On Continuing Creativity

Colin Clark
Inclusive Design Research Centre
OCAD University
cclark@ocadu.ca

Sepideh Shahi
Inclusive Design Research Centre
OCAD University
sshahi@ocadu.ca

Abstract
For decades, the software industry has struggled with change, continually devising new methods to better control and manage the risk to software development projects. This paper attempts to reconsider change as a positive force that can produce better, more resilient software. It argues for providing "users" with greater creative influence throughout the design process, as co-designers; and to support them with material software tools that will allow them to enact unanticipated changes after design is complete.

1. Inward-Facing Software Methods
Change is hard. Much of the history of software design methodology has focused on devising strategies for controlling, minimizing, or formalizing change. Some methods attempt to reduce the possibility of disruption by "getting it right" up-front, via the use of formalized requirements gathering strategies and user research models[1]. Others aim to "embrace change" by establishing management and programming tactics that reduce the cost of responding to changes introduced at any time during the software development process[2]. All these methods invariably look inwards, at the working practices of teams of expert designers and programmers who have some measure of influence over the ways in which change is conceived, managed, and responded to during the process of creating a piece of software.

Iterative, or agile, design and development methods involve the incremental evolution of a software product over the course of many short cycles of research, design, implementation, and stakeholder feedback. This approach offers effective opportunities for technical teams to change direction quickly and to recalibrate a program's features, user interface, and other designed qualities more quickly and at a lower cost. However, it is notable that many of the most popular agile approaches to software development, such as Lean, Scrum, and others, still retain an explicitly industrialist mindset, often mimicking the assembly line manufacturing processes and production strategies of consumer objects such as automobiles. For example, Toyota's "lean" Production System is often cited as a significant influence on agile software development methodologies[3]. While most agile processes have a "customer" or "product owner" role within the team, this role is usually performed by a single person who is embedded within or nominated by the management structure of the organization, and who may not represent the diverse needs and perspectives of the day-to-day users of the software.

Here, an acceptance of change means cultivating strategies and technologies that support increased modifiability, parameterization, or reuse of software artefacts *during the production process*. Industrialized product design methods—including even the most flexible agile practices—still assume a conventional

[1] As an example, Alan Cooper's interaction design methods primarily provide design-based management strategies that centre around the creation of formalized design models such as personas and scenarios at the beginning of the software lifecycle. See Cooper, Alan et al (2014). *About Face: The Essentials of Interaction Design 4th Edition.* Wiley.

[2] Beck, Kent. (2000) *Extreme programming explained: embrace change*. Addison-Wesley Professional.

[3] In his "Agile Versus Lean," Martin Fowler states that "There was a connection between lean manufacturing and agile software from the beginning in that many of the developers of the various agile methods were influenced by the ideas of lean manufacturing." https://martinfowler.com/bliki/AgileVersusLean.html Mary Poppendieck has elaborated an entire software development methodology based on the direct application of Toyota's automobile manufacturing methods to programming, such as in her "Lean Software Development." https://dl.acm.org/citation.cfm?id=1248986

producer/consumer dynamic, in which an object (in this case, the software product), though it may have changed frequently and freely over the course of its creation, is delivered to a consumer in a finalized and largely unchangeable form[4].

2. The Politics of Use

For users, on the other hand, change is arguably even harder. Lacking direct influence over the process of its creation, software products tend to be a "take it or leave it" proposition for users—they work as they do, with perhaps limited configurability or the possibility of buying costly custom tailoring provided by vendors or consultants. If an individual user needs something different, often their only recourse is to look elsewhere, at other products. Yet at the same time, change is increasingly imposed on users, with the rise of mandatory software updates and cloud-based software-as-a-service platforms. Beloved features may disappear, move, or be recast by software designers at any time and without notice or permission, leaving users to adapt or relearn their hard-earned workflows.

There is, of course, a power dynamic at work here. While the difficulties of change are felt by software designers and users alike, the power to enact (or forgo) changes, to manage their impacts, scales and timing, rests overwhelmingly in the hands of those who initially created the software. Although users often pay for their software, substantive ownership of it—that is, creative control over what it is, when it changes, and how it is intended to be used—remains too often locked up with those who originate it. This dynamic is reflected in the methods by which much software is made, the models by which it is deployed and distributed, and by structure of many programming idioms themselves—compiled, unidirectional, insular[5].

