Estudo Simples sobre Agile Modeling (AM)

Lucas P. Tonussi *
* Departamento de Informática e Estatística
Universidade Federal de Santa Catarina
tonussi@inf.ufsc.br

***Resumo*—Minha abordagem princípal é entender melhor o desenvolvimento ágil, estudando mais aprofundadamente a modelagem ágil.**

***Index Terms*—Agile, software development model, agile modeling, agile documentation.**

I. Introdução

Modelagem ágil (Agile Modeling - AM) deve ser entendida como um complemento à métodos. Métodos de desenvolvimento de software, esses, podendo ser por exemplo RUP, Unified Process, XP eXtreme Programming, SCRUM, dentre outros. Assim a busca nesse artigo é tentar entender, desmistificar a modelagem ágil baseando-se no livro do Scott Ambler[1].

II. Visão Princípal

Modelagem Ágil (AM - Agile Modeling) é uma prática para **modelagem e documentação** efetiva (ou seja não necessáriamente ágil significa documentação de lado e produção a todo vapor). AM é um método que visa, desenvolvimento e projeto de software como sistema. Agile Modeling (AM) é uma coleção de valores, princípios e práticas. Para modelagem de software que pode ser aplicada no desenvolvimento de um projeto de software de maneira simples e efetiva.

Os valores da modelagem ágil, adotam e exportam do eXtreme Programming, que são: **comunicação, retorno (feedback), coragem, e humildade**. O objetivo é que você tenha comunicação efetiva. Como por exemplo telefones sempre a mão dos analístas de sistema, engenheiros de usabilidade, e até mesmo dos programadores, para que eles possam acionar rapidamente o cliente, ou a equipe. Se o cliente, então para poder desvendar alguma história de usuário (se você estiver trabalhando com SCRUM) ou ainda CRC Cards (SCRUM ou eXtreme Programming ou FDD). A comunicação deve ser eficiente, rápida entre todos os stakeholders do projeto em ação. O motivo, obter feedback mantendo o resultado do esforço da equipe, constante durante o começo do projeto, que é onde muitas desmistificações acontecem. E também onde riscos do projeto são entendídas. Para ter a coragem de fazer e ter certeza de suas decisões, e também ter a humildade para admitir que você pode não saber tudo. Que outros tem valor para adicionar melhorias ao seu eforço em cima do projeto. Ou ainda, criticar o seu trabalho (esforço aplicado no projeto).

III. Princípios

Finally, a focus on quality work is important because nobody likes to produce sloppy work and that local adaptation of AM to meet the exact needs of your environment is important.

AM é baseado numa coleção de princípios, tais como assumir simplicidade quando você está modelando e **abraçar a mudança** conforme você está trabalhando, por causa que os requerimentos irão mudar com o passar dos ciclos iterativos. Você deveria reconhecer/aceitar que o variação incremental do seu sistema pelo tempo possibilita agilidade e que você deveria almejar em obter rápido retorno (feedback) no seu trabalho para se assegurar que isso reflete com precisão as necessidades do stackholders do projeto. Você deveria modelar com propósito, pois se você não sabe a razão pela qual você está produzindo esse artefato (modelo/documento) então você pode estár gastando esforços em algo desnecessário. Lembremos que a equipe deverá trabalhar junta, isso que dizer que usando de coragem (princípio) você aponta para a possibilidade de modelar/documentar artefatos $Art_1, Art_2, ..., Art_n$ então na reunião com a equipe, proponha isso e alguma sugestão, alguma retorno vai ter isso. Pior é se você não participar executando os princípios e práticas daquele método de desenvolvimento que a empresa custuma adotar para a construção de software e administração da equipe. Um conceito crítico é que modelos não são necessariamente documentos. Uma realização que possibilita que você se organize melhor, desempenhe melhor no seu trabalho, como analísta ou desenvolvedor, descartando a maioria de seus modelos uma vez que eles cumpriram sua missão. Modeladores ágeis acreditam que conteúdo é mais importânte que representação, que existem muitos caminhos para você modelar a mesma coisa, da maneira que a equipe vai pegar a ideia. Para ser um modelador de efeito, você precisa reconhecer que ser **aberto e honesto** na sua comunicação é comumente a melhor política a ser seguida, para asegurar eficássia. Finalmente foco em trabalho qualificado (eXtreme Programming por exemplo aponta que Pair Programming é muito eficaz pois o código sai 1: testado 2: revisado) é importante pois o gerênte de projeto e a equipe também não irão gostar de trabalho desqualificado, fora do padrão. Não dá para ficar arriscando quando se trata de competição no mercado de software. Adaptação é ponto vital (importânte) na modelagem ágil, isso permite abordar necessidades inusitádas vindas de dentro do ambiente (Bibliotecas melhores, Tecnologia diferente) ou externamente ao ambiente (Cliente).

