Cognição e Técnica no desenvolvimento de um programa computacional em uma plataforma online

Victor Vieira Paulo[1]

Resumo: A ubiquidade da Internet vem suscitando interesse em várias áreas do conhecimento a respeito dos impactos da mesma sobre a cognição humana. Uma das linhas de pesquisa a este respeito volta-se para a investigação das formas como as tecnologias digitais têm propiciado a realização de atividades cognitivas por coletivos conectados pela Internet e compostos tanto por seres humanos como por artefatos técnicos. Tais desenvolvimentos apresentam grande interesse para a antropologia, já que dirigem atenção para questões clássicas da disciplina, como as relações entre a cognição humana e os contextos socioculturais onde ela opera, e também para questões atuais, como a relação entre humanos e não humanos. Este artigo busca tratar destas questões através da descrição etnográfica do processo de desenvolvimento de um programa computacional de código aberto realizado por pessoas geograficamente dispersas conectadas pela plataforma online GitHub, voltada para o suporte deste tipo de atividade.

Palavras chave: Cognição, Técnica, Github.

Introdução

As questões a respeito da cognição humana estão entre os problemas fundacionais da Antropologia. O grau de universalidade ou particularidade das capacidades cognitivas entre os membros da espécie humana e a relação da cognição com os contextos

[1] Mestrando no programa de Pós-graduação em Antropologia Social/UFSC. Bolsista CNPQ. Membro do Coletivo de Estudos em Ambientes, Percepções e Práticas (CANOA).

socioculturais em que ela opera são questões encontradas nos escritos dos expoentes clássicos da disciplina, desde os evolucionistas culturais até Lévi-Strauss. De fato, o trabalho deste último autor fornece uma das inspirações iniciais para a pesquisa da qual trata este texto.

Em *A estrutura dos Mitos* (1955), Lévi-Strauss argumenta que não existem diferenças qualitativas entre os aparatos cognitivos dos primitivos e aqueles dos atuais cientistas. A fim de conciliar essa noção de um mecanismo cognitivo universal com as aparentes diferenças nos produtos da atividade intelectual das diversas populações humanas, o autor lança uma interessante hipótese, segundo a qual as diferenças entre o modo de pensar dos primitivos e aquele dos modernos seriam causadas não por capacidades mentais díspares, mas sim por diferenças nos objetos materiais sobre o qual esses modos de pensar incidem.

A hipótese de Lévi-Strauss aponta para um novo programa no estudo da cognição humana, no qual a investigação passa a incidir sobre a relação entre a mente e o mundo material no qual (e por meio do qual) ela opera. Algumas das perspectivas teóricas mais recentes no campo da antropologia cognitiva tem levado adiante tal programa. Dentre estas, cabe destacar o trabalho de Edwin Hutchins, antropólogo cognitivo americano. Numa passagem que se assemelha a uma radicalização da hipótese de Lévi-Strauss, Hutchins (1995, p.169, tradução nossa) afirma que "Os humanos criam seus poderes cognitivos ao criar os ambientes nos quais eles exercem esses poderes."[2]. Neste trecho observa-se não apenas a proposição de uma relação mais "forte" entre cognição e o mundo material do que aquela encontrada em Lévi-Strauss, mas também o estabelecimento de um vínculo entre cognição e técnica, já que a técnica é o meio essencial através do qual os seres humanos modificam o mundo a seu redor.

A relação entre cognição e técnica postulada certamente é válida para qualquer período da história humana, e para populações em quaisquer níveis de desenvolvimento tecnológico. No entanto, ela ganha especial significância no período atual, onde o desenvolvimento das tecnologias da informação, e em especial da Internet, tem gerado significativas mudanças nos ambientes socioculturais (e portanto materiais) nos quais grande parte dos seres humanos vivem. De fato, a ubiquidade destas tecnologias tem motivado um intenso debate acerca de seus impactos sobre a cognição humana(SMART, 2013).

[2] No original: "Humans create their cognitive powers by creating the environments in which they exercise those powers."

Tal debate apresenta significativas oportunidades para a antropologia, na medida em que permite tratar tanto de questões clássicas da disciplina como de algumas das linhas de inquérito mais contemporâneas. Para além do já citado tema da relação entre cognição e o mundo material, questões a respeito do caráter social dos processos cognitivos também têm sido trazidas a tona, dado que o avanço das tecnologias da Web tem apresentado forte ênfase na criação de plataformas que suportam a coordenação de pessoas geograficamente dispersas para a realização de atividades de natureza cognitiva (SMART, 2013).

Inspirado por questões desta natureza, me propus a realizar uma pesquisa etnográfica a respeito do desenvolvimento de um programa computacional de código aberto através de uma plataforma online. A questão central da pesquisa, na qual se origina este artigo, diz respeito às formas pelas quais o ambiente da plataforma e os diversos artefatos técnicos a ele associados impactam o trabalho cognitivo de elaboração do programa. Este campo etnográfico foi selecionado por dois motivos. Em primeiro lugar, a atividade a ser analisada trata-se de uma forma de trabalho cognitivo que ocorre através da Internet e envolve múltiplas pessoas e artefatos técnicos, o que permite o tratamento das questões de interesse. Em segundo lugar, a natureza do desenvolvimento de código aberto demanda que os esforços de desenvolvimento sejam realizados em uma página de público acesso na Internet, expondo a prática de desenvolvimento à observação do pesquisador.

