# Estratégias de design para contextos complexos

*Design strategies for complex contexts*

ALÃO, Rui; Prof. Dr.; Centro Universitário Belas Artes

ruialao@gmail.com

## Resumo

Dada a crescente complexidade dos problemas de design, é comum hoje projetarmos para contextos complexos que apresentam variáveis cambiantes e perspectivas imprevisíveis. Pretendemos, com este artigo, levantar algumas propostas no campo da metodologia projetual que tentam contemplar este tipo de desafio e propõem algumas mudanças para os paradigmas tradicionais de projeto. Dois campos do projeto são explorados na tentativa de encontrar elementos de projeto convergentes: o do metadesign e o do desenvolvimento de software *open source*.

**Palavras Chave:** complexidade, metodologia, design

## *Abstract*

*Given the increasing complexity of the design problems, it is usual to design for complex contexts featured with inconstant variables and unpredictable prospects. In this paper we want to bring forward some proposals in the field of design methodology that try to cope with this type of challenge and that propose some changes for the traditional design paradigms. Two different fields are covered in the hope of finding convergent design elements: metadesign and the open source software development.*

***Keywords:*** *Complexity, methodology, design*

## 1 Complexidade crescente dos problemas de design

Os problemas colocados para os designers atuais são de uma natureza diversa daqueles colocados há três ou mais décadas. Não se trata apenas de uma mudança na natureza da dificuldade dos problemas, que acompanha o avanço das tecnologias e recursos hoje disponibilizados, mas uma mudança mais profunda, no próprio caráter da sociedade como um todo, nas relações humanas e na forma como se produz conhecimento.

Estas mudanças são frequentemente menosprezadas, mas basta voltarmos o nosso olhar para a vida há 50 ou 100 anos para que estas características sejam evidenciadas. Há cem anos a maioria da população mundial vivia na área rural. Fazendeiros vivam em relativo isolamento, tinham os mesmos tipos de instrumentos e ferramentas que seus vizinhos. O ambiente no qual se trabalhava era constante. As relações entre trabalho e resultado eram lineares, previsíveis, estáveis. As relações pessoais e comerciais se davam no âmbito local. A velocidade da troca de informações entre partes distantes do mundo era lenta. Praticamente todos os aspectos da vida se davam no contexto local.

A vida hoje é muito diferente, não vivemos mais em uma comunidade com ligações esparsas entre seus membros. Formaram-se redes de troca de informação e de interdependência que antes não haviam. Temos muito mais *diversidade*, *conectividade*, *interdependência* e *adaptação* em vários aspectos de nossas vidas, como destaca Scott Page (2009), professor da Universidade de Santa Fé ligado às ciências da complexidade. Estas são, não por coincidência, as quatro características fundamentais dos sistemas complexos adaptativos. Estes sistemas são compostos de inúmeros agentes que atuam influenciando-se mutuamente, adaptando-se a novas configurações e criando laços de interdependência. O que um agente faz influencia o que outro pode ou deve fazer. O argumento de Page é, portanto, o de que vivemos hoje num ambiente muito mais complexo que o de cem anos atrás.

Este ambiente e suas características são transpostos, é claro, para os problemas que os designers enfrentam. A complexidade traz consigo um certo grau de incerteza, já que são tantos os fatores que entram em jogo em um sistema complexo.

Projetar para este tipo de ambiente cheio de incertezas é um desafio colocado por Silvia Pizzocaro, pesquisadora do Politecnico di Milano, através das seguintes perguntas:

> Quais são as formulações da complexidade que se adequam à indústria? Como isto se relaciona com os processos de design emergentes? Como as empresas aprendem a crescer em um contexto de complexidade? Qual é o impacto das formulações da complexidade na capacidade de inovar de uma organização? Como a complexidade pode ajudar no aumento de a inteligência de uma organização? Como esta compreensão pode ajudar a área de pesquisa em design? (PIZZOCARO, 2000, p. 249)[1]

O ambiente e o mercado para o qual o designer projetava há algumas décadas praticamente

[1] As citações foram traduzidas pelo autor do artigo. No original: What are the appropriate industry-related formulations of complexity? How do these relate to emergent design processes? How do corporations learn to grow in a complexity context? What is the impact of complexity formulations in the capacity of an organization to innovate? How can complexity help increasing organization intelligence? How can this understanding help in design research? Now is the time when we need a way of evaluating of what comes next, when we face a world that has gone in a very short period of time from seemingly linear (simple) to complex and non-linear (chaotic). When we move into a world that is inherently more complex, the result is concussive, its disorientating effects surround us, and our responses either individually or at an organisational level result in reflexes and perspectives that can be dangerously corrosive or inappropriate.

não existe mais. Projeta-se hoje levando em conta simultaneamente o contexto em tempo real, pois o mercado muda muito rapidamente, e uma estimativa de cenários futuros, tentando prever mudanças de comportamento, o movimento de concorrentes e projetando estratégias de longo prazo.

