Prova 1

Maria Luísa B. C. Silva[1]

[1]Faculdade de Computação – Universidade Federal do Mato Grosso do Sul (UFMS)

maria_luisa@ufms.br

1. Questão 1

As subseções a seguir contém os conceitos sobre **Mineração de Repositórios de Software** (MSR) e ***User Experience*** (UX).

1.1. Mineração de Repositórios de Software (MSR)

Por definição, repositório é tudo aquilo que guarda, abriga algum tipo de informação, portanto repositórios de software abrigam informações relacionadas ao domínio de software. O ato de minerar este tipo de repositório significa recuperar e analisar dados disponíveis para descobrir informações úteis e interessantes sobre sistemas e projetos de software. Uma vez que repositórios de software contêm o passado de um projeto ou sistema além de seu estado atual. As etapas típicas do processo de MSR podem ser definidas como: Aquisição e Preparação de Dados, Síntese, Análise, e por fim Compartilhamento e Replicação.

A aquisição e preparação dos dados diz respeito a identificação do quê será extraído (i.e., códigos, *bugs*, *issues*) e o pré-processamento desses dados para que eles possam ser de fato utilizados para extração de informações. Na etapa de Síntese é necessário decidir qual estratégia será utilizada, ou seja, dados textuais podem ser minerados, os dados podem ser agrupados (*clustering*), e dados temporais podem ser dispostos em séries temporais. A etapa de Análise refere-se a avaliação e interpretação da abordagem decidida no passo anterior, e pode ser considerada a etapa mais crítica e importante para o processo de MSR [Hemmati et al. 2013]. Por fim, a etapa de Compartilhamento e Replicação é responsável por compartilhar as descobertas e reportar os processos com a finalidade de permitir a validação e replicação.

Para mitigar possíveis falhas durante o processo de MSR é necessário definir um planejamento a ser seguido. Durante o planejamento se faz necessário definir questões a serem respondidas com o resultado da mineração, controlar a fonte de onde as informações serão extraídas, definição de métricas para definir o efeito (resultados esperados), estabelecer a população a ser observada no estudo e determinar as abordagens a serem utilizadas na etapa de Síntese.

A escolha de repositório é de extrema importância para os resultados. É necessário investigar a fonte assim como determinar o período de mineração. Para controlar os repositórios a serem minerados durante a etapa de planejamento são definidos critérios de inclusão e exclusão para a seleção. A seleção podem ser dos seguintes tipos:

- **Corrigido**: Repositórios específicos predefinidos manualmente.
- **Filtrado**: Uma amostra a submetida aos critérios de inclusão/exclusão.

- **Aleatório**: Uma amostra original muito grande, que não poderia ser tratada por motivos de limitações (i.e., computacional), tem sua quantidade reduzida e determinada por meio de métodos esteatíticos apropriados e deve passar pelo processo de filtragem.
- **Completo**: Uma filtragem mínima é aplicada e em seguida todos os repositórios são baixados.

1.2. User Experience (UX)

A *User Experience* (UX) compreende na experiência gerada com as interações de um usuário com determinada solução ou produto de uma marca. Ao se tratar de UX é notório que marcas e produtos queiram alcançar uma boa experiência para isso exitem definidos 3 pilares: a desejabilidade, usabilidade e utilidade. Desejabilidade refere-se a vontade do usuário em utilizar o produto e a utilidade diz respeito a necessidade de uso quanto ao produto. Já a usabilidade tem relação a facilidade de uso de uma ferramenta, ou seja, é maneira com a qual o dispositivo ou funcionalidade cumpre o seu objetivo.

Jakob Nielsen, especialista em usabilidade, foi o responsável por definir 10 heurísticas que são até hoje utilizadas no desenvolvimento de software. São elas:

1. Visibilidade de qual estado estamos no sistema.
2. Correspondência entre o sistema e o mundo real.
3. Liberdade de controle fácil ao usuário.
4. Consistência e padrões.
5. Prevenção de erros.
6. Reconhecimento em vez de memorização.
7. Flexibilidade e eficiência de uso.
8. Estética e design minimalista.
9. Ajuda aos usuários a reconhecerem, diagnosticarem e recuperarem-se de erros.
10. Ajuda e documentação.

Como complemento da UX temos a interface do usuário (do inglês, User Interface (UI)) que está relacionado a tudo o que é visível ao usuário de uma solução. A UX e UI são complementares e fazem parte da estratégia de design do produto. UX diz respeito a interação do usuário e UI o layout que o usuário irá utilizar nesta interação.

