Daniel Koudi Nakano

Modelo de desenvolvimento de software em pequenos grupos de pesquisa

Florianópolis, SC
2012

Daniel Koudi Nakano

Modelo de desenvolvimento de software em pequenos grupos de pesquisa

Trabalho de conclusão de curso apresentado como parte das atividades para obtenção do título de bacharel em Ciências da Computaçãoda Universidade Federal de Santa Catarina- UFSC, Departamento de Informática e Estatística- INE.

Orientador:
Prof. Dr. Julibio David Ardigo
Co-orientador:
Prof. Dr. Ricardo Felipe Custódio

Florianópolis, SC
2012

Daniel Koudi Nakano

Modelo de desenvolvimento de software em pequenos grupos de pesquisa

Trabalho de conclusão de curso sob o título *"Modelo de desenvolvimento de software em pequenos grupos de pesquisa"*, defendida por Daniel Koudi Nakano.

Trabalho aprovado. Florianópolis, SC, 21 de Dezembro de 2012:

Prof. Dr. Julibio David Ardigo
Orientador
Universidade Estadual de Santa Catarina-UDESC

Prof. Dr. Ricardo Felipe Custódio
Coorientador
Universidade Federal de Santa Catarina-UFSC

Prof. Dr. Júlio da Silva Dias
Universidade Estadual de Santa Catarina-UDESC

Prof. Me. Omar Abdel Muhdi Said Omar
Universidade Estadual de Santa Catarina-UDESC

Florianópolis, SC
2012

"*So once you know what the question actually is, you'll know what the answer means*"
Deep Thought - Douglas Adams

Resumo

Este trabalho faz uma proposta de um modelo de desenvolvimento de *software* aderente ao processo de desenvolvimento dentro de pequenos grupos de pesquisa, onde, muitas vezes, as condições de desenvolvimento são divergentes em relação ao desenvolvimento comercial tradicional do qual tratam os modelos de desenvolvimento de *software* conhecidos no mercado. Apresenta uma breve análise de alguns dos modelos de desenvolvimento existentes e, com base nestes modelos e nas necessidades do LabTIC - ESAG - UDESC, apresenta um modelo aderente as necessidades de pequenos grupos de pesquisa. O modelo proposto foi aplicado no desenvolvimento de uma solução dentro do próprio laboratório onde foi desenvolvido tendo, ao seu final, alcançado os objetivos desejados.
Palavras-chaves: Modelo. Desenvolvimento. Software. Latex. Waterfall. RAD. SCRUM. VMODEL. XP.

Abstract

This work presents a software development model which can be adopted by small researcher groups, where, many times, the development environment is different from the usual industrial environments, which is usually the target of the well known software development models. The work shows a brief analysis of some of these software development models and the needs of LabTIC - ESAG - UDESC. With this, presents a model that can be adopted by these small researchers groups. The model presented in this work was used to develop a project inside the group itself and in the end the results were satisfactory.
Palavras-chaves: Model. Development. Software. Latex. Waterfall. RAD. SCRUM. VMODEL. XP.

Lista de ilustrações

Figura 1 Fluxograma da proposta de modelo 28
Figura 2 Adição de nova funcionalidade na lista de objetivos 29

Figura 3 BPMN para desenvolvimento de modelo de fluxo de projetos 31
Figura 4 Diagrama de atividade UML para desenvolvimento de modelo de fluxo de projetos 32
Figura 5 BPMN para atribuição de projeto a um aluno 33
Figura 6 BPMN para adicionar arquivo de exemplo da etapa 33

Figura 7 modulo 49
Figura 8 Visualização dos projetos de um modelo 50
Figura 9 Visualização dos projetos B 51
Figura 10 Criação de modelo - passo 1 51
Figura 11 Criação de modelo - passo 2 52
Figura 12 Criação de um novo modelo 52
Figura 13 Edição do fluxo do modelo 53
Figura 14 Edição do fluxo do modelo - alterar nome de uma etapa 53
Figura 15 Edição do fluxo do modelo - salvar alteração 54
Figura 16 Edição do fluxo do modelo - Apresentação do fluxo alterado 54
Figura 17 Edição do fluxo do modelo - Adição de uma nova etapa 55
Figura 18 Edição do fluxo do modelo - Fluxo com o novo nodo 55
Figura 19 Edição do fluxo do modelo - Criação de subetapas 56
Figura 20 Edição do fluxo do modelo - Edição de etapa com a subetapa 56
Figura 21 Edição do fluxo do modelo - Fluxo de subetapas 57
Figura 22 Edição do fluxo do modelo - Etapa com subetapas 57
Figura 23 Edição do fluxo do modelo - Exclusão de uma etapa desnecessária . . . 58
Figura 24 Edição do fluxo do modelo - Itens disponíveis para inclusão nos projetos 59
Figura 25 Criação de projeto 60
Figura 26 Criação de projetos B 60
Figura 27 Edição dos dados do projeto - Ferramenta disponível 61
Figura 28 Edição dos dados do projeto - Alteração do título (passo 1) 61
Figura 29 Edição dos dados do projeto - Alteração do título (passo 2) 62
Figura 30 Edição dos dados do projeto - Alteração do título (passo 3) 63