As organizações devem aprender rapidamente a se adaptar a lidar com estes contextos. Este movimento gera, frequentemente, adaptações no interior das próprias organizações, que passam a se articular de forma mais maleável e a quebrar algumas de suas formas tradicionais de organização. As hierarquias de caráter puramente *top-down* começam a dar lugar a articulações mais soltas e flexíveis. A previsão de cenários, antes calcadas em pesquisas de mercado assertivas, agora contam com outras ferramentas de trabalho, como o monitoramento de redes sociais, para acompanhamento em tempo real de suas ações. As empresas que antes estavam acostumadas a lidar com um contexto de trabalho e de mercado relativamente previsíveis, cujo comportamento podia ser representado através de gráficos lineares, agora tem que se adaptar à não linearidade constante, à imprevisibilidade dos mercados, à inconstância dos contextos.

> Agora é o momento em que precisamos de uma forma de avaliar o que vem a seguir, quando encaramos um mundo que passou, em um curto período de tempo, de linear (simples) para complexo e não-linear (caótico). Quando nos movemos em um mundo que é inerentemente mais complexo, o resultado é contundente, seus efeitos desorientadores no rodeiam e nossas reações, tanto individuais quanto nas organizações, resultam em reflexos e perspectivas que podem ser perigosamente corrosivas e inadequadas. (MOORE, 2011, ln. 129)[2]

A atuação do designer que projeta para um ambiente altamente variável e imprevisível deve se adaptar a esta nova realidade. Uma questão que surge é o fato de que a maioria dos designers não foi preparada para lidar com estas realidades complexas. A maior parte dos profissionais da área aprendeu, em sua formação acadêmica, através do paradigma que os teóricos chamam de *design by drawing*, isto é, o projeto através do desenho (JONES, 1992; LAWSON, 2006).

Este processo envolve a representação em papel das soluções de design, num *loop* que alterna entre a geração de um modelo mental feito pelo designer e sua representação. A representação, por sua vez, gera alterações no modelo mental feito pelo designer, que gera uma nova iteração, processo descrito por Schön (1983) como o "estabelecimento de uma conversa com o desenho". Esta conversa é renovada e, conforme se desenrola e se aproxima de uma solução viável, atravessa uma série de etapas consagradas pela prática. Na arquitetura, por exemplo, estas etapas são frequentemente estabelecidas como estudo preliminar, o anteprojeto e o projeto executivo. Dependendo da complexidade e da escala do projeto, podem haver pequenas variações, mas o processo tem sempre uma forte tendência à linearidade.

Esta prática, no entanto, embora dê conta de uma gama enorme de projetos, não consegue enfrentar problemas complexos pois a representação em papel raramente consegue aglutinar todos os aspectos dos sistemas complexos.

Para enfrentar problemas complexos usando os métodos tradicionais, não resta alternativa senão a simplificação do problema em uma versão mais simples, a qual o método de *design by*

[2] Now is the time when we need a way of evaluating of what comes next, when we face a world that has gone in a very short period of time from seemingly linear (simple) to complex and non-linear (chaotic). When we move into a world that is inherently more complex, the result is concussive, its disorientating effects surround us, and our responses either individually or at an organisational level result in reflexes and perspectives that can be dangerously corrosive or inappropriate.

*drawing* possa enfrentar. No caso dos problemas complexos, esta simplificação, no entanto, corre o risco de retirar do problema a sua principal característica: a interdependência entre seus múltiplos agentes. O filósofo Paul Cilliers trata exatamente deste problema:

> O poder da tecnologia abriu novas possibilidades para a ciência. Uma das ferramentas mais importantes têm sido o método analítico. Se algo é muito complexo para ser apreendido de uma só vez, é dividido em partes mais gerenciáveis as quais podem ser analisadas separadamente e novamente agrupadas mais tarde. No entanto, o estudo dos sistemas complexos revelou uma falha fundamental no método analítico. Um sistema complexo não é constituído apenas pela soma de seus componentes, mas também pelas relações intrincadas entre estes componentes. Ao cortar um sistema em pedacinhos, o método analítico destrói o que ele deseja entender. (CILLIERS, 2000, pp. 1-2)[3]

A necessidade de adequação entre uma prática projetual geralmente linear e a não-linearidade crescente dos problemas de design é evidente. Temos, portanto, um método *top-down* para um problema que é largamente *bottom-up*, ou seja, surge das múltiplas interações entre inúmeros agentes.

O processo de *design by drawing*, portanto, não é o suficiente para dar conta dos aspectos não-lineares dos problemas contemporâneos de design. Projetar para contextos complexos e cambiantes parece pedir um processo projetual igualmente complexo, pois se o ambiente que vai receber o resultado do processo projetual é mutável, o projeto não pode simplesmente se contentar com uma solução que atende a apenas um momento desta flutuação. Em resumo, se o contexto é complexo, o produto do processo projetual também deve dar conta desta complexidade.

