UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

VICTOR DE ALMEIDA PICCOLI FERREIRA

Design Patterns para Data Science

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação

Orientador: Prof. Dr. Marcelo Soares Pimenta

Porto Alegre
2021

RESUMO

O uso de *design patterns* já é bem difundido em engenharia de *software* e arquitetura de sistemas. O mesmo não pode ser dito para projetos na área de *Data Science* e, consequentemente, *Machine Learning*. Este trabalho investiga, a partir da revisão de literatura, alguns *design patterns* direcionados a projetos de *data science*, organizando suas descrições com definições e exemplos para ilustrar os seus usos e relata uma avaliação de suas efetividades conduzida com profissionais que atuam como cientistas de dados.

Palavras-chave: Padrões de Design. Ciência de dados. Aprendizado de máquina. Engenharia de software.

Design patterns for Data Science

ABSTRACT

The use of design patterns is already widespread in software engineering and systems architecture. The same cannot be said for projects in the Data Science area and, consequently, Machine Learning. This work investigates, from literature review, some design patterns directed to Data Science projects, organizing their description with definitions and examples to illustrate their uses and reports an evaluation of their effectiveness conducted with professionals who act as data scientists.

Keywords: Design Patterns. Data Science. Machine Learning. Software engineering.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

DS	Data Science
ML	Machine Learning
DP	Design Patterns
API	Application Programming Interface

SUMÁRIO

1 INTRODUÇÃO

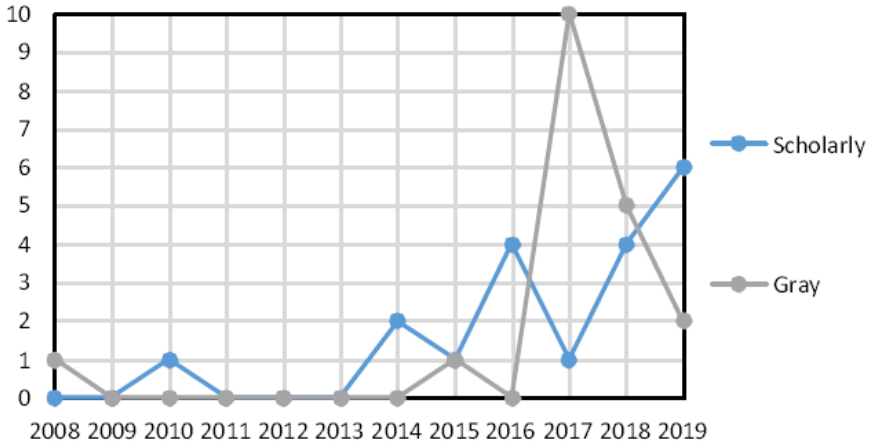
Uma definição, dentre as muitas definições, para o termo *pattern* é uma solução fixa para um problema que acontece repetidas vezes dentro de um contexto específico (DEARDEN; FINLAY, 2006). No contexto de *software*, um *design pattern* seria uma solução sugerida em formato de estrutura de código para um problema de qualidades que são desejadas, em relação a construção e comportamento de um *software* (DEARDEN; FINLAY, 2006). Estas soluções para problemas recorrentes têm o grande benefício de melhorar a comunicação entre desenvolvedores, facilitar o aprendizado de iniciantes na área de desenvolvimento de *software* e melhorar a colaboração entre integrantes de equipe (BORCHERS, 2000)

No contexto de dados, a grande quantidade de dados disponível é algo que as empresas de hoje em dia estão utilizando para ganhar uma vantagem competitiva nas indústrias. Entretanto, este grande volume de dados é impossível de ser analisado manualmente, necessitando de algoritmos e formatos de dados específicos para suas análises (PROVOST; FAWCETT, 2013). A esta área que lida com os problemas relacionados a princípios, processos e técnicas para a interpretação e análise de dados dá-se o nome de *data science* (DS) (PROVOST; FAWCETT, 2013).

Com o avanço da área de *data science* e suas áreas relacionadas como *machine learning* (ML) e *data analytics*, um custo associado está crescendo: a manutenção dos sistemas a longo prazo (SCULLEY et al., 2015). Este custo, associado ao problema de como compartilhar as melhores práticas e os conhecimentos referentes a problemas recorrentes, pode ser atenuado pela aplicação de *design patterns* (DP) direcionados à área de *data science* (PETROV, 2018).

Apesar da popularidade das áreas de *machine learning* e *data science*, há poucos estudos de *patterns* aplicáveis à implementação de sistemas de análise de dados (WASHIZAKI et al., 2019), possivelmente pelo grande escopo da *data science* e seus estudos relacionados (PETROV, 2018). Mesmo com esse número pequeno de estudos relacionados a *design patterns* em projetos de *data science*, observa-se um crescimento na quantidade de estudos focados em temas relacionados ao longo dos anos, descrito pela Figura 1.1. O gráfico da Figura 1.1 ilustra o número de documentos de *design patterns* para *machine learning* ao longo dos anos, sendo os documentos de literatura cinzenta (materiais produzidos em meios não convencionais e não acadêmicos) descritos por "*Gray*" e os documentos de literatura acadêmica descritos por "*Scholarly*".

Figura 1.1: Número de documentos relacionados a *design patterns* para *machine learning* ao longo dos anos.



Fonte: (WASHIZAKI et al., 2020)

1.1 Objetivo

O objetivo deste trabalho é estudar a literatura relacionada a melhores práticas de organização, desenvolvimento e manutenção de projetos de *data science*, selecionar, definir e exemplificar alguns *patterns* propostos na literatura e, finalmente, avaliar os *patterns* definidos, a partir de uma análise de opinião de profissionais da área. A partir desse objetivo, pode-se definir os seguintes objetivos específicos:

1. Investigar na literatura a existência *design patterns* relacionados a *data science*.
2. Definir e ilustrar com pequenos exemplos *design patterns* de *data science* a partir da revisão de literatura sobre *data science*, bem como de outras áreas relacionadas a *software*.
3. Avaliar a efetividade dos *patterns* definidos.

1.2 Estrutura do texto

Este trabalho está estruturado em 7 capítulos. O capítulo 1 introduz o tema de estudo e traz os objetivos geral e específicos que nortearam o trabalho. O capítulo 2 apresentará a fundamentação teórica, na qual conceitos essenciais para o entendimento do trabalho serão descritos, em especial a definição e formato de *design patterns*, explicação sobre *data science*, *machine learning* e apresentação do conceito de *design patterns* para *data science*. No capítulo 3, serão apresentados os trabalhos relacionados encontrados na

revisão de literatura sobre o tema *design patterns* para *data science* em específico. No capítulo 4, será explicada a metodologia utilizada para o desenvolvimento do trabalho, apresentando também o modelo utilizado para a descrição de cada *pattern*, incluindo o processo de validação destes a partir de um questionário *online* disponibilizado para profissionais da área de *data science*. No capítulo 5, os *patterns* selecionados a partir da revisão da literatura serão descritos detalhadamente utilizando o modelo explicado no capítulo prévio. No capítulo 6, será feita uma análise das respostas do questionário *online*, a partir de uma avaliação preliminar da relevância de cada *pattern*, investigando a clareza de escrita e de seus objetivos, a capacidade de resolução de problemas recorrentes, a utilidade e a praticidade de seu uso. Por fim, no capítulo 7 serão comentados contribuições e resultados deste estudo, além de pontuar suas limitações, trazendo sugestões de trabalhos futuros relacionados ao tema.

2 FUNDAMENTAÇÃO TEÓRICA

Essa seção explica os conceitos necessários para o entendimento do trabalho.

2.1 Design patterns

Cristopher Alexander propôs, na década de 70, a ideia de *patterns* na arquitetura. Essa ideia teve como princípio a reutilização de boas práticas da arquitetura para produzir designs de qualidade (ALEXANDER, 1977). No contexto de *software*, após a maturidade da área, a documentação de boas práticas começou a ser valorizada e descrita (GAMMA et al., 1995).

Design patterns são técnicas que devem ser testadas para ter seu valor comprovado (SCHMIDT; FAYAD; JOHNSON, 1996) e são eficientes quando utilizados em conjunto com outros *patterns* (PETROV, 2018). Os *patterns* são o resultado de observações práticas, e podem ser descobertos via soluções de design, geralmente originadas por tentativa e erro e por observação (DEARDEN; FINLAY, 2006).

Quando se fala de *patterns*, a ideia geral não é a de regras que não podem ser quebradas, mas sim de sugestões de técnicas que foram comprovadas como boas práticas para um desenvolvimento saudável de um projeto (SCHMIDT; FAYAD; JOHNSON, 1996). Os *patterns* em si não são práticas sem um custo envolvido, há sempre uma troca envolvida em termos de qualidade, um foco maior numa característica do projeto pode acarretar em uma perda de qualidade em outra característica do projeto (AMPATZOGLOU; CHARALAMPIDOU; STAMELOS, 2013).

Para descrever cada *pattern*, necessita-se de um formato. No famoso livro sobre *patterns* para *software* orientado a objetos, Gamma et al. (GAMMA et al., 1995) descreveram os elementos essenciais de cada *pattern* como:

1. O **nome do *pattern*** usado para explicar um problema e suas soluções e consequências em poucas palavras.
2. O **problema** em que o *pattern* deve ser aplicado, explicando o problema e seu contexto.
3. A **solução** descrevendo os elementos de design, seus relacionamentos, responsabilidades e colaborações. Essa solução não é uma descrição de uma implementação concreta, mas sim uma solução geral de organização de elementos para um pro-

blema abstrato.

4. As **consequências** de aplicação do *pattern*. Essas seriam as trocas de qualidades envolvidas ao utilizar-se do *pattern*. Elas são críticas para entender os custos e benefícios ao aplicar cada *pattern*.

2.2 Data science

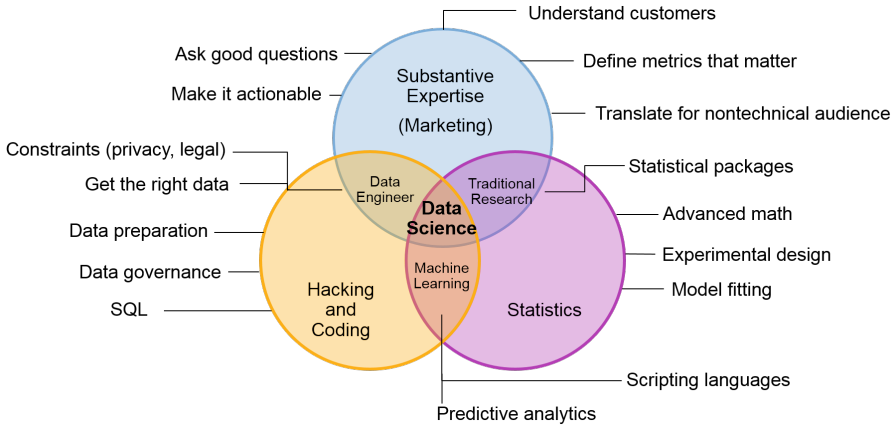
Data science pode ser definida como a arte de extrair conhecimento e significado a partir de um conjunto dados (PROVOST; FAWCETT, 2013). Ela é uma área que envolve várias disciplinas como estatística, informática, computação, comunicação, sociologia e manejo de dados (CAO, 2017).

Data science envolve a automatização de processos análise e manipulação de dados (PROVOST; FAWCETT, 2013). Essas análises são cruciais para a tomada de decisões baseadas em dados (PROVOST; FAWCETT, 2013), bem como para a apresentação de dados num formato significativo (DONOHO, 2017).

Geralmente, nos projetos de *data science*, os dados brutos são visualizados para descrever uma hipótese, que é investigada. Como *data science* tem uma natureza experimental, os resultados finais podem indicar que um novo caminho deve ser investigado e a hipótese inicial deve ser reformulada. A ideia geral é que *data science* é uma ciência empírica, atuando diretamente em dados (CHANG; GRADY et al., 2015).

Com uma grande quantidade de dados, é possível analisá-los para extrair padrões preditivos (JORDAN; MITCHELL, 2015). Para obter essa grande quantidade de dados, é necessário uma mineração de dados, preferencialmente de qualidade (LAVRAČ et al., 2004). Com esse grande escopo, *data science* é uma área que envolve muitos conceitos e engloba assuntos como *big data*, *machine learning*, *data mining* e até mesmo gestão de projetos (CAO, 2017). Isso leva à caracterização de *data science* como uma área composta de engenheiros e cientistas (PETROV, 2018) utilizando uma estrutura de princípios para extrair informações importantes de um grande volume de dados (PROVOST; FAWCETT, 2013). A Figura 2.1 descreve as áreas de conhecimento que *data science* engloba.

Figura 2.1: Diagrama Venn sobre *data science*.



Fonte: (EUBANKS, CHRISTI, 2016)

2.3 Machine learning

Machine learning é uma área de grande crescimento no escopo de *data science*, por conta de sua capacidade de produzir bases de conhecimento que não existiam ou melhorar as bases que já existiam (LAVRAČ et al., 2004). A ideia geral da área de *machine learning* é a de construir sistemas que melhoram automaticamente por meio de experiência (JORDAN; MITCHELL, 2015).

O objetivo primário de *machine learning* é desenvolver algoritmos de propósito geral, que resolvem um problema prático. Mais importante, o objetivo de um algoritmo de aprendizado é produzir um resultado que é o mais preciso possível (GOLLAPUDI, 2016). Para o aprendizado dos sistemas de *machine learning*, são necessários dados. Esses dados podem estar em qualquer formato, podem ser recebidos a qualquer frequência e podem ser de qualquer tamanho (GOLLAPUDI, 2016).

As saídas das implementações de *machine learning* são chamadas de modelos. Em vários casos, os modelos são aplicados em novos conjuntos de dados para que o modelo aprenda novos comportamentos e também faça a predição desses dados (GOLLAPUDI, 2016). A capacidade preditiva dos modelos de *machine learning* é dependente na qualidade e volume dos dados que são utilizados para o treino dos modelos (JORDAN; MITCHELL, 2015). No contexto de *data science*, *machine learning* se encaixa no processo de criação de modelos preditivos (DONOHO, 2017).

É importante salientar que a natureza de projetos de *machine learning*, assim como a de *data science*, é exploratória e baseada em hipóteses que podem ser descartadas, repensadas e modificadas. Por exemplo, em um dado projeto pode-se descobrir uma fonte mais significativa de dados dos que estavam sendo utilizados, um modelo pode ser

repensado, melhores algoritmos podem ser descobertos. Todas essas situações acarretam em novas iterações de um dado projeto (AMERSHI et al., 2019).

Quando se fala em *data science*, inerentemente está sendo discutido *machine learning* e as outras áreas que *data science* engloba (PETROV, 2018). Um número crescente documentos de *design patterns* para *machine learning* vem sendo escrito (WASHIZAKI et al., 2019), o que demonstra um crescimento no interesse em documentar processos que podem ser reaplicados em outros projetos para resolver problemas recorrentes em projetos de *data science*.

2.4 Design patterns para data science

A partir das definições de *design patterns* e *data science*, podemos definir *design patterns* para *data science* como: práticas de design, definidas para resolver um problema recorrente em projetos, que podem ser reutilizadas em diversos projetos que envolvem a extração de conhecimento e significado a partir do estudo de dados (PETROV, 2018).

3 TRABALHOS RELACIONADOS

Após pesquisas em ferramentas de busca de literatura como Google Scholar e Scopus, utilizando-se de palavras-chave como "*data science design patterns*", "*data science*" e "*design patterns*" foi possível observar que há poucos trabalhos relacionados diretamente a *design patterns* para *data science*. Especificamente, foi encontrado apenas um estudo (PETROV, 2018) que faz uma revisão da literatura formal e informal para definir e exemplificar por meio de código *patterns* para projetos em *data science*, e o livro de Todd Morley (Morley, Todd, 2020), que ainda não foi publicado. Este trabalho busca definir e avaliar novos *patterns* diretamente relacionados a *data science*, expandindo a lista de *patterns* já definidos em (PETROV, 2018). Quando se limita o escopo das pesquisas para as áreas relacionadas a *data science*, é possível encontrar uma série de estudos relevantes ao tópico de *design patterns*. Uma explicação para essa situação pode ser dada pelo tamanho do escopo da área de *data science* (PETROV, 2018).

Ao considerar as áreas relacionadas a *data science*, uma quantidade maior de literatura de *patterns* é encontrada. Em (ARENAS et al., 2019), o foco do trabalho é definir *patterns* de segurança em projetos de *data science* na nuvem, diferindo deste trabalho em seu objetivo, que é definir *patterns* relacionados a uma melhor longevidade de projetos. Na área de dados, há trabalhos como (DHAR; MAZUMDAR, 2014), em que há um interesse em apresentar melhores práticas em ambientes de *big data*, (PETRASCH, 2019), sobre *patterns* de integração de dados para mapeamento de dados e o livro (RIDGE, 2014), que dá soluções práticas para situações em projetos de *data analytics*. O presente trabalho tem a intenção de descrever *patterns* que estão disponíveis na literatura e avaliar estes em sua utilidade e capacidade em resolver problemas recorrentes em projetos de *data science*.

Uma outra área relacionada a *data science* em que está crescendo um interesse na definição de *patterns* é a de *machine learning*, essa tendência podendo ser vista em trabalhos como (WASHIZAKI et al., 2019), (AMERSHI et al., 2019), (SCULLEY et al., 2015) e no livro recentemente publicado *"Machine Learning Design Patterns"* (LAKSHMANAN, VALLIAPPA AND ROBINSON, SARA AND MUNN, MICHAEL, 2020). Os *patterns* que buscam ser definidos neste trabalho se aplicam a sistemas de *machine learning*; todavia, não são apenas aplicáveis a esse tipo de sistema, e sim a projetos de *data science* em geral.

Este trabalho procura expandir a lista de *patterns* para *data science* definido em

(PETROV, 2018), utilizando um método similar para a pesquisa e definição de *patterns*, porém apenas tendo a pesquisa dos materiais na literatura acadêmica. O intuito de apenas se basear na literatura acadêmica foi de se ter fontes mais consolidadas, com materiais que já foram validados de alguma forma.

4 METODOLOGIA

Para a obtenção dos resultados deste trabalho, uma pesquisa de caráter exploratório foi abordada, com o intuito de descobrir processos, técnicas e metodologias que possam ser possíveis candidatos a se tornarem *patterns*. Esse tipo de pesquisa normalmente envolve levantamento bibliográfico, entrevistas com pessoas que tiveram experiências práticas com o problema pesquisado e análise de exemplos que estimulem a compreensão, com o objetivo de proporcionar maior familiaridade com o problema (GIL et al., 2002).

Após a obtenção de candidatos a *patterns*, esses tiveram suas descrições definidas e exemplos criados. O formato que foi utilizado para descrever cada *pattern* foi baseado numa versão do formato proposto em (PETROV, 2018). Para a descrição, o seguinte modelo foi utilizado: **Nome** do *pattern*; **Problema** descrevendo uma situação que será considerada; **Fatos** sendo examinados para a solução de um problema e porque é difícil de resolver esse problema; **Solução** detalhando o caminho escolhido para resolver um problema, levando em conta os fatos; **Consequências** positivas e negativas ao utilizar esta solução, baseadas na experiência do autor e na literatura relacionada; **Exemplos** ilustrando o seu uso.