IV. Work Item List: Disciplined Agile

A abordagem da figura 1 reflete o que se espera de uma construção ágil disciplinada de fila de Work Items. Essa técnica é explicada pela Entregas Ágeis Disciplinadas (Disciplined Agile Delivery - DAD). As vantagens da Lista

de Work Items são que facilmente suportam diferenças nas iterações, ou seja tipos diferentes de Work Items. E ”Overhead fudge factor”(Fudge factor é um hack para fazer com que um modelo qualquer se encaixe nas expectativas) requerido é muito menor ou não existente se comparado com a abordagem do SCRUM de product backlog. As desvantagens são: Ter que mander Work Items da fila priorizados paralelamente com prioridades sendo alteradas durante o decorrer da iteração. Requer múltiplas filas de Work Items, ou um esquema de priorização mais complexo, para suportar Classes of Service.

A. *Classes de Serviço*

David J. Anderson categoriza trabalho por risco e tipo e então associa isso com uma classe de serviço específica como: Classes para Work Items que irão salvar tempo no cronograma, classes para Work Items que violam copyright, classes com prioridade para Work Items de maior risco para o sistema. Ou seja a ideia é nenhum Work Item deveria ser tratado igual. Isso permite aos programadores entregar software com risco controlado, gerando maior fluxo de trabalho, e aumentando previsão do produto a cada iteração[2].

B. *Valores dos Riscos*

De valor aos riscos. Sobre isso podemos incluir o risco de precisar chegar a uma conclusão com os stakeholders o mais cedo possível do projeto. O risco pode ser melhor entendído se a equipe tem visão ampla e melhor ainda se os engenheiros estão acostumados com método ágil. Outro risco que podemos ligar é o risco de provar que a sua visão de arquitetura funciona. Como mostrar para o Cliente que funciona ? O jeito bastante bom é codificar uma arquitetura esqueleto. Desenvolvedores espertos irão olhar para os Work Items da fila, no começo do projeto e identificar riscos nos requisitos para já tentar criar alternativas para resolve-los já logo no primeiro Sprint. Caso o requisito risco, que foi identificado não está em cima da fila para ser resolvido primeiro ? Convoca uma reunião e mostra o por que daquele Work Item deveria estár na frente da fila, naturalmente que os desenvolvedores, analístas e stakeholders irão chegar a um ”concensus”.

C. *Modele um pouco a frente*

É bom muitas vezes para quando a fila for sendo esvaziada de Work Items, os próximos requisitos vão sendo mais fáceis de desmistificar. Se um Work Item é muito complexo vale a pena investir mais tempo nele, usando uma técnica chamada Look-Ahead também usando no SCRUM, mas com o nome de Backlog Grooming. Basicamente é você olhar para 2 3 Work Itens a frente e entender como aquele Work Item atual é complexo, o quanto ele é. Onde ele afeta ou não o sistema. Nisso você está trabalhando ágilmente sobre riscos. A figura 1 ilustra como a fila de Work Items funciona.

V. PRÁTICAS

Para modelar no espírito Ágil você deverá aplicar práticas ágeis de modelagem. Práticas fundamentais incluem criar modelagens em paralelo, aplicando os artefatos corretamente

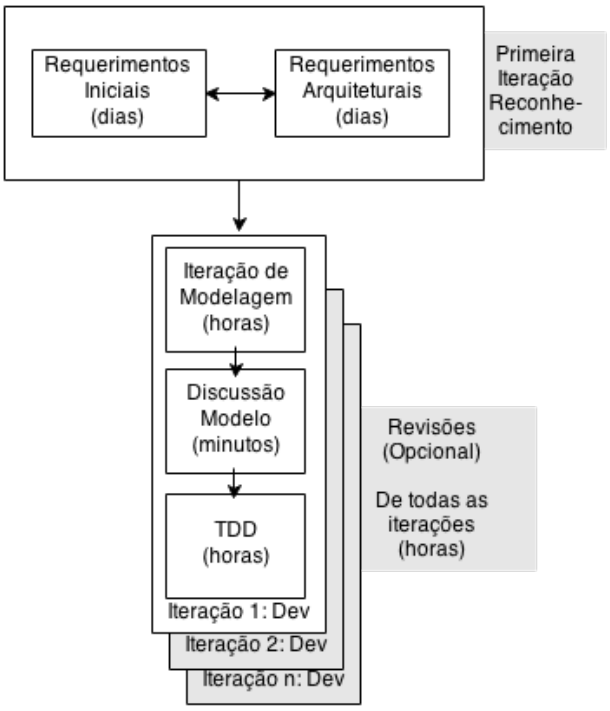

Figura 1. Esquematização do funcionamento básico de manipulação de ”Working Itens”.

para cada situação, interagir em outros artefatos para continuar movendo eles para frente até um estado aceitável, regular também é válido. Modelar em pequenos incrementos, conforme a necessidade é mais aceitável do que criar modelagens massivas de uma vez só, o que não tem boa aceitação. Até porque a equipe gostaria de escalar todos juntos, ao invéz de um faz a escalada sozinho, chega no topo e fica esperando a equipe nivelar com ele novamente.