## 2 Projetar para contextos complexos

Gostaríamos, a partir desta exposição inicial, de elencar algumas propostas de sistematização de pesquisadores da área de design que enfrentam a questão de projetar para um contexto complexo. Num segundo momento, tentaremos destacar elementos comuns a estas propostas.

Vamos iniciar pelas propostas conceituais de metadesign de Elisa Giaccardi e de Greg Van Alstyne e Robert Logan, que, apesar de distintas, guardam entre si bastantes pontos em comum para figurarem juntas. Em seguida vamos expor o método de criação de software *open source* através do seu mais famoso manifesto, o texto de Eric Raymond *The Cathedral and the bazaar*. Estaremos longe de esgotar o tema, já que existem outras propostas em discussão mas, para o escopo deste artigo, pensamos ser o bastante para instigar a reflexão do enfrentamento de tais problemas no cenário contemporâneo da área.

### 2.1 O metadesign de Elisa Giaccardi e Greg Van Alstyne

A pesquisadora Elisa Giaccardi, da Delft University of Technology, propôs em sua tese de doutorado e em vários artigos subsequentes uma abordagem na qual o designer pudesse, ao invés de projetar diretamente o produto de design, projetar o ambiente no qual tal produto pudesse ser gerado. A este paradigma deu o nome de metadesign pois, ao invés do designer projetar a solução,

[3] The power of technology has opened new possibilities for science. One of the most important scientific tools has always been the analythical method. If something is too complex to be grasped as a whole, it is divided into manageable units which can be analysed separately and then put together again. However, the study of complex systems has uncovered a fundamental flaw in the analytical method. A complex system is not constituted merely by the sum of its components, but also by the intricate relationships between these components. In 'cutting up' a system, the analytical method destroys what it seeks to understand.

ele se ocupa de projetar o processo de design. Seria, portanto, um projeto do projeto de design. A ideia é a de construir um ambiente complexo o bastante para que este possa construir, através de múltiplas interações, o produto desejado.

> Metadesign é um ambiente conceitual emergente direcionado para a definição e criação de infraestruturas sociais e técnicas nas quais novas formas de design colaborativo podem surgir. Ele estende a noção tradicional de design de sistema para além do desenvolvimento original para incluir um processo coadaptativo entre usuários e o sistema, onde os usuários se tornam codesenvolvedores ou codesigners. (GIACCARDI e FISCHER, 2004)[4]

O metadesign também assume desde sua conceituação que não é plausível o designer conhecer totalmente todos os aspectos do problema no momento do projeto, já que estes são complexos demais e sempre estão em mutação. Assim, é mais adequado projetar uma solução aberta que possa evoluir junto com o próprio problema.

> Faz parte das premissas básicas que usos e problemas futuros não podem ser completamente antecipados no momento do design, quando um sistema é desenvolvido. Usuários, no momento do uso, descobrirão descompassos entre suas necessidades e o suporte que um dado sistema pode fornecer. Estes descompassos podem levar a colapsos que servirão como fonte potencial de novos insights, novos conhecimentos e novos entendimentos. (GIACCARDI e FISCHER, 2004)[5]

Gregory Van Alstyne, cofundador do Strategic Innovation Lab (sLab) no OCAD, Ontario College of Art & Design e fundador do Departamento de New Media do MoMA em Nova York, e Robert Logan, professor emérito da Universidade de Toronto, propõem o uso de fenômenos emergentes dentro do processo de design.

Eles comparam diretamente os conceitos de design e emergência pois não existe, para eles, um processo de design que incorpora fenômenos emergentes, mas sim o próprio processo emergente como equivalente do processo de design, criando novas formas e soluções a partir de processos *bottom-up*.

> Nós propomos que o design deve cultivar o processo de emergência: pois é somente através do desenvolvimento de emergência massivamente iterativo e *bottom-up* que novos e melhores produtos e serviços são refinados adequadamente, introduzidos e difundidos no mercado. (VAN ALSTYNE e LOGAN, online)[6]

Assim como Giaccardi, Van Alstyne e Logan defendem o design participativo, no qual o designer deve criar, antes de tudo, uma comunidade ao redor da ideia de solução, caso esta não exista. Esta comunidade então é que deve chegar à solução de design.

> Uma característica da emergência é o envolvimento de agentes múltiplos e autônomos que, no caso de um design inovador, se traduz em uma comunidade de usuários. O designer inovador deve, portanto, ”projetar com” uma comunidade existente ou tentar construir

[4] Metadesign is an emerging conceptual framework aimed at defining and creating social and technical infrastructures in which new forms of collaborative design can take place. It extends the traditional notion of system design beyond the original development of a system to include a co-adaptive process between users and a system and a system, and the users become co-developers or co-designers.

[5] It is grounded in the basic assumption that future uses and problems cannot be completely anticipated at design time, when a system is developed. Users, at use time, will discover mismatches between their needs and the support that an existing system can provide for them. These mismatches will lead to breakdowns that serve as potential sources of new insights, new knowledge, and new understanding.