O valor dos *patterns* vêm da sua capacidade de serem utilizados para resolver problemas. Um *pattern* pode ser documentado, mas seu valor só é conhecido após ele ter sido testado (SCHMIDT; FAYAD; JOHNSON, 1996). Com o intuito de avaliar os *patterns*, após a descrição, eles foram disponibilizados a comunidades de *data science* na internet por meio de um formulário estruturado *online* utilizando uma escala Likert. Essa escala é descrita como tendo o propósito de entender as opiniões/percepções dos participantes sobre um único fenômeno de interesse ou variável (JOSHI et al., 2015). No caso deste trabalho, usa-se esse método com o objetivo de avaliar a utilidade de cada *pattern* definido e de verificar sua capacidade de resolver problemas recorrentes.

5 DESCRIÇÃO DOS *PATTERNS* PARA *DATA SCIENCE*

Após a inspeção da bibliografia, explicada no capítulo anterior, uma série de DP emergiram através da observação de técnicas e metodologias descritas na literatura. Mais especificamente, sete candidatos a *pattern* foram encontrados. Este capítulo constrói a definição de cada um desses candidatos e descreve-os em uma estrutura comum.

5.1 Pattern 1: Qualidade de dados

Problema

Como trabalhos de DS e *analytics* normalmente são exploratórios, não padronizados, não automatizados e personalizados e têm como objetivo de descobrir evidências para tomadas de decisões, eles sofrem de problemas de qualidade como validade, veracidade, variabilidade e confiabilidade dos dados (CAO, 2017).

Fatos

- Dados estão constantemente mudando. Após um certo período de tempo algum dado que era válido pode não ser mais relevante.
- Dados são incertos. Cada fonte de dados tem um nível de confiabilidade, deve-se assumir um grau de risco associado à veracidade dos dados providos para um projeto.
- A qualidade dos dados determinam a qualidade da saída (*Garbage in, garbage out*) (CAO, 2017).

Solução

A validade dos dados e da análise determina se um modelo de dados, conceito, conclusão ou medida são bem fundamentados e correspondem exatamente às características dos dados e fatos do mundo real, tornando-o capaz de fornecer resposta correta. Da mesma forma, a veracidade de dados e análises determina a correção e precisão de dados e análises de resultados. A escolha de dados que tenham maior validade e veracidade são determinantes na qualidade final do produto de processamento dos dados (CAO, 2017).

Consequências

Positivas:

- Maior validade dos dados. A escolha de dados assume que os dados escolhidos foram validados de alguma forma e são relevantes para o projeto.

- Menor variabilidade de dados. Ao retirar dados com menor validade ou veracidade, o conjunto de dados torna-se mais enxuto, formado por registros mais relevantes.

Negativas:

- Dados bons não necessariamente geram resultados bons (CAO, 2017).
- Os resultados obtidos nem sempre são significativos. Por causa das limitações de sistemas de dados existentes, projeções do mundo físico nem sempre podem englobar todo o problema, não sendo possível de se visualizar todo o contexto por qualquer tipo de exploração de dados (CAO, 2017).

Exemplos

Ao considerar um projeto fictício que requer uma lista de habilidades técnicas necessárias para um cargo, uma certa experiência em algo que era relevante há alguns anos atrás já não pode mais ser relevante hoje em dia. Caso se utilize um conjunto de dados que contém habilidades técnicas comumente utilizadas para desenvolver um software, antigamente estes dados provavelmente envolveriam conhecimento em linguagens de programação como COBOL, Fortran ou Pascal. Comparado com os conhecimentos mais utilizados hoje em dia, que são linguagens em como Javascript, Java e C# (Lestal, Justin, 2020), pode-se perceber que houve uma mudança nos dados que são considerados relevantes e que os resultados de um projeto que requer esses tipos de dados pode gerar resultados não significativos caso a validade dos dados não for considerada.

A distribuição demográfica também é importante, pois os dados podem variar de acordo com a região. Se um programa tivesse como objetivo sugerir uma progressão de cargos, um trabalho que está em alta nos Estados Unidos poderia não ser relevante como próximo emprego na Índia. A veracidade dos dados pode estar atrelada a específicas regiões, sendo importante a análise e escolha de dados significativos para cada situação.

5.2 Pattern 2: Modelo de dados canônico

Problema

Quando lidamos com a junção de dados de fontes de dados diferentes, os dados geralmente têm formatos diferentes, porque cada formato foi desenhado com só uma aplicação em mente (HOHPE; WOOLF, 2004). Assim, pode ocorrer situações em que dados equivalentes têm representações diferentes ou formatos diferentes.

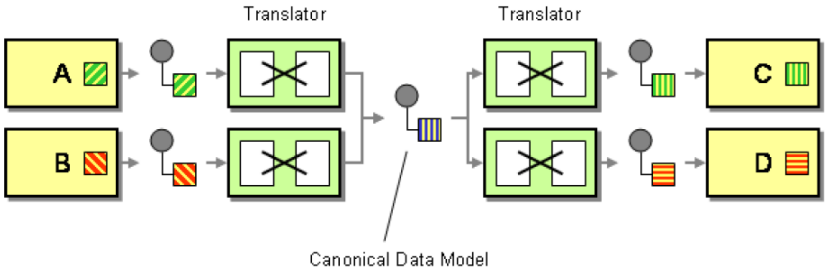
Fatos

- Cada aplicação utiliza o formato de dados que melhor funciona para sua função.
- Formatos de dados podem variar substancialmente dependendo da sua origem e aplicação. Uma junção de dados de fontes distintas pode ser algo difícil de coordenar sem que os resultado da junção fique excessivamente complexo.
- Dados equivalentes podem ter representações diferentes.
- Dados independentes de aplicação têm formato menos variáveis e serem mais estáveis.

Solução

Modelo de dados canônico, onde os dados são independentes de aplicações. Cada aplicação deve produzir e consumir conteúdo nesse formato (HOHPE; WOOLF, 2004). É necessário que cada programa traduza o conteúdo do modelo canônico para consumir os dados. Se cada conjunto de dados relacionados ao projeto tem um formato diferente, com informações diferentes, estes dados devem ser adaptados ao modelo canônico antes de serem utilizados no projeto. O modelo canônico deve estabelecer todas as regras de formatação para campos onde os valores podem variar dependendo do contexto.

Figura 5.1: Representação do modelo de dados canônico.



Fonte: (HOHPE; WOOLF, 2004)

Consequências

Positivas:

- Ao adaptar diferentes fontes de dados a um modelo único, os dados se tornam menos variáveis e com um formato mais estável.
- Os dados se tornam mais facilmente escaláveis, pois cada aplicação nova deve se adaptar ao modelo canônico e não o inverso.
- Dados com diferentes representações podem ser mapeados para apenas uma representação removendo ambiguidade de dados.

Negativas:

- Adiciona mais uma camada de complexidade à aplicação. Os dados devem ser preparados antes de serem utilizados.
- O *overhead* relacionado à tradução de dados é aumentado.

Exemplos

Um exemplo da aplicação desse *pattern* seria agrupar diferentes títulos de cargos equivalentes de uma empresa que está presente em diversas regiões, cada região tendo sua nomenclatura própria de cargos, em uma representação comum. Um "Desenvolvedor de software", um "Programador", um "Desenvolvedor de Aplicativos" e um "Engenheiro de Software Pleno" podem ser agrupados e considerados um único cargo. Estes cargos poderiam ser mapeados a um título comum como "Engenheiro de Software", facilitando a geração de modelos e análises envolvendo dados em formato granular, podendo reduzir a ambiguidade entre dados. A Figura 5.2 exemplifica esse mapeamento de registros para um modelo canônico.

Figura 5.2: Tabela de conversão de cargos para modelo canônico

CARGO	CARGO CANÔNICO
Desenvolvedor de Software	Engenheiro de Software
Programador	Engenheiro de Software
Desenvolvedor de Aplicativos	Engenheiro de Software
Engenheiro de Software Pleno	Engenheiro de Software

Fonte: elaborado pelo autor

Um outro exemplo, descrito na Figura 5.3, é a definição de um formato comum de datas. Um conjunto de dados formado por fontes distintas pode ter suas datas em formatos diferentes, devido a convenções regionais, como o formato brasileiro e o americano. O formato também pode variar pela maneira de escrever os dados, utilizando uma forma mais compacta ou por extenso. O agrupamento desses formatos é essencial para uma análise correta dos dados, possibilitando uma ordenação cronológica.

Figura 5.3: Tabela de conversão de datas para modelo canônico

DATA	FORMATO	DATA CANÔNICA
18/02/2019	DD/MM/YYYY	2019-02-18
03/02/2019	MM/DD/YYYY	2019-03-02
2020-07-06T09:30:01	timestamp	2020-07-06
29 de Dezembro de 2020	DD de MONTH de YYYY	2020-12-29

Fonte: elaborado pelo autor

5.3 Pattern 3: Controle de versionamento

Problema

Ao se trabalhar com DS, há diversas entidades que podem afetar os resultados de uma análise ou processamento, caso elas sejam modificadas. Podemos separar essas entidades em categorias como: dados brutos, rotinas do código que manipula os dados, código que faz a análise dados, bibliotecas externas e parâmetros de ajuste. Uma mudança em qualquer um dos elementos descritos pode mudar o resultado de um ou mais processos de análise de dados (RIDGE, 2014). Caso haja a necessidade de se restaurar um resultado a uma versão anterior, é muito difícil de obter os mesmos resultados se não houver alguma forma de recuperar a estrutura anterior às modificações.

Fatos

- Preservar a origem dos dados é importante para poder analisá-los.
- O resultado de uma versão de uma análise pode ser importante em algum outro momento.
- Os resultados devem ser reproduzíveis em qualquer ambiente.

Solução

Não modificar dados brutos após eles serem importados ao ambiente de manipulação de dados. Quando um código de análise de dados for utilizado para um produto, não modificar o código caso ele seja revisitado. Fazer uma nova versão desse código. Mapear as versões de rotinas utilizadas pelo código de manipulação de dados e fazer o controle de versionamento delas utilizando a ferramenta apropriada. Mapear as versões de bibliotecas e linguagens de programação, se houver a necessidade de atualizações de linguagens e bibliotecas, atualizar mantendo as versões antigas. Anotar parâmetros entrados em algoritmos e testes estatísticos para que eles possam ser repetidos e possam produzir o mesmo resultado. (RIDGE, 2014).

Consequências

Positivas:

- A origem dos dados é preservada, habilitando análises sobre os dados originais.
- Resultados de outras versões são resgatáveis. Caso haja uma necessidade de recuperar resultados de versões antigas, é necessário apenas retomar a versão da saída que se deseja resgatar.
- Resultados são reproduzíveis em qualquer sistema. Valores aleatórios com semen-

tes salvas podem ser reproduzidos. Ao conhecer todas as variáveis de um sistema determinístico a sua saída será única.

Negativas:

- Múltiplas versões das entidades podem causar confusão se não forem claramente nomeadas e explicadas.
- Utilização maior de espaço de armazenamento, por conta da replicação de arquivos.
- Uma organização maior é necessária para separar cada categoria de arquivo, quando comparado com uma estrutura sem controle de versionamento.

Exemplos

A Figura 5.4 descreve uma estrutura de pastas e arquivos de um projeto fictício, onde cada entidade que pode mudar o resultado de uma análise está separada em uma pasta, com suas dependências e parâmetros documentados. É possível observar que os dados brutos não são alterados e que cada conjunto de parâmetros está associado ao arquivo que os utilizam. As versões de dependências são documentadas na pasta "version" e são necessárias para reprodutibilidade dos resultados.

Figura 5.4: Exemplo de estrutura de um projeto preparado para controle de versionamento.

Fonte: elaborado pelo autor.

Na Figura, a pasta "001", que está mais próxima da raiz do projeto, representa um controle de versionamento por meio de pastas numeradas, cada número representando uma versão. Cada nova versão desse projeto teria a replicação de todas as pastas, não importando se apenas um dos elementos fora atualizado. Essa estratégia, apesar de válida, pode ser substituída por controle de versionamento com o uso de ferramentas modernas

para um melhor controle de versão, como Git, podendo separar cada versão do projeto em um *branch* do repositório.

5.4 Pattern 4: Identificador único

Problema

Em operações com dados, há a possibilidade de perda da proveniência dos dados, por conta da natureza das operações ao juntar fontes de dados, manipular formatos de dados ou analisar os dados. Essa perda das fontes torna difícil a identificação da fonte de problemas nos resultados obtidos com os dados (RIDGE, 2014).

Fatos

- Saber a fonte de erros em dados é importante para a qualidade dos resultados.
- Dados manipulados podem ter suas proveniências mais difíceis de serem localizadas.
- O processamento de dados pode requerer múltiplas manipulações dos dados.

Solução

Uma simples solução para conseguir identificar os dados originais é a criação de um identificador único para cada registro, ajudando a manter a ancestralidade dos dados. Uma maneira de criar um identificador único de um registro é a execução de uma função hash no registro completo (RIDGE, 2014). Uma função hash é uma função de uma via com uma string de tamanho arbitrário na entrada e uma string de tamanho fixo na saída (STEVENS, 2007), muito raramente resultando em valores de saída iguais para valores diferentes de entrada.

Consequências

Positivas:

- Fontes de erros se tornam mais fáceis de serem identificadas. Se problemas são encontrados em um determinado passo no fluxo de dados, eles podem ser rastreados à sua fonte sem comparações complexas.
- Se registros forem perdidos, é mais fácil de encontrar quais registros foram perdidos dos dados originais.
- Os testes aplicados aos dados podem ser feitos mais facilmente, pois os dados são rastreáveis no fluxo de dados (RIDGE, 2014).

- Registros duplicados podem encontrados ao aplicar uma função hash. Hashes iguais muito provavelmente significam dados repetidos.

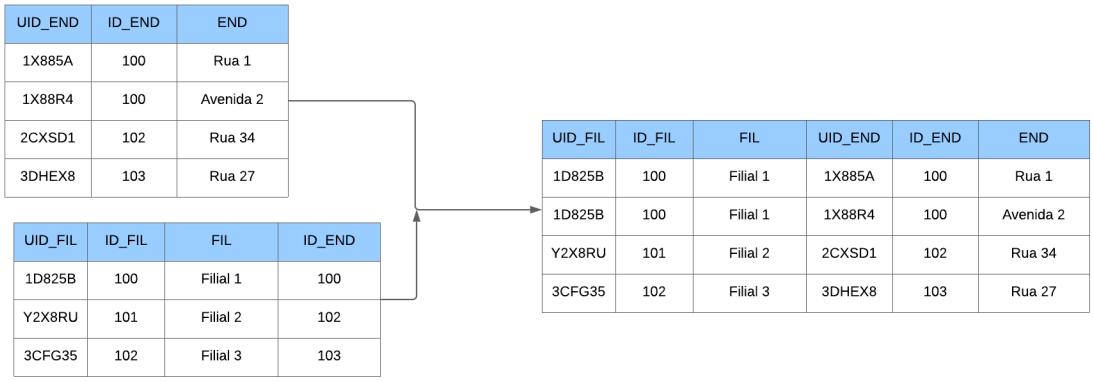
Negativas:

- Aumento na utilização de espaço para os dados. Cada registro terá pelo menos uma coluna a mais com seu identificador único.
- Adiciona um overhead ao processamento dos dados. Um novo conjunto de dados deve ser processado e o identificador único atribuído a cada um dos registros.
- Mudanças nos dados brutos implicam em um novo processamento para gerar identificadores únicos. Como os identificadores podem depender do conteúdo dos registros, há uma necessidade de reprocessamento dos identificadores em caso de mudanças, para manter a consistência dos identificadores.

Exemplos

Um projeto fictício, representado pela Figura 5.5 contém duas tabelas, uma com as filiais de uma empresa e os identificadores, representado por ID, de cada endereço e uma tabela com os endereços. Uma função hash foi aplicada para cada registro da tabela, para gerar cada identificador único, representado por UID. Nesse exemplo, uma filial só pode ter um endereço. Após a junção, é possível identificar uma repetição de identificadores únicos de uma filial, indicando um problema no conjunto de dados. O problema pode ser rastreado a sua fonte, ao utilizar o identificador único do endereço com ID repetido.

Figura 5.5: Junção de endereço à filial.



Fonte: (RIDGE, 2014) - adaptado pelo autor.

Nesse caso, ao corrigir o problema, fazendo as alterações na tabela de endereços, e possivelmente na tabela de filiais, seria necessário manter a consistência dos hashes. Os identificadores únicos dos registros alterados teriam de ser recalculados.

5.5 Pattern 5: Fluxo de dados contínuo

Problema

Quando se trabalha com dados é comum a aplicação de várias linguagens de programação a um produto. Cada tarefa pode ser mais fácil em uma linguagem do que em outra, sendo atrativo a utilização de diferentes linguagens para diferentes tarefas. Essas linguagens de programação geralmente não funcionam em conjunto nos mesmos arquivos de código. Com essas condições, ocorre saltos entre programas nos quais, sem convenções de como esses saltos ocorrem, a execução dos programas se torna excessivamente complexa e difícil de acompanhar (RIDGE, 2014).

Fatos

- Uma linguagem de programação pode ser mais conveniente do que outra para realizar uma determinada tarefa.
- Pulos entre arquivos são necessários para executar diferentes códigos.
- Pulos sem regras podem transformar o fluxo de execução em algo muito complexo.

Solução

Criar um fluxo de dados contínuo dentro dos programas. Um fluxo de dados não deve sair de um determinado arquivo de código para executar em outro processo para depois retornar ao mesmo código e, então, continuar o processo (RIDGE, 2014). Cada arquivo de código deve preferencialmente executar de início a fim, sem saltos para um ou mais programas durante sua execução.

Consequências

Positivas:

- O fluxo dos dados se torna menos complexo de entender. A execução dos arquivos pode ser mapeada mais facilmente, sem a necessidade da interpretação dos códigos em si para se entender o fluxo dos dados.
- O programa de cada arquivo executa de início a fim. Isso evita que se crie códigos com efeitos colaterais não esperados, que podem acarretar em bugs, ou dados quebrados no processo (RIDGE, 2014).
- Cada programa fica com uma responsabilidade única, o que pode reduzir o acoplamento entre trechos de código (SINGH; HASSAN, 2015).
- Um fluxo contínuo de dados resulta na criação de vários arquivos de dados intermediários para a correta execução de uma análise. Esses arquivos de dados inter-

mediários podem ser inspecionados para a realização de testes ou depuração.