Neste artigo buscarei descrever um processo básico do desenvolvimento de código aberto: a inclusão de novas contribuições de código fonte ao programa. Seguindo os objetivos da pesquisa, a descrição apresenta especial ênfase no papel que os artefatos assumem neste processo. O material etnográfico examinado inclui documentos que definem o processo de tratamento de contribuições utilizado na criação do programa em questão, registros da operação deste processo disponibilizados pela plataforma online e código fonte dos artefatos técnicos mobilizados (já que eles mesmos são programas computacionais).

A próxima seção busca esclarecer a natureza do software cujo desenvolvimento é alvo desta pesquisa.

A linguagem Rust

O projeto de desenvolvimento de software de código aberto a ser descrito é o da linguagem de programação **Rust**. Linguagens de programação são linguagens formais

utilizadas para criar algoritmos computacionais, que consistem em conjuntos de instruções a serem executadas por computadores para a resolução de tarefas específicas. Uma rápida pesquisa em um motor de busca qualquer revela a existência de centenas (ou mesmo milhares) de linguagens de programação, e várias outras que estão atualmente em processo de elaboração.

A elaboração de uma linguagem de programação geralmente envolve a escrita de um documento de especificação e a criação de uma implementação de referência. O documento de especificação descreve os principais componentes da linguagem, geralmente divididos entre sintaxe e semântica. Enquanto a sintaxe se refere à forma da linguagem, definida pelas regras que especificam como conjuntos de símbolos podem ser combinados para criação de expressões sintaticamente corretas, a semântica se refere ao significado que tais expressões possuem em termos do tipo de instruções computacionais que elas engendram. Já a implementação de referência é o programa computacional responsável por executar programas escritos na linguagem em questão. Além disso, é comum que os desenvolvedores de uma dada linguagem forneçam também alguns segmentos de código que implementam funcionalidades ou ferramentas básicas que podem ser utilizados para desenvolver programas na linguagem. Estes ferramentas ou funcionalidades básicas são geralmente chamados de Biblioteca padrão da linguagem[3]. O desenvolvimento da linguagem Rust envolve todos esses aspectos, voltando-se para elaboração e melhoria da especificação dos seus componentes, da sua implementação de referência, e da biblioteca padrão da linguagem.

Código aberto

Rust é uma linguagem de **código aberto**. A noção de código aberto diz respeito sobretudo a um modelo de desenvolvimento de software, que opera em oposição ao software proprietário. O **código fonte** de um programa computacional consiste na definição, em uma linguagem de programação, do conjuntos de instruções computacionais que compõem tal programa. No modelo de software proprietário, o código fonte de um programa é propriedade exclusiva da empresa ou organização que produz o software.

[3] O termo biblioteca é utilizado para designar uma coleção de código que implementa uma dada função (ou um conjunto de função relacionadas), e oferece uma interface para que outros programas possam utilizá-la para executar esta função.

A possibilidade de acessar, examinar ou alterar este código é restrita aos membros de tal organização, enquanto os usuários do programa recebem apenas uma cópia executável do software. Por outro lado, um software de código aberto é distribuído sob licenças de propriedade que permitem sua livre apropriação, modificação e mesmo redistribuição. Isto significa que qualquer um que deseje realizar uma modificação no funcionamento de um programa de código aberto pode obter uma cópia do código fonte, alterá-la e enviar o código modificado para os desenvolvedores do software, que podem ou não aceitar essa modificação no código de seu projeto. Mesmo que não aceitem, ainda é possível ao propositor da modificação distribuir o software modificado por ele como uma versão alternativa do projeto principal. A ampla disponibilização do código fonte de um programa computacional cria a possibilidade de uma forma colaborativa de desenvolvimento, na qual qualquer um (desde que dotado das capacidades necessárias) pode engajar-se na criação e aperfeiçoamento do software. Como a disponibilização do código tende a acontecer em páginas públicas na Internet, cria-se a possibilidade de engajamento de uma grande número de contribuidores voluntários. No entanto, cabe ressaltar que a inclusão dos contribuidores voluntários não exclui a presença de programadores que recebem remuneração de alguma organização para dedicarem-se ao desenvolvimento de um programa de código aberto. É relativamente comum que projetos deste tipo dotados de grande envergadura sejam levados adiante por uma mistura de contribuições voluntárias e trabalho remunerado. De fato, o desenvolvimento de Rust segue este padrão.

A linguagem Rust nasceu em 2006 como um projeto pessoal do programador Graydon Hoare, e no ano de 2009 passou a ser patrocinada pela empresa Mozilla, empregadora de Hoare. Após um período inicial no qual o desenvolvimento foi realizado exclusivamente no interior da empresa, o código fonte da implementação de referência e os documentos relativos ao desenvolvimento da linguagem foram publicamente disponibilizados em uma plataforma online utilizada para desenvolvimento de programas computacionais chamada GitHub. A partir deste momento, o esforço de desenvolvimento passou a ser levado adiante através deste canal, e o projeto passou a receber também contribuições de código fonte de voluntários, para além daquelas realizadas pelos desenvolvedores empregados pela Mozilla.

Dado que GitHub ocupa um lugar central na configuração das práticas através das quais a linguagem Rust é desenvolvida, torna-se necessário tratar do funcionamento desta plataforma.

