Universidade de São Paulo
Instituto de Matemática e Estatística
Bachalerado em Ciência da Computação

Caio Teixeira da Quinta
Eugênio Augusto Jimenes

Plataforma Web para divulgação e centralização de eventos aplicando conceitos de Métodos Ágeis e Lean Startup

São Paulo
Dezembro de 2016

Plataforma Web para divulgação e centralização de eventos aplicando conceitos de Métodos Ágeis e Lean Startup

Monografia final da disciplina
MAC0499 – Trabalho de Formatura Supervisionado.

Supervisor: Prof. Dr. Dr. Alfredo Goldman vel Lejbman
Cosupervisor: Jorge Melegati

São Paulo
Dezembro de 2016

Agradecimentos

Resumo

<em desenvolvimento>

É notável que a Cidade Universitária possui uma imensa diversidade acadêmica e cultural que se manifesta em uma variedade de eventos sendo promovidos e organizados pela Universidade de São Paulo além de iniciativas da própria comunidade para ocupar o espaço público.

Levando em consideração a quantidade de eventos acontecendo de forma simultânea, a extensão física do campus, a pulverização dos eventos espalhando-se por toda sua extensão e a quantidade de informações dispersas entre as várias redes de comunicação oficiais ou não muitas vezes não tomamos conhecimento a tempo de alguma oportunidade que poderia ser interessante.

Foi pensando na centralização e divulgação de eventos que foi proposto criar o USP Eventos, uma plataforma web colaborativa na qual qualquer membro frequentador no campus pode se informar do que acontece no mesmo além de organizar seu próprio evento.

Tomando como base uma metodologia de projeto baseada nos conceitos de Métodos Agéis e Lean startupfoi proposto desenvolver um sistema em Ruby on Rails utilizando 3 ciclos de Build Measure Learn sempre pautando o desenvolvimento o sistema segundo oaprendizado obtido através dos comentários e críticas dos usuários.

Palavras-chave: eventos, métodos ágeis, lean startup, desenvolvimento web.

Abstract

Elemento obrigatório, elaborado com as mesmas características do resumo em língua portuguesa.

Keywords: keyword1, keyword2, keyword3.

Contents

Lista de Abreviaturas

MVP	Produto Mínimo Viável (*Minimum Viable Product*)
CoC	Convençaõ sobre Configuração (*Convention over Configuration*)
DRY	Não se repita (*Don't Repeat yourself*)
ORM	Mapeamnto Objeto Relacional(*Object Relational Mapping*)

Chapter 1

Introdução

1.1 Motivação e Objetivos

É notável que a Cidade Universitária possui uma imensa diversidade acadêmica e cultural que se manifesta em uma variedade de eventos sendo promovidos e organizados pela Universidade de São Paulo além de iniciativas da própria comunidade para ocupar o espaço público.

Levando em consideração a quantidade de eventos acontecendo de forma simultânea, a extensão física do campus, a pulverização dos eventos espalhando-se por toda sua extensão e a quantidade de informações dispersas entre as várias redes de comunicação oficiais ou não muitas vezes não tomamos conhecimento a tempo de alguma oportunidade que poderia ser interessante.

Foi proposto criar um Sistema Web colaborativo para centralizar e divulgar os eventos que são organizados pela Universidade de São Paulo assim como pela comunidade que frequenta o campus e para isso adotamos uma metodologia de projeto seguindo os prececeitos de Lean Startup e Métodos Ágeis.

A metodologia Lean Startup tem como foco evitar o desperdício de tempo e recursos focando em um rápido aprendizado dos interesses do público alvo para refinamento do produto portanto foi adotada no desenvolvimento do USP Eventos pois dada sua natureza colaborativa do projeto era vital podermos coletar informações sobre o nosso público, a comunidade usuária do campus da Cidade Universitária, a fim de que o projeto estivesse alinhado com seus interesses.

Para atingir os objetivos propostos pela metodologia Lean Startup aplicamos conceitos de Métodos Ágeis tais como testes, desenvolvimento interativo, incremental e contínuo.