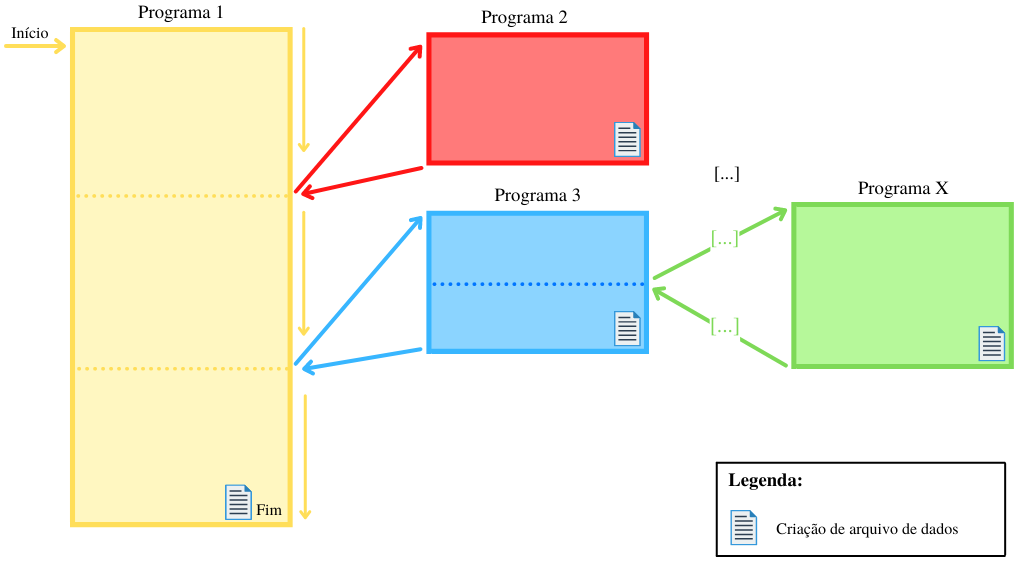
Negativas:

- Aumento no número de arquivos de programas e de dados intermediários. Como cada programa só age em um passo no ciclo de vida dos dados, são necessários múltiplos códigos para uma execução completa de um processo. Caso não seja feita uma organização concisa dos arquivos, o fluxo de dados pode se tornar complexo.
- Programas devem escrever e ler mais arquivos de dados. Cada etapa pode criar um arquivo intermediário e ler um arquivo de uma etapa anterior. Esse processo pode levar a um tempo de execução aumentado para o projeto.

Exemplos

Uma série de programas com fluxo de dados não contínuo tem sua execução representada pela Figura 5.6, onde uma série de saltos entre arquivos pode ocorrer. Dependendo da escala do projeto, cada pulo entre arquivos pode resultar em mais fluxos de ida e vinda entre arquivos, resultando em um sistema que pode ter sua lógica de execução difícil de ser acompanhada. A imagem também apresenta a criação de arquivos, nesse caso sendo feita ao final de cada programa.

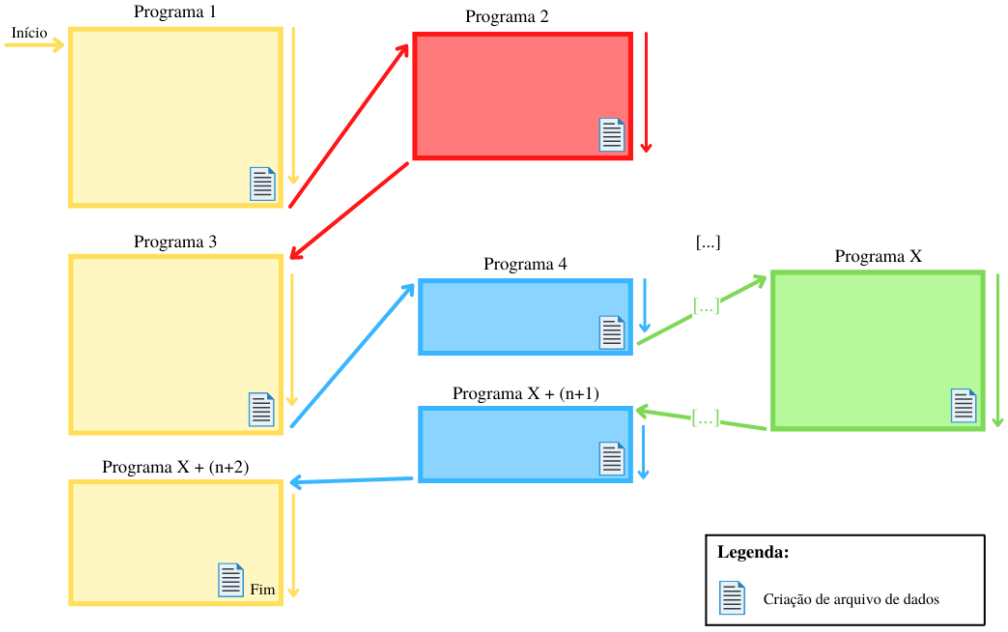
Figura 5.6: Diagrama de execução de fluxo de dados não contínuo.



Fonte: elaborado pelo autor.

A Figura 5.7 apresenta um diagrama de execução utilizando o conceito de fluxo de dados contínuo. Nessa execução, cada programa é executado até o fim, gerando um arquivo intermediário. Os programas que tinham seu fluxo interrompido para executar outros programas foram fragmentados, resultando numa cadeia de programas menores.

Figura 5.7: Diagrama de execução de fluxo de dados contínuo.



Fonte: elaborado pelo autor.

Após a conversão da lógica de execução para uma contínua, o número de programas é aumentado, bem como o número de arquivos intermediários. A nomeação dos arquivos indicando a etapa e a ordem é essencial para que o fluxo execução seja mais facilmente compreendido.

5.6 Pattern 6: Caixa preta de APIs

Problema

Pesquisadores de ML geralmente desenvolvem soluções genéricas como pacotes auto-contidos. O uso de pacotes genéricos muitas vezes resulta em um padrão de design de sistema de *glue code*, no qual uma enorme quantidade de código de suporte é escrita para inserir e retirar dados de pacotes de uso geral. Um código maduro pode ser 95% *glue code* e 5% código de ML. Como um caso especial de *glue code*, *pipeline jungles* muitas vezes aparecem em preparação de dados para sistemas de ML. A manutenção desses *pipelines*, detecção de erros e recuperação é muitas vezes complicada e custosa (SCULLEY et al., 2015).

Fatos

- Cada pacote tem suas entradas e saídas em formatos diferentes.
- A preparação de dados pode ocorrer de forma incremental e organicamente.

- Sem cuidados, o sistema resultante pode ficar muito complexo, com várias etapas intermediárias.

Solução

Uma estratégia importante para combater o *glue code* é envolver os pacotes de caixa preta em Application Programming Interfaces (APIs) comuns. A API teria a responsabilidade de fazer as transformações de dados de entrada e saída para um padrão do código (SCULLEY et al., 2015). Uma interface padrão deve ser criada para a utilização das APIs.

Consequências

Positivas:

- Permite que a infraestrutura de suporte seja mais reutilizável (SCULLEY et al., 2015). É necessário escrever uma única vez código para a utilização de uma biblioteca. O código, após escrito, pode ser importado por outros programas que utilizam os pacotes adaptados pela caixa preta.
- Torna o código menos acoplado a pacotes específicos de terceiros, reduzindo o custo associado a mudanças de pacotes. A interface de entrada e saída de dados é a definida pelo time do projeto, tornando o código agnóstico a bibliotecas específicas.
- Permite que mudanças a pacotes sejam feitas num ponto central. Como o código de suporte fica contido dentro da caixa preta, qualquer mudança necessária aos pacotes utilizados nos projetos podem ser feitas dentro do código da API, evitando a necessidade de adaptação de múltiplos arquivos quando há alterações no projeto.

Negativas:

- Adiciona um overhead à utilização de novos pacotes. Cada biblioteca deve ser adaptada à interface comum do projeto para ser utilizada, aumentando o tempo necessário para o desenvolvimento do projeto.
- Adiciona complexidade em termos de utilização e tradução de dados. A transformação para o formato padrão de uma interface pode ser complexo dependendo do pacote a ser utilizado.

Exemplos

Data frames são estruturas de dados essenciais para a manipulação de dados em memória, sem a necessidade de depender de aplicações de terceiros (PETROV, 2018). Considerando que a estrutura de dados do tipo DataFrame é o formato padrão para a aná-

lise e manipulação de dados num projeto, uma interface de conversão de formato de dados foi proposta, descrita na Figura 5.8. A caixa preta contém métodos de leituras de arquivos de diferentes formatos, tendo como saída a estrutura em memória, conversões de estruturas de dados de bibliotecas de terceiros para o formato comum escolhido e conversões de DataFrames para os formatos que as bibliotecas utilizam para seus métodos.

Figura 5.8: Exemplo de estrutura de classe de conversão de formato de dados.

```
Class modelApi:
// Readers - Will load the dataset into a Pandas DataFrame for further analysis
// source https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html
- DataFrame read_csv(path, encoding)

- DataFrame read_xls(path, encoding)

- DataFrame read_txt(path, encoding)

- DataFrame read_json(path, encoding)

- DataFrame read_sql(path, encoding)

// Converters to DataFrame

- DataFrame from_tensor()

- DataFrame from_torch()

- DataFrame from_numpy()

// Converters from DataFrame

- tf.data       to_tensor(cols)

- torch.tensor  to_torch(cols)

- np.array      to_numpy(cols)
```

Fonte: elaborado pelo autor.

Nesse caso, qualquer conversão para o formato de dados de outras ferramentas devem ser adicionadas à caixa preta de APIs para evitar o *glue code* na parte não estratégica da solução.

5.7 Pattern 7: Remoção de Códigos Experimentais Mortos

Problema

Uma consequência de projetos complexos de Data Science é a tendência de se implementar experimentos como caminhos condicionais no código principal de produção, a fim de agilizar a experimentação de novos métodos e diminuir o custo de recriar a infraestrutura ao redor do código. Ao longo do tempo, esses caminhos acumulados criam um débito técnico, fazendo com que seja cada vez mais difícil manter compatibilidade a

versões antigas e aumentando a complexidade (SCULLEY et al., 2015).

Fatos

- Experimentos no código principal são mais rápidos e mais fáceis de serem feitos.
- O código morto pode se acumular ao longo do tempo.
- Um código obsoleto pode ser executado e comportamentos inesperados podem ocorrer.
- Com o acúmulo de experimentos, arquivos podem ficar num estado onde o fluxo de execução é difícil de entender (RIDGE, 2014).

Solução

Similar ao caso de *dead flags* em engenharia de software, a revisão periódica de caminhos experimentais pode ser feita para analisar quais partes do código são realmente utilizadas e quais não são, podendo ser feita a remoção de códigos experimentais mortos. A quantidade de código abandonado pode ser expressivamente maior do que a de código utilizado (SCULLEY et al., 2015).

Positivas:

- Evita a execução de códigos obsoletos, que podem causar danos aos resultados esperados. Um exemplo famoso de execução errônea de códigos obsoletos foi quando o sistema da Knight Capital perdeu 465 milhões de dólares em 45 minutos, por causa de comportamentos inesperados no código ao executar caminhos experimentais obsoletos (SCULLEY et al., 2015).
- Reduz a complexidade do código. Ao garantir que o código exploratório não está nos arquivos, os programas devem executar sem pular por métodos experimentais, que muitas vezes podem ser velhos e que podem quebrar a execução do código (RIDGE, 2014).
- Aumenta a legibilidade do código. Qualquer fragmento de código que não é relevante ao produto final deve ser removido ou comentado. Isso evita que alguém revisando o código se distraia ao tentar entender blocos irrelevantes do programa (RIDGE, 2014).

Negativas:

- Experimentos com os dados são comuns e utilizados para melhor entender o contexto dos dados. A remoção de código exploratório pode ser uma tarefa longa, dependendo da complexidade do projeto e da quantidade de experimentos presentes

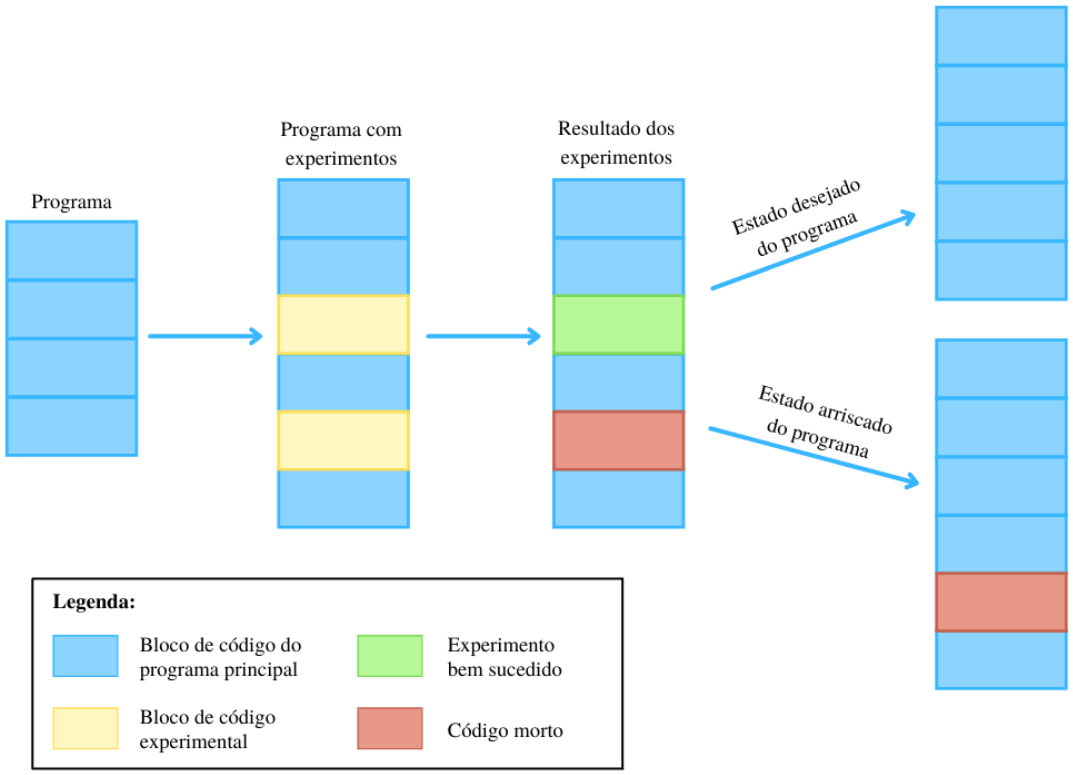
nos arquivos.

- A análise de códigos mortos deve ser feita cuidadosamente para que não seja removido código que esteja sendo utilizado. Dependendo da clareza do código, pode haver fragmentos de códigos que parecem não utilizados, porém servem algum propósito no resultado final do projeto.
- As prioridades de um projeto podem ter natureza volátil, bem como os procedimentos relacionados. Um experimento abandonado pode ser importante em outro momento.

Exemplos

Um processo de experimentação é representado pela Figura 5.9, onde vários experimentos são adicionados a um programa, a fim de facilitar a exploração de novos processos. Os experimentos podem vir a ser beneficiais ao projeto como um todo e integrados ao código do programa ou não serem utilizados no processo, possivelmente sendo executados somente em contextos muito específicos. Com a remoção de códigos mortos experimentais, o estado final desejado do arquivo seria um programa somente com os blocos que foram bem sucedidos em seus experimentos e foram considerados importantes para a execução do código.

Figura 5.9: Processo construção e validação de experimentos em um programa.



Fonte: elaborado pelo autor.

Dependendo da quantidade de experimentos feitos em um programa, o processo de remoção dos blocos experimentais pode se tornar complexo. Por exemplo, um trecho de código que é essencial para o correto funcionamento do programa pode ser interpretado como um experimento, por estar próximo a blocos de código morto; o fluxo do programa pode ser difícil de acompanhar; e o código pode não estar muito claro devido à quantidade de ramificações no programa.

6 AVALIAÇÃO DOS *PATTERNS* POR PROFISSIONAIS DE *DATA SCIENCE*

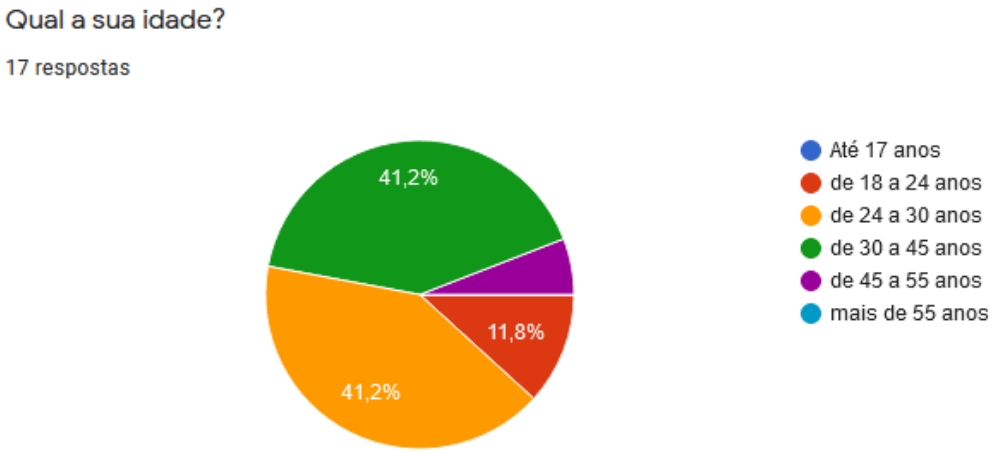
Neste capítulo, o resultado do questionário *online* será analisado com o intuito de avaliar os DP definidos no capítulo anterior, com foco na usabilidade e relevância dos *patterns* em projetos de DS em geral.

No total, 17 pessoas que trabalham na área de *data science* responderam o questionário. O questionário foi divulgado em diversos grupos de discussão sobre informática e *data science* com o objetivo de obter um perfil diverso de respondentes. Todos os respondentes analisaram a descrição de cada *pattern*, avaliaram o entendimento e clareza da definição e responderam perguntas que tentavam entender se os DP apresentados neste trabalho se aplicam a situações reais em projetos de DS.

6.1 Perfil dos respondentes

Em geral, a maioria dos respondentes foram pessoas com idade superior a 24 anos e inferior a 45 anos. Essa distribuição pode ser explicada devido à recente popularização de cargos como *data scientist* no mercado e de recentes avanços em pesquisas sobre *data science* em geral.

Figura 6.1: Idade dos respondentes.



Fonte: formulários Google.

Tabela 6.1: Idade dos respondentes.

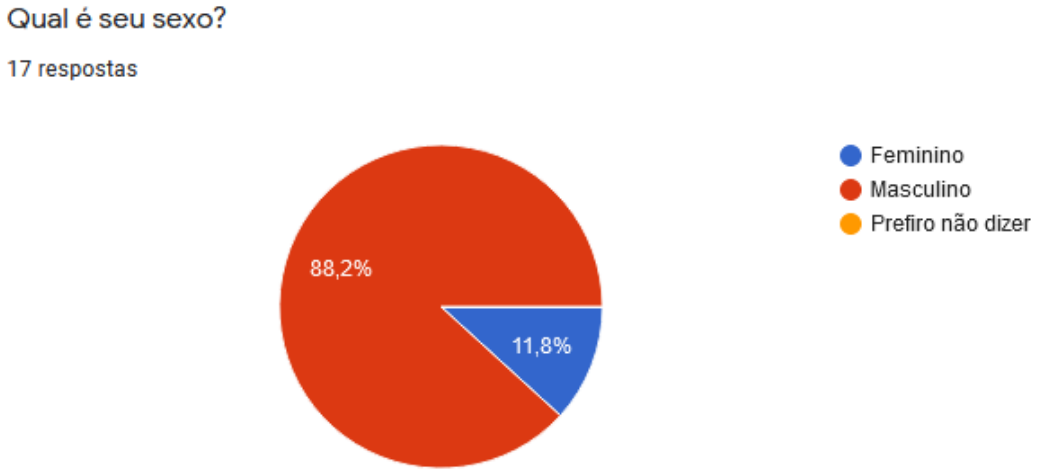
Idade	Frequência	%
Até 17 anos	0	0 %
de 18 a 24 anos	2	11,8 %
de 24 a 30 anos	7	41,2 %
de 30 a 45 anos	7	41,2 %
de 45 a 55 anos	1	5,9 %
mais de 55 anos	0	0 %
Total	17	100 %

Fonte: elaborado pelo autor.