Figura 31 Edição dos dados do projeto - Edição de texto de outros elementos do documento e informações sobre o projeto 64
Figura 32 Edição dos dados do projeto - Etapas 65
Figura 33 Edição dos dados do projeto - Subetapas 65
Figura 34 Edição dos dados do projeto - Observações do orientador (passo 1) . . 66
Figura 35 Edição dos dados do projeto - Observações do orientador (passo 2) . . 66
Figura 36 Edição dos dados do projeto - Finalização do projeto 67
Figura 37 Edição dos dados do projeto - Observação de conclusão 67
Figura 38 Edição dos dados do projeto - Finalização do projeto 68
Figura 39 BPM - Orientar projeto 68
Figura 40 BPM - Criar fluxo de desenvolvimento de modelo 68
Figura 41 BPM - Adicionar referências e citações 69
Figura 42 BPM - Definir textos de elementos textuais da monografia 69
Figura 43 BPM - Edição do projeto 69
Figura 44 BPM - Acompanhar a composição do texto 69
Figura 45 BPM - Finalizar projeto 70

Lista de Tabelas

Tabela 1 Comparação de produção dentro da acadêmia e na indústria 14

Sumário

Lista de ilustrações **6**

Lista de Tabelas **8**

Sumário **9**

1 Introdução **11**
1.1 Contextualização 11
1.2 Objetivos 13
1.2.1 Objetivo geral 13
1.2.2 Objetivos específicos 13
1.3 Delimitações 13
1.4 Justificativa 13
1.5 Metodologia 15
1.6 Organização da obra 16

2 Fundamentação teórica **17**

3 Modelos **19**
3.1 Modelo em Cascata (Waterfall Model) 19
3.2 Incremental 20
3.3 Modelo Espiral (*Spiral Model*) 20
3.4 Rapid Application Development - RAD (Desenvolvimento Rápido de Aplicações) 21
3.4.1 XP 22
3.4.2 SCRUM 22
3.5 V-model 23

4 Proposta de modelo **25**
4.1 Levantamento de requisitos 25
4.1.1 Ambiente de desenvolvimento 25
4.1.2 Necessidades do laboratório 26
4.2 Proposta de modelo 26

5 Aplicação do modelo **30**

6 Considerações Finais **35**

Referências **37**

APÊNDICE A Artigo - Modelagem de Projetos **39**

APÊNDICE B Manual Magi - Casper **47**

B.1 Apresentação 47
B.1.1 Objetivo do Magi-Casper 47
B.1.2 Funcionamento 47
B.1.2.1 Criação de modelos 49
B.1.2.2 Atribuição de projetos 58
B.1.2.3 Desenvolvimento do projeto e interação com orientador 60
B.1.2.4 Finalização de projeto 63
B.1.3 Funções 65
B.1.4 Resultado esperado 67
B.2 Requisitos para funcionamento 69
B.2.1 Cliente 69
B.2.2 Servidor 70
B.2.2.1 Modelo de documento 71

1 Introdução

1.1 Contextualização

O desenvolvimento adequado de *software* não é uma atividade trivial. O grau de dificuldade é incrementado quando os mesmos são desenvolvidos em laboratórios de pesquisa cujo escopo principal não é o desenvolvimento de *software* em si. Neste caso, além da falta de uma estrutura e metodologia adequada, as constantes mudanças de escopo podem comprometer o prazo, o orçamento e aumentar o risco de abandono do sistema antes de sua conclusão ou da não adoção mesmo após ser finalizado (BARBARA, 1987).

As metodologias de desenvolvimento existentes no mercado podem ser de difícil adoção por parte dos grupos de pesquisa, pois o escopo dos projetos de Pesquisa e Desenvolvimento (P&D) podem ser de variadas naturezas, tais como, projetos de pesquisa financiados por terceiros (entidades públicas ou privadas), projetos de extensão, teses, dissertações, trabalhos de conclusão de curso (TCC) etc.

O processo de desenvolvimento de *software* diferencia-se de outras cadeias produtivas por não apresentar passos de manufatura, não necessitar de matéria prima e, muitas vezes, ao final do projeto de P&D, já disponibilizar um produto funcional (BARR; TESSLER, 1996).

Além da diferença de desenvolvimento do *software* em relação a outros produtos há também, diferenças entre o processo produtivo convencional e o que ocorre dentro de um ambiente de grupos de pesquisa de P&D. Estes dois ambientes, em geral, possuem diferenças operacionais e estruturais e mesmo podendo apresentar os mesmos problemas, a relevância destes podem não ser as mesmas e consecutivamente o seu tratamento também deve variar (PAIVA; TURINE; FORTES, 2008).

Em geral, quando um grupo de pesquisa visa desenvolver um *software*, o faz com objetivos de materializar uma pesquisa e e embora possa haver expectativas de que o *software* desenvolvido venha a produzir um protótipo de uma solução maior, muitos acabam sendo utilizados apenas nos projetos que os viabilizaram.

