendors of API do not allow it.

The significance of Open and Free software, as Kelty notes, is the constitution of *recursive public*: 'constituted by a shared concern for maintaining the means of association through which they come together as a public' (2008, p. 28). Programmers find affinity, and work to maintain this affinity (Kelty, 2008, p. 28), with their recursive public, facilitating contribution and collaboration. Furthermore, Kelty argues that social media like YouTube, MySpace and Flickr take inspiration from Free Software but do not forge a recursive public that promotes collaboration, because 'most of them are commercial entities whose structure and technical specifications are closely guarded and not open to modification' (Kelty, 2008, p. 303). As 'intellectual property is a central and dominating concern', the proprietary nature of these social platforms limits the potential for collaboration (Kelty, 2008, p. 303). Open APIs are also proprietary; they do not build 'recursive publics', but instead function as a strategy for vendors to encourage 'interested users to become consumers in more and more sophisticated ways'(Kelty, 2008, p. 303). As Open APIs borrow the name of 'open' but are inherently commercial entities whose structure is closed, they lack this 'recursive' commitment. As developers become increasingly in favour of using Open APIs, it seems that market-based capitalist production wins this time over peer production. Peer production and recursive features of coordination were removed because vendors reinforce the closure on Open APIs and make them proprietary and closed source. As a result, what left is inherently exploitative.

Given the situation that developers are increasingly programming with exploitative Open APIs, it is essential to examine how exploitation happens and what the political implications of such an engagement are. The issues surrounding these questions can be clarified through the discussion of exploitation and domination from a political economic perspective. The remainder of this chapter will then focus on predominant forms of capital accumulation and exploitation underlying the usage of these non common-based Closed Source APIs.

Exploitation in the Usage of Digital Technology

A critique of gratification and exploitation.

First, it is essential to identify the new form of exploitation in digital era. Exploitation sometimes does not feel like exploitation, even though it nonetheless exists in the everyday use of social media. As discussed in last section, if we only look at users' enthusiastic participation in social media, we might ignore the broader possibilities and limitations of participatory democracy and its implications for the Internet. If we consider the capitalist infrastructure behind it, we can see that 'An Internet that is dominated by corporations that accumulate capital by exploiting and commodifying users can never, in the theory of participatory democracy, be participatory and the cultural expressions of it cannot be expressions of participation' (Fuchs, 2013, pp. 65-66). Thus, we cannot only look at the benefits of using corporate social media and users' willingness for participation while neglecting aspects of class and power behind this use. As Fuchs claims, domination and exploitation should be understood via the language of critical theory in terms of property and controls:

> Critical theory questions all thought and practices that justify or uphold domination and exploitation. Domination means that one group benefits at the expense of others and has the means of violence at hand that they can use for upholding the situation where the one benefits at the expense of others.

> Exploitation is a specific form of domination, in which one group controls property and has the means to force others to work so that they produce goods or property that they do not own themselves, but that the owning class controls. (Fuchs, 2013, pp. 13-14)

Drawing on this view, Fuchs examines Google's so-called 'playtime' policies designed to please employees with 'fun' work environments, arguing how these strategies are aimed at making employees work longer and more efficiently, which 'result in a lack of work-life-balance and feelings of stress' (Fuchs, 2013, pp. 149-150). Another platform which, according to Fuchs, follows a communist production model that is independent from corporate power, but is nonetheless under the risk of capitalist exploitation is Wikipedia: 'These communist potentials are, however, antagonistic because of the use of the Creative Commons Attribution-Share Alike license that allows the selling of Wikipedia content as commodity. In those cases where an article is sold, all underlying voluntary work is unpaid labour and the involved Wikipedians are infinitely exploited' (Fuchs, 2013, p. 248).

In this sense, social media and other digital technologies from corporations capitalise on users' desire for social, intellectual and cultural worth in order to exploit their labour and make them create monetary value. It is difficult to separate play and labour in digital media use:

> The exploitation of the Internet prosumer commodity is an expression for a stage

> of capitalism in which the boundaries between play and labour have become fuzzy and the exploitation of play labour has become a new principle. Exploitation tends to feel like fun and becomes part of free time. (Fuchs, 2013, p. 122)

Ross (2013) claims that this feeling of fun as a gratification is still a kind of self-exploitation. He identifies that even if the crowd cannot help making creative works, their willingness to produce content can be a form of self-exploitation: ‘job gratification comes at a heavy cost—longer hours in pursuit of the satisfying finish, price discounts in return for aesthetic recognition, self-exploitation in response to the gift of autonomy, and dispensability in exchange for flexibility’ (Ross, 2013, p. 21). From this point of view, even if a corporate platform fosters a new form of creativity and interaction it should not obscure the underlying new forms of exploitation, and new principles should be outlined when exploring exploitation in people’s participation in digital media.

Principle of exploitation in digital media usage.

For Fuchs, exploitation happens when one group controls property and forces others to produce goods or property for this group. Property control and force are two key points in examining exploitation (Fuchs, 2013, pp. 13-14). In Andrejevic’s (2013, p. 154) critique of digital labour, he introduces Hesmondhalgh’s (2010) discussion of the three principles of exploitation as an entry point:

> First, exploitation occurs when the material welfare of one class is causally dependent upon the material deprivation of another. The capitalist class in modern societies could not exist without the deprivations of the working classes. Second, that causal dependence depends in turn on the exclusion of workers from key productive resources, especially property. Third, the mechanism through which both these features (causal dependence and exclusion) operate is appropriation of the labour of the exploited. (Hesmondhalgh, 2010, p. 274)

Drawing from these principles, Andrejevic identifies that users in the digital era have to surrender their personal information to access the informational resources from commercial organisations. This is a form of material deprivation and enclosure, as it separates users from the infrastructure that supports their communicative activities. As a result, 'the structure of social relations wherein a small group controls the productive resources used by the many and allows economic advantages to accrue from this control' are reinforced (Andrejevic, 2013, p. 155).

Relating these principles back to the political economic study of Fuchs and Marx, we can conclude that the principle of exploitation in the digital era is the about the private ownership of collaborative digital product and the power that maintains this production process. First, users' activities generate content for vendors, which is a kind of surplus value production. When users are engaging in the use of 'free' digital platforms, platform vendors usually gather their user-generated content and personal information. This content and information can be appropriated as surplus value when turned into profit by vendors without paying wages to users. Another term, alienation, or estrangement, means one's labour becomes an external object outside him and a power confronting him. It is called alienation because it usually refers to one's own labour becoming independent, hostile and alien (Marx, 2012, p. 8). In a traditional factory, the worker's labour-power is separated from the worker's body, becoming an object in the capitalist's possession. In the digital era, while the vendors of a platform control the productive resources, users' activities on that platform can possibly be privatised and monetised by vendors. The deprivation of users' activities is a form of alienation in digital era. Thus, the first step to explore contemporary forms of exploitation should look at what kind of products are generated through users' activities, who owns them, and how they profit from them as a way of property control.

The further point to be made is that the asymmetric power that maintains this mechanism is another principle to identify exploitation. As Andrejevic points out, exploitation is not simply about a loss of monetary value but also a loss of control over one's productive and creative activity (Andrejevic, 2013, p. 154). Exploitation is always accompanied by power, which compels producers to offer their labour. Traditionally, power works in the form of physical or economic coercion (Elster, 1986, p. 83), which is usually embodied in contractual requirements that compel workers to work overtime or deprive them from ownership of product. This coercion still exists currently in the use of digital media, but is not simply a conspicuous force that requires workers to work in the assembly line. Power in the digital era exists not only in the form of physical violence or economic coercion, but can be managed through ideologies that urge people to donate labour for gratification, or through disciplinary power from code that standardises users' activities. Identifying how corporations enforce such coercion and why people accept it is another principle to examine exploitation in the use of digital technologies.

From these two principles, we can tease out the attributes of exploitation in the digital era and relate them to APIs by asking the following questions: What kind of product is generated when developers use APIs? Who owns these products and benefits from them? What surplus value is produced in the use of APIs? Do external developers lose their independence, or are ultimately coerced through some form of force? In short, individual choice and pleasure are not the measurement of exploitation from a political economic perspective—exploitation can only be examined in terms of social relations from the perspective of political economy. Chapter Four will examine the production of surplus value in the use of Open APIs, and the force that ensures this exploitation will be explored in Chapter Five.

Chapter 4: The Extraction of Surplus Value in the Use of Open APIs

This chapter explores how Open APIs exploit external developers and end users in terms of the extraction of surplus value. It first examines how vendors collect information from various resources without paying. For one thing, vendors are collecting the data and innovative ideas of external developers. For another, vendors are mining the personal information of the everyday users of applications or websites that are developed by external programmers. This chapter then discusses the relationship between collecting information and exploitation from a theoretical point of view, and points out that vendors are monetising so called ‘free’ APIs in three major ways: displaying targeted advertisements on external programs; promoting Ads APIs to external developers; and developing non-free complementary services which are based on APIs.

Collecting information without paying

Collecting information and innovative ideas from external developers.

One typical form of exploitation happens when developers’ use of Open APIs contributes to the improvement of APIs without the payment of wages. This contribution differs from those of the FLOSS community as APIs are proprietary and are used in the pursuit of profit. For example, in the case of Twitter's APIs, third-party users' work is systematically fed back into the underlying Twitter services (Bucher, 2013). An interviewee from Bucher's study explains this logic:

> Software companies have to develop a very small part of the software and they get millions of developers for free (almost) all around the world, creating new improvements to the original software. The companies can even incorporate those modifications into the software if they become important enough, and the cycle keeps going round! Basically they are taking, what is probably their biggest direct cost, programmers, and dilute that cost amongst a huge base of programmers that not only code for less but also provide the largest source of new ideas for their software. They are making sure they stay in the retail software game as it evolves. (Bucher, 2013)

Vendors do not simply provide Open APIs as a kind of service to developers, they also monitor a developer's usage, gathering data for the purposes of adjusting and enhancing their platforms. For example, Google also takes advantage of the information from third-party developers through their APIs:

> Google may share non-personally-identifiable information about you, including Web site URLs, site-specific statistics, and similar information collected by Google, with advertisers, business partners, sponsors and other third parties. In

> addition, you grant Google the right to access, index and cache your Websites, or any portion thereof, including by automated means including Web spiders or crawlers. (Gangadharan, 2009)

Google acknowledges that programmers’ innovative ways of using APIs help them improve their services (Chiu, 2015). While vendors keep updating their services based on the contributions of developers, developers themselves are excluded from the process of developing APIs, which remain vendor owned and controlled, reserving the right of any future modification to the APIs. In addition, developers are excluded from being paid for their contribution to APIs, and they cannot redistribute APIs to gain profit as they are proprietary and copyrighted. In this sense, external programmers donate their unpaid labour time to develop the services of vendors, this generate surplus value in the process of production.

Collecting personal information from end users.

Taking a step back, if we look at what services Open APIs provide, we can see most fundamental resources they offer are not produced by vendors but come from exploiting end users (not programmers but the everyday users of social media). For many social platforms such as Twitter and Facebook, end users’ information is collected by vendors for profit. These social platforms usually have a large user base because of their popularity, so APIs based on these social platforms are attractive for external developers who are interested in building applications based on social networks. Andrejevic (2013) points out these interactive services are mining users' information for the purposes of marketing:

> The entire app layer of interactive services, for example, provides as many new dimensions of computer monitoring as it does innovative conveniences. If you download an application for surfing or knitting, you have simultaneously joined a new demographic group for the purposes of target marketing. The more targeted or unique the service, the more detailed and unique the information about the user it provides. Even upgrades to the computer language (hypertext markup language, or HTML) that supports the web incorporate new capacities for online monitoring and tracking. (Andrejevic, 2013, p. 149)

End users submit their personal information for sharing online, join the social platform and treat it as an instrument for being connected. However, vendors see it another way: they treat this personal data as proprietary (Vis, 2013; Fuchs, 2013, p. 164-166; van Dijck, 2013), using it to attract programmers and advertisers by allowing access to data via their API. For example, the Facebook Games API provides game developers with a substantial user base, but this sometimes depends on exposing end users' personal information. After third-party game developers integrate with the Facebook Games API, it recommends these third-party games to specific end users of Facebook from their personal front page or sends notifications of advertisement to end users.

