UnB - UNIVERSIDADE DE BRASÍLIA

FGA - FACULDADE UNB GAMA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA BIOMÉDICA

OPEN GAIT ANALYTICS - IMPLEMENTANDO UM SOFTWARE COMO SERVIÇO PARA ANÁLISE E SIMULAÇÃO DE MARCHA HUMANA

Roberto Aguiar Lima

ORIENTADORA: Dra. Lourdes Mattos Brasil

COORIENTADORA: Dra. VERA REGINA DA SILVA MARÃES

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA BIOMÉDICA

PUBLICAÇÃO: NUMERAÇÃO / 2015

BRASÍLIA/DF : Setembro– 2015

UnB - UNIVERSIDADE DE BRASÍLIA

FGA - FACULDADE UNB GAMA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA BIOMÉDICA

OPEN GAIT ANALYTICS - IMPLEMENTANDO UM SOFTWARE COMO SERVIÇO PARA ANÁLISE E SIMULAÇÃO DE MARCHA HUMANA

Roberto Aguiar Lima

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA BIOMÉDICA DA FACULDADE GAMA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA BIOMÉDICA.

APROVADO POR:

Prof. Dra. Lourdes Mattos Brasil
(Orientadora)

Prof. Dra. VERA REGINA DA SILVA MARÃES
(Coorientadora)

Prof. Dr(a).
(Examinador Externo)

BRASÍLIA/DF , 01 DE Setembro DE 2015

FICHA CATALOGRÁFICA

Roberto Aguiar Lima
OPEN GAIT ANALYTICS - IMPLEMENTANDO UM SOFTWARE COMO SERVIÇO PARA ANÁLISE E SIMULAÇÃO DE MARCHA HUMANA, [Distrito Federal] 2015.
NUMERAÇÃO . 38 p., 210 x 297 mm (FGA/UnB Gama, Mestre, Engenharia Biomédica, 2015). Dissertação de Mestrado - Universidade de Brasília. Faculdade Gama. Programa de Pós- Graduação em Engenharia Biomédica.
1. análise de marchar. 2. aprendizado de máquina
3. joelho. 4. simulação
I. FGA UnB Gama/ UnB. II.*OPEN GAIT ANALYTICS* - IMPLEMENTANDO UM SOFTWARE COMO SERVIÇO PARA ANÁLISE E SIMULAÇÃO DE MARCHA HUMANA
CDU: Nº da CDU (biblioteca)

REFERÊNCIA BIBLIOGRÁFICA

LIMA, R. A. (ANO). TÍTULO. Dissertação de Mestrado em Engenharia Biomédica, Publicação NO./ANO, Programa de Pós-Graduação em Engenharia Biomédica, Faculdade Gama, Universidade de Brasília, Brasília, DF, 38 p.

CESSÃO DE DIREITOS

AUTOR: Roberto Aguiar Lima
TÍTULO: *OPEN GAIT ANALYTICS* - IMPLEMENTANDO UM SOFTWARE COMO SERVIÇO PARA ANÁLISE E SIMULAÇÃO DE MARCHA HUMANA

GRAU: Mestre
ANO: 2015

2015.
ENDEREÇO.
CEP: ..., Brasília, DF – Brasil

DEDICATÓRIA

Para ..., com amor.

AGRADECIMENTOS

...

O Mestre na arte da vida faz pouca distinção entre o seu trabalho e o seu lazer, entre sua mente e seu corpo, entre sua educação e sua recreação. Ele simplesmente persegue sua visão de excelência em tudo o que faz, deixando para os outros a decisão de saber se está trabalhando ou se divertindo. Ele acha que está sempre fazendo as duas coisas simultaneamente.

Texto Budista

RESUMO

OPEN GAIT ANALYTICS - IMPLEMENTANDO UM SOFTWARE COMO SERVIÇO PARA ANÁLISE E SIMULAÇÃO DE MARCHA HUMANA

Autor: Roberto Aguiar Lima
Orientadora: Profa. Dra. Lourdes Mattos Brasil
Coorientadora: Dra. VERA REGINA DA SILVA MARÃES
Programa de Pós-Graduação em Engenharia Biomédica
BRASÍLIA/DF 2015

Texto corrido sem parágrafo. 1 página.

Palavras-chaves: análise de marchar, aprendizado de máquina, joelho, simulação.

ABSTRACT

OPEN GAIT ANALYTICS - IMPLEMENTING A SOFTWARE AS A SERVICE FOR HUMAN GAIT ANALYSIS AND SIMULATION

Author: Roberto Aguiar Lima
Supervisor: Prof. Dra. Lourdes Mattos Brasil
Co-supervisor: Dra. VERA REGINA DA SILVA MARÃES
Post-Graduation Program in Biomedical Engineering
Brasília, Month of Year.

Texto corrido sem parágrafo. 1 página.

Key-words: gait analysis, machine learning, knee, simulation.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Program Interface*
CSS	*Cascade Style Sheet*
FCE	Faculdade Ceilândia
FGA	Faculdade Gama
HTML	*HiperText Markup Language*
HTTP	*HiperText Transfer Protocol*
GNU	GNU is Not Unix
GUGT	*Get Up and Go Test*
LIS	Laboratório de Informática em Saúde
LPH	Laboratório de Performance Humana
MVC	*Model View Controller*
REST	*Representational State Transfer*
SPA	*Single Page Application*
UnB	Universidade de Brasília
URL	*Uniform Resource Locator*
VM	*Virtual Machine*

LISTA DE SÍMBOLOS

Símbolos Latinos

F_1, F_2	Força (N)

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO E FORMULAÇÃO DO PROBLEMA

1.2 OBJETIVOS

1.2.1 Objetivo Geral

O presente trabalho visa iniciar um projeto de desenvolvimento de software como serviço para análise e simulação de marcha humana.

1.2.2 Objetivos Específicos

Os objetivos específicos são:

- Definir um processo de desenvolvimento ágil adequado ao projeto;
- Explicitar uma visão arquitetural inicial do software;
- Escolher componentes de software a serem usados na solução;
- Criar um *backlog* inicial de histórias de usuários;
- Selecionar um conjunto mínimo de histórias de usuários suficientes para implementação de uma *release* funcional e testes.

1.3 REVISÃO DA LITERATURA

Foram usados os seguinte serviços *web* para o levantamento bibliográfico deste trabalho:

- *IEEE Xplore Digital Library* (http://ieeexplore.ieee.org);
- *PubMed* (http://www.ncbi.nlm.nih.gov/pubmed);
- Portal de Periódicos CAPES/MEC (http://periodicos.capes.gov.br).

Foram utilizadas as seguintes chaves de pesquisa em cada um dos serviços acima: "*Gait Analysis Software*".

Os artigos considerados mais relevantes para o trabalho foram escolhidos, levando-se em consideração, entre outros tópicos, a descrição de características interessantes a serem implementadas no software a ser desenvolvido.

Quando o assunto se trata de análise de marcha, a obra mais aclamada, inclusive citada em muitas das referências pesquisadas, é (PERRY; BURNFIELD, 2010). Como

sugerido por (MALAS, 2010), esta é uma obra obrigatória a qualquer um que deseje estudar análise de marcha.

Em (VIEIRA et al., 2015) um sistema de análise e classificação de marcha é proposto como alternativa a soluções de mercado mais caras. A proposta inicial é coletar dados a partir de marcadores posicionados no corpo do paciente, através de câmeras de vídeo, classificando padrões de marcha com aprendizado de máquina.

Em (DUHAMEL et al., 2004) é apresentada uma ferramenta para melhorar a confiabilidade de curvas para um paciente, classificar pacientes em determinadas populações e comparar populações. Trata-se de uma ferramenta estatística para análise de marcha.

Detecções de eventos do ciclo de marcha, são características interessantes para um software de análise de marcha. Em (GHOUSSAYNI et al., 2004) são documentados métodos para detecção de 4 eventos: contato do calcanhar, elevação do calcanhar, contato do dedão do pé e elevação do dedão do pé.

Uma comparação entre dois pacotes distintos para análise de marchar foi realizada em (MORAES; SILVA; BATTISTELA, 2003). Neste trabalho dados captados por câmeras e plataformas de força são coletados e passados aos pacotes de software *Kin Trak* e *Ortho Trak*.

Uma amostra de como um software pode ser utilizado para gerar bases de dados de análise de marcha, é visto em (MORENO et al., 2009). Neste artigo os autores capturam dados de crianças sadias, afim de obterem padrões para serem utilizados em sistemas de análise de movimentos.

Um sistema de aquisição e análise de marcha, foi desenvolvido e demostrado em (FERREIRA; CRISOSTOMO; COIMBRA, 2009). Neste trabalho, o hardware para captura de dados e o software para análise dos dados, foram desenvolvidos num único projeto. Com os resultados gerados pelas análises feitas por este projeto, foi possível construir um robô bípede, que apresentou resultados satisfatórios caminhando num ciclo de marcha confortável.