[6] We propose that design must harness the process of emergence; for it is only through the bottom-up and massively iterative unfolding of emergence that new and improved products and services are successfully refined, introduced and diffused into the marketplace.

uma comunidade ao redor da nova ideia. (VAN ALSTYNE e LOGAN, online)[7]

O metadesign, portanto, prega a criação de um ambiente no qual o problema se insira e, a partir da implementação deste ambiente, o problema deve ser tratado. Está embutido nesta proposta construir, portanto, uma instância (o ambiente ou a comunidade) tão complexa quanto o problema a ser tratado, para que uma complexidade possa dar conta da outra. A sobreposição de problema e solução é a base sobre a qual as propostas de metadesign se implementam. Veremos, a seguir que este processo também acontece nas propostas de desenvolvimento de software livre.

## 2.2 A catedral e o bazar

Não é comum considerar o projeto de software como uma área do design, como a arquitetura, o desenho industrial ou gráfico. Mas, nos tempos atuais, estas fronteiras têm sido questionadas. Com a transposição dos conteúdos impressos para as plataformas digitais, o designer gráfico ou o programador visual se viram frente ao desafio de projetar não simples cartazes ou revistas, mas em fazer os conteúdos dentro de um projeto editorial se comunicarem entre si. Em poucas palavras, ao migrar para suportes mais flexíveis, o design digital teve que lidar com uma complexidade informacional natural do suporte digital.

E logo vieram os portais, logo depois os blogs e redes sociais, e hoje estamos passando por uma fase na qual o próprio software, antes habitante dos computadores pessoais, passa a morar na rede. Ao invés de usar o software localmente, como era tradicional no pacote Office da Microsoft, hoje usa-se um editor de texto ou uma planilha eletrônica que são processados em sites como o Google Drive (drive.google.com) ou o Office Live (office.live.com). Na medida em que os sites ficam mais “espertos”, ou seja, que lhes é embutido novos recursos inteligentes, é necessário lidar mais com a programação do que com o layout, como ocorria no início da web.

Desde sempre o designer digital lidou com código. Inicialmente era um código apenas de marcação como o HTML (Hypertext Markup Language), que cuidava de traduzir a hierarquia gráfica e informacional dos documentos impressos para a nova plataforma digital. Agora, no entanto, ele é chamado a programar sites inteiros, a fazê-los mais “responsivos”, a fazer o site “reagir” à intervenção do usuário.

Na medida em que isto acontece, o designer digital tem que lidar, ele também, com o desenvolvimento de software, não porque a web tenha se enamorado do universo do software, mas simplesmente porque o software migrou para a rede.

Assim, entendemos ser natural comparar processos projetuais de objetos, edifícios e máquinas com processos projetuais de software se quisermos refletir sobre o campo contemporâneo do design. É também natural que esta área de projeto tenha entrado em contato mais rapidamente com a complexidade de que falamos no início deste artigo, já que faz parte do universo do software modelar processos humanos e técnicos e trazer as suas relações, quase sempre múltiplas e complexas, para um fluxo de processamento. Deste modo, entendemos que a área de gerenciamento de desenvolvimento de software tem muito a contribuir para as metodologias de design em suas áreas de tradição.

A obra *The Cathedral and the Bazaar*, de Eric Raymond (2001) compara duas formas de

[7] One characteristic of emergence is the involvement of multiple, autonomous agents, which in the case of innovative design translates into a community of users. The innovative designer should therefore “design into” an existing community or seek to build a community around a new idea.

desenvolvimento de software. O título se refere ao contraste entre a catedral, como paradigma de obra planejada, estática e imutável, construída de forma hierárquica e *top-down*, e o bazar, construção improvisada, flexível e aparentemente desorganizada. Ao perceber que havia já, na prática de projeto de software, uma técnica de projeto não ortodoxa com o projeto do sistema operacional LINUX, Raymond resolveu tentar entender quais as características deste tipo de projeto que fazem com ele prospere e demonstre várias vantagens em relação ao que havia sido feito nas grandes corporações.

Embora ele faça menção a um programa chamado Fetchmail, cujo processo de desenvolvimento usou para testar as lições aprendidas com as práticas *open source*, o principal projeto no qual baseia suas observações é o desenvolvimento do sistema operacional Linux, capitaneado por Linus Torvalds.

Esta produção se define pela colaboração em larga escala de indivíduos que se agregam voluntariamente na produção de software de forma auto-organizada, isto é, sem que haja um forte controle *top-down*.

Bem, as grandes empresas, até meados da década de 90, sempre utilizaram paradigmas de projeto de software que vinham de projeto de objetos físicos. Uma pequena equipe de desenvolvimento elaborava modelos de software, os quais, quando chegavam a um certo grau de maturidade, eram chamados de versão *alpha*. Estes eram postos em teste geralmente por uma equipe interna, que resultava em *feedback* para a equipe de desenvolvimento. Esta implementava ajustes e melhorias até chegar a um modelo usável. Este modelo era usualmente chamado de versão *beta*. O software era então distribuído a um grupo restrito de usuários (os chamados *beta testers*) que geravam um novo ciclo de *feedback* e ajustes correspondentes. Finalmente, chegava-se a uma versão RC *(Release Candidate)* que, com a devida maturação, era lançada como versão definitiva ao público em forma de um pacote.

