Práticas Ágeis para o Desenvolvimento de Software Científico

Francisco Alves

2015

Resumo

A ciência computacional e a ciência dos dados tem ganhado relevância nos últimos anos a ponto de serem consideradas respectivamente o terceiro e quarto pilares da ciência, ao lado da teoria e da experimentação. Um dos pontos de consenso é a necessidade de que os pesquisadores envolvidos nestas áreas aprimorem suas habilidades computacionais, especialmente aquelas ligadas ao processo de desenvolvimento de software. O objetivo deste trabalho é identificar práticas ágeis de engenharia de software que pesquisadores podem adotar para o desenvolvimento de softwares científicos. Para isso foi realizado uma revisão de literatura de como pesquisadores desenvolvem software atualmente bem como uma pesquisa de quais práticas ágeis se mostrariam úteis tendo em vista o contexto atual. Os resultados indicam que pesquisadores podem se beneficiar de diversas práticas fundamentais de engenharia de software, como controle de versão e build automatizado, bem como de práticas ágeis, como backlog e desenvolvimento orientado a testes.

Palavras-chave: software científico. engenharia de software. manifesto ágil. práticas ágeis.

Introdução

A evolução da infra-estrutura computacional disponível para pesquisadores nos mais diversos campos do conhecimento tem gerado o desenvolvimento de duas novas disciplinas intrinsicamente relacionadas que visam utilizar estes recursos para a resolução de problemas, quais sejam, a ciência computacional e a ciência dos dados.

De acordo com PITAC (2005) a ciência computacional pode ser entendida como um campo de estudos interdisciplinar que visa a utilização de recursos computacionais avançados para o entendimento e resolução de problemas complexos. Ao contrário da ciência da computação, que visa o estudo da computação a partir de uma perspectiva científica, o foco da ciência computacional é a resolução de problemas de outras disciplinas científicas, o foco é a aplicação.

Por sua vez, uma das definições mais aceitas de ciência dos dados provém de Cleveland (2014) que utilizou o termo para se referir a uma proposta de expansão das áreas de interesse da estatística com foco na análise de dados. Dentre as áreas de expansão sugeridas encontram-se computação com dados e avaliação de ferramentas.

Estes dois campos estão tão relacionados e vem ganhando tanta força, que tem sido chamados, respectivamente, de terceiro e quartos pilares da ciência, ao lado da teoria e da experimentação. Apesar posições contrárias a inclusão desses campos como pilares da ciência, como aquela de Vardi (2010), é possível encontrar no mínimo dois pontos de unanimidade: 1) a ciência computacional e a ciência dos dados estão proporcionando avanço científico, econômico e social a ponto de serem consideradas uma revolução; 2) cada vez mais pesquisadores irão precisar de habilidades computacionais para aproveitar os novos recursos a disposição para resolução de seus problemas.

Este trabalho irá se debruçar sobre uma das componentes dessa revolução que se faz presente tanto na ciência computacional quanto na ciência de dados. O desenvolvimento de software. Mais especificamente, o tema deste artigo é a utilização de práticas ágeis de engenharia de software por pesquisadores para o desenvolvimento de softwares científicos.

A justificativa para este trabalho deriva da importância destes novos campos para o avanço científico e da constatação de que pesquisadores gastam uma parcela cada vez maior do seu tempo desenvolvendo software sem o treinamento adequado. (WILSON et al., 2014). Isso gera pesquisadores autodidatas que não utilizam práticas de engenharia de software que já se tornaram *mainstream* na indústria. O conselho de Gentzkow e Shapiro (2014) é extremamente válido nesse contexto:

> If you are trying to solve a problem, and there are multi-billion dollar firms whose entire business model depends on solving the same problem, and there are whole courses at your university devoted to how to solve that problem, you might want to figure out what the experts do and see if you can't learn something from it. (GENTZKOW; SHAPIRO, 2014, pg. 5).

Além disso, o processo de desenvolvimento de software científico aparenta ter características similares a aquelas endereçadas pelo manifesto ágil, como responsividade a mudança e colaboração (SLETHOLT et al., 2012), tornando essas práticas especialmente interessantes de serem estudadas para fins de utilização por pesquisadores. Deste modo, o problema de pesquisa desta trabalho pode ser formulado como: Quais práticas ágeis de engenharia de software são adequadas as necessidades dos pesquisadores que precisam desenvolver software no âmbito de sua pesquisa?