GitHub

Em certo sentido plataforma GitHub pode ser descrita como um ambiente que objetiva fornecer condições propícias para a operação de projetos de desenvolvimento de software. Como já dito, o modelo de desenvolvimento de código aberto baseia-se na ampla disponibilização do acesso ao código fonte, em geral através da sua publicação em páginas da Internet. A plataforma online GitHub é um dos vários serviços criados com a finalidade de facilitar o processo de publicização de código fonte, e permitir que programadores engajem-se nessa modalidade de desenvolvimento.

GitHub é um serviço de hospedagem de código fonte que oferece a seus usuários a possibilidade de criação de repositórios para armazenamento do código fonte de programas. Estes repositórios são compostos basicamente por arquivos computacionais divididos em diretórios, e podem ser de acesso privado ou público. Os repositórios de desenvolvimento de código aberto evidentemente recaem sobre a segunda categoria, já que eles não permitem apenas que todos os usuário da plataforma tenham acesso ao código armazenado, mas também que estes enviem contribuições de código fonte para o repositório, cabendo ao administrador do repositório em questão aceitar ou rejeitar a contribuição.

Embora cada projeto de desenvolvimento de software de código aberto estabeleça diretrizes particulares sobre o procedimento de envio e recebimento de contribuições, os projetos hospedados no GitHub costumam estabelecer seus procedimentos de contribuição a partir das funcionalidades que a plataforma oferece para esta finalidade. Sendo assim, a plataforma estabelece um certo nível de padronização na forma como os repositórios contidos nela administram o recebimento de contribuições de código fonte. Esse fator aparentemente contribui para a inclusão de novos colaboradores nos projetos da plataforma, já que qualquer usuário que for capaz de dominar as funcionalidades de GitHub terá um leque de milhares de possibilidades de projetos para contribuir. Esta padronização baseada nas funcionalidades oferecidas pela plataforma acaba por determinar os modelos de envio e aceitação de contribuições usados nos projetos hospedados no GitHub.

Como já dito, cabe aos administradores de um repositório determinar quais modificações serão aceitas e incorporadas no repositório, e quais serão rejeitadas. Isso ocorre pois apenas o administrador tem a permissão para executar modificações no repositório de código fonte. Qualquer outro usuário que não seja o administrador pode examinar o

código, mas não modificá-lo. O administrador pode ceder as suas permissões em relação ao repositório aos usuários que desejem contribuir, permitindo que eles modifiquem-no diretamente, sem necessidade de aprovação. Caso essa opção seja realizada, configura-se o modelo de contribuição chamado **Repositório compartilhado,** no qual todos os contribuidores tem permissão para modificar o código do repositório.

Este modelo é muitas vezes usado em repositórios privados, mas não é o mais comum para repositórios de código aberto, já que seria pouco prático exigir que aqueles que desejam contribuir tenham que pedir permissões de acesso antes de poder enviar suas contribuições, e também indesejável permitir que todos os contribuidores possam modificar o repositório diretamente. Por esta razão os repositórios de código aberto do GitHub – inclusive o da linguagem Rust – costumam adotar outro modelo, chamado **Fork And Pull.** Ao contrário do modelo de repositório compartilhado, o Fork And Pull não requer que os autores de contribuições tenham permissão de modificação de acesso ao repositório principal, facilitando a entrada de novos voluntários e garantindo maior controle aos administradores do repositório.

O usuário do GitHub que deseja realizar uma contribuição no modelo Fork And Pull começa por fazer um **fork** do repositório em questão. Isto é, ele utiliza uma funcionalidade da plataforma que permite criar um repositório GitHub que é uma cópia daquele do repositório para o qual se deseja contribuir, mas sob posse do autor do fork. Para realizar as modificações no código presente no Fork, o usuário deve fazer o download do repositório para seu computador, utilizando uma ferramenta chamada Git. Git é um software que permite a criação de repositórios de arquivos computacionais em um dado computador, facilita a administração destes repositório, e permite a comunicação entre diferentes repositórios localizados em computadores distintos. De fato, o GitHub foi construído sobre a base fornecida pelas funcionalidades do Git. Uma vez que possua o repositório em seu computador, o usuário pode realizar as modificações desejadas, e enviar utilizar novamente o Git para enviar o repositório modificado novamente para o fork no GitHub. A partir daí, a versão modificada do código se torna a versão principal contida no fork, e a versão pré-modificação é arquivada[4].

[4] É importante notar que as versões anteriores dos arquivos de código fonte contidas em um repositório do GitHub nunca são descartadas. O GitHub – ou, mais especificamente, o sistema Git sobre o qual ele foi construído – garante que todas as versões de cada arquivo contido em um repositório permanecem armazenadas por tempo indefinido, fornecendo um histórico completo das mudanças realizadas nos arquivos.