1.2 Capítulos

Chapter 2

Lean Startup

2.1 Lean Startup: O que é

2.1.1 Startup: uma definição

Através da popularização da internet e dos computadores pessoais nos anos 90, nos Estados Unidos, o termo startup foi generalizado para classificar pequenas empresas com propostas inovadoras, sejam por atuarem com as novas tecnologias que surgiram para o grande mercado na época, como, as chamadas empresas online ou empresas "ponto com" ou pelo seu novo modo de organização e processo de produção. Apesar de não ser um rótulo exclusivo para mercado de Tecnologia da Informação, alguns locais e atividades foram particularmente associados à classificação devido a revolução tecnológica promovida pela bolha da internet, como no Vale do Silício, área norte do estado da Califórnia, EUA, conhecida até hoje por ser um ecossistema constituído de empresas inovadoras.

Com o passar dos anos e com o impacto da internet no mercado global o termo amadureceu para empresa, grupo ou organização que busca um modelo de negócios escalável, geralmente envolvida em implementações de processos inovadores de desenvolvimento e pesquisa de mercado-alvo. Grosseiramente uma startup surge sob uma ideia ou foco que pode ou não dar certo mas consciente desta instabilidade.

Para Steve Blank, academico e empreendedor do Vale do Silício, uma startup vai de fracasso à fracasso com objetivo de aprender com cada falha e assim definir o que não funciona no processo na qual a empresa esta engajada. Por isso é considerado um modelo de negocio escalavel: deve ser flexivel parente a constante reação do mercado e da própria produção.

2.1.2 Lean Startup

Lean Startup (Startup Enxuta) é um conceito introduzido em 2008 por Eric Ries, empreendedor de diversas startups do Vale do Silício. Trata-se de uma metodologia de negócios derivada da combinação de outros padrões de desenvolvimento como *Minimal Viable Product* (produto viavél mínimo), *Customer Development* (desenvolvimento voltado ao cliente) e *Agile software development* (desenvolvimento ágil de software ou Método Ágil).

Ries propõe que é possível encurtar os ciclos de implementação de um produto (ou solução) adotando uma combinação de testes, hipóteses de negócio e degustações por parte do público-alvo. Através do lançamento periódico de cada versão do produto é possível avaliar não apenas quesitos técnicos, mas como também a reação do mercado. Consequentemente o retorno de cada interação afeta o planejamento do produto e suas futuras versões. Essa

economia em cada ciclo provém de validated learning (aprendizagem de validação), ou seja, toda ideia, funcionalidade ou vertente de produto deve ser primeiramente experimentada e testada. Isso evita desperdício de desenvolvimento, abre oportunidades para adaptações e alterações de projeto em casos de falha ou rejeição do cliente.

Versões simples do produto sob cada ciclo de avaliação é uma estratégia derivada do padrão Minimal Viable Product, proposta em 1996 por Frank Robinson, CEO (Chief Executive Officer) da empresa SyncDev, porém popularizado anos depois por Steve Blank.

Robinson propõe, basicamente, o lançamento de uma versão o mais simples possível de modo à antecipar a análise de mercado e assim minimizar o risco de retorno por parte da empresa. A popularidade Steve Blank foi em adaptar a estratégia incluindo o lado do cliente na balança, o que ele chama de Customer Development. Blank vai além de apenas minimizar o risco de retorno: busca compreender as necessidades do cliente. Lean

Startup aprimora ainda mais o conceito para avalições sob cada interação e assim maximizar o aprendizado e evolução do projeto, alinhado com o desejo do mercado, o chamado ciclo *Build-Measure-Learn* (Construir-medir-aprender).

2.2 Produto Mínimo Viável (MVP)

Uma definição de Produto Mínimo Viável por Eric Ries: *"A Minimum Viable Product is that version of a new product which allows a team to collect the maximum amount of validated learning about customers with the least effort."*.

A idéia central do conceito de MVP é maximizar a validação de aprendizado sobre um produto utilizando o menor esforço possível. Como na maioria das startups os recursos financeiros e humanos são bastante escassos o tempo de validação do produto e determinação do interesse do público é um fator decisivo para o sucesso da mesma.