A distribuição de software, até então, era feita através de disquetes ou CDs. Mas o software era ainda considerado um produto e distribuído em caixas; sua venda era feita em lojas de software. Todo este ciclo era bastante longo, já que deveria assegurar que o software poderia ser usado nas mais diferentes situações e contextos possíveis, por usuários com backgrounds diferentes e em computadores com os mais diferentes hardwares.

A forma de desenvolvimento de software que surgiu com a elaboração do sistema Linux era diferente em vários aspectos. A equipe de desenvolvimento era, ao invés de contratada, voluntária: pessoas ao redor do mundo doavam seu tempo e expertise para desenvolver o sistema. As equipes eram voláteis: desenvolvedores que estavam ligados a uma parte do projeto, por um motivo ou outro, se desligavam e eram substituídos por outros, que tinham que se inteirar do projeto na medida em que produziam código. As versões do software eram disponibilizadas ao público muito cedo, enquanto ainda haviam muitos *bugs* (problemas de software) para teste. A distribuição era feita, no mais das vezes, através da própria rede, por meio de downloads.

O código é aberto, isto é, a qualquer momento um desenvolvedor poderia pegar o código feito por centenas de programadores e desenvolver uma versão paralela, e chamar outros para o ajudar. Daí as múltiplas distribuições dos sistemas Linux até hoje: Arch, CentOS, Debian, Fedora, Gentoo, Linux Mint, OpenSUSE, Red Hat, Slackware, Ubuntu para citar algumas.

Este modo aparentemente caótico de gerenciamento de software, com adesões e deserções

fortuitas, a despeito das inúmeras críticas quanto à sua confiabilidade, gerou o sistema operacional mais usado como servidor web e que roda em 90% dos computadores mais rápidos do mundo.

Descobrir, portanto, como funcionou historicamente o gerenciamento usado por Linus Torvalds era, ao mesmo tempo, entender uma metodologia de desenvolvimento que conseguia lidar com recursos variantes, com objetivos abertos.

Depois de fazer a sua análise deste modo de desenvolvimento, Raymond chegou a alguns "mandamentos" do desenvolvimento *open source*.

### 2.3 Release early, release often

Ou seja, disponibilize (o *software*) cedo e frequentemente. A ideia por trás da frase é a de que é fundamental colocar o produto em contato com seu público. No caso do software, em contato com seus usuários finais. Assim, deve-se disponibilizar o software o quanto antes para o usuário, mesmo que isto acarrete em grande número de versões disponíveis no mercado ao mesmo tempo. Mesmo que o produto não esteja acabado e perfeito ele deve ser colocado à disposição do usuário para que haja feedbacks quanto à falhas e propostas de melhorias e complementações. O desenvolvimento de software facilita este tipo de implementação já que pode ser distribuído como serviço e desenvolvido em módulos intercambiáveis.

A questão fundamental aí pode ser colocada da seguinte forma: em um ambiente estável, previsível, talvez faça sentido esperar pelo amadurecimento do produto para depois disponibiliza-lo. Já em um ambiente imprevisível e instável, o produto nunca estará pronto. A própria noção do que vem a ser um produto pronto se perde, se levamos em consideração que o ambiente que o receberá é instável. Temos, assim, a evolução constante do software em resposta ao ambiente mutável ao ele está exposto.

### 2.4 Given enough eyeballs, all bugs are shallow

Ou seja, se tivermos um grande número de olhos, todos os bugs (problemas de código) ficarão expostos. Em seu relato, Eric Raymond optou por desenvolver um programa que estava já em andamento ou invés de iniciar um projeto próprio. Um dos motivos de sua decisão foi o fato de poder contar com a comunidade de usuários pré-existente.

> E então herdei Popclient. Tão importante quanto foi o dado de herdar sua base de usuários. Usuários são coisas maravilhosas de se ter, e não somente por mostrar que você está servindo a um propósito, que fez algo direito. Cultivados apropriadamente, eles podem se tornar co-desenvovedores. (...) Tratando os seus usuários como co-desenvolvedores é a rota menos atribulada para rápida melhoria de código e um processo de debug efetivo. (RAYMOND, 2001)[8]

O grupo de usuários que se junta em torno de um projeto *open source* é, portanto, um elemento de sucesso do projeto já que, ao contrário do usuário tradicional, que simplesmente consome o software passivamente, o típico usuário de software *open source* dá uma grande vitalidade ao ciclo de desenvolvimento do software. O projeto é lançado ao grupo de usuários ainda em desenvolvimento e estes, como disse o autor, funcionam como co-desenvolvedores pois não só encontram erros como também sugerem novos recursos aos desenvolvedores. A comunicação

[8] And so I inherited Popclient. Just as importantly, I inherited pop-client's user base. Users are wonderful things to have, and not just because they demonstrate that you're serving a need, that you've done something right. Properly cultivated, they can become co-developers. (…)Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging

entre a comunidade de usuários *open source* e seus desenvolvedores é intensa. O número massivo de desenvolvedores e usuários é, portanto, fundamental.

