Submission date: 20/04/2017
Resubmission date: 19/08/2017
Resubmission date: 28/09/2018
Camera ready submission: 18/01/2019

Section: thematic issue
Complexity of Information Systems

1st round notification: 18/07/2017
2nd round notification: 29/08/2018
3rd round notification: 21/12/2018
Available online: 11/02/2019
Publication date: 04/04/2019

Um Método para Determinar a Complexidade de uma Microtask em Ambientes Crowdsourcing

Title: A Method to Determine the Complexity of a Microtask in Crowdsourcing Environments

William Simão de Deus[1], José Augusto Fabri[1], Alexandre L'Erario[1]

[1]Programa de Pós-Graduação em Informática – Universidade Tecnológica Federal do Paraná (UTFPR)
Cornélio Procópio, Paraná – Brasil

williamsimao@outlook.com, {fabri, aleario}@utfpr.edu.br

Abstract. *Crowdsourcing (CS) software development employs the outsourcing of design, development and testing tasks, known as microtasks, to an anonymous group of participants. The use of microtaks in CS environments has been promoting the reduction of costs and diminution of the time of a project, which remits its importance. Despite this, the literature still does not concentrate an effective way on how to determine its difficulty of execution. Thus, the objective of this work is to present a method capable of evaluating the complexity that a microtask possesses. For this a method of determining complexity based on the characteristics of a microtask, was generated. Such a method was evaluated via a case study on a CS software development platform. The results showed that the method was efficient achieving an index of close to 90% of assertiveness in classifying microtasks as simple or complex in the analyzed sample. In addition, the method also showed that the simpler microtasks achieved a higher index of registration and submission than the complex microtasks.*

Keywords*. Microtasks; Complexity; Crowdsourcing Software Development; Software Complexity.*

Resumo. *O desenvolvimento de software crowdsourcing (CS) emprega a terceirização das tarefas de design, desenvolvimento e teste, conhecidas como microtasks, para um grupo anônimo de participantes. O uso das microtaks em ambientes CS vem promovendo a diminuição de custos e redução de tempo de um projeto, o que remete sua importância. Apesar disso, a literatura ainda não concentra uma maneira eficaz sobre como determinar a sua dificuldade de execução. Assim, o objetivo deste trabalho é apresentar um método capaz de avaliar a complexidade que uma microtask possui. Para isso foi gerado um método de determinação de complexidade baseado nas próprias características de uma microtask. Tal método foi avaliado via estudo de caso em uma plataforma de desenvolvimento de software CS. Os resultados mostraram que o método foi eficiente alcançando um índice próximo de 90% de assertividade em classificar microtasks como simples ou complexas na amostra analisada. Além disso, o método também evidenciou que as microtasks mais simples*

Cite as: Deus, W. S., Fabri, J. A. & L'Erario, A. (2019). Um Método para Determinar a Complexidade de uma Microtask em Ambientes Crowdsourcing. iSys: Revista Brasileira de Sistemas de Informação (Brazilian Journal of Information Systems), 11(4), 5-30.

alcançaram maior índice de registro e submissão em relação às microtasks complexas.

Palavras-Chave. *Microtasks; Complexidade; Desenvolvimento de Software Crowdsourcing; Complexidade de Software.*

1. Introdução

O crowdsourcing (CS) é um modelo de negócios que utiliza uma chamada aberta para atração de colaboradores anônimos no desenvolvimento de atividades ou na solução de problemas [Howe 2006b]. Sendo assim, o CS representa o ato de uma pessoa ou organização terceirizar uma porção de trabalho para um grupo de colaboradores anônimos, especialistas ou entusiastas, interessados na execução de uma atividade [Mao *et al*. 2017].

O CS é um modelo emergente no setor produtivo de software. Isso se deve ao fato de empresas de grande porte buscarem soluções diversificadas, rapidez e paralelização de atividades em plataformas *online* que suportam o desenvolvimento de software CS [TopCoder 2017, de Deus *et al*. 2016]. Dessa forma, com o mercado aquecido, diversas plataformas são criadas para suprir a crescente demanda industrial [Dwarakanath *et al*. 2015].

O desenvolvimento de software CS suportado nessas plataformas é baseado em atividades de curta duração para programação, teste ou *design* de um projeto de software [LaToza *et al*. 2014]. Essas atividades são conhecidas como *microtasks* por representarem pequenas porções de trabalho, reduzirem custos e possuírem curtos prazos de execução. Porém, a literatura ainda é incipiente e possui diversas lacunas relacionadas ao modo de mensurar a complexidade de uma *microtask*.

Em um mapeamento sistemático realizado por [de Deus *et al*. 2017] os autores revisaram a literatura e demonstraram que as a complexidade das *microtasks* é um dos principais fatores que torna complexo o gerenciamento de projetos CS. Além disso, o mapeamento identificou uma carência de estudos sobre métodos abordando o assunto de complexidade de execução. Os estudos revelados pelo mapeamento foram desenvolvidos apenas para suportar apenas projetos CS complexos, ou seja, projetos que possuem várias *microtasks* ou muitos participantes [Xiao e Paik 2014, Kittur *et al*. 2011].

Ademais, os mecanismos e métricas tradicionais adotados para mensurar a complexidade de atividades de software nos modelos de desenvolvimento similares não podem ser adotados diretamente em projetos CS. Segundo [Naik 2016] o CS é um modelo de desenvolvimento único que apresenta variáveis diferentes e mais complexas que outros modelos de desenvolvimento devido aos participantes anônimos e dispersos, atomicidade das *microtasks* e remuneração entre os pares. Desse modo, as métricas existentes para aferir a complexidade de atividades ou projetos de software apresentam falhas durante sua aplicação nas *microtasks* executadas em ambientes CS.

Nesse sentido é importante destacar o motivo pelo qual as métricas disponíveis (como *Mood*, Chidamber-Kemerer (CK), *Halstead*, complexidade Ciclomática (CC), *Mean Time Between Failures* (MTBF) e suas variações) não podem ser aplicadas nas *microtasks*. As métricas *Mood* e CK descrevem o uso de códigos de software baseado no paradigma orientado a objetos. Desse modo, ambas as métricas utilizam um mecanismo de encapsulamento, herança, polimorfismo e troca de mensagens para calcular a

complexidade de um sistema ou atividade de software [Chandra *et al*. 2017]. Entretanto, de acordo com [Mao *et al*. 2017], o CS utiliza *microtasks* voltadas a etapa de *design, mockups* e protótipos e as métricas *Mood* e CK não conseguem aferir a complexidade delas.

Esse problema é similar ao encontrado na métrica *Halstead* que consegue aferir a complexidade algorítmica, descartando os demais tipos de *microtasks*. A CC é uma métrica que mede a quantia de caminhos distintos que um software pode executar a partir de um código fonte. Adotar CC em projetos CS pode aumentar consideravelmente o tamanho de um projeto, visto que as *microtasks* são porções reduzidas de trabalho que devem ser testadas individualmente durante a integração do projeto. As métricas *Mean Time Between Failures* (MTBF) e suas variações (*Mean Time Between System Aborts* (MTBSA), *Mean Time Between Critical Failures* (MTBCF), *Mean Time Between Unscheduled Removal* (MTBUR) e *Mean Time To Failure* (MTTF)) só podem ser aplicadas em projetos concluídos, pois aferem com base nas falhas, abortos, erros críticos ou remoção de um sistema em execução. Cenário pouco eficiente para ser aplicado em ambientes CS já que os projetos encontram-se particionados em diversas *microtasks*.

Portanto, além da escassez de estudos sobre complexidade de execução de *microtask*, as métricas atuais incorporam particularidades dos ambientes em que foram concebidas e não abrangem as características dos ambientes CS. Amparados nessas lacunas, este trabalho gerou um método para calcular a complexidade de uma *microtask*. Para realizar a avaliação de tal método foi conduzido um estudo de caso em uma plataforma de desenvolvimento de software CS.

O restante deste trabalho encontra-se organizado da seguinte forma: a próxima seção apresenta os trabalhos relacionados, seguido pela revisão bibliográfica dos temas abordados. O método para determinar a complexidade da *microtask* é apresentado na seção 4 e logo após a sua avaliação. Os resultados são discutidos na seção 6 e conclusões são apresentadas no final deste artigo.

2. Trabalhos Relacionados

Existem diversas contribuições na literatura com objetivos sinérgicos ao desta pesquisa. Desse modo, é necessário explorar e revisitar o que já foi pesquisado e produzido. Logo, o objetivo desta seção é analisar trabalhos semelhantes exibindo suas contribuições e também limitações.

Inicialmente, é pertinente destacar a recente contribuição de [Jacques 2018] que explorou a fronteira que abstrai uma *microtask* de uma *macrotask* com base na complexidade de execução. A pesquisa do autor serviu como um importante marco para avançar a literatura sobre complexidade, todavia, a sua contribuição é restrita a sistematização do conhecimento existente, limitando-se em uma abordagem teórica. Ademais, o foco do trabalho também é restrito ao fornecer apenas intuições sobre o *design* e estrutura das atividades CS, não focando efetivamente na determinação da complexidade.

O trabalho desenvolvimento pelos autores [Aipe e Guadiraju 2018] revelou que a complexidade é um dos principais elementos que caracterizam e afetam a execução de uma atividade CS. Todavia, os autores direcionaram o foco do trabalho na investigação de semelhanças entre as atividades, reduzindo o impacto das contribuições sobre a

complexidade. Além do mais, o trabalho se apoiou em uma equação matemática para estabelecer o grau de semelhança tornando obtuso aos leitores a compreensão de como foi extraído o grau de complexidade.

Além destes estudos também é importante evidenciar o trabalho de [Yang *et al*. 2016] que investigou como a complexidade das atividades CS é percebida pelos participantes anônimos. Em resumo, o trabalho dos autores buscou detectar como a complexidade de uma atividade CS afetava a sua execução. Apesar do avanço proporcionado na literatura o trabalho possui uma série de limitações, como a dificuldade de generalização de resultados para o desenvolvimento de software CS (as atividades investigadas eram oriundas de diversas áreas, como tradução de textos e imagens) e as limitações da abordagem empregada (que focou somente na descrição e ignorou outros elementos, como o tempo disponível e o total de tecnologias empregadas nas *microtasks*).