Como é mostrado na Figura 6.1, cerca de 82,4% dos respondentes estão na faixa dos 24 a 45 anos, e 94,2% têm menos de 45 anos. Dos respondentes, 5,9% tinha entre 45 a 55 anos e nenhum respondente tinha mais de 55 anos. Também não houve o caso de um respondente ter 17 anos ou menos, provavelmente devido ao fato do tema em questão ser geralmente aprendido em cursos de ensino superior.

O gênero dos respondentes foi de grande maioria masculino, representando 88,2% dos respondente, enquanto que 11,8% foram respondentes do sexo feminino. Não houve pessoas que preferiram não informar o seu gênero.

Figura 6.2: Gênero dos respondentes.



Fonte: formulários Google.

Tabela 6.2: Gênero dos respondentes.

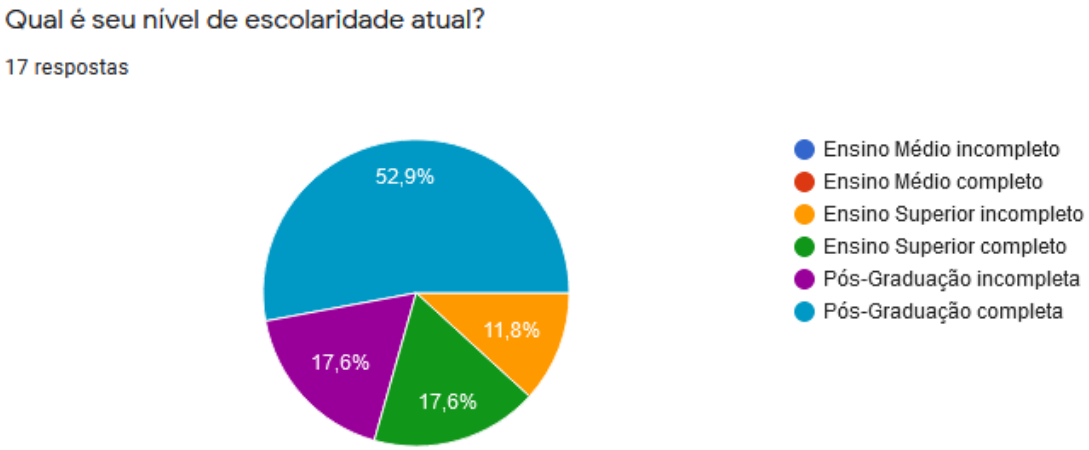
Sexo	Frequência	%
Masculino	15	88,2 %
Feminino	2	11,8 %
Prefiro não dizer	0	0 %
Total	17	100 %

Fonte: elaborado pelo autor.

Essa distribuição de gênero evidencia o número substancialmente menor de mulheres nas áreas relacionadas à computação, onde o número de mulheres concluintes de cursos relacionados à computação segue uma taxa similar à taxa obtida no questionário online. No ano de 2013, a distribuição de pessoas do sexo feminino concluintes de cursos do campo da computação no Brasil foi de 14%, enquanto que a porcentagem de homens concluintes desses cursos foi de 86% (MAIA, 2016).

Cerca de 88,2% das pessoas que responderam o questionário possuíam Ensino Superior completo, sendo 52,9% do total de respondentes pessoas com Pós-Graduação completa. Também é possível observar na tabela 6.3 que todos os respondentes tiveram algum tipo de escolaridade em ensino superior.

Figura 6.3: Nível de escolaridade dos respondentes.



Fonte: formulários Google.

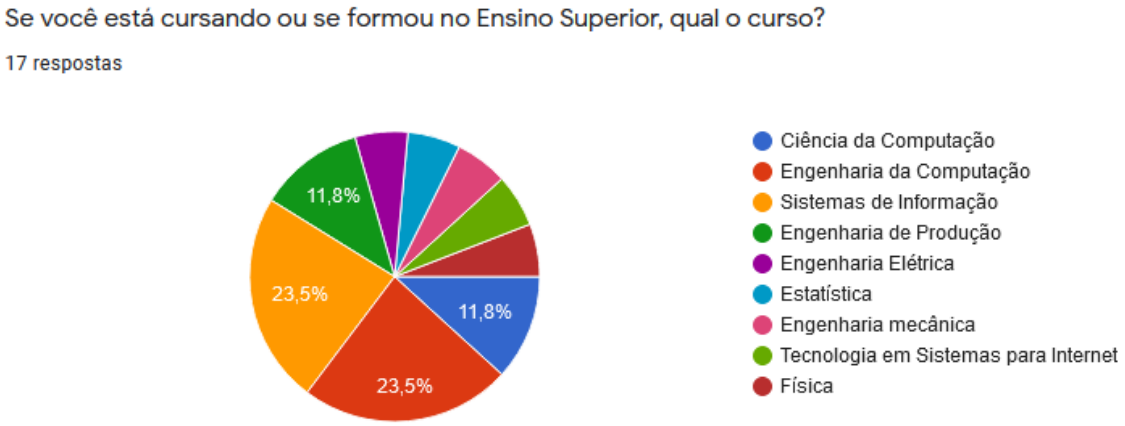
Tabela 6.3: Nível de escolaridade dos respondentes.

Nível de escolaridade	Frequência	%
Ensino Médio incompleto	0	0 %
Ensino Médio completo	0	0 %
Ensino Superior incompleto	2	11,8 %
Ensino Superior completo	3	17,6 %
Pós-Graduação incompleta	3	17,6 %
Pós-Graduação completa	9	52,9 %
Total	17	100 %

Fonte: elaborado pelo autor.

Todos os respondentes se formaram ou estão em cursos das áreas de Ciência, Tecnologia, Engenharia e Matemática. Como DS é uma ciência exata, pessoas de cursos da área de exatas tendem a ser atraídas ao assunto. Os respondentes foram de uma variedade de cursos dessas áreas. Como visto na tabela 6.4, cerca de 64,7% dos respondentes são de áreas diretamente relacionadas à computação (Ciência da Computação, Engenharia da Computação, Sistemas de informação, Tecnologia em Sistemas para Internet).

Figura 6.4: Curso dos respondentes.



Fonte: formulários Google.

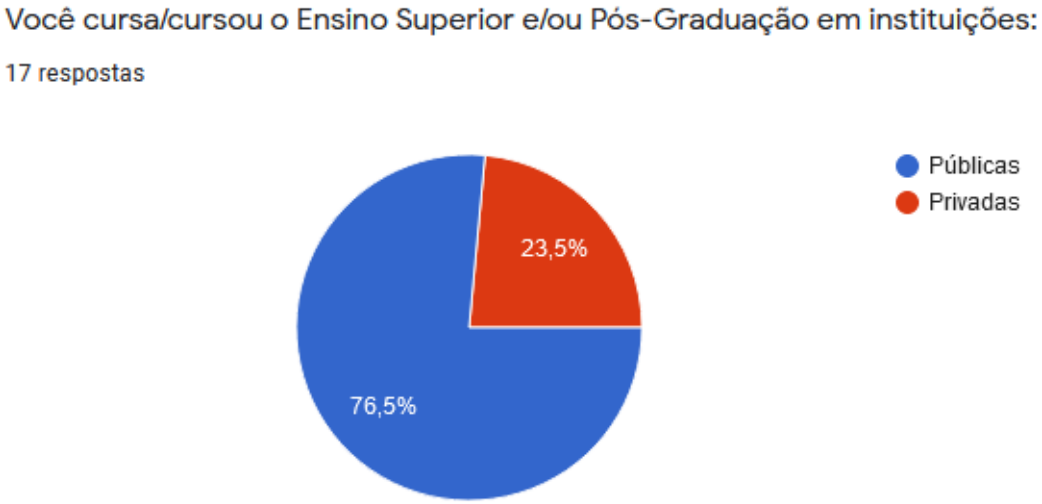
Tabela 6.4: Curso dos respondentes.

Curso	Frequência	%
Engenharia da Computação	4	23,5 %
Sistemas de informação	4	23,5 %
Ciência da Computação	2	11,8 %
Engenharia de Produção	2	11,8 %
Tecnologia em Sistemas para Internet	1	5,9 %
Engenharia Elétrica	1	5,9 %
Estatística	1	5,9 %
Engenharia mecânica	1	5,9 %
Física	1	5,9 %
Total	17	100 %

Fonte: elaborado pelo autor.

A maioria dos respondentes cursa ou realizou sua graduação ou pós-graduação em instituições públicas. Cerca de 76,5% deles afirmaram cursar ou ter cursado em instituições públicas enquanto que 23,5% concluíram seus estudos ou estudam em instituições privadas. Essa proporção de respondentes pode ser reflexo do método utilizado para recrutar os participantes, visto a divulgação maior ter ocorrido em grupos de discussão de uma universidade pública da região sul do brasil.

Figura 6.5: Instituição de ensino dos respondentes.



Fonte: formulários Google.

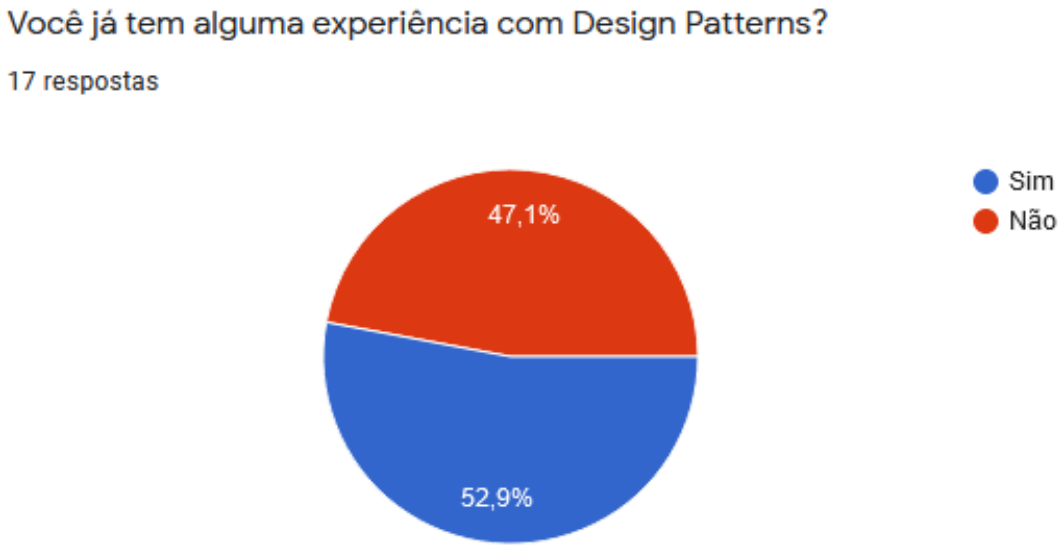
Tabela 6.5: Instituição de ensino dos respondentes.

Instituição	Frequência	%
Públicas	13	76,5 %
Privadas	4	23,5 %
Total	17	100 %

Fonte: elaborado pelo autor.

Um ponto importante que foi considerado era a familiaridade dos respondentes com o conceito de *design patterns*. Descrito na Figura 6.6, quase metade (47,1%) dos respondentes afirmaram não ter experiência com DP, o que pode significar uma falta de incentivo na utilização de boas práticas reutilizáveis no ramo de *data science*. As 9 pessoas que responderam que já tinham alguma experiência com DP adquiriram esta experiência de diversas formas, sem um claro método comum a todos, como ilustrado pela Figura 6.7.

Figura 6.6: Experiência dos respondentes com *patterns*.



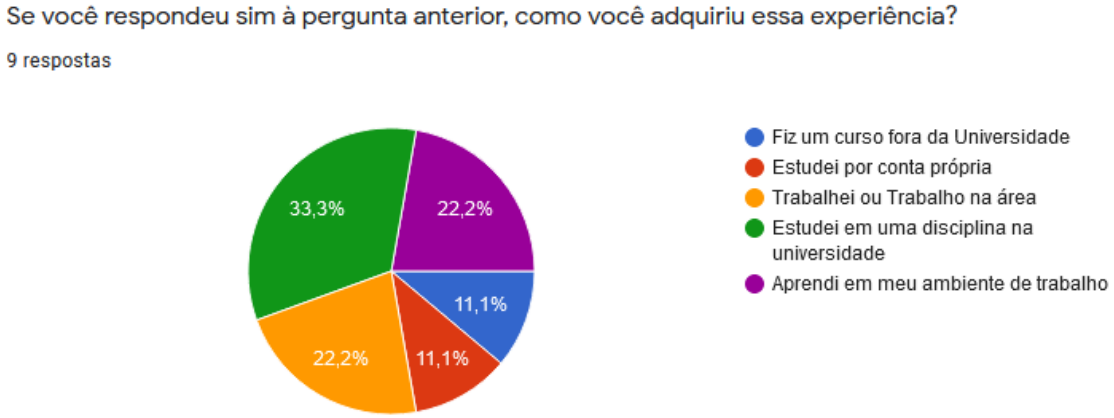
Fonte: formulários Google.

Tabela 6.6: Experiência dos respondentes com *patterns*.

Resposta	Frequência	%
Sim	9	52,9 %
Não	8	47,1 %
Total	17	100 %

Fonte: elaborado pelo autor.

Figura 6.7: Como foi adquirida a experiência dos respondentes.



Fonte: formulários Google.

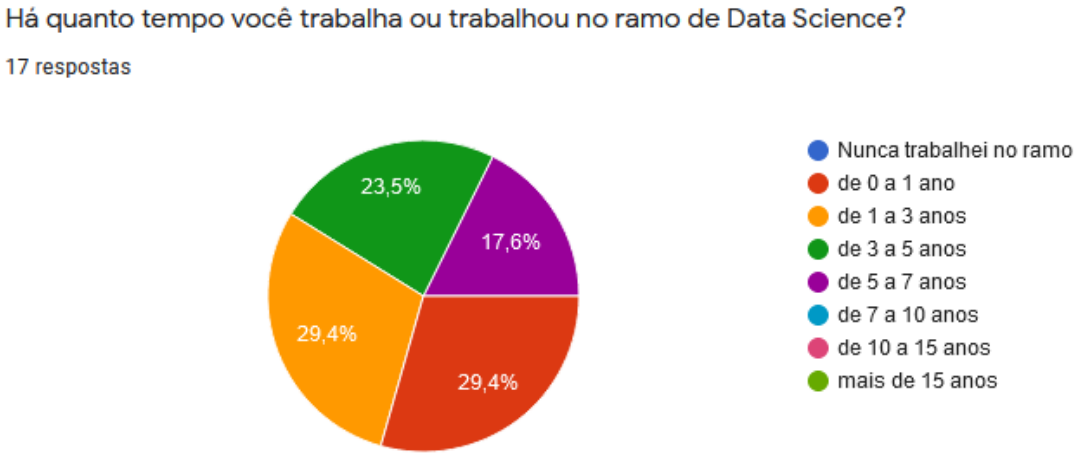
Tabela 6.7: Como foi adquirida a experiência dos respondentes.

Resposta	Frequência	%
Estudei em uma disciplina na universidade	3	33,3 %
Trabalhei ou Trabalho na área	2	22,2 %
Aprendi em meu ambiente de trabalho	2	22,2 %
Estudei por conta própria	1	11,1 %
Fiz um curso fora da Universidade	1	11,1 %
Total	9	100 %

Fonte: elaborado pelo autor.

Para entender a experiência dos respondentes na área de *data science*, foi perguntado há quantos anos eles trabalharam ou trabalham no ramo. Todos os respondentes haviam trabalhado com DS, sendo o perfil desejado do questionário, pois pessoas familiarizadas com o tema poderiam basear suas respostas em situações em projetos reais. Não houve respondentes com mais de 7 anos de experiência na área, possivelmente devido à recente popularização do ramo, com o avanço de novas tecnologias. A Figura 6.8 ilustra a distribuição de tempo de trabalho dos respondentes em DS.

Figura 6.8: Tempo de trabalho dos respondentes no ramo de *data science*.



Fonte: formulários Google.

Tabela 6.8: Tempo de trabalho dos respondentes no ramo de *data science*.

Tempo de trabalho	Frequência	%
Nunca trabalhei no ramo	0	0 %
de 0 a 1 ano	5	29,4 %
de 1 a 3 anos	5	29,4 %
de 3 a 5 anos	4	23,5 %
de 5 a 7 anos	3	17,6 %
de 7 a 10 anos	0	0 %
de 10 a 15 anos	0	0 %
mais de 15 anos	0	0 %
Total	17	100 %

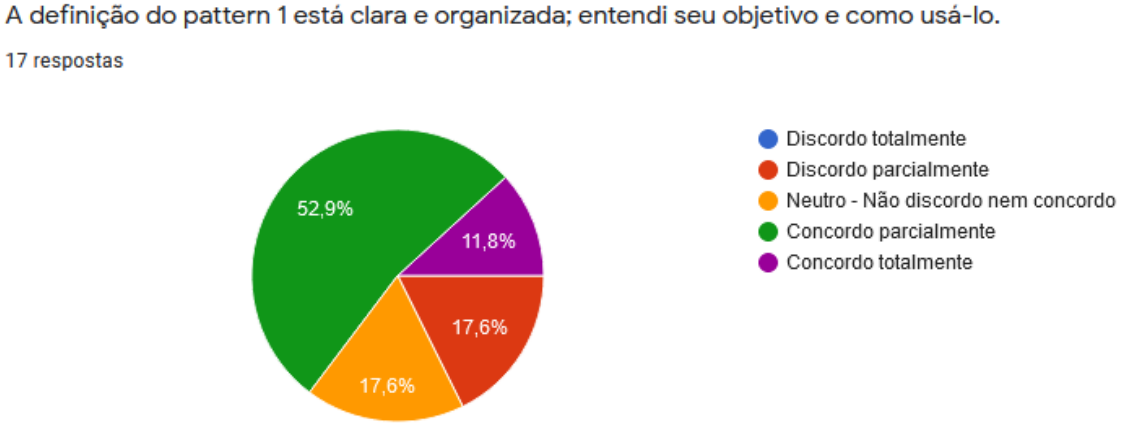
Fonte: elaborado pelo autor.

6.2 Entendimento dos *patterns*

Na seção de entendimento dos *patterns* do questionário, foi apresentada a descrição de cada um dos DP para os respondentes. Antes de responder perguntas sobre a utilidade dos *patterns* que foram definidos, foi considerado importante analisar a clareza e organização das definições deles, para avaliar a qualidade de escrita de cada descrição, bem como indicar a necessidade de eventuais refinamentos futuros em suas descrições.

Com relação ao *Pattern 1: Qualidade de dados*, 52% das pessoas afirmaram concordar parcialmente que a definição está clara e organizada e que entenderam seu objetivo e como usá-lo, 11,8% afirmaram concordar totalmente, 17,6% permaneceram neutros e 17,6% discordaram parcialmente.