A partir da análise de marcha, é possível criar métodos para se estabelecer o grau de desvio do ciclo de marcha que um paciente pode apresentar. Em (BEYNON et al., 2010) é apresentado o método *Gait Profile Score*. O método em si é um bom candidato a funcionalidade em um software de análise de marcha, pois serviria de auxílio clínico ao profissional da área de saúde. Uma outra funcionalidade inspirada no campo clínico é mostrado em (CIPPITELLI et al., 2015). Neste trabalho os autores propõem a automatização do método *Get Up and Go Test*(GUGT), que é usualmente utilizado em análise de marcha no campo da reabilitação.

1.4 ORGANIZAÇÃO DO TRABALHO

2 FUNDAMENTAÇÃO TEÓRICA

2.1 ANÁLISE DE MARCHA

2.2 MÉTODOS ÁGEIS

A muito tempo vários métodos para desenvolvimento de software são propostos. Em 2001, um grupo de pessoas muito experientes em desenvolvimento de software, juntaram-se em Salt Lake City, Utah, para resolverem problemas de desenvolvimento de software (GREENE; STELLMAN, 2014). Como resultado deste encontro, foi criado o manifesto ágil, que é reproduzido a seguir (BECK et al., 2001):

> Estamos descobrindo maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo. Através deste trabalho, passamos a valorizar:
>
> Indivíduos e interações mais que processos e ferramentas;
>
> Software em funcionamento mais que documentação abrangente;
>
> Colaboração com o cliente mais que negociação de contratos;
>
> Responder a mudanças mais que seguir um plano.
>
> Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda.

Meses após a criação do manifesto, estas pessoas também criaram os princípios ágeis e a Aliança Ágil (LAYTON, 2012).

Os princípios ágeis são um conjunto de 12 itens com o objetivo de auxiliar na implantação de metodologias ágeis. Eles são reproduzidos a seguir a título de ilustração (BECK et al., 2001):

> 1. Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado.
>
> 2. Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.
>
> 3. Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.
>
> 4. Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
>
> 5. Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.
>
> 6. O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face.
>
> 7. Software funcionando é a medida primária de progresso.

> 8. Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
>
> 9. Contínua atenção à excelência técnica e bom design aumenta a agilidade.
>
> 10. Simplicidade–a arte de maximizar a quantidade de trabalho não realizado–é essencial.
>
> 11. As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis.
>
> 12. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

Este manifesto serviu de marco agregador de métodos e técnicas, que já existiam a época, mas não eram amplamente difundidas como *Scrum*, *Extreme Programming*, *kanban*, *lean*, entre outros. Estas técnicas, apesar de anteriores ao manifesto, possuem em seus cernes, muito em comum com os valores ágeis. A Figura 1 tenta ilustrar esta ideia.

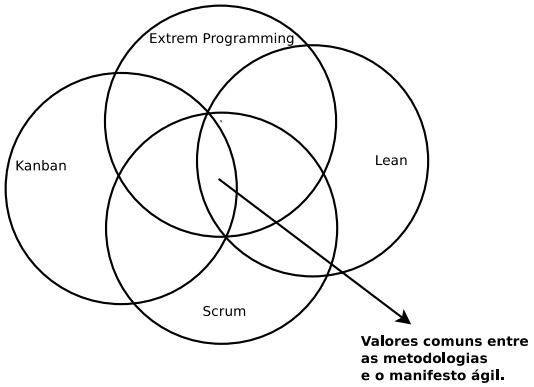

Figura 1 – Valores em comum. Adaptado de (GREENE; STELLMAN, 2014).

A adoção dos métodos ágeis hoje é praticamente unanimidade. Isto se deve aos modelos de desenvolvimento adotados até a década de 1990. Esses modelos eram na sua grande maioria baseados no modelo *waterfall*, que imitava uma linha de produção onde o software era desenvolvido em fases. A saída de cada fase era a entrada da próxima. A Figura 2 mostra um exemplo de processo baseado no modelo *waterfall*. O maior problema do modelo *waterfall*, era que este foi completamente averso a mudanças de requisitos. Hoje, é notório que a grande maioria dos softwares necessitam ser bastante receptivos a mudanças.

(RUNYAN; ASHMORE, 2014) compara o modelo *waterfall* como o modelo ágil. Esta comparação é resumidamente apresentada na tabela 1.

Como evolução do modelo *waterfall*, surgiram os processos baseados em modelos iterativos. A diferença agora é que o processo de desenvolvimento passa a ser baseado em ciclos. Cada ciclo passa por cada uma das fases do *waterfall*. Este tipo de processo, apresentava melhoras, mas ainda era concebido sob a forma de um processo que visava

resolver problemas determinísticos, como o das linhas de produção das fábricas. Mas, como descrito em (BECK, 2004), o processo de se construir software é mais parecido com o ato de dirigir. O motorista sabe o destino a que quer chegar, porém, durante o percurso pode haver um acidente e tem-se que mudar um pouco a rota, ou alguém está passando por uma faixa de pedestre e necessita-se parar, ou ainda um sinal de transito pode ficar vermelho. Esta é a principal motivação para que este trabalho, privilegie métodos ágeis de desenvolvimento.

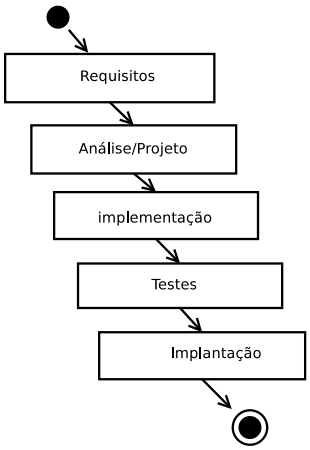

Figura 2 – Exemplo de processo baseado no modelo *waterfall*

Tabela 1 – Comparando Modelos de Desenvolvimento de Software

Waterfall	*Ágil*
Prescritivo	Abstrato
Documentação extensiva	Mínimo de documentação
Sequencial	Contínuo
Formal	Informal
Foco no processo	Foco na comunicação
Mudança gradual	Mudança rápida

Fonte: (RUNYAN; ASHMORE, 2014)

2.2.1 SCRUM

O *Scrum*, segundo (RUBIN, 2012), é um método ágil para desenvolvimento de produtos e serviços inovadores. A Figura 3, mostra uma visão geral do *Scrum*.

Basicamente o fluxo do scrum consiste na criação de um *backlog* de produto. Este *backlog* é uma lista de requisitos a serem implementados no processo de desenvolvimento, e é mantido e priorizado pelo *product owner*. A lista em si pode receber contribuições dos mais diversos envolvidos no processo, mas a última palavra na priorização é sempre do *product owner*. A orientação mais aceita para a priorização dos requisitos, é levar em conta aqueles que mais gerarão valor para o usuário final, no momento em questão.

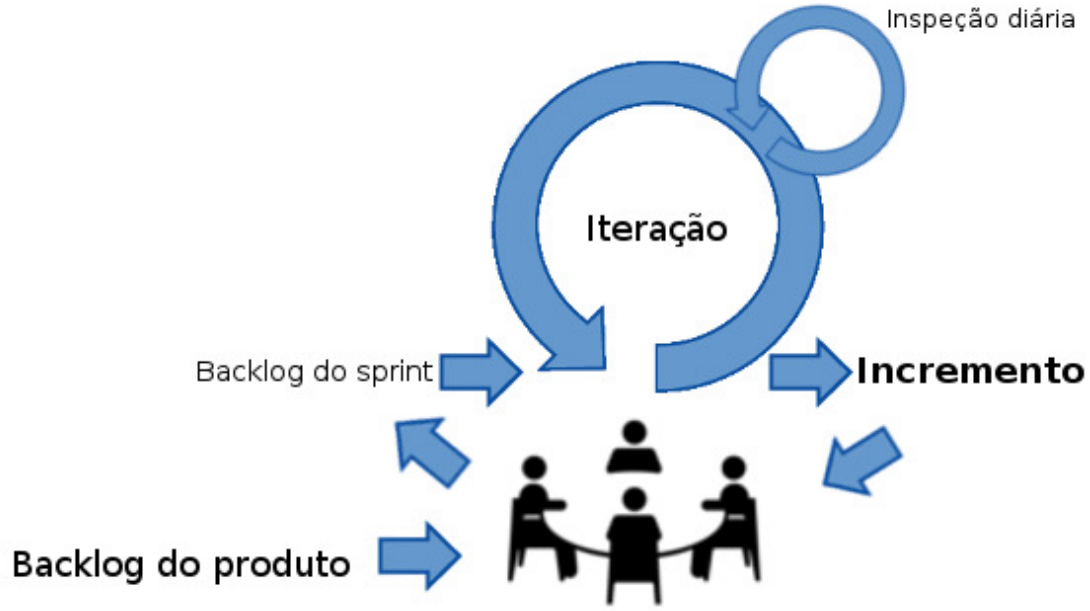

Figura 3 – Visão Geral do *Scrum*. Adaptado de (SCHWABER, 2004).