Tendo operado as modificações em seu fork, o usuário pode enviar sua contribuição para o repositório para o qual deseja contribuir por meio de mecanismo da plataforma GitHub chamado **Pull Request** (doravante PR)**.** Ao iniciar um PR, o autor de modificações em um fork emite um pedido para os administradores do repositório principal para que as alterações contidas em seu fork sejam incluídas neste repositório. Esse pedido não apenas gera uma notificação para o administrador do repositório principal, mas cria automaticamente uma página da web na qual podem ser visualizadas as alterações no código fonte sendo enviadas, bem como uma descrição e explicação dessas alterações feita pelo autor do PR no momento de sua criação. Nesta página, o administrador do repositório pode lançar mão de uma série de funcionalidades providas pelo Github para realizar comentários a respeito da contribuição em questão, revisar o código exigindo mudanças para que ele seja integrado ao repositório, operar essa integração, ou rejeitar as alterações encerrando o PR. Por sua vez, o autor do PR pode prestar esclarecimentos e defender sua proposta de modificação através de comentários ou realizar alterações no código submetido no PR mediante demanda dos administradores. Outros usuários do GitHub que não são nem autores de um PR e nem administradores do repositório também tem acesso aos PR de um repositório (e podem até mesmo emitir comentários), já que a página de cada repositório público do GitHub contém uma seção com os links dos PRs associados. Caso o PR seja aceito, a versão modificada do código presente no fork é adicionada ao repositório principal, passando a ser a versão vigente deste repositório, e a versão anterior é arquivada.

Após esta descrição genérica dos processos de envio e aceitação de contribuições normalmente usados pelos projetos de desenvolvimento de software hospedados no GitHub, buscarei tratar das práticas de desenvolvimento da linguagem Rust, evidenciando as particularidades da modalidade de contribuição utilizada neste âmbito.

Organização do desenvolvimento de Rust

Embora a discussão até aqui tenha se focado no desenvolvimento da linguagem Rust, é preciso assinalar que a atividade dos envolvidos no projeto Rust não se circunscreve à elaboração e aprimoramento dos já citados componentes de uma linguagem de programação. Pelo contrário, parte significativa do trabalho realizado no projeto Rust envolvem atividades que servem de suporte não apenas ao desenvolvimento da

linguagem, mas também a sua difusão e uso em projetos de desenvolvimento de software. Um exame da forma com os desenvolvedores da linguagem se organizam em equipes pode ajudar a ilustrar esse ponto.

O site oficial da linguagem Rust[5] cita um total de 10 times de desenvolvedores associados a diferentes áreas de atuação no projeto Rust. Enquanto alguns deles, tais como o Compiler team (responsável pela implementação de referência da linguagem) e o Language team (responsável por desenvolver novas funcionalidades e características para a linguagem), operam diretamente no desenvolvimento de Rust, existem outras que realizam tarefas de natureza bastante diferente. Exemplos deste último caso incluem o Dev tools team, envolvido no desenvolvimento de ferramentas para facilitar o trabalho dos programadores que pretendem utilizar Rust para desenvolver programas, e até mesmo o Community team, responsável por coordenar e dar apoio aos eventos e atividades que agregam a comunidade de desenvolvedores e usuários de Rust.

Os times costumam conter ou um mais líderes e por volta de uma dezena de membros, sendo frequentemente divididos em subtimes que atuam em segmentos circunscritos da área de atuação do time principal. Existem ainda unidades organizacionais chamadas de grupo de trabalho, usualmente voltadas para o desenvolvimento de projetos de natureza mais específica, que podem ou não ser associados a times. É frequente que desenvolvedores atuem em mais de um time, ou mais de um subtime, e a participação em um subtime ou grupo de trabalho não tem por requisito a participação no time ao qual este está atrelado. Para além de sua atuação no GitHub, que constitui o foco deste artigo, os times também possuem canais de comunicação (alguns privativos, outros disponíveis para não membros não membros) como listas de e-mail ou chats online. Embora parte significativa dos membros dos times sejam empregados da empresa patrocinadora do projeto Rust, este não é um requisito necessário.

Essa multiplicidade de segmentos no interior do projeto Rust impacta sua organização na plataforma GitHub. Os principais componentes da linguagem, incluindo sua implementação de referência, biblioteca padrão e documentação sobre seu funcionamento, são desenvolvidos em um único repositório[6]. No entanto, o projeto Rust

[5] https://www.rust-lang.org/. Acessado em 19/04/2018.

[6] https://github.com/rust-lang/rust. Acessado em 19/04/2018.

conta com uma grande quantidade de outros repositórios que contém conteúdos dos mais diversos e sediam várias modalidades de atividade. Enquanto alguns repositórios contém segmentos de código fonte associados ao desenvolvimento de Rust (mas separados de seu repositório "principal"), outros contém o código de softwares que visam facilitar o uso da linguagem para a confecção de programas, e outros ainda contém documentos organizacionais dos times (tais como atas das reuniões realizadas) ou sediam processos de tomada de decisão a respeito de propostas de mudança nas características e funcionalidades da linguagem[7]. Neste artigo será descrito apenas o processo de contribuição utilizado no repositório que contém a implementação de referência da linguagem, já que não seria possível dar conta da multiplicidade de formas pelos quais esse processo é realizado em cada um dos repositórios do projeto Rust.

O processo de contribuição

Como é comum em repositórios do GitHub, o repositório da linguagem Rust contém entre seus arquivos um documento que explica o processo de contribuição, e fornece as normas que devem ser seguidas por aqueles que desejam realizar uma modificação ou adição no código contido no repositório[8]. Neste documento, a especificação das etapas do processo de contribuição apresenta significativa ênfase no detalhamento de como devem ser manejadas as ferramentas técnicas utilizadas em cada etapa. A fim de investigar o papel que tais ferramentas possuem no desenvolvimento de Rust, a descrição do processo de contribuição será dividida em seções que tratam da forma como as ferramentas técnicas citadas no documento são usadas durante a realização de uma contribuição.