Sometimes the Games API provides monetisable methods for in-game purchasing through Facebook (Facebook Developers, 2014), thus stimulating the in-game consumption of end users. Another API—The Facebook Connect API—allows end users of Facebook to log onto external websites with their Facebook accounts. Thus, end users do not need to create a new account if the websites adopt Facebook Connect API—a Facebook account can be a passport for them all. However, this API also opens end users' data to external developers, and these data include profile information, friends and privacy preference. Moreover, some location-based APIs such as Facebook Open Graph provide external websites and applications with end users' geographic data, including information such as location-based check-in status of end users (Bodle, 2011).

In this regard, social platforms collect end users' personal information, and use Open APIs as a way for 'selling' it. Vendors make end users a commodity to attract external sites and applications, redirecting traffic to Facebook's services, and attract partners such as advertisers to gain more profit. Social platforms and APIs are together being utilised by vendors for the extraction of personal information from the end user.

The relationship between collecting information and exploitation.

For using the services of vendors, end users of social platforms and external developers of APIs have signed agreements that allow vendors to use their data. However, as discussed earlier in this chapter, arguments about end users' and programmers' choices overlook the generation of surplus value as the central attribute of exploitation: ownership of the final product and who profits from it. As we have examined, through APIs, external programmers' efforts and end users' personal information are going back to improve the services of vendors, which are totally owned by vendors. To some extent, their information becomes the private property of commercial entities and can be monetized. Third-party developers' labour becomes an external object outside developers, and generating value for vendors. This is a process of alienation and deprivation, and it is legalised by user agreements and TOS, which are defined by vendors.

No matter how much job gratification external programmers get from utilising Open APIs, no matter how happy end users feel in using social media, their information become entangled with vendors' services, and generates surplus value for commercial entities. Andrejevic (2013) points out:

> A critique of the exploitation of free labour, then, would highlight the very real ways in which control of new productive resources is concentrated in the hands of

> the few, allowing them to appropriate and profit from the activity of the many who must surrender their personal information to secure access to the productive and informational resources of the digital era. (p. 155)

This kind of exploitation is happening in the use of Open APIs. Vendors control the productive resources and use Open APIs as a way for commodifying end users and external developers. Some programmers have expressed their worry about the popularity of APIs: 'Companies offer web services to get free ideas, exploit free R&D, and discover promising talent […Companies] offer the APIs so people can build clever toys, the best of which the company will grab—thank you very much—and develop further on their own' (Linden, 2007). To some extent, end users and programmers are totally open to APIs vendors, but Open APIs maintain proprietary and close sourced.

These activities reinforce the social relations of capitalism on which exploitation relies, as Comor (2010) identifies: 'as long as private property, contracts and exchange values are dominant mediators of our political economy, disparities and exploitative relationships will remain largely unchallenged—unchallenged, at least, through the auspices of resumption' (2011, p.323). As long as APIs are the private property of commercial organisations, users' activities can never escape the exploitation of vendors if vendors intend to do so.

Monetization of Open APIs—'Open' as a Commercial Strategy

Another question remains: As most Open APIs are free to access and use, how can vendors profit from them to achieve the full circle of exploitation? Even if vendors ensure their domination over a product, they still need to 'sell' it as a commodity to complete the circle of exploitation. This section will focus on how vendors commodify developers and get the financial profit from Open APIs.

Theorists of political economy have identified the importance of financial profit in the study of exploitation. Fuchs (2013, pp. 65-66) identifies the central economic role of monetisation in the political economy of social media. He argues that the social benefits that emerge from platform use cannot cover the truth that vendors are exploiting users for profit. Money, as Fuchs makes clear, 'has a central importance in capitalism because it is a general equivalent of exchange: it is the only commodity that can be exchanged against all other commodities' (2013, pp. 63-64). Therefore, who actually gains profit in the form of money is key for a political economic examination of today's digital culture. In the case of Open APIs, vendors make their APIs 'open' to use 'free of charge' to enable collaboration, but in truth this 'collaboration' through social media monopolies and programmers is nonetheless about the pursuit of profit.

Unlike programmers who are part of a community which promotes the non-commercial use of software, programmers of corporations are serving stockholders and partners who show more interest in money. Some Open APIs are free of charge when they are initially developed, but once these APIs, owned by profit-seeking vendors as they are, are adopted by developers and accepted as an effective community standard, vendors could start harvesting developers in various ways for monetary value.

The exploitative of targeted advertising within APIs.

Advertising is one major way vendors used to monetise popular Open APIs, which perpetuates a form of exploitation through their use. The relationship between advertising and exploitation has been pointed out by Smythe (1981). He argues that media programs are a ‘free lunch’ used for producing audiences for advertisers. People are attracted by this ‘free lunch’ and their attention and time becomes a commodity, which is sold to advertisers. Based on this theory, Fuchs (2013) explores exploitation through the capital accumulation of social media. For him, new media corporations do not or hardly pay users for their information and user-generated content. Rather, their strategy is to use free access to services and platforms to attract users to produce content, and sold as a commodity to advertisers.

> No product is sold to the users, but the users are sold as a commodity to advertisers. The more users a platform has, the higher the advertising rates can be set. The productive labour time that capital exploits involves, on the one hand,

> the labour time of the paid employees and, on the other hand, all of the time that is spent online by the users. (Fuchs, 2013, p. 110)

Furthermore, targeted online advertisements as a new form of advertisements are revealing their significance. It is because they usually base on the collecting and analysing of audiences' information. According to Marx, the value that unpaid labour generates is surplus value, which constitutes exploitation (Marx, 1867, p. 376). In digital era, the unpaid activities and personal information of users are the new unpaid labour, which generates surplus value for platform vendors. Marx also point out the production of surplus value consists of absolute surplus value production and relative surplus value production. Surplus value produced by prolonging the working day is absolute surplus value, and those produced by increase efficiency and productivity is called relative surplus value (Marx, 1867, p. 431). Fuchs (2013) interprets targeted advertising as a form of relative surplus value production. Traditionally, all watchers see the same advertisements at the same time. But in targeted online advertising, advertisers present different ads to targeted users at the same time. Thus, more advertisements that fit the interest of audience are shown in the same period, so the efficiency of advertising is increased and more relative surplus value is generated (Fuchs, 2013, pp. 105-106).

The same kind of exploitation is happening in Open APIs. Initially, Google Maps API was free of charge, with no ads displayed in the result after developers integrated with API, but its TOS shows that it reserves the right to display advertisements in the future (Google Maps/Google Earth APIs TOS, 2013). Google Maps API did this in October of 2011:

> There are also a number of changes that relate to advertising and usage limits. To ensure that we can continue to offer the Maps APIs to developers for free, we now require that any new Maps API applications going forward display any advertising delivered in the maps imagery'. (Google Geo Developers Blog, 2011)

From then on, advertisements are binding to Google Maps API. Moreover, the Google Maps API's Terms of Service ensure the legality of displaying advertisements:

> (a) In Places Results. Google reserves the right to include advertising in the places results provided to you through the Maps API(s). By using the Maps API(s) to obtain places results, you agree to display such advertising in the form provided to you by Google.
>
> (b) In Maps Images. Google also reserves the right to include advertising in the maps images provided to you through the Maps API(s)…(Google Maps/Google Earth APIs TOS, 2013)

In addition, the advertisements usually target specific users based on their personal information. Google says that it 'collect[s] information to provide better services'. However, this collection of information also mines users' profiles and habits that emerge from data collected from the usage of Google's services: 'We may collect information about the services that you use and how you use them, like when you visit a website that uses our advertising services or you view and interact with our ads and content' (Google Privacy Policy, 2014).

To be clear, the 'users' that advertisements target are not the external developers who adopt APIs in their software, but the end users of software that is developed by external developers. For example, if a mobile app adopts the Google Maps API, then the mobile app end users are forced to see the advertisements when they are using the app which includes an embedded version of Google Maps through its use of Google's API. Open APIs retrieve their private information and give targeted advertising, which includes collecting users' actual location and delivering potential advertisements for nearby restaurant. When developers embed Google Maps in their apps via API, the end users of apps could see on-map advertisements regardless of the intent of the developers. For instance, end users can see a recommendation for Massage Envy Spa for recommendation when they are searching for Yerba Buena Center for Arts. (Google Maps Embed API, 2014):

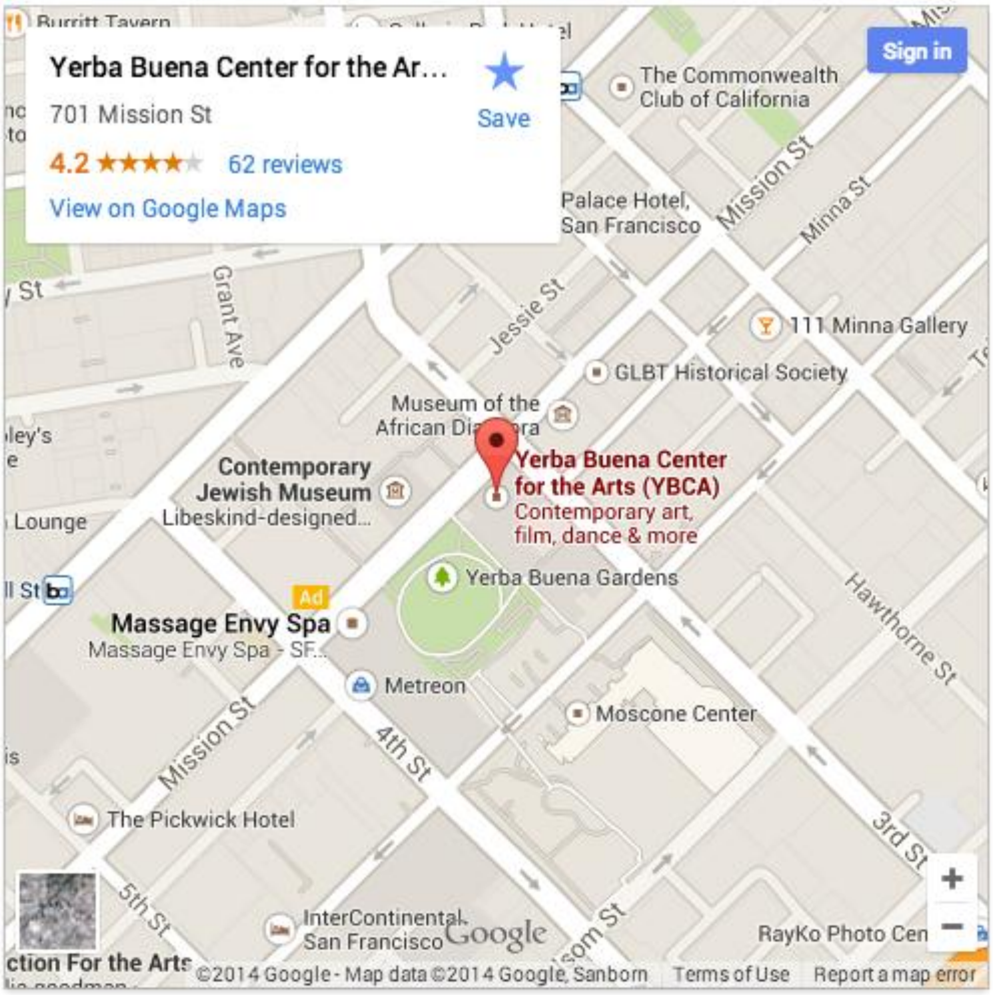


Figure 4.1: Advertisement of Google Maps Embed API

While the targeted on-map ads are displayed in developers' programs, developers themselves excluded from controlling or even noticing these ads: 'The ad format and the set of ads shown in any given map may change without notice' (Google Maps Embed API, 2014). As content that developers retrieve via APIs are controlled by vendors, delivering targeted ads together with necessary data is a typical strategy for vendors to gain profit. Both external developers and application users are being exploited as they are producing surplus value that is owned by vendors without paying.

This exploitation from advertisement is more complicated than what Fuchs and Smythe examined about social media, because this exploitation contains two steps. First, the direct users of vendors' services are external developers, and developers themselves are not being served ads. Developers only use APIs to retrieve functionality from underlying software, while the embedded ads are also being retrieved and displayed in their final software product. In the second step, the final users of external programmers' product are being served ads. In this sense, developers' programs, which might have plenty of visitors, are subject to vendors whose motivations are creating audiences and delivering ads. Developers' labour is being exploited as it helps vendors to profit from ads.