Programas de computador comerciais são produzidos de acordo com regras de boas práticas e modelos de desenvolvimento onde existem processos para cada etapa da produção e regras para manter o controle de qualidade de acordo com normas, nacionais ou internacionais. Embora este seja o cenário adequado, as inúmeras regras que estes modelos apresentam somada às próprias características dos pequenos grupos de pesquisa, como alto *turnover*, troca constate de escopo e tempo limitado, podem inviabilizar ou descaracterizar o uso do modelo, consequentemente a sua utilização pode vir a ser descartada

total ou parcialmente.

Um exemplo típico do cenário descrito é um *software* sendo desenvolvido por um pesquisador de um pequeno grupo de pesquisa multidisciplinar. Somente este pesquisador, com a ajuda de, no máximo, um bolsista, está envolvido na produção do *software*. Por não ter domínio pleno do problema, ou, pelo escopo do problema alterar constantemente, acaba não conseguindo formular uma documentação adequada e, por ventura, ao sair da equipe, o *software* poderá ser abandonado por exigir muito esforço e tempo para aprender e dominar as tecnologias utilizadas ou reiniciado pelo pesquisador que veio a substituí-lo por ser difícil entender o código existente. O uso de um modelo de desenvolvimento de *software* tenta sanar estes problemas, contudo, ao ser aplicado poderá comprometer a viabilidade ou os prazos do projeto.

É possível perceber, neste exemplo, que existem problemas quanto aos recursos humanos, a falta de um escopo bem definido e uma documentação inadequada. Podem ainda, existir, outros problemas relacionados ao tempo, pois dificilmente um pesquisador pode dedicar-se exclusivamente a uma função dentro de um projeto, Além disso, a duração do projeto pode estender-se além do tempo de permanência do pesquisador dentro da instituição.

Dentro do desenvolvimento industrial estes problemas são amenizados por haver uma equipe de desenvolvimento com menor rotatividade entre os membros, um escopo e etapas bem definidos e o programador pode dedicar-se ao *software* durante todo o seu tempo de trabalho dentro da empresa. Por apresentar um cenário mais regrado, a utilização dos modelos de desenvolvimento torna-se viável. E conforme o processo e a empresa amadurecem podem haver algumas alterações no modelo para adequá-lo a filosofia da empresa e ao ritmo da equipe. Há contudo, tentativas de viabilizar projetos de pesquisa da indústria em conjunto com o meio acadêmico. Definido-se uma estrutura organizacional onde a equipe, formada por especialistas de ambas as partes, possa desenvolver o projeto e todos contribuírem com o desenvolvimento (CHANG; TRUBOW, 1990).

Este trabalho visa apresentar uma metodologia para o desenvolvimento de *software* dentro destes grupos de pesquisa para tentar mitigar os problemas causados pela falta de tempo, alteração constante de escopo, troca dos integrantes da equipe entre outros problemas. A metodologia a ser apresentada deriva do estudo de modelos desenvolvimento comerciais e da literatura existente, confrontado com as práticas adotadas pelo Laboratório de Tecnologia da Informação e Comunicação (LabTIC) do Centro de Ciências da Administração e Socioeconômicas (ESAG) da Universidade do Estado de Santa Catarina (UDESC).

1.2 Objetivos

1.2.1 Objetivo geral

Definir um modelo de desenvolvimento de *software* que seja passível de adoção por grupos de pesquisa a fim de aumentar as chances de finalização do *software*, do desenvolvimento sobreviver à rotação dos integrantes da equipe e ser partilhável com outros grupos.

1.2.2 Objetivos específicos

Para atingir-se o objetivo principal é fundamental que se atinja os seguintes objetivos:

- Levantar os modelos de desenvolvimento de *software* mais utilizados no mercado;
- Averiguar quais procedimentos são fundamentais em cada modelo;
- Inferir quais destes procedimentos são aderentes ao desenvolvimento de *software* no contexto dos pequenos grupos de pesquisa;
- Propor um modelo de desenvolvimento de *software* aderente ao contexto previsto;
- Validar o modelo proposto aplicando-o dentro de um pequeno projeto para averiguar os custos de implementação do modelo;

1.3 Delimitações

O presente estudo não irá abranger todos os modelos de desenvolvimento de *software* existentes, sendo estudados somente os que apresentarem alguma relevância para o desenvolvimento no contexto acadêmico e amplamente difundidos. Neste contexto, entende-se por relevância a flexibilidade do modelo frente ao ambiente no qual está sendo inserido.

Adicionalmente, devido ao fato do modelo ser testado em um único projeto de um laboratório, não é possível assumir que sua utilização será adequada em qualquer grupo e situação.

1.4 Justificativa

Segundo (LIU; XU; BROCKMEYER, 2008), caracteristicamente no meio acadêmico os pesquisadores fazem todo o projeto sozinho, pois este não tem muita experiência com projetos grandes e dificilmente apresentam a habilidade para trabalhar em equipe. Todavia, fora do meio acadêmico é usual os programadores trabalharem como um time. Dentro da academia os pesquisadores têm outras tarefas além de programar enquanto fora dela ele

pode se focar neste trabalho durante toda sua carga diária. Essas são algumas das diferenças quanto ao pesquisador em relação ao programador fora da academia (LIU; XU; BROCKMEYER, 2008).