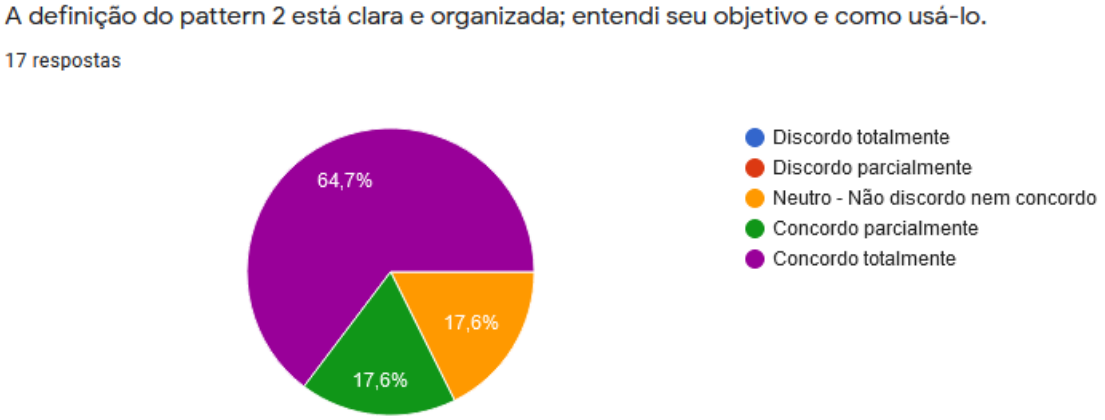
Figura 6.9: Entendimento do *Pattern 1: Qualidade de dados.*



Fonte: formulários Google.

No *Pattern 2: Modelo de dados canônico*, a maioria das pessoas (82,3%) concordou totalmente ou parcialmente que a descrição estava clara e objetiva. Cerca de 17,6% das pessoas permaneceram neutras. Não houve pessoas que discordaram da afirmação.

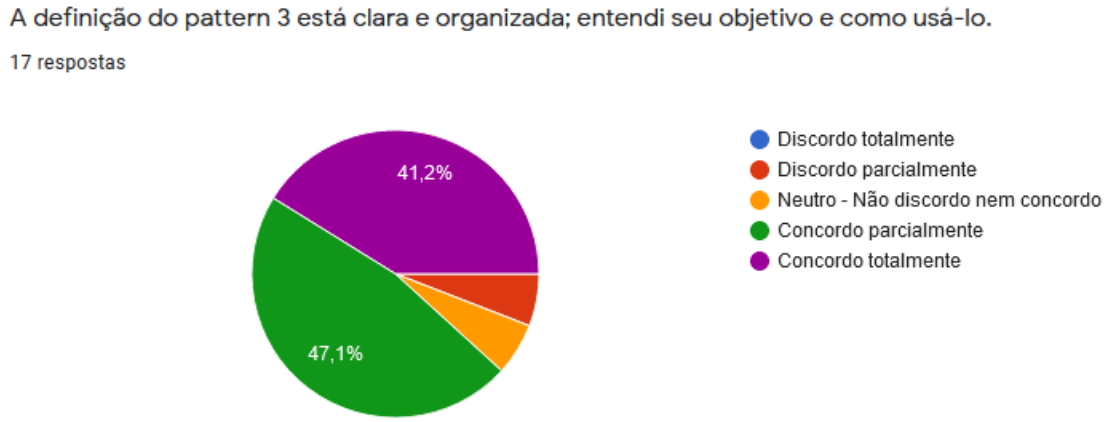
Figura 6.10: Entendimento do *Pattern 2: Modelo de dados canônico*.



Fonte: formulários Google.

No *Pattern 3: Controle de versionamento*, a maioria das pessoas (88,3%) concordou totalmente ou parcialmente que a descrição estava clara e objetiva. Cerca de 5,9% das pessoas permaneceram neutras e 5,9% das pessoas discordaram parcialmente da afirmação.

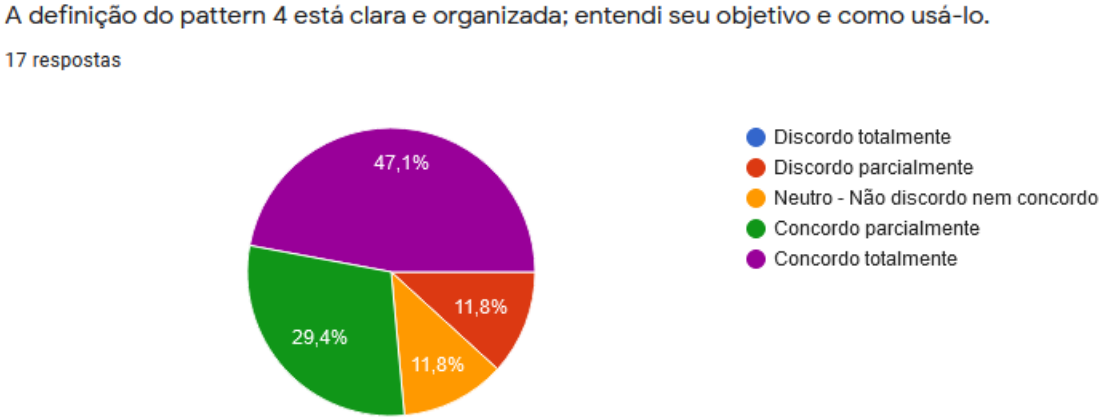
Figura 6.11: Entendimento do *Pattern 3: Controle de versionamento*.



Fonte: formulários Google.

No *Pattern 4: Identificador único*, a maioria das pessoas (76,4%) concordou totalmente ou parcialmente que a descrição estava clara e objetiva. Cerca de 11,8% das pessoas permaneceram neutras e 11,8% das pessoas discordaram parcialmente da afirmação.

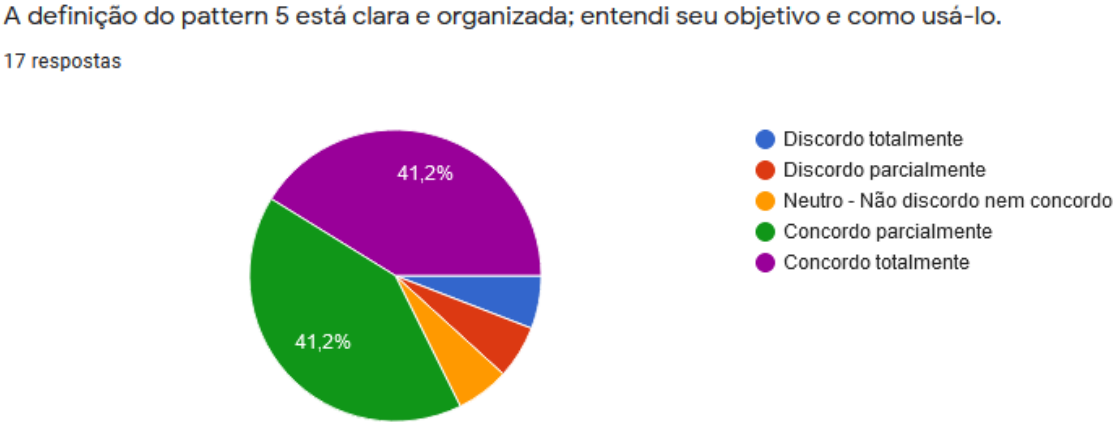
Figura 6.12: Entendimento do *Pattern 4: Identificador único*.



Fonte: formulários Google.

No *Pattern 5: Fluxo de dados contínuo*, a maioria das pessoas (82,4%) concordou totalmente ou parcialmente que a descrição estava clara e objetiva. Cerca de 5,9% das pessoas discordaram parcialmente e 5,9% das pessoas discordaram totalmente da afirmação. A taxa de respostas neutras foi de 5,9%.

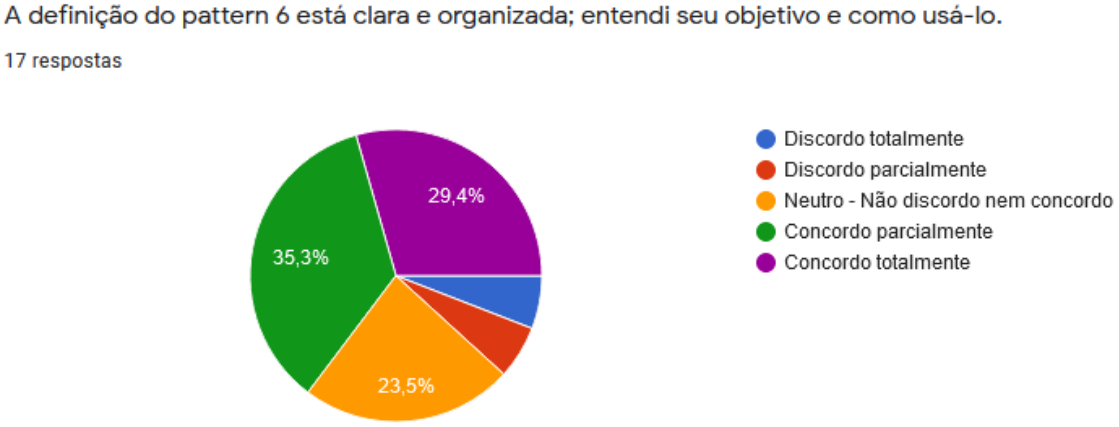
Figura 6.13: Entendimento do *Pattern 5: Fluxo de dados contínuo*.



Fonte: formulários Google.

No *Pattern 6: Caixa preta de APIs*, grande parte das pessoas (64,7%) concordou totalmente ou parcialmente que a descrição estava clara e objetiva. A taxa de pessoas que permaneceram neutras foi de 23,5%, 5,9% das pessoas discordaram parcialmente e 5,9% das pessoas discordaram totalmente da afirmação.

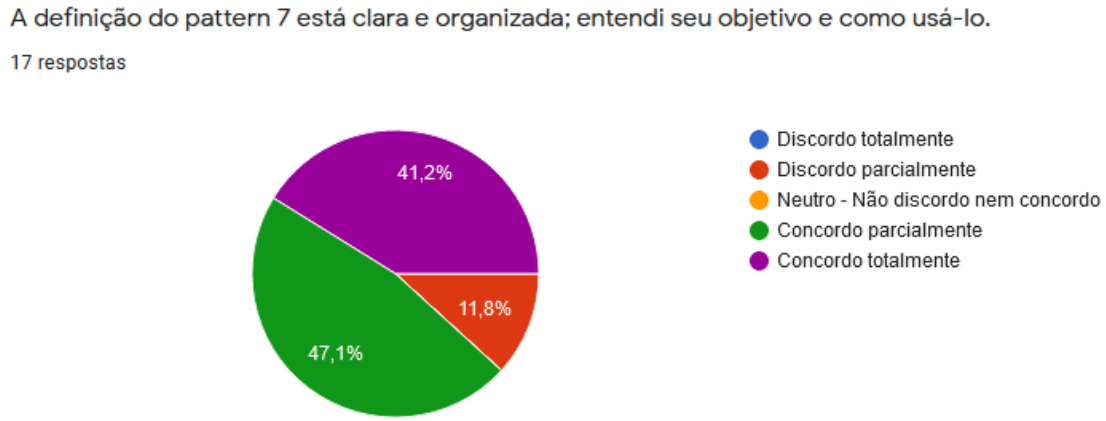
Figura 6.14: Entendimento do *Pattern 6: Caixa preta de APIs*.



Fonte: formulários Google.

No *Pattern 7: Remoção de Códigos Experimentais Mortos*, a maioria das pessoas (88,3%) concordou totalmente ou parcialmente que a descrição estava clara e objetiva. Cerca de 11,8% das pessoas discordaram parcialmente da afirmação.

Figura 6.15: Entendimento do *Pattern 7: Remoção de Códigos Experimentais Mortos*.



Fonte: formulários Google.

Analisando as respostas, o DP que foi mais claramente definido, segundo os respondentes, foi o *Pattern 2: Modelo de dados canônico*. O *Pattern 1: Qualidade de dados* foi o DP com a menor taxa de pessoas concordando totalmente com a afirmação, e maior taxa de pessoas discordando parcialmente com a afirmação. Isso provavelmente pode ser explicado devido à maior abstração de conceitos do *pattern* em relação às outras práticas descritas, bem como à falta de ilustrações exemplificando a técnica abordada.

No geral, pode-se concluir que houve um bom entendimento dos objetivos e clareza dos *patterns*. Não houve uma descrição em que a maioria dos respondentes discordou

da afirmação seja parcialmente ou totalmente, nem houve o caso da maioria dos respondentes se permanecerem neutros à afirmação.

6.3 Validação dos *patterns*

Na etapa de validação, o foco das perguntas era entender o quão útil os *patterns* apresentados pareciam ser em resolver problemas reais, e também entender a opinião geral sobre os DP definidos neste trabalho.

A ideia de repetição de boas práticas é a resolução ou mitigação de problemas que acontecem frequentemente em projetos. Para avaliar se os *patterns* definidos eram relevantes em atividades profissionais de DS, foi considerada a afirmação de que os DP apresentados relacionavam-se a problemas em projetos. Como mostrado na Figura 6.16, a maioria concordou totalmente com a afirmação, boa parte concordou parcialmente, enquanto que uma pequena parte discordou parcialmente ou permaneceu-se neutro.

Figura 6.16: Relevância dos *patterns*.

Fonte: formulários Google.

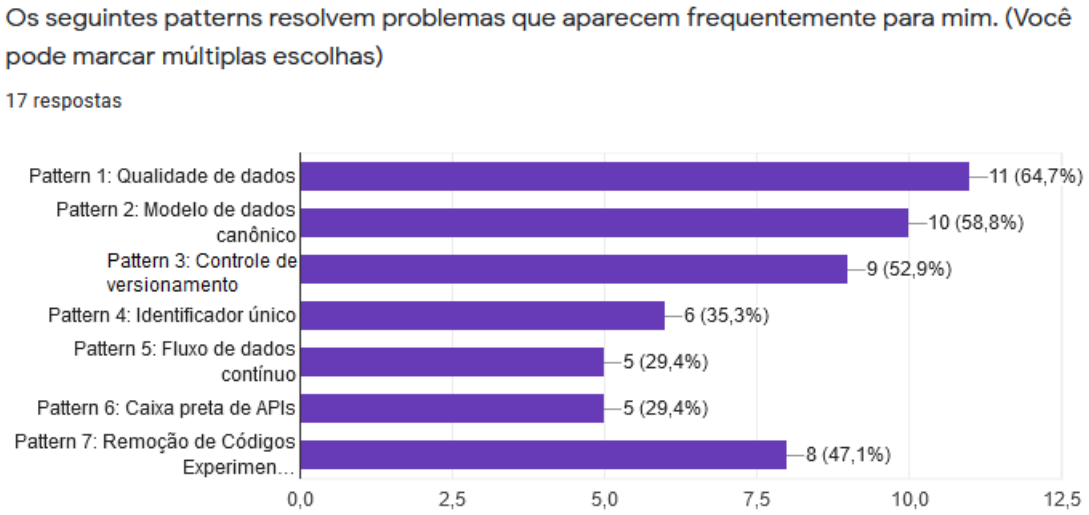
Tabela 6.9: Relevância dos *patterns*.

Resposta	Frequência	%
Discordo totalmente	0	0 %
Discordo parcialmente	1	5,9 %
Neutro - Não discordo nem concordo	1	5,9 %
Concordo parcialmente	4	23,5 %
Concordo totalmente	11	64,7 %
Total	17	100 %

Fonte: elaborado pelo autor.

Esses resultados indicam que os *patterns* apresentados são relevantes em boa parte dos projetos de *data science* em que os respondentes fazem parte. Para entender quais DP eram mais relevantes nos projetos em geral, foi pedido aos respondentes para escolherem quais *patterns* resolvem problemas que aparecem frequentemente para eles.

Figura 6.17: *Patterns* que resolvem problemas frequentes dos respondentes.

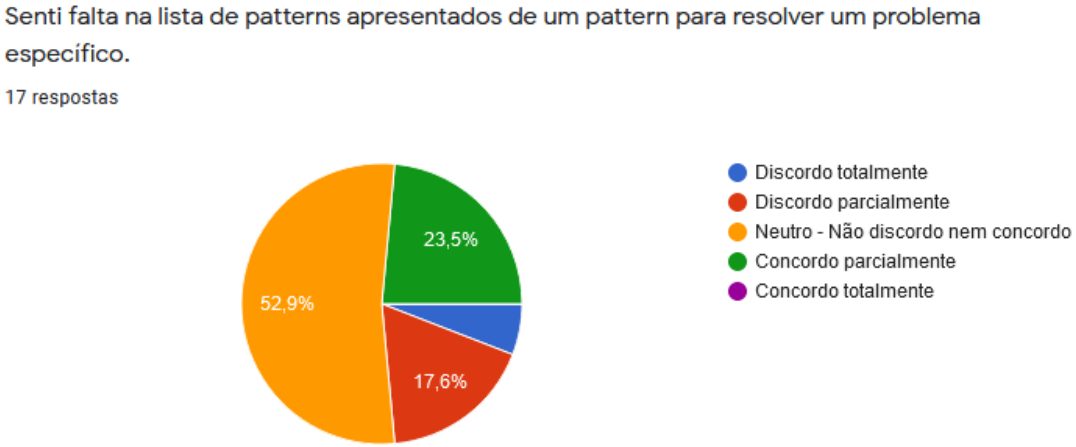


Fonte: formulários Google.

Não houve um *pattern* que não foi escolhido como solucionador de problemas recorrentes, mas houve diferenças entre o número de pessoas que consideravam o *pattern* relevante para a solução de problemas nos seus projetos. O *Pattern 1: Qualidade de dados* foi considerado o *pattern* que resolvia os problemas que aparecem frequentemente de mais pessoas, enquanto que o *Pattern 5: Fluxo de dados contínuo* e o *Pattern 6: Caixa preta de APIs* foram os menos escolhidos.

Para entender se os *patterns* apresentados eram o suficiente para as atividades profissionais dos respondentes, foi considerada a afirmação de que faltou algum *pattern* para resolver um problema específico. Cerca de metade dos respondentes se permaneceram neutros à afirmação e houve uma divisão equilibrada de pessoas que concordaram parcialmente com a afirmação e pessoas que discordaram parcialmente ou totalmente da afirmação, ilustrado pela Figura 6.18.

Figura 6.18: Falta de um *pattern* para resolver um problema específico.



Fonte: formulários Google.

Tabela 6.10: Falta de um *pattern* para resolver um problema específico.

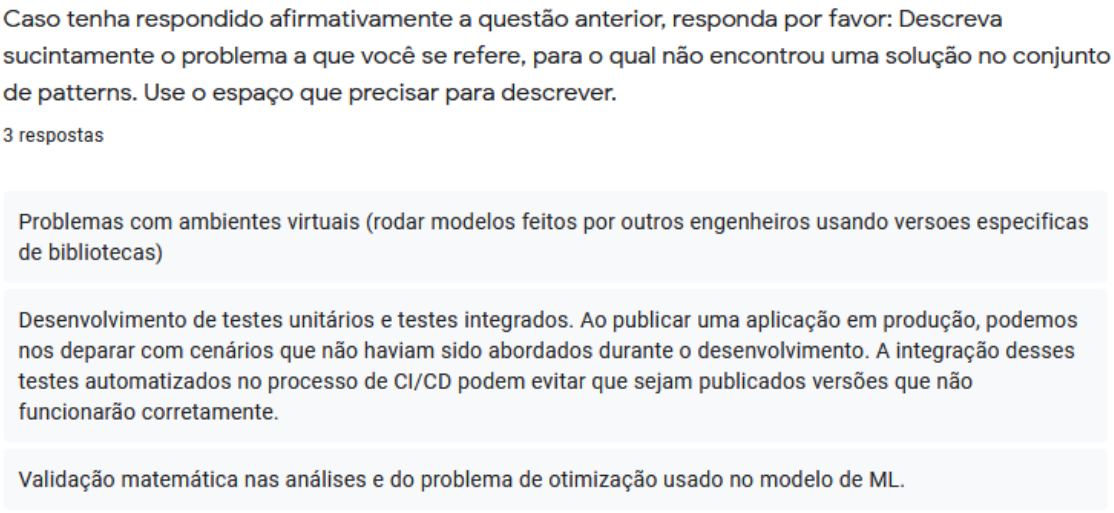
Resposta	Frequência	%
Discordo totalmente	1	5,9 %
Discordo parcialmente	3	17,6 %
Neutro - Não discordo nem concordo	9	5,9 %
Concordo parcialmente	4	23,5 %
Concordo totalmente	0	0 %
Total	17	100 %

Fonte: elaborado pelo autor.