Throughout, the subjectivity of external developers becomes instrumental for vendors. Companies commercialise third-party programmes by firstly integrating their innovative ideas into its own services, and incorporating external programs and their final users into vendors' own advertising business. Via Open APIs, corporations use capital accumulation models based on the exploitation of the unpaid labour of external developers and end users. Open APIs are the 'free lunch' that uses programming opportunity to attractive external developers. When Open APIs are adopted, both external developers and end users become commodity and sold to advertisers for targeted advertising. This is how corporation profit from so-called free to use Open

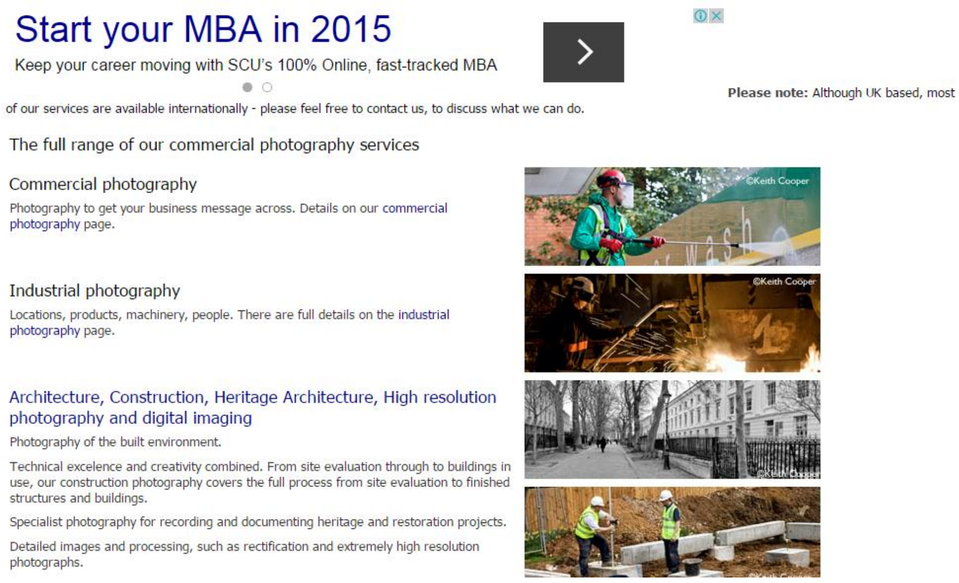
APIs and accumulate surplus value.

Profit from Ads APIs.

In addition to the delivery of targeted advertising, vendors also provide Ad APIs as a new way to collaborate with advertisers and external developers. As APIs can provide access to any underlying data, they can also provide a way for programmers to integrate with advertisements directly. Ads APIs are a kind of API that links advertisements to external applications, which offer an easy way for developers to display and manage ads on their websites or apps. They specify a way for programmers to embed advertisements in their products, so what programmers need to do is just add a string of code of Ads APIs in their source code of websites or apps. Then external developers can share the profit when vendors getting paid from advertisements. In Ads APIs, advertisements themselves are services that vendors provide for external developers. This gives external developers the opportunity to make fast money from advertisements without contacting advertisers by themselves. However, end users of developers' websites and apps and being exploited by ubiquitous advertisements from Ads APIs.

Google's AdSense API is a typical instance of these kinds of APIs. Google's AdSense program lets external developers (which Google describes as 'network members') make money from their sites: 'Show ads that relate to the content and users of your website' (Google Ads Adsense, 2014). If one site or application draws traffic, it can make money from Google AdSense with little effort. Turow (2012) examines how Google AdSense works on monitoring external developers' websites: 'a network of websites that agreed to allow Google to serve "contextual" text ads on them based on Google's analysis of the content of the sites' Web pages that the visitor was reading' (Turow, 2012, , p. 75). To apply Google AdSense to applications is easier than dealing with advertisers directly if developers need profit from ads. Developers only need to register an account and make ads space available on their websites or applications, then paste ad code and choose where they want the ads to appear. Advertisers will bid to show ads, while the highest paying ad will show in the ads space. In this way, AdSense will show relevant ads in the designated space targeted to external developers' sites and audience. Both external developers and Google can share the money from advertisers (Google Adsense, 2014). External users can also deploy the AdSense Management API to get reports on AdSense earnings and manage AdSense inventory (Google Developers, 2015).

Keith Cooper is a commercial photographer who built his website *Northlight Images* initially intending to promote his business. This website displays Cooper's works and articles about photography to attract visitors. As the website became increasingly popular, Cooper started looking for advertising to earn profit. However, most advertisements are too much for him to handle: 'I looked into it, and most of the advertisers were only suitable for larger businesses, where you can have a dedicated employee managing the advertising. As a small UK business with a global site audience, I needed an efficient solution with ads that were relevant to visitors in their geographical area' (Google AdSense Case Study, 2013). According to him, traditional advertising is time intensive and has diminishing returns for small businesses like *Northlight Images*. However, Google AdSense offers a high level of flexibility and helps him to collect income. The ads appearing on his websites are very relevant, and he does not need to worry about updating them (Google AdSense Case Study, 2013).



Start your MBA in 2015

Keep your career moving with SCU's 100% Online, fast-tracked MBA

Please note: Although UK based, most of our services are available internationally - please feel free to contact us, to discuss what we can do.

The full range of our commercial photography services

Commercial photography

Photography to get your business message across. Details on our commercial photography page.

Industrial photography

Locations, products, machinery, people. There are full details on the industrial photography page.

Architecture, Construction, Heritage Architecture, High resolution photography and digital imaging

Photography of the built environment.

Technical excellence and creativity combined. From site evaluation through to buildings in use, our construction photography covers the full process from site evaluation to finished structures and buildings.

Specialist photography for recording and documenting heritage and restoration projects.

Detailed images and processing, such as rectification and extremely high resolution photographs.

Figure 4.2: AdSense on *Northlight Images*

As Figure 4.2 (Northlight Images, 2015) shows, the top area entitled 'Start your MBA in 2015' is the ads displayed by Google AdSense. Developers of this website only need to set a space for Google AdSense, and Google will decide what will be shown in the specified area. The ads vary based on visitors' personal information, which means the same place could advertise games if a visitor is a gamer.

The case of *Northlight Images* is one of the ‘successful stories’ on Google AdSense’s official site, and it helps Google demonstrate how great AdSense is while masking the underlying risk of adopting it. First, it uses the words ‘traditional advertising’, ‘time intensive’ and ‘diminishing returns’ to manifest how innovative, convenient and profitable Google AdSense is for developers seeking ads. Second, by introducing Keith’s comments on Google AdSense, Google intends to highlight how successful AdSense is in sending targeted advertisements, regardless of the fact that it is mining information. For example, Keith needed relevant ads for ‘visitors in their geographical area’, and his site has two different parts, which attract different types of visitors. Even so, Google AdSense still fit ‘particularly well’ in each of the pages and Keith ‘pronounces himself highly satisfied with the quality of the ads’ and ‘hasn’t ‘found this to be a problem’ (Google AdSense Case Study, 2013). These statements highlight the merit of Google AdSense while covering up the context of targeted advertising—Google is mining developers’ content and websites visitors’ personal information. These statements also convey a message that Google’s AdSense adapts particularly well to developers’ websites, so no supervision of advertisements is needed for developers. However, as Figure 4.2 shows, the ads on MBA class is not even relevant to photography. Visitors who are interested in an MBA might be distracted and click the link to the advertisers’ page. In this sense, Google AdSense is indeed doing ‘particularly well’ in sending target ads and attracting visitors, but may not be doing a good job in helping websites gain visitors. Moreover, using the discourse like ‘Northlight Images, was initially intended solely to promote his business’, ‘As a small UK business’, Google is promoting the feasibility of its AdSense. Google wants to inform developers that any website—whether or not it is commercial, no matter how big the scale is—can use Google Adsense. The claim ‘income from AdSense lets him develop his photography business in the way that he wants to’ tells developers that profit from AdSense can free them from financial strain. In this way, Google intends to attract any developers who need money to use Google AdSense, so as to incorporate external developers into Google’s business.

Again, the Ads APIs is another way vendors monitor information from their partners and display targeted advertisements, and this time it looks like a win-win game for external developers as they are sharing the profit from ads with vendors. However, end users become commodities and are sold to advertisers. In addition, Google is becoming dominant in online advertising as increasingly more developers adopt Ads APIs as an easy way for earning profit. According to Google’s Annual Report in 2013 (United States Securities and Exchange Commission, 2014, p. 27), the total advertisement revenue of Google reached $US 50 billion in 2013, increasing about 20% each year since 2011. More than 90% of the total revenue of Google came from advertisements in 2013. Websites of Google itself received more than $37 billion in

2013, accounting for 67.4% of the total advertising revenue. In addition, Google Network Members' websites (websites of external developers) accounted for 23.6%, which was about $13 billion in 2013 (United States Securities and Exchange Commission, 2014, p. 27-28). As increasing external developers adopt Ads APIs for profit, more end users will be exploited by targeted advertisements.

Profit from services.

In addition to the advertisements, vendors still have various ways to profit from Open APIs. They can set request limitations and charge for extra usage as a way to gain profit. From October 2011, Google Maps API amended its licenses: any application that generates more than 2500 requests per day from Google maps incurs charges (Google Developers, 2014). In addition, commercial applications that generated profit have to pay for the Google Maps API once they adopt it—even if they do not exceed stated limitations (Google Maps API FAQ, 2015). From then on, developers who intended to develop a commercial application, along with those who exceeded set limitations, have had to pay for access through the API.

Moreover, the sale of complementary hardware or software is another typical strategy of vendors to gain profit with their so-called 'free of charge' software. As mentioned, Open APIs disclose only part of the function of original software, while keeping the remaining secret. Thus, vendors can develop derivative APIs based on the same underlying software for purposes of sale. Google Maps for Business has more functionalities than ‘free’ Google Maps API as it charges. Both of these APIs are based on the informational resources of the same services: Google Maps. However, the advantage of using Google Maps for Business includes, but is not limited to, a larger number of requests permitted, advertisement control, technical support, and a higher resolution (Google Maps for Business, 2014). In this sense, the original Google Maps API becomes a sort of light version as it lacks features included in the paid ‘Business’ version. If developers intend to gain more functionality or get rid of advertisements, they pay for the ‘premium’ version, which in this case is Google Maps for Business, to enjoy better performance than through the ‘free’ version. In what sense can we call these APIs ‘free’, as they neither give users freedom to modify and distribute, nor are they truly free of charge?

Although Open APIs may look like they are free and open, vendors still receive profit from them because of their proprietary nature. Via advertisement and other methods for monetising content and access, vendors get significant income without paying developers and end users who keep donating their free labour for ‘participation’, thus surplus value is produced by means of this new ‘collaborative’ technology. However, the generation of surplus value is only one principle when identifying exploitation. Power, which maintains this production of surplus value, is also a significant principle when doing political economic critique.

Chapter 5: Force of Open APIs

The previous chapter explored how vendors profit via Open APIs by exploiting developers and end users of programs. This chapter focuses on the power that maintains this exploitation. It answers these questions: from where does the asymmetric power that ensures the private ownership of collaborative digital product in API use emerge? Specifically, this chapter identifies two forms of power maintained by vendors through Open APIs. The first form of power comes from the legal contract. Vendors control vital programming resources, use contracts and copyright law to limit redistribution of APIs and derivative works to ensure their domination over programmers. The second form of power comes from an ideology that persuades developers to cooperate with Open APIs rather than use other collaborative methods. Discursive practices overexaggerate the openness of Open APIs, making developers willing to join a vendor-dependent infrastructure for short-term advantage without realising the risk. As a result, programming with Open APIs creates a new form of sociality integrally tied to commercial infrastructure.

The Force of Law

Vendors always publicise the openness of APIs, but ‘open’ in programming is not just about the right to view and use. More explicitly, ‘open’ is also about modifying and distributing, as we examined in Chapter Two. The fact is: Open APIs are not open at all.