A próxima etapa no fluxo é a reunião de *sprint*. Esta reunião consiste na demonstração das funcionalidades implementadas na última iteração e na seleção das funcionalidades que serão construídas na próxima iteração. As funcionalidades são escolhidas pelos desenvolvedores, que se comprometem a entregá-las ao final da iteração.

Após a reunião do *sprint*, inicia-se a iteração. A iteração é também chamda de *sprint* e consiste num prazo de alguns dias ou alguns meses. Isto depende do projeto em questão. Geralmente sprints menores são mais aconselhados, uma semana por exemplo (SCHWABER, 2004).

Durante a iteração ou *sprint*, ao final do dia preferencialmente, realiza-se uma reunião de inspeção diária. Esta reunião é também chamada de *daily scrum*. Nesta reunião, os envolvidos diretamente no desenvolvimento do produto ou serviço, expõe o andamento de suas atividades pessoais e os impedimentos para a conclusão destas. A reunião é feita com todos de pé, e cada um dispõe apenas alguns minutos de exposição, por exemplo 3 minutos. A intromissão de pessoas externas deve ser evitada. A ideia é manter a objetividade e o foco nas tarefas do *sprint*. Ao final da reunião o *scrum master* deve procurar intender todos os impedimentos expostos. A partir deste conhecimento ele deve tomar medidas necessárias para removê-los, assim garantindo o bom funcionamento da equipe de desenvolvimento. Ao final do sprint um incremento, ou seja, um pedaço de software com funcionalidades implementadas é construído e preferencialmente implantado. O próximo passo é uma nova reunião de *sprint*.

Os papéis definidos pelo *scrum* são:

- *Product Owner*;
- *Scrum Master*;
- Desenvolvedor.

Nas organizações existirão outros papeis, como gerentes, diretores, patrocinadores, usuários finais, entre outros. O importante a se observar, é que apenas os três papeis, *product owner*, *scrum master* e desenvolvedor, é que fazem parte da equipe *scrum*. Os demais papeis participam do projeto, mas é necessário, o entendimento por parte da organização, que demandas de implementação devem ser colocadas no backlog do produto. Só o *product owner* pode priorizar os novos requisitos. Sem este entendimento e comprometimento, fica inviável respeitar prazos e escopo, tornando o processo caótico.

Devido ao *Scrum* servir de espinha dorsal a um processo de desenvolvimento ágil, também sendo amplamente utilizado pelo mercado de desenvolvimento de software, ele foi selecionado como método de gestão deste trabalho. Todo o processo do *Scrum* pode ser minunciosamente estudado em (SCHWABER; BEEDLE, 2001).

2.2.2 HISTÓRIAS DO USUÁRIO

Histórias do usuário é uma técnica que vem em detrimento da produção massiva de requisitos. Até hoje é comum achar projetos de desenvolvimento de software, com documentos de requisitos enormes. Com raras exceções, este não é um modelo muito produtivo. O grande problema é que muitas vezes gasta-se muito tempo, às vezes anos, para se criar estes documentos. Enquanto isso pouco ou nada de software funcional é produzido. Uma consequência comum desta demora, são as prioridades dos usuários mudarem, ou seja, o que foi especificado inicialmente, depois de um ano, por exemplo, já não tem o mesmo valor para os mesmos.

(COHN, 2004) diz que histórias do usuário descrevem funcionalidades que serão de valor para um usuário ou comprador de um sistema de software. Histórias do usuário são compostas por três aspectos:

- Uma descrição escrita da história usada para planejamento e como lembrete;
- Conversações sobre a história para detalhar a mesma;
- Testes que trazem e documentam detalhes e que podem ser usadas para determinar quando a história está completa.

Histórias do usuário podem ser documentadas usando-se *story cards*. Um exemplo é mostrado na Figura 4.

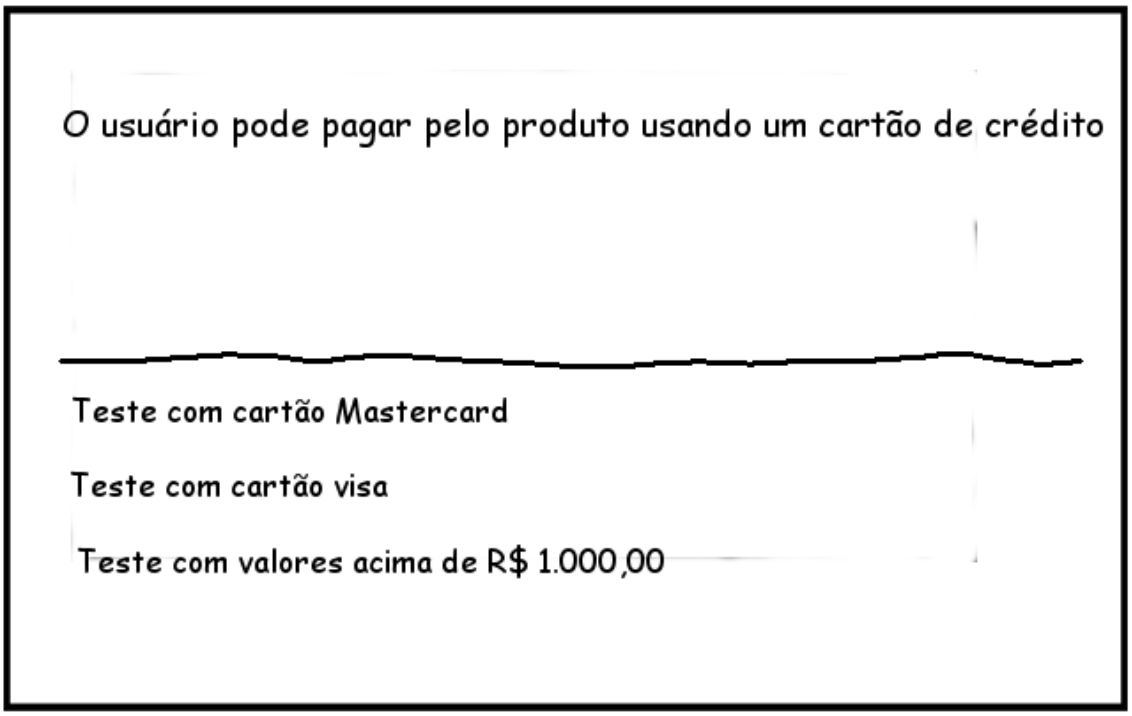

Figura 4 – Exemplo de *story card*.

No *story card* em questão, a parte superior é a descrição do item que é de valor para um usuário. A parte inferior relata testes que devem ser executados. Estes testes também servem para determinar se a história do usuário foi completamente implementada ou não.

Dentro de uma metodologia como o *Scrum*, as histórias de usuário podem ser utilizadas para comporem o *backlog* do produto. Elas são excelentes durante a fase de planejamento do *sprint*.

Como as histórias do usuário são documentos mais informais, é necessário que as mesmas sejam mais detalhadas durante o processo de desenvolvimento. Isto pode ser feito com uma conversa entre o desenvolvedor e o *product owner*. Pode se chegar a conclusão que a história é na verdade um ”épico”, ou seja, é uma história composta de outras histórias. Neste caso a história dever ser quebrada e suas partes enviadas para planejamento novamente.

2.3 SOFTWARE COMO SERVIÇO

A ideia de software como serviço é bastante difundida atualmente. Redes sociais, serviços de busca, serviços de *streaming* de vídeo, são amplamente usados por todos. (FOX; PATTERSON, 2012) define software como serviço, como o software que entrega software e dados como serviços sobre a *Internet*, usualmente via um programa como um *browser* que roda num dispositivo cliente local, em detrimento de código binário que precisa ser instalado e que roda totalmente no dispositivo.