Por fim, [Nakatsu *et al*. 2014] desenvolveram uma taxonomia do CS baseado na complexidade das atividades. A taxonomia, apesar de ampla, apresenta algumas limitações de aplicação prática no campo de desenvolvimento de software. Nesse sentido destaca-se a falta de atualização no estudo, a fragilidade em sua estrutura que considerou apenas três dimensões e a dificuldade de generalizar os resultados para o desenvolvimento de software CS.

3. Revisão Bibliográfica

Esta seção contextualiza os temas tratados por este trabalho. Desse modo é brevemente sumarizado o funcionamento do CS, a forma como o CS é aplicado no desenvolvimento de software e o como são organizadas e estruturadas as *microtasks*.

3.1. *Crowdsourcing*

O termo CS surgiu no ano de 2006 para apresentar um novo modelo de negócios que estava emergindo no cenário produtivo. De acordo com os criadores do termo, Jeff Howe e Mark Robinson, diversas empresas estavam modificando o seu eixo tradicional de desenvolvimento terceirizando funções (antes realizadas por seus colaboradores) para uma rede anônima de pessoas [Howe 2006b]. É importante destacar que o CS era uma personalização de um modelo bem estruturado e conhecido, o *outsourcing*. Todavia, como [Howe 2006a] destacou, a principal diferença entre CS e o *outsourcing* residia na realização de uma chamada aberta (campanha ou convite flexível) buscando atrair uma potencial rede de participantes anônimos.

Apesar dos riscos na participação anônima o CS demonstrou grande êxito de uso em diversas áreas da sociedade, como por exemplo, na astronomia, área em que o CS possibilitou a recente descoberta de um sistema multiplanetário [Greicius 2018]; ou na geografia, campo que o CS vem influenciando a produção de conhecimento [Sui *et al*. 2012]; ou ainda na Engenharia de Software, setor que está aplicando o CS para as etapas de *design*, desenvolvimento e teste de software [Mao *et al*. 2017].

O modelo de funcionamento do CS é praticamente o mesmo em todos os setores aos quais vem sendo utilizado. Em resumo, uma organização possui um projeto a ser desenvolvido, divide-o em uma lista de atividades a serem realizadas e cadastra essa coleção de atividades em um site ou plataforma *online*. Em seguida os participantes interessados em desenvolver alguma atividade submetem suas soluções. A organização,

ao final de um prazo estabelecido, seleciona a(s) melhore(s) solução(ões) e retribui os seus criadores por meio de uma premiação. Observe na Figura 1 uma abstração deste processo.

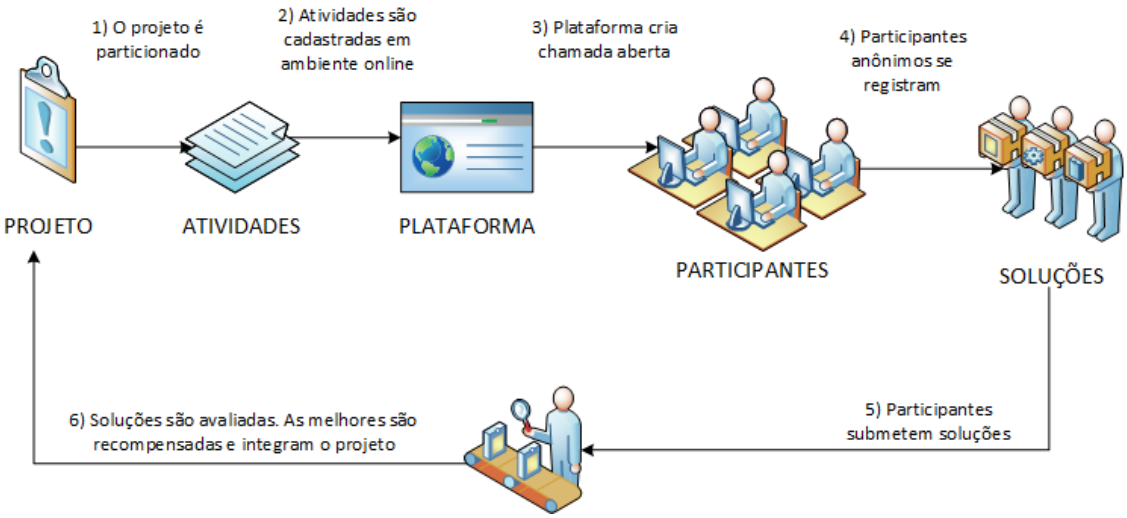

Figura 1. O funcionamento básico do *crowdsourcing*

Apesar da natureza dinâmica e anônima referente ao funcionamento do CS diversos benefícios são produzidos, como a flexibilidade geográfica e horária, a dispensa de trâmites burocráticos para contratação de profissionais e o acesso a uma extensa amplitude de participantes com habilidades distintas [Abhinav *et al*. 2018] e [Kurve15 *et al*. 2018]. Todos esses fatores acabaram impulsionando o CS em diferentes formas de utilização, principalmente no desenvolvimento de software. Assim, é pertinente elucidar sobre tal aplicação.

3.2. Desenvolvimento de Software *Crowdsourcing*

De acordo com [Mao *et al*. 2017], o desenvolvimento de software CS é o ato de executar qualquer atividade do ciclo de vida de um projeto por meio um grupo anônimo e potencialmente grande de participantes formado por meio de uma chamada aberta.

Para isso, o desenvolvimento de software CS geralmente envolve três tipos de atores: o *crowdsourcer*, que é o líder técnico e responsável pelo projeto; os *crowd workers*, que são os participantes anônimos; e a plataforma, sistema *online* acessado pelo *crowdsourcer* e pelos *crowd workers*.

Apesar das semelhanças com outros modelos de desenvolvimento, o CS apresenta um conjunto amplo de características que tornam seus desafios únicos quando comparados aos outros modelos de desenvolvimento de software [Li *et al*. 2015].

Nesse sentido, destaca-se que o gerenciamento do CS é praticamente inexistente visto que não existem mecanismos para acompanhamento do trabalho dos *crowd workers*. Além disso, o CS emprega participação colaborativa (todos procurando erros ou otimizações) ou competitiva (primeira submissão aprovada é considerada vencedora). O processo de desenvolvimento do CS é fracamente aberto, visto que ele é necessário possuir o cadastro em alguma plataforma CS para ter acesso as *microtask* disponíveis. Por fim, as atividades representam vetores opostos de funcionamento. No *open source* a comunidade define o que deve ser feito, enquanto que no *outsourcing* líder e equipe

selecionam as atividades, e no CS apenas o líder do projeto seleciona o trabalho a ser desenvolvido, sem consultar os participantes. A Tabela 1 resume as principais diferenças entre os modelos similares de desenvolvimento e o modelo CS.

Tabela 1. Comparação Entre Modelos de Desenvolvimento de Software Baseado em [Li *et al.* 2015] e [Naik *et al.* 2016]

	Open source	***Outsourcing***	***Crowdsourcing***
Gerenciamento	Colaboração com ferramentas online	Ferramentas online	Praticamente inexistente
Participação	Colaborativa	Times definidos	Colaborativa ou competitiva
Processo	Aberto	Fechado	Fracamente aberto
Atividades	Comunidade define	Líder e equipe definem por contrato	Líder define o contrato

3.3. *Microtasks*

As *microtasks* são atividades de porções reduzidas executadas no desenvolvimento de software CS, todavia, a literatura concentra um amplo dicionário de termos distintos, como "*tasks", "tasks crowd", "minitasks"* entre outras variações [Karim *et al*. 2018, Winkler *et al*. 2017, Hosseini *et al*. 2014] e [Mao *et al*. 2013]. Apesar das opções usadas, o termo *microtask* é o mais conhecido e difundido em grande parte dos estudos da área. Assim, para facilitar a compreensão do tema e evitar dúvidas deste sentido este trabalho adotou o termo "*microtasks*" para designar as atividades que são executadas no desenvolvimento de software CS.

Um exemplo de *microtask* aplicada ao desenvolvimento de software CS é encontrado na Figura 2, captada em uma plataforma CS para ilustrar a sua estrutura básica. Essencialmente existem três grupos de informações na *microtask*: descrição (contendo instruções de execução), perfil de *crowd workers* (neste caso, restrito aos trabalhadores que possuem determinados navegadores e sistemas operacionais) e o valor de retribuição .

Segundo [LaToza e Hoek 2015] as *microtask* representam a menor porção de trabalho possível. Entretanto, é praticamente impossível afirmar que todas são simples e representam apenas uma única porção de trabalho. Quando se põem em perspectiva o desenvolvimento de software, com atividades que possuem interdependência, é uma utopia dizer que todas as tarefas estão completamente atomizadas e reduzidas.

Apesar disso, as *microtasks* possuem grande portabilidade de aplicação. Isso é evidenciado pela crescente diversidade de múltiplas plataformas CS. Por exemplo, a MKTurk suporta *microtasks* de múltiplas naturezas, variando desde a usabilidade de sistemas até o processamento de dados. Já a Samasource é uma plataforma em que *crowdsourcers* enviam projetos para serem "quebrados" em *microtasks*. Além dessas plataformas, existem outros indicadores que demonstram o crescente emprego do conceito de micro atividades. Nesse sentido é importante destacar as considerações de [Goke e Freitag 2014] que mostraram o crescimento do interesse de diversas empresas, como a IBM, Google e BMW, que atualmente estão solucionando desafios por meio de *microtasks*.