Um MVP deve possuir 3 características:

1. Ter valor suficiente para que uma pessoa queira utilizá-lo ou comprá-lo.

2. Possui suficientes benefícios para reter os chamados usuários pioneiros.

3. Ser capaz de prover um ciclo de feedback suficiente para guiar o desenvolvimento.

Durante a concepção do projeto são definidas algumas hipóteses sobre o produto e na etapa do MVP é definido então qual será o seu núcleo, ou seja, quais funcionalidades ou estratégias queremos testar de modo que possamos validar as nossas hipóteses iniciais e obter o máximo de aprendizado possível.

Além da validação de aprendizado outra vantagem significativa provida pelo MVP é evitar desperdícios de tempo e recursos como por exemplo através do aprendizado resultante do MVP é possível evitar que se desperdice mais tempo em recurso em uma funcionalidade que a princípio parecesse útil e necessária mas que foi constatado através de uma validação bem construída que a mesma não era interessante para os usuários podendo assim ser revista antes que fossem gastas mais horas de desenvolvimento.

Os termos de "mínimo" e "máximo" frequentemente se mostram vagos na documentação do que é um MVP, citando Eric Ries: *"[...]the definition's use of the words maximum and minimum means it is decidedly not formulaic. It requires judgment to figure out, for any given context, what MVP makes sense. "*, levando em consideração é importante frisar que um MVP não é necessariamente um produto completo com as funcionalidades mínimas e sim um conjunto de características mínimas que configuram o serviço ou produto que está

sendo oferecido. Dessa forma um MVP pode ser apenas um protótipo, um produto completo ou mesmo apenas um *mock-up* do que será oferecido na versão completa.

Alguns tipos de MVP são:

- *Vídeo Explicativo:* Um vídeo curto contendo uma explicação clara do que o seu produto faz e porque as pessoas deveriam utilizá-lo. Esse é o caso do dropbox que fez um video com cerca de 5 minutos explicando o que era o seu serviço.

- *Landing Page:* Criar uma página inicial contendo uma explicação detalhada do que é o produto que você irá oferecer juntamente com um formulário de contato. Através de uma configuração simples pelo Google Analytics é possível manter um registro de conversões (no caso cadastros do formulario) a fim de medir o interesse das pessoas no seu produto.

- *MVP "Mago de OZ":* A idéia é criar uma página visutalmente compelta que funcione como se fosse o produto final mas que por trás existe alguém manualmente executando as tarefas. Esse foi o caso da Zappos hoje a maior vendedora de sapatos dos Estados Unidos.

- *MVP com Consierge:* Ao invés de prover um produto você realiza manualmente o serviço realizando exatamente os mesmos passos que o usuário teria utilizando seu serviço. É um método não escalável e lento para executar pois requer que você esteja em contato direto com o cliente e realize as tarefas manualmente porém isso permite um rápido aprendizado sobre o produto e o cliente.
  A empresa Food on the Table ajuda seus consumidores a criarem listas de compras, acharem receitas e conseguirem descontos nos ingredientes em seus supermercados favoritos, inicialmente seus fundadores encontraram uma senhora interessada no serviço e por 10 doláres/semana eles mantinham as listas de compra e procurava, por descontos nos supermercados em que ela fazia compras.

2.3 O ciclo de Build-Measure-Learn

Um ciclo de build measure learn é uma abordagem de desenvolvimento do produto que aprimora o tradicional modelo de desenvolvimento "Cascata" utilizado de forma abrangente durante o século XX.

2.3.1 O modelo Cascata

O nome "Cascata" é bastante literal e tem origem na própria estrutura do método que seguia uma série de passos de forma sequencial de maneira bastante direta, como se fosse uma cascata.

O método consiste em um desenvolvimento sequencial e não-interativo inicializando com a parte de análise de requisitos e de forma subsquente temos as etapas de design de projeto, implementação, verificação e manutenção.

Uma rápida explicação dos passos:

- *Requerimentos:* Realizar a análise de requisitos do projeto.