O objetivo geral deste trabalho é identificar quais práticas ágeis de engenharia de software podem ser utilizadas por pesquisadores para o desenvolvimento de softwares científicos. Para tanto, podem ser listados como objetivos específicos deste trabalho:

- Documentar as práticas de engenharia de software caracterizadas pela agilidade

- Avaliar a similaridade entre o contexto de trabalho de engenheiros de software e pesquisadores

- Identificar quais práticas podem ser utilizadas com poucas adaptações

- Identificar práticas que não podem ser utilizadas tendo em vista a diferença entre os contextos

Pode-se encontrar na literatura especializada incontáveis e absolutamente diversas classificações que se aplicam a metodologia. No presente trabalho, quatro critérios de classificação serão adotados: aquele em relação a natureza, em relação aos objetivos gerais da pesquisa, em relação a abordagem empregada – se qualitativa ou quantitativa, e aquele em relação aos métodos empregados.

Em relação a natureza trata-se de uma pesquisa aplicada que visa gerar soluções para problemas específicos relacionados a aplicação de práticas de engenharia de software para a pesquisa científica. A abordagem aplicada será qualitativa caracterizada pelo aprofundamento nas questões subjetivas do fenômeno em detrimento da produção de medidas quantitativas. Quanto aos objetivos será uma pesquisa exploratória tendo em vista que seu objetivo é buscar familiaridade com problemas pouco conhecidos. Quanto aos procedimentos técnicos será uma pesquisa bibliográfica especialmente por meio de artigos científicos que descrevam as práticas de engenharia de software e o contexto do desenvolvimento de softwares científicos.

Revisão da Literatura

Desenvolvimento de Software Científico

O objetivo desta seção é caracterizar o processo de desenvolvimento de software científico, com foco especial nas particularidades que existem em função do produto final (ie: software científico) e dos indivíduos envolvidos (ie: pesquisadores) neste processo.

O software científico pode ser entendido como aquele desenvolvido por pesquisadores com o objetivo de resolução de um problema em sua área de conhecimento de interesse. Apesar de simples, essa definição tem encontrado respaldo na literatura. Sletholt et al. (2012) define software científico como sendo aquele desenvolvido por cientistas e para cientistas, seja para a resolução de sistemas de equações matemáticas ou para geração de simulações. Heaton e Carver (2015) segue a mesma linha ao afirmar que software científico é aquele desenvolvido por cientistas e engenheiros para substituir ou expandir a experimentação física.

De maneira geral, um processo de desenvolvimento de software pode ser entendido como um conjunto de atividades relacionadas que levam a produção de um software. No caso da produção de software científico, a regra parece ser a ausência de um processo formal, com a realização de tarefas ad-hoc que geram o resultado desejado. Além da ausência de um processo formal, algumas das peculiaridades já documentadas na literatura dizem respeito ao levantamento de requisitos; a ausência de testes; a forma de relacionamento com o cliente; e ao treinamento recebido pelos responsáveis por desenvolver softwares científicos.

Em relação ao levantamento de requisitos, duas características principais se destacam. Em primeiro lugar, os requisitos de um software científico aparentam ser especialmente mais indefinidos no início de um projeto em comparação com softwares tradicionais. É usual que o problema de pesquisa de um cientista evolua ao longo de sua pesquisa, e, consequentemente, os requisitos acompanham essa transformação. Apesar de requisitos que emergem ao longo do tempo também existirem em contextos de softwares tradicionais, é razoável supor que ao menos o problema é mais bem definido e irá mudar menos em contextos tradicionais do que no científico. Além disso, uma segunda característica que explica as particularidades do levantamento de requisitos de softwares científicos deriva do fato de que os desenvolvedores, em grande medida cientistas, possuem vasto domínio de negócio. Isso implica que os requisitos podem ser levantados de forma extremamente genérica, tendo em vista que o indíviduo responsável pela implementação possui conhecimento suficiente para conceber o que deveria ser implementado em primeiro lugar. Nas palavras de um pesquisador que estava acostumado a desenvolver software e depois a especificar:

> So all I told one person [the developer] was: I want you to find a way of

> doing a ... fast graph matching problem, in an interface that is easy to use and shows you everything you need to know on the interface, and that's all I said. And he went away for a year and came back and here is the system. (SEGAL, 2008).