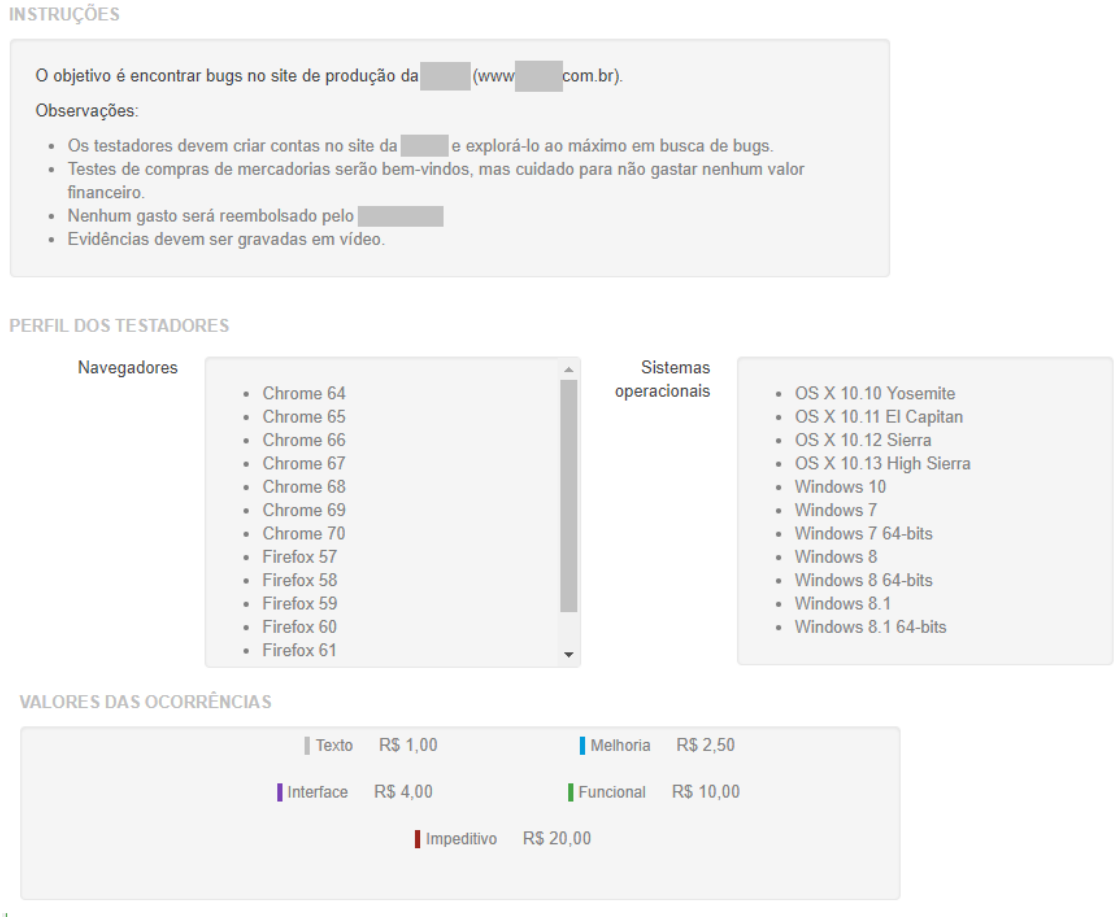

Figura 2. Exemplo de uma *Microtask*

Finalmente, também é importante destacar que as *microtasks* exercem múltiplas influências no desenvolvimento de software CS. O *crowdsourcer* é o responsável técnico e por isso realiza a definição e o cadastro da *microtask*, além de fiscalizar a sua execução. Já a plataforma é responsável pelo processo de submissão, que engloba a realização da chamada aberta para atrair participantes, suportar o registro e soluções. Por fim, existem os *crowd workers* que são os trabalhadores que executam e enviam soluções para as *microtasks*.

4. Método de Determinação de Complexidade

O objetivo desta seção é apresentar o método proposto neste trabalho para determinar a complexidade de uma *microtask*. Para isso a seção encontra-se estruturada em duas partes: na primeira porção ocorre uma conceptualização literária apresentando o processo de investigação sobre como a complexidade das *microtasks* é estimada; na segunda porção é exposta a estruturação, modelagem e aplicação do método proposto.

4.1. Definindo Complexidade

Definir efetivamente o que é a "complexidade" em uma *microtask* pode ser uma tarefa difícil e sinuosa. Isso ocorre em virtude de três fatores principais: primeiro, o ambiente dinâmico das próprias *microtasks*, que podem representar tarefas de *design*, desenvolvimento e teste de software; segundo, pela ausência de regulamentação nas plataformas de desenvolvimento de software CS, pois como foi apresentado na Tabela 1,

o gerenciamento em tais projetos é praticamente inexistente, faltando mecanismos que abstraiam o que é ou não a complexidade; e terceiro, a participação anônima do CS que torna obtuso a realização dos *feedbacks* por parte dos *crowd workers*. Desse modo, para evitar discussões pouco fecundas sobre o que vem a ser a complexidade de uma *microtask* este trabalho procurou por meios próprios contextualizá-la.

As dificuldades anteriormente citadas motivaram os autores deste estudo a realizarem uma investigação manual em trabalhos que tratavam o tema, buscando minerar informações em torno da complexidade das *microtasks*. Como resultado desta busca percebeu-se que as publicações analisadas (ver [Jacques 2018, Dubey *et al*. 2016, LaToza e Hoek 2015] e [Tranquillini *et al*. 2015]) apontavam para um caminho comum em torno do tema: as características de uma *microtask*. Em suma, notou-se que todas – ou pelo menos grande parte das *microtasks* – possuem características similares que podem ser quantificadas tornando a complexidade mais evidente. Observe na Tabela 2 uma síntese das características que foram captadas durante as análises dos estudos citados.

Tabela 1. As características identificadas na literatura sobre *microtasks*

ID	Nome	Escala
01	Requisitos	Um para muitos
02	Tecnologias	Uma para muitas
03	Tempo disponível	Minutos para semanas
04	Limitação	Nenhuma para muitas
05	Expertise	Nenhuma para muitas
06	Dependência	Nenhuma para muitas

Deve-se alertar que o campo *id* foi adicionado apenas para identificar as características, não servindo como base para estabelecer nenhum tipo de hierarquia ou prioridade. A seguir ocorre uma breve descrição de cada item identificado:

- **01 - Requisitos:** As *microtasks* possuem um conjunto de requisitos a serem completados. Somente quando todos os requisitos são executados e concluídos uma *microtask* é considerada finalizada [Dubey *et al*. 2016]. Além disso, [Jacques 2018] também lembra que conforme maior a lista de requisitos maior será a sua complexidade.
- **02 - Tecnologias:** Refere-se à quantia de tecnologias (linguagens de programação, bibliotecas, etc) que uma *microtask* necessita para ser implementada [Dubey *et al*. 2016]. O nível de complexidade de uma *microtask* aumenta em similaridade ao seu número de tecnologias.
- **03 - Tempo disponível:** Ao registrar uma *microtask* o líder deve selecionar o período disponível para a sua conclusão [Dubey *et al*. 2016, LaToza e Hoek 2015]. Com isso, o tempo disponível para a finalização deve ser considerado como um fator de complexidade. É importante destacar que no modelo CS o tempo disponível demonstra a seguinte lógica: menos tempo reflete em dificuldades para encontrar participantes qualificados. Ou seja, uma *microtask* que possui apenas 60 minutos para ser executada é considerada mais complexa que uma com 3 dias de execução. Isso ocorre pelo fato da segunda *microtask* possuir uma vitrine maior de tempo para ser encontrada e selecionada por um *crowd worker*. Ademais,

grande parte das plataformas CS não empregam mecanismos para gerenciar a granularidade de uma *microtask*. Assim, apenas o líder define o prazo de execução. Na prática isso representa que uma atividade com maior tempo disponível não é, obrigatoriamente, uma tarefa mais complexa de ser executada.

- **04 - Limitação:** [Dubey *et al*. 2016] e [Tranquillini *et al*. 2015] alertam sobre as *microtasks* que possuem limitações para participantes, como mínimo de experiência, fluência em um determinado idioma, etc. Similar as tecnologias, conforme maior o número de limitações, mais complexa a *microtask* será considerada.
- **05 - Expertise:** Algumas *microtasks* podem requerer o domínio de algum conhecimento (softwares de edição, componentes específicos, etc) para serem desenvolvidas [LaToza e Hoek 2015, Tranquillini *et al*. 2015]. Desse modo, as *microtasks* que demandam um nível de expertise apresentam complexidade maior.
- **06 - Dependência:** Segundo [Jacques 2018] e [LaToza e Hoek 2015], existem *microtasks* que alteram o fluxo, a entrada, a saída ou o processamento de outras tarefas/rotinas. Desse modo, o nível de dependência contribuiu para aumentar a complexidade de uma *microtask*.

As características foram classificadas em três diferentes escalas: i) um(a) para muitos(as), ii) minutos para semanas e nenhum para muitos. No caso das características 01 e 02 sempre haverá ao menos um requisito e uma tecnologia associada a cada *microtask* por serem considerados obrigatórios. Na característica 03 a contagem ocorre por meio da duração em minutos até semanas. Por fim, as características 04, 05 e 06 são opcionais, podendo estar ou não presentes nas *microtasks*, sendo que a ausência de tais características demanda mais simplicidade para a execução.

4.2. Conversão

Após compreender como as características atuam em uma *microtask* foi necessário realizar um procedimento de quantificação. Para isso foi gerado um método que converte e quantifica as características, determinando a complexidade da *microtask*.

De acordo com a literatura consultada e discutida na subseção anterior, as duas primeiras características, requisitos e tecnologias, variam entre 1 e N para cada *microtask* enquanto as características referentes as limitações, expertises e interdependências variam entre 0 e N. Com isso, a inclusão de alguma dessas características adiciona um peso +1 ao cálculo de complexidade da *microtask*. Para parametrizar a característica restante, o tempo disponível, foram criadas 5 escalas: i) *microtasks* com duração maior ou igual a 30 dias (+1), ii) *microtasks* com duração até 30 dias (+2), iii) *microtasks* com duração até 7 dias (+3), iv) *microtasks* com duração de até um dia (+4) e v) *microtasks* com até 60 minutos de duração (+5). Essas escalas foram geradas com base na percepção que o tempo disponível atua como um fator de complexidade na *microtask,* como discutido anteriormente.

Para facilitar a compreensão da conversão, a Figura 3 ilustra o procedimento completo de conversão. A imagem apresenta que a estrutura de uma *microtask* descende de 6 características. Por sua vez, cada característica possui algum tipo de variação, sendo

de 1 para N (requisitos e tecnologias), minutos para semanas (tempo disponível) e 0 para N (limitação, expertise e dependência).