(FOX; PATTERSON, 2012) cita várias vantagens tanto para usuários quanto para desenvolvedores de software. São elas:

1. Desde de que os usuários não necessitam instalar a aplicação, eles não precisam se preocupar se possuem um hardware específico, ou se possuem uma versão específica de sistema operacional;
2. Como os dados associados ao serviço geralmente são mantidos com o serviço, os usuários não precisam em fazer *bakcups*, ou os perderem devido a mal funcionamento, ou até mesmo os perderem devido ao extravios;
3. Quando um grupo de usuários querem coletivamente interagir sobre os mesmos dados, software como serviço é um veículo natural;
4. Faz mais sentido manter grandes quantidades de dados centralizados e manter o acesso remoto a estes.
5. Os desenvolvedores podem controlar as versões de software e sistema operacional que executam o serviço. Isto evita problemas de compatibilidade com distribuição do software devido ao grande número de plataformas que os usuários possuem. Além disso, é possível que o desenvolvedor teste novas versões do software usando uma pequena fração dos usuários temporariamente, sem causar distúrbios na maioria dos usuários;
6. Como os desenvolvedores controlam a versão de execução do software, eles podem mudar até mesmo a plataforma dos mesmos, desde que não violem as API's de interface com o usuário.

Para quem duvida do poder do software como serviço, é só observar que produtos consagrado como o *Microsoft Office* já possuem versão como serviço, no caso o *Microsoft Office 365*. Outros exemplos seriam o *Twitter*, *Facebook*, entre outros.

2.4 COMPONENTES, FRAMEWORKS E FERRAMENTAS

Neste item serão analisados os components, frameworks e ferramentas que foram selecionadas para construção do *Open Gait Analytics*. Este software será um serviço disponibilizado via *web*, sendo assim, estas tecnologias são próprias para este fim.

2.4.1 APLICAÇÕES WEB COM ANGULAJS

O AngularJS é um framework de aplicação *web*. Ele foi projetado especificamente para rodar em *browsers que suportam HTML 5*. Ele também é adequado a criação de aplicações *web* que rodam em *smartphones*. É um framework bastante completo e rico para sua

finalidade. (BRANAS, 2014) é um guia introdutório conciso no assunto. Segundo (FREEMAN, 2014), outro guia no assunto, o *AngularJS* se baseia no padrão de projeto *Model-View-Controller (MVC)* e sua enfase é em permitir a criação de aplicações: extensíveis, manuteníveis, testáveis e padronizadas. A Figura 5 mostra uma representação de uma aplicação fazendo uso do *AngularJS*.

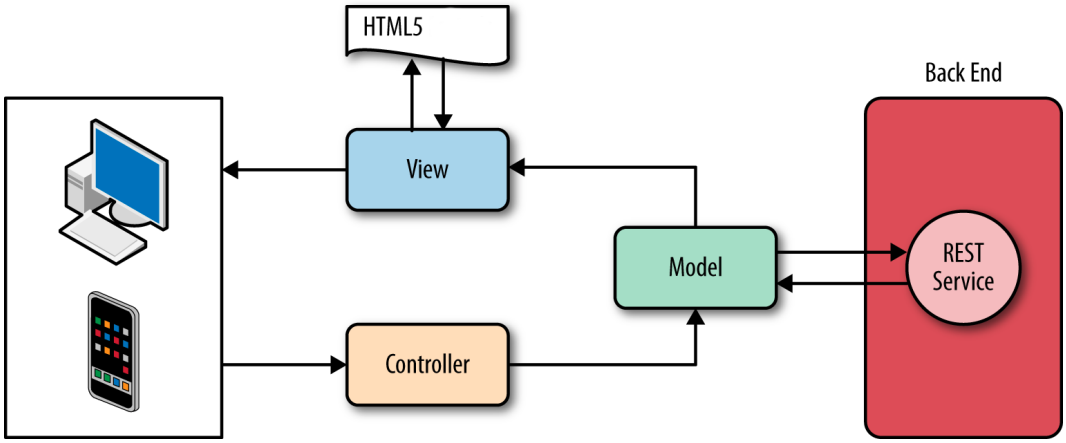

Figura 5 – Diagrama de uma aplicação *MVC* usando *AngularJS*.
Fonte: (WILLIAMSON, 2015).

Segundo (WILLIAMSON, 2015), a Figura 5 mostra o diagrama de uma aplicação *AngularJS* e os componentes *MVC*. Uma vez que a apliação é lançada, os componentes *model, view* e *controller*, juntamente com todos os documentos *HTML* são carregados no *desktop* ou *smartphone* do usuário e rodam completamente num destes hardwares. A aplicação *AngularJS* conversa com o *backend* via o protocolo *http*. O *backend* é um servidor *web* que mantém chamadas *REST* (explicado na seção 2.4.4), sendo responsável pela execução da lógica e de processos de negócio. Outra característica bastante apreciada pelos desenvolvedores que usam *AngularJS*, é a possibilidade de criar *single-page aplications (SPA)*. *SPAs* são aplicações que tem uma página *HTML* de entrada. Esta página tem seu conteúdo dinamicamente adicionado e removido da mesma. Esta abordagem permite criar aplicações bastante interativas, lembrando mesmo, aplicações *desktop* escritas em linguagens como *Visual Basic* e *Delphi*.

2.4.2 SOLUÇÃO DE DESIGN WEB COM *ANGULAR-MATERIAL*

Como até o momento de conclusão deste trabalho, o projeto não conta com um *web designer*, uma solução técnica para minimizar as consequências deste problema, teve de ser adotada. Os usuários finais do software, profissionais da área de saúde, dificilmente se interessariam pelo mesmo sem uma interface atraente e amigável. A solução adotada foi utilizar a biblioteca *angular-material*. Esta biblioteca, como indicado pelo seu nome, é construída com o *AngularJS*. Ela disponibiliza serviços e diretivas que podem ser usados para construir a interface gráfica da aplicação. Diretivas são componentes que podem ser

inseridos diretamente no código HTML da aplicação, dando a aparência de estender a própria HTML. Por exemplo, a diretiva *md-button* da biblioteca é um tipo de botão que não é próprio do HTML. Outros exemplos de diretivas são: caixas de diálogos, barras de ferramentas, barras de progresso, grades, *tooltip*, etc. Esta biblioteca é baseada na especificação *Material Design* criada pela empresa *Google*. A especificação discorre sobre padrões de designe gráfico e interação com usuário e é baseada no princípio da metáfora de materiais. Esta metáfora é uma teoria unificada de um espaço racionalizado e sistemas de movimento, isto segundo (GOOGLE, 2015). Outra vantagem da biblioteca é que ela é projetada para se adaptar a diferentes tipos de dispositivos com telas de tamanhos diferentes.

A Figura 6, mostra um exemplo de uma tela criada com as diretivas do angular-material.

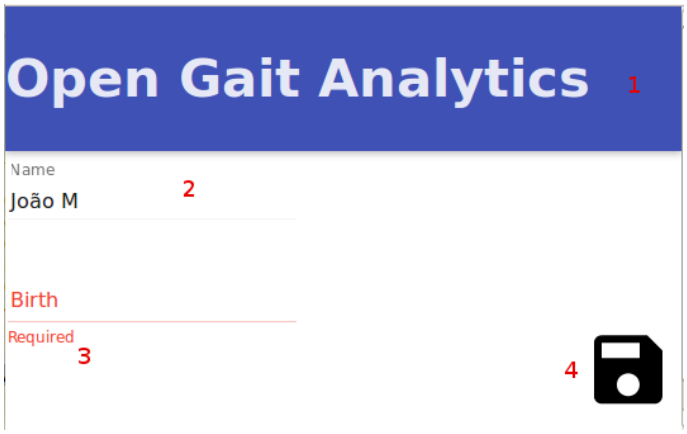

Figura 6 – Exemplo de uma tela criada com *angular-material*. 1) Diretiva *md-toolbar* ; 2) Diretiva *md-input-container*; 3) Diretiva *ng-messages* em conjunto com a *md-input-container*; 4) Diretiva *md-button*.

2.4.3 GRÁFICOS E ANIMAÇÕES 3D NA *WEB* COM THREEJS

Os browsers modernos hoje, inclusive os dos *smartphones*, suportam o novo padrão *WebGL*. Este é um padrão *web* multi-plataforma de *application program interface (API)* de baixo nível para gráficos 3D, expostos através de *HTML*. Este padrão suporta o acesso da *API* usando-se a linguagem *GLSL*. Uma vantagem desta API é que ela suporta nativamente *GPUs* disponibilizadas pelo hardware que está executando o cliente *web*. Isto torna possível até mesmo a criação de jogos de alta definição em 3D que rodam no *browser*.

O problema com a *WebGL* é que, como dito anteriormente, a *API* é de baixo nível. No contexto de computação gráfica, isto significa que ela fornece primitivas básicas para modelagem 3D e outras opções de otimização do hardware. Para resolver este problema bibliotecas em *javascript* foram desenvolvidas, disponibilizando funções e objetos de alto nível, como cilindros, planos, esferas, animações, entre outros. Uma opção é o *ThreeJS*,

descrito em (DIRKSEN, 2015). Esta biblioteca apresenta um grande número de funcionalidades e é possível encontrar aplicações e jogos em 3D de nível profissional. Um exemplo de animação renderizada que utiliza *ThreeJS* é mostrada na Figura 7. Mais exemplos podem ser vistos no site *threejs.org*.