James Garrett define elementos necessário a UX, sendo estes: estratégia (o que é esperado pelo usuário), escopo (transforma estratégia em requisitos), estrutura (como conectar os requisitos), esqueleto (cria a estrutura), superfície (conexão de tudo o que foi levantado).

1.3. UX e MSR no apoio a estimativas de software

No artigo "Moving on from the software engineers' gambit: an approach to support the defense of software effort estimates", Matsubara et al. apresenta uma abordagem de princípios de negociação no contexto de estimativas de software encapsulados em lentes de defesa. O objetivo desse trabalho é investigar a utilidade das lentes de defesa dentro de equipes de software nas quais estimativas para desenvolvimento de funcionalidades são realizadas recorrentemente. Essas estimativas muitas vezes são distorcidas por diversos motivos, o que gera pressão para entrega podendo levar a um produto de má qualidade e estresses emocionais à equipe [Matsubara et al. 2023]. Matsubara et al. entrega uma

solução baseada nos princípios da negociação, mas é possível também trazer o contexto de MSR e UX para apoiar essa problemática.

A mineração dos repositórios internos de uma equipe (repositórios que são mantidos por essa mesma equipe) pode contribuir na melhora das estimativas com o apoio de evidências reais, além de ser argumento para apoiar a decisão do time em determinada estimativa. Por exemplo, cada funcionalidade dentro da equipe pode ser pontuada com uma simples métrica de complexidade (baixa, média ou alta). A partir disso, através de séries temporais, é possível coletar informações de quanto tempo em média foi necessário para desenvolver funcionalidades (completas) de cada categoria de complexidade, e usar essa métrica como *baseline* ao decidir futuras estimativas para funcionalidades de mesma categoria. Esse processo poderia ser repetido (replicado) periodicamente a fim de atualizar as métricas da equipe. Essa mesma série temporal pode ser aplicada como argumentação para defender as estimativas diante de pressão ou solicitação de mudança por parte de clientes ou gerentes.

A proposta de lentes de defesa dos autores pode ser de complexo entendimento. Para isso utilizando os princípios de UX, uma interface de usuário poderia ser criada a fim de expor e explicar os conceitos de lentes de defesa, adicionado de exemplos que remetem a realidade junto das lentes que poderiam beneficiar a defesa da estimativa do exemplo. Ao prezar por uma boa usabilidade, utilizando por exemplo as heurísticas de Nielsen coerentes a esse contexto, esse produto contribuiria na disseminação do conhecimento dos princípios de negociação desenvolvidos por Matsubara et al., e possivelmente colaborar positivamente no cenário de estimativas de software.

1.4. Mapa mental

A imagem do mapa mental entre os conceitos de Engenharia de Software, MSR e UX pode ser visualizado na Figura 1 e no link: Mapa Mental.