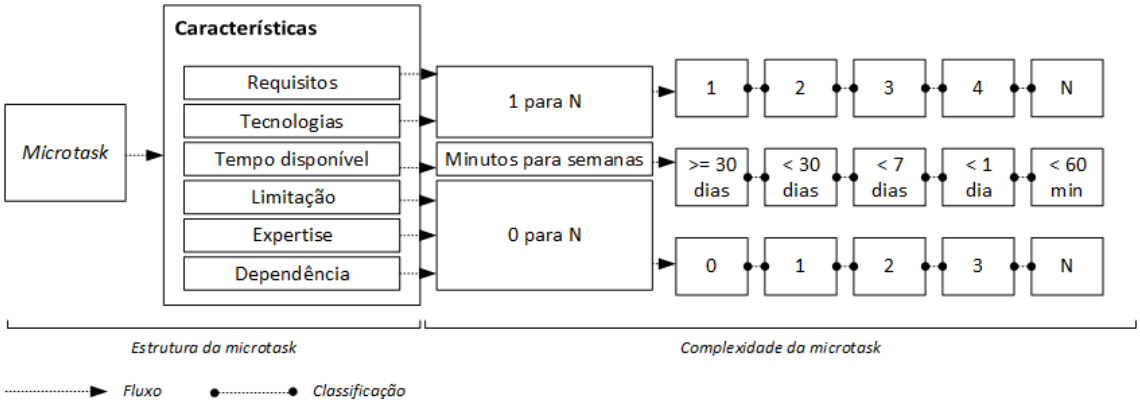

Figura 3. Procedimento para converter características das *microtasks* em complexidade

O método proposto identifica e classifica seis características de uma *microtask*. Após isso, cada característica deve ser quantificada por meio de suas ocorrências. Ao fim, a soma total das características determina a complexidade da *microtask*.

5. Estudo de caso

Para avaliar o método de determinação de complexidade proposto neste trabalho foi conduzido um estudo de caso em uma plataforma de desenvolvimento de software CS. Assim, foi abordada a complexidade das *microtasks* dentro de seu contexto de uso, retratando-as como um elemento essencial e intrinsecamente necessário para que o cenário de CS exista. Para conduzir o estudo de caso foi utilizado os conceitos apresentados por [Yin 2015] e as considerações de [Wohlin *et al*. 2012]. As próximas subseções descrevem a organização e a configuração do estudo de caso conduzido.

5.1. Questões de Pesquisa

A primeira etapa do estudo de caso foi definir as questões de pesquisa (QPs) a serem solucionadas para avaliar a eficácia o método proposto. Desse modo, foram estabelecidas duas seguintes QPs:

- ***(QP$_1$)*** *A complexidade de uma microtask influenciou a sua porcentagem de finalização?*
- ***(QP$_2$)*** *A complexidade de uma microtask influenciou a quantidade de registros e de submissões?*

A teoria mostra que o ambiente CS possui diversas restrições, como a dificuldade de comunicação e ausência de reuniões entre *crowd workers*, que dificultam a compreensão e a execução de atividades com maior grau de complexidade. Assim, a QP$_1$ foi elaborada para avaliar se o método proposto seria capaz de identificar como a complexidade de uma *microtask* poderia influenciar a sua finalização. Em suma, para avaliar sua eficiência, as *microtasks* consideradas simples pelo método deveriam ter uma porcentagem de finalização superior as *microtasks* consideradas complexas.

A QP_2 explorou como a complexidade de uma *microtask* poderia influenciar os registros e submissões por parte dos *crowd workers*. De forma objetiva foi investigado se o método proposto seria eficiente e capaz de revelar alguma diferença do desempenho de registro e submissão entre as *microtasks* consideradas simples e complexas.

5.2. Seleção do Caso

Após definir as QPs a serem analisadas foi iniciado o processo para a seleção de caso. Os critérios definidos para tal seleção foram estipulados de acordo com as seguintes regras:

i) Selecionar apenas organizações que desenvolvam software em ambientes CS;

ii) Selecionar apenas organizações que possam suportar a execução de *microtasks* de *design*, desenvolvimento e teste de software.

Deve-se salientar que o primeiro critério foi estabelecido para filtrar apenas as organizações que são o foco deste estudo, cujo propósito é desenvolver software em ambientes CS. Enquanto que o segundo critério filtrou as organizações que suportavam a execução de *microtasks* em pelo menos três etapas do ciclo de vida de um projeto de software. Dentro desse cenário apenas uma plataforma foi contemplada, denominada neste trabalho como plataforma X, que por fatores estratégicos não possuirá a sua identificação revelada.

A plataforma X opera como uma das grandes lideranças dentro do mercado de desenvolvimento de software CS, sendo uma das responsáveis por disseminar tal prática.

O lançamento da plataforma X ocorreu no ano de 2001. No início de sua operação apenas realizava competições de programação. Nessa época os participantes se cadastravam na plataforma, selecionavam desafios de programação e enviam soluções próprias. As soluções mais bem avaliadas eram recompensadas por meio de um processo de premiação financeira. Este modelo de competição foi evoluindo conforme o avanço dos anos, e em 2006 com a disseminação do CS a plataforma X diversificou a sua atuação. Nesse período a plataforma começou a oferecer contratos para clientes que desejassem solucionar problemas e desafios próprios, como por exemplo, a execução de atividades focadas em *design web* e também ideias de aplicações, aumentando assim a sua base de clientes e também a sua comunidade de participantes. Em vista dessa estratégia, a plataforma conseguiu sobressair e crescer exponencialmente dentro do mercado internacional. Atualmente, a plataforma X possui cerca de 1.200.000 *crowd workers*. Tais trabalhadores possuem experiência em áreas como *design*, desenvolvimento, ciência de dados e programação competitiva. Além disso, a plataforma já distribuiu mais de US$ 80.000.000 em premiação para a sua comunidade por meio da execução de desafios[1].

Em relação ao seu processo de desenvolvimento, a plataforma X emprega um conjunto de características e regras para aplicar CS no desenvolvimento de software. De modo simplificado tal processo pode ser dividido em quatro etapas distintas, organizadas da seguinte maneira:

[1] Cabe destacar que a plataforma rótula as atividades como "desafios" para tornar o seu produto mais atrativo aos clientes e participantes. Todavia, o termo "desafio" é considerado apenas uma terminologia opcional ao termo "*microtask*", adotado neste trabalho.

1) Um *crowdsourcer* cadastra uma *microtask* na plataforma. Nesse ato, o *crowdsourcer* deve preencher os seguintes campos: nome, descrição, tipo (design, desenvolvimento ou teste), duração (em dias) e premiação;
    a. É importante destacar que não existe fiscalização na etapa de cadastro, ou seja, os *crowdsourcers* podem cadastrar atividades com informações obtusas, faltantes ou simplesmente impraticáveis;
    b. O tempo de duração é estipulado entre duas datas (data inicial e data final), representado o total de dias disponíveis para a execução da *microtask*. Alcançado a data final a *microtask* vai para avaliação e não é possível realizar novas submissões;
    c. A plataforma possui outras categorias de classificação, não ficando restrita apenas ao *design*, desenvolvimento e teste de software CS. Contudo, o estudo de caso focou apenas nos tipos investigados.
2) Após o cadastro a *microtask* fica disponível na plataforma e *crowd workers* podem se registrar para a execução;
3) Após um *crowd worker* se registrar, ele possui habilitação para submeter várias submissões até a data final da *microtask*;
    a. Nenhum *crowd worker* é punido se não submeter soluções.
4) Após a finalização do prazo, o *crowdsourcer* avalia todas as submissões e seleciona a(s) melhor(es) solução(ões), retribuindo os respectivos *crowd workers*.
    a. Nesta etapa ocorrem dois cenários possíveis: a *microtask* é finalizada com sucesso (alguma submissão foi escolhida vencedora) ou falha (nenhuma submissão foi escolhida vencedora ou não houve nenhuma submissão).

5.3. Procedimento de Coleta de Dados

Para realizar a coleta de dados os autores consultaram a *baseline* da plataforma X. Tal *baseline* apresentava uma coleção de *microtasks* já desenvolvidas, contendo seus metadados (como nome, descrição, categoria, duração, premiação, etc) e status de execução (sucesso ou falha).

Os dados recuperados na consulta da *baseline* foram organizados e tabulados em uma planilha eletrônica. Cada linha da planilha correspondia a uma *microtask* e era identificada por um código único.

Após isso, todos os códigos presentes na planilha foram inseridos na ferramenta *Random Picker*. Essa ferramenta realizava uma escolha aleatória revelando o código de uma *microtask*. Assim, após cada escolha a respectiva *microtask* selecionada era analisada para compor este estudo de caso. Naturalmente, o código colhido era removido da lista para novas seleções, evitando que uma *microtask* fosse selecionada duas ou mais vezes.

Foram realizadas 30 seleções aleatórias durante esse processo de escolha. O critério de parada para seleção foi definido com base nas seguintes condições: primeiro, a amostra coletada deveria possuir o mesmo número de *microtasks* com sucesso e falha; segundo, a amostra coletada deveria envolver *microtasks* de design, desenvolvimento e

teste na mesma proporção. Ao alcançar 30 seleções as duas condições foram satisfeitas e foi iniciado a próxima etapa do estudo.

5.4. Procedimento de Análise

As *microtasks* selecionadas foram tabuladas em uma nova planilha eletrônica, e com base nessa planilha, foi iniciado o seguinte processo de análise:

- Cada linha da nova planilha correspondia a uma *microtask*. E cada item possuía uma série de informações coletadas em diferentes domínios, como em campos textuais (nome e descrição da *microtask)*; campos de datas (início e final dos registros); e campos numéricos (total de tecnologias empregadas). Assim, foi necessário analisar os dados individualmente para padronizar as informações em toda a amostra. Durante essa etapa, os autores buscaram extrair e organizar com exatidão as informações referentes aos requisitos, tecnologias, tempo disponível, limitações, expertise e dependência de cada *microtask*.
- As informações extraídas foram quantificadas conforme as premissas apresentadas na subseção 4.2 deste estudo. Para exemplificação, considere que determinada *microtask* tivesse apresentada em sua descrição que seria obrigatório completar dois requisitos para sua finalização (exemplo: submeter a solução em extensão *.zip* e descrevê-la em um arquivo com extensão *.txt*), a *microtask* receberia o valor +2 para a característica de requisitos. Esse procedimento foi realizado para todas as características (requisitos, tecnologias, tempo disponível, limitações, expertise e dependência) existentes em cada *microtask*.