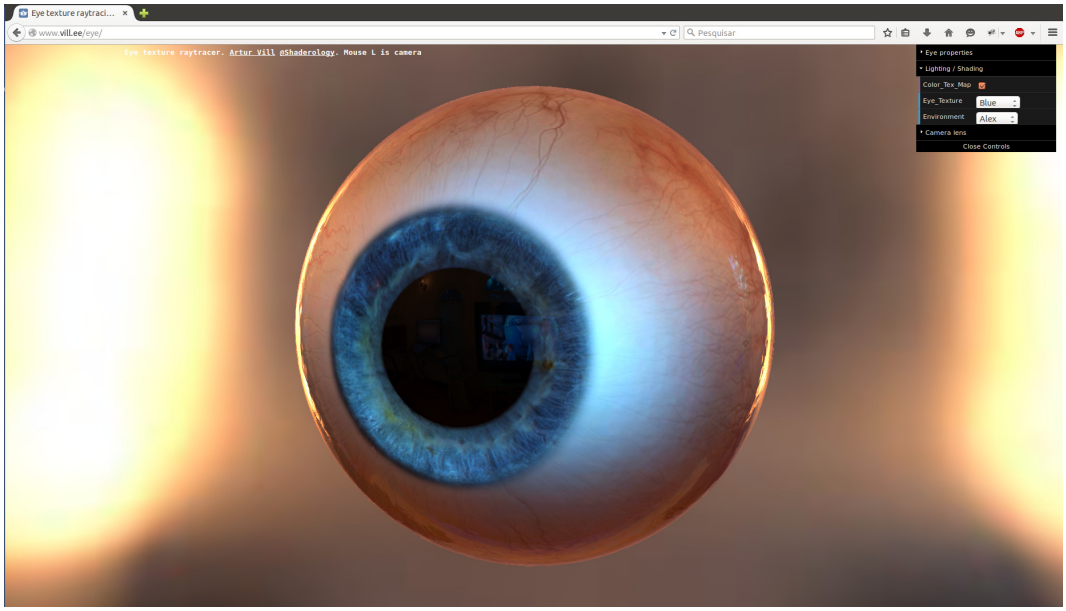

Figura 7 – Exemplo de aplicação que utiliza *ThreeJS*.

Fonte: ⟨http://www.vill.ee/eye/⟩.

2.4.4 SERVIÇOS REST COM FLASK

O *Representational State Transfer (REST)* foi primeiramente descrito por (FIELDING, 2000). Ele é definido como um estilo arquitetural de sistemas hipermídia e tem as seguintes características descritas por (GRINBERG, 2014):

1. Cliente-Servidor. Há uma separação clara entre quem consome, cliente, e quem serve e executa a lógica de negócio, servidor.

2. *Stateless*. O cliente precisa incluir nas suas requisições, todas as informações necessárias para o servidor processar o pedido. O servidor não armazena informações de estado do cliente entre as requisições deste.

3. Interface Uniforme. O protocolo que os clientes usam para acessar o servidor precisa ser consistente, bem definido e padronizado. O protocolo comumente usado por serviços *REST* é o *HTTP*.

4. Sistema em Camadas. Servidores *proxy*, *caches* ou *gateways*, podem ser inseridos entre clientes e servidores quando necessários, para melhorar a performance, confiabilidade e escalabilidade.

5. Código Sob Demanda. Clientes podem opcionalmente fazer o *download* de código do servidor e executá-lo em seu contexto.

A principal idéia de serviços *REST* é o fornecimento de recursos. Por exemplo o cliente requere um usuário, blog, comentários, entre outros. Cada recurso deve possuir uma *URL* que o identifica unicamente, por exemplo http://www.minhaurl.com.br/usuario/123, onde 123 é um identificador único do usuário.

Para que um serviço *REST* funcione, ele precisa ser implementado para suportar requisições *HTTP*, ou seja, ele teria que ser um servidor *web* completo. A biblioteca escolhida para desempenhar esta função foi a *Flask*. Esta é uma biblioteca escrita na linguagem *Python* e fornece todos as ferramentas necessárias para criar aplicações *webs*. Segundo (MAIA, 2015), o *Flask* vem sendo adotado, por sua filosofia minimalista que não impõe uma arquitetura específica de projeto, assim permitindo que um projeto comece pequeno e simples, evoluindo para um modelo mais complexo.

2.5 CMAC

A *Cerebellar Model Articulation Controller* (CMAC) foi criada por James Sacra Albus (ALBUS, 1975a). Ele se inspirou no cerebelo dos mamíferos para criá-la. O mesmo autor havia feito um extenso trabalho sobre o funcionamento do cerebelo (ALBUS, 1971). Trabalho este, que resultou numa tese de doutorado (ALBUS, 1972). Aplicações da CMAC podem ser vistas em (ALBUS, 1975b), (ALBUS, 1979), (SABOURIN, 2006) e (LIN; SONG, 2002).

Na Figura 8 é possível ver o funcionamento básico da CMAC. Os sinais S entram no sistema, que mapeiam o mesmo para um conjunto de pesos W^* que devem ser somados para ativação. Note que apenas uma pequena fração de pesos é realmente selecionada para participar na ativação. O conjunto de pesos disponíveis na CMAC é necessariamente maior que o número de pesos ativados W^*.

A CMAC da Figura 8 também pode ser classificada como um sistema *Multiple Input Single Output* (MISO), ou seja, suporta a entrada de vários sinais de entrada e processa um sinal de saída. Para se produzir uma CMAC *Multiple Input Multiple Output* MIMO, bastaria implementar várias MISOs, compartilhando as mesmas entradas.

Os passos para que os sinal seja computado são descritos a seguir. Estes passos são os mesmos descritos em (ALBUS, 1975a).

Primeiramente, define-se o número de pesos $NW*$ a serem ativados para comporem a saída da CMAC.

O segundo passo é quantizar os possíveis valores para cada item do vetor de entrada S. Por exemplo, se o primeiro item $s1$ de S aceita valores de -1 até 1 e se quer 5 valores

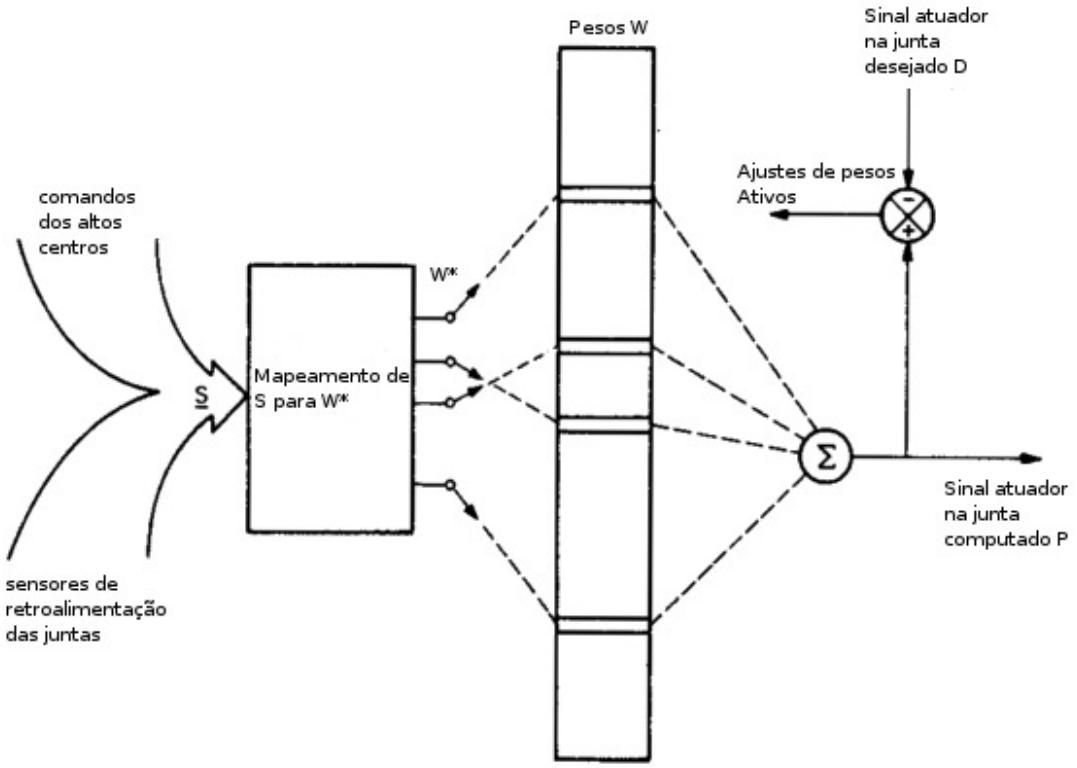

Figura 8 – CMAC para controle de uma junta. Adaptado de (ALBUS, 1975a).

possíveis, quantiza-se então os valores de -1 até 1, conforme Figura 9.

