Politics of HCI and the User–Programmer Continuum

Tomás Laurenzo

School of Creative Media
Hong Kong
tomas@laurenzo.net

Abstract

In this paper, we propose characterising Human-Computer Interaction (HCI) as a negotiation between a specific design and its context of use. We then argue that HCI design is a political activity, and that the classification between users and programmers it commonly uses reflects a political stance, deeply rooted in its socio-political context. Finally, we propose that HCI can take inspiration from new media art's subversive appropriation of technological knowledge.

Keywords

Politics, Users, Programmers, HCI, New Media Art.

HCI as Negotiation

Human–Computer Interaction (HCI) is hard to define. Not only it is difficile to agree upon the range of topics that form it (Hewett et al., 1992) but, being a multi-disciplinary field, it also admits several different approaches.

Understanding HCI requires a holistic approach. Instead of focusing only on the design of the interface that a given product offers, it is necessary to take all its related phenomena into account. This implies understanding interaction itself as a significant and distinct object, subject to being studied and characterized. The *verbal* dimension of interaction needs to be studied, that is, HCI consists on something that happens over time when users employ a particular product to solve a particular problem in a particular context.

When focusing not only on the design of the interfaces devices offer but also on the sequence of actions that emerge from their use, this sequence of actions becomes a design object.

A rather trivial example would be that when a HCI practitioner designs a coffeemaker, not only its image and behavior are the design's outputs but also, more importantly, the *way* in which the user prepares coffee is.

It can be argued that HCI's scope is broader than this, for it is easy to imagine areas of interest that seem not to fit this framing. One example could be the design of the software of a call–center. In this case, with a captive user base, an ad hoc design, and a stable and known context of use, the HCI practitioner would not be at all concerned with the emergence of interaction but, instead, would focus on other, predefined, objectives (for example, maximizing the number of calls answered by an operator per unit of time).

Even in rather extreme cases like this one, the designed object still is the interaction that emerges, and the difference resides in the objectives that would guide such design.

In general, designers have limited control of the context of use of their products, and cannot control the characteristics, desires, and needs of their users. HCI practitioners, often only design a negotiation between the actions that their products propose and the specific characteristics of their use (the context, the users, the problems to be solved, etc.).

A main subject of interest of HCI is, then, the design of this negotiation and the cultural phenomena that surround it.

Users and Functionaries

> *Users of tools are much more prevalent than makers of tools. This imbalance has traditionally been rooted in the vast difference in skill levels required for using a tool compared to making a tool: To use a tool on the computer, you need do little more than point and click; to create a tool, you must understand the arcane art of computer programming.* -- John Maeda (Maeda, 2004).

Maeda's quote assumes that there exists a border, a frontier, which divides computer programmers from computer users. This assumption is very prevalent and common analyses of interaction and digital media are

routinely constructed using these two actors: users and programmers. Many users of tools and few makers of tools.

In the early stages of computing history, however, all users where programmers and there existed not a conceptual division between programming a computer and using one.

The creation of the *user interface* as a distinct concept is, interestingly, technological–centered, existing in a world where the computer is assumed, yet the user must be specified (Grudin, 1990).

Vilém Flusser introduced the concept of "functionaries". According to him, the functionary dominates an *apparatus* by controlling its exterior, its interface, and is in turn dominated by the ignorance of its interior. Functionaries are persons who dominate a game for which they are not competent (Flusser, 2013).

The "functionarization" of users is a deeply political process rooted in the role that software production plays in capitalist society. In effect, the separation between producers and consumers of digital artifacts constitutes a market necessity.

Software's increasing intricacy and power creates a new stratum of interactive complexity that, in turn, generates a new layer opaquing the interior of the digital apparatus. Yet, the very same powerfulness of new software products requires more refined expressive abilities from its users.

This apparent contradiction has been often tackled by inserting programming languages into user–oriented software products. These programming languages operate within a restricted context, and users become "contextual programmers", acquiring the needed expressiveness with-out subverting their assigned role.

This new layer of programmable complexity also operates in a self–contradictory way: it offers an appropriation path that diminishes the opacity of the inside of the apparatus, while at the same time creating a new level of abstraction that—in a Kantian turn—further separates the users from this inside, increasing the apparatus' opacity.