They are not built for collaboration and development throughout the IT industry, but deployed by profit-seeking companies who try to reduce transaction costs, incorporate developers, and gain a long-term advantage. To achieve this and ensure the central position of their commercial services, vendors protect their APIs by controlling the programming process and redistribution. In addition to the power of protocol, vendors resort to Terms of Service (TOS) agreements and other copyright laws to ensure this domination. Vendors utilise this power in two ways. On the one hand, vendors use their own TOS to protect the data of underlying software. On the other hand, vendors use copyright law to protect the code of interface and try to make their own interface an industrial standard for any collaboration.

Deploying TOS to protect the benefit of underlying software.

In the last few years, the TOS of some APIs are increasingly showing corporations' intention of limiting external developers in programming. For example, the *Guardian* API explicitly uses its terms to stabilise its central position among the developers who using their API: 'You are allowed 5,000 API queries per day and 2 queries per second by default. You may store *Guardian* content for no longer than 24 hours' (The Guardian Open Platform, 2014a). Similar terms like this limit the development of external programmers' applications and ensure the market domination of original *Guardian* services. These terms, as The *Guardian* explains, aim to 'Protect us and our partners from unwanted behaviour' (The Guardian Open Platform, 2014b), but the ‘unwanted behaviour’ could be creating competitive services which can threaten vendors’ dominant position.

Developers need to obey the TOS released by vendors, otherwise will they lose their access or even get into legal trouble. It is acceptable for developers to sign an agreement with vendors, but the central issue about the agreement is: developers need to strictly follow vendors’ preferences as vendors reserve the right to change their TOS when they want (Google developers, 2013; The Guardian, 2009; Flicker, 2014). Moreover, the descriptions of some terms are vague:

> For uses of Flickr APIs over a certain rate or for certain types of commercial applications, Flickr reserves the right to charge fees for future use of or access to the Flickr APIs. (Flicker, 2014)

While ‘certain rate’ and ‘certain types’ remain undefined, vendors reserve the right of final decision and interpretation—they become the only decision-makers, legislators and executors. In this situation, the only ‘right’ external developers who adopt Open APIs have is the right to opt out. Thus, external developers become subordination because of the legislation that defined by vendors. The legislation, in turn, reinforces the domination of APIs and intensifies the exploitation.

As vendors are pursuing profit, they show intolerance toward any kind of redistribution or uncontrolled access that might threaten their ‘openness’, and this intolerance is revealed in their detailed clauses. For instance, TOS of Google Maps

API forbid almost any form of uncontrolled access and redistribution (Google Maps/Google Earth APIs TOS, 2014). It first forbids any modification or detachment of its services:

> [10.1.1.] (c) No Reverse Engineering. You must not attempt to reverse engineer or decompile the Services or any component, unless this is expressly permitted by applicable law.
>
> (d) Substitute Service. You must not attempt to create a substitute or similar service through use of or access to the Services
>
> (e) No Modification of Search Results. You must not modify, reorder, augment or manipulate search results (such as those provided in the Google Places API) in any way unless you explicitly notify the end user of your actions.
>
> (f) No Modification of Links. You must not modify, replace, obscure, or otherwise hinder the functioning of links to Google or third-party websites provided in the Content. For the avoidance of doubt, titles for place results must link to the applicable URL provided in the result, unless the title is intended to be selected only for purposes of navigation by an end user accessing your Maps API Implementation from a device with appropriately space-constraining user interface options. In these cases, the title linking to the Google-provided URL must be displayed as the top and primary link on the subsequent landing page or user interface component […]. (Google Maps/Google Earth APIs TOS, 2014)

These clauses reveal the fact that Google Maps API and the underlying data it retrieves can only be displayed in external programs as an entirety. External users are not permitted to modify software, nor are they allowed to decompile any component to fit their need. Ironically, the restrictions on modification do not function rigorously if vendors choose not to enforce them. Google Maps was released in February 2005 as an application. At that moment, it only aimed to provide geo-location services to web users instead of letting developers program with it. A Google Maps API also existed at this point, but it was a private one under development—not an open one for external developers to integrate with until June 2005. However, once familiar with the functions of Google Maps, developers were keen to hack its API and develop applications based on it. Before the publcation of the official Google Maps API in June 2005, many programmers reverse engineered it and integrated map data into their own applications without Google's permission. However, Google did not stop these practices because it found that the developer community's reverse engineering of Google services had led to 'great things' (Chiu, 2015). The innovative ways that developers used APIs inspired vendors to improve it, which led to the creation of official Google Maps API. Although Google Maps API was a private API at that time and reverse engineering is a violation of TOS, Google allowed this hacking and modification as it is beneficial for companies.

Second, Google Maps TOS 10.1.1.(h), 10.1.3 and 10.2 (a) explicitly forbid external use and storage of underlying content from Google Maps except via its APIs, which limits the redistribution of derivative programs of Google Maps:

> 10.1.1. (h) No Use of Content without a Google Map. You must not use or display the Content without a corresponding Google map, unless you are explicitly permitted to do so in the Maps APIs Documentation […]
>
> 10.1.3 Restrictions against Copying or Data Export.
>
> (a) No Unauthorized [sic] Copying, Modification, Creation of Derivative Works, or Display of the Content. You must not copy, translate, modify, or create a derivative work (including creating or contributing to a database) of, or publicly display any Content or any part thereof except as explicitly permitted under these Terms. […]
>
> (b) No Pre-Fetching, Caching, or Storage of Content. You must not pre-fetch, cache, or store any Content, […] For example, you must not use the Content to create an independent database of "places" or other local listings information.
>
> (c) No Mass Downloads or Bulk Feeds of Content. You must not use the Service in a manner that gives you or any other person access to mass downloads or bulk feeds of any Content, including but not limited to numerical latitude or longitude coordinates, imagery, visible map data, or places data (including business listings). For example, you are not permitted to offer a batch geocoding service that uses Content contained in the Maps API(s)'. (Google Maps/Google Earth APIs TOS,

2014)

As these terms show, although the data from Google Maps are available for public use via APIs, Google still claims it is proprietary and maintains control on these data. Thus, it is not ‘open’ compared with FLOSS. To be specific, these clauses mean the only thing external developers can do with Google Maps is display the maps they retrieved from APIs, and developers do not have the right to separate any data from maps except explicitly permitted. For example, once an external developer uses Google Maps API to retrieve Google Maps, an interactive map will be displayed in the chosen position of this application. As a result, visitors will have a better experience when using this application as they have an interactive map to view. However, the data in this map including ‘coordinates, imagery, visible map data, or places data’ are not permitted to be stored or fabricated—vendors do not give away the proprietary of these data. More importantly, these terms also mean any usage of Google Maps’ data should go through Google Maps API itself rather than from other external developers who adopt API, which limit redistribution. The content from maps cannot be used ‘without a corresponding Google Maps’, and ‘Modification, Creation of Derivative Works’ of content are not permitted according to TOS. For example, if an open source software integrates with Google Maps API, then it is no longer ‘open source’ because it contains an API which cannot be open sourced according to TOS of Google. Other developers who want to program with an application can see the data on the embedded Google Maps of this application, but they cannot use them in their own programs. Any derivative works of this software can only use its original open source component and try to go through Google Maps API for the remaining part—the component from Google Maps API can only be integrated through Google directly rather than through third-party developers. This limits the sharing and developing between different external developers, and ensures vendors' dominance over developers. A user of APIs points out Open APIs are not open enough as the licenses limit the redistribution:

> The point about Free Software is not that you can write software to work in somebody else's walled garden, but that you can modify it and distribute the results under the same or compatible licenses. This is true just as much for cloud computing as for desktops and other devices. Being able to use only the vendor's instance of software is not nearly open or Free enough. (Mokurai, 2012)

According to Google Maps API TOS, any derivative works have to ask Google for permission and get an API key, thus programmers cannot collaboratively develop derivative works but bond to Google Maps in the redistribution process. These restrictions help vendors maintain any external usage of their own data to be monitored and controlled, which empower vendors in programming community.

Third, specific types of third-party applications are strictly prohibited to use Google Maps APIs:

‘10.2 Restrictions on the Types of Applications that You are Permitted to Build with the Maps API(s). Except as explicitly permitted in Section 8 (Licenses from Google to You) or the Maps APIs Documentation, you must not (nor may you permit anyone else to) do any of the following:

(a) No “Wrapping.” You must not create or offer a “wrapper” for the Service. For example, you are not permitted to: (i) use or provide any part of the Service or Content (such as map imagery, geocoding, directions, places, or terrain data) in an API that you offer to others; or (ii) create a Maps API Implementation that reimplements or duplicates Google Maps/Google Earth.[…]

(b) No Business, Residential, or Telephone Listings Services […]

(c) No Creation or Augmentation of an Advertising Product. You must not use any business listings Content to create or augment an advertising product.

(d) No Navigation, Autonomous Vehicle Control, or Enterprise Applications. [...](Google Maps/Google Earth APIs TOS, 2014)

All of these forbidden applications including Maps APIs, product advertisments and navigation services share a common feature: Google is currently running similar services and all these types of applications can be threatening to Google’s profit. That might be the major reason why Google considers the development of these applications as an infringement of TOS. In short, vendors ensure the closure of underlying data by deploying licenses that regulate developers’ activities and limit redistribution. This is the first way that vendors implement the power from legalisation. In the second way, vendors also use copyright law to protect the code of their interfaces which specify how two software component integrate.

The deployment of copyright law to achieve the monopoly of interface.

While the underlying data are closed according to the TOS, the interface itself is also closed by vendors. Despite the underlying data, an interface itself defines a specification for integrating with underlying data, and this specification can gets its popularity and finally become an industrial programming standard for developers. Thus, vendors sometimes limit the redistribution of the interfaces solely in order to maintain their domination on the specification for integration. Practically, even if the source code of an API is visible and usable for developers, vendors sometimes claim its copyright and restrict any program from being developed based on its code. Once the intellectual property of an interface is infringed, there could be a conflict. The lawsuit between Oracle and Google from 2010 is a typical example.

The Java language is a common programming language that being broadly used among developers. The Java API (also known as class libraries) is a tool that allows programmers to retrieve pre-written code of Java language rather than write their own code from scratch (Oracle v. Google, 2014, pp.7-11). If programmers want to give specific commands to computers, they do not need to write every line of new code if they deploy Java API. For example, if a programmer needs a step in his program to return the greater of two numbers, he does not need to write a new string of code for that function. Rather, he can call upon the corresponding *java.lang.Math.max* (one Java API package) to use ready-made functions from Java. The code of API to achieve this step can be very simple: 'int a = java.lang.Math.max (x,y)'. After deploying this line of code, the computer would find the 'Max' method under the 'Math' class in the 'java language' package; the word 'int' means an integer is returned by the method of 'Max'; the words 'x' and 'y' are input numbers, and the bigger one will be returned and set as the value of 'a'; (Oracle v. Google, 2012, pp.10-13). This API specifies a way to retrieve the function of 'Max' method in Java language. As it shows, programmers can simply choose pre-written programs from Java APIs to retrieve ready-to-use codes of the Java language, which makes programming process efficient (Oracle v. Google, 2012, p.5). As the Java API simplifies the process of programming, it is prevalent among the programming community, and many programmers are accustomed to using Java API to relieve workload.

In 2010, Oracle incorporated Sun Microsystems (who developed Java) and acquired the copyright of Java. In 2005 when Google was developing its Android software platform for mobile devices, it used some elements of the Java API for reference, making Android easy for programmers to familiarise themselves with. The final Android platform includes 168 API packages, 37 of which correspond to the Java API (Oracle v. Google, 2014, p.10). After Oracle discovered this correspondence, it sued Google for infringing the copyright of their Java APIs.