5.5. Procedimento de Avaliação

Nesta etapa foram empregados diferentes recursos da estatística descritiva como forma de auxiliar o processo investigativo. No que concerne tal procedimento cabe destacar que foram aplicados cálculos de frequência para compreender a natureza da distribuição dos dados. Além destes, também recorreu-se a aplicação de medidas de dispersão na companhia de técnicas de visualização gráfica para descrever visualmente a disposição de dados e facilitar a sua compreensão. Por fim, cabe destacar que todo o procedimento de avaliação foi realizado com base no uso de um software estatístico denominado minitab®18. Esse software auxiliou o procedimento de avaliação, automatizando as etapas da análise de frequência e de dispersão e gerando automaticamente os gráficos.

Além disso, para reduzir a incidência de riscos, destaca-se que a captação de *microtasks* foi realizada de modo aleatório, evitando coletar dados viciados ou repetidos. Além disso, todas as *microtasks* selecionadas foram investigadas manualmente pelos pesquisadores.

6. Resultados

O objetivo desta seção é sumarizar os principais resultados obtidos com a execução do estudo de caso. Assim, a seção encontra-se estruturada em cinco porções: a primeira porção mostra uma visão geral da amostra analisada no estudo de caso. A segunda e a terceira partes focam, respectivamente, na análise da complexidade *versus* o índice de finalização (QP_i) e na investigação da complexidade *versus* os registros e as submissões

(QP_3) a quarta porção apresenta uma síntese dos resultados obtidos e a última seção aborda as limitações do estudo.

6.1. Análises iniciais

O estudo de caso coletou dados de 30 *microtasks*. Inicialmente, a amostra foi organizada e dividida de acordo com o tipo de sua aplicação (*design*, desenvolvimento ou teste) e seu status de finalização (sucesso ou falha). A relação que apresenta as frequências e as descrições de cada tipo de *microtasks* presente na amostra é encontrada na Tabela 3. Além disso, a frequência referente ao status de finalização está exposta na Tabela 4.

Tabela 3. Visão geral da amostra em relação ao tipo

Etapa	Descrição	Total	Frequência
Design	protótipos e interfaces	10	33.33%
Desenvolvimento	implementação de códigos	10	33.33%
Teste	correção e busca de erros	10	33.33%

Tabela 4. Visão geral da amostra em relação ao status de finalização

Etapa	Sucesso	Falha
Design	50%	50%
Desenvolvimento	50%	50%
Teste	50%	50%

Em relação ao status de finalização das *microtasks*, cabe destacar o conjunto de regras adotadas.

- SUCESSO: para definir uma *microtask* com sucesso de execução é fiscalizado se todas as seguintes condições foram satisfeitas:
    - Houve ao menos um registro e uma submissão;
    - Houve ao menos uma submissão selecionada;
    - A solução selecionada foi enviada até o prazo estipulado;
    - A solução selecionada cumpriu todos os requisitos expressos na descrição da *microtask*.
- FALHA: já para uma *microtask* possuir o status de cancelada (falha) podem ocorrer as seguintes condições:
    - Não houve nenhum registro ou nenhuma submissão;
    - Nenhuma submissão foi selecionada;
    - As submissões não respeitaram o prazo estipulado ou não possuíam atendido algum requisito;
    - A *microtask* se mostrou inviável por possuir muitos requisitos a serem cumpridos.

Em via dessas considerações a amostra foi dividida igualitariamente (metade com sucesso e a outra metade com falha) para balancear a amostra e evitar a concentração de

dados em torno de somente um status de finalização. Tal disposição é apresentada por meio da Tabela 5, que evidencia ainda a etapa de execução (design, desenvolvimento ou teste), os registros e as submissões recebidas.

Tabela 5. Visão geral da amostra em relação aos registros e submissões

Identificador	Etapa	Status	Registros	Submissões
2	Design	Sucesso	8	2
4	Desenvolvimento	Sucesso	18	3
497	Teste	Sucesso	13	3
511	Design	Sucesso	14	1
694	Desenvolvimento	Sucesso	26	4
804	Teste	Sucesso	8	1
806	Teste	Sucesso	9	4
1455	Teste	Sucesso	12	3
1579	Teste	Falha	13	0
2484	Teste	Sucesso	7	1
2615	Desenvolvimento	Falha	28	3
2897	Design	Falha	19	2
3122	Desenvolvimento	Sucesso	10	2
3657	Desenvolvimento	Falha	10	0
4734	Design	Sucesso	10	4
6094	Design	Falha	7	0
6356	Desenvolvimento	Sucesso	12	3
6968	Design	Falha	3	0
8604	Desenvolvimento	Falha	23	0
8704	Desenvolvimento	Falha	7	0
9354	Design	Falha	12	0
9942	Design	Sucesso	8	3
10591	Teste	Falha	6	0
12257	Teste	Falha	17	0
12743	Design	Sucesso	7	2
12888	Desenvolvimento	Sucesso	43	17
15181	Teste	Falha	7	0
15407	Desenvolvimento	Falha	13	1
15749	Teste	Falha	12	0
15900	Design	Falha	7	0

A teoria central do método apresentado neste trabalho propõe quantificar as características de uma *microtask,* obtendo assim a sua complexidade final por meio de uma soma. Nesse sentido, a Tabela 6 mostra a quantificação de cada característica obtida na amostra analisada. Um interessante recorte dessa etapa foi a identificação que a "Expertise" representou a característica que menos vezes foi acionada nas *microtasks*. Ou seja, em grande parte da amostra, os *crowdsourcers* não estipularam domínios de nenhum conhecimento aos *crowd workers*. Enquanto isso, em termos gerais, os "Requisitos" representaram a característica mais utilizada nas *microtasks*. Essas percepções podem demonstrar certa tendência do próprio CS, já devido ao trabalho anônimo é extremamente difícil para o *crowdsourcer* comprovar que os *crowd workers* possuíam realmente as expertises solicitadas. Além do mais, devido a dificuldade de comunicação é comum que os requisitos detalhem com maior objetividade o que de fato deve ser feito para executar a *microtask*.

Tabela 6. Extração e conversão dos dados das *microtasks*

Identificador	Requisitos	Tecnologias	Tempo	Limitações	Expertise	Dependência	Complexidade
2	2	1	3	2	0	1	9
4	3	4	3	2	0	1	13
497	15	1	2	0	0	0	18
511	1	2	3	0	1	5	12
694	4	1	3	0	0	2	10
804	4	1	3	0	0	11	19
806	6	1	3	0	0	4	14
1455	4	4	2	0	0	3	13
1579	17	2	3	1	0	1	24
2484	4	4	2	1	0	1	12
2615	12	1	3	1	0	1	18
2897	3	4	3	0	0	5	15
3122	5	6	3	0	0	1	15
3657	4	4	3	6	0	1	18
4734	8	0	3	0	0	4	15
6094	9	2	3	2	0	1	17
6356	7	2	3	1	0	1	14
6968	13	0	3	1	0	0	17
8604	5	1	3	1	4	4	18
8704	6	3	3	1	1	3	17
9354	4	3	3	2	4	1	17
9942	6	1	3	0	0	2	12
10591	27	2	2	7	0	4	42
12257	9	4	2	0	0	13	28
12743	2	1	3	1	1	2	10
12888	7	1	3	0	0	3	14
15181	10	2	2	3	0	5	22
15407	4	2	2	0	0	1	9
15749	2	2	2	2	0	12	20
15900	26	2	3	2	0	0	33

Nota: Complexidade = $R + T + Te + L + E + D$, em que R = requisitos, T = tecnologias, Te = tempo, L = limitações, E = expertise e D = dependência.

6.2. A complexidade da uma *microtask* influenciou a sua porcentagem de finalização?

Após realizar a análise inicial dos dados, foi investigada a solução da QP_1. Em resumo, o objetivo desta etapa foi verificar se as *microtasks* mais complexas da amostra foram finalizadas com sucesso ou falha.

Para identificar as *microtasks* mais complexas foi adotado o cálculo de quartis. De modo resumido pode-se afirmar que os quartis são utilizados para exibir a dispersão e a tendência central de uma amostra. Existem 3 intervalos de quartis, o primeiro quartil (Q1) que representa 25% dos menores dados encontrados na amostra; o segundo quartil (Q2) que apresenta a mediana amostral e o terceiro quartil (Q3) que exibe os 75% dos maiores dados amostrais. A relação dos quartis obtidas na analisada amostra é encontrada na Tabela 7.

Tabela 7. Complexidade das *microtasks* separada entre quartis

Quartil	Condição	Valor

Q1	Menor ou igual que	12.75
Q2	Menor ou igual que	16.00
Q3	Menor ou igual que	18.25

Após a observação dos intervalos entre quartis foi adotado que o Q2 como representaria a fronteira para definir e classificar as *microtasks* como simples ou complexas. Essa seleção ocorreu em virtude de existirem apenas dois status de finalização (sucesso ou falha). Assim, o Q2 demonstrou ser o valor mais sensato, dividindo a amostra igualitariamente. Ademais, cabe destacar que foi considerado que as *microtasks* com complexidade menor ou igual a 16 eram simples e que com complexidade superior a 16 foram consideradas complexas, como mostra a Tabela 8.

