EUD from a Historical Perspective as Cause to Redefine it in Today's Landscape

Nikolaos Batalas
nikolaos.batalas@gmail.com
Eindhoven University of Technology

Javed-Vasilis Khan
v.j.khan@tue.nl
Eindhoven University of Technology

Panos Markopoulos
p.markopoulos@tue.nl
Eindhoven University of Technology

ABSTRACT

This paper aims to bring into focus contradictions in prevailing approaches to defining End User Development (EUD). Motivated by a paradox we observed given these definitions, we invite researchers to reconsider the scope of EUD as encompassing the whole of software development, and point to the historical evolution of the circumstances under which software is developed for reasons to do so.

1 INTRODUCTION

End-User Development (EUD) is a field of research dedicated to making software.[1] It differs from the traditional academic disciplines dedicated to the task, such as computer science, in that it aims to allow people to build computer programs that help them perform certain tasks, without demanding that these programmers have to rely on the depth of knowledge and expertise that it takes to program computers on the technical level that the traditional disciplines are concerned with, such as the construction of algorithms, or the implementation and manipulation of data structures.

Contradictory definitions of EUD are in use

Consensus is not uniform amongst researchers in EUD with regard to a definition of the nature of the issue and the cohort under investigation. Commonly cited definitions, tend to be inclusive, summative descriptions of research that has taken place in the field, rather than the activity, End-User Development, itself. For example, a definition often cited in contemporary literature is offered by Lieberman et al [14]:

> End-User Development can be defined as a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artefact.

As is the case in this definition, the tendency to define end-users and their activities as opposites to professional developers and their activities, is pervasive throughout literature. Attempts at defining each role tend to be mostly in service of illustrating a particular point that the author is making, and can be found to contradict other definitions. For example, in [14], [13], [23], end-users are regarded as novices or less skilled than programmers. Nardi [20] regards end-users as professionals in unrelated fields, who do not care about computers, but want to get their work done. Ko et al. [12] acknowledge the problems of considering end-user programmer characteristics a matter of personal identity and instead propose that the characterization end-user vs professional is a matter of intent. End-user or professional developer then is a role one performs at any given time, the degree of which is determined not by skill and expertise, but by the number of users they are serving their product to, with serving one's own needs being the definition of end-user development.

Authors offer definitions of EUD in order to get their point across to readers, and to do so attempt to capture aspects that feature prominently, but might not be exhaustive. Evidently, definitions of EUD evolve over time, but also co-exist with each other. Also, the dichotomy between end-users and professionals remains undisputed, and the task of programming, especially when referring to professionals, is treated wholesale, without nuance as to what is being programmed.

A paradox in Ambulatory Assesment

Ambulatory Assessment is a field of research within clinical psychology with the purpose of capturing bio-psycho-social processes in the context of daily life [8], gathering self-reports and sensor measurements from groups of people over time. Data collection happens predominantly via mobile platforms and considerable effort goes into the making of tools to enable clinical psychologists to define what data these platforms collect and how [3]. With these tools, psychologists effectively write software which they distribute to potentially hundreds of users, their study participants, to use for recording data. Depending on what aspect of the effort one focuses on, under Ko et. al [12] psychologists could be considered akin to professional developers for intending their software to be used by numerous end-users, or they could be considered end-user developers for intending to get their own work done.

[1]Related terms such as End-User Programming (EUP) or End-User Software Engineering (EUSE) all address specific aspects of software creation/maintenance, program creation in the first instance, reliability, reuse and maintainability in the latter. Since EUP and EUSE take place in the service of development, we can consider EUD to be the more encompassing term.

Definitions of EUD are important

The definition of EUD is an essential issue. Lieberman et al. [14] stress the importance of EUD for the participation of citizens in the Information Society. The stakes therefore are high, and the field needs to remain vigilant towards the clarity of its vision and communications. Amoroso et al. [2] warn against the dangers of variable definitions. Differing perspectives might render research results incomparable, and researchers are forced to recreate frameworks in which to work. Fischer et al. [11] also point out that there exists a challenge for the EUD research community in understanding and supporting existing and new cultures, and call for an analytic model to understand and explain phenomena and environments. They also note that the impact of EUD research has been limited. The fact that a deep understanding of the phenomena under investigation is still sought after, could be a contributing factor to EUD's limited impact.