Ainda diferenciando os ambientes a tabela 1 mostra um comparativo entre o desenvolvimento de *software* dentro e fora da academia.

Tabela 1 – tabela adaptada de artigo *Investigation on Academic Research Software Development* (LIU; XU; BROCKMEYER, 2008)

Projetos		**Projetos de *software* em pesquisas acadêmicas**	**Projetos de *software* na indústria**
Propósitos	**Propósito**	orientado à pesquisa as vezes, sem qualquer benefício prático	benefícios práticos
	Contribuição científica	pesquisas matemáticas intrigantes, ideias novas com contribuição científica	normalmente sem contribuição científica
Pessoas	***Expertise* do programador**	pouca / mediana	mediana / alta
	Controle da equipe	pouca cooperação	hierárquico
	Troca do programador	trocas normalmente previsíveis	trocas imprevisíveis
	Projeto (design)	professor normalmente não interfere	*designer* de *software*
Processos	**Processo de desenvolvimento**	normalmente, ágil	Várias: *waterfall*, ágil
	Reuniões da equipe	raras, ou quando necessário	regulares
	Controle do código	depende do pesquisador, costumeiramente não utilizada	um sistema de controle
	Recursos	poucos recursos necessários	grande relevância
	Testes	sem uma rotina sistemática de teste	por equipe de teste
	Prazos	brando, as vezes indefinido	prazo limite definido, com 70% de atrasos
Produtos	**Atores do *software***	faceta única	normalmente multi-facetado
	Tamanho e complexidade do *software*	de pequena a grande, os algoritmos podem ser complexos	de média a grande, ambos, design e implementação podem ser complexas
	Interface	sem muita preocupação com a interface	mais requisitos na interface
	Documentação	pouca	normalmente completa

É possível notar ao visualizar a tabela 1 que no desenvolvimento de *software* dentro da acadêmia existem menos controles e acompanhamento tanto do projeto quanto dos envolvidos nele.

Para a elaboração da tabela foram entrevistados 10 pesquisadores em projetos de porte médio em diferentes áreas, e durante a entrevista constatou-se que mesmo em equipes de 2 a 3 integrantes os programadores pouco discutem entre si mesmo que suas partes dependam das dos outros e raramente sabem o que os outros estão fazendo (LIU; XU; BROCKMEYER, 2008).

Comercialmente, as equipes de desenvolvimento de *software* apresentam certa estabilidade no que diz respeito a rotação dos integrantes, carga horária certa para trabalharem com o *software* e um escopo definido, considerando que ou trabalham em produtos fechados para venda ou então em solução de algum problema específico, quando saem dessas opções normalmente estão indo para o lado da pesquisa e os problemas se encontram com os encontrados na acadêmia (GILB, 1985). No ambiente comercial estável a aplicação de modelos de desenvolvimento ajudam a organizar e estruturar o trabalho ajudando tanto no desenvolvimento quanto no controle (BARBARA, 1987).

Mas, apesar de existir inúmeros modelos de desenvolvimento de *software*, nenhum parece aceitável quando o escopo do *software* é volátil o que torna a aplicação dos modelos, principalmente no que tange à documentação, pesarosa e de difícil manutenção. Quando

o projeto exige velocidade no desenvolvimento do *software*, a documentação tende a ficar ainda mais difícil de ser mantida, é importante então averiguar qual a validade de tal documentação e a real necessidade dela desde que o *software* fique funcional.

Estudar a viabilidade técnica de modelos estabelecidos comercialmente para o desenvolvimento de *software* dentro do ambiente de um laboratório de pesquisa, poderá levar a conclusão que uma simples adequação no modelo poderá ser suficiente para adoção do mesmo dentro deste ambiente.

Entretanto, mesmo com esta adequação, o modelo pode ser insuficiente se a carga do trabalho for excessiva, ou ainda, se o pesquisador não acreditar no modelo. Pode também, apresentar-se impraticável pela falta de tempo para a sua adoção.

1.5 Metodologia

Para a elaboração do modelo de desenvolvimento de *software* que esta obra visa propor, as seguintes etapas serão executadas: Estudo de modelos de desenvolvimento de *software* existentes, levante das necessidades que o laboratório apresenta no que se refere ao ciclo de desenvolvimento de um projeto, elaboração da proposta e teste do modelo proposto.

Etapa 1 - Estudo de modelos de desenvolvimento de *software* existentes

Nesta fase, serão averiguados, os modelos de desenvolvimento de *software* existentes que apresentem pontos relevantes ao desenvolvimento de *software* por pequenos grupos de pesquisa, baseando-se as necessidades deste laboratório (LabTic - ESAG)

Etapa 2 - Levante das necessidades que o laboratório apresenta no que se refere ao ciclo de desenvolvimento de um projeto

Esta fase deve ocorrer de forma quase concomitante à primeira, avaliando-se quais são os pontos relevantes quando um novo projeto é elaborado e desenvolvido. Conhecimentos empíricos de pesquisadores do laboratório podem ser utilizados para julgamento de importância de cada item levantado.