Although 37 packages of Android system correspond to the Java API and share the same name and function, 97% of the underlying code is different. Google merely borrowed the name and functions of these APIs:

> As to the 37 packages at issue, Google believed Java application programmers would want to find the same 37 sets of functionalities in the new Android system callable by the same names as used in Java. Code already written in the Java language would, to this extent, run on Android and thus achieve a degree of interoperability'. (Oracle v. Google, 2012, p. 6)

The other 3% of Android code was the same as that of the Java APIs—they have identical lines of code that specify names, parameters and functionality (Oracle v. Google, 2012, p. 7). Google claims that it did not steal any data or services from Java but only borrowed the protocol—the way to retrieve data. Google did not copy the underlying implementations of Java but instead develop its own implementations of the 37 API packages. What Google 'copied' was the specification that helps developers to retrieve data from the Google's own implementations. However, Oracle still accused Google of infringing its copyright because they claim Google 'had replicated the structure, sequence and organization [sic] of the overall code for the 37 API packages' (Oracle v. Google, 2014, p. 2).

In this lawsuit, whether or not an interface solely is copyrightable is controversial. Usually at the level of interface, the structure of code and the specification do not have the protection of copyright law. This is not only because an API itself is meaningless without the underlying software, but also because it consists of a command structure which is necessary for interoperability: 'This command structure is a system or method of operation under Section 102(b) of the Copyright Act and, therefore, cannot be copyrighted. Duplication of the command structure is necessary for interoperability' (Oracle v. Google, 2014, p.4). Drawing from this spirit, in the year 2012, United States District Court for the Northern District of California concluded that Java API is not copyrightable as it is a general expression—'there is only one way to write' the names and declaration for the command (just like common parameters 'Math', 'Max'), and the 'merger doctrine bars anyone from claiming exclusive copyright ownership of that expression' (Oracle v. Google, 2014, p. 13). However, in 2014 United States Court of Appeals for the Federal Circuit reversed the judgement and acknowledged the copyright of API, which made Google pay billions of US dollars for compensation (Oracle v. Google, 2014, p. 69).

Google states this decision is disappointing, as it damages computer science and software development (Bort, 2014). Law professor Samuelson (2014) points out the appeals court should not conflate the code of APIs with the code of underlying software. Underlying software can be copyrighted, but program structures such as APIs are inherently functional and 'aimed at achieving technical goals of efficiency', that is why they should not be protect by copyright law (p. 3). Making interfaces copyrightable is detrimental to software development, because the design choices of subsequent programmers can be constrained by the interface designs embodied in earlier programs (Samuelson, 2014, p. 4). Indeed, only giant companies like Google can afford a two-year-long lawsuit and afford the penalty. The only thing small or middle scale companies can do is to accept these 'closed' interfaces and their unequal TOS.

Kelty (2008) points out the 'goal' of commercial organisations is not just creating competitive product, but also creating and maintaining a standard infrastructure of programming based on them, which was what the vendors of UNIX intended to do in the 1980s:

> In short, the challenge was not just the creation of competitive products but the creation of a standard infrastructure, dealing with the technical questions of availability, modifiability, and reusability of components, and the moral questions of the proper organization [sic] of competition and collaboration across diverse domains: engineers, academics, the computer industry, and the industries it computerized [sic]. (Kelty, 2008, p. 152)

Could it also be the vendors of Open API trying to achieve? Corresponding to the API lawsuit, it seems that in the level of interface, power from vendors are still implementing in maintaining their domination on the interface rather than merely underlying software. The constraint on the modification and redistribution of APIs shows vendors' intention of building a standard infrastructure on their software product. Indeed, Google can choose not to collaborate with the dominating Java API. However, if Google starts building a new API for programming from scratch, how much effort will it take? How many external users are likely to learn a new API and new language to establish their programme?

APIs were built for achieving interoperability, but making it copyrightable excluded any developers from freely building programs on it. Furthermore, when a proprietary API becomes a merger doctrine for software integration, every developer who does not want to be excluded has to follow the rule of vendors. APIs, in this sense, are making it easier for vendors to control collaboration in software industry. APIs, to some extent, are rendered even more closed, and they can never be as 'open' as FLOSS. In what sense can we call it open? If the ultimate goal of vendors is not making their services open for collaboration but for achieving monopoly, then, the 'open' of APIs could be an ideology that exaggerate interoperability and convince developers to join a vendor dependent infrastructure.

The Power of Ideology in Open APIs

In traditional productive activity, capitalists force workers to work in the factory to earn low wages for their survival. Power in this era is in the form of economic coercion or physical force, which is essential for exploitation. In the usage of Open API, vendors do not force developers to program with their Open APIs—Open APIs are just optional services and they are not necessary components for developers' programs. Another kind of force, which is subtle but functions significantly in maintaining exploitation, comes from the ideological manipulation by vendors. Ideology is usually a required component for the maintenance of domination and exploitation, and is naturalised by the use of discourse for specific goals by specific organizations. Andrejevic (2013, p.154) states that force in exploitation can refer to commercial organisations reproducing the forms of scarcity that compel freely given submission. Presently capitalists do not force workers to work in an assembly line, but control vital resources for production and use this advantage to compel freely given submission of labours.

In today's context, commercial organisations are those who control vital resources for programming in digital era, and they convey an ideology which tells developers that Open APIs are 'open enough' and are the best choice for collaboration, urging developers to use Open APIs without enough consideration. Fuchs identifies the goal of ideology and its relationship with exploitation:

> Ideology aims at instilling the belief in the system of capital and commodities into humans' subjectivity. The goal is that human thoughts and actions do not go beyond capitalism, do not question and revolt against this system and thereby play the role of instruments for the perpetuation of capitalism. It is of course an important question to what extent ideology is always successful and to what degree it is questioned and resisted, but the crucial aspect about ideology is that it encompasses strategies and attempts to make human subjects instrumental in the reproduction of domination and exploitation. (Fuchs, 2013, p. 22)

Ideology maintains domination and exploitation by claiming the particular social structure is natural. Thus, using ideology critique to denaturalise these false consciousness can be a complementary necessity for political economy study of digital media (Fuchs, 2013, p. 17). This section examines the ideology of Open APIs conveyed by vendors, points out that vendors are attempting to naturalise the 'openness' of their proprietary APIs and persuade external developers to use APIs without question their validity.

An ideology that misleads developers.

For external developers who are interested in one API, they are likely to go to the official websites of the API and examine what it is and what it can do. However, for most APIs that are provided by commercial organisations, official websites do not have a clear introduction about what their APIs are and do. Rather, they constantly point out the advantage developers can get when adopt their APIs, and sometimes they just post links to tutorials that tell developers how to use the API.

For example, on the frontpage of Google Maps API, developers can see Google Maps are displayed in three screens and followed by a sentence ‘Google Maps, available on every screen’ (Google Maps APIs, 2015).

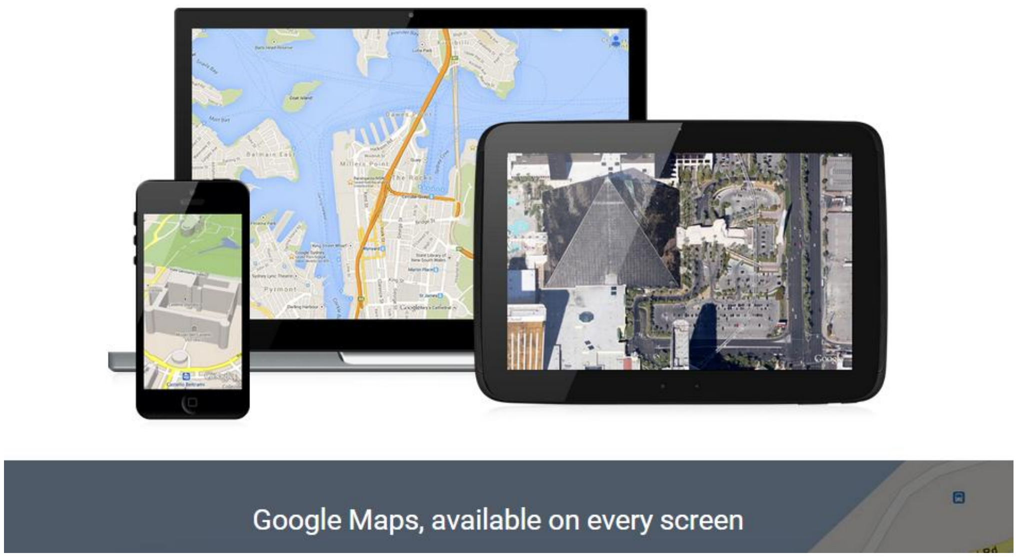


Figure 5.1: Frontpage of the Google Maps API

These three screens represent three platforms, IOS, PC and Android. Below these pictures, there are words listed with the title ‘IOS’, ‘Web’, ‘Android’ and ‘Everywhere’. This information that vendors give to developers is apparently showing the ubiquity of Google Maps API and lets developers choose the platform they intend to use. There is only one sentence that ‘describes’ a particular platform under each title:

> iOS: 55 million downloads can't be wrong. Bring the popularity of Google Maps to your app with the Google Maps SDK for iOS;
>
> Android: Bring more accurate, comprehensive and easy-to-use maps to your mobile app with the Google Maps Android API.
>
> Web: Deliver a seamless browser experience with the Google Maps JavaScript API v3 and kill two birds with one HTML5-powered stone.
>
> Everywhere: It's a bird. It's a plane. It's Google Maps, now available in the dashboard of your car, in your gaming console, and with you while you work out. (Google Maps APIs, 2015)

These descriptions are not telling developers what these APIs are, but are showing the ubiquity of their APIs. It seems that Google Maps APIs are popular among the programming community, and programmers have to use it or they will get left behind. Rather than introductions, they are more likely to be advertisements that promote Google Maps. In addition to the information above, the remainder of this page lists two titles: 'Design a map to call your own' and 'When only accurate and comprehensive maps will do'. Both of these titles are followed by some functionality of Google Maps API and descriptions about them. Similarly, these descriptions are all emphasising how amazing the functionalities are. For instance, the description of Google Place API is 'The Google Places API can be used to find places your users care about like restaurants, businesses and landmarks. The database of 95 million businesses is verified by owners and users on a daily basis'. In short, from this website external developers cannot understand what, explicitly, Google Maps API is, they only know Google Maps sounds great for developers and it looks like everyone is using it.

If one developer is willing to use Google Maps API on his websites, he might click the link for the 'Web' version and look for details. As Google is using the Google Maps JavaScript API v3 for its web service now, developers will go to the official webpage of Google Maps JavaScript API. However, from the frontpage of Google Maps JavaScript API (Google Maps JavaScript API, 2015), we can only see a Google interactive digital map and a section on 'Product Features' which still stresses how good the API is. If a developer wants to continue, the next stage is 'get started', which goes to the part of tutorial teaching programmers how to program with Google Maps JavaScript API v3. As of yet, there is still no clear description of what the API is and how vendors can enforce control through it. This information reveals the vendors' intention: let developers know how good their APIs are, and encourage them to start to learn how to use them and adopt them for external applications as soon as possible. External developers will only realise the limitations of using API when applying an API key and see the TOS (which might be too long and tedious for some developers to read thorough). The same thing also happens on the websites of the Facebook Graph APIs, where there is only one short paragraph about what Graph API is: 'The

Graph API is the primary way to get data in and out of Facebook's platform. It's a low-level HTTP-based API that you can use to query data, post new stories, manage ads, upload photos and a variety of other tasks that an app might need to do' (Facebook Graph API, 2015). It conveys a simple message: the Graph API can let developers get data from Facebook, and an app might need that data. After this short introduction, it goes on to describe technical details about the programming language of Graph API and how to use it. Not until developers review Facebook's Platform Policy (Facebook Developers, 2015) can they realise limitations on modification and redistribution. Among the websites of APIs, all this information is about promoting openness and teaching developers how to use APIs, and there is little information about the limitation and closure except in the TOS.

It is fair enough to assume that vendors believe external developers know the basic background of programming and what exactly an API is, but the websites that introduce APIs can still lead to misunderstandings. Vendors just keep emphasising that they open their data to third-party developers and how fantastic this opportunity is, but there is no explicit description about the openness and limitation. On the one hand, the limitation and restriction that ensure the closure are depicted in the tedious TOS that some developers may not read completely before adopting APIs. On the other hand, the accessibility that APIs provide is emphasized by vendors, but the potential of monitoring and controlling are not outlined clearly. These vague descriptions about API can render confusion, which makes many programmers do not know what API refers to. Patterson observes that the term API is colloquially used to describe both the specification and underlying service in recent years (Patterson, 2015). As a result, some external developers tend to think that vendors open their services and data for programmers just like FLOSS did, without realising the fact that the services are closed source and there are interfaces serving as an agent between external applications and vendors' services. While the underlying software is always closed, even the interface itself is just partly 'open' as it is copyrighted and ultimately closed off to modification. However, the 'open' interface and substantial functionality it connects to might cover the truth of their closure. Indeed, external programmers can benefit a lot from APIs when they adopt them, as we examined in the beginning of Chapter Three. This convenience makes developers regularly use APIs without too much hesitation, as they believe APIs are not just an option for collaboration but the best and only way to advance their own application efficiently.