In the rest of this text we hope to show that the conception of the end-user and the professional developer is historically evolving according to the platform being used, and also that the history of computer programming is really a history of end-user programming. Given the paradox in Ambulatory Assessment, we will attempt a definition of EUD driven by platform, rather than identity or intent.

2 END-USER DEVELOPERS HISTORICALLY

The term end-user is arguably an invention of IBM in the 50s [16]. It was used to point to people such as corporate executives who would be the budget holders responsible for commissioning the technology[21], and separate them from intermediate users, usually experts who would operate the machines[22], working in data processing departments, tasked with answering questions posed to them by management.

In the 60s and 70s, solid state transistors and the microchip brought speed and power to mainframes, and gave rise to minicomputers. Computing became dramatically cheaper and thus more accessible to members of organizations working outside of computer centers. End-users were now people with access to machines, and computing departments started dealing with the strategies for organizations to provide their members with access to applications of the enterprise, as well as manage their workload.

As the trend continued In the late 70s and 80s, employees were able to independently procure their own personal microcomputers[4]. End-users were the users of application software which they used to do their own data processing, and the field of End User Computing (EUC) came more prominently into being, specifically concerned with computing within organizational settings. Growing demand for software solutions led EUC research to consider how to let these end-users become developers of their own programs [17]. For the purpose of studying the use of computers in organizational settings, taxonomies of users were proposed [7] [17] [24] [9]. Classifications of this sort make sure to set apart data-processing professionals, who are employed to write code for others. Workers of the organization who are trained in other domains are classified as amateurs, who according to Weinberg [26] only program for themselves. This is perhaps the reason why end-user developers are juxtaposed against professional developers, to this day.

During this time and into the 90s with the popularization of the Graphical User Interface (GUI), desktops were adopted even more widely, and the electronic spreadsheet became one of the most popular applications for data-processing by end-users. It offered an easy to understand and visually manipulate data structure, and allowed users to perform automated operations on it in ways more intuitive than the type definitions, loops and memory management of typical programming. It became one of the most prominent success stories in literature for End User Programming.

Gradually, EUC in organizations became less concerned with application software and EUD. Data processing departments evolved into Information Technology (IT) departments, managing the technology infrastructure which allowed an organization to run, i.e. hardware, networks, software licences and data storage, and supporting end-users in accessing it. Security and scalability became the more prominent issues.

EUD moved into the domain of Human Computer Interaction (HCI) [19], where research was invested in understanding and supporting programming tasks, both as a general issue, and also within specific application domains. As computing became a staple of daily life in various forms, discussions on EUD also became disentangled from explicitly organizational settings. It seems plausible however that conceptions of end-users and professional programmers have their origins in those settings and their specific concerns.

3 PROFESSIONAL DEVELOPERS

Programmers have always tried to build platforms that would allow them to function as end-user developers on. That is, as people who want to get their work done and should not have to care about (some aspects of) the computer, as in Nardi [20]. We maintain that the evolution of computer programming and its related tools is very much an EUD success story.

Historically, many advances in computer programming have come in the form of layers of abstraction, whereby two things are achieved; the creator of the abstraction is able to suppress details of the underlying layers, which are irrelevant to a certain programming task, and also, to invent and express the model of a machine which is relevant to the task, and perhaps even already familiar to the user of the abstraction [15].

After initial innovations in performing binary operations with relays and switches[25] and the first electronic computers in the 40s, the 50s saw the rise of the stored program and the programmable computer, where the hardware does not need to be re-wired per program. In 1952, Adams [1] discusses how subroutines, accessible as symbols of abbreviated words make it possible for the increasing number of computer users to produce usable programs of numerical analysis. His focus lies on allowing entry level programmers to achieve results, and envisions that a verbal statement of the problem will be sufficient for the computer.

In subsequent years, a host of programming languages and compilers were invented to people who wished to program computers in terms closer to their level of expertise or to their application domain. Many of the innovations we regard today as arcane were driven by the personal needs of their inventors to get their job done. For example, FORTRAN, offering a way to define algebraic expressions, was heralded as a "revolution [..] to have engineers, scientists, and other people actually programming their own problems without the intermediary of a professional programmer"[10]. UNIX came into being because of the desire of its makers to have their own time-sharing system. The C language offers programmers the model of an abstracted computer with certain features to manage its memory, and allows them to (largely) not care about the particulars of the hardware itself.