Tabela 8. Fronteira entre *microtasks* simples e complexas baseada no segundo quartil amostral

Classificação	Identificador	Etapa	Status	Complexidade
SIMPLES	15407	Desenvolvimento	Falha	9
SIMPLES	2	Design	Sucesso	9
SIMPLES	12743	Design	Sucesso	10
SIMPLES	694	Desenvolvimento	Sucesso	10
SIMPLES	9942	Design	Sucesso	12
SIMPLES	511	Design	Sucesso	12
SIMPLES	2484	Teste	Sucesso	12
SIMPLES	1455	Teste	Sucesso	13
SIMPLES	4	Desenvolvimento	Sucesso	13
SIMPLES	806	Teste	Sucesso	14
SIMPLES	6356	Desenvolvimento	Sucesso	14
SIMPLES	12888	Desenvolvimento	Sucesso	14
SIMPLES	2897	Design	Falha	15
SIMPLES	4734	Design	Sucesso	15
SIMPLES	3122	Desenvolvimento	Sucesso	15
		Fronteira estabelecida por Q2		
COMPLEXA	6094	Design	Falha	17
COMPLEXA	6968	Design	Falha	17
COMPLEXA	8704	Desenvolvimento	Falha	17
COMPLEXA	497	Teste	Sucesso	18
COMPLEXA	8604	Desenvolvimento	Falha	18
COMPLEXA	3657	Desenvolvimento	Falha	18
COMPLEXA	2615	Desenvolvimento	Falha	18
COMPLEXA	804	Teste	Sucesso	19
COMPLEXA	15749	Teste	Falha	20
COMPLEXA	15181	Teste	Falha	22
COMPLEXA	1579	Teste	Falha	24
COMPLEXA	12257	Teste	Falha	28
COMPLEXA	15900	Design	Falha	33
COMPLEXA	10591	Teste	Falha	42

As *microtasks* consideradas simples obtiveram um desempenho de finalização superior em relação as complexas. Essa diferença foi próximo à casa de 90% de finalização com sucesso. Da amostra analisada, 13 *microtasks* (86,67%) consideradas simples foram finalizadas com sucesso e apenas 2 (13,33%) não obtiveram êxito na finalização. Essa porção é inversamente proporcional ao índice de finalização das

microtasks complexas, que tiveram apenas 2 (13,33%) finalizadas com sucesso, ao ponto que 13 (86,67%) não foram concluídas, como sintetizado na Tabela 9.

Tabela 9. Complexidade das *microtasks* e índice de finalização

Complexidade	Sucesso	Falha	Finalização (%)
SIMPLES (<= 16)	13	2	86,67%
COMPLEXAS (> 16)	2	13	13,33%
TOTAL	**15**	**15**	**100,00%**

Apesar dos indicadores positivos que demonstraram a aderência do método proposto para detectar sucesso de finalização das *microtasks* simples das *microtasks* complexas, 4 amostras despertaram a atenção. Sobre isso, os autores investigaram mais profundamente as *microtasks* 15407 e 2897 que não foram concluídas com êxito, mas ainda assim foram consideradas simples; e as *microtasks* 497 e 804, que apesar de serem classificadas como complexas foram finalizadas com sucesso.

15407: Essa *microtask* possuía o objetivo de providenciar a implementação de um *framework* para validar e converter documentos. A descrição da *microtask* estava dentro de um padrão aceitável de detalhes, com um prazo de tempo plausível e com instruções claras e precisas. Todavia, ao aprofundar a investigação ficou evidente qual o motivo da falha de execução: cancelada na revisão. Em suma, isso quer dizer que a *microtask* era simples e obteve submissões, todavia, nenhuma submissão seguiu corretamente os requisitos especificados e por isso foi considerada falha.

2897: A finalidade dessa *microtask* era oferecer uma solução para realizar uma integração entre dois sistemas. Similar a *microtask* anteriormente analisada, essa também contava com uma descrição clara e com tempo compatível. E, similarmente, ocorreu o mesmo motivo de falha, a de ser reprovada na revisão.

Após observar os dois cenários das *microtasks* ficou evidente que o método proposto foi assertivo por considera-las simples. Assim, as falhas de finalização não se restringiram necessariamente na complexidade, mas sim na falta de atenção dos *crowd workers* em relação aos requisitos de cada tarefa.

497: Essa *microtask* foi classificada como complexa e ainda assim obteve sucesso na sua finalização. A sua complexidade se deu, principalmente, no total de requisitos necessários (15), fato que não impediu a sua conclusão exitosa. Dos fatores observados o que se mostrou mais eficiente em explicar tal comportamento foi o foco ao qual a *microtask* estava condicionada. Tal tarefa era de busca por erros, e isso pode explicar a facilitação de conclusão já que os *crowd workers* deveriam apenas identificar erros. Apesar de possuir uma alta quantia de requisitos, a busca de erros é mais simples que as atividades de outras etapas, como a criação de protótipos (*design*), a codificação (desenvolvimento) e a correção de erros (teste), pois tende a ser executada rapidamente.

804: Apesar dos esforços em identificar qual o fator que culminou em classificar essa *microtask* como complexa e ainda assim ser finalizada com sucesso, não foi obtido nenhuma hipótese eficiente. A *microtask* 804 possuía muitas dependências (11), um prazo considerado curto (3 dias) e ainda assim foi finalizada com sucesso.

Ao fim percebeu-se que o método proposto foi pouco eficiente em classificar essas duas *microtasks*. Apesar disso, tal consideração ainda não invalidou o método proposto

(que alcançou quase 90% de sucesso), mostrando possíveis melhorias a serem desenvolvidas.

6.3. A complexidade de uma *microtask* influenciou a quantidade de registros e submissões?

Como foi dito anteriormente na seção 5.2 deste estudo, a plataforma X emprega duas etapas na execução de *microtasks*: na primeira etapa os participantes interessados se registram, e na segunda etapa os participantes submetem as suas soluções. Assim, naturalmente o número de submissões tende a ser menor que o número de registros. Tendo isso em vista foi elaborada a QP_2 e o objetivo desta subseção é analisar se existe alguma relação entre o índice de registro/submissão e a complexidade que estipulada pelo método proposto. O que se deseja com tal conjectura é identificar se as *microtasks* mais complexas tiveram menos submissões que as *microtasks* mais simples.

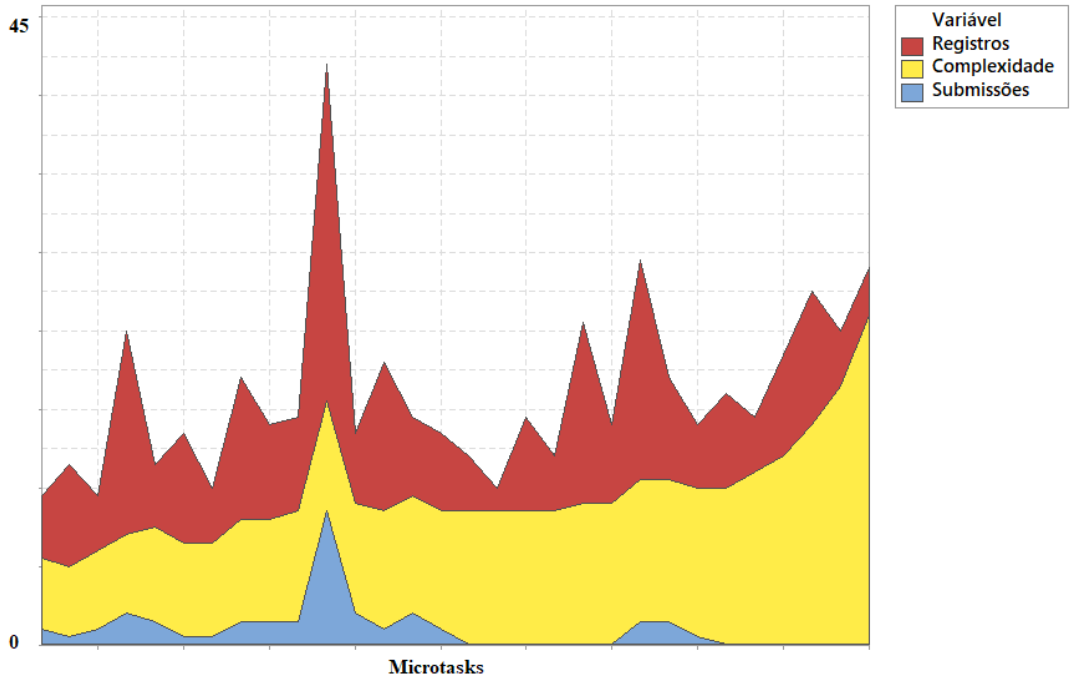

Figura 4. Relação entre complexidade e registros/submissões das *microtasks*

O ponto de partida da investigação foi visualizar graficamente o comportamento entre a complexidade, os registros e as submissões das *microtasks*. Para isso foi gerado um gráfico de dispersão sobre os índices analisados, como mostrou a Figura 4.

Ao analisar o gráfico ficou evidente que a complexidade seguiu um padrão bem similar aos registros e submissões. Em suma, o método proposto foi capaz de adequar a dificuldade com o índice de *crowd workers* que se registraram e submeteram soluções. Em relação a isso, pode-se fazer as seguintes observações:

- A área vermelha destacada no gráfico representa o índice de registros. E na análise inicial deste índice não mostrou nenhum comportamento lógico. Um dos principais fatores que justificam essa observação é o fato das *microtasks* apresentam picos e reduções de registros em diversos pontos.

- Já a área em azul apresentou a relação de submissão. Ao analisar essa visão, percebe-se que as submissões estão concentradas na zona a esquerda do gráfico, ao ponto que existe apenas uma pequena ilha na zona a direita que mostra *microtasks* com submissões.
- Por fim, a área amarela exibiu a complexidade das *microtasks*, e ao unificar a visão das três áreas notou-se dois comportamentos importantes: o primeiro representa o fato da complexidade seguir o índice de registros, destacando a aderência do método. O segundo apresenta que conforme houve um acréscimo da complexidade houve redução de submissão. Ou seja, naturalmente ao aumentar a complexidade da *microtask* ocorreu menos submissão.

Tendo em vista as interpretações anteriores, a análise foi aperfeiçoada gerando um gráfico de dispersão (encontrado na Figura 5). Esse gráfico foi separado em quatro quadrantes distintos, que representam uma matriz de complexidade *versus* registros.