Essa opinião quase que neutra pode indicar que, enquanto que os *patterns* apresentados resolvam problemas enfrentados nas atividades dos respondentes, há um interesse de expansão da lista de *patterns* definidos.

Para os respondentes que responderam afirmativamente a questão, foi pedido para descrever o problema que os *patterns* não resolviam. Dos 4 que responderam que concordavam parcialmente com a afirmação, 3 descreveram um problema e 1 se absteve de responder.

Figura 6.19: Descrição do problema.



Fonte: formulários Google.

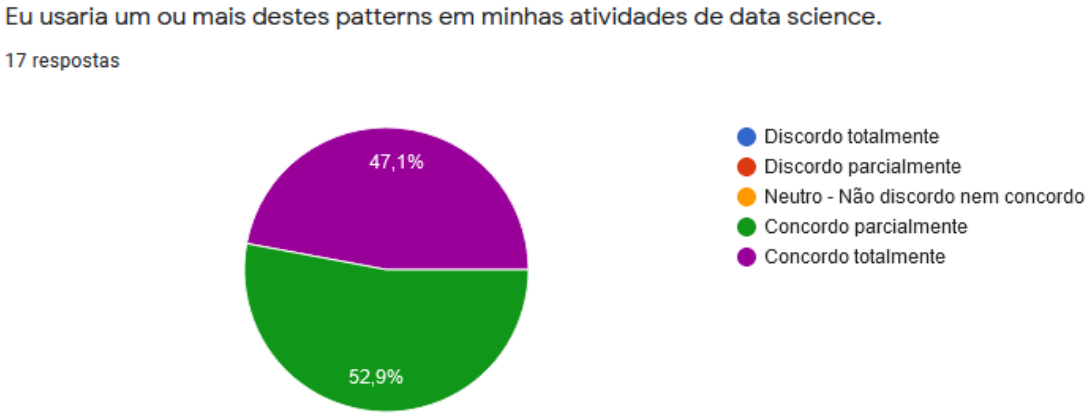
As respostas foram variadas e sem um problema em comum, mas igualmente relevantes. Há um interesse de resolução de problemas relacionados a versionamento em ambientes virtuais, em desenvolvimento de testes unitários e de integração e de validação matemática em modelos de *machine learning*. Como *data science* contém um grande escopo de aplicações, a lista de *patterns* pode ser expandida para a resolução de diversos tipos de problemas em diferentes etapas no desenvolvimento das aplicações.

Para avaliar a utilidade dos *patterns* definidos, foram consideradas três afirmações: O conjunto de *patterns* apresentado é útil para minhas atividades de data science; eu usaria um ou mais destes *patterns* em minhas atividades de data science; eu recomendaria estes *patterns* a um colega. Essas afirmações foram avaliadas com o intuito de entender se, além de relevantes, as descrições trazem valor em termos de resolução de problemas em projetos reais e se são realistas em suas aplicações.

Figura 6.20: Utilidade dos *patterns*.

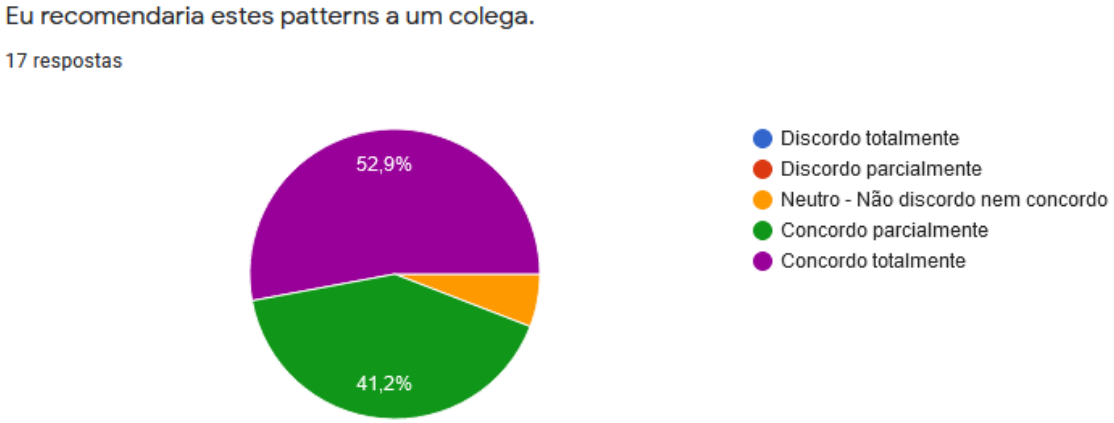
Fonte: formulários Google.

Figura 6.21: Possível uso dos *patterns* definidos.



Fonte: formulários Google.

Figura 6.22: Recomendação dos *patterns* a um colega.

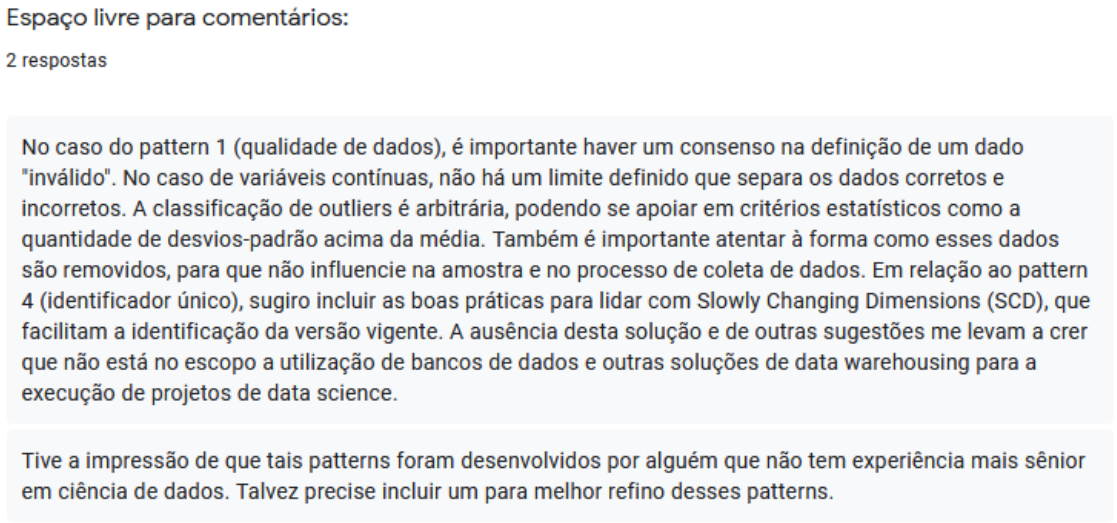


Fonte: formulários Google.

A maioria dos respondentes concordou totalmente ou parcialmente com as afirmações. No caso da primeira afirmação 58,8% das pessoas concordaram totalmente com a afirmação, 35,3% concordaram parcialmente e 5,9% discordaram parcialmente. Na segunda afirmação 47,1% das pessoas concordaram totalmente com a afirmação e 52,9% concordaram parcialmente. Na terceira afirmação 52,9% das pessoas concordaram totalmente com a afirmação, 41,2% concordaram parcialmente e 5,9% permaneceram neutras. Esse resultado pode indicar que os DP definidos foram considerados úteis pelos respondentes e que eles possivelmente poderiam ser utilizados nos projetos, adicionando valor às atividades relacionadas a DS.

Por fim, abriu-se um espaço livre para comentários, para que as opiniões dos respondentes fosse registrada, possivelmente adicionando valor à análise dos resultados. Duas pessoas adicionaram seus comentários ao questionário, como mostra a Figura 6.23.

Figura 6.23: Comentários dos respondentes.



Fonte: formulários Google.

Os dois comentários podem ser interpretados como uma sugestão de aprofundamento dos conceitos dos *patterns*. O primeiro comentário dá sugestões de como as descrições podem ser aprofundadas e ajustadas para gerar mais valor ao leitor e o segundo comentário critica a falta de aprofundamento dos conceitos nas definições dos *patterns*. Os *patterns* definidos neste trabalho podem ser considerados preliminares e têm bastante espaço para ajustes e aprofundamentos de conceitos para que eles se tornem mais robustos e mais aplicáveis nas atividades de DS.

7 CONCLUSÃO

O objetivo deste trabalho foi investigar, descrever, dar exemplos e avaliar *patterns* existentes para projetos de *data science*. Para isso, foi feita uma revisão da literatura relacionada por meio de uma pesquisa de caráter exploratório e um questionário *online*.

A partir da revisão da literatura, sete padrões de projetos emergiram e foram definidos, tendo suas descrições com um formato baseado no formato definido em (PETROV, 2018) e com os elementos de *patterns* descritos em (GAMMA et al., 1995). Esses sete padrões foram, então, avaliados por meio de um questionário *online* por profissionais da área de *data science*, permitindo uma análise preliminar da utilidade e valor dos *patterns* definidos neste trabalho.

Pode-se dizer que o resultado das avaliações dos profissionais sobre os *patterns* definidos foi bom. O nível de entendimento dos objetivos e uso de cada padrão pode ser considerado razoável, com a opinião majoritária de que os *patterns* descritos são relevantes e úteis. Houve criticas sobre o aprofundamento nos conceitos abordados nos *patterns* e sugestões de refinamentos dos padrões, mas como os *patterns* definidos podem ser considerados preliminares, é esperado que haja essa falta de refino e que os conceitos abordados possam ser aprofundados.

O *Pattern 1: Qualidade de dados* foi o que os respondentes tiveram maior dificuldade para entender sua organização e modo de uso. Porém, chama a atenção que este foi escolhido como a descrição que resolve os problemas do maior número de respondentes do questionário. Ainda que a qualidade de dados seja primordial para um bom resultado em atividades de *data science*, há espaço para um melhor detalhamento das operações descritas no processo de escolha e limpeza dos dados, como explicado no primeiro comentário da Figura 6.23.

Outro contraste importante de ser pontuado é que os dois *patterns* que foram mais escolhidos como resolvedores de problemas frequentes (*Pattern 1: Qualidade de dados* e *Pattern 2: Modelo de dados canônico*) são diretamente relacionados a dados e sua organização, enquanto que os dois que foram menos escolhidos (*Pattern 5: Fluxo de dados contínuo* e *Pattern 6: Caixa preta de APIs*) foram relacionados a escrita e organização de arquivos de código. Isso pode ser um indicativo de que a qualidade e organização de dados seja mais importante para a resolução de problemas em *data science* do que a organização e padronização de arquivos de código no momento. Sugere-se pesquisas futuras para testar esta hipótese.

As fontes consultadas para a escrita deste trabalho foram, em sua maior parte livros e literatura acadêmica. Nesses meios, ainda há uma escassez de descrições de *patterns* de projetos em *data science*, o que dificultou o detalhamento de cada *pattern* definido. A literatura cinzenta relacionada a esse tema é muito maior e pode ser explorada para um possível refino das descrições deste trabalho.

Os resultados do questionário indicaram uma boa aceitação dos conceitos dos padrões definidos, mas DP devem ser utilizados para serem realmente testados. Somente após o uso do *pattern* por um período de tempo um benefício como maior organização, menor tempo de desenvolvimento, maior facilidade de desenvolvimento ou maior uniformidade do projeto pode ser avaliado e, assim, solidificar o *pattern* como uma técnica útil para ser reutilizada em outros projetos.

Para trabalhos futuros, indica-se um aprofundamento nos conceitos dos *patterns* descritos neste trabalho, a partir da revisão da literatura cinzenta ou de entrevistas com especialistas na área de *data science*. Também, há espaço para a utilização dos DP definidos a fim de testar a usabilidade e capacidade de resolver problemas de cada *pattern*. Como há muitas etapas num projeto de *data science*, há muitos *patterns* que podem ser descritos para resolver problemas de diferentes etapas dos projetos, como *patterns* sobre as partes de estatística e de armazenamento de dados, por exemplo. Outros DP para DS podem ser definidos e descritos para complementarem os já descritos neste trabalho.

Há um grande espaço para a expansão da lista de *design patterns* para *data science*. Com o recente crescimento e popularização da área de DS, muitos profissionais de diversos campos de estudo têm se interessado em trabalhar com *data science*. Isso faz com que haja uma grande necessidade de definições de boas práticas de projetos, para que tanto iniciantes como pessoas experientes na área possam recorrer a uma série de técnicas comprovadamente eficientes para a resolução de problemas recorrentes em suas atividades profissionais.

REFERÊNCIAS

ALEXANDER, C. **A pattern language: towns, buildings, construction**. [S.l.]: Oxford university press, 1977.

AMERSHI, S. et al. Software engineering for machine learning: A case study. In: IEEE. **2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)**. [S.l.], 2019. p. 291–300.

AMPATZOGLOU, A.; CHARALAMPIDOU, S.; STAMELOS, I. Research state of the art on gof design patterns: A mapping study. **Journal of Systems and Software**, Elsevier, v. 86, n. 7, p. 1945–1964, 2013.

ARENAS, D. et al. Design choices for productive, secure, data-intensive research at scale in the cloud. **arXiv preprint arXiv:1908.08737**, 2019.

BORCHERS, J. O. A pattern approach to interaction design. In: **Proceedings of the DIS2000 International Con-ference on Designing Interactive Systems**. New York, NY: ACM, 2000.

CAO, L. Data science: challenges and directions. **Communications of the ACM**, ACM New York, NY, USA, v. 60, n. 8, p. 59–68, 2017.

CHANG, W. L.; GRADY, N. et al. **NIST Big Data Interoperability Framework: Volume 1, Big Data Definitions**. [S.l.], 2015.

DEARDEN, A.; FINLAY, J. Pattern languages in hci: a critical review. **Human-Computer Interaction**, v. 21, n. 1, p. 49–102, 2006.

DHAR, S.; MAZUMDAR, S. Challenges and best practices for enterprise adoption of big data technologies. In: IEEE. **2014 IEEE International Technology Management Conference**. [S.l.], 2014. p. 1–4.

DONOHO, D. 50 years of data science. **Journal of Computational and Graphical Statistics**, Taylor & Francis, v. 26, n. 4, p. 745–766, 2017.

EUBANKS, CHRISTI. **Three Lessons CrossFit Taught Me About Data Science**. 2016. <https://blogs.gartner.com/christi-eubanks/three-lessons-crossfit-taught-data-science/>. Accessed: 2020-11-03.

GAMMA, E. et al. Elements of reusable object-oriented software. **Reading: Addison-Wesley**, 1995.

GIL, A. C. et al. **Como elaborar projetos de pesquisa**. [S.l.]: Atlas São Paulo, 2002.

GOLLAPUDI, S. **Practical machine learning**. [S.l.]: Packt Publishing Ltd, 2016.

HOHPE, G.; WOOLF, B. **Enterprise integration patterns: Designing, building, and deploying messaging solutions**. [S.l.]: Addison-Wesley Professional, 2004.

JORDAN, M. I.; MITCHELL, T. M. Machine learning: Trends, perspectives, and prospects. **Science**, American Association for the Advancement of Science, v. 349, n. 6245, p. 255–260, 2015.

JOSHI, A. et al. Likert scale: Explored and explained. **Current Journal of Applied Science and Technology**, p. 396–403, 2015.

LAKSHMANAN, VALLIAPPA AND ROBINSON, SARA AND MUNN, MICHAEL. **Machine Learning Design Patterns**. 2020. <https://www.oreilly.com/library/view/machine-learning-design/9781098115777/>. Accessed: 2020-11-04.

LAVRAČ, N. et al. **Introduction: Lessons learned from data mining applications and collaborative problem solving**. [S.l.]: Springer, 2004.

Lestal, Justin. **History of programming languages**. 2020. <https://devskiller.com/history-of-programming-languages/>. Accessed: 2021-03-20.

MAIA, M. M. Limites de gênero e presença feminina nos cursos superiores brasileiros do campo da computação. **cadernos pagu**, SciELO Brasil, n. 46, p. 223–244, 2016.

Morley, Todd. **Data Science Design Patterns**. 2020. <https://www.amazon.com.br/Data-Science-Design-Patterns-Morley/dp/0134000056>. Accessed: 2020-11-04.

PETRASCH, R. Data integration patterns in the context of enterprise data management. In: SPRINGER. **International Conference on Computing and Information Technology**. [S.l.], 2019. p. 235–244.

PETROV, D. **Discovering Data Science Design Patterns with Examples from R and Python Software Ecosystem**. Dissertation (Master) — University of Erlangen, Nuremberg, mar. 2018.

PROVOST, F.; FAWCETT, T. Data science and its relationship to big data and data-driven decision making. **Big data**, Mary Ann Liebert, Inc. 140 Huguenot Street, 3rd Floor New Rochelle, NY 10801 USA, v. 1, n. 1, p. 51–59, 2013.

RIDGE, E. **Guerrilla Analytics: A Practical Approach to Working with Data**. [S.l.]: Morgan Kaufmann, 2014.

SCHMIDT, D. C.; FAYAD, M.; JOHNSON, R. E. Software patterns. **Communications of the ACM**, ACM New York, NY, USA, v. 39, n. 10, p. 37–39, 1996.

SCULLEY, D. et al. Hidden technical debt in machine learning systems. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2015. p. 2503–2511.

SINGH, H.; HASSAN, S. I. Effect of solid design principles on quality of software: An empirical assessment. **International Journal of Scientific & Engineering Research**, v. 6, n. 4, 2015.

STEVENS, M. **On collisions for MD5**. [S.l.]: Citeseer, 2007.

WASHIZAKI, H. et al. Studying software engineering patterns for designing machine learning systems. In: IEEE. **2019 10th International Workshop on Empirical Software Engineering in Practice (IWESEP)**. [S.l.], 2019. p. 49–495.

WASHIZAKI, H. et al. **Machine Learning Architecture and Design Patterns**. 2020.

ANEXO A — FORMULÁRIO PARA VALIDAÇÃO DOS PATTERNS

Design Patterns para Data Science

O objetivo deste questionário é validar e analisar a efetividade de sete design patterns para projetos de data science, que foram definidos baseados numa pesquisa de caráter exploratório da literatura relacionada.

O valor dos patterns vêm da sua capacidade de serem utilizados para resolver problemas. Um pattern pode ser documentado, mas seu valor só é conhecido após ele ter sido testado. Com o intuito de validar a utilidade de cada pattern definido e de verificar sua capacidade de resolver problemas recorrentes, o presente questionário apresentará cada pattern e fará perguntas ao leitor sobre a utilidade dos patterns apresentados.

Os resultados deste questionário serão utilizados em um trabalho de conclusão de curso da Universidade Federal do Rio Grande do Sul, do curso de Engenharia de Computação.

