Software Project Canvas v1.0

Geraldo Xexéo

2023-08-28 11:08:27

Sumário

Lista de Figuras

Resumo

Esse documento apresenta o Software Project Canvas, uma ferramenta ágil que facilita a colaboração das partes interessadas na discussão e definição inicial de um projeto de software.

Para a sua criação foram observadas 6 dificuldades ao ensinar ou utilizar o *Project Model Canvas* no contexto de projetos de software.

As modificações feitas visam facilitar a explicação e o uso de um canvas no planejamento inicial de um projeto de desenvolvimento de software, além de evitar perda de tempo, durante a reunião de concepção do projeto, com discussões semânticas sobre o significado das coisas que não adiantam a definição do projeto.

1 Introducão

Esse artigo apresenta o Software Project Canvas[1], representado na Figura 1.1, uma ferramenta ágil cujo objetivo é facilitar a colaboração das partes interessadas na discussão e definição inicial de um projeto de software.

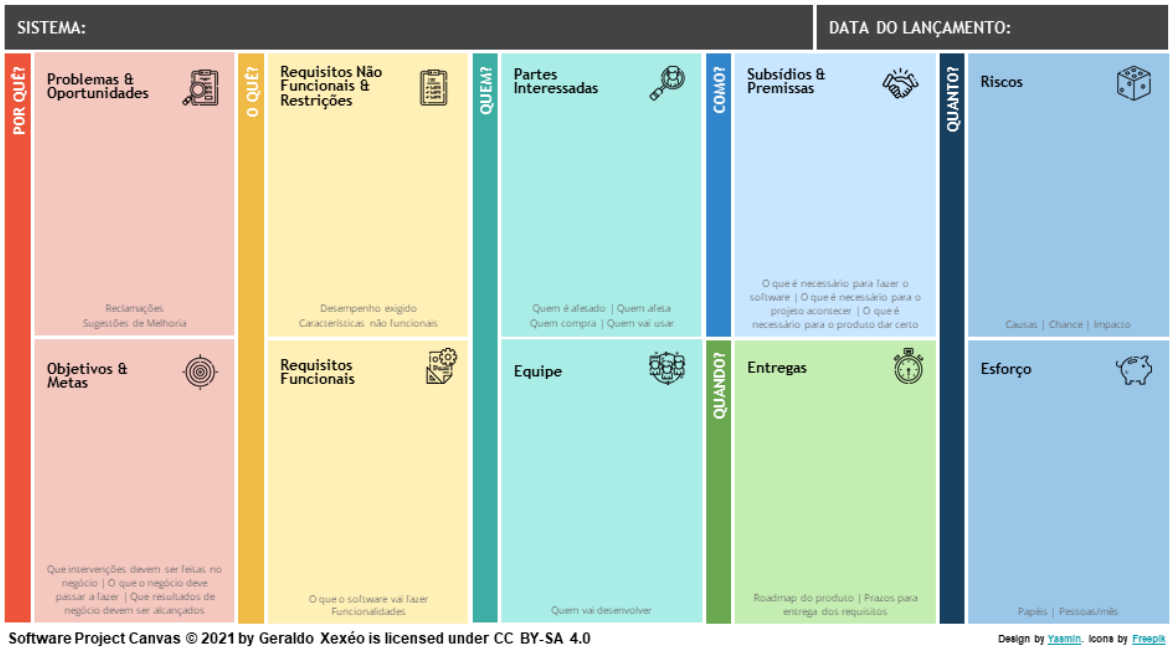


Figura 1.1: O Software Project Canvas. Imagem do autor

O *Software Project Canvas* foi motivado por certas dificuldades encontradas ao ensinar e usar o *Project Model Canvas* (Finocchio Jr., 2013) no contexto de projetos de Engenharia

[1]Este documento e arquivos com o Software Project Canvas podem ser encontrados em `https://github.com/xexeo/Software-Project-Canvas`

de Sofware. Após identificar essas dificuldades, foram feitas propostas de correção, e variações do modelo foram avaliadas por alguns especialistas e iniciantes na área.

2

O que são Canvas

Canvas são ferramentas colaborativas que servem de ponto focal em uma atividade onde algo vai ser controlado, decidido ou planejado. São compostos de um quadro branco dividido em várias áreas, cada uma representando uma pergunta, assunto ou tópico em discussão, onde são colocados *post-its* contendo a informação desejada.

Alguns autores, como Finocchio Jr. (2013), associam os canvas a neurociência, tanto por usar de características de agrupamento e proximidade, que auxiliam a compreensão dos conceitos, como por apoiarem uma atividade colaborativa que tende a criar engajamento nos membros da reunião em torno do assunto.

O primeiro canvas de grande impacto na indústria foi o *Business Model Canvas*, proposto por Osterwalder, Pigneur e Clark (2010), apresentado ainda vazio na Figura 2.1. Ele é destinado ao planejamento de um modelo de negócio para um empreendimento. Nele, cada área receberá, em uma ordem pré-estabelecida e ao longo de uma reunião facilitada, *post-its* que dão respostas às perguntas indicadas nos quadros. Por exemplo, no quadro "Customer Segments" pode aparecer um *post-it* onde está escrito "adolescentes praticantes de Skate".

A partir do sucesso do *Business Model Canvas*, outros canvas foram propostos por pesquisadores, consultores ou pessoas interessadas em algum problema específico. No Brasil, Finocchio Jr. propôs, com bastante sucesso, o *Project Model Canvas* (Finocchio Jr., 2013), que serviu de inspiração para nossa proposta.

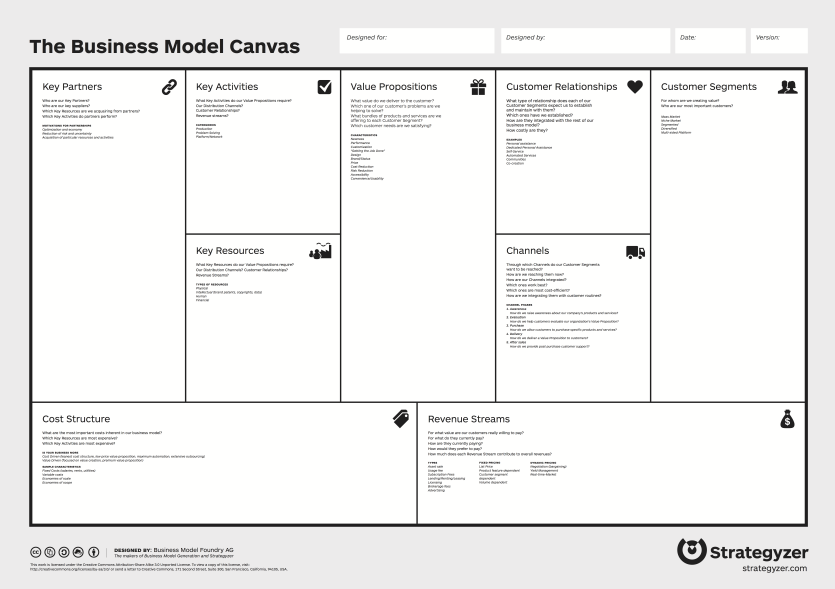


Figura 2.1: O Business Model Canvas. Fonte: Osterwalder, Pigneur e Clark (2010)

3

Projetos, Agilidade e Canvas

A Gestão de Projetos é uma área de conhecimento essencial ao sucesso de um projeto. Segundo Kerzner (2017), um projeto tem sucesso se acaba no prazo, dentro do orçamento, com o desempenho especificado, aceito pelos clientes e usuários, com mudanças mínimas no escopo acordadas entre as partes interessadas, sem causar distúrbio ao fluxo de trabalho da organização e, quando desejado, sem mudar a cultura da organização. Além disso, é necessário satisfazer as partes interessadas (PMI, 2017). Para alcançar todos esses objetivos é necessário planejar, monitorar e controlar o projeto ao longo de sua realização.