- *Design de Projeto:* Focando na estrutura de dados, arquitetura do software, detalhes procedais e caracterização das interfaces é formulado um documento de forma a apresentar os requerimentos de uma forma que possa ser interpretado pelos programadores.

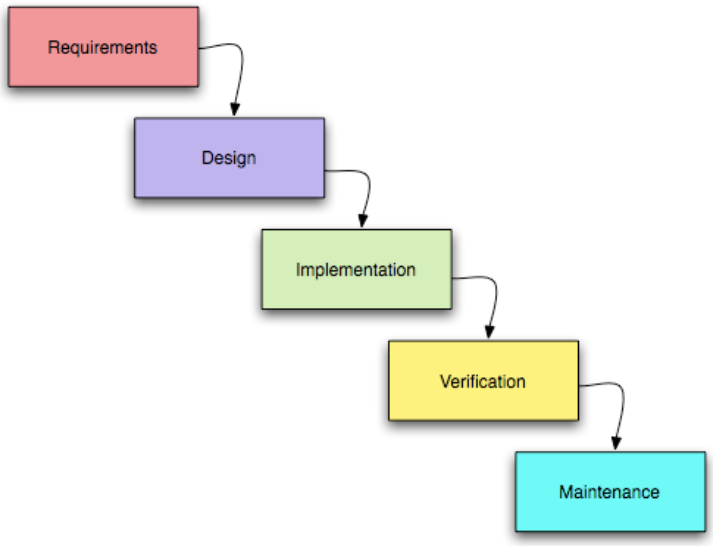

Figure 2.1: *O Modelo Cascata.*

- *Implementação:* Etapa da codificação do projeto propriamente dita.
- *Verificação:* Etapa para teste do produtos visando eliminar qualquer bug que possa ter passado despercebido refinar a lógica interna do sofware caso necessário.
- *Manutenção:* Etapa para instalação do sistema no cliente, configuração de servidores, etc.

Uma das grandes das críticas dessa abordagem é que dificilmente um desenvolvimento de software segue todas as etapas da forma como o modelo propõe e nem sempre o cliente sabe definir bem os requisitos resultando em tempo e desenvolvimento desperdiçado em funcionalidades que não resolvem o problema do cliente além de mudanças tardias no escopo do projeto que, além de encarecer o custo total, poderiam ter sido evitadas e contornadas de maneira mais satisfatória em um modelo com uma maior interação entre cliente e desenvolvedor.

Ao contrário do modelo cascasta no qual o contato do cliente com o software se dá apenas no fim do processo de desenvolvimento na etapa de testes o método de Build Measure Learn privilegia desde o início uma interação contínua com o cliente.

2.3.2 Build-Measure-Learn (Construir-medir-aprender)

O ciclo de Build Measure Learn é uma das idéias fundamentais do Lean Startup e consiste de um ciclo de feedback rápido cujo principal objetivo é eliminar as incertezas sobre as hipóteses do produto através de um aprendizado rápido sobre o mesmo assim minimizando os riscos e custos desnecessários que persistir em uma idéia equivocada poderiam causar.

Além das 3 idéias fases do ciclo vamos incluir no diagrama outras 3 menores (*ideas, data e code*) para ilustrar melhor o objetivo de cada uma das etapas.

- *Build:* Inicialmente temos algumas idéias que foram definidas a partir das hipóteses do produto que precisam ser implementadas (*code*) no MVP.
- *Measure:* Implementado o MVP coletamos dados(*data*) de uso e seguindo algumas métricas já definidas avaliamos seu desempenho. Todo o ciclo é baseado na idéia de