### 2.5 O beta eterno

A radicalização do discurso do *"release early, release often"* resulta no "beta eterno", isto é, no desenvolvimento contínuo de software, sem versionamento.

Como dissemos antes, os softwares passam por etapas de desenvolvimento típicas: alpha, beta, RC. Hoje, no entanto, esta prática se encontra radicalizada. É uma tendência atual a de se disponibilizar software ainda em versão *beta* para expô-lo o quanto antes ao maior número de usuários possível, para acelerar o ciclo de desenvolvimento, fazendo com que os usuários sejam guardiões da qualidade do software que usam e forneçam feedback para novos ciclos de desenvolvimento. Diferentemente do que acontecia antes, quando o software beta rapidamente evoluía para RC e depois para versão definitiva, hoje é comum o software ficar em beta por semanas, meses ou anos. O Gmail, por exemplo, ficou em beta por quatro anos.

> Seguindo o exemplo do Google, muitas companhias grudam "beta" em seus logos e deixam ele lá por meses ou anos. Já longe se vão os dias em que betas eram lançados a um grupo limitado de usuários conhecidos, com entrevistas formais de avaliação. O conceito de "beta" como um período de tempo ou estágio de desenvolvimento não mais se aplica, e foi substituído por beta como uma forma de estabelecer expectativas, ou desculpar falhas no estágio atual da aplicação. (HEDLUND, online)[9]

Assim, o uso comum do beta eterno é um efeito da necessidade de aceleração do desenvolvimento de software e é uma radicalização da prática do *"release early, release often"*.

> O dito *open source*, "lance cedo e lance sempre", na verdade transformou-se em uma posição ainda mais radical, "o beta eterno", no qual o produto é desenvolvido abertamente, com recursos implementados num ciclo de meses, semanas ou mesmo dias. Não é coincidência que serviços como Gmail, Google Maps, Flickr, del.icio.us e outros provavelmente retenham o logo "beta" por anos. (O'REILLY, online)[10]

As fases de desenvolvimento, assim, se diluem. Dado que um aplicativo já é usado amplamente pelo seu público final em sua fase beta de desenvolvimento, e continua sendo desenvolvido ininterruptamente, talvez não haja mais sentido no versionamento de software, isto é, na atribuição de um número ou nome de controle de sua versão. Deve-se assumir que o desenvolvimento é contínuo, e que sofrerá mudanças com mais ou menos frequência dependendo da atividade que o projeto comportar. Em um produto convencional, talvez, a tendência seria a do ritmo de mudanças se estabilizar e depois cair, na medida em que o produto se adequa às solicitações dos usuários. Mas no caso de produtos digitais, o desenvolvimento tende a continuar por muito tempo, já que existem exigências dinâmicas que fazem com que o aplicativo que hoje é adequado, amanhã não seja.

No caso do Gmail, por exemplo, o surgimento de novos dispositivos (smartphones e tablets)

[9] Following Google's lead, many companies stick "beta" on their logos and leave it there for months or years. Gone are the betas that get released to a limited set of known but external testers, with formal product management follow-up interviews. The concept of "beta" as a time period or stage of development has fallen away, and been replaced with beta as a way of setting expectations, or excusing faults, about the current state of the application.

[10] The open source dictum, "release early and release often" in fact has morphed into an even more radical position, "the perpetual beta", in which the product is developed in the open, with new features slipstreamed in on a monthly, weekly, or even daily basis. It's no accident that services such as Gmail, Google Maps, Flickr, del.icio.us, and the like may be expected to bear a "Beta" logo for years at a time.

ao longo dos anos 2000 fez com que o projeto fosse revisto continuamente.

Com a implementação do chamado "beta eterno" fica claro que a linearidade do projeto não existe mais. A própria noção de produto é questionada uma vez que este muda a toda hora. O que há é um propósito que permeia todo o projeto. Novamente, no caso do Gmail, este propósito é gerenciar uma conta de e-mail. Ao redor deste propósito são integrados novos serviços na medida em que estes surgem e podem doar ao aplicativo novos recursos. Em termos de metodologia de projeto, o produto se eclipsa, e o que há é um processo de projeto que nunca se estabiliza, pois se move em resposta ao ambiente, também instável, dentro do qual se insere.