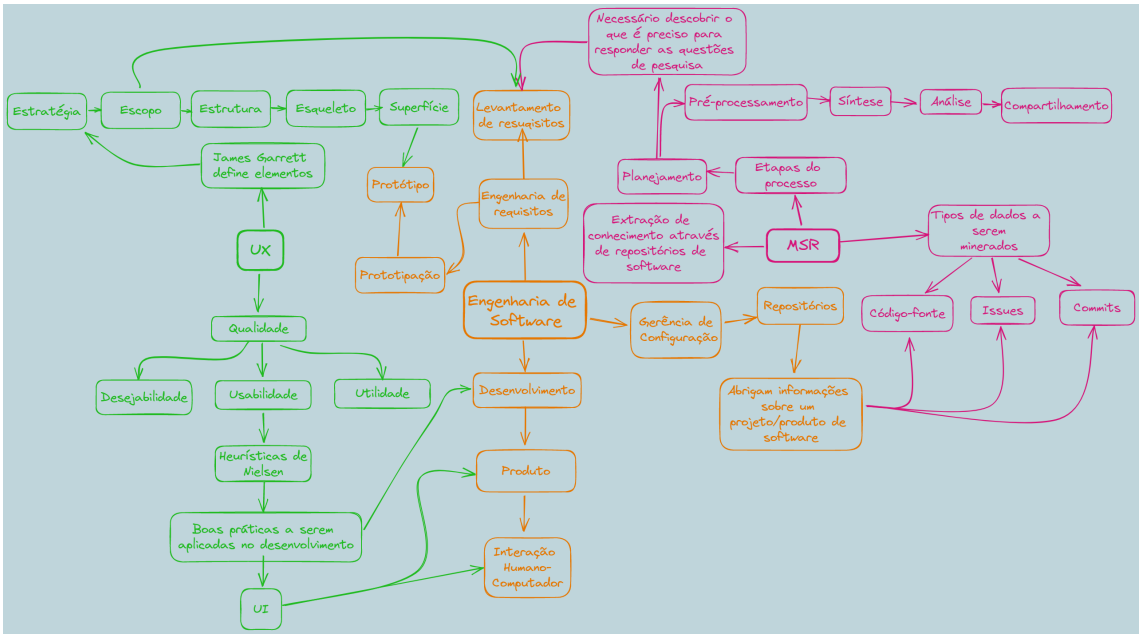

Figura 1. Mapa mental entre os conceitos de Engenharia de Software, MSR e UX

2. Questão 2

Sistemas de aprendizado de máquina (AM) são sistema de software que possuem componentes de Inteligência Artifical (IA) [Khomh et al. 2018]. Dado o crescimento do AM no domínio de software, o simpósio SEMLA (do inglês, *Software Engineering for Machine Learning Applications*) foi criado com a finalidade de discutir soluções de Engenharia de Software (ES) para sistemas de AM. Na edição de 2018 do SEMLA, Khomh et al. identificou duas atividades de ES que precisam ser aplicadas no contexto de aprendizado de máquina, estas são: testes e verificação de acurácia dos sistemas.

O levantamento de requisitos em sistemas de aprendizado máquina possuem suas próprias particularidades e desafios, como por exemplo, os requisitos são vistos como hipóteses que devem ser testadas via experimentos [Wan et al. 2019]. Requisitos de sistemas AM geralmente são especificados por métricas quantitativas que por vezes são desconhecidas pelas partes interessadas, caso a métrica não esteja clara para os objetivos de negócio, isso pode influenciar o cliente a acreditar que o sistema não entrega o que foi pedido [Giray 2021]. Essas particularidades demonstram a necessidade de que as técnicas de levantamento de requisitos (processo indispensável na ES) sejam adaptadas aos contextos de sistemas de AM.

Em um estudo de caso dentro da Microsoft, Amershi et al. descreveu as boas práticas da organização para o desenvolvimento de aplicações de aprendizado de máquina. Como resultado, os autores apontam a existência e necessidade de processos de ES presentes no *workflow* de desenvolvimento de AM, entretanto foram necessárias adaptações desses processos para esse contexto. As três principais observações quanto a essas adaptações são: gerenciamento e descoberta de dados, customização e reuso, e modularidade [Amershi et al. 2019].

3. Questão 3

Scrum é uma metodologia ágil amplamente usada no desenvolvimento de software. Entretanto, podemos combinar técnicas ao Scrum de forma a extrair o máximo proveito da estratégia. A seguir será definida uma abordagem da combinação de conceitos do Shape Up (processo de desenvolvimento da Basecamp) ao Scrum, considerando *squads* de produto e de apoio a produtos.

O processo do Scrum tradicionalmente (figura 2) possui *Sprints* iterativas que visam cumprir os objetivos levantados durante a *Sprint Planning*. Uma adição proposta às *squads* de produto, é a atividade de "Cool Down" definida no Shape Up, dessa forma a equipe tem previsto um tempo que será empregado para correção de bugs, refatoração e estudos (pagamento de dívida técnica), sem a necessidade de encaixar essas tarefas durante o ciclo de desenvolvimento de uma nova funcionalidade. Pode-se também variar a periodicidade de *Sprints Cool Down* (SCD), ou seja, dependendo da necessidade da equipe, as SCDs podem ocorrer em alternância com a *Sprint* comuns, ou a cada duas *Sprints* comuns, por exemplo. Para impedir que apenas SCDs sejam realizadas, para esta proposta fica como regra o impedimento da realização SCDs consecutivas, ou seja, qualquer planejamento em que uma SCD preceda outra deve ser vetado. Importante pontuar que as *Sprints Cool Down* possuem o mesmo *timebox* que uma *Sprint* comum, respeitando então a definição tradicional do Scrum.