A. *Representações Abstratas*

Modelos são apenas representações abstratas de software, abstrações que podem não ser precisar, você deveria buscar provar que seu **código** funciona como você o tinha idealizado anteriormente. Isso não tem mistério uma vez que você estuda TDD e não deixa o TDD apenas na teoria, mas se força a aplicar TDD, justamente para provar o que você construiu. Participação dos stakeholders é imprecindível para que a equipe tenha esforço gerando sucesso com as modelagens. Simplesmente por que os stakeholders do projeto sabem o que eles querem, e eles podem te retornar explicações do que eles querem. O princípio de assumir simplicidade é suportado pela prática de criar conteúdo simples, focando apenas no aspecto que você precisa para modelar e não tentar criar um extremamente super modelo detalhado.

B. *Configuração da Comunicação*

Comunicação atravéz de mural que mostra os modelos publicamente, atravéz de um website apenas para mostrar publicamente, como linha do tempo o que se tem feito, onde a equipe está, o que se planeja. Um exemplo é o Redmine

que é uma ferramenta gratuíta e aberta, de administração de projeto e ferramenta de perseguição de falhar no produto. Inclue calendário e diagramas Gantt que visam ajudar com representações visuais do projeto e suas deadlines.

VI. TDD e Agile Modeling

O primeiro passo é entender o TFD, Test First Development onde você adiciona rapidamente um test, e testa se ele passar você adiciona mais testes, se ele falhar você adiciona uma pequena mudança (note que a meta é adicionar teste que falha). E então roda os testes novamente, se falhar então mais uma pequena mudança e fica nesse ciclo até não falhar mais, e então se passarem. Você volta ao início e adiciona mais testes novos. **Sempre pensando em adicionar aos poucos. E modificar aos poucos.** Assim você tem entendimento do que você está fazendo. Com prática você irá programar mais rápido do que sair direto jogando código funcional. O ciclo do TFD acaba quando você voltou ao início dessa atividade e reconheceu que você terminou de implementar o que precisava. O TDD entra aqui. Quando você tem feito TFD e Refatorado. Essas operações juntas são o TDD sendo aplicado. Porém o TDD vai um pouco mais fundo, existem atividades de ATDD. ATDD é aceitação do TDD. É uma atividade que engloba o TFD. Onde você roda testes de aceitação para os seus testes de funcionalidade. Muitos programadores ignoram ATDD. Pois hoje em dia os Frameworks de Test, auxiliam muito nessa parte. Tecnologias que você pode estár abordando são: para ATDD RSpec, Fitnesse e para Agile TDD programadores poderão usar xUnit Frameworks. No site: http://agiledata.org você poderá encontrar mais sobre TDD e outras formas de desenvolvimento dirigido a testes, você poderá descobrir também que existem formas de modelagem de banco de dados e testes de transações com banco de dados tudo dirigido a testes.

A. Scalando TDD via Agile Model-Driven Development (AMDD)

TDD é uma ferramenta de especificação e validação. Mas não pensemos em usar TDD apenas sob grandes problemas de design, mas também como as pessoas irão usar o sistema (engenharia de usabilidade). Ou por exemplo: na interface de usuário. Modelar ou sendo mais específico utilizar Agile model-driven development (AMDD) junto com TDD para completar problemas como escalabilidade ágil, mas o AMDD implementa escalabilidade ágil (veja figura 2).

VII. Requisitos

Poderiamos dizer que o AM é uma medida ágil para modelagem, que o núcleo do AM é simplesmente uma coleção de práticas que refletem os princípios e valores que foram construídos por desenvolvedores de software muito experiêntes, que viram a necessidade de publicar isso. Com o Modelo de Desenvolvimento Ágil Dirigido (Agile Model Driven Development - veja figura 2 2). A abordagem é aplicar modelagem alto-nível no começo do projeto para entender o escopo e o potencial da arquitetura. E então durante o desenvolvimento das iterações você modela as partes da sua iteração planejando atividades e então aplica modelagem just-in-time (JIT) (na forma de ”brain storming”) onde você modela por muitos minutos antes de ficar por muitas horas codificando. Vamos entender essa imagem 2). No primeiro retângulo superior o que precisa ser entendido é que a equipe deveria: Identificar o escopo de alto-nível. Identificar a pilha de requisitos do sistema (iniciais). E identificar a visão arquitetural. Logo abaixo ao retângulo superior vemos as iterações de desenvolvimento onde deveríamos entender: Que modelagem é parte do esforço iterativo, modelar o suficiente para ter boas estimativas. É como se você estivesse em uma Rally. Você é o motorista e irá dirigir atravéz das iterações. E existe um navegador que entende o percurso para o piloto. Essa filosofia de fazer o caminho dando passos pensados é bastante a cara do método ágil. Você precisa também planejar o trabalho da equipe para aquela iteração (naturalmente). Trabalhar atravéz de problemas usando JIT. Stakeholders como participantes ativos do processo. Requisitos do sistema, evoluem com o desenrolar do projeto. Modelo o suficiênte para o momento, você poderá voltar e re-modelar depois. Construa software via através de test-first.