## 3 Considerações

Algumas características são encontradas tanto no paradigma do metadesign quanto no universo de desenvolvimento de software *open source.* Elas nos parecem ser comuns a qualquer solução que se proponha a lidar com problemas complexos em design e lidar com suas demandas variáveis. Entendemos que estas características são desejáveis para qualquer projeto que se proponha a lidar com problemas complexos, que mudem com o passar o tempo, em curto prazo, e que sejam dependentes de múltiplas variáveis que fogem ao controle imediato do projetista. Estas características são, portanto, desejáveis para qualquer projeto que lide com este tipo de complexidade. Nesta lista de qualidades repousa a utilidade de nosso artigo para futuros projetos de contexto complexo.

Vamos, a seguir, elencar estas características.

### 3.1 Produto em contato com usuário

Dos exemplos colocados, fica claro a importância o fato de que o produto, seja ele qual for, tangível ou não, deve ser colocado em contato com o seu público o mais rápido possível. Não nos surpreende que esta estratégia seja usada já que, sendo o ambiente complexo, o produto deve se movimentar de modo a ajustar-se a esta constante variação. Do ponto de vista do desenvolvimento, quanto antes este produto for colocado à disposição, antes se obterá um feedback a seu respeito, e novas iterações de projeto podem acontecer. Este é mais um fator que desaconselha o uso de um paradigma de projeto linear, com etapas pré-definidas, com começo, meio e fim, sem iterações e retomadas.

### 3.2 Usuário como coautor

Em segundo lugar vem o fato de se considerar o usuário como coautor. Como vimos, autores tanto do campo do metadesign quanto do desenvolvimento de software são explícitos quanto ao fato de que o usuário é parte integrante do processo projetual. Um projeto sem um público definido e atuante — isto é, um público com o qual se possa contar — não poderá se adequar a um ambiente complexo. Para este tipo de problema, portanto, o usuário deixa de ter o perfil de simples receptor passivo de um produto acabado para ser o crítico e eventualmente o coautor de um produto que se re-projeta a cada ciclo. O pesquisador australiano Axel Bruns fala mesmo de um novo perfil de usuário, chamado de *produser*, junção do perfil de *producer* (produtor) e *user* (usuário). (BRUNS, online).

### 3.3 Diluição das fases de desenvolvimento

A colocação do produto em contato com o usuário, como vimos, funciona para que haja um contínuo feedback para o re-design, em ciclos de projeto que avançam junto com a variação típica

de um ambiente complexo. Estas sucessivas iterações projetuais configuram uma diluição das fases de desenvolvimento tradicionais. Um processo que antes era linear e tinha começo, meio e fim, agora tende a ser repetido inúmeras vezes, em espiral, principalmente nos produtos — que aos poucos se tornam serviços — digitais e intangíveis. A cada novo ciclo projetual o produto tende a se atualizar de acordo com novos *inputs* por parte dos usuários e de outros fatores variáveis do ambiente.

### 3.4 Uma complexidade para dar conta da outra

O conceito mais interessante, no entanto, quando pudemos investigar estas propostas de incorporação da complexidade nos projetos de design é, em nossa opinião, o de que é necessário criar uma instância de complexidade de projeto para atender a uma complexidade do ambiente para o qual se projeta. Ou seja, se o ambiente é complexo, cambiante, ou, como quer Pizzocaro (2000), turbulento, é fundamental construir uma outra complexidade projetual que venha de encontro a ele. Um projetista solitário, ou mesmo uma equipe, não é suficiente.

Esta noção fica clara quando Eric Raymond fala da contribuição do desenvolvimento do sistema operacional Linux para os novos paradigmas do desenvolvimento *open source*.

> A inovação de Linus não está muito em fazer lançamento rápidos incorporando feedback dos usuários (algo parecido já era tradição no mundo Unix por muito tempo), mas em escalar isto a um nível de intensidade que se adequava à complexidade do que estava sendo desenvolvido. (RAYMOND, 2001)[11]

Na área de pesquisa específica em design também há um entendimento quanto à evolução conjunta entre problema e solução de design. Para o tratamento deste tipo de problema, os pesquisadores Kees Dorst e Nigel Cross (2001) falam em “espaço de problema” e “espaço de solução”, numa referência ao fato de que tanto um quanto outro são, desde o princípio, indeterminados, e podem sofrer deslocamentos durante o processo de projeto. Não haveria, portanto, a fixação de um problema para a busca de sua solução, mas sim um mapeamento de espaços de convergência entre pares problemas-soluções.

Esta noção, nos parece, também parece apontar para uma convergência entre complexidades ao invés da imposição de soluções definidas para problemas cujos contextos mudam à toda hora.

> Dorst e Cross (2001) haviam tentado encontrar uma forma de chegar a uma descrição mais próxima da solução dos problemas indeterminados usando um estudo empírico que descreve a co-evolução do problema de design e da solução de design. Se olharmos mais de perto a criação de soluções para problemas indeterminados de design, ela parece ser um processo muito mais gradual, como uma evolução. Parece que o design criativo não é algo onde primeiro se fixa o problema (através de análise objetiva ou a imposição de um modelo) para depois buscar um conceito de solução satisfatória. O design criativo parece mais ser algo que se desenvolve e refina tanto a formulação do problema quanto as ideias para uma solução, com constante iterações no processo de análise, síntese e avaliação entre os dois “espaços” de design – espaço do problema e espaço da solução. No design criativo, o designer busca gerar um par problema-solução que se encaixe, através da co-evolução do problema e da solução. Nossas observações confirmam que o design criativo envolve um período de exploração no qual os espaços de problema e de solução estão

[11] Linus's innovation wasn't so much in doing quick-turnaround releases incorporating lots of user feedback (something like this had been Unix-world tradition for a long time), but in scaling it up to a level of intensity that matched the complexity of what he was developing.