[7] Embora os repositórios da plataforma GitHub sejam usados principalmente para o armazenamento de arquivos de código fonte, a tecnologia empregada pela plataforma permite que quaisquer arquivos computacionais possam ser armazenados em seu repositório. Além disso, as funcionalidades disponibilizadas pela plataforma para o desenvolvimento coletivo de arquivos de código fonte funcionam igualmente bem para o tratamento de quaisquer arquivos de natureza textual.

[8] O foco deste artigo se volte para o processo de envio e recebimento de contribuições de código fonte, mas é importante notar que as normativas de contribuição do repositório de Rust discorrem sobre vários tipos possíveis de contribuição. A submissão de relatórios de possíveis erros encontrados no funcionamento de Rust ou a escrita de documentação técnica sobre o funcionamento da linguagem são consideradas contribuições necessárias, tendo seu processo especificado no documento em questão.

Tidy

A primeira das ferramentas de software citadas no documento entra em uso antes do envio da contribuição de código fonte para o repositório. O guia de contribuição indica que o usuário de Github que pretende contribuir com a linguagem Rust deve começar pelos passos básicos do modelo de Fork and Pull: criação de um fork pessoal do repositório da linguagem, utilização do sistema Git para download do conteúdo deste fork para seu computador, realização das modificações almejadas, e envio do conteúdo modificado para seu fork no GitHub. Entretanto, é pedido que antes deste envio o usuário execute um programa chamado Tidy, contido em um dos arquivos do repositório que foi baixado para seu computador. Este programa é responsável por examinar todos os arquivos de código contido no repositório e verificar se eles se conformam a uma série de normas, que incluem: Padrões de escrita de código na linguagem Rust[9] (ex: Nenhuma linha de código deve ultrapassar 100 caracteres), restrições quanto ao tipo de arquivo que pode ser incluído no repositório (Ex: arquivos binários, como imagens ou vídeos, são vetados), exigência de que todos os arquivos contidos no repositório estejam sob licenças de que permitam seu uso em aplicações comerciais, etc. Caso detecte alguma violação destas normas, o programa provê um aviso do problema o usuário, permitindo que este adeque sua contribuição antes de enviá-la. O documento indica ainda que o usuário utilize uma funcionalidade do sistema Git que permite que o programa Tidy seja executado automaticamente antes de cada envio do código modificado para o fork. Desta forma, o usuário é poupado não apenas do trabalho de checar a adequação do seu código as normas, mas também do trabalho de executar o programa que faz isto.

A função performada por Tidy permite observar um padrão recorrente no processo de desenvolvimento examinado: através da modificação do ambiente de desenvolvimento, os programadores de rust são capazes de criar barreiras contra a realização de modificações consideradas errôneas ou indesejáveis no objeto sobre o qual trabalham. A recomendação de que Tidy seja integrada à funcionalidade do sistema Git que permite sua execução automática demonstra um esforço para que as funções executadas pelas ferramentas não demandem qualquer tipo de ativação do programador para além da configuração inicial da ferramenta. Pelo contrário, o que se propõe é justamente fixar

[9] A implementação de referência de Rust é ela mesma escrita em Rust. Várias linguagens de programação são implementadas desta mesma forma.

no ambiente de desenvolvimento do contribuidor – seu computador pessoal – a barreira que o Tidy ergue contra certas modificações no repositório de Rust.

O exemplo de Tidy ilustra que a possibilidade de modificação do trabalho realizado pelo programador através da modificação do ambiente em que ele opera. Nesse caso específico, a modificação tem um caráter de limitação da ação humana, a fim de evitar cursos de ação indesejáveis na execução de uma dada tarefa – no caso da ferramenta em questão, trata-se de limitar a forma pelo qual uma pessoa pode modificar um elemento material[10].

Creio ser precisamente esse o ponto apresentado pelo antropólogo cognitivista Edwin Hutchins (1995, p.154) quando ele afirma que uma das funções providas pelas ferramentas na desempenho de tarefas cognitivas é a imposição de constrangimentos[11] na organização das ações. Tal função é importante na medida em que não apenas impede que erros sejam realizados na execução das tarefas, como também "externaliza" os custos cognitivos necessários para a manutenção da vigilância contra esses erros. O programador de Rust que configura seu ambiente de desenvolvimento para que o Tidy seja executado automaticamente antes do envio de uma contribuição livra-se do encargo de manter a disciplina e atenção necessárias para evitar realizar modificações não aceitas pelas normas de contribuição de Rust através do repasse da tarefa para um implemento material.

Além disso, é importante notar que não se trata apenas de evitar os custos cognitivos de seguir as normas, mas evitar até mesmo a necessidade de sabê-las. De fato, o documento de normas de contribuição é bastante sucinto ao citar o uso de Tidy, apresentando-a apenas como uma ferramenta que checa o código para ver se ele segue os padrões de estilo de escrita da linguagem Rust. No entanto, a análise do código de Tidy revela que o programa realiza diversas outras checagens, das quais apenas algumas foram citadas aqui. A inclusão da checagem automática abole a necessidade de que várias normas e convenções sejam explicitadas: o programador só é informado a respeito da

[10] Se a limitação pelo Tidy não parece demasiado rígida, sendo facilmente burlável ou contornável, vale notar que a sua inclusão no computador do programador serve apenas para que ele possa ser informado da violação das regras antes antes de submeter seu código, já que estas violações tem por consequência a reprovação automática da contribuição quando o Tidy é novamente executado para verificá-la (desta vez pelos sistemas acoplados ao repositório) após a submissão do código para a plataforma GitHub.