As Open APIs facilitate interaction between different applications, they are sometimes conceived of as a substitute for 'open source software' for some programmers, and some of them even believe that Open APIs are more open because of their flexibility and low transaction cost. Lyman, a programmer and software analyst wrote a blog that said open APIs are 'the new Open Source' (Lyman, 2012). Lyman suggests that open APIs as a kind of open services, are better than open source for integration: 'they (programmers) soon find that they are dealing much more with APIs than with source code. Both customers and providers indicate an initial interest in open source and source code, but they soon find the APIs to be the more appropriate point of interface and integration' (Lyman, 2012). Lyman believes that although proprietary software keep their underlying code secret, they offer Open APIs as convenient methods for integration so they are 'open enough'. Lyman and programmers who support him share a similar argument: the proprietary computing resources that vendors of API own are too massive for some external developers to handle. Providing access to the services via Open APIs instead of providing the entire source code might be better for external developers as it save costs.

In terms of interoperability, APIs might have done a great job, sometimes better than providing entire source code of software as we discussed in Chapter Two, which is why Lyman and other developers believe in it. However, the partly open 'source code' of these interfaces does not make them 'open source'. If this openness is not as 'open' as FLOSS, interoperability and fast integration can still lead to closure and exploitation. All these discursive practices are ideological; they claim adopting APIs for programming is the best mode of collaboration, while neglecting the closure of APIs and the risk of exploitation. The fact that many developers benefit from APIs in the short term even 'proves' such a claim about the collaboration of APIs, thus intensify the proliferation of this ideology.

Once developers are convinced by this ideology and join the infrastructure of vendors, they are under control that they are not even aware of. The Power of pre-defined code and legal documents are controlling developers and their applications or websites. The interaction and transaction between different programs which use APIs are becoming relied upon by vendors. Programmers are becoming increasingly dependent on the infrastructure controlled by a few commercial entities, too. They lose their independent way of programming while switching to the way that commercial entities prefer, just because their way is more convenient and entertaining. A commentator responded to Lyman's article (JimPlamondon, 2012):

> In the 1990s, as a Microsoft employee, I sold this very same argument on Microsoft's behalf: that the Windows API was 'open enough'. AMAZINGLY [sic], people bought this argument, back then. Every line of code that they wrote to Windows' API locked them more firmly into the Windows API and hence into Microsoft as the sole vendor of that API (failed 'zombie projects' such as WINE, WABI, and Bristol aside). Writing a line of code to the Windows API was like giving Microsoft a line of credit against your future earnings...but people did it, because it gave them a short-term time-to-market advantage. SURELY [sic], the industry hasn't forgotten Microsoft's utter dominance of the PC computing industry...has it?

For external developers, this trend of using APIs without enough consideration increases the potential of domination. For vendors of API, this ideology is helping them to build a vendor-dependent infrastructure that gains more profit in the long term. The next section will identify how ‘open’ becomes a marketing ideology from the perspective of vendors.

Openness as a marketing ideology.

Some programmers notice the exploitive nature of Open APIs and express their concerns. Duncan Geoff is a commenter and programmer who shows interest in the changing policy of the Google Maps API. He examined the Google Maps policy and believes that ‘The idea was to make Google’s online mapping service a de facto standard for digital maps on the Web and in mobile devices. Once Google had established a serious presence in the digital mapping marketplace, it would figure out a way to monetize [sic] the service’ (Duncan, 2012). From his point, vendors establish Open APIs only for ensuring their market domination and aim for more profit rather than considering facilitating collaboration.

With the development of digital technology and increasingly data mining, covering the data is no longer easy for vendors, especially for those who provide data as public services. Given the situation that ‘open’ their services are inevitable, vendors might be considering: why not make this ‘open’ more controllable? Deploying Open APIs can be a strategy for vendors to follow the trend of ‘open’ and become more competitive. Aitamurto and Lewis (2013) examine the APIs of news websites and point out why vendors provide APIs to their news content. For them, offering APIs to developers is just a better way to gain more profits:

> We made the push. We sold it internally. There was some apprehension; some wonder what you're talking about, 'Why are we going to give away the farm for free? This is our baby. We're giving away the content'. However, the point is 'The content is on the website. Anyone can come and scrape it. Anyone can capture a stream. The content is out there. It's free anyways'. Some developers realized [sic] open part of APIs was an unavoidable, thus they working on develop Open APIs for purposes of better R&D and thus for greater profits. (Aitamurto & Lewis, 2013)

Moreover, some vendors do not hesitate to express how they conceive the term 'open' and how they can manage an open system and lead the trend of 'open'. In Google's official blog, Product Manager Jonathan Rosenberg published the post in 2009 titled 'The Meaning of Open', which identifies 'open' as a kind of marketing strategy that works better than 'closed' strategies. Google believes open systems will win out because they 'lead to more innovation, value, and freedom of choice for consumers, and a vibrant, profitable, and competitive ecosystem for businesses' (Rosenberg, 2009). For him, the competitive advantage of open systems derives from understanding the fast-moving system and using that knowledge to generate better, more innovative products:

> The successful company in an open system is both a fast innovator and a thought

> leader; the brand value of thought leadership attracts customers and then fast innovation keeps them. This isn't easy—far from it—but fast companies have nothing to fear, and when they are successful they can generate great shareholder value. (Rosenberg, 2009)

An open system, for Google, is a way to generate shareholder value. And Google uses the customers of this open system for this purpose. Google claims they 'harness the intellect of the general population and spur businesses to compete, innovate, and win based on the merits of their products and not just the brilliance of their business tactics' (Google Official Blog, 2009).

For companies like Google, openness is a strategy for them to gain leadership and become more competitive. It looks like this strategy is also benefiting external developers, but the truth is the interest of external developers' and the development of whole software industry is just a bonus from vendors, not vendors' main goal. Once they conflict with the interest of vendors, there is no chance for negotiation. For example, many vendors intensify the restriction of their APIs to control external developers' activities, only to maximise vendors' own interest. These limitations are infringing upon external developers' benefits and eroding the innovation and development of software industry. Janet Wagner, a data journalist who focuses on APIs, points out how overly restrictive API policies are killing innovation:

> In recent years, there has been a growing trend of companies initially providing

> public APIs and an open platform, only to enact API restrictions or even close access to the platform later on. This pattern has become rather commonplace among social networking service providers. Facebook, Google, Twitter and LinkedIn have all made changes to their API terms of service in recent years, limiting access to their platforms in varying degrees. (Wagner, 2014)

As a result, ‘Innovative applications built with social networking APIs has [sic] been drastically reduced’ (Wagner, 2014). However, vendors seem to not care about this. Thor Mitchel, Product Manager of Google, in response to the commentary of Wagner on Google APIs increasing restriction, points out the fact directly: ‘An API is not a successful product until that adoption translates into value for the provider. If your use of an API is not generating value for the provider, then all it is generating is cost and you’re skating on very thin ice by relying on it’ (Mitchel, 2014, August 7). In this case, value, or profit for commercial organisations, is why companies open APIs for their software. To some extent, ‘openness’ is a bait which is provided in the name of encouraging innovation and openness. For vendors, they do not really care about open or not, they only care about profit. Just like Spivack says: ‘“All that matters is the money” (to quote Shark Tank). And the money says, closing the APIs is sub-optimal. You can make more money by opening them with a good monetization model’ (Spivack, 2012, August 16). While an increasing number of Closed Source APIs are developed and opened for public use, the increasing number of third-party users is drawn into a monetisation mode of commercial organisations.

Overestimating the collaboration facilitated by Open APIs advances an ideology that Open APIs are beneficial for both external developers and the whole industry, without taking into account how the interests of vendors are dominating and shaping programming practices. In this sense, ideology of openness in Open APIs affects the subjectivity of external developers, making them instrumental for advocating and improving vendors' services, and finally aggravates domination and exploitation.

Exploitation, Power and the Central Position of Vendors

The significant power of Open APIs cannot be attributed to technology itself, instead, vendors are the major factor of this. Vendors play the key role in facilitating the pre-defined power of code, legal contract and ideological manipulation. In fact, Open APIs alone do not create this power, neither can they themselves lead to exploitation. It is embedded into the power structures of contemporary society, where profit-seeking corporations are exploiting the potentials of these digital technologies for commercial purposes. As discussed, Open APIs contain Open Source APIs and Closed Source APIs, both of which have the potential to foster new technical innovation and creativity. Nevertheless, the blooming of Closed Source APIs and exploitation among their usage only reveals the social relations of capitalism.

As long as an API is proprietary, it cannot be as open as FLOSS where exploitation has little opportunity to happen. Developers volunteer to use APIs to get a short-term advantage: retrieve geographic information from Google Maps API, or gain a significant user base via Facebook or Twitter API. In effect, their own software become dependent on the API vendors, and the mode of programming is tied to developing commercial infrastructures. When programmers are increasingly fond of APIs, they are becoming more and more reliant upon the provision these Closed Source APIs rather than the actual source code of underlying software upon which they used to rely. The proprietary nature of it limits the possibility for API to be truly open.

Chesbrough (2003) points out how openness in digital technology is used to forge new pathways for commercialising innovations crafted within and beyond the boundaries of a firm. Today the openness of APIs is such an open innovation strategy, where a company integrates external ideas and information into its own business and commercialises third-party programmes (Chesbrough, 2003; Aitamurto, & Lewis, 2013). From a critical Marxist perspective, exploitation is ‘evil’ because it ‘involves force and domination in manifold ways and because it deprives workers of control that should be theirs’ (Andrejevic, 2013, p. 157). In this regard, Google’s motto ‘Don’t be evil’ might need to be reconsidered as its Closed Source APIs manifest their power and exploitive.

Conclusion: Danger of Vendor Dependent Structure based on API

Kelty (2013) has made an argument that ‘there is no free software’. For him, even if software is truly free, it has the potential to make a ‘heterogeneous ecology of small and medium sized nodes (single servers and small clusters of servers) to a handful of enormous data centres and server farms running virtualized [sic] versions of the Internet inside them.’ Kelty is warning us about the potential domination of free software, as they might lead to monopoly: giants control all the services and become a central server. The biggest of these, such as Google and Amazon, are all running ‘open source’ software, but the concept of ‘open source’ is making increasingly little sense the larger and more controlled these systems become. For companies, ‘they only do so at the pleasure of the sovereign, so to speak—the once beloved power to “fork” the software disappears in a world where centralized infrastructure is everything’ (Kelty, 2013). Kelty is warning us about the danger of centralised infrastructure, and the central point of this claim is cause for alarm: once a handful of software publishers gain popularity and become de facto standards, the self-organised network of the programming community will inevitably turn to centralised infrastructure—even if the standards are facilitated by free software. This centralised infrastructure deprives developers’ independence, and once it becomes commercialised, exploitation may happen. Birkinbine (2014) explicitly identifies how large software companies get involved in the development of popular FLOSS projects, change the ownership structure or make profitable business on these projects. In this sense, even the common based peer labour can be exploited as corporations are seeking to make commercial use of the contributions of developers.

The issue of Open APIs is more complicated as it is proprietary and closed in the first place and conceived as ‘open’ for programmers. Vendors use Open APIs to achieve market dominance and cultivate dependency, which establish an infrastructure that supports unequal social relations. The problem is not due to the technical feature of Open APIs, rather, APIs provide a truly innovative way for software collaboration. The central issue is whether APIs are ‘vendor-dependent’, or, whether capitalism is affecting technical implementation of APIs. If an API belongs to commercial organisations, vendors can use it to privatise and commercialise developers’ labour as they want:

> The privatization and commercialization [sic] of the Internet is a form of material deprivation and enclosure insofar as it separates users from the infrastructure that supports their communicative activities. It reinforces and reproduces the structure of social relations wherein a small group controls the productive resources used by the many and allows economic advantages to accrue from this control. The ownership class that includes the founders of Facebook, Google, Yahoo, and so on could not exist without capturing and controlling components of the productive

> infrastructure. (Andrejevic, 2013, p.155)

In the use of APIs, vendors gain economic advantages from developers' programming activities, and use different forms of power to maintain this unequal relationship. These closed source API reinforces the structure of social relations that intensifies exploitation and domination. As long as APIs are affected by capital, it could lead to domination and monopoly, which is detrimental to the development of computing.