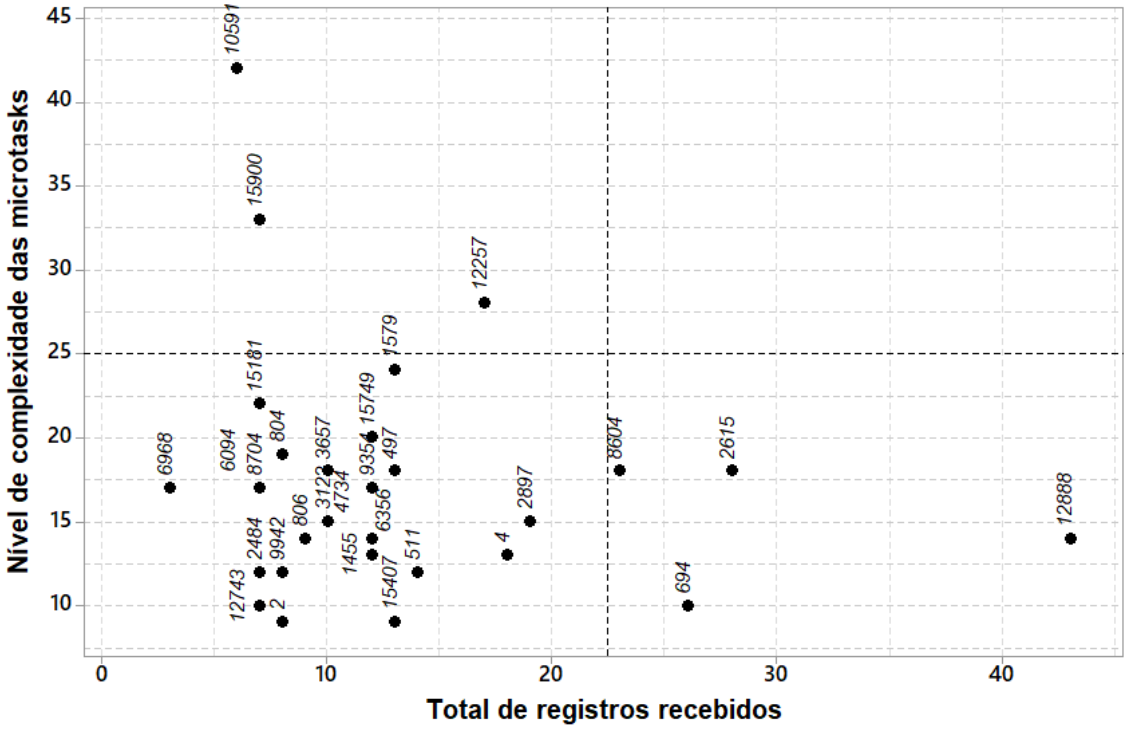

Figura 5. Dispersão entre complexidade e registros das *microtasks*

Percebeu-se que os dois quadrantes inferiores - esquerdo e direito - concentraram o maior índice de registros de *microtasks*. Não obstante, esses quadrantes representaram a disposição das *microtasks* consideradas mais simples mostrando que tarefas com menor grau de dificuldade receberam organicamente mais registros que as atividades complexas.

Os quadrantes superiores – esquerdo e direito – mostram a incidência de apenas três registros. Isso exibiu que as *microtasks* mais complexas, além de possuírem menor atenção dos *crowd workers*, não conseguiram ultrapassar cerca de 20 registros. Isso é exposto pelo quadrante superior direito, que ficou vazio, expondo que não houve nenhuma ocorrência de *microtask* complexa com mais de 20 registros.

Após analisar o índice de registro foi analisado o índice de submissão. O que se buscou observar foi identificar se as *microtasks* classificadas como simples receberiam

mais submissões que as *microtasks* consideradas complexas, e o gráfico de dispersão que ilustra esse diagnóstico é encontrado na Figura 6.

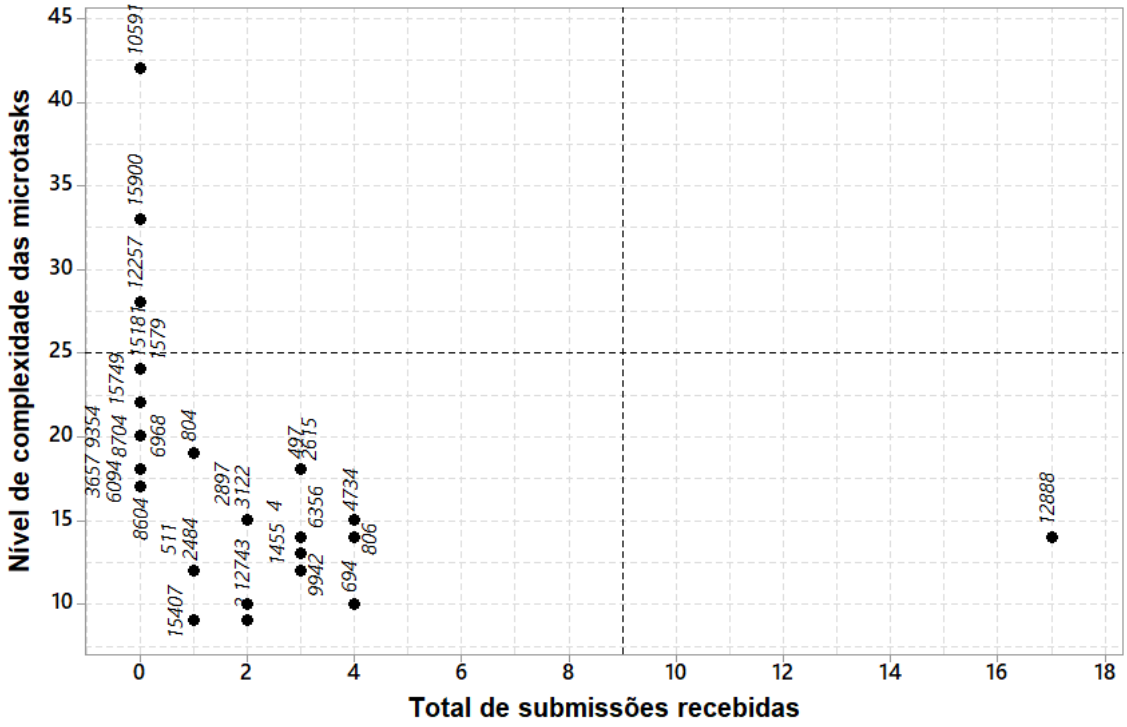

Figura 6. Dispersão entre complexidade e submissões das *microtasks*

A tendência natural das *microtasks* mais simples concentraram maior índice de submissão também foi satisfeita. Observou-se que tais tarefas, expostas nos quadrantes inferiores, alcançaram um índice entre 1 e 5 submissões. Destacando que a simplicidade, além de ter angariado maior registro também alcançou uma ou mais submissões. Isso é diametralmente oposto aos quadrantes superiores em que as três *microtasks* mais complexas não alcançaram nenhuma submissão.

6.4. Discussão

As QPs que nortearam este trabalho investigaram como a complexidade influenciou e afetou a finalização, o registro e a submissão em cada *microtask* analisada. O foco dessa análise foi avaliar se o método proposto seria eficiente em mensurar a complexidade e identificar a sua influência na execução de uma *microtask*.

Os resultados obtidos com a investigação da QP_1 demonstraram que o método proposto havia classificado corretamente as *microtasks* em referência a sua complexidade e sua taxa de finalização. Essa percepção foi alcançada, principalmente, devido aos índices alcançados na amostra analisada. O que foi notado durante essa análise é que as *microtasks* classificadas como simples pelo método alcançaram quase 90% de conclusão, cenário contrário ao das *microtasks* complexas, que tiveram pouco mais de 13% de finalização.

Ainda tratando-se da análise da QP_1 também é pertinente destacar que houve casos amostrais que não foram classificados corretamente. Duas *microtasks* simples não foram executadas e duas *microtasks* complexas lograram êxito em sua finalização. O aprofundamento desse tópico demonstrou que poderiam existir outras variáveis

influenciando a própria complexidade da *microtask*, como o tipo da tarefa (teste, desenvolvimento ou *design*), além da falta de atenção aos requisitos obrigatórios. Apesar dessas constatações o método proposto foi capaz de constatar a complexidade corretamente nas outras 26 amostras analisadas, demostrando que o mesmo pode ser aperfeiçoado futuramente para tratar tais questões.

No caso da análise da QP_2 notou-se como a complexidade de uma *microtask* também atuou no índice de registros e de submissões. E, novamente, o método foi eficiente ao classificar as *microtasks* como complexas ou simples e identificar essa influência. Os resultados exibiram que as tarefas mais simples receberem maior atenção dos *crowd workers*, e naturalmente, tiveram maiores índices de submissões. O que remete a sua taxa de sucesso de finalização. A respeito dessa questão ainda é pertinente destacar que as *microtasks* complexas podiam até atrair participantes, entretanto, o fluxo de submissão era extremamente baixo.

Nesse sentido ainda é pertinente destacar uma dinâmica do modelo de desenvolvimento de software CS e sua relação com a complexidade. Em uma ponta do modelo residem os *crowdsourcers,* que buscam solucionar seus desafios via *microtasks*. Já na outra extremidade os *crowd workers* desejam submeter soluções para angariar as premiações e as conquistas. Em ambos os casos, porém, o que se notou por meio da condução deste estudo é que todos podem alcançar melhores resultados através da simplificação das *microtasks*. No caso dos *crowdsourcers*, as *microtasks* simples aumentam as chances de êxito na finalização. Já os *crowd wokers* despertam a sua atenção e foco movidos pela facilidade de submissão que uma tarefa simples emprega. Tornando assim o modelo mais sustentável e eficiente no ponto de vista de todos os participantes, inclusive da própria plataforma de desenvolvimento, que pode atingir melhores índices de sucesso.

6.5. Limitações

A condução deste estudo possui limitações que podem impactar diretamente na validade das conclusões obtidas. A despeito disso é relevante discuti-las, e sob tais circunstâncias, foram sumarizados os seguintes aspectos de validade propostos por [Wohlin *et al*. 2012]:

- **Validade de construção:** refletir se os resultados expostos nas QPs estão diretamente relacionados ao que foi observado. Nesse ponto é importante destacar que foi realizada uma investigação nos metadados das *microtasks*, descartando outras análises, como entrevistas ou questionários, que poderiam expandir a concepção sobre o motivo de duas *microtasks* consideradas simples terem sido canceladas e de duas *microtasks* consideradas complexas serem executadas com sucesso.
- **Validade interna:** analisar a possibilidade de fatores desconhecidos ou desconsiderados atuarem na amostra analisada. Nessa perspectiva pode-se destacar que não foi possível assumir que as *microtasks* analisadas tiveram o mesmo destaque durante a etapa de registro. Ou seja, a plataforma X pode ter realizado campanhas de *marketing* patrocinado por clientes, deixando determinadas *microtasks* com maior destaque de visualização (como *banners*, propagandas ou anúncios no *site*), atraindo assim maior atenção dos *crowd workers* e recebendo maior índice de registro.