It is becoming increasingly clear that equitably-designed software systems—those that are capable of including a diversity of needs and experiences—may never be fully realized using inward, expert team-facing processes that result in static software "products." Software's complexity, and the dizzyingly varied, situated needs of different people and communities, suggest that we may never be able to design one single artefact that fits everyone. At the same time, the costs of specialized, isolated, and incompatible systems are increasingly unsustainable[6]. Today's methods leave designers in a situation where their only substantive means for dealing with change is via the power of omission—intentionally leaving out features that have been measured, through one process or another, as having limited value, scope, or utility to a hypothetical average, norm, or majority of users[7]. For users on the margins, such as those who with disabilities or who have difficulty with literacy of complex digital systems—and thus who may need features that are specialized or individualized in some way—this "economy of the mainstream" perpetuates marginalization and exclusion.

Change is hard, yet we need more of it—in new forms. We need design methods that open up new vectors of change, which include the reciprocal participation of diverse individuals and communities from the very beginning, and which allow those users to continue the design process—configuring, adding and removing features, connecting software artefacts together, sharing customizations with each other—even after the

[4] On the other hand, it is notable that agile methods have flourished within in-house or long-term contracting software development teams, where there is an ongoing relationship between the software's developers and the organization sponsoring it. This enables changes to be made continually throughout the lifetime of the software's use within the organization. The scale and cost of this relationship, however, is far out of reach of ordinary users, and is typically reserved for large enterprises who can afford to continuously fund an active software development team.

[5] The consequences of encapsulation, and a discussion of alternative, open programming models, can be found in Clark, C., & Basman, A. (2017). "Tracing a Paradigm for Externalization: Avatars and the GPII Nexus." In *Proceedings of the 2017 International Conference on the Art, Science, and Engineering of Programming Workshop: Salon des Refusés*.

[6] The assistive technology market has been plagued by the issues associated with specialized, poorly interoperable solutions, as documented in Treviranus, Jutta (2018). Let's not spend public funds to perpetuate digital disparity. Medium. Available at https://medium.com/@jutta.trevira/lets-not-spend-public-funds-to-perpetuate-digital-disparity-a4413132ca

[7] Treviranus, Jutta (2018) If you want the best design, ask strangers to help. Medium. Available at http://openresearch.ocadu.ca/id/eprint/2191/21/Treviranus_Strangers_2018.pdf

originary design process is finished[8]. By reorienting our design processes around the idea of *continuing creativity,* we raise the stakes for the embrace of change while aiming to rebalance the relational dynamics of software's design and use. This perspective emphasizes the role of non-expert "users" as co-designers during the initial process of software's conception and creation, as well as advocating for new programming tools that support the continued redesign of software artefacts by non-programmers, even after they have been designed, coded, and shipped.

3. Inclusive Co-Design

Co-design is designing *with,* not simply *for*. It involves asking the people who might otherwise just be "users," particularly those on the margins of today's technology experiences, to be part of the design process from the beginning. Lacking the proceduralism of industrialized design methods, co-design typically starts with a process of discovering and negotiating roles—asking participants how, when, and how often they want to be involved, and making space to accommodate different "scales" of investment and engagement. As a result, co-design takes time to do well. Its processes need to be tailored to the unique context and situation that a design intervenes into, and it demands that all participants have equal access to the information—plans, ideas, prototypes, and works in progress—that is essential for full decision-making and responsible contribution. A starting point for this involves an opening up of agile's iterative and incremental processes to be more porous and include a broader range of team participants and modes of engagement. An example of this is the Fluid Project's co-design practices, which combine designing and planning in an open wiki, remote participation in design crits and decision-making processes, and "embedded" co-design activities, all within the context of an open development community[9]. Fluid's embedded co-design process involves creating toolkits of activities and resources that are intended to help engage people in the design process, situated within the context of their own communities, participants, and places. These toolkits are given to communities to organize and use themselves, without Fluid's designers present, along with training and mentorship if needed. Thus, multiple levels of co-design can be performed independently and at different locations without central organization, all facilitated by those who are trusted within the communities they are practicing within.

[8] This echoes Pelle Ehn's concepts of "use before use" and "design after design." See Ehn, Pelle. (2008). Participation in Design Things. In *Proceedings of the Tenth Conference on Participatory Design, PDC* 2008. Available at https://www.researchgate.net/publication/221631329_Participation_in_Design_Things

[9] The Fluid Project is an open source community that designs new material technologies and co-design methods. Their work is described in Clark C., Ayotte D., Basman A., Treviranus J. (2016) About Us, with Us: The Fluid Project's Inclusive Design Tools. In: Antona M., Stephanidis C. (eds) Universal Access in Human-Computer Interaction. Methods, Techniques, and Best Practices. UAHCI 2016. Lecture Notes in Computer Science, vol 9737.