Obrigado pela sua participação.
Aluno de graduação: Victor de Almeida Piccoli Ferreira
Professor orientador: Marcelo Soares Pimenta

Breve descrição de patterns

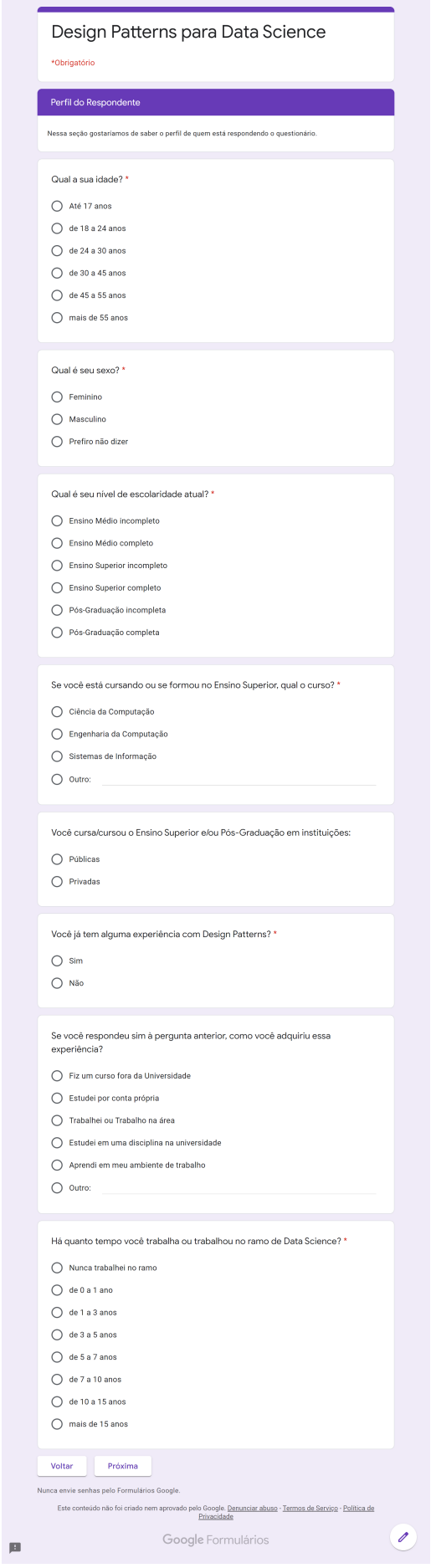
Design patterns têm como princípio a reutilização de boas práticas de arquitetura para produzir designs de qualidade. A ideia geral quando se fala de patterns, não é a de regras que não podem ser quebradas, mas sim de sugestões de técnicas que foram comprovadas como boas práticas para um desenvolvimento saudável de um projeto. Os patterns em si, não são práticas sem um custo envolvido, há sempre uma troca envolvida em termos de qualidade, um foco maior numa característica do projeto pode acarretar em uma perda de qualidade em outra característica do projeto.

Próxima

Nunca envie senhas pelo Formulários Google.

Este conteúdo não foi criado nem aprovado pelo Google. Denunciar abuso - Termos de Serviço - Política de Privacidade

Google Formulários



Design Patterns para Data Science

*Obrigatório

Perfil do Respondente

Nessa seção gostaríamos de saber o perfil de quem está respondendo o questionário.

Qual a sua idade? *

- Até 17 anos
- de 18 a 24 anos
- de 24 a 30 anos
- de 30 a 45 anos
- de 45 a 55 anos
- mais de 55 anos

Qual é seu sexo? *

- Feminino
- Masculino
- Prefiro não dizer

Qual é seu nível de escolaridade atual? *

- Ensino Médio incompleto
- Ensino Médio completo
- Ensino Superior incompleto
- Ensino Superior completo
- Pós-Graduação incompleta
- Pós-Graduação completa

Se você está cursando ou se formou no Ensino Superior, qual o curso? *

- Ciência da Computação
- Engenharia da Computação
- Sistemas de Informação
- Outro:

Você cursa/cursou o Ensino Superior e/ou Pós-Graduação em instituições:

- Públicas
- Privadas

Você já tem alguma experiência com Design Patterns? *

- Sim
- Não

Se você respondeu sim à pergunta anterior, como você adquiriu essa experiência?

- Fiz um curso fora da Universidade
- Estudei por conta própria
- Trabalhei ou Trabalho na área
- Estudei em uma disciplina na universidade
- Aprendi em meu ambiente de trabalho
- Outro:

Há quanto tempo você trabalha ou trabalhou no ramo de Data Science? *

- Nunca trabalhei no ramo
- de 0 a 1 ano
- de 1 a 3 anos
- de 3 a 5 anos
- de 5 a 7 anos
- de 7 a 10 anos
- de 10 a 15 anos
- mais de 15 anos

Voltar Próxima

Nunca envie senhas pelo Formulários Google.

Este conteúdo não foi criado nem aprovado pelo Google. Denunciar abuso - Termos de Serviço - Política de Privacidade

Google Formulários


Design Patterns para Data Science
Entendimento dos patterns
Nas próximas seções, apresentaremos a formalização de cada pattern e buscaremos verificar se os patterns estão claros e objetivos.
Para a formalização, o seguinte modelo foi utilizado:
1. Nome do pattern;
2. Problema descrevendo uma situação que será considerada;
3. Forças sendo examinadas para a solução de um problema e porque é difícil de resolver esse problema;
4. Solução detalhando o caminho escolhido para resolver um problema, levando em conta as forças;
5. Consequências positivas e negativas ao utilizar esta solução;
6. Exemplos ilustrando o seu uso.
Voltar
Próxima
Nunca envie senhas pelo Formulários Google.
Este conteúdo não foi criado nem aprovado pelo Google. Denunciar abuso - Termos de Serviço - Política de Privacidade
Google Formulários

Design Patterns para Data Science

*Obrigatório

Pattern 1: Qualidade de dados

Problema:

Como trabalhos de Data Science e analytics normalmente são exploratórios, não padronizados, não automatizados e personalizados e têm como objetivo de descobrir evidencias para tomadas de decisões, eles sofrem de problemas de qualidade como validade, veracidade, variabilidade e confiabilidade dos dados.

Forças:

- Dados estão constantemente mudando. Após um certo período de tempo algum dado que era válido pode não ser mais relevante.
- Dados são incertos. Cada fonte de dados tem um nível de confiabilidade, deve se assumir um grau de risco associado à veracidade dos dados providos para um projeto.
- A qualidade dos dados determinam a qualidade da saída (Garbage in, garbage out).

Solução:

A validade dos dados e da análise determina se um modelo de dados, conceito, conclusão, ou a medida é bem fundamentada e corresponde exatamente às características dos dados e fatos do o mundo real, tornando-o capaz de fornecer resposta correta. Da mesma forma, a veracidade de dados e análises determinam a correção e precisão de dados e análises de resultados. A escolha de dados que tenham maior validade e veracidade são
determinantes na qualidade final do produto de processamento dos dados.

Consequências:

Positivas:

- Maior validade dos dados. A escolha de dados assume que os dados escolhidos
foram validados de alguma forma e são relevantes para o projeto.
- Menor variabilidade de dados. Ao retirar dados com menor validade ou veracidade,
o conjunto de dados se torna mais enxuto, formado por registros mais relevantes.

Negativas:

- Dados bons não necessariamente geram resultados bons.
- Os resultados obtidos nem sempre são significativos. Por causa das limitações de sistemas de dados existentes, projeções do mundo físico nem sempre podem englobar todo o problema, não sendo possível de se visualizar todo o contexto por qualquer tipo de exploração de dados.

Exemplos:

Ao considerar um projeto fictício que requer uma lista de habilidades técnicas necessárias para um cargo, uma certa experiência em algo que era relevante há alguns anos atrás já não pode mais ser relevante hoje em dia. Caso se utilize um conjunto de dados que contém habilidades técnicas comumente utilizadas para desenvolver um software, antigamente estes dados provavelmente envolveriam conhecimento em linguagens de programação como COBOL, Fortran, ou Pascal. Comparado com os conhecimentos mais utilizados hoje em dia, que são linguagens em como Javascript, Java e C#, pode se perceber que houve uma mudança nos dados que são considerados relevantes e que os resultados de um projeto que requer esses tipos de dados pode gerar resultados não significativos caso a validade dos dados não for considerada.

A distribuição demográfica também é importante, pois os dados podem variar de acordo com a região. Se um programa tivesse como objetivo sugerir uma progressão de cargos, um trabalho que está em alta nos Estados Unidos poderia não ser relevante como próximo emprego na Índia. A veracidade dos dados pode estar atrelada à específicas regiões, sendo importante a análise e escolha de dados significativos para cada situação.

A definição do pattern 1 está clara e organizada; entendi seu objetivo e como usá-lo. *

- ◯ Discordo totalmente
- ◯ Discordo parcialmente
- ◯ Neutro - Não discordo nem concordo
- ◯ Concordo parcialmente
- ◯ Concordo totalmente

Voltar Próxima

Design Patterns para Data Science

*Obrigatório

Pattern 2: Modelo de dados canônico

Problema:

Quando lidamos com a junção de dados de fontes de dados diferentes, os dados geralmente têm formatos diferentes, porque cada formato foi desenhado com só uma aplicação em mente. Assim, pode ocorrer situações em que dados equivalentes têm representações diferentes ou formatos diferentes.

Forças:

- Cada aplicação utiliza o formato de dados que melhor funciona para sua função.
- Formatos de dados podem variar substancialmente dependendo da sua origem e aplicação. Uma junção de dados de fontes distintas pode ser algo difícil de coordenar sem que os resultado da junção fique excessivamente complexo.
- Dados equivalentes podem ter representações diferentes.
- Dados independentes de aplicação têm formato menos variáveis e serem mais estáveis.

Solução:

Modelo de dados canônico, onde os dados são independentes de aplicações. Cada aplicação deve produzir e consumir conteúdo nesse formato. É necessário que cada programa traduza o conteúdo do modelo canônico para consumir os dados. Se cada conjunto de dados relacionados ao projeto tem um formato diferente, com informações diferentes, estes dados devem ser adaptados ao modelo canônico antes de serem utilizados no projeto. O modelo canônico deve estabelecer todas as regras de formatação para campos onde os valores podem variar dependendo do contexto. A figura 1 ilustra o modelo de dados canônico.

Consequências:

Positivas:

- Ao adaptar diferentes fontes de dados a um modelo único, os dados se tornam menos variáveis e com um formato mais estável.
- Os dados se tornam mais facilmente escaláveis, pois cada aplicação nova deve se adaptar ao modelo canônico e não o inverso.
- Dados com diferentes representações podem ser mapeados para apenas uma representação removendo ambiguidade de dados.

Negativas:

- Adiciona mais uma camada de complexidade à aplicação. Os dados devem ser preparados antes de serem utilizados.
- O overhead relacionado à tradução de dados é aumentado.

Exemplos:

Um exemplo da aplicação desse pattern seria agrupar diferentes títulos de cargos equivalentes de uma empresa que está presente em diversas regiões, cada região tendo sua nomenclatura própria de cargos, em uma representação comum. Um "Desenvolvedor de software", um "Programador", um "Desenvolvedor de Aplicativos" e um "Engenheiro de Software Pleno" podem ser agrupados e considerados um único cargo. Estes cargos poderiam ser mapeados à um título comum como "Engenheiro de Software", facilitando a geração de modelos e análises envolvendo dados em formato granular, podendo reduzir a ambiguidade entre dados. A figura 2 exemplifica esse mapeamento de registros para um modelo canônico.

Um outro exemplo, descrito na figura 3, é a definição de um formato comum de datas. Um conjunto de dados formado por fontes distintas pode ter suas datas em formatos diferentes, devido a convenções regionais, como o formato brasileiro e o americano. O formato também pode variar pela maneira de escrever os dados, utilizando uma forma mais compacta ou por extenso. O agrupamento desses formatos é essencial para uma análise correta dos dados, possibilitando uma ordenação cronológica.

Figura 1: Representação do modelo de dados canônico

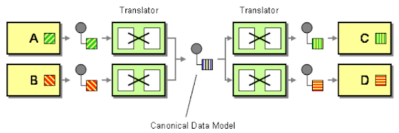


Figura 2: Tabela de conversão de cargos para modelo canônico

CARGO	CARGO CANÔNICO
Desenvolvedor de Software	Engenheiro de Software
Programador	Engenheiro de Software
Desenvolvedor de Aplicativos	Engenheiro de Software
Engenheiro de Software Pleno	Engenheiro de Software

Figura 3: Tabela de conversão de datas para modelo canônico

DATA	FORMATO	DATA CANÔNICA
18/02/2019	DD/MM/YYYY	2019-02-18
03/02/2019	MM/DD/YYYY	2019-03-02
2020-07-06T09:30:01	timestamp	2020-07-06
29 de Dezembro de 2020	DD de MONTH de YYYY	2020-12-29

A definição do pattern 2 está clara e organizada; entendi seu objetivo e como usá-lo. *

- ◯ Discordo totalmente
- ◯ Discordo parcialmente
- ◯ Neutro - Não discordo nem concordo
- ◯ Concordo parcialmente
- ◯ Concordo totalmente

Voltar Próxima

Design Patterns para Data Science

*Obrigatório

Pattern 3: Controle de versionamento

Problema:

Ao se trabalhar com Data Science, há diversas entidades que podem afetar os resultados de uma análise ou processamento, caso elas sejam modificadas. Podemos separar essas entidades em categorias como: dados brutos, rotinas do código que manipula os dados, código que faz a análise dados, bibliotecas externas e parâmetros de ajuste. Uma mudança em qualquer um dos elementos descritos pode mudar o resultado de um ou mais processos de análise de dados. Caso haja a necessidade de se restaurar um resultado à uma versão anterior, é muito difícil de obter os mesmos resultados se não houver alguma forma de recuperar a estrutura anterior às modificações.

Forças:

- Preservar a origem dos dados é importante para poder analisá-los.
- O resultado de uma versão de uma análise pode ser importante em algum outro momento.
- Os resultados devem ser reproduzíveis em qualquer ambiente.

Solução:

Não modificar dados brutos após eles serem importados ao ambiente de manipulação de dados. Quando um código de análise de dados for utilizado para um produto, não modificar o código caso ele seja revisitado. Fazer uma nova versão desse código. Mapear as versões de rotinas utilizadas pelo código de manipulação de dados e fazer o controle de versionamento delas utilizando a ferramenta apropriada. Mapear as versões de bibliotecas e linguagens de programação, se houver a necessidade de atualizações de linguagens e bibliotecas, atualizar mantendo as versões antigas. Anotar parâmetros entrados em algoritmos e testes estatísticos para que eles possam ser repetidos e possam produzir o mesmo resultado.

Consequências:

Positivas:

- A origem dos dados é preservada, habilitando análises sobre os dados originais.
- Resultados de outras versões são resgatáveis. Caso haja uma necessidade de recuperar resultados de versões antigas, é necessário apenas retomar a versão da saída que se deseja resgatar.
- Resultados são reproduzíveis em qualquer sistema. Valores aleatórios com sementes salvas podem ser reproduzidos. Ao conhecer todas as variáveis de um sistema determinístico a sua saída será única.

Negativas:

- Múltiplas versões das entidades podem causar confusão se não forem claramente nomeadas e explicadas.
- Utilização maior de espaço de armazenamento, por conta da replicação de arquivos.
- Uma organização maior é necessária para separar cada categoria de arquivo, quando comparado com uma estrutura sem controle de versionamento.

Exemplos:

A figura 4 descreve uma estrutura de pastas e arquivos de um projeto fictício, onde cada entidade que pode mudar o resultado de uma análise está separada em uma pasta, com suas dependências e parâmetros documentados. É possível observar que os dados brutos não são alterados e que cada conjunto de parâmetros está associado ao arquivo que os utilizam. As versões de dependências são documentadas na pasta "version" e são necessárias para reprodutibilidade dos resultados.

Na figura, a pasta "001", que está mais próxima da raiz do projeto, representa um controle de versionamento por meio de pastas numeradas, cada número representando uma versão. Cada nova versão desse projeto teria a replicação de todas as pastas, não importando se apenas um dos elementos fora atualizado. Essa estratégia, apesar de válida, pode ser substituída por controle de versionamento com o uso de ferramentas modernas para um melhor controle de versão, como Git, podendo separar cada versão do projeto em um branch do repositório.

Figura 4: Exemplo de estrutura de um projeto preparado para controle de versionamento.

```
Project
└ 001
| ├ dataset
| | └ cleaned_products.xls
| ├ data_processing
| | ├ clean_products.py
| | └ clean_product_parameters.json
| ├ model
| | ├ products_model.bin
| | └ products_model_hyper_parameters.json
| ├ raw_data
| | └ products.xls
| ├ training
| | ├ eval.py
| | ├ test.py
| | └ train.py
| └ version
| | ├ dataset_version.json
| | └ requirements.txt
```

A definição do pattern 3 está clara e organizada; entendi seu objetivo e como usá-lo. *

- ○ Discordo totalmente
- ○ Discordo parcialmente
- ○ Neutro - Não discordo nem concordo
- ○ Concordo parcialmente
- ○ Concordo totalmente

Voltar Próxima

Design Patterns para Data Science

*Obrigatório

Pattern 4: Identificador único

Problema:

Em operações com dados, há a possibilidade de perda da proveniência dos dados, por conta da natureza das operações ao juntar fontes de dados, manipular formatos de dados, ou analisar os dados. Essa perda das fontes torna difícil a identificação da fonte de problemas nos resultados obtidos com os dados.

Forças:

- Saber a fonte de erros em dados é importante para a qualidade dos resultados.
- Dados manipulados podem ter suas proveniências mais difíceis de serem localizadas.
- O processamento de dados pode requerer múltiplas manipulações dos dados.

Solução:

Uma simples solução para conseguir identificar os dados originais é a criação de um identificador único para cada registro, ajudando a manter a ancestralidade dos dados. Uma maneira de criar um identificador único de um registro é a execução de uma função hash no registro completo. Uma função hash é uma função de uma via com uma string de tamanho arbitrário na entrada e uma string de tamanho fixo na saída, muito raramente resultando em valores de saída iguais para valores diferentes de entrada.

Consequências:

Positivas:

- Fontes de erros se tornam mais fáceis de serem identificadas. Se problemas são encontrados em um determinado passo no fluxo de dados, eles podem ser rastreados à sua fonte sem comparações complexas.
- Se registros forem perdidos é mais fácil de encontrar quais registros foram perdidos dos dados originais.
- Os testes aplicados aos dados podem ser feitos mais facilmente, pois os dados são rastreáveis no fluxo de dados.
- Registros duplicados podem encontrados ao aplicar uma função hash. Hashes iguais muito provavelmente significam dados repetidos.

Negativas:

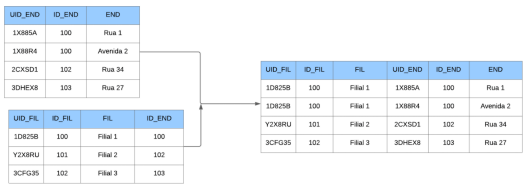
- Aumento na utilização de espaço para os dados. Cada registro terá pelo menos uma coluna a mais com seu identificador único.
- Adiciona um overhead ao processamento dos dados. Um novo conjunto de dados deve ser processado e o identificador único atribuído a cada um dos registros.
- Mudanças nos dados brutos implicam em um novo processamento para gerar identificadores únicos. Como os identificadores podem depender do conteúdo dos registros, há uma necessidade de reprocessamento dos identificadores em caso de mudanças, para manter a consistência dos identificadores.