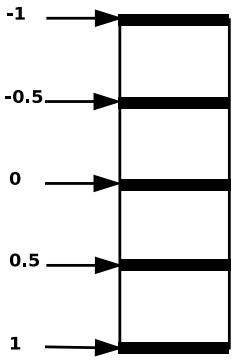

Figura 9 – Discretização de $s1$

Isto significa que quaisquer que sejam os valores de $s1$ os mesmos devem ser convertidos para -1, -0,5, 0, 0,5 e 1. Por exemplo, se o valor de $s1$ for 0,75, será convertido para o valor 1, se for -0,75 será o valor 0 e se for 0,25 será o valor 0,5. A esta quantização dá-se o nome de resolução da CMAC.

O próximo passo é criar uma tabela para cada um dos sinais discretizados de entrada do vetor S. Supondo que o vetor S possui 2 sinais de entrada $s1$ e $s2$ e um número de ativações $NW*$ igual a 3, deve-se criar duas tabelas, uma para cada sinal, a

Tabela 2 e a Tabela 3. Para facilitar o entendimento, irá se considerar os valores possíveis de $s1$ iguais aos inteiros de 1 até 6 e os valores possíveis de $s2$ iguais aos inteiros de 1 até 4. Cria-se uma coluna com os valores do sinal em questão. Para cada valor, cria-se 3 ($NW*$) valores novos. Para o primeiro valor do sinal, usa-se os 3 primeiros inteiros não negativos. Para o segundo valor do sinal, substitui-se o primeiro dos três itens pelo próximo número após o terceiro item do sinal anterior. Para o terceiro sinal, substitui-se o segundo dos três itens pelo próximo inteiro não negativo. Assim sucessivamente conforme a Tabela 2 e Tabela 3.

Tabela 2 – Mapeamento de $s1$

Valore de $s1$	*Mapeamento* $m1$
1	0, 1, 2
2	3, 1, 2
3	3, 4, 2
4	3, 4, 5
5	6, 4, 5
6	6, 7, 5

Tabela 3 – Mapeamento de $s2$

Valore de $s2$	*Mapeamento* $m2$
1	0, 1, 2
2	3, 1, 2
3	3, 4, 2
4	3, 4, 5

Depois de mapeado cada valor de cada item de entrada, deve-se combinar os mapeamentos de acordo com a Tabela 4.

Tabela 4 – Mapeamento para os pesos W

$s1$ / $s2$	*1*	*2*	*3*	*4*
1	(0, 0), (1, 1), (2, 2)	(0, 3), (1, 1), (2, 2)	(0, 3), (1, 4), (2, 2)	(0, 3), (1, 4), (2, 5)
2	(3, 0), (1, 1), (2, 2)	(3, 3), (1, 1), (2, 2)	(3, 3), (1, 4), (2, 2)	(3, 3), (1, 4), (2, 5)
3	(3, 0), (4, 1), (2, 2)	(3, 3), (4, 1), (2, 2)	(3, 3), (4, 4), (2, 2)	(3, 3), (4, 4), (2, 5)
4	(3, 0), (4, 1), (5, 2)	(3, 3), (4, 1), (5, 2)	(3, 3), (4, 4), (5, 2)	(3, 3), (4, 4), (5, 5)
5	(6, 0), (4, 1), (5, 2)	(6, 3), (4, 1), (5, 2)	(6, 3), (4, 4), (5, 2)	(6, 3), (4, 4), (5, 5)
6	(6, 0), (7, 1), (5, 2)	(6, 3), (7, 1), (5, 2)	(6, 3), (7, 4), (5, 2)	(6, 3), (7, 4), (5, 5)

O mapeamento da Tabela 4 é feito da seguinte forma: Coloca-se cada valor quantizado de $s1$ nomeando cada uma das linhas da tabela. Coloca-se cada valor discretizado de s2 nomeando cada uma das colunas das tabelas. Cada célula da tabela é obtida recuperando-se os itens de s1 e de s2, que estão na Tabela 2 e na Tabela 3, e concatenando-se cada item de acordo com sua posição. Por exemplo, para o valor de $s1$ igual a 5, recuperam-se os itens (6, 4, 5), para o valor de $s2$ igual a 2, recuperam-se os itens (3, 1,

2). Agora é só criar novos itens, na mesma quantidade do valor $NW*$, concatenando-se cada item de acordo com sua posição. O lista resultante é ((6, 3), (4, 1), (5, 2)).

Cada vetor de 2 posições da Tabela 4, por exemplo (0, 0), (4, 1), é um rótulo de uma entrada na tabela de pesos W.

Para se calcular a saída do sistema, suponha que os valores de $s1$ e $s2$, ainda sejam 5 e 2, respectivamente, e $NW*$ ainda seja 3. Neste caso, recupera-se o item da linha 5 e da coluna 2, que é ((6, 3), (4, 1), (5, 2)). Agora é acessada a tabela de pesos W e são recuperados os pesos cujas chaves são (6, 3), (4, 1) e (5, 2). Note-se que serão recuperados exatamente 3 pesos, que é exatamente o número de ativações $NW*$ desejadas. Os valores recuperados constituem um vetor chamado $W*$. A saída é calculada somando-se os valores de W*.

2.5.1 TREINAMENTO DA CMAC

Sendo a CMAC uma RNA com aprendizado supervisionado, seus pesos podem ser atualizados simplesmente computando-se o erro e atualizando-se apenas os pesos que participaram do computo do sinal P. O processo é detalhado nos próximos parágrafos.

Para se treinar a CMAC, primeiro define-se o conjunto de dados para treinamento que devem ser usados. Pode ser usado algo entre 20% e 50% dos dados coletados para desenvolvimento do sistema. Cada item destes dados deve conter um vetor de sinais de entradas S e a resposta desejada D para estes sinais. A tabela de pesos deve ser inicializada com valores randômicos. Para cada vetor S deve ser calculado o vetor $W*$, que contém os endereços dos pesos a serem ativados. Calcula-se então a saída da rede P para o vetor S, conforme definido no item 2.5, somando os itens do vetor W. Atualizam-se os pesos sinápticos conforme a Equação 1 adaptada de (SABOURIN; YU; MADANI, 2012).

$$w_i = w_i + \frac{\alpha(D - P)}{NW^*} \tag{1}$$

A variável w_i é um item qualquer dentro da tabela de pesos W. Em cada iteração no conjunto de dados para treinamento, deve-se atualizar apenas os pesos descritos em $W*$ naquele momento. A variável α é o coeficiente de aprendizado, deve ser um valor entre 0 e 1. A variável $NW*$ é o número de valores a serem utilizados para ativação, que consequentemente equivale ao número de itens do vetor $W*$.

Deve-se determinar o número de iterações *batch* para o sistema. Uma iteração *batch* consiste no processamento dos pesos para cada item dentro do conjunto de dados para treinamento. O sistema deve continuar iterando até atingir o número de iterações *batch*.

Apesar do número de iterações *batch* ser válido como critério de parada para o treinamento da CMAC, nada impede o uso de outros critérios, por exemplo, o erro quadrado médio da saída Y em relação ao desejado D.

3 METODOLOGIA

3.1 AMBIENTE DE ESTUDO

Como este trabalho trata de um projeto de desenvolvimento de software na área de análise de marcha, dois ambientes de trabalho distintos foram amplamente utilizados. São estes:

1. Laboratório de Informática em Saúde (LIS) na Faculdade Gama (FGA) da Universidade de Brasília (UnB);
2. Laboratório de Performance Humana (LPH) na Faculdade Ceilândia (FCE) da UnB.

No LIS foram desempenhadas as tarefas relativas a engenharia de software e disponibilização do software. Foram utilizadas estações de trabalho do tipo *Power Mac* com sistema operacional *MAC OS X* 10.10.3, sendo que uma foi preparada para funcionar como servidor de aplicação e roteador de rede. A estação preparada, serve de hospedeira de duas máquinas virtuais (*Virtual Machines* - VMs) rodando através do software *Virtualbox* 4.3. Cada VM utiliza sistema operacional *Debian Wheezy GNU/Linux*. Uma destas VMs foi configurada como roteador e *firewall* e a outra como servidor de aplicações. Outra estação de trabalho *Power Mac*, idêntica, e um notebook rodando *Ubuntu 14.04 GNU/-Linux* foram utilizados como máquinas de desenvolvimento e simulação. Um diagrama da rede criada no LIS é mostrado na Figura 10.