Etapa 3 - Elaboração da proposta

Através dos dados coletados nas duas etapas anteriores, será formulado um modelo que possa cobrir estes pontos sem comprometer a estrutura de trabalho atual, ou se for o caso, descrever qual o motivo da impossibilidade de tal modelo de desenvolvimento de *software*.

Etapa 4 - Teste do modelo proposto

Esta fase ocorre apenas se o modelo conseguir ser proposto. Seu objetivo é averiguar se a proposta de modelo é aplicável e avaliar os custos de sua aplicação.

1.6 Organização da obra

A presente obra está estruturada em 6 capítulos, onde o primeiro capítulo trata da introdução do tema, define o contexto da obra assim como os aspectos nos quais desenvolve-se o projeto, tratando dos objetivos, delimitações e a justificativa do mesmo.

O segundo capítulo apresenta os conceitos sobre os quais esta obra está embasada, estendendo ao terceiro capítulo, um detalhamento de cada um dos modelos estudados.

O modelo de desenvolvimento, foco deste projeto, é apresentado no quarto capítulo, junto com uma caracterização do ambiente e necessidades do laboratório no qual está inserido.

O quinto capítulo trata da aplicação do modelo com objetivo de testar a viabilidade da proposta de modelo.

Finalmente, o sexto capítulo, apresenta uma conclusão sobre o modelo e sua aplicação.

2 Fundamentação teórica

Um *software* pode ser feito de modo simples e sem seguir qualquer regra e ao final do processo é possível que o programa resultante atenda as necessidades mas sem qualquer garantia, tanto da qualidade do *software* quanto do resultado advindo dele. Contudo, através de uma prática metodológica onde pressupõe-se um caminho a ser seguido, existem premissas que devem ser levadas em consideração e o resultado final é planejado, as chances do programa resultante alcançar seus objetivos aumentam em relação ao primeiro cenário. Os modelos podem seguir variadas estruturas, incremental, cascata, espiral, prototipação, entre outros, e estes modelos podem ser combinados e continuam a evoluir de acordo com a necessidade (GAO, 2010).

Existem comercialmente modelos de desenvolvimento de *software* que visam mostrar este caminho, determinam técnicas para identificar as premissas, detalhar o resultado esperado, aplicar testes, identificar papéis, ações, regras etc.

Estas metodologias de desenvolvimento frequentemente acabam resultando em *framework* de desenvolvimento, tal como o *Rational Unified Process* (RUP), ou, em tradução livre Processo Unificado Racional, que visa padronizar o desenvolvimento de *software* a fim de ter um controle deste processo, ajudar na comunicação entre os membros da equipe ou entre as equipes do projeto e permitir a identificação e mitigação dos riscos de projeto.

O RUP é um *framework* desenvolvido pela *Rational Software Corporation*, mais tarde incorporado a IBM (*International Business Machines Corporation*), considerado uma metodologia tradicional, é uma implementação do modelo de desenvolvimento de Processo Unificado (UP - *Unified Process*), vem sendo aprimorado desde o começo da década de 90 e abrange várias etapas de desenvolvimento de *software* sendo dividido em 4 grandes fases, concepção, elaboração, construção e transição, onde cada fase é focada em um resultado.

Na concepção procura-se definir, o ambiente, os objetivos, os custos tudo em acordo entre os envolvidos, uma fase curta para verificar a viabilidade do projeto. A elaboração, fase de projeto, é voltada para a modelagem e identificação mais detalhada dos requisitos. A construção, é a fase de implementação do código e testes. Finalmente na transição, fase de entrega do *software*, é onde é feita a implantação do *software*, treino no uso do mesmo, assim como acompanhamento para controle de qualidade.

Considerado uma metodologia pesada e longa acaba apresentando dificuldades quando aplicada em projetos de porte pequeno e médio porém bastante adequada a projetos grandes devido ao seu ciclo, o número de processos dentro de cada fase e a documentação produzida. Há contudo, várias tentativas de adaptação do RUP para que este adeque-se a projeto menores (BORGES; MONTEIRO; MACHADO, 2011) (AHMED; CAPRETZ, 2008).

Em contrapartida às metodologias tradicionais como o RUP surgiram as metodologias ágeis, que apresentam como foco, velocidade no desenvolvimento, por exemplo, o *Extreme Programming* ou simplesmente XP. Este, apresenta uma estrutura onde o próprio código deve ser auto-explicativo, reduzindo a necessidade da produção de documentação, para isso há um estimulo à programação pareada (*pair programming*), que tenta fazer com que o código a seja melhor descrito, pois precisa ser entendido por ambos os programadores.

Contudo, na programação XP o desenvolvimento rápido demanda, muitas vezes, um domínio grande do escopo para que o projeto venha a ser finalizado com sucesso, pois propõe-se que o desenvolvimento seja voltado á solução. Nesta situação, a falta de conhecimento do processo pode levar a modificações muito bruscas e quando estas mudanças ocorrem em estágio avançado do desenvolvimento pode levar o projeto ao fracasso.