O processo de teste de software científico também apresenta diferenças marcantes em relação ao processo de teste de softwares tradicionais. Em primeiro lugar, uma das diferenças que será explorada adiante também ocorre no contexto de testes, qual seja, pesquisadores não são, via de regra, treinados em como testar software. Além disso, a característica exploratória de diversos softwares científicos faz com que seja difícil definir, *a priori*, qual o comportamento esperado de um programa. Por fim, o teste de algoritmos númericos de estimação apresentam complexidades inerentes a forma como computadores armazenam números flutuantes. As passagens abaixo capturam a essência de testes em um ambiente de software científico:

> Testing is of the cursory nature which would enrage a software engineer ('does the software do what I expect it to do with inputs of the type I would expect to use? If I don't have any real expectations of the output, does the software behave in a way I really would not expect given inputs of the type I would expect to use?'). (SEGAL, 2008).

Em relação ao relacionamento com os clientes, o *modus operandi* tradicional de desenvolvimento de software científico é que um pesquisador que possui familiaridade com o desenvolvimento de software, provavelmente em virtude de contato durante a elaboração da sua tese de doutorado, é responsável por desenvolver os softwares de um grupo de pesquisa ou de um laboratório. Ou seja, o desenvolvedor não só possui conhecimento da área de negócio, ele trabalha diariamente com os demais clientes.

A última grande diferença é relativa ao treinamento (ou falta dele) que os responsáveis pelo desenvolvimento de software científico usualmente possuem. Enquanto no caso do desenvolvimento de software tradicionais os responsáveis usualmente possuem treinamento específico em ciência da computação e engenharia de software, a realidade de pesquisadores é que eles nunca são treinados em como implementar software. Wilson et al. (2014) observa que estudos recentes mostram que cientistas tipicamente gastam 30% ou mais do seu tempo desenvolvendo software mas 90% ou mais são autodidatas, e, portanto, nunca foram expostos a práticas básicas de engenharia de software como controle de versão, revisão de código, teste unitário e automação de tarefas.

Em relação ao treinamento de cientistas, merece destaque o Software Carpentry <https://software-carpentry.org/>, iniciativa que visa treinar cientistas em habilidades computacionais básicas que permitem um ganho de eficiência por parte de pesquisadores envolvidos em computação científica.

Práticas Ágeis de Desenvolvimento de Software

As características apresentadas na última seção objetivaram mostrar as particularidades do processo de desenvolvimento de software científico que devem ser reconhecidas por qualquer prática de engenharia de software que pretenda ser efetiva no referido contexto. O objetivo desta seção é apresentar um conjunto de práticas que aparentam se moldar especialmente bem as necessidades de pesquisadores. A essas deu-se o nome de práticas ágeis.

O movimento de agilidade teve início formal com a publicação do manifesto ágil em 2001. Na ocasião, um grupo de 17 profissionais se reuniram com o objetivo de encontrar similaridades entre os processos de desenvolvimento de software advogados por cada um dos participantes. Todos as metodologias se apresentavam como uma alternativa aos processos cascata de desenvolvimento bastantes difundidos na época. A integra do manifesto é

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> Individuals and interactions over processes and tools
>
> Working software over comprehensive documentation
>
> Customer collaboration over contract negotiation
>
> Responding to change over following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more.

Além do manifesto, também foram publicados doze princípios que dão sustenção aos valores expostos pelos agilistas. No entanto, o objetivo deste trabalho é debruçar-se sobre as práticas ágeis, entendidas como as atividades concretas realizadas no dia a dia para desenvolvimento e gestão de projetos de software de maneira consistente com o manifesto ágil. A justificativa para essa abordagem deriva do entendimento que pesquisadores não irão se interessar pelo movimento ágil como um todo a menos que sejam capazes de observar melhorias concretas no seu dia a dia advindas das práticas aqui expostas.

As práticas ágeis listadas abaixo foram extraídas do Guia de Práticas Ágeis <http://guide.agilealliance.org/> publicado pela *Agile Alliance*. Quando necessário, foram pesquisados outras referências bibliográficas para melhor caracterização da mesma, bem como a forma pela qual as especificidades do software científico são bem atendidas. As práticas foram agrupadas de acordo com as principais particularidades existentes no processo de desenvolvimento de software científico conforme identificado na seção , além de um tópico adicional destinado a agrupar práticas que podem ser consideradas como boas práticas em engenharia de software em geral, e não necessariamente associadas ao movimento ágil.