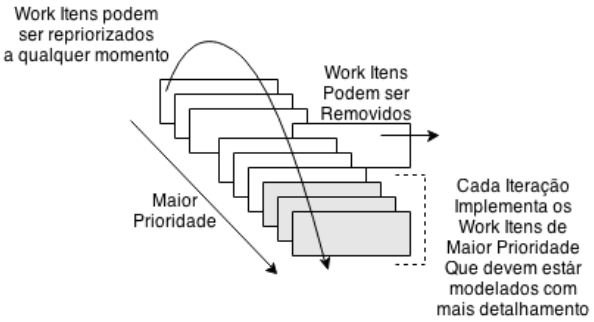

Figura 2. Esquematização do funcionamento básico de manipulação de ”Working Itens”.

Os valores da modelagem ágil, adotam e exportam do eXtreme Programming, que são: **comunicação, retorno (feedback), coragem, e humildade**. O objetivo é que você tenha comunicação efetiva. Como por exemplo telefones sempre a mão dos analístas de sistema, engenheiros de usabilidade, e até mesmo dos programadores, para que eles possam acionar rapidamente o cliente, ou a equipe. Se o cliente, então para poder desvendar alguma história de usuário (se você estiver trabalhando com SCRUM) ou ainda CRTCards (SCRUM ou eXtreme Programming ou FDD). A comunicação deve ser eficiente, rápida entre todos os stakeholders do projeto em ação. O motivo, obter feedback mantendo o resultado do esforço da equipe, constante durante o começo do projeto, que é onde muitas desmistificações acontecem. E também onde riscos do projeto são entendídas. Para ter a coragem de fazer e ter certeza de suas decisões, e também ter a humildade para admitir que você pode não saber tudo. Que outros tem valor para adicionar melhorias ao seu eforço em cima do projeto. Ou ainda, criticar o seu trabalho (esforço aplicado no projeto).

VIII. Conclusão

A modelagem ágil deve ser agregada a outros métodos de desenvolvimento. Deve-se entender que Agile Modeling não é para equipes grandes. E principalmente não vai ser aplicada a equipes sem experiência. Equipes maiores que 30 pessoas, é um risco a ser tomado. Equipes com analístas e programadores com menos de 2 anos de experiência, é outro risco a ser tomado. Programadores que não executaram projetos utilizando métodos ágeis, anteriormente na sua carreira, é mais um risco a ser tomado. O autor observa a possibilidade de integrar AM com eXtreme Programming. Eu particularmente, conhecendo XP, não vejo como. Mas todavia eu entendo um pouco a sugestão dele, que é de quando você tem uma equipe experiênte que se adapata ao XP mas eles precisam utilizar em alguns projetos boas quantidades de modelagem. Então Modele utilizando os princípios do AM pois você não irá perder nada com isso, você vai estár dentro do método ágil. Agora UP e AM eu vejo como eles poderiam se unir pois UP é bastante flexível e UP aborda a idéia de que software deve ser entregue iterativamente e incrementalmente que é a mesma idéia de AM. E UP aborda explícitamente disciplinas de modelagem e é simples em identificar onde as práticas do AM poderiam ser usadas para melhorar o UP.

Nota

Lembrando ao leitor que esse trabalho, na forma de artigo. É um estudo à AM (Agile Modeling). O material contido nesse artigo está contido no site do autor desse livro na referência. A maioria do texto nesse artigo são traduções do que está contido nos sites agilemodeling.com e agiledata.org que pertencem ao autor Scott Ambler. Já as figuras desse artigo, foram refeitas no site draw.io simplesmente para não copiar diretamente as figuras do site. Mas elas também foram baseadas nas figuras do site agilemodeling.com.

Referências

[1] Ambler, Scott W. and Lines, Mark, "Disciplined Agile Delivery: *A Practitioner's Guide to Agile Software Delivery in the Enterprise*" 1ed. IBM Press, 2012. Website: http://www.agilemodeling.com.

[2] "Classes of Service and Policies" Posted on June 10, 2009 by dp-joyce. Website: http://leanandkanban.wordpress.com/2009/06/10/classes-of-service-and-policies/