Innovations with regard to making code reusable, rendered so by programming-language constructs such as classes, objects, encapsulation, and distributing as code libraries, is a way of making these software artifacts end-user programmable. Notably, programmers in their daily practice set intermediate personal goals to structure their code in such ways as to build abstractions and interfaces and hide its complexity, so as to later render themselves end-users of it, and make it easier for themselves to get their job done by using it as a functional unit.

Furthermore, software development is not a single domain, and does not imply a singular technical profile of a practitioner. Different programmers develop their craft in vastly different technical problems that they are trying to solve, have domain knowledge on different levels of abstraction within the software-hardware stack, many of which have their own elaborate theoretical backgrounds (e.g. graphics programming vs database programming) and are end-users of various tools and platforms in order to carry out their work. Illustrative of this are job listings seeking professional programmers, which advertise not only for a particular application domain (e.g. front-end development) or a specific programming language (e.g. JavaScript) but for familiarity with specific code libraries and APIs (e.g. Angular vs React).

Also nowadays, it seems that tools typically used in the realm of professional software development, such as expert-exhanges (e.g. stackoverflow), where discussions are held on how various pieces of code can be used, code repositories and issue trackers (e.g. github), where people reuse and modify existing programs, report bugs, and propose desired features, or code sandboxes (e.g. jsfiddle) which illustrate code execution, are in the service of facilitating cultures of participation [11], as envisioned for EUD.

4 DISCUSSION

End-User vs Professional

Researchers in the field choose to define the End-User Developers by opposing them to Professional Developers, because ordinarily, someone who receives a salary to develop code would have a radically different approach to the problems, informed by radically different concerns. However, the term end-user describes a person's relationship to a platform, while the second an organized way to make a living. These terms are not in opposition to each other, but rather orthogonal concerns, unrelated to each other.

This can be illustrated by considering an EUD method, process or tool that would come to be so successful, as to produce reliable and useful software artifacts for large numbers of users, and that would allow the authors of that software to make a living out of it, and devote themselves to it professionally. Surely that process would still be an EUD process, and the developers using it, would still be end-user developers since none of its intrinsic characteristics would change.

End-User vs Technical, as seen by Platform

In his Theoretical Introduction to Programming, Mills [18] devotes a section to the notion of Technical Programming:

> Technical programming is about defining a specific problem as clearly as possible, and obtaining a clear solution.[...] It has much in common with the technical (rather than bureaucratic) aspects of all engineering disciplines.[...] Precise subproblems are identified.[...] What is, or is not, technical depends on the techniques available.

Perhaps then, in the place of Professional Programming, we can adopt the term Technical Programming, one that focuses only on the task itself, and the kind of engagement with problem-solving that demands good grounding in methodology, the ability to identify sub-problems, and to give structure to the problem domain [18]. Where End-User Programming would consist of, e.g., using function calls on a platform/abstraction, Technical Programming would be building the platform/abstraction and the functions that end-user programmers can call.

Finally, we propose that a platform-driven lens can be developed to help determine the nature of one's software development task at a specific point in time as being end-user development or technical development. Platform is a framework (be it hardware or software) that supports other programs. Platform studies [5] offer the theoretical framework both for conducting a discourse on platforms and when and whether it is useful to view a given system as such [6], not only from a technical, but also from a cultural perspective.

In adopting a platfor-driven view, one would have to acknowledge that End User Development is part and parcel of any creative programming endeavour, and practiced daily in professional or other settings, since making use of the abstractions a platform offers, is to perform EUD on it. As these abstractions are used in the service of more technical problems, so does the development task become of a technical nature, possibly leading to a new layer of abstraction, and the cycle repeats.

Paradox resolution

A platform-driven view of End User Development could help resolve the paradox mentioned in Ambulatory Assessment (AA), by considering the population of mobile phone users in the data collection effort as a system of two layered platforms. The first is the AA mobile software platform, the abstractions of which the clinical psychologist uses as an end-user developer,to program each mobile phone's behaviour, e.g for signaling a participant to provide self-reports. Collectively, the cohort of participants with their mobile phones, comprise a higher-level platform that generates the data that the clinical psychologist can feed into their analysis.