Still, a very significant cultural phenomenon appears: users writing code. There is an appropriation of the ubiquitous underlying technology, software. These hybrid users can acquire very sophisticated vocabularies in the languages proposed tools for which they become experts.

This empowerment becomes particularly important when we consider that software–creation techniques are usually translatable from one programming environment to the other; the logical building blocks of the clear majority of programming languages are extremely similar.

Creative Users

In a parallel phenomenon to the complexification of software and its need of allowing users to express themselves programmatically, new programming languages have been designed to facilitate the artistic appropriation of the digital medium. This is usually aimed at cultural producers and not at "regular" users. Again, this appropriation implies the appropriation of the underlying technology: the ability to write computer code.

A new name for this activity has been coined: creative computing. This coinage seems to only have the intention to demystify computer programming and encourage non– programmers to learn how to code, while simultaneously reclaiming the pertinence of "creative" individuals to this new environment. Interestingly, the division between users and programmers is so deeply rooted that cultural operators intuitively renamed computer programming in order to help users mentally cross the fictional user–programmer frontier.

However, this democratization of creative computing is a bounded one: only "creative" types are invited to the land of software creation.

Many efforts have been made to help this mental crossing, and several art and design–oriented programming languages and frameworks have been created. There are programming languages that do not look like programming languages (e.g. *patchers* like Max/MSP, PD, or VVVV), and programming languages that look like programming languages but are inserted into an environment where its essence is less noticeable (e.g. shader programming in 3D computer graphics software like Autodesk's Maya or 3Ds Max).

Some of the main characteristic behind the success of these "creative computing" environments include: a simplified syntax that does not hinder power; a consistent, step– by–step, online documentation; a custom, simple, programming environment; multiple platforms, including web; easiness to migrate to other (art–oriented or not) programming languages; and an active community and an open– source model (Laurenzo, 2009).

The User–programmer Continuum

There are, then, two symmetrical and complementary tendencies: the increasing programmability of software products, and the creation of programming environments that try to dissimulate their nature.

It should be obvious all programmers are also users. Even the most skilled in expressing themselves programmatically, when writing code are pure users of the operating system, the compiler, the IDE, and whatever other tools they happen to be using.

The truth is that there is not any conceptual difference between users and programmers, for programming languages are a specific subset of all the interaction languages.

In 1985, Hutchins, Hollan, and Norman introduced the concepts of articulatory and semantic distances as variables of direct manipulation interfaces (Hutchins, Hollan, & Norman, 1985). Semantic distance measures *what* is possible to be expressed in the interaction language and *how concisely* it can be expressed. Articulatory distance, in turn, measures *how similar* the interaction expression is to the idea behind it, how close a metaphor the interaction language offers.

Programming a computer consists on encoding orders in a specific interaction language. Programming languages are interaction languages with short semantic distance and usually long articulatory distance.

The only conceptual difference between a user clicking on a button and a programmer writing a specific algorithm are the different values for the semantic and articulatory distances of the interaction language used.

HCI is a Political Activity

The binary division between users and programmers is a crystallization of an ideological distribution of power. This distribution is propagated by a reductionist taxonomy of software that does not reflect the complexity of the interaction modes between humans and computers, and is functional to a power distribution schema that empowers a certain subset of interactors to the detriment of the vast majority of interactors, demeaned as "users".

The difference between users and programmers resides in the attitude that governs the interaction and not in the specific activities. Programmers naturally adopt an appropriating attitude that dives into the opacity of the apparatus, trying to understand its functioning and to profit from the freedom that emerges.

HCI has a peculiar relationship with power, for one of the main roles assigned to an HCI practitioner is to represent the users, being as a sort of users' advocate within the software construction process (Iivari, 2006).

HCI traditionally conceptualizes the user as a powerless entity to whom solutions are to be provided (even in participatory and user–centered processes). In the end, there is an ideological dispensation of power that demarcates the operational conceptual field and conceals a potentially richer field of appropriating interaction modes between humans and technology.

New Media Art as a Subversive Practice

New media art can be thought of as the art that emerges when artists appropriate the knowledge behind a certain (usually digital) technology (Laurenzo, 2013).

New media art's systematic explicitation of the appropriated technologies operates by situating the artist on different places on the user–programmer continuum. Every artistic production situates the artist somewhere in this axis (it always implies a certain relation with technology); how-ever, it is new media art's appropriation that turns it explicit. The location within this axis becomes a fundamental part of the art practice; *how* an art piece is created becomes a defining part of the artwork.