Ao longo dos anos, a área de gestão de projetos desenvolveu processos eficazes e reconhecidos, sendo que muitos foram consolidados em padrões ou livros de referência (Kerzner, 2017; PMI, 2017). A Engenharia de Software, disciplina relativamente nova em relação as outras disciplinas onde esses processos foram desenvolvidos, foi criada com o intuito de aplicar as práticas de Engenharia ao desenvolvimento de software, incluindo as práticas de gestão (IEEE Computer Society, 2014; Naur e Randell, 1969; Pressman e Maxim, 2019).

Com a experiência de desenvolver software sob a pressão do mundo de negócios, algumas dessas práticas, ou seus princípios, começaram a ser contestadas por não serem adequadas ao desenvolvimento de software, exigirem uma carga muito alta de trabalho adicional (*overhead* alto), ou exigirem muitos documentos. Isso resultou no fortalecimento dos defensores de processos de desenvolvimento de software mais leves, capazes de reagir rápido a mudanças, que acabaram sendo nomeados métodos ágeis ou enxutos. Essa tendência acabou por afetar toda a indústria, como mostra a ampla adoção dos métodos ágeis (digital.ai, 2022).

Entre os princípios que não funcionaram na Engenharia de Software está o da execução de projetos de forma estritamente sequencial, onde uma fase é realizada de forma completa e fornece subsídios para a próxima fase, sendo a entrega feita, em sua totalidade, no final do projeto, o que é conhecido como Processo em Cascata. Atualmente, há ampla

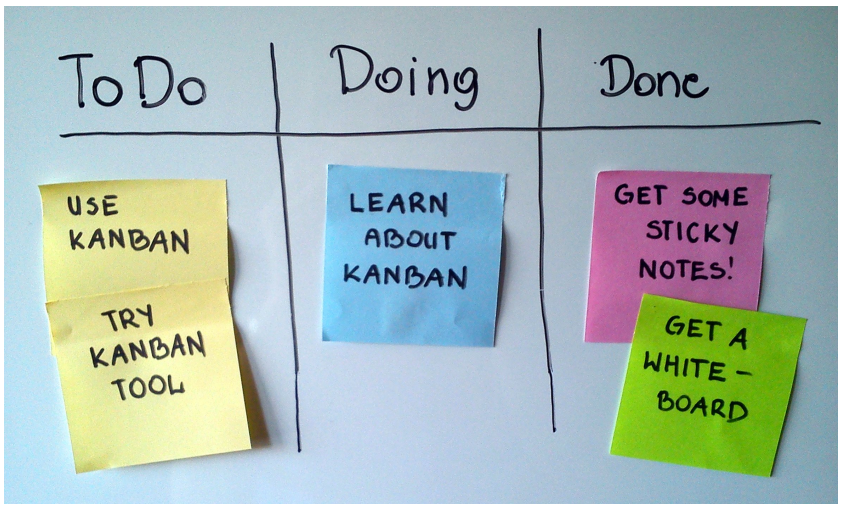


Figura 3.1: Um quadro Scrum/Kanban muito simples. Fonte: por Jess Iasovski, CC BY-SA 3.0

convicção que é essencial usar processos iterativos e incrementais, onde as entregas são pequenas e feitas ao longo do projeto, principalmente devido a sua maior taxa de sucesso (Pressman e Maxim, 2019; The Standish Group, 2015).

Uma prática comum dos métodos ágeis é o uso de quadros brancos com áreas delimitadas por assunto, indicando, de forma transparente a todos, alguma informação controlada por meio do quadro. O mais famoso desses quadros é provavelmente o a adaptação, para o processo *Scrum* (Satpathy et al., 2016), do *Kanban*, apresentado em uma forma simplificada na Figura 3.1. O Kanban é um método de produção que pode ser considerado um predecessor dos métodos ágeis e faz parte de um conjunto de práticas utilizadas na Toyota e que iniciou uma revolução de conceitos de produção e qualidade, alterando o até então paradigma *Fordista* de produção (Gross e McInnis, 2003). Outros métodos adotaram quadros semelhantes, como o SEMAT (Jacobson, Lawson e Ng, 2019), que usa cartas pré-definidas posicionadas no quadro para indicar o estado de um processo.

Na Figura 3.1 também é possível notar que o quadro não precisa ser feito de forma impressa, podendo ser simplesmente desenhado em um quadro branco, ou mesmo na parede com fitas, e como os *post-its* são usados, nesse caso para indicar tarefas a fazer, sendo feitas e prontas.

3.0.1 O *Project Model Canvas*

O *Project Model Canvas* (Finocchio Jr., 2013), apresentado na Figura 3.2, é um canvas destinado ao planejamento inicial de um projeto, baseado em uma metodologia de 4 grandes etapas, descrita na Figura 3.2. Ele possui 12 quadros que são preenchidas passo a passo na etapa de Concepção, aprimorados nas fases de Integração e Revisão, e depois tem seu resultado comunicado as partes interessadas na fase de Compartilhamento. Sendo

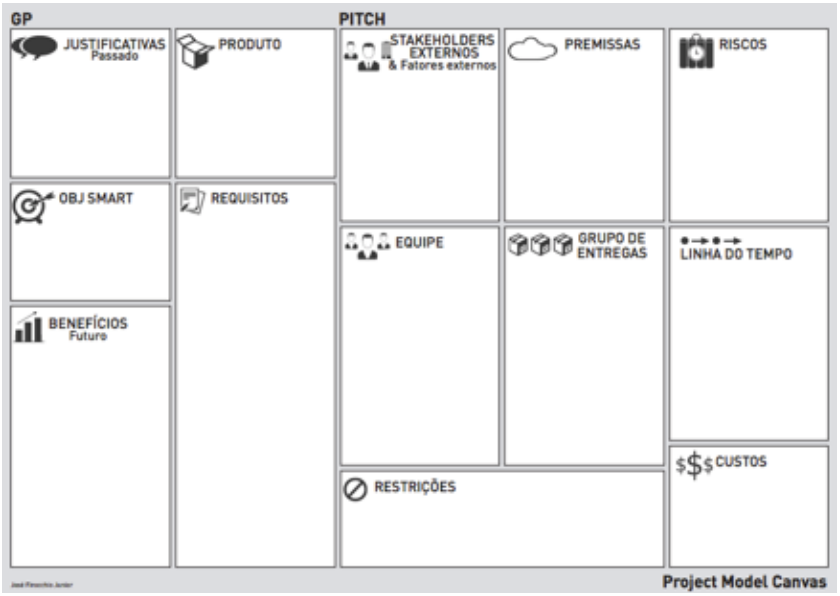


Figura 3.2: O processo de planejamento usando o Project Model Canvas. Fonte: Finocchio Jr. (2013)

destinado a qualquer tipo de projeto, pode ser também aplicado a projetos de software, porém não necessariamente de forma perfeita.

Esse canvas é uma abordagem típica de projetos ágeis ou enxutos, aplicada ao planejamento de projetos de qualquer tipo. Isso é, simultaneamente, uma vantagem e uma desvantagem do *Project Model Canvas*, já que, sendo geral, pode ser aplicado com sucesso em qualquer área, mas não sendo específico, acaba por ter algumas incompatibilidades com projetos de áreas específicas, como, e em especial, a Engenharia de Software.

Projetos de software têm características diferentes de projetos tradicionais que afetam diretamente o planejamento. Entre elas, podemos citar: a dificuldade de medir ou prever o tamanho da entrega, medido em esforço ou em prazoBoehm, Abts e Brad Clark; Cohn; Pressman e Maxim, a necessidade de aprendizado da equipe em relação ao domínio da aplicação ao longo do projeto e a má-adequação às formas tradicionais e hierárquicas de equipe (Constantine, 1993). Essas e outras características que diferenciam o projeto de desenvolvimento de software afetam diretamente a capacidade de planejamento, exigindo estratégias diferenciadas em relação as estratégias tradicionais.