Tanto o UP quanto o XP seguem um ciclo de vida de desenvolvimento de *software* incremental onde o projeto pode ser desenvolvido de uma forma iterativa mais fácil do que nos modelos que seguem um ciclo de vida de desenvolvimento em cascata (*waterfall*), em que pressupõe-se que o projeto deve ser inteiramente planejado antes de começar a ser construído e todo o desenvolvimento planejado deve ser construído, e havendo mudanças todo o projeto é refeito desde o seu início. O modelo da prototipação consiste em rápida e repetidas implementações com o objetivo de entender os requisitos do cliente. O modelo orientado a componentes tenta fazer com que o desenvolvimento seja desmembrado em componentes para que estes possam ser reutilizados mais tarde (GAO, 2010).

Segue no próximo capítulo uma abordagem mais aprofundada sobre cada um dos modelos estudados.

3 Modelos

Modelos de desenvolvimento de *software*, têm como objetivo descrever o que é necessário fazer para que um *software* seja produzido, não lhes cabendo dizer como fazer. Como implementar um modelo é trabalho de um *framework* de desenvolvimento, muitas vezes estes *frameworks* levam o mesmo nome do modelo em si mas podem haver diferentes *frameworks* para um mesmo modelo.

Os modelos podem ser categorizados em lineares, iterativos e mistos onde, nos lineares as etapas são sequenciais e o final de uma etapa leva ao inicio da próxima e ao final de todas as etapas o *software* está pronto, enquanto nos iterativos é possível que ao final do último passo volte-se ao inicial revisitando cada etapa ou alguma em especial para alguma modificação. Nos mistos segue-se de modo geral a linha dos modelos iterativos e em certo momento é definido que se siga o linear para que seja possível ter uma versão do sistema (RUPARELIA, 2010).

3.1 Modelo em Cascata (Waterfall Model)

Considerado o primeiro modelo de desenvolvimento de *software*, foi inicialmente documentado por Benington (BENINGTON, 1987), e posteriormente adaptado por Royce, na década de 70, (ROYCE, 1987; RUPARELIA, 2010). Apresenta basicamente 7 passos: análise operacional, especificação operacional, especificações de projeto e codificação, desenvolvimento, teste, entrega e avaliação. Na adaptação de Royce, ele sugere que se faça um *feedback*, para evitar os problemas de projeto que não foram previstos.

Vale lembrar que na época de Benington, a programação era feita em fita e os computadores eram máquinas que precisavam ser previamente agendadas para utilizá-las. Desenvolver e testar um *software* nessas condições exigia um esforço bem maior do que atualmente e problemas de compilação e lógica exigiam muito mais esforço para identificação.

Por essa razão a documentação se faz essencial para identificação dos objetivos, características, entre outras informações. Royce sugere que ao menos 6 tipos de documentos devem ser produzidos:

1. Documento de especificação de requisitos;
2. Projeto preliminar;
3. Especificações de design de interface;

4. Projeto definitivo;

5. Plano de testes;

6. Manual de operação ou instruções.

A produção destes documentos ajudam o cliente a controlar a produção, os desenvolvedores a saber o que deve realmente ser produzido, projetar e realizar os testes. Facilitam a manutenção e utilização do programa pelo usuário. E ao realizar aprimoramentos, no futuro, ajuda a identificar onde devem ou podem ser realizados as mudanças e ou reprojeto. (ROYCE, 1987)

Este modelo foi desenvolvido pensando em projetos de grande porte e foi a base para grande parte dos modelos que vieram a surgir posteriormente.

3.2 Incremental

Modelo desenvolvido como uma forma iterativa do modelo *waterfall*. O projeto tende a ser divido tomando como base suas funcionalidades, tornando cada projeto resultante um projeto menor e mais fácil de implementar. Apresenta vantagens em relação ao modelo *waterfall* tradicional pois *feedbacks* de iterações anteriores podem ser aproveitadas nas iterações posteriores, aproximando-se do modelo espiral. Os envolvidos podem interagir durante cada iteração e ajudar a identificar os riscos mais cedo. Entrega-se rapidamente uma primeira versão pois esta versão apresenta um pequeno escopo, este escopo é incrementado a cada iteração ampliando as características do programa à cada rodada. E, por ser incremental, permite que as mudanças sejam monitoradas e os problemas isolados e resolvidos com menor esforço (RUPARELIA, 2010).

/A entrega de uma primeira versão é adiantada uma vez que o escopo reduz ampliando a cada iteração. E por ser incremental, permite que as mudanças sejam monitoradas e os problemas isolados e resolvidos com menor esforço (RUPARELIA, 2010).

3.3 Modelo Espiral (Spiral Model)

Modelo baseado no *waterfall*, apresentado por Boehn este modelo tem como principal objetivo aumentar a produtividade na produção do *software* com a restrição de manter os custos em um patamar aceitável. É um modelo voltado as pessoas e exige alto grau de comprometimento, os papéis devem ser bem definidos, apresenta uma estrutura pouco aderente a mudanças e os planos para as futuras iterações devem ser feitas com certa antecedência ao final da etapa atual (BOEHM, 1986). [1]

[1] Em seu artigo *A spiral model for software development and enhancement* Boehn mostra a estrutura típica que o modelo deve apresentar.