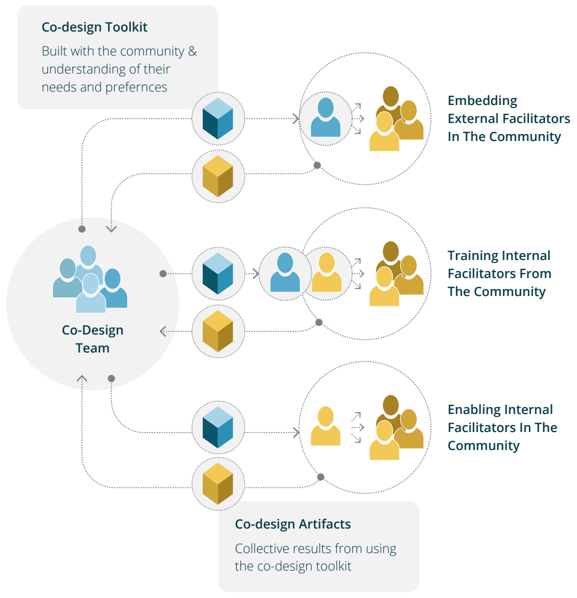
Co-design Toolkit
Built with the community & understanding of their needs and prefernces
Embedding External Facilitators In The Community
Co-Design Team
Training Internal Facilitators From The Community
Enabling Internal Facilitators In The Community
Co-design Artifacts
Collective results from using the co-design toolkit

Diagram of Fluid's embedded co-design process, which includes creating toolkits, training, and enabling communities to co-design within their own context and leadership structures.

Co-design must be reciprocal. As Sherry Arnstein notes, participation is power[10]. Co-design is not a case of designers *allowing* users to participate, but rather, fully engaging them as citizens of the technological spaces that they are increasingly inhabiting, working, and expressing themselves within. This involves practicing in ways that are self-aware of the profound power and privilege that technologists hold, and in finding ways to fully share and give up that power. It is not enough simply to ask people for feedback; participants in co-design need to know that their input and ideas can have real power and influence in the resulting software. Co-design demands the knowledge that ideas will be heard, that they can have a direct influence, and that the mechanisms and processes by which they will potentially be enacted are clear and accountable. To this end, the *Co-Designing Inclusive Cities* project is currently developing a toolkit of co-design practices and activities, along with community-led measures that can be used by participants to assess the extent and effectiveness of their engagement[11]. The project's toolkit will be used to support the community-led design of civic infrastructure and connected cities such as Toronto's Quayside project—an area in which there is currently significant risk of placation and consultation tokenism, and thus co-design practices are acutely needed.

Co-design is not a new approach. It draws its roots from Scandinavian cooperative design projects such as Utopia, in which labour union members directly contributed to the design of new computer workstations[12].

[10] "Citizen participation is a categorical term for citizen power. It is the redistribution of power that enables the have-not citizens, presently excluded from the political and economic processes, to be deliberately included in the future. It is the strategy by which the have-nots join in determining how information is shared, goals and policies are set, tax resources are allocated, programs are operated, and benefits like contracts and patronage are parceled out. In short, it is the means by which they can induce significant social reform which enables them to share in the benefits." Arnstein, Sherry. (1969). "A Ladder of Citizen Participation." AIP, Vol. 35, No. 4, July 1969, pp. 216-224.

[11] Inclusive Design Research Centre. (n.d.) Co-designing Inclusive Cities. Retried from https://cities.inclusivedesign.ca/

[12] Bødker, S., Ehn, P., Kammersgaard, J., Kyng, M., & Sundblad, Y. (1987). A Utopian experience: On design of Powerful Computer-based tools for skilled graphic workers. In Computers and Democracy - a Scandinavian challenge. ed. / Gro Bjerknes; Pelle Ehn; Morten Kyng. Gower Publishing, 1987. p. 251-278.

Elsewhere, Elizabeth B.N. Sanders has emphasized the value of engaging participants creatively within what she calls the "fuzzy front end" of design—the early, ambiguous, and chaotic phase of the design process prior to a product being fully conceptualized[13].

While co-design clearly opens new possibilities for more equitable participation by workers and citizens within software design teams during the scope of initial creation, ongoing change remains a factor that needs to be addressed. People change over time, our needs change, and unanticipated possibilities invariably emerge from use and experience. This suggests, then, that there is a need to find ways to extend the creative participation that co-design offers early in the process, continuing it even after the initial design process is finished.