Requisitos

Backlog

O *backlog* é uma lista com itens que representa todo o trabalho que deve ou pode ser realizado em um dado projeto. Ele apresenta uma característica evolutiva por natureza, ao representar tudo o que o software pode vir a ser com base nas definições advindas dos stakeholders (clientes, usuários, desenvolvedores, etc). A característica evolutiva advém do fato de que a qualquer momento do tempo novos itens podem ser inseridos ou removidos do *backlog* com base em novas prioridades e requisitos. Não existe um formato físico pré-definido para o backlog e nem um nível de granularidade para os itens a ele pertencentes.

Tendo em vista a natureza informal do levantamento e documentação de requisitos no desenvolvimento de softwares científicos, o *backlog* se apresenta como uma alternativa razoável que se posiciona entre dois extremos, a documentação pesada de processos cascata e a inexistente de softwares científicos em geral. A flexibilidade do seu formato e uso torna sua utilização por pesquisadores factível.

Design Simples

O termo *Design* Simples faz referência ao fato de que a estratégia de definição do *design* do software deve ser baseado no reconhecimento de que essa atividade é recorrente, que sua qualidade deve ser baseada na sua simplicidade e que decisões de design devem ser realizadas no último momento responsivo. Outro termo utilizado para essa prática é design emergente, enfocando o fato de que um design de qualidade irá emergir ao prestarmos atenção as qualidades locais do código.

O *Design* Simples é especialmente interessante para softwares científicos que em geral precisam de alterar os requisitos de acordo com o rumo da pesquisa científica que justifica seu desenvolvimento.

Testes

Testes Unitários

Testes unitários são pequenos trechos de código que visam exercitar o código sob teste para garantir que o mesmo se comporta da forma esperada. Usualmente os testes unitários são escritos na mesma linguagem de programação do código sob teste, testam uma funcionalidade restrita do código sob teste, explicando o adjetivo unitário, e geralmente rodam de maneira extremamente rápida.

Testes unitários podem representar um seguro para pesquisadores de que mudanças no código fonte no futuro não irão gerar problemas nas funcionalidades já existentes. De forma geral, eles servem para certificar para o pesquisador que o software está de fato funcionando da meneira pretendida.

Desenvolvimento Orientado a Testes - TDD

O Desenvolvimento orientado a testes representa um estilo de programação onde o teste unitário é escrito antes do código sob teste, gerando uma situação de falha inicial na primeira execução do teste. Posteriormente a código sob teste é escrito até que o teste unitário passe. Por fim, o código sob teste é refatorado como firma de manter sua simplicidade.

Acredita-se que o desenvolvimento orientado a testes é capaz de melhorar o design do código existente bem como diminuir o acoplamento e aumentar a coesão. A principal dificuldade da adoção desta prática por pesquisadores é decorrente da alteração da lógica usual de desenvolvimento.

Relacionamento com Cliente

Time

O time em um contexto ágil representa um grupo pequeno de pessoas, até nove pessoas no SCRUM, que trabalham juntas para a concretização de um projeto. A idéia de time tenta conjurar a idéia de que nenhum indíviduo é pessoalmente responsável pelo sucesso ou fracasso de um projeto, mas sim o time como um todo. Além disso, todas as habilidades necessárias para finalização do projeto devem estar presentes no time, sejam elas técnicas ou ligadas ao negócio.

A noção de time é adequada especialmente para pesquisadores que trabalham em uma mesma instituição de pesquisa.

Reunião Diária

Diariamente o time se reune para compartilhar informações em relação ao progresso do trabalho até então. Essa reunião é breve, no máximo 15 minutos, e é estruturada em torno de três perguntas principais, quais sejam: O que você completou desde a última reunião?; O que você planeja completar até a próxima reunião?; e Quais impedimentos você encontrou?

A reunião diária pode se apresentar como uma forma adequada de acompanhamento da evolução do desenvolvimento de softwares científicos.