For example, an artwork created "functionarily" manipulating a certain piece of software such as Adobe After Effects is ontologically different from an identically looking piece programmed using the Processing programming language.

The difference lies in the (explicitly) different relationship with the technology. Even if the results can be the same, the appropriating relationship only in one case conceptualizes the technological manipulation as part of the artwork.

This was intuitively understood by new media artists, and the already mentioned renaming of programming into "creating computing" reclaims some power, proposing a different relationship between artists and the creation of software.

However, every technological product is understood from within a certain conceptual framework. For example, bitmap drawing software could be seen as tools for creating drawings, or as pieces of software conceived to allow their users to modify the values in a specific area of a computer's memory.

The meaning and significance of a specific tool is dependent on its own conceptual opacity. Understanding

the cultural significance of a new media art piece also requires an analysis rhetoric that understands (that appropriates) the piece's technology.

For example, we can compare a record player and a violin, both understood as musical instruments. If we were to pick the best musical instrument, it is easy to find reasons for both (a violin can only sound like a violin, while a record player can sound as many things. A record player only can play whatever is recorded on the available records while a violin can play whatever its operator chooses. This choosing is very hard in a violin, while it's much easier on a record player. A violin does not need an external power source while a record player does, and so on and so forth).

What cannot surprise us is the conceptualization of the record player as an instrument, for our knowledge of its possibilities and the involved technology allow for it.

Conclusions

The immanent politicality of HCI and its deep intertwining with its sociopolitical context is often overlooked. Contemporary discourses aim at understanding the implications of HCI practice but usually fail in grasping that some of its fundamental blocks are dependent of this context. Especially important is the characterization of users and programmers as distinct entities, which obeys to a market need and not to an ontological reality.

This has been intuitively understood by new media artists who have reclaimed some of the power that has been taken away from users.

New media art is an intrinsically subversive activity. Its appropriation of the knowledge behind technology production effectively challenges this division of power, up to the point that new names for traditional practices had to be employed.

If the future of HCI includes perceptual interfaces and context awareness (Canny, 2006), then this reality will be even harder to dissimulate, for HCI practitioners will explicitly be describing the social contexts of the problems to be solved.

If HCI is to contribute to a better, more just world, it needs to question its philosophical stance. Its rhetoric cannot ignore its inextricable politicality. If one possible objective of the artistic practice is to know more about the human condition, HCI, with its systematic study of the human interactor, may take inspiration from new media art's questioning of the power behind the assumptions in our relationship with technology.

References

Canny, J. (2006). The future of human-computer interaction. Queue, 4(6), 24-32.

Flusser, V. (2013). Filosofia da caixa preta: ensaios para uma futura filosofia da fotografia (Coleção Comunicações) (Por- tuguese Edition), 66.

Grudin, J. (1990). The computer reaches out: the historical continuity of interface design.

Hewett, T. T., Baecker, R., Card, S., Carey, T., Gasen, J., Mantei, M. Verplank, W. (1992). ACM SIGCHI curricula for human-computer interaction. New York: ACM.

Hutchins, E. L., Hollan, J. D., & Norman, D. A. (1985). Direct manipulation interfaces. Human–Computer Interaction, 1(4), 311-338.

Iivari, N. (2006). Understanding the work of an HCI practitioner.

Laurenzo, T. (2009). New media art. MSc in Computer Science, PEDECIBA / Universidad de la República, Montevideo. Laurenzo, T. (2013). Decoupling and context in new media art. PhD in Computer Science. PEDECIBA / Universidad de la República, Montevideo.

Maeda, J. (2004). Creative Code: Aesthetics + Computation (Thames & Hudson ed.). Thames & Hudson.

Author Biography

Tomás Laurenzo is an artist and academic who works with both physical and digital media exploring the artistic construction of meaning and its relation with power and politics.

Laurenzo's production spans across different practices, including installation, interactive art, music, live cinema, and digital lutherie. His artworks and performances have been shown in the Americas, Europe, Asia, and Oceania.

He is Assistant Professor at the School of Creative Media of the City University of Hong Kong.

He has several publications, mainly in the areas of New Media Art, NIMEs, and HCI.

Please see http://laurenzo.net for more information.