4. Material Systems

Mads Dahlke is the Danish host of a popular do-it-yourself YouTube channel called *Sail Life*[14]. For the past several years, Dahlke has documented his process of taking a fully functional, thirty-year-old sailboat, disassembling it in a variety of artful and intrusive ways, and rebuilding it to suit his own needs and tastes as a sailor with ambitions of crossing oceans in it. Among other projects, Dahlke has removed and replaced the boat's entire deck, designed custom-fit fuel tanks, completely reconfigured the layout of a cabin to better suit his needs as a workspace, and repaired many flaws resulting from oversights or cut corners during the boat's original design and construction. Indeed, watching Dahlke's weekly uncovering of new issues and challenges suggests that the boat's original designers simply never conceived that their product would still be in active use today, nor did they design it with any intention of it being modified in the ways that Dahlke has accomplished. Yet Dahlke admits he is not a professional boat builder or expert restorer. He has pursued his project by acquiring some generalized technical skills and commodity tools, while participating in a larger community of other sailors and do-it-yourselfers who have worked on similar projects and shared their own learnings.

Although Dahlke's endeavour may not be particularly unique when seen from the perspective of DIY repair and renovation, it represents something that is very difficult, if not impossible, to do with software systems today. Coincidentally, Dahlke's "day job" is as a professional software developer. And yet, despite his elite technology skills, Dahlke would nonetheless be hard-pressed to perform a comparable series of transformations and personalizations on a (closed source) piece of software that someone else had designed before him—especially at a comparable cost. His power to change his sailboat after the fact of its design far exceeds his power to change software under similar circumstances. For a non-professional—an "ordinary user"—such transformations would certainly be out of reach entirely. The cost and complexity of unanticipated change in software is always significant[15], often intractable, and even sometimes inconceivable[16].

Material software is software that provides the power to be adapted, configured, re-presented, augmented, or separated in various ways, without needing to have been part of the original software development process or to have access to "elite-level" programming knowledge or tooling. In its simplest form, this materiality may take the form of simple transformations of a software's user interface, such as those provided by the UI Options accessibility preferences framework[17]. Fluid Infusion (of which UI Options is

[13] Elizabeth B.-N. Sanders & Pieter Jan Stappers (2008). Co-creation and the new landscapes of design, CoDesign, 4:1, 5-18.

[14] Sail Life. (n.d.) Home [YouTube Channel]. Retrieved from https://www.youtube.com/channel/UC5xDht2blPNWdVtl9PkDmgA

[15] On the order of 80% of software development costs are related to maintenance due to changing requirements. See Kniesel, et. al. (2002) "Unanticipated Software Evolution." In J. Hernandez and A. Moreira (Eds.): ECOOP 2002 Workshops, LNCS 2548, pp. 92–106.

[16] "The fundamental problem, supported by 40 years of hard experience, is that many changes actually required are those that the original designers cannot even *conceive* of." Bennet, K.H. and Rajlich, V.T. (2000) "Software Maintenance and Evolution: A Roadmap." *Future of Software Engineering*. pp. 75-87.

[17] https://build.fluidproject.org/infusion/demos/prefsFramework/

a part) is an effort to build a software framework that supports an open authorial ecosystem in which any software expression can be modified, refined, and replaced by consecutive authors, without breaking the linkages and connections amongst this network of expressions[18].

Material metaphors for software, of course, have distinct limitations. Software, we might imagine, is a "material of the mind," yet one which is expressed in a uniquely computational form—infinitely reproducible, demanding of precision and detail. At least in theory, highly pliable. And yet anyone who has, for example, diligently attended to a complex writing project knows that this may not be literally the case. Ideas, in system, are complex and even sometimes inconceivable outside of the context in which they were originally situated and elaborated. Software is not yet, and perhaps never can be, a craft[19]. Its material qualities may be far too different, ultimately more subject to change and systematic contingencies than artefacts in the physical world. Nonetheless, these metaphors from the physical and craft worlds may help us to see more clearly the potentialities in the medium that we have overlooked or hidden away in computation's dominant formalistic and neo-Romantic creative constructions.

5. Conclusion

So how can we start to come to terms with change, to give it space within our software designs as an opportunity, not just a risk? There are, of course, no silver bullets; no easy-to-follow checklists or master methodologies that will produce software that has the resilience, flexibility, and longevity that is needed to meet and include users where they are, rather than continuing to force them to adapt and compromise. Two interrelated strategies, described here as continuing creativity methods, suggest potential ways to fully embrace change. First, narrowing the gap between use and design via co-design, particularly by engaging diverse users as equals in the process from the beginning. Secondly, creating software tools, programming frameworks, and authoring environments that will increasingly support users in modifying and redesigning them, even after the software has shipped. These approaches represent an early and speculative collective work-in-progress, which will undoubtedly benefit from continued experimentation, exploration, mistake-making, and active participation by "users," designers, and programmers alike.

[18] See Basman, A. et al. "The Open Authorial Principle" and Basman, A. et al "Software and How it Lives On – Embedding Live Programs in the World Around Them."

[19] Basman, Antranig. (2016). "Software is Not Yet A Craft." http://www.ppig.org/sites/default/files/2016-PPIG-27th-Basman2.pdf