O modelo pode ser dividido em 4 fases por iteração, onde, o término dessas fases seria um marco para entrega ou avaliação do projeto. A primeira fase tem como objetivo identificar os objetivos, a segunda analisar alternativas para a solução do problema, identificar riscos e mitigá-los a medida do possível, a terceira desenvolver e testar e a quarta e última planejar a próxima iteração.

A cada ciclo completo, tem-se um protótipo do projeto com certas funcionalidades desenvolvidas e validadas. Caso os riscos diagnosticados na segunda fase sejam de ordem de desenvolvimento então o caminho a seguir é o do modelo *waterfall* para que estes possam ser tratados. Caso os riscos sejam, apenas de ordem de performance, segue-se para o próximo passo do modelo espiral, garantindo que o protótipo produzido anteriormente apresentem o menor risco possível associado.

Posteriormente foram sugeridas alterações no modelo que possibilitam um melhor controle do ciclo de desenvolvimento. A alteração proposta por Iivari no que chamou de modelo espiral hierárquico (IIVARI, 1987), consistia em subdividir cada uma das fases em duas para que haja um planejamento para cada etapa e possam ser estabelecidos metas e prazos para cada etapa.

Uma vantagem do modelo espiral é o seu trabalho em volta do risco mas sua adoção requer que os projetos tenham um controle de projeto que consiga se adaptar a ele, os contratos precisam ter certo nível de flexibilidade e depende do analista conseguir identificar os riscos corretamente para não afetar o desenvolvimento (RUPARELIA, 2010).

3.4 Rapid Application Development - RAD (Desenvolvimento Rápido de Aplicações)

Também conhecido como desenvolvimento ágil de aplicações é outro modelo que utiliza-se de protótipos para identificação de requisitos e validação, desenvolvida, principalmente, por Martin, é semelhante ao modelo de desenvolvimento de *software* de código aberto (*open source software development model*), também conhecido como *the cathedral and bazaar model* (RAYMOND, 1999), que segue a filosofia do "Libere cedo; Libere frequentemente; Ouça o seu cliente" (*Release early; Release often; Listen to your customers*).

O modelo utiliza-se do protótipo para unir todos os atores envolvidos no processo em um desenvolvimento iterativo. Onde os clientes são incentivados a participar da prototipação e dos testes unitários (*Unit Tests*) para que possam ser identificados o maior número de requisitos nas fases iniciais do desenvolvimento, e os programadores possam se familiarizar com o fluxo das regras de negócio (RUPARELIA, 2010).

Focados no desenvolvimento do produto, com alta produtividade e alta adaptatividade, seu uso torna-se inadequado quando o escopo é impreciso, é necessário um grande domínio de ao menos um integrante da equipe, normalmente o gerente do projeto, para que a implementação seja bem sucedida.

A integração da equipe também é importante, quanto maior o entrosamento da equipe mais fluído é o processo de desenvolvimento, por singularizar os problemas a troca de membros pode ser feita sem muitos problemas, mas a velocidade do processo quando um novo membro é incluído pode acarretar em um processo mais longo, devido a necessidade de adaptação do novo membro dentro da equipe.

3.4.1 XP

Também segue a filosofia RAD. O desenvolvimento na metodologia XP (*eXtreme Programming*) é o que apresenta a menor necessidade de documentação pois um código em XP bem feito deve, ser auto explicativo.

A programação XP é bem aderente ao Modelo Orientado a Objetos, mas como a abstração é pessoal o que pode ser considerado um código auto-explicativo para um desenvolvedor, para outro pode não ser tão claro.

Ela é bem empregada em equipes bem pequenas onde o grupo está habituado com o código dos companheiros ou em projetos individuais. Como desenvolve somente o necessário, quando preciso, é extremamente eficiente na resolução de casos específicos e que apresentam tempo bem limitado, para estruturas de sistemas complexos e com equipes grande acaba necessitando de bastante trabalho para que fique adequado as necessidades.

3.4.2 SCRUM

Metodologia com alta aderência a mudanças (RISING; JANOFF, 2000; ABRAHAMSSON et al., 2002), o scrum[2] apresenta "rodadas"de desenvolvimento chamados "*Sprints*". Dentro de um *sprint* há uma reunião chamada *Planning Meeting* (encontro de planejamento), nesta reunião toda a equipe deve participar para decidir o que será desenvolvido e quais os parâmetros de métrica para decidir se o objetivo foi alcançado, sugere-se que a reunião seja feita a cada duas semanas com isso a reunião deve durar no máximo quatro horas (SUTHERLAND; SCHWABER, 2012a).

Dentro destas duas semanas é realizado diariamente uma reunião que deve durar no máximo 15 minutos, onde somente os membros da equipe de produção devem participar e outros integrantes da equipe que desejem participar da reunião podem somente escutar sem direito a opiniões (RISING; JANOFF, 2000).