Ao ensinar e usar o *Project Model Canvas* para o planejamento de projetos de desenvolvimento de software, também podem ser percebidas algumas dificuldades dos alunos e praticantes com os quadros, listadas a seguir.

1. A dificuldade de separar o pitch do produto e do objetivo SMART, e depois separar este do produto. Essas duas primeiras dificuldades advêm do fato de ser realmente difícil diferenciar o produto específico sendo criado do objetivo para o qual é criado. Isso, porém, pode não ser necessário no projeto de software, já que justificativas, benefícios e requisitos cumprem bem esse papel.

2. A limitação a apenas um objetivo SMART para o software, quando é comum softwares atenderem várias funcionalidades que são normalmente criadas para atender objetivos;
3. A dificuldade de separar requisitos não-funcionais de restrições;
4. A não separação dos requisitos entre funcionais e não funcionais, tradicional na área de software, e importante no projeto de software, o que afeta diretamente a capacidade de estimar os custos do software, já que a prática mais frequente, formalizada nos Pontos de Função (IFPUG, 2012), implica em calcular a funcionalidade entregue e depois melhorar a estimativa a partir de exigências feitas pelos requisitos não funcionais.
5. Um excesso de esforço na separação entre grupos de entregas e linha do tempo, que ainda muitas vezes obriga a re-arranjar entregas ou requisitos, quando as práticas mais atuais, por exemplo baseadas em *road-maps* do produto, tomam essa decisão de forma simultânea.
6. A dificuldade de definir custos, já que basicamente todo custo de um projeto de software vem da mão de obra, ou seja, do esforço necessário para desenvolve-lo, geralmente medido em pessoas-mês ou pessoas-hora.

Todos esses problemas levam a uma ineficiência no processo de ensino e de uso do *Project Model Canvas*, já que o tempo é gasto, repetidas vezes, na elucidação de conceitos ou restrições que não são essenciais ao projeto de software.

Para resolver essas dificuldades, este artigo propõe o uso do *Software Project Canvas*. Além de ser criado especificamente para projetos de software, ele trata os problemas detectados no *Project Model Canvas* da seguinte forma:

- elimina o Pitch, substituindo-o pelo produto de software a ser criado;
- não apresente o objetivo SMART único, considerando que o objetivo é entregar o software;
- trata objetivos e metas com uma abordagem aceita na Engenharia de Software (Ruble, 1997);
- unifica no processo o tratamento de restrições e requisitos não-funcionais, para facilitar o processo de trabalho;
- separa os requisitos funcionais e não funcionais, os últimos com adição das restrições, em áreas diferentes do canvas, atendendo a prática comum da Engenharia de Software (Pressman e Maxim, 2019);
- unifica entregas e linha do tempo em uma caixa, onde as entregas já devem vir com uma data, ou será criado um roadmap do produto, típica de gestão ágil, e
- transforma todo o cálculo do custo em uma previsão do esforço (Cohn, 2005).

Como resultado, o processo é transformado de 12 para 9 passos, já que mesmo em áreas separadas no canvas os requisitos são todos levantados no mesmo momento, facilitando sua explicação. Além disso, são evitadas muitas discussões desnecessárias, como onde colocar algo que tanto pode ser visto como uma restrição ou como um requisito não-funcional, por diferentes partes interessadas.

4

O *Software Project Canvas*

O Software Project Canvas utiliza 10 quadros, que serão descritos nesta seção. Para cada quadro foi buscado explicar o que ele trata, métodos que podem ser usados para auxiliar seu preenchimento e alguns exemplos. Os quadros se organizam em 6 grupos que respondem às perguntas 5W2H (Nakagawa, 2012): *Why, What, Who, When, How* e *How Much*. Não é necessário, em geral, reponder a pergunta *Where* para projetos de software.

- Por que: o que justifica o projeto, os Problemas e Oportunidades, e os Objetivos e Metas;
- O que: o que é o projeto, indicado pelos Requisitos Funcionais, Requisitos Não-Funcionais e Restrições;
- Quem: indicado pelas Partes Interessadas e a Equipe;
- Como: indicado pelos Subsídios e Premissas;
- Quando: indicado pelas Entregas, e
- Quanto: indicado pelos Riscos e Esforço.

4.0.1 Problemas e Oportunidades

Todo projeto se inicia porque há algum problema a ser resolvido ou uma oportunidade a ser aproveitada (Kerzner, 2017; OMG, 2018). Ou seja, se deseja uma mudança no estado da organização, incluindo o lançamento de produtos ou serviços para o mercado. Nas discussões sobre projetos de software, os problemas logo aparecem na forma de descontentamento com um sistema atual, informatizado ou não, que funciona com falhas, erros ou não-conformidades, ou uma expectativa para um produto novo.

Um problema é uma diferença entre um estado percebido e um estado desejado, que deverá ser tratada pelo projeto. São dificuldades a serem resolvidas, desvios de um padrão, regulamento ou lei, falta de capacidade de realizar uma tarefa, etc.

Problemas trazem prejuízo, sejam eles financeiros ou de outra forma, como impacto na imagem da empresa. Quando esses prejuízos podem ser medidos de forma quantitativa,

como o tempo de atendimento, deve ser indicada a forma de medi-los, de maneira que isso possa ser usado para verificar os benefícios que o projeto trará. Quando forem prejuízos mais qualitativos, como percepção dos clientes, também se deve buscar formas de avaliar o impacto do projeto.

Os problemas podem ser divididos de várias formas. Uma delas é entender que existem problemas por não fazer algo, problemas funcionais, ou por não fazer algo como deve ser feito, problemas operacionais. Ambos têm efeitos no negócio, que são problemas de negócio, ou sintomas. Um problema de negócio pode ser a existência de longas filas em um restaurante, que pode ser causado por um problema operacional, a demora no cálculo da conta. Outro problema de negócio pode ser a dificuldade de entrar em contato com os clientes, que pode ser causado por um problema funcional, como a não existência de um cadastro de clientes.

Problemas podem ser descobertos e priorizados com uso de método de Pareto. É possível, porém, que o método de Pareto detecte apenas os sintomas. Por exemplo, se usarmos o método de Pareto para analisar as reclamações feitas em um restaurante, podemos descobrir que a maioria é sobre o excesso de tempo de espera na fila. Porém, esse não é o verdadeiro problema, mas sim um sintoma causado por, entre outras coisas, a demora em fechar uma conta. Para identificar as causas-raiz dos problemas, ou seja, o verdadeiro problema que deve ser resolvido, podem ser usadas técnicas como o Diagrama de Espinha de Peixe, ou os 5 Porquês (Gray, Brown e Macanufo, 2010). Existem outras técnicas disponíveis, principalmente nas literaturas de Controle de Qualidade em geral, Qualidade de Software, Gestão e Marketing.

Já as oportunidades são percepções que algo pode ser feito para que a organização tenha uma vantagem em relação ao estado atual. Oportunidades podem ser levantadas de várias formas, incluindo o uso de *brainstorming* (Tracy, 2015).

Problemas e oportunidades podem ser priorizados por meio de técnicas como MoSCoW (Stapleton, 2003) e dot-voting (Gray, Brown e Macanufo, 2010).

Exemplos de problemas: documentos perdidos, perda de vendas ao longo do processo, fila excessiva, reclamações dos clientes, reclamações dos fornecedores, muitos atrasos nas entregas, produtos ou serviços que dão prejuízo, etc.

Exemplos de oportunidades: novo produto vai ser lançado, dados disponíveis para *data mining*, invenção de um novo serviço, detecção de uma nova necessidade dos clientes, etc.

4.0.2 Objetivos e Metas

Nesta seção do *Software Project Canvas* serão apresentadas as melhorias que o sistema traz ao negócio. Há certa confusão dos termos objetivos e metas, então é importante definir esses termos para o *Software Project Canvas*. Outro termo que pode ser usado é benefício.