[11] Aqui o termo constrangimento foi adotado como tradução para a palavra inglesa *constraint*. A tradução infelizmente padece de imprecisões: o termo ao mesmo tempo deixa de captar todos os significados da palavra – que não se restringe ao sentido meramente negativo de impedimento, possuindo também um sentido positivo de estabelecimento de um limite cuja existência habilita a realização de uma ação – e capta significados não presentes – como a ideia de vergonha.

vigência da regra ao descumpri-la, já que só nessa ocasião receberá uma mensagem de erro emitida pelo software.

Highfive

Caso a contribuição se adeque às normas, o processo continua com o envio dos arquivos modificados para o fork, e a abertura de um Pull Request no repositório da linguagem Rust pedindo a integração das mudanças contidas no fork. As normas de contribuição estabelecem que cada cada Pull Request deve ter um revisor responsável por examinar o código e identificar possíveis problemas, sugerindo ou indicando soluções para o autor do PR, ou manifestar sua aprovação assim que julgar que a contribuição está pronta para ser integrada no repositório. A responsabilidade da seleção de um revisor cabe a uma ferramenta automatizada chamada Highfive. Este software interage com o repositório da linguagem Rust no GitHub através de um mecanismo fornecido pela plataforma. Quando um Pull Request é aberto no repositório Rust, o software é notificado pela plataforma GitHub, entrando em ação. Highfive identifica quais arquivos do repositório o PR deseja modificar, e a partir disso determina quais dos desenvolvedores da linguagem Rust podem ser selecionados como revisor. Para isso, o programa consulta seu arquivo de configuração para o repositório de Rust[12]. Este arquivo contém uma lista dos times que atuam neste repositório, e dos usuários GitHub dos programadores associados a cada time. Além disso, o programa possui uma associação entre cada time e os diretórios do repositório que contém arquivos sob responsabilidade do time. Portanto, ele é capaz de identificar qual time é responsável por atuar no diretório que contém os arquivos modificados pelo PR, e obter uma lista dos desenvolvedores associados àquele time. O programa então seleciona aleatoriamente um dos desenvolvedores dessa lista para ser o revisor do projeto. É possível também que a seleção de um revisor seja realizada pelo próprio autor do Pull Request, se ele assim o desejar. Para tanto, basta que o autor insira no texto de seu PR um comando para o software em questão indicando o usuário do Github de um dos desenvolvedores de Rust, no formato "r? @[usuário do GitHub]". Se isto for feito, Highfive apenas responsabiliza o usuário selecionado como revisor, sem realizar procedimento de seleção.

[12] O mesmo programa é utilizado em vários dos repositórios do desenvolvimento de Rust, e portanto possui um arquivo de configuração associado a cada repositório.

Após realizar a seleção ou receber o comando citado, Highfive volta a interagir com a plataforma GitHub utilizando uma funcionalidade que permite marcar um usuário do GitHub como responśavel pelo PR – o que indica que ele deve realizar as revisões. O revisor selecionado é comunicado instantaneamente sobre sua nova responsabilidade, dado que a plataforma Github emite uma notificação para o usuário que foi responsabilizado, e Highfive também é capaz de notificar automaticamente o usuário através do IRC, um dos canais de comunicação utilizados pelos desenvolvedores de GitHub.

O programa em questão faz a operação de responsabilização e notificação de um desenvolvedor através de sua própria conta de usuário GitHub, possuidora das mesmas características que um usuário utilizada por um humano, e listada entre os contribuidores do projeto Rust na plataforma GitHub (o que lhe dá permissão de modificação do repositório). Uma pessoa recém introduzida a plataforma GitHub teria mesmo dificuldade em distinguir humano e software, dado que os sinais visíveis deixados na tela de Pull Request pela ação deste programa são os mesmos gerados pela ação de um usuário humano quando este realiza as mesmas funções. Por este motivo, o sistema Highfive é considerado um **bot**, abreviação da palavra robot, termo utilizado para designar softwares que simulam ações humanas de forma repetida e padronizada na Internet.

Há algum tempo os cientistas sociais que pesquisam o desenvolvimento de código aberto tem dado destaque ao papel desempenhado pelos artefatos. Uma linha de pesquisa especialmente influenciada pela Teoria Ator-Rede argumenta que os artefatos utilizados no desenvolvimento "governam" a comunidades de desenvolvedores envolvida em um projeto de código aberto (PAOLI & D'ANDREA, 2008). Segundo esse argumento, um artefato é capaz de prover coordenação e organização para o trabalho dos programadores mediante a incorporação e operacionalização de regras que agem sobre o comportamento dos mesmos.

Ao tratarmos da operação do Tidy, já aludi ao fato de que esta ferramenta possui a função de detectar violações das normas que ditam como devem ocorrer as modificações nos arquivos do repositório, e que modificações não podem ser realizadas. Assim, nos termos destes autores, o Tidy pode ser considerado um artefato que incorpora diversas regras e age sobre a comunidade de desenvolvedores mediante a operacionalização destas normas.