> evoluindo e são instáveis até (temporariamente) fixados por uma ponte emergente que identifica um pareamento entre problema e solução. (DORST, online)[12]

Finalmente, parece-nos que estas características levantadas apontam para uma mesma direção, a de que os problemas complexos na área de design não podem ser tratados através do processo tradicional e linear de design. O paradigma do *design by drawing* não consegue dar conta da complexidade dos problemas em questão. Para lidar com contextos de design complexos é necessário sofisticar também o processo projetual.

As soluções aqui levantadas apontam para o estabelecimento, desde a primeira hora, de um contato direto com o ambiente no qual estes produtos serão usados para que haja uma espécie de acomodação mútua entre problema e solução. Quem pode estabelecer esta sincronia é a comunidade de usuários e os demais contextos de cada projeto. A dinâmica fornecida por ciclos de feedback faz desta comunidade híbrida de usuários/designers a instância na qual o problema se coloca. É através dela que qualquer solução deve passar. A postura que os autores tratados no âmbito deste artigo parecem endereçar é a de que, apesar da área do design trazer problemas extremamente complexos, diversos e pouco determinados, o afastamento dado pela atitude do metadesign, a de contemplar o "projeto de design" (e não o produto) como um problema passível de análise por parte dos designers, pode trazer novos *insights* à área.

## 4 Referências

BRUNS, Axel. **Welcome to produsage**. 2007. Disponível em <http://produsage.org> Acessado em agosto de 2011.

CILLIERS, Paul. **Complexity and postmodernism**. London: Routledge, 2000.

DORST, Kees. *The problem of design problems*. In: **University of Technology Sydney**. Disponível em <http://research.it.uts.edu.au/creative/design/papers/23DorstDTRS6.pdf> Acessado em junho de 2015

GIACCARDI, Elisa; FISCHER Gerhard. *Metadesign: a framework for the future of end-user development*. In: **End user development: empowering people to flexibly employ advanced information and communication technology**. Dordrecht, The Netherlands: Kluwer Academic Publishers, 2004.

HEDLUND, Marc. **Web Development 2.0**. In: O'Reilly Radar Disponível em <http://radar.oreilly.com/2006/02/web-development-20.html> Acessado em março de 2018.

JONES, John Chris. **Design Methods**. New York: John Wiley & sons. 1992

[12] Dorst and Cross (2001) have tried to find a way to arrive at a closer description of underdetermined problem solving by using an empirical study to describe the co-evolution of the design problem and the design solution. If we take a closer look at the creation of solutions to underdetermined design problems, it seems to be a much more gradual process, like an evolution. It seems that creative design is not a matter of first fixing the problem (through objective analysis or the imposition of a frame) and then searching for a satisfactory solution concept. Creative design seems more to be a matter of developing and refining together both the formulation of a problem and ideas for a solution, with constant iteration of analysis, synthesis and evaluation processes between the two notional design 'spaces' - problem space and solution space. In creative design, the designer is seeking to generate a matching problem-solution pair, through a 'co-evolution' of the problem and the solution. Our observations confirm that creative design involves a period of exploration in which problem and solution spaces are evolving and are unstable until (temporarily) fixed by an emergent bridge which identifies a problem-solution pairing.

LAWSON, Bryan. **How designers think. The design process demystified.** Oxford: Elsevier. 2006.

MOORE, Alan. **No straight lines.** Cambridge: Bloodstone Books, 2011.

O'REILLY, Tim. **What is web 2.0**. In: O'Reilly Radar. Disponível em <http://www.oreilly.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html?page=4> Acessado em maio de 2016

PAGE, Scott. **Understanding Complexity**. Chantilly: The Great Courses. 2009.

PIZZOCARO, Silvia. *Complexity, uncertainty, adaptability: Reflections around design research*. In: **Doctoral education in design: Foundations for the future**. London: Staffordshire University Press, 2000.

RAYMOND, Eric. **The Cathedral and the Bazaar**. Sebastopol: O'Reilly Media, 2001. Kindle edition.

SCHÖN, D. A. **The reflective practitioner: how professionals think in action.** London: Temple Smith. 1983.

VAN ALSTYNE, Greg; LOGAN, Robert. *Designing for Emergence and Innovation: Redesigning design.* In: **ARTIFACT***. Disponível em* <http://scholarworks.iu.edu/journals/index.php/artifact/article/view/1360/1390> Acessado em junho de 2017.