Práticas Fundamentais

Controle de Versão

Controle de versão representa o ato de rastrear as alterações que foram realizadas em determinado documento. Via de regra, no contexto de desenvolvimento de software, são controladas todas alterações no código fonte de uma aplicação. Além da funcionalidade básica de rastrear alterações, os sistemas de controle de versão oferecem funcionalidades para facilitar o trabalho em equipe, como localização de alterações conflitantes e comparações arbitrárias do histórico das versões.

Na indústria de software o controle de versão é tido como prática básica, que possibilita a adoção de outras tidas como mais complexas, como a integração contínua. No entanto, no caso de software científicos, é usual que os pesquisadores não saibam nem que isso exista, ainda controlando versões do código via troca de emails.

Desenvolvimento Iterativo e Incremental

O desenvolvimento de software é considerado iterativo na medida em que intencionalmente existe repetição de atividades ligadas ao processo de desenvolvimento com revisão dos produtos já entregues. Além disso, ele é considerado incremental quando cada versão do produto é composta por um produto completo, ainda que sem o escopo total. O oposto de incremental é o processo que a cada etapa entrega uma componente técnica, como a base de dados, as regras de negócio e a interface.

O desenvolvimento iterativo e incremental é naturalmente adequado as características exploratórias presentes em qualquer pesquisa científica ao permitir que pesquisadores avaliem de forma adequada se o software está atendendo aos objetivos da pesquisa.

Build Automatizado

Uma build no vocabulário de engenharia de software representa o processo de construir o produto final do software que será utilizado pelos usuários a partir do código fonte e demais arquivos necessários mantidos pelos desenvolvedores. Um build é considerado automatizado quando não é necessário intervenção humana direta e o processo pode ser realizado somente com as informações disponíveis no sistema de controle de versão. Um

dos principais benefícios de qualquer processo de automatização consiste na diminuição de erros decorrentes da execução de etapas manuais.

Por usualmente não serem treinados em práticas de engenharia de software, pesquisadores também não costumam conhecer ferramentas de build como make, que permitem capturar e documentar não so a construção de softwares mas o *workflow* de uma análise de dados e da posterior geração de relatórios com *LaTeX* por exemplo. Um dos grandes ganhos neste caso é o aumento da reprodutibilidade na ciência computacional [1], tema que vem ganhando relevância nos últimos anos.

Treinamento

Programação em Pares

A programação em pares é a prática de dois desenvolvedores dividirem a mesma estação de trabalho, ou seja, apenas um computador é utilizado. O desenvolvedor com o comando do teclado é chamado de *driver*, e o desenvolvedor responsável por focar principalmente no direcionamento geral do trabalho é chamado de *navigator*. Apesar de ser uma prática que não pode ser simplesmente forçada sobre os desenvolvedores, ela é especialmente adequada para aumentar a difusão de conhecimento e a transferência de habilidades entre o time.

Para o caso de pesquisadores que usualmente são autodidatas, a possibilidade de programar com outro pesquisador mais experiente possui grande potencial de minimizar o efeito de 'redescobrir a roda'.

Considerações finais

Este trabalho visou identificar práticas ágeis de engenharia de software que podem ser utilizadas por pesquisadores para o desenvolvimento de softwares científicos.

O processo de desenvolvimento de software científico pode ser caracterizado pela utilização de processos ad-hoc de desenvolvimento, requisitos extremamente volatéis e muitas vezes implícitos, baixo nível de testes em específico e baixo nível de utilização de práticas de engenharia de software tidas como básicas em geral.

Este trabalho identificou que além da utilização de práticas de engenharia de softwares fundamentais, pesquisadores também podem se beneficiar de práticas ágeis de engenharia de software. As práticas ágeis surgiram formalmente com a publicação do manifesto ágil em 2001 e representaram um rompimento em relação ao modelo tradicional cascata de desenvolvimento existente predominantemente na época.

Em relação as práticas fundamentais merecem destaque a utilização de sistemas de controle de versão e de builds automatizados. Consideradas básicas na indústria, essas práticas são capazes de aumentar a eficiência de pesquisadores que desenvolvem softwares.

Além disso, em relação as práticas ágeis, merecem destaque o *backlog* e o desenvolvimento orientado a testes (TDD). Ambos atacam duas dificuldades que são comuns no desenvolvimento de softwares científicos, requisitos e testes, e possuem diversos relatos de que de fato funcionam, condição essencial para que sejam adotadas por pesquisadores que

[1] Para uma discussão do tema de reprodutibilidade na ciência computacional vide Peng (2011)

não possuem como objetivo principal o desenvolvimento de software, e, sim, a resolução de problemas científicos.