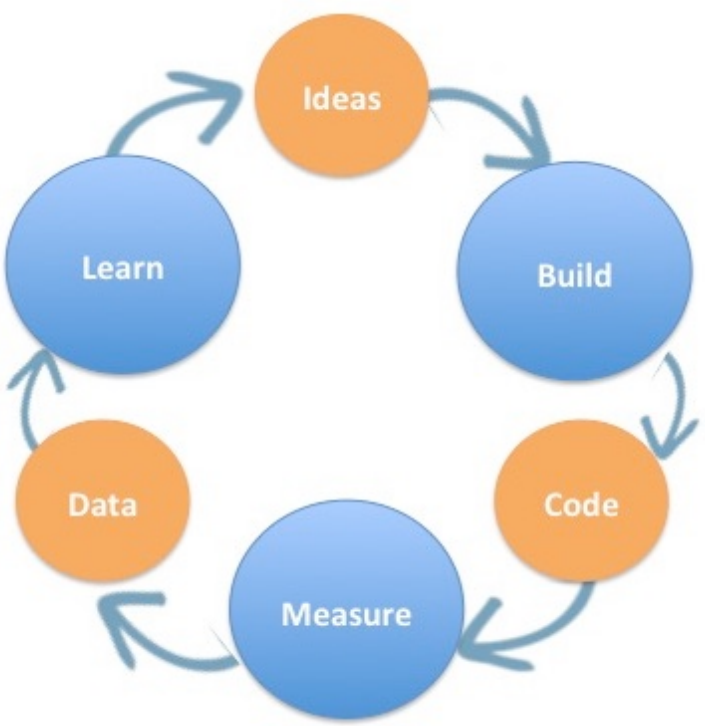

Figure 2.2: *O ciclo de Build Measure Learn.*

aprendizado rápido para aprender o máximo possível sobre como os usuários reagem as idéias implementadas.

- *Learn:* A partir então da análise dos dados coletados podemos inferir sobre continuar sobre como continuar o desenvolvimento e o que funcionou ou não. Feito isso implementamos as novas idéias geradas e começamos novamente o ciclo para validá-las.

As etapas do ciclo não precisam necessariamente ocorrer em ordem podendo se sobrepor ou mesmo serem unidas dependendo de como for o ciclo de desenvolvimento.

As alterações de software precisam ser feitas de maneira rápida de modo a testar o mais rápido possível novas idéias portanto é importante ressaltar que as funcionalidades devem se manter simples e diretas pois o foco é o aprendizado gerado e não desenvolver um software ou um protótipo completo pois dessa maneira é possível evitar que sejam desperdiçados recursos em uma funcionalidade que pode não ser bem recebida e não ter continuidade.

Apesar do desenvolvimento ser focado na simplicidade isso não quer dizer produzir um produto de baixa qualidade. Existem ferramentas que auxiliam na integração contínua do software e é comum realizar testes automatizados garantindo que seja possível manter um desenvolvimento consistente e confiável sem comprometer o tempo de execução pois um produto de baixa qualidade ou que possua bugs pode resultar em uma taxa de rejeição inesperada devido a problemas de imeplementação fazendo com que as conclusões do ciclo possam ser enviesadas.

Cada ciclo de build measure learn testa idéias específicas portanto é comum realizar diversos ciclos implementando e testando idéias diferentes

Dada sua natureza interativa e o desenvolvimento incremental que o o ciclo de Build Measure Learn propõe o o projeto final costuma estar mais refinado e alinhado com as expectativas dos clientes em contraponto aos problemas apresentados pelo modelo em Cascata sendo assim uma evolução bastante interessante do mesmo.

Chapter 3

Tecnologias

3.1 Ruby on Rails

3.1.1 Ruby

A criação da linguagem Ruby data de 1995 no Japão por Yukihiro "Matz" Matsumoto sob forte influência de outras linguagens como Perl, SmallTalk, Eiffel, Ada e Lisp e tinha como objetivo equilibrar programação funcional, imperativa e orientação a objetos, segundo o próprio Matz: "Eu queria uma linguagem interpretada que fosse mais poderosa do que Perl e mais orientada as objetos do que Python2.".

- Flexibilidade:

  Ruby cresceu devido a sua grande flexibilidade sendo possível alterar, remover ou acrescentar partes da linguagem à vontade, imagine o seguinte exemplo: um usuário prefere utilizar a palavra plus ao invés do operador matemático(+), ele poderia então adicionar esse método à classe natica do Ruby Numeric pois em ruby os operadores matemáticos são considerados açúcares sintáticos.
  Exemplo:

```
1 class Numeric
2       def plus(x)
3           self.+(x)
4       end
5   end
6
7   y = 5.plus 6
8   # y agora é igual a 11
```

- Closures:

  Em ruby o closures são chamadas de blocos e são funções que podem ser tratadas como uma variável, isso quer dizer que podem ser pasasdas como argumentos de métodos, serem atribuidas a outras variáveis, etc.