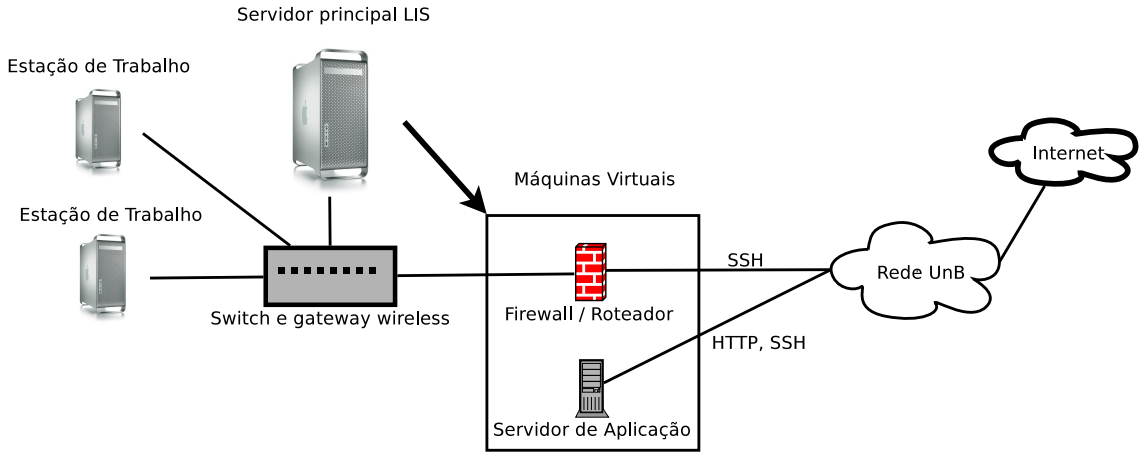

Figura 10 – Rede LIS

O LPH/FCE foi utilizado para captura de dados de marcha humana. Este laboratório está equipado para coletar dados de plataformas de força, eletromiógrafos e de marcadores posicionados no corpo do paciente através de câmeras de vídeo (*Motion Capture - MOCAP*). Para este trabalho foi utilizado o software *QTM 3.2* da *Qualisys*, que é responsável pela coleta de de dados *MOCAP*.

3.2 DELIMITAÇÃO DO ESTUDO

Este trabalho tem como foco estabelecer uma metodologia de desenvolvimento inicial a um sistema de análise e simulação de marcha. Ele não busca ser extensivo o suficiente para criar um produto pronto para o mercado, mas pretende, através da implementação de funcionalidades reais, estabelecer uma arquitetura mínima e funcional que sirva de base para a construção do sistema. Como consequência, o projeto também integra os principais componentes desta arquitetura, por exemplo, o serviço de banco de documentos, com a *Application Program Interface* (API) *web*. Um software como este, robusto o suficiente para ser viável no mercado, seria muito caro. Por exemplo, um desenvolvedor sênior no mercado de Brasília, não custaria menos de R$ 100.000,00 por ano.

3.3 VISÃO

Apesar deste trabalho ter um objetivo específico e delimitado, dele nasce um projeto maior, cuja visão é o desenvolvimento de um software como serviço para análise e simulação de marcha, utilizando o estado da arte em técnicas para este fim. O software deve ser construído utilizando-se métodos ágeis e terá uma arquitetura adaptável que permita evolução contínua.

A estratégia é lançar a versão inicial do software como projeto de código livre, conseguir parceiros e procurar um modelo de negócio sustentável para mantê-lo.

3.4 MODELO DE GESTÃO

O software terá um modelo de gestão baseado no método *SCRUM*, como definido por (SHWABER; BEEDLE, 2002). O método não é adotado na plenitude, sendo adaptado segundo as limitações de recursos do projeto.

Nesta fase inicial, o processo conta com um desenvolvedor, que também assume o papel de *scrum master*, e um *product owner*. Devido ao tamanho reduzido da equipe e da localização distinta dos membros, não há *scrum* diário, mas problemas de trabalho cotidianos são resolvidos por telefone, *email* ou mensagens instantâneas.

O princípio de *time boxing* é mantido. Ficou definido que o *sprint* consiste do prazo de duas semanas. Ao final do *sprint* uma reunião em duas fases é realizada. A primeira fase consiste na revisão do *sprint* anterior. Já a segunda fase é o planejamento do próximo *sprint*.

Um *backlog* de produto é mantido. Na reunião de final de *sprint* é criado um *backlog* de *sprint*. Os itens de *backlog* são mantidos na forma de estórias de usuários, conforme (COHN, 2004). Na Figura 11 é apresentada a visão geral do processo.

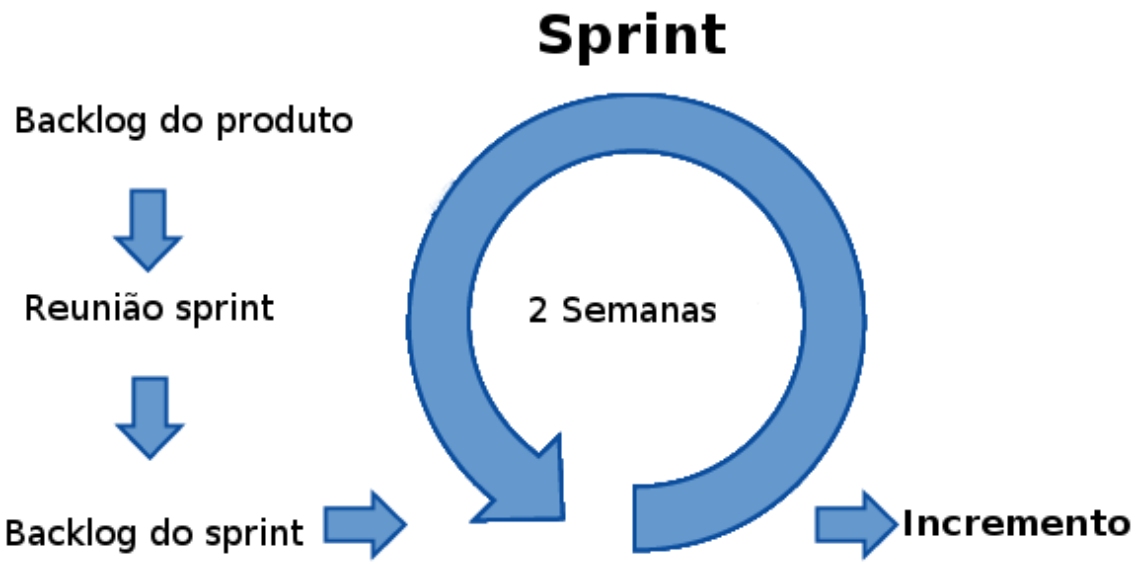

Figura 11 – Processo de desenvolvimento

3.5 MODELO DE ARQUITETURA

O modelo de arquitetura no seu nível mais elevado, pode ser visto como um modelo de três camadas, conforme a Figura 12.

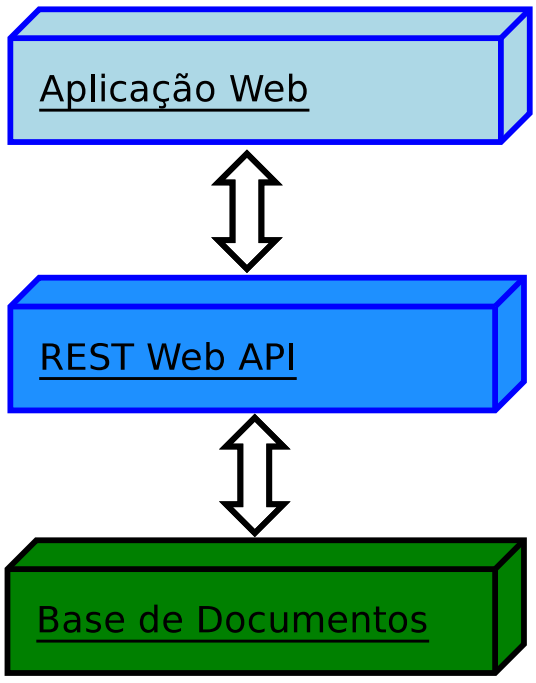

Figura 12 – Camadas arquiteturais

A camada *web* é responsável pela interação com o usuário. A camada *Web API* é responsável pela lógica de negócio. A camada de base de documentos é responsável pela

persistência dos dados da aplicação.