Sugere-se para trabalhos futuros a realização de experimentos de introduzir práticas ágeis em equipes de pesquisadores que previamente não as utilizavam para identificar empiricamente se elas realmente são capazes de melhorar a produtividade de pesquisadores que desenvolvem software científico.

Agile Practices for Scientific Software Development

Francisco Alves

2015

Abstract

Computational science and data science have been gaining relevance in the last few years to the point where they are considered respectively the third and fourth pillars of science, right next to theory and experimentation. One point of consensus is the necessity that researchers involved in those areas improve their computational skills, especially those connected to the software development process. The objective of this work is to identify software engineering agile practices that researchers could adopt for the development of scientific software. For that a literature review of how researchers currently develop software and what agile practices would show themselves useful in the current context. The results indicate that researchers could benefit from several fundamental software engineering practices, like version control and automated builds, but also from agile practices, like backlog and test driven development.

Keywords: scientific software. software engineering. agile manifesto. agile practices.

Referências

AGILE-ALLIANCE. *Guide to Agile Practices*. Acesso em: 24 set. 2015. Disponível em: <http://guide.agilealliance.org/>. Nenhuma citação no texto.

BECK, K.; ANDRES, C. *Extreme Programming Explained: Embrace Change*. 2nd edition. ed. Boston, MA: Addison-Wesley, 2004. ISBN 978-0-321-27865-4. Nenhuma citação no texto.

BECK, K. et al. *Manifesto for Agile Software Development*. 2001. Disponível em: <http://www.agilemanifesto.org/>. Nenhuma citação no texto.

CLEVELAND, W. S. Data science: An action plan for expanding the technical areas of the field of statistics. *Statistical Analysis and Data Mining*, v. 7, n. 6, p. 414–417, 2014. ISSN 1932-1872. Disponível em: <http://onlinelibrary.wiley.com/doi/10.1002/sam.11239/abstract>. Citado na página 1.

GENTZKOW, M.; SHAPIRO, J. M. *Code and Data for the Social Sciences: A Practitioner's Guide*. 2014. Disponível em: <http://faculty.chicagobooth.edu/matthew.gentzkow/research/CodeAndData.pdf>. Citado na página 2.

HEATON, D.; CARVER, J. C. Claims about the use of software engineering practices in science: A systematic literature review. *Information and Software Technology*, v. 67, p. 207–219, nov. 2015. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584915001342>. Citado na página 3.

PENG, R. D. Reproducible research in computational science. *Science*, v. 334, n. 6060, p. 1226–1227, dez. 2011. ISSN 0036-8075, 1095-9203. Disponível em: <http://www.sciencemag.org/content/334/6060/1226>. Citado na página 8.

PITAC. *Computational Science: Ensuring America's Competitiveness*. 2005. Acesso em: 20 ago. 2015. Disponível em: <https://www.nitrd.gov/pitac/reports/20050609_computational/computational.pdf>. Citado na página 1.

SCHWABER, K.; SUTHERLAND, J. *The Scrum Guide*. 2013. Acesso em: 24 set. 2015. Disponível em: <http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf>. Nenhuma citação no texto.

SEGAL, J. Models of Scientific Software Development. In: *First International Workshop on Software Engineering for Computational Science and Engineering (SECSE08)*. [s.n.], 2008. Disponível em: <http://www.cse.msstate.edu/~SECSE08/Papers/Segal.pdf>. Citado na página 4.

SLETHOLT, M. T. et al. What do we know about scientific software development's agile practices? *Computing in Science & Engineering*, v. 14, n. 2, p. 24–37, March-April 2012. Citado 2 vezes nas páginas 2 e 3.

VARDI, M. Y. Science has only two legs. *Communications of the ACM*, v. 53, n. 9, p. 5–5, 2010. ISSN 00010782. Disponível em: <http://portal.acm.org/citation.cfm?doid=1810891.1810892>. Citado na página 2.

WILSON, G. et al. Best Practices for Scientific Computing. *PLoS Biol*, v. 12, n. 1, p. e1001745, jan. 2014. Disponível em: <http://dx.doi.org/10.1371/journal.pbio.1001745>. Citado 2 vezes nas páginas 2 e 4.