Indeed, vendor-neutral software and vendor-neutral Open APIs do exist such as OpenStack APIs (OpenStack, 2014). Open source APIs could be a way out of the influence of capital as Gangadharan says:

> In the future, we believe that more open APIs will begin to be licensed using open source licenses. While the source code of the interface of an open API is always available, open sourcing an open API makes the source code of the API implementation available in addition to the source of its interface. In this case, users would be able to modify an API, or derive new APIs from an open API. (Gangadharan, 2009)

Vendor-neutral APIs like OpenStack API and open source computing are adopted by the computing industry today but their numbers and scale are too small compared to proprietary services. This is why there is so much confusion and misuse about the word 'Open API' and 'Open source'. Open API itself does not lead to ideological manipulation and domination. The trigger, one the one hand, is uncritical external developers who adopt APIs without full consideration. On the other hand, are profit-seeking vendors who have significant resource, keep promoting their business and alter the technical feature of APIs for pursuing profit. As most open APIs nowadays are proprietary, the development of programming is limited by capitalist structures of ownership and profit-seeking vendors.

To advance a democratic programming environment, it is crucial to build a real open infrastructure like FLOSS which is not subject to market logic. More importantly, it is crucial to let developers realise the essence of the openness of FLOSS and its significance for software development. Even if there will be more 'open' technologies for software collaboration, corporations can still make them closed and find a way to monetise them. Only when developers fully understand the principles of FLOSS and develop a critical view on the openness of software can they make a right decision before adopting new software tools. In this way, developers can avoid being affected by ideologies, and protect their peer labour from unwanted commercial use.

References

AA. (2014). The AA Route Planner. Retrieved September 25, 2014 from http://www.theaa.com/route-planner/index.jsp

Aitamurto, T., & Lewis, S. C. (2013). Open innovation in digital journalism: Examining the impact of Open APIs at four news organizations. *New media & Society*, 15(2), 314-331.

Albanesius, C. (2013, March 14). Online Petitions Protest Google Reader Shutdown. [Web Log]. Retrieved from http://www.pcmag.com/article2/0,2817,2416602,00.asp

Andrejevic, M. (2013). Estranged Free Labor. In Scholz, T. *Digital Labor, The Internet as Playground and Factory*, pp.149-164. UK: Taylor & Francis.

API Evangelist.(2012). History of APIs. Retrieved December 20, 2012 from http://apievangelist.com/2012/12/20/history-of-apis/

Authorize. Net. (2013). *SOAP API Documentation.* Retrieved September 27, 2013 from http://www.authorize.net/support/CIM_SOAP_guide.pdf

Autonavi LBS Open Platform. (2014a). Location Tag. Retrieved September 16, 2014 from http://lbs.amap.com/api/lightmap/guide-2/point/

Autonavi LBS Open Platform. (2014b). Maps API tools. Retrieved September 16, 2014 from http://lbs.amap.com/console/show/tools

Awsdocumentation.(2006). *Amazon Simple Storage Service Developer Guide.* Retrieved from Awsdocumentation: http://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html.

Bauwens, M. (2012). Thesis on Digital Labor in an Emerging p2p Economy. *Digital Labor: The Internet as Playground and Factory*, 207.

Berners-Lee, T., Hendler, J. and Lassila, O. (2001). "The Semantic Web". *Scientific American*, 284(5). pp 34-43.

Berry, D. M. (2011). *The philosophy of software.* London, UK: Palgrave Macmillan.

Benkler, Y. (2006). *The wealth of networks: How social production transforms markets and freedom*. New Haven, CT: Yale University Press

Birkinbine, B. J. (2014). *Incorporating the Commons: Towards a Political Economy of Corporate Involvement in Free and Open Source Software* (Doctoral dissertation), University of Oregon. Eugene, US.

Bodle, R. (2011). Regimes of sharing: Open APIs, interoperability, and Facebook. *Information, Communication & Society,* 14(3), 320-337.

Bort, J. (2014). Google: *This Oracle Lawsuit Could Damage The Whole Software Industry*. Business Insider Australia. Retrieved May 10, 2014 from http://www.businessinsider.com.au/google-oracle-lawsuit-is-damaging-2014-5

Bucher, T. (2013). Objects of intense feeling: The case of the Twitter APIs. *Computational Culture, 3*. Retrieved 2013 from http://computationalculture.net/article/objects-of-intense-feeling-the-case-of-the-twitter-api

Carrillo-Tripp, M., Shepherd, C. M., Borelli, I. A., Venkataraman, S., Lander, G., Natarajan, P., Johnson, J. E.,Brooks, S. L. & Reddy, V. S. (2009). VIPERdb2: an enhanced and web API enabled relational database for structural virology. *Nucleic acids research*, 37(suppl 1), D436-D442.

Castells, M. (2011). *The rise of the network society: The information age: Economy, society, and culture* (Vol. 1). Hoboken, NJ: John Wiley & Sons.

Chesbrough, H. (2003). *Open innovation: The new imperative for creating and profiting from technology.* Cambridge, MA: Harvard Business School Publishing.

Chiu, P. (2015). Update on the Autocomplete API. Retrieved July 24, 2015 from http://googlewebmastercentral.blogspot.com.au/2015/07/update-on-autocomplete-api.html

Chun, W. H. K. (2011). *Programmed visions: Software and memory*. Cambridge, MA:

Mit Press.

Comor, E. (2011). Contextualizing and Critiquing the Fantastic Prosumer: Power, Alienation and Hegemony. *Critical Sociology*, 37(3), 309-327

Creative Commons. (2014). *Open Definition 2.0 released.* Retrieved October 7, 2014 from http://creativecommons.org/tag/open-definition

Cubit, S. (2013). The Political Economy of Cosmopolis. In *Scholz, T, Digital Labor, The Internet as Playground and Factory,* pp.58-68. UK: Taylor & Francis.

De Souza, C. R., &Redmiles, D. F. (2009). On the roles of APIs in the coordination of collaborative software development. *Computer Supported Cooperative Work*, 18(5-6), pp. 445-475.

Duncan, G. (2012). *Why are companies defecting from Google Maps?* Retrieved July 8, 2014 from http://www.digitaltrends.com/mobile/why-are-companies-defecting-from-google-maps/#!DsbY0

Duvander, A. (2014). *Most popular apis: At least one will surprise you.* Programmableweb. Retrieved July 14, 2014 from http://www.programmableweb.com/news/most-popular-apis-least-one-will-surprise-you/2014/01/23

Elster, J. (1986). *An Introduction to Karl Marx*. Cambridge, UK: Cambridge University

Press.

Facebook Developers. (2014). *Games Overview.* Retrieved December 8, 2014 from https://developers.facebook.com/docs/games/overview

Facebook Developers. (2015). *Platform Policy.* Retrieved March 25, 2015 from https://developers.facebook.com/policy/#properuse

Facebook Graph API. (2015). *Graph API Overview.* Retrieved June 5, 2015 from https://developers.facebook.com/docs/graph-api/overview/

Fairclough, N. (1995).*Critical discourse analysis: The critical study of language*. London, UK: Longman.

Fairclough, N. (2013).*Critical discourse analysis: The critical study of language.*2nd ed. New York, NY: Routledge.

Fisher, D. (2015, March 26). *Instagram API Bug Could Allow Malicious File Downloads.*[Web log]. Retrieved from https://threatpost.com/instagram-api-bug-could-allow-malicious-file-downloads/111784

Flicker. (2014). *Flickr APIs Terms of Use*. Retrieved September 12, 2014, from https://www.flickr.com/services/api/tos/

Floyd, I. R, Jones, M. C.Rathi, D&Twidale, M. B (2007). Web mash-ups and patchwork prototyping: User-driven technological innovation with web 2.0 and open source software. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference*, p. 86. IEEE.

Fox,Z.(2012, March 01).Foursquare Says Farewell to Google Maps, Joins OpenStreetMap Movement. [web log].Retrieved from http://mashable.com/2012/02/29/foursquare-openstreetmap/

Fuchs, C. (2013). *Social media: A critical introduction*. Los Angeles, CA: Sage.

Galloway, A. R. (2004). *Protocol: how control exists after decentralization*. Cambridge, MA: MIT Press.

Gangadharan, G.R. (2009). *Licensing of Open APIS.* Retrieved 2014, from http://timreview.ca/article/243

GNU Operating System. (2014a). *What is Copyleft*. Retrieved September 12, 2014 from https://www.gnu.org/copyleft/copyleft.en.html

GNU Operating System. (2014b). *FAQ*. Retrieved September 12, 2014 from http://www.gnu.org/licenses/gpl-faq.html#DoesFreeSoftwareMeanUsingTheGPL

GNU Operating System. (2015). *Licenses.* Retrieved September 12, 2014 from https://www.gnu.org/licenses/licenses.en.html

Google. (2013). *A second spring of cleaning.* Retrieved March 3, 2015 from http://googleblog.blogspot.com.au/2013/03/a-second-spring-of-cleaning.html

Google AdSense. (2014). *How it works.* Retrieved December 6, 2014 from https://www.google.com/adsense/start/how-it-works.html

Google Ads AdSense. (2014). *Overview.* Retrieved March 2, 2015 form http://www.google.com/adsense/start/

Google AdSense Case Study. (2013). *Google AdSense creates a profitable sideline for British photographer.* Retrieved June 30, 2015 from http://services.google.com/fh/files/misc/uk_northlight_adsense.pdf

Google Developers. (2014). *Google Maps API Licensing.*Retrieved December 25, 2014 from https://developers.google.com/maps/licensing

Google Developers. (2015). AdSense Management API. Retrieved June 26, 2015 from https://developers.google.com/adsense/management/index

Google Geo Developers Blog. (2011). Updates to the Google Maps/Google Earth APIs Terms of Service. [web log post]. Retrieved from http://googlegeodevelopers.blogspot.com.au/2011/04/updates-to-google-maps-ap igoogle-earth.html

Google Maps APIs. (2015). Retrieved June 30, 2015 from https://developers.google.com/maps/

Google Maps API FAQ. (2015). Retrieved March 15, 2015 from https://developers.google.com/maps/faq

Google Maps APIs Overview. (2015). Retrieved June 16, 2015 from https://www.google.com/work/mapsearth/products/mapsapi.html

Google Maps/Google Earth APIs TOS. (2013). Retrieved April 5, 2014 from https://developers.google.com/maps/terms?csw=1

Google Maps Embed API. (2014). Retrieved June 3, 2014from https://developers.google.com/maps/documentation/embed/guide

Google Maps JavaScript API. (2015). Retrieved March 5, 2015 from https://developers.google.com/maps/documentation/javascript/

Google Maps Tutorials. (2014). *Customizing Google Maps: Custom Markers.* Retrieved June 15, 2014 from https://developers.google.com/maps/tutorials/customizing/custom-markers

Google Privacy Policy. (2014). Retrieved December 12, 2014 from http://www.google.com/policies/privacy/

Graham, P. (2007). Political economy of communication: A critique. *Critical perspectives on International Business,* 3(3), pp. 226-245. Retrieved from QUT Digital Repository: http://eprints.qut.edu.au/19150/1/19150.pdf

Graham, P.W & Luke, A. (2011) Critical discourse analysis and political economy of communication: understanding the new corporate order. *Cultural Politics,* 7(1), pp.

103-132.

Gray, I.A. (2015). Build Your First Twitter App Using PHP in 8 Easy Steps. [web log post]. Retrieved March 1st from http://iag.me/socialmedia/build-your-first-twitter-app-using-php-in-8-easy-steps/

Hackus. (2011, December 30). Re:What's the point?. [web log comment]. Retrieved from http://news.slashdot.org/story/11/12/30/1317216/open-source-increasingly-replaced-by-open-apis

Hall, S., Critcher, C., Jefferson, T., Clarke, J., & Roberts, B. (1978).*Policing the crisis: Mugging, the state and law and order.* London, UK: Macmillan.