Objetivos são mudanças que desejamos na organização. Esses objetivos devem ser SMART, isto é:

- **S**pecíficos,
- **M**ensuráveis,
- **A**lcançáveis,
- **R**ealistas ou **R**ealizáveis, e
- delimitados no **T**empo.

Um exemplo de objetivo SMART simples é: construir um muro (S), de 2 metros de altura (M), com tijolos (A), a um preço de R$2000,00 (R), em um mês(T).

Em Engenharia de Software, nem sempre é fácil determinar alguns desses fatores, como a medida que indica que o projeto está completo, mas é necessário se esforçar para isso, já que é preciso, em algum momento, se certificar que o projeto alcançou seus requisitos mais básicos.

Metas são resultados que esperamos quando houver a mudança trazida pelo projeto, ou seja, os benefícios trazidos ao negócio por termos alcançados os objetivos. Metas precisam de medidas, subjetivas ou objetivas, também chamadas de qualitativas ou quantitativas, associadas a elas. Por exemplo, o muro pode ter sido colocado para diminuir a quantidade de pessoas que entra em um local, o que pode ser contado, ou para aumentar a sensação de segurança dos moradores, o que é um sentimento subjetivo. Cada objetivo deve ter pelo menos uma meta, e cada meta uma forma de avaliar (Ruble, 1997). Todo objetivo, e suas metas, deve endereçar um item que aparece na justificativa (Finocchio Jr., 2013).

Uma das várias formas de buscar benefícios é usar a heurística IRACIS[1] (Gane e Sarson, 1979; Ruble, 1997), que os divide em 3 tipos: aumentar receitas, evitar cursos e melhorar serviços.

Exemplo de objetivo: detecção de necessidade de repor o estoque, com a meta diminuir reclamações sobre falta de item no estoque e as métricas: número de reclamações que um item não estava no estoque, e número de vendas perdidas porque o item não estava no estoque.

4.0.3 Requisitos Funcionais, Não-Funcionais e Restrições

Segundo a norma ISO/IEC/IEEE 24765:2017, " um requisito é uma condição ou capacidade necessária para uma parte interessada resolver um problema ou atingir um objetivo.", ou, ainda segundo a mesma norma, "Ele deve ser possuído ou alcançado por um sistema, componente de um sistema para satisfazer um contrato, padrão, especificação ou outro documento formalmente imposto" (IEEE, 2017).

Requisitos podem ser divididos em dois grandes grupos: funcionais ou não-funcionais. Restrições são tipos especiais de requisitos não-funcionais impostos ao projeto. Requisitos

[1] Increase Revenue, Avoid Costs and Improve Service

funcionais dizem que funcionalidade o software entrega, na prática, que funcionalidades o usuário exige do software. Já requisitos não-funcionais especificam como essas funcionalidades devem acontecer ou desempenhar. Requisitos funcionais são, em geral, independentes de tecnologias McMenamin e Palmer, 1984.

Os Requisitos não-funcionais, funcionais e as restrições podem ser levantados no mesmo passo, apesar de haver uma separação no quadro. Isso facilita a discussão do que deve ser entregue, pois não é necessário dizer "guarde esse requisito para o próximo passo".

4.0.4 Requisitos Funcionais

Requisitos funcionais dizem o que o software deve fazer. Recomendamos que isso seja feito na forma de histórias do usuário, cujo formato padrão é uma frase do tipo "Como gerente, eu quero listar os vendedores e suas vendas, para determinar os melhores vendedores da empresa" (Cohn, 2004).

Outra forma possível é usar objetivos do usuário, ou seja, os nomes de casos de uso, como: listar melhores vendedores (Cockburn, 2000).

É importante usar uma estratégia para não deixar a lista de requisitos funcionais muito longa. A primeira é usar casos de uso ou história de usuário de nível de abstração alto, ou seja, usar uma estratégia *top-down* no desenvolvimento dessa lista. Dessa forma, os requisitos poderiam ser semelhantes aos encontrados em uma história do usuário de alto nível (Jacobson, Spence e Bittner, 2011), em uma DFD de nível 0 (Gane e Sarson, 1979), ou ainda em diagrama de casos de uso do nível pipa ou nuvem (Cockburn, 2000). O outro é listar apenas os mais importantes. Para isso, pode ser usada a técnica de priorização MoSCoW (IIBA, 2011; Stapleton, 2003) para eliminar os requisitos não adequados à fase inicial.

Exemplos de requisitos funcionais são:

- no caso de usar casos de uso: comprar livros, enviar mensagem, relatar ausência de item no estoque, e
- no caso de histórias do usuário: como gerente eu quero listar os vendedores e suas vendas para descobrir os melhores vendedores, como leitor eu quero listar os novos lançamentos para poder decidir que livro vou comprar.

4.0.5 Requisitos Não-Funcionais e Restrições

Requisitos não-funcionais e restrições falam da forma como os requisitos funcionais vão ser alcançados. Apesar de podermos considerar que os requisitos não-funcionais são desejados e as restrições são impostas, é difícil para um pessoa não-treinada fazer essa separação de forma rápida, então não nos preocupamos em fazê-la agora. Em outra fase do processo de criação do *Software Project Canvas*, as restrições, ou requisitos não-funcionais, que inviabilizam o trabalho, podem ser eliminadas ou amenizadas.

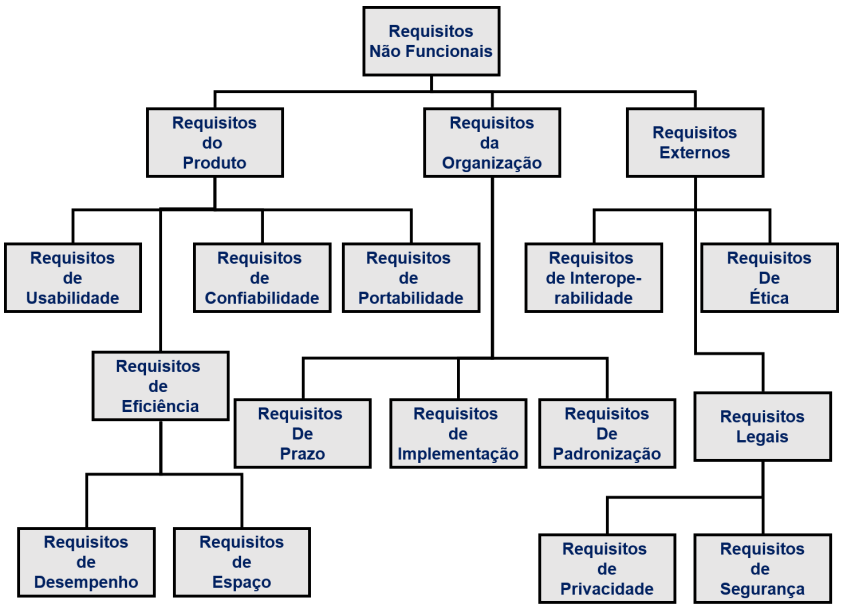


Figura 4.1: Taxonomia de Requisitos Não Funcionais. Fonte: Sommerville (2015)

Estes requisitos podem ser de vários tipos e conhecer uma taxonomia pode ajudar a levantá-los. Sommerville (2015) apresenta uma hierarquia de requisitos não funcionais que foi adaptada na Figura 4.1.

Restrições limitam o trabalho realizado e devem ser bem específicas, quantificadas e ainda indicar por quem são impostas.

Os requisitos não-funcionais podem ser associados a todos ou apenas a alguns requisitos funcionais. Se for o último caso, é interessante fazer essa indicação.

Exemplo de requisito não-funcional: mensagens devem ser na forma de e-mail, o envio de um e-mail deve demorar no máximo 2 minutos, a senha deve seguir as regras de segurança definidas pela empresa. Exemplos de restrições: a linguagem de programação só pode ser Java, o banco de dados usado pelo projeto tem que ser Oracle 10.