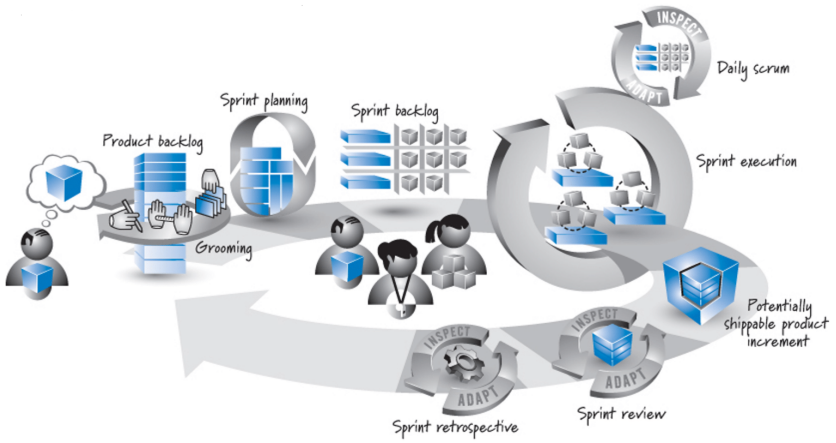

Figura 2. Ilustração do processo tradicional do Scrum.

Outro conceito do Shape Up que pode beneficiar o Scrum (tanto *squads* de produto como as de apoio ao produto) é o tamanho dos times. A opção por times menores facilita a sua autonomia. Como regra do Scrum, um profissional de cada área necessário para implementação deve compor a equipe. Portanto, por muitas vezes o número do Shape Up (dois desenvolvedores e um designer) pode não ser satisfatório as regras do Scrum, porém reduzir ao máximo a quantidade de pessoas dentro do time e criar times focados em objetivos (diferentes *squads*) pode ser uma vantagem no desenvolvimento do produto.

Para o *Product Backlog*, o Scrum define o processo de *Grooming* responsável pela criação, refinamento, estimativas e priorização de estórias. Um conceito do Shape Up que pode ser adicionado a essa etapa do Scrum é a atividade de *Pitch*. Além das tradicionais tarefas relacionadas as estórias, a estratégia do *Pitch* pode ser adicionada a fim de enriquecer os requisitos. Ao contrário do Shape Up, todo o time seria encarregado dessa atividade. O *Pitch* pode ser visto nesse caso como um acréscimo ao Scrum no sentido de auxiliar o entendimento do produto, no qual os integrantes da equipe, podem já em um primeiro momento, adicionar, por exemplo, esboços da solução. Ao chegar o momento de desenvolver dada funcionalidade, que foi contemplada pelo *Pitch*, os desenvolvedores já teria uma melhor noção de como iniciar o desenvolvimento, possivelmente aumentando a produtividade do time.

Um esboço do processo proposto é mostrado na figura 3.

4. Questão 4

A literatura cinza (da sigla em inglês, GL) é de grande utilidade como fonte de conhecimento, nesta seção serão apresentados argumentos a fim de apresentar os benefícios da GL com base no estudo “Benefitting from the Grey Literature in Software Engineering” [Garousi et al. 2020]. No dia-a-dia do desenvolvimento de software é comum, enquanto profissionais da área, depararmos com conceitos desconhecidos. Para isso, o mais comum é buscar conhecimento em fontes que entregam a informação de maneira pragmática e que seja de simples acesso, normalmente fontes de literatura cinza. A GL é um formato de literatura que não passou pelo processe formal de revisão por pares, e essa característica

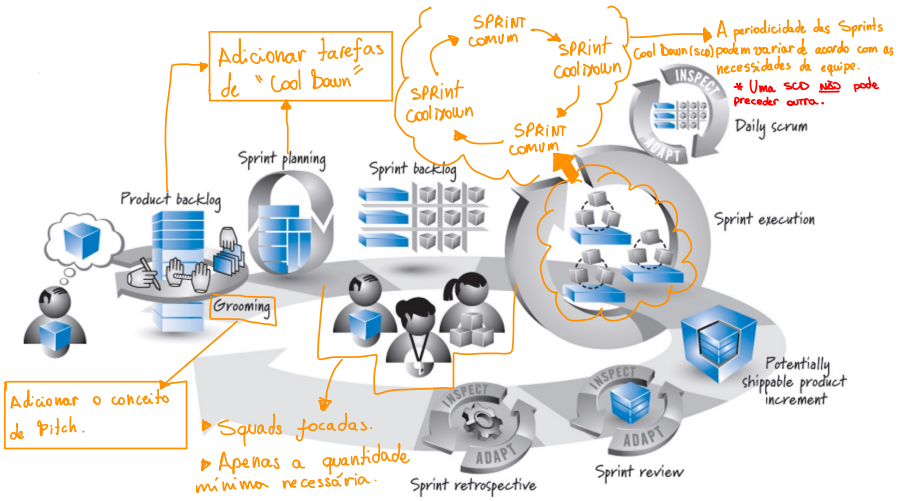

Figura 3. Processo proposto da combinação Scrum e Shape Up

que faz com que esse tipo de literatura seja tão acessível.