5 CONCLUSION

Both end-user and developer are largely inventions of the platform being used and developed on. As platforms evolve through history, our understanding of who end-user developers are and what they do evolve as well. Increasingly, software development takes place on such multiple layers of abstraction, with platforms and tools already delivered to the developers, that an EUD aspect is always involved. A platform-driven definition of EUD, inclusive of all types of developers as beneficiaries of its findings, could better reflect this state of things, and also anticipate the future.

REFERENCES

[1] Charles W Adams. 1952. Small problems on large computers. In *Proceedings of the 1952 ACM national meeting (Pittsburgh)*. 99–102.

[2] Donald L Amoroso. 1992. Using end user characteristics to facilitate effective management of end user computing. *Journal of Organizational and End User Computing (JOEUC)* 4, 4 (1992), 5–16.

[3] Nikolaos Batalas, Vassilis-Javed Khan, Minita Franzen, Panos Markopoulos, and Marije aan het Rot. 2019. Formal representation of ambulatory assessment protocols in HTML5 for human readability and computer execution. *Behavior Research Methods* 51, 6 (2019), 2761–2776. https://doi.org/10.3758/s13428-018-1148-y

[4] David H Benson. 1983. A field study of end user computing: Findings and issues. *Mis Quarterly* (1983), 35–45.

[5] Ian Bogost and Nick Montfort. 2007. New media as material constraint: An introduction to platform studies. In *Electronic Techtonics: Thinking at the Interface. Proceedings of the First International HASTAC Conference.* 176–193.

[6] Ian Bogost and Nick Montfort. 2009. Platform studies: Frequently questioned answers. (2009).

[7] Codasyl End User Facilities Committee et al. 1979. Codasyl end user facilities committee status report.

[8] Tamlin S Conner and Matthias R Mehl. 2015. Ambulatory assessment: Methods for studying everyday life. *Emerging Trends in the Social and Behavioral Sciences: An Interdisciplinary, Searchable, and Linkable Resource* (2015).

[9] William W Cotterman and Kuldeep Kumar. 1989. User cube: a taxonomy of end users. *Commun. ACM* 32, 11 (1989), 1313–1320.

[10] Nathan L Ensmenger. 2012. *The computer boys take over: Computers, programmers, and the politics of technical expertise.* Mit Press.

[11] Gerhard Fischer. 2009. End-user development and meta-design: Foundations for cultures of participation. In *International Symposium on End User Development.* Springer, 3–14.

[12] Andrew J Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, et al. 2011. The state of the art in end-user software engineering. *ACM Computing Surveys (CSUR)* 43, 3 (2011), 1–44.

[13] Henry Lieberman. 2001. *Your wish is my command: Programming by example.* Morgan Kaufmann.

[14] Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf. 2006. End-user development: An emerging paradigm. In *End user development.* Springer, 1–8.

[15] Barbara Liskov and Stephen Zilles. 1974. Programming with abstract data types. *ACM Sigplan Notices* 9, 4 (1974), 50–59.

[16] Wendy E Mackay. 1990. *Users and customizable software: A co-adaptive phenomenon.* Ph.D. Dissertation. Citeseer.

[17] Ephraim R McLean. 1979. End users as application developers. *MIS quarterly* (1979), 37–46.

[18] Bruce Ian Mills. 2005. *Theoretical introduction to programming.* Springer Science & Business Media.

[19] Brad A Myers, Andrew J Ko, and Margaret M Burnett. 2006. Invited research overview: end-user programming. In *CHI'06 extended abstracts on Human factors in computing systems.* 75–80.

[20] Bonnie A Nardi. 1993. *A small matter of programming: perspectives on end user computing.* MIT press.

[21] Jan Noyes and Chris Baber. 1999. *User-centred design of systems.* Springer Science & Business Media.

[22] Stephen P Plusch. 1984. *The Evolution from Data Processing to Information Resource Management.* Technical Report. ARMY WAR COLL CARLISLE BARRACKS PA.

[23] Alexander Repenning and Andri Ioannidou. 2006. What makes end-user development tick? 13 design guidelines. In *End user development.* Springer, 51–85.

[24] John F Rockart and Lauren S Flannery. 1983. The management of end user computing. *Commun. ACM* 26, 10 (1983), 776–784.

[25] Claude E Shannon. 1938. A symbolic analysis of relay and switching circuits. *Electrical Engineering* 57, 12 (1938), 713–723.

[26] Gerald M Weinberg. 1998. The Psychology of Computer Programming; 1971. *New York: von Nostrand Reinhold* (1998).