Certamente o Highfive pode ser considerado um instrumento similar, embora dotado de uma função diferente. Em seus arquivos de configuração encontram-se

elementos cruciais para a organização e coordenação do trabalho, na forma de associações que indicam que grupos ou indivíduos são responsáveis por revisar e aceitar mudanças em determinados setores do repositório. Da mesma forma, seu código fonte contém o procedimento através do qual um dos indivíduos responsáveis por um setor do repositório é designado como revisor de um Pull Request. Essas associações e procedimentos compõe as regras inscritas neste elemento técnico. A operacionalização destas regras pelo bot permite que ele aja como uma espécie de gerente automatizado que distribui tarefas entre os seres humanos.

Bors

A ação do Highfive inicia a etapa de revisão. Como já dito, cabe ao revisor de um Pull Request a tarefa de examinar o código fonte que compõe a contribuição e demandar que o autor do PR faça as modificações consideradas necessárias. Se após este processo o revisor considerar que a contribuição deve ser integrada ao repositório, ele possui autonomia para realizar a integração. As normativas de contribuição detalham como esse processo deve acontecer, indicando novamente o uso de ferramentas de software para esse fim.

A própria plataforma GitHub oferece um meio simples de realizar a integração. Ao acessar a tela de um Pull Request, os usuários da plataforma tem a sua disposição um botão onde está escrito "Merge pull request". No jargão de GitHub, realizar um **merge** significa integrar o código de um Pull Request ao código já existente no repositório. Esse botão evidentemente só pode ser apertado pelos administradores dotados de permissão de modificação do repositório para o qual se dirige o PR.

Os desenvolvedores do projeto Rust escolheram não fazer uso desta funcionalidade oferecida pelo GitHub, optando novamente pelo uso de um bot para realizar a execução do merge. As normativas de contribuição esclarecem que o revisor de um Pull Request deve manifestar sua intenção de integrar o código em questão ao repositório através de um comando para um bot chamado Bors na forma "r+ @bors". Assim como Highfive, Bors é um software que opera em integração com a plataforma GitHub e possui uma conta de usuário Github que é utilizada para interagir nos Pull Requests. Essa conta de usuário possui as permissões necessárias para execução de modificações no repositório de Rust.

Ao receber o comando citado, o processo de merge se inicia. Em primeiro lugar, Bors realiza um comentário na tela do PR (simulando a ação de um humano), com uma mensagem indicando que o commit foi aprovado e citando o usuário que lhe deu o comando de aprovação. No entanto, a integração não é necessariamente executada imediatamente. Antes de integrar o código ao repositório Rust, Bors realiza uma série de testes automatizados. Esse tipo de teste configura uma prática comum no desenvolvimento de software, tanto de código aberto quanto proprietário, e tem a finalidade de garantir que as modificações realizadas no código de um software não terão por efeito a introdução de erros no funcionamento do mesmo. Os testes nada mais são do que programas computacionais capazes de executar o software sendo testado, inserindo determinados inputs neste e verificando se os outputs retornados se conformam ao que é considerado o comportamento esperado do mesmo.

Bors não executa estes testes automaticamente. Pelo contrário, ele coloca o programa em questão numa fila de Pull Requests que estão aguardando teste para finalização do merge. Os testes incluem muitas verificações diferentes, que são repetidas em cada um dos sistemas operacionais suportados por Rust[13]. Evidentemente os teste na fila são executados automaticamente por diversas ferramentas de software. Enquanto o teste se realiza, bors realiza um novo comentário no Pull Request, indicando que as checagens estão sendo performadas. Assim que o teste de um dado Pull Request é concluído, o bot verifica o resultado, e o comunica através de um comentário na plataforma GitHub. Se o teste falhou, o comentário reportará a falha e fornecerá um link para o sistema que contém os resultados do teste e a mensagem de erro, permitindo que o programador possua as informações para realizar as correções necessárias. Se, pelo contrário, o teste for bem sucedido, o comentário incluirá, além do resultado, o nome do desenvolvedor que aprovou a integração do código e uma mensagem indicando que o comentário está sendo integrado. Assim que o merge é finalizado, o status do PR na plataforma GitHub muda, e a tela do Pull Request aponta que o código foi integrado por Bors. A atividade incessante na realização dos merges tem garantido ao este bot o primeiro lugar no ranking do site do projeto Rust que apresenta os autores de contribuições no desenvolvimento

[13] A execução de testes em plataformas diferentes é possível porque os testes executados por Bors não são realizados no computador pessoal e um desenvolvedor, como Tidy, mas sim em servidores utilizados especificamente para as tarefas do desenvolvimento da linguagem Rust.

da linguagem Rust.[14] Como a lista é gerada automaticamente e leva em conta os merges realizados nos repositórios Rust, Bors é listado com mais de 14 mil contribuições, quase três vezes o número de contribuições sob autoria do segundo colocado (que, por sua vez, é um ser humano).

Bors apresenta uma característica semelhante a Tidy, na medida em que ele constitui uma ferramenta acoplada ao ambiente de desenvolvimento que põe limites sobre que tipo de código pode ser integrado ao repositório de Rust – afinal, os testes buscam justamente identificar se as mudanças contidas no Pull Request causam erros no funcionamento da linguagem, e barrar a integração de mudanças problemáticas.