Publicar em grandes revistas, conferências e outros canais da literatura branca é uma tarefa que demanda tempo e esforço, uma vez que esses canais precisam manter a qualidade dos textos publicados. Isso impede que pessoas, que não fazem parte e não estão habituadas com a escrita científica, possam publicar suas experiências. Muitas vezes, essas mesmas pessoas estão diretamente conectadas com o estado da prática de determinados assuntos, isso significa que a barreira da publicação (da literatura branca) pode impedir a disseminação de conteúdos valiosos sobre a realidade da indústria. Sendo assim, a literatura cinza combate essa barreira democratizando o acesso e publicação de conhecimento para qualquer pessoa que desejar compartilha-lo.

Uma das desvantagens e preocupações intrínsecas a GL é qualidade e veracidade do que se é publicado. Garousi et al. inclusive cita Williams and Rainerqunato que afirmam que os materiais oriundo da GL precisam ser rigorosos, relevantes, bem escritos e baseados em experiência para serem considerados confiáveis para pesquisadores de Engenharia de Software [Williams and Rainer 2017]. Portanto, a qualidade acessível da GL não impede a descoberta de literatura válida.

No âmbito científico ao realizar tarefas de sistematização de determinado assunto por meio da literatura cinza, se faz necessário o alto rigor quanto a qualidade do estudo, a fim de apresentar resultados verídicos. Entretanto, quanto a busca despretensiosa e cotidiana, que também necessita de dados confiáveis, possuem um rigor menor. Mesmo assim é possível verificar a fonte. Existem grandes canais de literatura cinza que são de vasta utilidade para o profissionais e pesquisadores da Engenharia de Software, como por exemplo a plataforma Medium que possui perfis oficiais de grandes organizações como a Netflix.

Comumente, a literatura branca costuma trazer estudos primários contendo o estado da arte de determinado assunto. Isso é de extrema importância para a evolução da Engenharia de Software (ES). Contudo, conhecer o estado da prática é indispensável dentro da ES. A natureza da literatura cinza é que seus autores são praticantes da indústria de

ES, isso significa que a GL é uma fonte valiosa para conhecer o que a indústria está aplicando. Dito isso, a GL precisa fazer parte do meio científico da Engenharia de Software, assim como faz parte do cotidiano de vários profissionais da área.

Referências

[Amershi et al. 2019] Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B., and Zimmermann, T. (2019). Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 291–300. IEEE.

[Garousi et al. 2020] Garousi, V., Felderer, M., Mäntylä, M. V., and Rainer, A. (2020). Benefitting from the grey literature in software engineering research. In *Contemporary Empirical Methods in Software Engineering*, pages 385–413. Springer.

[Giray 2021] Giray, G. (2021). A software engineering perspective on engineering machine learning systems: State of the art and challenges. *Journal of Systems and Software*, 180:111031.

[Hemmati et al. 2013] Hemmati, H., Nadi, S., Baysal, O., Kononenko, O., Wang, W., Holmes, R., and Godfrey, M. W. (2013). The msr cookbook: Mining a decade of research. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 343–352. IEEE.

[Khomh et al. 2018] Khomh, F., Adams, B., Cheng, J., Fokaefs, M., and Antoniol, G. (2018). Software engineering for machine-learning applications: The road ahead. *IEEE Software*, 35(5):81–84.

[Matsubara et al. 2023] Matsubara, P., Steinmacher, I., Gadelha, B., and Conte, T. (2023). Moving on from the software engineers' gambit: an approach to support the defense of software effort estimates. *arXiv preprint arXiv:2302.07229*.

[Wan et al. 2019] Wan, Z., Xia, X., Lo, D., and Murphy, G. C. (2019). How does machine learning change software development practices? *IEEE Transactions on Software Engineering*, 47(9):1857–1871.

[Williams and Rainer 2017] Williams, A. and Rainer, A. (2017). Toward the use of blog articles as a source of evidence for software engineering research. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, pages 280–285.