4.0.6 Partes Interessadas

Uma parte interessada (P.I.), ou *stakeholders*, é "um indivíduo, grupo ou organização que pode afetar, ser afetado, ou perceber que será afetado, por uma decisão, atividade, ou resultado, parcial ou final de um projeto" (PMI, 2017).

As partes interessadas podem ter opiniões diferentes em relação a como projeto vai afetá-las positiva ou negativamente, seja por sua participação ou interesse, quanto a percepções, corretas ou incorretas. Esse efeito pode ser causado tanto pelo projeto como um todo, quanto por partes específicas do mesmo. Algumas serão consideradas *champions* ou evangelistas do projeto, outras podem ser neutras, e ainda outras abertamente, ou secretamente, contra o projeto.

Identificar as todas as partes interessadas é o passo inicial dos processos de gestão das partes interessadas (Jacobson, Lawson e Ng, 2019; PMI, 2017), com o objetivo de garantir a satisfação das mesmas, e com isso o sucesso do projeto. No SEMAT, por exemplo, identificar as partes interessadas é o primeiro estado para o Alpha *Stakeholders*, seguido da escolha de representantes (Jacobson, Lawson e Ng, 2019). Os estados seguintes são nomeados: envolvidas, em acordo, satisfeita para a implantação e satisfeita com o uso (Jacobson, Lawson e Ng, 2019).

Nessa parte do *Software Project Canvas* são listadas todas as partes interessadas do projeto. Devem ser consideradas especialmente três tipos de P.I., os usuários finais, o patrocinador e o governo, como órgão legislador a que o projeto pode estar submetido. Para listá-las, pode ser necessário o uso de técnicas como *brainstorming* (Tracy, 2015) ou *writestorming* (Gray, Brown e Macanufo, 2010).

Outras partes interessadas comuns de encontrarmos são: clientes dos usuários, fornecedores dos usuários, futuros operadores do software, sistemas que podem ser acessados ou acessar o software, e seus mantenedores, organizações e suas partes que podem ter algum interesse no resultado do software.

É importante que sejam usados, sempre que possível, termos ligado ao negócio. Assim, um médico não tem clientes, mas pacientes, enquanto um banco tem correntistas.

Fatores externos fora do controle do projeto também podem ser listados aqui. Não é incomum que projetos de software sofram o impacto do valor do dólar, por exemplo.

Por conveniência do processo de definição do projeto de software, a equipe de desenvolvimento, que é, em geral, uma parte interessada, não deve ser considerada nesse momento.

Exemplos de partes interessadas são: vendedores, compradores, pacientes, patrocinador, diretoria de operações, etc.

4.0.7 Equipe

Nesse quadro é indicada a equipe necessária para o desenvolvimento do projeto. Cada *post-it* deve indicar um papel a ser tomado pelos membros da equipe. Quando for necessário mais de uma pessoa, isso deve ser indicado no mesmo *post-it*. Possivelmente, algumas pessoas podem estar nomeadas no mesmo *post-it*, ou em outro, de outra cor, possivelmente menor, colocado sobre o primeiro.

Alguns papéis são esperadosFinocchio Jr., como o gerente do projeto e um ou mais programadores. Alguns papéis também são esperados em funções do processo de desenvolvimento de software sendo usado. Por exemplo, se é usado o Scrum, esperamos encontrar o Scrum Master, o Product Owner, que pode, por outro lado, ser listado como parte interessada, e o Time, que será formado de várias pessoas.

4.0.8 Subsídios e Premissas

Nesta seção do *Software Project Canvas* deve aparecer tudo que é necessário para que o projeto tenha sucesso. Tipicamente, em projetos de software, subsídios e premissas estão ligados ao acesso da equipe às Partes Interessadas, e a participação das mesmas quando necessário.

Também é comum que existam premissas ligadas a disponibilidade da infraestrutura necessária para o sucesso do projeto. Em muitos projetos também é necessário tratar aqui a disponibilidade financeira.

Premissas e subsídios são garantidas pelas partes interessadas, em favor da equipe. Caso uma parte interessada não cumpra seu compromisso em garantir a premissa, ocorre um risco do projeto. Assim, a análise de risco deve considerar todas as premissas feitas.

Exemplo de premissas ou subsídios: os computadores novos serão entregues dia 15 de maio, os computadores possuirão uma placa gráfica compatível com a NVidia 3060, serão contratados cinco novos programadores, o patrocinador disponibilizará 4 horas por semana para o projeto.

4.0.9 Entregas

Um projeto de software moderno é feito de forma iterativa e incremental (Pressman e Maxim, 2019; Rubin, 2013), o que implica em várias pequenas entregas, que se iniciam com protótipos, provas de conceito ou um produto mínimo viável, e se seguem de versões ou *releases* que fornecem cada vez mais valor aos seus usuários.

Essas entregas devem ser previstas, sendo que as entregas a curto prazo devem ser previstas com mais detalhe de que as de longo prazo, já que os métodos de desenvolvimento de software usados na indústria se tornaram, na sua maior parte, adaptativos (digital.ai, 2022).

Em cada entrega é entregue um conjunto de funcionalidades, ou seja, são cumpridos requisitos funcionais. Também são garantidos os requisitos não funcionais e as restrições. Assim, as entregas devem ser construídas a partir desses quadros anteriores, levando em consideração a equipe prevista.

É possível que as entregas sejam descritas de forma a implementar parcialmente vários requisitos de forma parcial, como o que é proposto por Jacobson, Spence e Bittner (2011). Por exemplo, um requisito que diga "Como professor eu quero enviar mensagens para alunos para dar avisos sobre a próxima aula", pode em uma entrega enviar e-mails, na seguinte incluir mensagens por WhatsApp, e assim por diante. Nesse caso, deve haver cuidado na descrição das entregas, para deixar claro que os requisitos não são cumpridos integralmente.

Entregas também podem ser desenvolvidas, mais tarde, ou como resultado intermediário para gerar o *Software Project Canvas*, por meio de uma Estrutura Analítica de Projeto

(EAP), também conhecida pelo nome em inglês, que faz mais sentido, *Work Breakdown Structure (WBS)*. Nesse caso, deve ser seguida a abordagem de criar uma hierarquia de entregas, e não de processos.

É importante que as entregas contenham prazos, que indicam o compromisso inicial da Equipe para satisfazer as Partes Interessadas.

Exemplos de entregas: lista de livros, seleção e compra - em 40 dias; relatórios de acompanhamento - em mais 20 dias, sistema de recomendação - em 60 dias;

4.0.10 Riscos

Riscos são incertezas que podem influenciar o resultado do projeto. Atualmente riscos são divididos em ameaças e oportunidades, isto é, existem riscos negativos e positivos.

É importante para a gestão do projeto gerenciar o risco, para isso devem ser conhecidos sua causa, isto é, porque ele acontece, sua probabilidade de ocorrer e seu impacto. Com isso é possível calcular o efeito do risco no projeto (Kerzner, 2017; PMI, 2017).

Os riscos podem ser aceitos desprezados, por terem um impacto baixo, aceitos e gerenciados, mitigados, ou, ainda, eliminados ou transferidos. Essas ações foram listadas do que deve ser feito para os riscos de impacto mais baixo até o mais alto.

Todo projeto tem riscos gerais e específicos. Riscos gerais são geralmente reconhecidos pela comunidade. Um risco geral típico de projeto de software é a perda de um membro importante da equipe, ou que possui um conhecimento único.

Toda premissa é candidata a gerar um risco, que deverá ser analisado. Existem categorias reconhecidas, como riscos técnicos, externos, organizacionais ou de gerenciamento (PMI, 2017). Para os riscos gerais podemos encontrar *checklists* a serem seguidas (Thomsett, 2002). Já para os riscos específicos é necessário analisar atentivamente o projeto, suas premissas, complexidades e outros fatores.

Existem várias heurísticas para tratar o risco, mas a princípio, para cada risco levantado deve ser estimada sua probabilidade e seu impacto. O produto dos dois implica em uma reação. Finocchio Jr. (2013) sugere usar duas escalas de 1 a 10 para avaliar a probabilidade e o impacto, fazendo o produto e obtendo as recomendações segundo a Tabela.