  As closures armazenam os valores das variávies que estavam no escopo quando a função foi denifida e são capazes de acessar tais variávies mesmo que sejam executadas em um escopo diferente.
  Exemplo:

```
1 search_engines =
2   %w[Google Yahoo MSN].map do |engine|
```

```
3     "http://www." + engine.downcase + ".com"
4   end
```

- Módulos:

  Diferente de outras linguagens orientadas a objetos o Ruby suporta apenas herança simples de forma proposital porém existe o conceito de módulos que são coleções de métodos que podem ser adcionadas à uma classe por meio de um mixin. As classes podem fazer o mixin de um método e receber todos os métodos dele diretament. Exemplo:

```
1 class MyArray
2       include Enumerable
3   end
```

3.1.2 Rails

David Heinemeier Hansson extraiu o Ruby on Rails do seu próprio trabalho realizado na empresa Basecamp e foi lançado de maneira open-source pela primeira vez em 2004, inicialmente ele estava desenvolvendo seu projeto utilizando PHP porém apesar de conseguir desenvolver de maneira bastante ágil a repetição de código

Rails é um framework web escrito em Ruby implementado seguindo o padrão mvc (model-view-controller) totalmente server-side sendo considerado portanto um framework back-end provendo uma estruturas para banco de dados, web service e web pages além de encorajar padrões de engenharia de software já consagrados tais como:

- *Convention over Configuration* (CoC):

  Apesar de algumas críticas por adicionar obscuridade pois não existe uma forma clara de verificar o comportamento esperado ruby adota alguns padrões de configuração visando padronizar o código e tirar do desenvoveldor a decisão de como usar o framework sem que isso tire sua flexibilidade. Por exemplo se existir um objeto chamado User então sua tabela por convenção se chamará users e correspondente controller será UsersController (no plural) pois esse é padrão definido para o framework e de outra forma o seu controller ou database não estaria associado ao modelo User.

- *Don't Repeat yourself* (DRY):

  Visando diminuir a repetição de informações de qualquer tipo de todas as formas. Em rails frequentemente ao utilizar o comportamento padrão vai parecer sem explicação já que sua implementação está escondida dentro do código do próprio Rails porém evitando que dessa forma o propgramador tenha que reescrever várias vezes um mesmo código para determinado comportamento.

- *Active Record Pattern*:

  O padrão Active Record sugere uma interface específica para acessar objetos em um banco de dados relacional contendo funções tais como INSERT, UPDATE, DELETE, etc.

  Uma tabela ou view será associada a uma classe então uma instância de objeto estará associada a uma única entrada na respectiva tabela.

  Em Rails a biblioteca ActiveRecord imeplementa o padrão ORM e além disso acrescenta herança e associações resolvendo dois problemas substanciais do padrão.

ActiveRecord é o "model" padrão do compomente MVC porém é possível trocá-lo por outra implementação do framework Rails caso o desenvolvedor prefira.

Em um sentido mais amplo Rails é mais que uma biblioteca de Software ou API: é um projeto central de uma vasta comunidade que produz plugins para facilitar e construir projetos complexos de web site. Foi graças a essa comunidade que compartilham ferramentas e contribuem para o projeto Rails criando bibliotecas open source (chamads de Ruby Gems ou apenas Gems) que o o uso de rails tem crescidos de forma significante nos últimos anos ganhando fama entre startups devido a sua facilidade e desevolvimento rápido que permite criar sistemas complexos e testar idéias rapidamente.

3.2 Heroku

TODO

3.3 Travis CI

TODO

3.4 Google Analytics

TODO

Chapter 4

Conclusões

Texto texto[1].

[1]Exemplo de referência para página Web: www.vision.ime.usp.br/~jmena/stuff/tese-exemplo

Appendix A

Título do apêndice

Texto texto.