3.5.1 CAMADA DE APLICAÇÃO WEB

Esta camada foi projetada para rodar em *browsers* que suportam *HTML* 5. Ela é desenvolvida usando-se *Javascript*, *CSS* e *HTML*. Além disso, adotou-se o *framework* de desenvolvimento *web AngularJS*. **ORGANIZAÇÃO DO CÓDIGO FONTE**

A criação de um ambiente de desenvolvimento *web*, para um software de média para grande complexidade, não é uma tarefa trivial de ser resolvida. É necessários criar padrões de organização de arquivos, configurar e instalar pacotes de software para desenvolvimento, testes, implantação, construção de builds, entre outros. Para facilitar esta tarefa, optou-se em utilizar o projeto *angular-seed*. A ideia deste projeto é servir de esqueleto de projetos *web* que utilizam o *framework AngularJS*. Para usar este projeto basta cloná-lo diretamente do seu repositório *git* no *site* github.com, conforme o comando abaixo.

```
git clone https://github.com/angular/angular-seed.git
```

Antes de começar a usar o *angular-seed* é necessário instalar o ambiente de execução *Javascript Node.JS*. Este ambiente é utilizado para execução de testes e construção de *builds*. Para uma introdução ao *Node.JS* veja (SYED, 2014).

4 RESULTADOS

5 DISCUSSÃO E CONCLUSÃO

6 TRABALHOS FUTUROS

REFERÊNCIAS BIBLIOGRÁFICAS

ALBUS, J. A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC). *... Systems, Measurement, and Control*, n. SEPTEMBER, p. 220–227, 1975. Disponível em: ⟨http://dynamicsystems.asmedigitalcollection.asme.org/article.aspx?articleid=1402197⟩. 10, 25, 26

ALBUS, J. Data Storage in The Cerebellar Model Articulation Controller (CMAC). *... Systems, Measurement, and Control*, 1975. Disponível em: ⟨http://dynamicsystems.asmedigitalcollection.asme.org/article.aspx?articleid=1402199⟩. 25

ALBUS, J. Mechanisms of planning and problem solving in the brain. *Mathematical Biosciences*, v. 293, 1979. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/0025556479900634⟩. 25

ALBUS, J. S. A robot conditioned reflex system modeled after the cerebellum. *Proceedings of the December 5-7, 1972, fall joint computer conference, part II on - AFIPS '72 (Fall, part II)*, ACM Press, New York, New York, USA, p. 1095, 1972. Disponível em: ⟨http://portal.acm.org/citation.cfm?doid=1480083.1480144⟩. 25

ALBUS, S. A Theory of Cerebellar Function. v. 10, p. 25–61, 1971. 25

BECK, K. *Extreme Programming Explained: Embrace Change*. 2. ed. [S.l.]: Addison-Wesley Professional, 2004. ISBN 9780321278654. 17

BECK, K. et al. *Manifesto para Desenvolvimento Ágil de Software*. 2001. Disponível em: ⟨http://www.agilemanifesto.org/iso/ptbr/⟩. 15

BEYNON, S. et al. Correlations of the Gait Profile Score and the Movement Analysis Profile relative to clinical judgments. *Gait and Posture*, Elsevier B.V., v. 32, n. 1, p. 129–132, 2010. ISSN 09666362. Disponível em: ⟨http://dx.doi.org/10.1016/j.gaitpost.2010.01.010⟩. 14

BRANAS, R. *AgularJS Essentials*. Birmingham: Packt Publishing Ltd., 2014. ISBN 978-1-78398-008-6. 22

CIPPITELLI, E. et al. Kinect as a Tool for Gait Analysis: Validation of a Real-Time Joint Extraction AlgorithmWorking in Side View. *Sensors*, v. 15, n. 1, p. 1417–1434, 2015. ISSN 1424-8220. Disponível em: ⟨http://www.mdpi.com/1424-8220/15/1/1417/⟩. 14

COHN, M. *Users Stories Applied*. Crawfordsville: Addison Wesley, 2004. ISBN 978-0-321-20568-1. 19, 30

DIRKSEN, J. *Learning Three.js - the JavaScript 3D Library for WebGL*. 2. ed. [S.l.]: Packt Publishing, 2015. ISBN 9781784392215. 24

DUHAMEL, a. et al. Statistical tools for clinical gait analysis. *Gait and Posture*, v. 20, n. 2, p. 204–212, 2004. ISSN 09666362. 14

FERREIRA, J. a. P.; CRISOSTOMO, M. M.; COIMBRA, a. P. Human gait acquisition and characterization. *IEEE Transactions on Instrumentation and Measurement*, v. 58, n. 9, p. 2979–2988, 2009. ISSN 00189456. 14

FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Tese (Doutorado) — UNIVERSITY OF CALIFORNIA, 2000. Disponível em: ⟨http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm⟩. 24

FOX, A.; PATTERSON, D. *Engineering Long-Lasting Software: An Agile Approach Using SaaS and Cloud Computing*. [S.l.]: Strawberry Canyon LLC, 2012. ISBN 0984881212. 20, 21

FREEMAN, A. *Pro AngularJS*. [S.l.]: Apress, 2014. ISBN 9781430264484. 22

GHOUSSAYNI, S. et al. Assessment and validation of a simple automated method for the detection of gait events and intervals. *Gait and Posture*, v. 20, n. 3, p. 266–272, 2004. ISSN 09666362. 14

GOOGLE. *Material Design*. 2015. Disponível em: ⟨www.google.com.br/design/spec/material-design/introduction.html⟩. 23

GREENE, J.; STELLMAN, A. *Learn Agile*. 1. ed. [S.l.]: O'Reilly Media, Inc., 2014. ISBN 9781449331924. 10, 15, 16

GRINBERG, M. *Flask Web Development*. 1. ed. [S.l.]: O'Reilly Media, Inc., 2014. ISBN 9781449372620. 24

LAYTON, M. C. *Agile Project Management For Dummies*. 1. ed. [S.l.]: For Dummies, 2012. ISBN 9781118235850. 15

LIN, J.-n.; SONG, S.-m. Modeling gait transitions of quadrupeds and their generalization with CMAC neural networks. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, v. 32, n. 3, p. 177–189, ago. 2002. ISSN 1094-6977. Disponível em: ⟨http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1097731⟩. 25

MAIA, I. *Building Web Applications with Flask*. 1. ed. [S.l.]: Packt Publishing, 2015. ISBN 9781784396152. 25

MALAS, B. *Book Review - Gait Analysis: Normal and Pathological Function, 2nd Edition*. 2010. Disponível em: ⟨http://www.oandp.org/reading/gaitfunction.asp⟩. 14

MORAES, J.; SILVA, S.; BATTISTELA, L. Comparison of two software packages for data analysis at gait laboratories. *Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (IEEE Cat. No.03CH37439)*, v. 2, p. 1780–1783, 2003. ISSN 1094-687X. 14

MORENO, a. et al. Development of the spatio-temporal gait parameters of Mexican children between 6 and 13 years old data base to be included in motion analysis softwares. *2009 Pan American Health Care Exchanges - PAHCE 2009*, n. 2, p. 90–93, 2009. 14

PERRY, J.; BURNFIELD, J. M. *Gait Analysis Normal and Pathological Function*. 2nd. ed. [S.l.]: SLACK Inc., 2010. ISBN 978-1-55642-766-4. 13

RUBIN, K. S. *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. [S.l.]: Addison-Wesley Professional, 2012. ISBN 9780137043293. 17

RUNYAN, K.; ASHMORE, S. *Introduction to Agile Methods*. [S.l.]: Addison-Wesley Professional, 2014. 16, 17

SABOURIN, C. Control Strategy for the Robust Dynamic Walk of a Biped Robot. *The International Journal of Robotics Research*, v. 25, n. 9, p. 843–860, set. 2006. ISSN 0278-3649. Disponível em: ⟨http://ijr.sagepub.com/cgi/doi/10.1177/0278364906069151⟩. 25

SABOURIN, C.; YU, W.; MADANI, K. Gait Pattern Based on CMAC Neural Network for Robotic Applications. *Neural Processing Letters*, v. 38, n. 2, p. 261–279, nov. 2012. ISSN 1370-4621. Disponível em: ⟨http://link.springer.com/10.1007/s11063-012-9257-6⟩. 28

SCHWABER, K. *Agile Project Management with Scrum*. [S.l.]: Microsoft Press, 2004. ISBN 9780735619937. 10, 18

SCHWABER, K.; BEEDLE, M. *Agile Software Development with Scrum*. [S.l.]: Pearson, 2001. ISBN 0130676349. 19

SHWABER, K.; BEEDLE, M. *Agile Software Development with Scrum*. [S.l.]: Pearson, 2002. ISBN 0-13-067634-9. 30

SYED, B. A. *Beginning Node.js*. [S.l.]: Apress, 2014. ISBN 9781484201879. 32

VIEIRA, A. et al. Software for human gait analysis and classification. In: *4th Portuguese BioEngineering Meeting*. Porto: [s.n.], 2015. 14

WILLIAMSON, K. *Learning AngularJS*. 1. ed. [S.l.]: O'Reilly Media, Inc., 2015. ISBN 9781491916759. 22