Exemplo de riscos: não conseguir contratar cinco novos funcionários com a habilidade necessária (probabilidade 7, efeito 8), servidor atual não aguentar a carga do novo sistema.

4.0.11 Esforço

Nesse quadro deve ser estimado o esforço para desenvolver o software em homens-mês ou homem hora. No momento da reunião onde o canvas é usado, não é possível ainda usar uma técnica fortemente quantitativa para isso, pois não se espera que seja possível

Probabilidade × Impacto	Ação
1 a 3	Aceitar passivamente
4 a 10	Aceitar ativamente, com um plano de resposta
11 a 36	Mitigar o risco
37 ou mais	Eliminar ou transferir o risco

Tabela 4.1: Ações recomendadas por (Finocchio Jr., 2013) de acordo grau de importância do risco

.

nem estimar o tamanho real do software, então isso deve ser feito a partir da experiência do grupo, com comparações com projetos anteriores.

Técnicas adequadas nesse ponto são variações do método de Delphi (Dalkey, 1966), como o *Planning Poker* (Cohn, 2005), ou uso de heurísticas como buscar previsões pessimistas (P), médias (M) e otimistas (O) e calcular uma média ponderada ($M_p = \frac{P+4\times M+O}{6}$) (PMI, 2017).

Outra possibilidade é usar técnicas rápidas de Contagem de Pontos de Função (IFPUG, 2010).

5 A Metodologia do *Software Project Canvas*

A metodododologia utilizada do *Software Project Canvas* possui 4 grandes passos, sendo inspirada na metodologia proposta pelo *Project Model Canvas*. Ele está descrita em BPMN nas Figuras 5.1 e 5.2

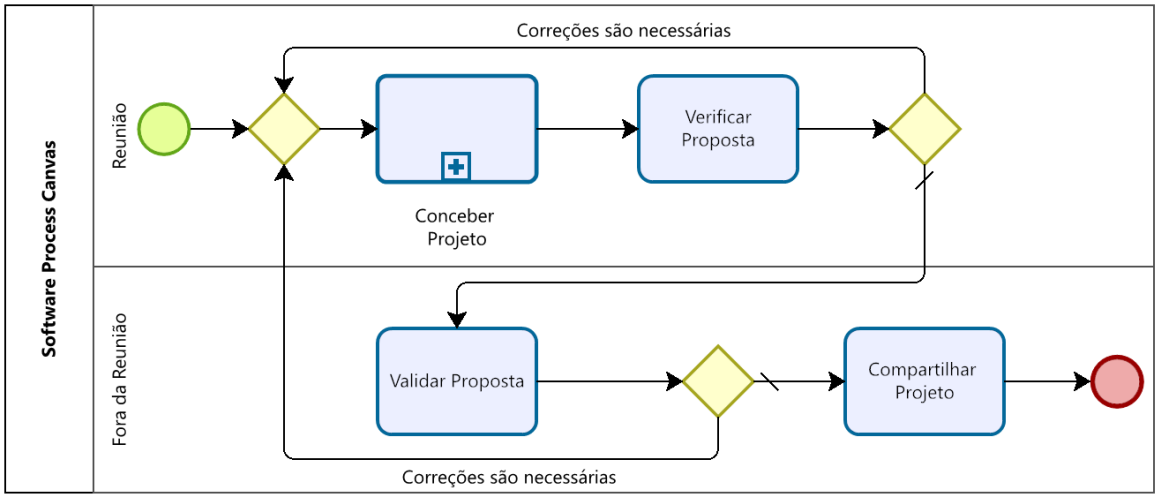


Figura 5.1: Descrição do processo de criação do Software Project Canvas. Fonte: imagem do autor

5.0.1 Conceber Projeto

A principal atividade criativa da metodologia é Conceber o Projeto. Isso é feito em uma, ou mais, reuniões facilitadas, em que o Canvas é usado como foco de colaboração.

Os passos dessa reunião são descritos na Figura 5.2, porém esse segundo processo deve ser entendido de forma genérica, já que não há proibição de se voltar a um passo anterior para melhorar, apagar ou propor novos itens.

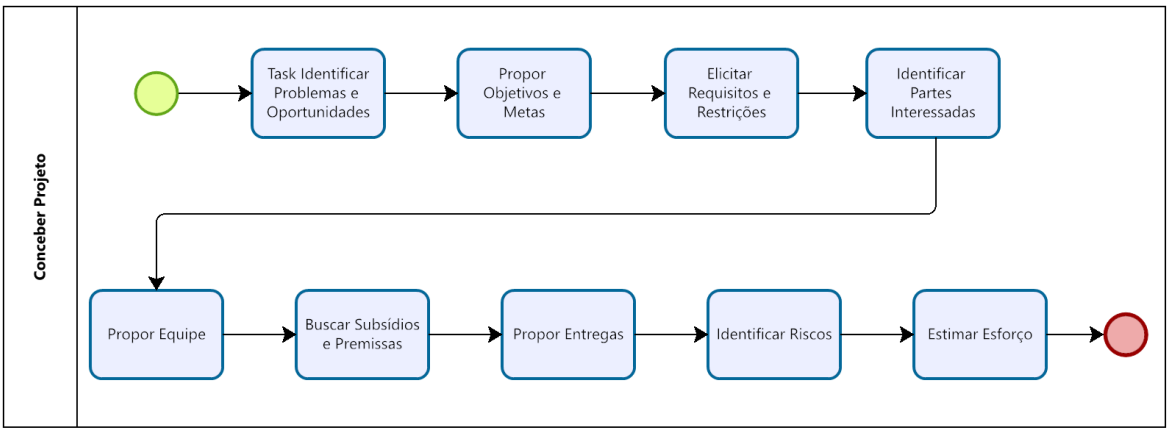


Figura 5.2: Descrição do sub-processo Conceber Projeto, do processo de criação do Software Project Canvas. Fonte: imagem do autor

Essas reuniões devem contar com representantes da equipe e das partes interessadas, incluindo o patrocinador, clientes, e membros do escritório de projetos ou assemelhados, se essa função existir na organização. É essencial que existam pessoas com a compreensão da gestão de projetos de desenvolvimento software para liderar a reunião, explicar conceitos e trazer sua experiência nas propostas e estimativas.

Durante a reunião, cada passo é focado em preencher um quadro, e para isso deve ser usada uma estratégia em diamante, onde em um primeiro momento deve ser usada uma prática de divergência, ou geração criativa de ideias, normalmente o *brainstorming* (Tracy, 2015) ou técnica derivada, ou assemelhada, e em um segundo momento uma técnica de convergência, seja ela de a busca de consenso, priorização ou votação, como por exemplo MoSCoW (Stapleton, 2003) ou *dot-voting* (Gray, Brown e Macanufo, 2010).

Os quadros devem ser inicialmente preenchidos sem limites, já que é necessário estimular a criatividade durante o momento de divergência. Já durante o momento onde se faz a convergência eles devem ser agrupados em ideias semelhantes e postos em ordem de prioridade, urgência ou importância. Além disso, é necessário garantir que todos os *post-its* estejam em um mesmo nível de abstração, se necessário agrupando vários em um só, ou separando um em vários. Em cada passo, então, são colocados, modificados e retirados *post-its* no canvas, até que os participantes estejam satisfeitos.

É importante notar que a metodologia faz, em um só passo, todo o levantamento de requisitos funcionais, não-funcionais, e restrições de alto nível. Isso foi decidido pelo fato de, ao usar outras metodologias, ser comum que um item desses tópicos seja proposto enquanto outro tópico é feito. Unificando esse momento, não é necessário parar para discutir questões de ordem da reunião.