- **Validade externa:** verificar até que ponto os resultados obtidos podem ser generalizados. Sob essa perspectiva a principal limitação refere-se ao tamanho amostral que pode dificultar a generalização da eficácia do método para outros ambientes de desenvolvimento de software CS.
- **Confiabilidade:** identificar a dependência entre os dados obtidos e a análise efetuada. Naturalmente, nesse cenário surge uma limitação imposta pela plataforma. Assim, para replicações futuras deve-se obter uma coleção de *microtasks* com disposição similar dos metadados utilizados nesta pesquisa. Ademais, a disponibilidade da *baseline* é de responsabilidade da própria plataforma X e independe aos autores desta pesquisa.

7. Conclusão e Trabalhos Futuros

Este trabalho apresentou um método para determinar a complexidade de uma *microtask*. De modo resumido, foi gerado um cálculo de complexidade, avaliado por meio de um estudo de caso em uma plataforma de desenvolvimento de software CS.

Os resultados mostraram que o método aproximou-se de 90% de assertividade na amostra analisada, evidenciando que as *microtasks* mais simples tinham um índice de sucesso bem superior as *microtasks* com maior complexidade. Além disso, também foi descoberto que o aumento da dificuldade de uma *microtask* diminuía diametralmente o seu índice de registro e de submissão.

O maior impacto dos resultados obtidos nesta pesquisa contribuiu em dois aspectos: o primeiro, e já evidenciado neste estudo, é o fato de ainda não existirem – até o momento da escrita deste texto – abordagens, métodos ou cálculos eficientes e dedicados a determinar a complexidade de uma *microtask* em ambientes CS. Sendo assim, o método proposto neste estudo representa uma inovação para uma área da literatura ainda incipiente e desabastecida. O segundo aspecto reflete sobre a facilidade e portabilidade de aplicação do método proposto. Como foi demonstrando, o método foi capaz de medir a complexidade de *microtasks* de diferentes tipos, alcançado alta aderência em seu objetivo de determinar a complexidade independente da natureza de cada tarefa.

Em relação aos trabalhos futuros deseja-se automatizar o cálculo de complexidade por meio de uma aplicação *online*, além disso, existe o interesse na realização de novas pesquisas para amadurecer o método proposto, integrando soluções ao total de requisitos e tipos de *microtasks*.

References

Abhinav, K., Dubey, A., Jain, S., Bhatia, G. K., McCartin, B., and Bhardwaj, N. (2018). Crowdassistant: A virtual buddy for crowd worker. In 2018 IEEE/ACM 5th International Workshop on Crowd Sourcing in Software Engineering (CSI-SE), pages 17–20. doi: https://doi.org/10.1145/3195863.3195865 [GS Search]

Aipe, A. and Gadiraju, U. (2018). Similarhits: Revealing the role of task similarity in microtask crowdsourcing. HT. [GS Search]

Chandra, G., Gupta, D. L., and Malviya, K. (2012). Some observations based on comparison of mood and ck software metrics suites for object oriented system. International Journal of Computer Science and Technology, 3(3).

de Deus, W. S., Barros, R. M., and L'Erario, A. (2016). Um modelo para o gerenciamento do crowdsourcing em projetos de software. In I Workshop sobre Aspectos Sociais, Humanos e Economicos de Software (WASHES'2016). [GS Search]

de Deus, W. S., Fabri, J. A., and L'Erario, A. (2017). The management of crowdsourcing software projects: A systematic mapping. In 2017 12th Iberian Conference on Information Systems and Technologies (CISTI), pages 1–7. doi: https://doi.org/10.23919/CISTI.2017.7975711 [GS Search]

de Freitas Junior, M., Fantinato, M., and Sun, V. (2015). Improvements to the function point analysis method: A systematic literature review. IEEE Transactions on Engineering Management, 62(4):495–506. doi: https://doi.org/10.1109/TEM.2015.2453354 [GS Search]

Dubey, A., Abhinav, K., Taneja, S., Virdi, G., Dwarakanath, A., Kass, A., and Kuriakose, M. S. (2016). Dynamics of software development crowdsourcing. In 2016 IEEE 11th International Conference on Global Software Engineering (ICGSE), pages 49–58. doi: https://doi.org/10.1109/ICGSE.2016.13 [GS Search]

Dwarakanath, A., Chintala, U., Shrikanth, N. C., Virdi, G., Kass, A., Chandran, A., Sengupta, S., and Paul, S. (2015). Crowd build: A methodology for enterprise software development using crowdsourcing. In 2015 IEEE/ACM 2nd International Workshop on CrowdSourcing in Software Engineering, pages 8–14. doi: https://doi.org/10.1109/CSI-SE.2015.9 [GS Search]

Goke, N. and Freitag, E. (2014). Microtask platforms a win/win/win situation. In ¨ Colletive Inteligence. Greicius, T. (2018). Multi-planet system found through crowdsourcing. disponivel em: https://www.nasa.gov/feature/jpl/multi-planet-system-found-throughcrowdsourcing. NASA.

Hosseini, M., Phalp, K., Taylor, J., and Ali, R. (2014). The four pillars of crowdsourcing: A reference model. In 2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS), pages 1–12. doi: https://doi.org/10.1109/RCIS.2014.6861072 [GS Search]

Howe, J. (2006a). Crowdsourcing: A definition. https://crowdsourcing.typepad.com/cs/2006/06/crowdsourcing_a.html, Janeiro, 2019.

Howe, J. (2006b). The rise of crowdsourcing. Wired magazine, 14(6):1–4. https://www.wired.com/2006/06/crowds/, Janeiro, 2019.

Jacques, J. T. (2018). Microtask Design: Value, Engagement, Context, and Complexity. PhD thesis, University of Cambridge. doi: http://dx.doi.org/10.17863/CAM.18777 [GS Search]

Karim, M. R., Yang, Y., Messinger, D., and Ruhe, G. (2018). Learn or earn? - intelligent task recommendation for competitive crowdsourced software development. doi: http://dx.doi.org/10.24251/HICSS.2018.700 [GS Search]

Kittur, A., Smus, B., Khamkar, S., and Kraut, R. E. (2011). Crowdforge: Crowdsourcing complex work. In Proceedings of the 24th Annual ACM Symposium on User Interface

Software and Technology, UIST '11, pages 43–52, New York, NY, USA. ACM. doi: https://doi.org/10.1145/2047196.2047202 [GS Search]

Kurve, A., Miller, D. J., and Kesidis, G. (2015). Multicategory crowdsourcing accounting for variable task difficulty, worker skill, and worker intention. IEEE Transactions on Knowledge and Data Engineering, 27(3):794–809. doi: https://doi.org/10.1109/TKDE.2014.2327026 [GS Search]

LaToza, T. D., Towne, W. B., Adriano, C. M., and van der Hoek, A. (2014). Microtask programming: Building software with a crowd. In Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology, UIST '14, pages 43–54, New York, NY, USA. ACM. doi: https://doi.org/10.1145/2642918.2647349 [GS Search]

LaToza, T. D. and v. d. Hoek, A. (2015). A vision of crowd development. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, volume 2, pages 563–566. doi: https://doi.org/10.1109/ICSE.2015.194

Mao, K., Capra, L., Harman, M., and Jia, Y. (2017). A survey of the use of crowdsourcing in software engineering. Journal of Systems and Software, 126:57 – 84.

Mao, K., Yang, Y., Li, M., and Harman, M. (2013). Pricing crowdsourcing-based software development tasks. In 2013 35th International Conference on Software Engineering (ICSE), pages 1205–1208. doi: https://doi.org/10.1016/j.jss.2016.09.015 [GS Search]

Naik, N. (2016). Crowdsourcing, open-sourcing, outsourcing and insourcing software development: A comparative analysis. In 2016 IEEE Symposium on Service-Oriented System Engineering (SOSE), pages 380–385. doi: https://doi.org/10.1109/SOSE.2016.68 [GS Search]

Nakatsu, R. T., Grossman, E. B., and Iacovou, C. L. (2014). A taxonomy of crowdsourcing based on task complexity. Journal of Information Science, 40(6):823–834.

Sui, D., Elwood, S., and Goodchild, M. (2012). Crowdsourcing Geographic Knowledge: Volunteered Geographic Information (VGI) in Theory and Practice. Springer Publishing Company, Incorporated. doi: https://www.doi.org/10.1007/978-94-007-4587-2

TopCoder (2017). Projects - topcoder crowdsourcing.

Tranquillini, S., Daniel, F., Kucherbaev, P., and Casati, F. (2015). Modeling, enacting, and integrating custom crowdsourcing processes. ACM Trans. Web, 9(2). Doi: https://doi.org/10.1145/2746353 [GS Search]

Wei Li, Michael N. Huhns, W.-T. T. W. W. (2015). Crowdsourcing Cloud-Based Software Development. Springer, first edition. doi: https://www.doi.org/10.1007/978-3-662-47011-4

Winkler, D., Sabou, M., Petrovic, S., Carneiro, G., Kalinowski, M., and Biffl, S. (2017). Improving Model Inspection with Crowdsourcing. IEEE/ACM 4th International Workshop on Crowdsourcing in Software Engineering (CSI-SE), pages 30-40. doi: http://doi.ieeecomputersociety.org/10.1109/CSI-SE.2017.2 [GS Search]

Wohlin, C., Runeson, P., Host, M., Ohlsson, M. C., Regnell, B., and Wesslen, A. (2012). Experimentation in software engineering. Springer Science & Business Media. doi: https://www.doi.org/10.1007/978-3-642-29044-2

Xiao, L. and Paik, H. Y. (2014). Supporting complex work in crowdsourcing platforms: A view from service-oriented computing. In 2014 23rd Australian Software Engineering Conference, pages 11–14. doi: https://doi.org/10.1109/ASWEC.2014.11 [GS Search]

Yang J. Y. Redi, J. Demartini, G. B. A. (2016). Modeling task complexity in crowdsourcing. Fourth AAAI Conference on Human Computation and Crowdsourcing.

Yin, R. K. (2015). Estudo de Caso: Planejamento e Métodos. Bookman editora.