Além disso, o funcionamento de Bors traz também outro ponto de interesse, que diz respeito à plataforma GitHub. Anteriormente, assinalei que a plataforma GitHub provê um certo nível de padronização do funcionamento dos processos de desenvolvimento em seus repositório, uma vez que estes processos se baseiam nas funcionalidades providas pela plataforma. O modelo de contribuição Fork and Pull utilizado nos repositórios de código aberto presentes em GitHub constitui um exemplo de procedimento padronizado. No entanto, a plataforma abre margem para o estabelecimento de modificações em seu funcionamento através da sua integração com outras ferramentas de software. A capacidade dos desenvolvedores de Rust de modificar o processo padronizado de merge mediante o uso do Bors está baseada na **Interface de Programação de Aplicações** fornecida pelo GitHub.

Ao acessar a plataforma GitHub através de seu navegador, o usuário tem acesso a uma interface gŕafica cuidadosamente desenvolvida para facilitar a visualização dos dados e execução das operações oferecidas pela plataforma. Esta interface, adaptada para as capacidades perceptuais e cognitivas de um ser humano, possui sua contraparte na Interface de Programação de Aplicações (IPA), modalidade de interface que representa os dados e dá acesso as operações de um programa numa forma que seja conveniente para um software. O Github oferece uma IPA de acesso público, e é essa ferramenta que permite que softwares como o Bors e Highfive interajam com a plataforma.

Através dessa possibilidade de integração com outros softwares, o ambiente padronizado da plataforma pode ser alterado de acordo com as necessidades do desenvolvimento praticado em um dado repositório, bastando para isso a utilização de uma

[14] https://thanks.rust-lang.org/rust/all-time. Acessado em 19/0/4/2019.

das muitas ferramentas de software disponíveis para integração com GitHub ou – como no caso dos bots citados – a criação de ferramentas específicas para as necessidades do repositório em questão.

Anteriormente argumentei que a plataforma GitHub facilita o agenciamento dos possíveis contribuidores humanos aos projetos de programas de código aberto, já que permite que eles contribuam para múltiplos projetos aprendendo a usar uma única ferramenta. Acredito que a exposição a respeito da IPA de GitHub evidencia que a plataforma também faz esforços para facilitar o agenciamento de não humanos – na forma de sistemas de software – como elementos constituintes do ambiente no qual ocorre o desenvolvimento. Ou mesmo, talvez, como membros da equipe de desenvolvimento – afinal, Bors encabeça a lista daqueles que mais contribuíram para o repositório de Rust. É comum que a plataforma GitHub seja descrita como uma rede social de programadores, na medida em que o objetivo da plataforma é prover um local no qual desenvolvedores possam se associar para a produção de programas. No entanto, a plataforma funciona também como uma rede social de programas, na medida em que busca prover formas pelas quais programas possam ser associados a ela e entre si mesmos, permitindo que eles sejam agenciados para a realização do trabalho performado nos repositórios.

A importância do agenciamento destes não humanos não deve ser subestimada, já que – como demonstra o caso do repositório de Rust – os processos e práticas de desenvolvimento são em grande parte configurados – e até mesmo "governados" – pelas ferramentas utilizadas na sua operação. Através da integração de novos sistemas ao ambiente da plataforma GitHub, os desenvolvedores de Rust são capazes de regular e organizar o trabalho humano necessário para o desenvolvimento da linguagem.

Conclusão

A exposição anterior evidencia o extenso papel desempenhado por elementos não humanos como sistemas de software no desenvolvimento de Rust. Foi demonstrado que estes sistemas desempenham uma série de funções necessárias ao desenvolvimento. Essas funções foram divididas em duas principais categorias: a imposição de limites na ação dos programadores – impedindo que eles realizem determinados tipos de modificações nos arquivos de código fonte – e a operacionalização de regras e procedimentos

que coordenam e organizam o trabalho humano. Enquanto Bors e Tidy recaem principalmente sobre a primeira categoria, Highfive e a própria plataforma GitHub, com seu modelo de Fork and Pull, exemplificam a segunda.

Além disso, as características do ambiente de trabalho fornecido pela plataforma GitHub foram ressaltadas. Tal ambiente desempenha um papel crucial no trabalho realizado no desenvolvimento de Rust, já que as suas funcionalidades ao mesmo tempo dão forma ao processo de contribuição adotado, e se permitem ser modificadas pela integração de elementos não humanos à atividade realizada no repositório.

Referências

HUTCHINS, Edwin. 1995. *Cognition in the Wild***.** Cambridge: Mit Press.

LÉVI-STRAUSS, Claude. 2009 (1955). A estrutura dos mitos. In: *Antropologia Estrutural*, São Paulo: Cosac Naify.

PAOLI, Stefano de, D'ANDREA, Vincenzo. 2008. "How artefacts rule web-based communities: practices of free software development". *International Journal Of Web Based Communities*, 4(2): 199-219, Disponível em: <http://www.inderscience.com/offer.php?id=17673>. Acesso em: 07 nov. 2018.

SMART, Paul. 2013. Understanding the Cognitive Impact of Emerging Web Technologies: A Research Focus Area for Embodied, Extended and Distributed Approaches to Cognition. *1st International Web for Wellbeing & Human Performance Workshop*, Paris.