É possível que durante um quadro, como por exemplo ‘´Premissas”, se descubra a necessidade de incluir um novo *post-it* em um quadro anterior, por exemplo, uma nova parte interessada. Isso é totalmente permitido, não sendo necessário refazer todo o trabalho do quadro anterior, porém devemos verificar com atenção esse novo item no próximo passo.

Ao longo da Concepção, é importante que os participantes da reunião estejam atentos às regras que serão usadas na verificação, principalmente no momento de convergência, para garantir desde início a qualidade do quadro.

É possível também usar *post-its* com cores e tamanhos diferentes para dar significados específicos. Por exemplo, podem ser usados *post-its* vermelhos para chamar atenção de itens que foram colocados em um quadro após eles já terem sido resolvidos em uma primeira passagem.

5.0.2 Concepção Iterativa e Incremental

É possível realizar a concepção do processo de forma iterativa e incremental, mesmo que em apenas uma seção. Para isso, se inicia cada iteração com um problema ou oportunidade, normalmente o mais importante, e se passa por todos os quadros atendendo apenas aquele problema, e mais nenhum outro. Ao terminar a iteração, deve ser analisado se é razoável fazer outra iteração ou o projeto já está definido.

Durante cada iteração pode ser necessário adaptar o resultado das iterações anteriores para atender os requisitos da nova iteração. Por isso é importante manter alguma memória ao final de cada iteração, por exemplo fotografando o quadro, ou o reproduzindo rapidamente em uma ferramenta digital, como Miro ou mesmo o Power Point.

Mesmo que a próxima iteração do projeto vá resolver apenas o primeiro problema, é interessante já começar com alguns problemas, e suas solução subsequente, determinados, levando a criação de um *road-map* do produto. Técnica similar é feita no Scrum, onde é mantido o Product Backlog e escolhido apenas um Sprint Backlog para realizar a cada iteração (Satpathy et al., 2016).

Se o quadro é mantido fixo na parede ao longo do projeto, pode ser revisado também a cada iteração.

5.0.3 Ordens Alternativas de Preenchimento

Apesar de ser comum planejar um projeto na ordem inicial Por que, O que e Quem, definindo o contexto para o qual as outras perguntas vão ser respondidas, é possível fazê-lo em outra ordem.

O Software Project Canvas não impõe, então, que seja usada apenas a ordem proposta, e ainda oferece canvas pré-prontos para duas ordens alternativas:

- iniciar com as Partes Interessadas, encontrar os Problemas e Benefícios que elas tem e só depois projetar o que será o projeto, e
- iniciar com o porquê, descobrir as partes interessadas e então passar para os requisitos.

Essas ordens alternativas representam situações diferentes na vida real.

Na nossa proposta inicial, apresentada na Figura 1.1 supomos que a organização tem problemas ou oportunidades, e a partir daí o projeto é criado e, então, olhando esse quadro, são determinadas todas as partes interessadas.

Na segunda proposta, primeiro foi identificado que algumas partes interessadas estão descontentes, ou precisam de algo. É provável que estas partes interessadas estejam inclusive presente na reunião, e começar por identificá-las pode ajudar a procurar os problemas e seguir com o projeto.

Finalmente, na terceira proposta, imaginamos que os problemas foram identificados, e logo depois identificamos quem está envolvido com esses problemas.

Outra situação comum é temos como demanda inicial o "O quê". Nesse caso, não é recomendável começar pelos requisitos, mas sim pelos Problemas ou pelas Partes interessadas, para garantir uma melhor elicitação dos verdadeiros requisitos do projeto.

5.0.4 Verificar Proposta

A segunda atividade do processo tem como objetivo verificar a correção do *Software Project Canvas* que será apresentado para validação na organização e com as outras partes interessadas.

Nessa fase, os participantes da reunião passam a se preocupar com a correção local e global do canvas, buscando garantir que ele apresenta um projeto coeso, racional e viável.

Para isso, além de verificações *ad-hoc*, é possível seguir um *check-list* apresentado a seguir.

- Verificações para os *post-its*
  - Todo *post-it* é claro e pode ser entendido por qualquer leitor ou parte interessada?
  - Todo *post-it* está no quadro a que deve pertencer?
- Verificações para os quadros
  - Os quadros estão organizados de forma que grupos de *post-its* representem ideias semelhantes e que a importância ou prioridade de cada *post-it* dentro do quadro ou assunto esteja representada?
  - Todos os *post-its* dentro de um quadro estão em um mesmo nível de abstração?
  - Existem, em um quadro, elementos que são um sub-grupo de outro elemento no quadro. Por exemplo, uma parte interessada pode ser "governo federal" e outra "Ministerio da Educação". Isso é realmente desejado?

(a) Forma alternativa onde o projeto se inicia com as partes interessadas. Fonte: imagem do autor

(b) Forma alternativa onde o projeto se inicia com os problemas e as partes interessadas. Fonte: imagem do autor.

Figura 5.3: Desenhos alternativos, que levam a processos alternativos para a fase de Conceber Projeto.

- Verificações para o canvas
  - Todo o canvas está no mesmo nível de abstração?
  - A linguagem é consistente entre os quadros e *post-its*?
  - Todo problema e oportunidade está atendido por meio de um objetivo ou meta?
  - Todo objetivo, e meta, busca atender um problema ou oportunidade?
  - Toda meta tem uma métrica?
  - Os requisitos funcionais e não-funcionais atendem o Por Que do projeto?
  - As partes interessadas estão envolvidas com algum por que ou algum requisito ou restrição do projeto?
  - Requisitos e entregas deixam as partes interessadas satisfeitas?
  - Os subsídios e premissas são suportados por alguma parte interessada?
  - A equipe consegue atender os requisitos e cumprir as entregas?
  - As entregas atendem o porquê do projeto?
  - Todo risco está associado a alguma premissa, entrega, requisito ou restrição?
  - Toda premissa tem um risco, ou o risco foi pelo menos avaliado e considerado nulo?
  - As estimativas são razoáveis para o projeto e deixam as partes interessadas satisfeitas?

5.0.5 Validação

Ao completar a verificação do canvas, é necessário validar o projeto com outras partes da organização, principalmente, com as partes interessadas interessadas que possuem, simultaneamente, poder, urgência e legitimidade sobre o projeto (Mitchell, Agle e Wood, 1997). Para isso, deve ser feito um documento inicial com a proposta do projeto, que pode ser uma cópia do próprio canvas.

Fora da reunião, então, o projeto é validado e possíveis *feedbacks* podem ser trazidos ao grupo original para melhorar, refinar, modificar ou mesmo abandonar o projeto, caso tenham sido encontradas falhas irremediáveis.

Este método não define uma forma fixa de se fazer a validação. Inclusive, grupos diferentes em uma mesma organização podem usar formas diferentes de validação. Podem ser usados reuniões, workshops, questionários, entrevistas, etc.

O resultado dessa validação também não é definido em formato, e depende muito do tipo e da qualidade do *feedback* recebido. A partir do resultado, o grupo responsável por lançar o projeto pode decidir não fazer nada, fazer pequenos ajustes ou voltar a fase de concepção.

5.0.6 Compartilhar Projeto

Na última fase do planejamento o projeto é compartilhado com a organização. Novamente, a forma de fazer isso depende do projeto, da organização e da parte interessadas sendo atingida. Pode ser necessário criar um plano de projeto formal para aprovação final em uma reunião anual, ou criar documentos mais ligados a métodos ágeis, como uma visão do projeto ou um *backlog* do produto para equipe de desenvolvimento, ou uma notícia para o jornal de divulgação interna.

6 Conclusão

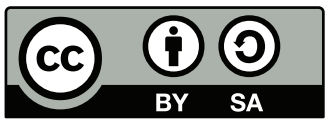
Este artigo apresenta o *Software Project Canvas* um novo canvas para o planejamento ágil de um projeto de software. Ele foi criado para ser usado em substituição a outros canvas destinados ao planejamento inicial de projetos em geral, que são menos adaptados a linguagem e a prática dos desenvolvedores de software. Para a sua criação foram observados 6 dificuldades encontradas ao ensinar ou utilizar o *Project Model Canvas*, que serviu de fonte de inspiração. As adaptações feitas visam facilitar a explicação e o uso, além de evitar perda de tempo com discussões que não adiantam a definição do projeto.