Hesmondhalgh,D. (2010). User-Generated Content, Free Labor and the Cultural Industries.*Ephemera* 10, no. 3/4 (2010): 267-248.

Jenkins, H. (2008). *Convergence Culture.* New York, NY: New York University Press.

JimPlamondon. (2012, Febraruy 16). Deja Vu all over again. [web log comment]. Retrieved from

http://www.linuxinsider.com/perl/board/mboard.pl?board=lnitalkback&thread=5891&id=5896&display=1#message_5896

Jones,B. (2012). The API Revolution. *Campaign,* 04/2012, p. 24.

Julien. (2013). A Google Reader Compatible API. [Web Log Post].Retrieved 19 Mar 2013 from http://blog.superfeedr.com/google-reader-compatible-api/

Kansa, E. C., & Wilde, E. (2008). Tourism, peer production, and location-based service design. *In Services Computing, 2008.SCC'08.IEEE International Conference*, Vol. 2, pp. 629-636. IEEE.

Kelty, C. M. (2008). *Two bits: The cultural significance of free software.* Durham, NC: Duke University Press.

Kelty,(2013). There is no Free Software. *Journal of Peer Production.* Issue #3*: The Critical Power of Free Software*. Retrieved 2014 from http://peerproduction.net/issues/issue-3-free-software-epistemics/debate/there-is-no-free-software/

Labourey, S and Harris,S. (2013). “Developer Freedomat Stake as Oracle Clings to Java API Copyrights In Google Fight”. *Techcrunch*, (3).

Lessig, L. (1999).*Code and Other Laws of Cyberspace*. New York, NY: Basic Books.

Lessig, L. (2006). *Code: Version 2.0.* New York, NY: Basic Books

Linden,G. (2007). *The truth about free APIs*. Retrieved March 2, 2014 from http://glinden.blogspot.com.au/2007/06/truth-about-free-apis.html

Lyman, J. (2012). *Open APIs are the New Open Source.* Retrieved January 10, 2014 from http://www.linuxinsider.com/story/74419.html

Lynd, S. (1965). The New Radicals and “Participatory Democracy”. *Dissent* 12 (3), pp. 324–333.

Marx, K. (1867). *Capital. Volume I.* London, UK: Penguin.

Marx, K. (2012). *Economic and philosophic manuscripts of 1844.* North Chelmsford, MA: Courier Corporation.

Mayberry, M. (2012). WinRT Revealed. New York, NY: Apress.

MBR1. (2012, February 14). *Open APIs can't be trusted the way Free Software can.* [web log comment]. Retrieved from http://www.linuxinsider.com/perl/board/mboard.pl?board=lnitalkback&thread=5891&id=5893&display=1#message_5893

Meehan, E.R., Mosco, V., &Wasko, J. (1993). Rethinking political economy: Continuity and change. *Journal of Communication*, 43(4), pp. 347-358.

Michel, J. P. (2013). *Web service APIs and libraries.* Chicago, IL: American Library Association.

Mitchel, T. (2014, August 7). *Balancing innovation and risk in API policy.* [web log]. Retrieved from http://icouldntpossiblycomment.blogspot.com.au/2014/08/balancing-innovation-and-risk-in-api.html

Microsoft.(2014). Window 32 API. Retrieved July 15, 2014, from http://www.onestopqa.com/resources/Win32%20API.pdf

MobileMerit. (2014). Location Based InstaGram Photos. Retrieved October 25, 2014, from http://mobilemerit.com/location-based-instagram-photos/

Mokurai. (2012, Feburary 14). Open APIs are still closed source and non-Free. [web log comment]. Retrieved from http://www.linuxinsider.com/perl/board/mboard.pl?board=lnitalkback&thread=5891&id=5892&display=1#message_5892

Mosco, V. (2009).*The political economy of communicatio*n (2nd ed.). Los Angeles, CA: Sage.

Northlight Images. (2015). Commercial Photography. Retrieved July, 2015 from http://www.northlight-images.co.uk/commercial_photo_services.html

Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., &Zagorodnov, D. (2009).The eucalyptus open-source cloud-computing system. In *Cluster Computing and the Grid,* 2009. *CCGRID'09.9th IEEE/ACM International Symposium*, pp. 124-131. *IEEE*.

Open Definition. (2014). *Open Definition Version 2.0.* Retrieved from http://opendefinition.org/od/

Openstreetmap. (2012). Welcome, Apple. Retrieved from https://blog.openstreetmap.org/2012/03/08/welcome-apple/

OpenStack. (2014). Resources for application development on private and public OpenStack clouds. Retrieved December 22, 2014 from http://developer.openstack.org/#api

Oracle V. Google, No. C 10-03561. (2012). Retrieved from https://www.eff.org/files/alsup_api_ruling.pdf

Oracle V. Google, No. 10-CV-3561. (2014) retrieved from http://www.cafc.uscourts.gov/images/stories/opinions-orders/13-1021.Opinion.5-7-2014.1.PDF

O'reilly, T. (2007). What is Web 2.0: Design patterns and business models for the next generation of software. *Communications & strategies*, (1), pp. 17-31.

OSI. (2014a). *Open Source License.* Retrieved August 10, 2014 from http://opensource.org/licenses/

OSI. (2014b). *The Licence Review Process.* Retrieved August 10, 2014 from http://opensource.org/approval

OSI. (2014c). *FAQ*. Retrieved August 10, 2014 from http://opensource.org/faq#free-software

OSI. (2014d). *The Open Source Definition.* Retrieved August 10, 2014 from http://opensource.org/osd

Patterson, M. (2015, April 3). What Is an API, and Why Does It Matter? [web log] Retrieved from http://sproutsocial.com/insights/what-is-an-api/

Rosenberg, M. J. (2009). The Meaning of Open. [web log]. Retrieved December 14, 2014 from http://googleblog.blogspot.com.au/2009/12/meaning-of-open.html

Samuelson, P. (2014). Google, Inc. v. Oracle America, Inc.United States Supreme Court No. 14-410, Brief of Amici Curiae Intellectual Property Professors in Support of Grant of Petition for a Writ of Certiorari to the United States Court of Appeals for the Federal Circuit. University of California, Berkeley. Retrieved July, 2015 from http://sblog.s3.amazonaws.com/wp-content/uploads/2014/12/Google_v_Oracle_IP-Profs-certpetition-amicus-brief.pdf

Silverstone, R. (1999). *Why study the media?* London, UK: Sage.

Slippey, M. (2012, August 16). Changes coming in Version 1.1 of the Twitter API. [web log] Retrieved from https://blog.twitter.com/2012/changes-coming-to-twitter-api

Soulskill. (2011, December 30). Open Source Increasingly Replaced By Open APIs.[web log post]. Retrieved from http://news.slashdot.org/story/11/12/30/1317216/open-source-increasingly-replaced-by-open-apis

Spivack, N. (2012, August 16).*The Twitter API Insanity–What Everyone Seems to Be Missing.* [web log]Retrieved from http://www.novaspivack.com/uncategorized/the-twitter-api-insanity-what-everyone-seems-to-be-missing

Stallman,R. (2010).*Who Does That Server Really Serve?* GNU Operating System. Retrieved May 9, 2014 from http://www.gnu.org/philosophy/who-does-that-server-really-serve.html

Stallman, R. (2014) *"Why 'Free Software' is better than 'Open Source.'"* Retrieved September 18, 2014 from http://www.gnu.org/philosophy/free-software-for-freedom.html.en.

Stallman, R. (2014) 'Why Open Source misses the point of Free Software'. Retrieved September 18, 2014 from http://www.gnu.org/philosophy/open-source-misses-the-point.html.en.

Sullivan, W. (2010). *The big winner at ONA: APIs for news organizations. Journerdism.* Retrieved from www.journerdism.com/the-big-winner-at-ona-apis-for-news-organizations.

Smythe, D. W. (1981). *Dependency road: Communications, capitalism, consciousness,* *and Canada.* New York, NY: Ablex Publishing Corporation.

Tapscott, D & Willams, A. D.(2007).*Wikinomics: How Mass Collaboration Changes Everything.* London, UK: Penguin.

TheCarp. (2011, December 30). Re: I see no problem here. [web log comment] Retrieved from http://news.slashdot.org/story/11/12/30/1317216/open-source-increasingly-replaced-by-open-apis

The Guardian Open Platform. (2014a). FAQ. Retrieved March 10th, 2014 from http://www.theguardian.com/open-platform/faq

The Guardian Open Platform. (2014b). Partner Programs. Retrieved March 10th, 2014 from http://www.theguardian.com/open-platform/partner-programs

Tkacz, N. (2014). *Wikipedia and the Politics of Openness.* Chicago, IL: University of Chicago Press.

Turow, J. (2012). *The daily you: How the new advertising industry is defining your identity and your worth.* New Haven, CT: Yale University Press.

Twitter. (2014). Create an Application. Retrieved October 15th, 2014 from https://apps.twitter.com/app/new

Twitter. (2015). Developer Agreement & Policy. Retrieved May 30, 2015, from https://dev.twitter.com/overview/terms/agreement-and-policy

Twitter Developers. (2013). GET statuses/user_timeline. Retrieved December, 16 from https://dev.twitter.com/rest/reference/get/statuses/user_timeline

Twitter Developers. (2015a). OAuth Tool. Retrieved December, 16 2015 from https://dev.twitter.com/oauth/tools/signature-generator/8028632?nid=801

Twitter Developers. (2015b). Rest APIs. Retrieved Januray 10, 2015 from https://dev.twitter.com/rest/public

Twitter Developers. (2014). Introducing. Retrieved December 19, 2014 from https://dev.twitter.com/

United States Securities and Exchange Commission. (2014). *2013 Annual report of Google Inc*. Retrieved from

https://investor.google.com/pdf/20131231_google_10K.pdf

Vaidhyanathan, S. (2001).*Copyrights and copywrongs: The Rise of Intellectual Property and How it Threatens Creativity*. New York, NY: New York University Press.

Van Dijck, J. (2013). *The culture of connectivity: A critical history of social media.*

Oxford, UK: Oxford University Press.

Vis, F. (2013). A critical reflection on Big Data: Considering APIs, researchers and tools as data makers. *First Monday,* 18(10), Retrieved from http://firstmonday.org/ojs/index.php/fm/article/view/4878

Wagner, J. (2014). Overly Restrictive API Policies Kill Innovation. ProgrammableWeb. Retrieved December 14, 2014 From http://www.programmableweb.com/news/overly-restrictive-api-policies-kill-inno vation/analysis/2014/07/16

Wen, X., Gu, G., Li, Q., Gao, Y., & Zhang, X. (2012). Comparison of open-source cloud management platforms: OpenStack and OpenNebula. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference*, pp. 2457-2461. IEEE.

West, J. (2003). How open is open enough? Melding proprietary and open source platform strategies. *Research policy*, 32(7), pp. 1259-1285.

Wellman, B. (2001). Physical place and cyberplace: The rise of personalized networking. *International journal of urban and regional research*, 25(2), pp. 227-252.

Wilken, R. C. (2013). Foursquare, the Politics of Location Platforms, and the

Importance of Geocoded Data. *Selected Papers of Internet Research,* 3.

Wright,E.O. (1997). *Class Counts: Comparative Studies in Class Analysis.* Cambridge, UK: University of Cambridge Press.

W3school. (2014). Google API. Retrieved July 3, 2014 from http://www.w3schools.com/googleAPI/tryit.asp?filename=tryhtml_map_first

Xie, T., & Pei, J. (2006, May). MAPO: Mining API usages from open source repositories. *In Proceedings of the 2006 international workshop on Mining software repositories*, pp. 54-57.

Yee, R. (2008). *Pro Web 2.0 mashups: remixing data and web services.* New York, NY: Apress.

Zhou, M., Zhang, R., Zeng, D., &Qian, W. (2010). Services in the Cloud Computing era: A survey. In *Universal Communication Symposium (IUCS), 2010 4th International,* pp. 40-46. *IEEE.*