Exemplos:

Um projeto fictício, representado pela figura 5 contém duas tabelas, uma com as filiais de uma empresa e os identificadores, representado por ID, de cada endereço e uma tabela com os endereços. Uma função hash foi aplicada para cada registro da tabela, para gerar cada identificador único, representado por UID. Nesse exemplo, uma filial só pode ter um endereço. Após a junção, é possível identificar uma repetição de identificadores únicos de uma filial, indicando um problema no conjunto de dados. O problema pode ser rastreado a sua fonte, ao utilizar o identificador único do endereço com ID repetido.

Nesse caso, ao corrigir o problema, fazendo as alterações na tabela de endereços, e possivelmente na tabela de filiais, seria necessário manter a consistência dos hashes. Os identificadores únicos dos registros alterados teriam de ser recalculados.

Figura 5: Junção de endereço à filial.



UID_END	ID_END	END
1X885A	100	Rua 1
1X88R4	100	Avenida 2
2CXSD1	102	Rua 34
3DHEX8	103	Rua 27

UID_FIL	ID_FIL	FIL	ID_END
1D825B	100	Filial 1	100
Y2X8RU	101	Filial 2	102
3CFG35	102	Filial 3	103

UID_FIL	ID_FIL	FIL	UID_END	ID_END	END
1D825B	100	Filial 1	1X885A	100	Rua 1
1D825B	100	Filial 1	1X88R4	100	Avenida 2
Y2X8RU	101	Filial 2	2CXSD1	102	Rua 34
3CFG35	102	Filial 3	3DHEX8	103	Rua 27

A definição do pattern 4 está clara e organizada; entendi seu objetivo e como usá-lo. *

- ◯ Discordo totalmente
- ◯ Discordo parcialmente
- ◯ Neutro - Não discordo nem concordo
- ◯ Concordo parcialmente
- ◯ Concordo totalmente

Voltar Próxima

Design Patterns para Data Science

*Obrigatório

Pattern 5: Fluxo de dados contínuo

Problema:

Quando se trabalha com dados é comum a aplicação de várias linguagens de programação a um produto. Cada tarefa pode ser mais fácil em uma linguagem do que em outra, sendo atrativo a utilização de diferentes linguagens para diferentes tarefas. Essas linguagens de programação geralmente não funcionam em conjunto nos mesmos arquivos de código. Com essas condições, ocorre pulos entre programas nos quais, sem convenções de como esses pulos ocorrem, a execução dos programas se torna excessivamente complexa e difícil de acompanhar.

Forças:

- Uma linguagem de programação pode ser mais conveniente do que outra para realizar uma determinada tarefa.
- Pulos entre arquivos são necessários para executar diferentes códigos.
- Pulos sem regras podem transformar o fluxo de execução em algo muito complexo.

Solução:

Criar um fluxo de dados contínuo dentro dos programas. Um fluxo de dados não deve sair de um determinado arquivo de código para executar em outro processo para depois retornar ao mesmo código e, então, continuar o processo. Cada arquivo de código deve preferencialmente executar de início a fim, sem pulos para um ou mais programas durante sua execução.

Consequências:

Positivas:

- O fluxo dos dados se torna menos complexo de entender. A execução dos arquivos pode ser mapeada mais facilmente, sem a necessidade da interpretação dos códigos em si para se entender o fluxo dos dados.
- O programa de cada arquivo executa de início a fim. Isso evita que se crie códigos com efeitos colaterais não esperados, que podem acarretar em bugs, ou dados quebrados no processo.
- Cada programa fica com uma responsabilidade única, o que pode reduzir o acoplamento entre trechos de código.
- Um fluxo contínuo de dados resulta na criação de vários arquivos de dados intermediários para a correta execução de uma análise. Esses arquivos de dados intermediários podem ser inspecionados para a realização de testes ou depuração.

Negativas:

- Aumento no número de arquivos de programas e de dados intermediários. Como cada programa só age em um passo no ciclo de vida dos dados, são necessários múltiplos códigos para uma execução completa de um processo. Caso não seja feita uma organização concisa dos arquivos, o fluxo de dados pode se tornar complexo.
- Programas devem escrever e ler mais arquivos de dados. Cada etapa pode criar um arquivo intermediário e ler um arquivo de uma etapa anterior. Esse processo pode levar a um tempo de execução aumentado para o projeto.

Exemplos:

Uma série de programas com fluxo de dados não contínuo têm sua execução representada pela figura 6, onde uma série de pulos entre arquivos pode ocorrer. Dependendo da escala do projeto, cada pulo entre arquivos pode resultar em mais fluxos de ida e vinda entre arquivos, resultando em um sistema que pode ter sua lógica de execução difícil de ser acompanhada. A imagem também apresenta a criação de arquivos, nesse caso sendo feita ao final de cada programa.

A figura 7 apresenta um diagrama de execução utilizando o conceito de fluxo de dados contínuo. Nessa execução, cada programa é executado até o fim, gerando um arquivo intermediário. Os programas que tinham seu fluxo interrompido para executar outros programas foram fragmentados, resultando numa cadeia de programas menores.

Após a conversão da lógica de execução para uma contínua o número de programas é aumentado, bem como o número de arquivos intermediários. A nomeação dos arquivos indicando a etapa e a ordem é essencial para que o fluxo execução seja mais facilmente compreendido.

Figura 6: Diagrama de execução de fluxo de dados não contínuo.

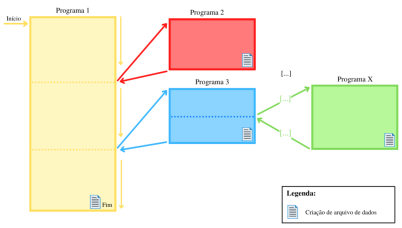
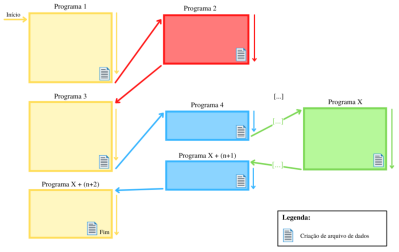


Figura 7: Diagrama de execução de fluxo de dados contínuo.



A definição do pattern 5 está clara e organizada; entendi seu objetivo e como usá-lo. *

- ○ Discordo totalmente
- ○ Discordo parcialmente
- ○ Neutro - Não discordo nem concordo
- ○ Concordo parcialmente
- ○ Concordo totalmente

Voltar Próxima

Design Patterns para Data Science

*Obrigatório

Pattern 6: Caixa preta de APIs

Problema:

Pesquisadores de Machine Learning geralmente desenvolvem soluções genéricas como pacotes auto-contidos. O uso de pacotes genéricos muitas vezes resulta em um padrão de design de sistema de glue code, no qual uma enorme quantidade de código de suporte é escrita para inserir e retirar dados de pacotes de uso geral. Um código maduro pode ser 95% glue code e 5% código de Machine Learning. Como um caso especial de glue code, pipeline jungles muitas vezes aparecem em preparação de dados para sistemas de Machine Learning. A manutenção desses pipelines, detecção de erros e recuperação é muitas vezes complicada e custosa.

Forças:

- Cada pacote têm suas entradas e saídas em formatos diferentes.
- A preparação de dados pode ocorrer de forma incremental e organicamente.
- Sem cuidados, o sistema resultante pode ficar muito complexo, com várias etapas intermediárias.

Solução:

Uma estratégia importante para combater o glue code é envolver os pacotes de caixa preta em Application Programming Interfaces (APIs) comuns. A API teria a responsabilidade de fazer as transformações de dados de entrada e saída para um padrão do código. Uma interface padrão deve ser criada para a utilização das APIs.

Consequências:

Positivas:

- Permite que a infraestrutura de suporte seja mais reutilizável. É necessário escrever uma única vez código para a utilização de uma biblioteca. O código, após escrito, pode ser importado por outros programas que utilizam os pacotes adaptados pela caixa preta.
- Torna o código menos acoplado à pacotes específicos de terceiros, reduzindo o custo associado à mudanças de pacotes. A interface de entrada e saída de dados é a definida pelo time do projeto, tornando o código agnóstico à bibliotecas específicas.
- Permite que mudanças a pacotes sejam feitas num ponto central. Como o código de suporte fica contido dentro da caixa preta, qualquer mudança necessária aos pacotes utilizados nos projetos podem ser feitas dentro do código da API, evitando a necessidade de adaptação de múltiplos arquivos quando há alterações no projeto.

Negativas:

- Adiciona um overhead a utilização de novos pacotes. Cada biblioteca deve ser adaptada à interface comum do projeto para ser utilizada, aumentando o tempo necessário para o desenvolvimento do projeto.
- Adiciona complexidade em termos de utilização e tradução de dados. A transformação para o formato padrão de uma interface pode ser complexo dependendo do pacote a ser utilizado.

Exemplos:

Data frames são estruturas de dados essenciais para a manipulação de dados em memória, sem a necessidade de depender de aplicações de terceiros. Considerando que a estrutura de dados do tipo DataFrame é o formato padrão para a análise e manipulação de dados num projeto, uma interface de conversão de formato de dados foi proposta, descrita na figura 8. A caixa preta contém métodos de leituras de arquivos de diferentes formatos, tendo como saída a estrutura em memória, conversões de estruturas de dados de bibliotecas de terceiros para o formato comum escolhido e conversões de DataFrames para os formatos que as bibliotecas utilizam para seus métodos.

Nesse caso, qualquer conversão para o formato de dados de outras ferramentas devem ser adicionadas à caixa preta de APIs para evitar o glue code na parte não estratégica da solução.

Figura 8: Exemplo de estrutura de classe de conversão de formato de dados.

```
Class modelApi:
// Readers - Will load the dataset into a Pandas DataFrame for further analysis
// source https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html
- DataFrame read_csv(path, encoding)

- DataFrame read_xls(path, encoding)

- DataFrame read_txt(path, encoding)

- DataFrame read_json(path, encoding)

- DataFrame read_sql(path, encoding)

// Converters to DataFrame

- DataFrame from_tensor()

- DataFrame from_torch()

- DataFrame from_numpy()

// Converters from DataFrame

- tf.data       to_tensor(cols)

- torch.tensor  to_torch(cols)

- np.array      to_numpy(cols)
```

A definição do pattern 6 está clara e organizada; entendi seu objetivo e como usá-lo. *

- ◯ Discordo totalmente
- ◯ Discordo parcialmente
- ◯ Neutro - Não discordo nem concordo
- ◯ Concordo parcialmente
- ◯ Concordo totalmente

Voltar Próxima

Design Patterns para Data Science

*Obrigatório

Pattern 7: Remoção de Códigos Experimentais Mortos

Problema:

Uma consequência de projetos complexos de Data Science é a tendência de se implementar experimentos como caminhos condicionais no código principal de produção, a fim de agilizar a experimentação de novos métodos e diminuir o custo de recriar a infraestrutura ao redor do código. Ao longo do tempo, esses caminhos acumulados criam um débito técnico, fazendo com que seja cada vez mais difícil manter compatibilidade à versões antigas e aumentando a complexidade.

Forças:

- Experimentos no código principal são mais rápidos e mais fáceis de serem feitos.
- O código morto pode se acumular ao longo do tempo.
- Um código obsoleto pode ser executado e comportamentos inesperados podem ocorrer.
- Com o acúmulo de experimentos, arquivos podem ficar num estado onde o fluxo de execução é difícil de entender.

Solução:

Similar ao caso de dead flags em engenharia de software, a revisão periódica de caminhos experimentais pode ser feita para analisar quais partes do código são realmente utilizadas e quais não são, podendo ser feita a remoção de códigos experimentais mortos. A quantidade de código abandonado pode ser expressivamente maior do que a de código utilizado.

Consequências:

Positivas:

- Evita a execução de códigos obsoletos, que podem causar danos aos resultados esperados. Um exemplo famoso de execução errônea de códigos obsoletos foi quando o sistema da Knight Capital perdeu 465 milhões de dólares em 45 minutos, por causa de comportamentos inesperados no código ao executar caminhos experimentais obsoletos.
- Reduz a complexidade do código. Ao garantir que o código exploratório não está nos arquivos, os programas devem executar sem pular por métodos experimentais, que muitas vezes podem ser velhos e que podem quebrar a execução do código.
- Aumenta a legibilidade do código. Qualquer fragmento de código que não é relevante ao produto final deve ser removido ou comentado. Isso evita que alguém revisando o código se distraia ao tentar entender blocos irrelevantes do programa.

Negativas:

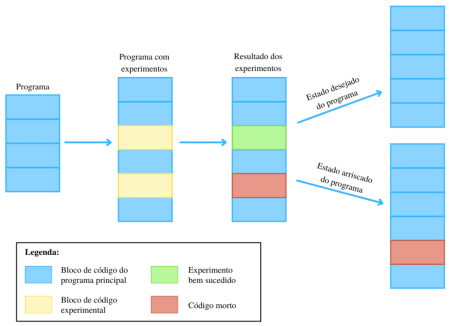
- Experimentos com os dados são comuns e utilizados para melhor entender o contexto dos dados. A remoção de código exploratório pode ser uma tarefa longa, dependendo da complexidade do projeto e da quantidade de experimentos presentes nos arquivos.
- A análise de códigos mortos deve ser feita cuidadosamente para que não seja removido código que esteja sendo utilizado. Dependendo da clareza do código, pode haver fragmentos de códigos que parecem não utilizados, porém servem algum propósito no resultado final do projeto.
- As prioridades de um projeto podem ter natureza volátil, bem como os procedimentos relacionados. Um experimento abandonado pode ser importante em outro momento.

Exemplos:

Um processo de experimentação é representado pela figura 9, onde vários experimentos são adicionados à um programa, a fim de facilitar a exploração de novos processos. Os experimentos podem vir a ser benéficiais ao projeto como um todo e integrados ao código do programa ou não serem utilizados no processo, possivelmente sendo executados somente em contextos muito específicos. Com a remoção de códigos mortos experimentais, o estado final desejado do arquivo seria um programa somente com os blocos que foram bem sucedidos em seus experimentos e foram considerados importantes para a execução do código.

Dependendo da quantidade de experimentos feitos em um programa, o processo de remoção dos blocos experimentais pode se tornar complexo. Por exemplo, um trecho de código que é essencial para o correto funcionamento do programa pode ser interpretado como um experimento, por estar próximo a blocos de código morto; o fluxo do programa pode ser difícil de acompanhar; e o código pode não estar muito claro devido à quantidade de ramificações no programa.

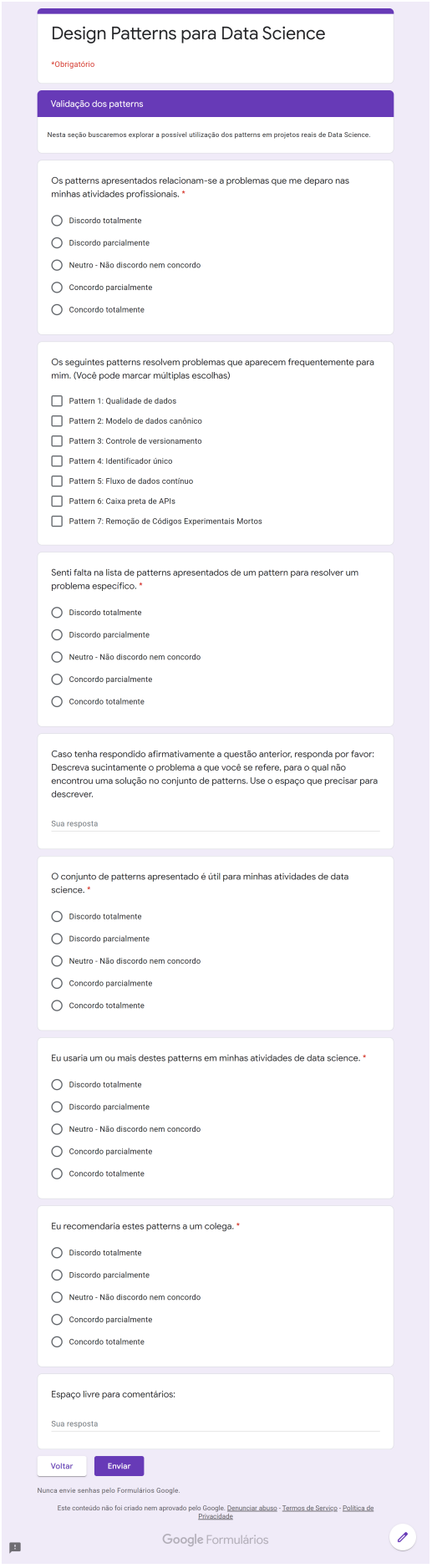
Figura 9: Processo construção e validação de experimentos em um programa.



A definição do pattern 7 está clara e organizada; entendi seu objetivo e como usá-lo. *

- ◯ Discordo totalmente
- ◯ Discordo parcialmente
- ◯ Neutro - Não discordo nem concordo
- ◯ Concordo parcialmente
- ◯ Concordo totalmente

Voltar Próxima



Design Patterns para Data Science

*Obrigatório

Validação dos patterns

Nesta seção buscaremos explorar a possível utilização dos patterns em projetos reais de Data Science.

Os patterns apresentados relacionam-se a problemas que me deparo nas minhas atividades profissionais. *

- ◯ Discordo totalmente
- ◯ Discordo parcialmente
- ◯ Neutro - Não discordo nem concordo
- ◯ Concordo parcialmente
- ◯ Concordo totalmente

Os seguintes patterns resolvem problemas que aparecem frequentemente para mim. (Você pode marcar múltiplas escolhas)

- ☐ Pattern 1: Qualidade de dados
- ☐ Pattern 2: Modelo de dados canônico
- ☐ Pattern 3: Controle de versionamento
- ☐ Pattern 4: Identificador único
- ☐ Pattern 5: Fluxo de dados contínuo
- ☐ Pattern 6: Caixa preta de APIs
- ☐ Pattern 7: Remoção de Códigos Experimentais Mortos

Senti falta na lista de patterns apresentados de um pattern para resolver um problema específico. *

- ◯ Discordo totalmente
- ◯ Discordo parcialmente
- ◯ Neutro - Não discordo nem concordo
- ◯ Concordo parcialmente
- ◯ Concordo totalmente

Caso tenha respondido afirmativamente a questão anterior, responda por favor: Descreva sucintamente o problema a que você se refere, para o qual não encontrou uma solução no conjunto de patterns. Use o espaço que precisar para descrever.

Sua resposta

O conjunto de patterns apresentado é útil para minhas atividades de data science. *

- ◯ Discordo totalmente
- ◯ Discordo parcialmente
- ◯ Neutro - Não discordo nem concordo
- ◯ Concordo parcialmente
- ◯ Concordo totalmente

Eu usaria um ou mais destes patterns em minhas atividades de data science. *

- ◯ Discordo totalmente
- ◯ Discordo parcialmente
- ◯ Neutro - Não discordo nem concordo
- ◯ Concordo parcialmente
- ◯ Concordo totalmente

Eu recomendaria estes patterns a um colega. *

- ◯ Discordo totalmente
- ◯ Discordo parcialmente
- ◯ Neutro - Não discordo nem concordo
- ◯ Concordo parcialmente
- ◯ Concordo totalmente

Espaço livre para comentários:

Sua resposta

Voltar Enviar

Nunca envie senhas pelo Formulários Google.

Este conteúdo não foi criado nem aprovado pelo Google. Denunciar abuso - Termos de Serviço - Política de Privacidade

Google Formulários