No futuro será feita uma avaliação do uso do canvas em sala de aula e em projetos de software, a fim de validar as modificações feitas.

Agradecimentos

Agradeço a Yasmin pelo design final do *Software Model Canvas* e aos alunos, como Eduardo Mangeli e Marcus Parreira, que fizeram críticas e sugestões.

Licença



Baseado no trabalho disponível em:

`https://github.com/xexeo/Software-Project-Canvas`.

Podem estar disponíveis autorizações adicionais às concedidas no âmbito desta licença com `email:xexeo@ufrj.br`.

O texto completo da licença pode ser obtido em: `https://creativecommons.org/licenses/by-sa/4.0/`

Bibliografia

Boehm, Barry, Chris Abts e and Sunita Devnani-Chulani Brad Clark (1997). *COCOMO II Model Definition Manual version 1.4* (ver p. 15).

Cockburn, Alistair (jan. de 2000). *Writing Effective Use Cases*. Addison-Wesley (ver p. 20).

Cohn, Mike (2004). *User Stories Applied: For Agile Software Development*. USA: Addison Wesley Longman Publishing Co., Inc. ISBN: 0321205685 (ver p. 20).

– (2005). *Agile Estimating and Planning*. Prentice Hall (ver pp. 15, 16, 25).

Constantine, Larry L. (out. de 1993). "Work organization". *Communications of the ACM* 36.10, pp. 35–43. DOI: 10.1145/163430.163435. URL: https://doi.org/10.1145/163430.163435 (ver p. 15).

Dalkey, Norman Crolee (out. de 1966). *Delphi*. Technical Report P-3704. Santa Monica, California: RAND Corporation. URL: https://www.rand.org/pubs/papers/P3704.html (acesso em 16/01/2020) (ver p. 25).

digital.ai (2022). *15th State of Agile Report. Agile adoption accelertes across the enterprise*. White Paper (ver pp. 13, 23).

Finocchio Jr., José (2013). *Project Model Canvas: gerenciamento de projetos sem burocracia*. Rio de Janeiro: Campus, Elsevier (ver pp. 5, 9, 11, 14, 15, 19, 22, 24, 25).

Gane, Chris P. e Trish Sarson (1979). *Structured Systems Analysis: Tools and Techniques*. 1ª ed. Prentice Hall Professional Technical Reference. ISBN: 0138545472 (ver pp. 19, 20).

Gray, Dave, Sunni Brown e James Macanufo (2010). *Gamestorming*. Sebastopol, CA: O'Reilly (ver pp. 18, 22, 28).

Gross, John M. e Kenneth R. McInnis (2003). *Kanban Made Simple: Demystifying and Applying Toyota's Legendary Manufacturing Process*. New York: AMACOM (ver p. 14).

IEEE (2017). *ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary*. Standard ISO/IEC/IEEE 24765:2017(E). ISO/IEC/IEEE, pp. 1–541. DOI: 10.1109/IEEESTD.2017.8016712 (ver p. 19).

IEEE Computer Society (17 de jan. de 2014). *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. Ed. por Pierre Bourque e Richard E. Fairley. 3 edition. IEEE Computer Society Press. 346 pp. ISBN: 978-0-7695-5166-1 (ver p. 13).

IFPUG (2010). *Function Point Counting Practices Manual Release 4.3.1*. Standard. International Function Point Users Group (ver p. 25).

– (2012). *The IFPUG Guide to IT and Software Measurement*. Auerbach Publications (ver p. 16).

IIBA (2011). *Um guia para o Corpo de Conhecimento de Análise de Negócios (Guia BABOK) Version 2.0*. Ontário, Canadá: International Institute of Business Analysis (ver p. 20).

Jacobson, Ivar, Harold "Bud" Lawson e Pan-Wei Ng (19 de jul. de 2019). *The Essentials of Modern Software Engineering: Free the Practices from the Method Prisons!* ACM Books. 400 pp. ISBN: 978-1-947487-24-6 (ver pp. 14, 22).

Jacobson, Ivar, Ian Spence e Kurt Bittner (dez. de 2011). *Use-Case 2.0: The Guide to Succeeding with Use Cases* (ver pp. 20, 23).

Kerzner, Harold (2017). *Project Management. A system approach to planning, scheduling and controlling*. 12ª ed. Hoboken, New Jersey: John Wiley & Sons (ver pp. 13, 17, 24).

McMenamin, Stephen M. e John F. Palmer (1984). *Essential Systems Analysis*. USA: Yourdon Press. ISBN: 0917072308 (ver p. 20).

Mitchell, Ronald K., Bradley R. Agle e Donna J. Wood (1997). "Toward a Theory of Stakeholder Identification and Salience: Defining the Principle of Who and What Really Counts". *The Academy of Management Review* 22.4, pp. 853–886. ISSN: 0363-7425. DOI: 10.2307/259247. URL: %5Curl%7Bhttp://www.jstor.org/stable/259247%7D (acesso em 15/01/2017) (ver p. 32).

Nakagawa, Marcelo (2012). *Ferramenta: 5W2H – Plano De Ação Estratégia e Gestão Para Empreendedores*. URL: https://www.sebrae.com.br/Sebrae/Portal%5C%20Sebrae/Anexos/5W2H.pdf (acesso em 20/08/2021) (ver p. 17).

Naur, Peter e Brian Randell, ed. (jan. de 1969). *Software Engineering. Report on a conference sponsored by the NATO SCIENCE COMMITTEEGarmisch, Germany, 7th to 11th October 1968* (ver p. 13).

OMG (out. de 2018). *Essence – Kernel and Language for Software Engineering Methods: Version 1.2*. specification. Object Management Group. URL: %5Curl%7Bhttps://www.omg.org/spec/Essence/%7D (ver p. 17).

Osterwalder, A., Y. Pigneur e T. Clark (2010). *Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers*. Strategyzer series. Wiley (ver pp. 5, 11, 12).

PMI (2017). *Um Guia do Conhecimento em Gerenciamento de Projetos (Guia PMBOK®)*. Sexta Edição. Newtown Square, PA: Project Management Institute. ISBN: 9788502223745 (ver pp. 13, 21, 22, 24, 25).

Pressman, Roger S. e Bruce Maxim (2019). *Software Engineering: A Practitioner's Approach*. 9ª ed. New York, NY: McGraw-Hill. 976 pp. ISBN: 978-1-260-54800-6 (ver pp. 13–16, 23).

Rubin, Kenneth S. (2013). *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Pearson Education (ver p. 23).

Ruble, David A. (1997). *Practical Analysis & Design for Client/Server and GUI Systems*. Upper Saddle River: Yourdon Press (ver pp. 16, 19).

Satpathy, Tridibesh et al. (2016). *A Guide to the SCRUM BODY OF KNOWLEDGE (SBOK™ Guide). A Comprehensive Guide to Deliver Projects using Scrum*. 2016ª ed. SCRUMStudy, VMEdu, Inc (ver pp. 14, 29).

Sommerville, Ian (3 de abr. de 2015). *Software Engineering*. 10 edition. Boston: Pearson. 816 pp. ISBN: 978-0-13-394303-0 (ver pp. 5, 21).

Stapleton, Jennifer, ed. (2003). *DSDM: Business Focused Develpment*. 2ª ed. London: DSDM Consortium, Addison Wesley (ver pp. 18, 20, 28).

The Standish Group (2015). *Chaos Report 2015*. Report. Standish Group (ver p. 14).

Thomsett, Rob (2002). *Radical Project Management*. Prentice Hall PTR. ISBN: 013458886X (ver p. 24).

Tracy, Brian (2015). *Creativity & Problem Solving*. New York: AMACOM (ver pp. 18, 22, 28).