O Scrum, um *framework* de desenvolvimento RAD, consiste em equipes com papéis e regras bem definidas. Propõe-se a ser uma metodologia leve, simples de entender mas extremamente difícil de dominar. Valoriza a transparência, inspecionabilidade e adaptabilidade (SUTHERLAND; SCHWABER, 2012b).

[2] Um guia oficial do funcionamento do Scrum pode ser encontrado em <http://www.scrum.org/Scrum-Guides >

3.5 V-model

Modelo desenvolvido pela NASA (*National Aeronautics and Space Administration*) agência do Governo Americano responsável pela pesquisa espacial. Este modelo também é uma variação do modelo *waterfall*. Apresentado na primeira conferência da NCOSE (*National Council on Systems Engineering*), que mais tarde veio se tornar a INCOSE (*International Council on Systems Engineering*), o nome é caracterizado por seu fluxo apresentar-se em forma de "V"onde a primeira metade (metade esquerda) foca na definição do requisitos, dividindo-os em componentes cada vez menores, enquanto na segunda parte (metade direita do V) o objetivo é integrar e verificar as etapas de implementação dos componentes detalhados na primeira parte (RUPARELIA, 2010).

Ao colocar este modelo em um eixo cartesiano onde o eixo das abcissas representa o tempo e o das ordenadas o nível de detalhamento dos requisitos podemos notar que quanto mais detalhado mais tempo será necessário para que o *software* fique pronto. O modelo em "V"é simétrico, o que significa que ambos os lados devem apresentar o mesmo número de passos, onde a verificação do lado direito deve corresponder a uma especificação apresentada no lado esquerdo.

O modelo pode ser dividido em cinco grandes passos dentro das duas etapas:

- Primeira etapa (decomposição e definição)
  - requisitos de sistema
  - requisitos da aplicação
- Desenvolvimento
- Segunda etapa (integração e verificação)
  - testes
  - validação

O Modelo em "V"tem como objetivo o desenvolvimento do *software*, visando minimizar os riscos de projeto através de uma estrutura em que formula-se os métodos de avaliação de cada requisito a medida em que estes são identificados. Identificar como avaliar os requisitos ajuda a prever possíveis problemas que ele acarreta.

Ao minimizar os problemas a verificação de cada etapa torna-se mais simples possibilitando a todos os envolvidos (de desenvolvedores a clientes) interagir e validarem cada ponto. Com a redução do problema a documentação também pode ser simplificada ajudando na comunicação e consecutivamente no acordo e validação entre os envolvidos do que deve ser entregue (RUPARELIA, 2010).

A identificação de problemas em etapas iniciais do desenvolvimento do projeto evita o desperdício de tempo e esforço no projeto reduzindo seus custos. Além disso a identificação

em estágios iniciais dos resultados esperados de modo pontual ajuda na estimativa tanto de custos quanto de esforço.

Contudo, o modelo é limitado ao desenvolvimento do *software*, não engloba como devem ser tratados a questão de contratos, operação, manutenção ou reparos (FORSBERG; MOOZ, 1991).[1]

Adaptações em cima do modelo em "V"foram propostas e por Mooz e Forsberg em (BOEHM, 1986) chegando em um modelo denominado "V^{+}". Nesta versão são adicionadas etapas de decomposição de análise e resolução do processo do lado esquerdo e verificação de análise e decomposição do processo do lado direito, para assim ter maior visão do processo em si possibilitando enxergar o processo do ponto de vista do usuário.

Como o *waterfall*, este modelo apresenta mais aderência ao desenvolvimento de *software* que sustentam a base dos programas (*back-end*) e as adaptações do "V^{+}"possibilitam sua expansão para *software* com interface para o usuário final.

O modelo em "V", contudo, é aconselhável para programas de grande porte onde envolvam uma equipe bem diversificada a fim de que todos devem verificar os possíveis erros e validar antes de continuar (RUPARELIA, 2010).

[1] Um tutorial de como aplicar o modelo em "V"pode ser encontrado em <http://v-modell.iabg.de/v-modell-xt-html-english/index.html#toc0>.

4 Proposta de modelo

Para o desenvolvimento do modelo é preciso entender como é o ambiente de desenvolvimento o qual o modelo se aplica e como se comporta um fluxo típico. Para tal, será analisado o desenvolvimento de um projeto e de problemas tipicamente enfrentados.

Após identificar o ambiente, em conformidade com os pontos que deseja-se ter controle no desenvolvimento de *software* dentro do laboratório, será descrito como o modelo deve apresentar-se e, ao termino da descrição, um fluxograma resumindo os passos deve ser apresentado para resumir a proposta.

4.1 Levantamento de requisitos

A partir da análise dos modelos estudados, é possível identificar quais pontos são comuns, o que torna cada modelo atraente e ao confrontar com as práticas do laboratório podemos notar quais destes pontos podem ser relevantes e através de uma análise crítica podemos então definir o que é importante culturalmente para um laboratório que visa desenvolver um *software*.

4.1.1 Ambiente de desenvolvimento

Dado as características do ambiente decidiu-se pela implementação de um módulo do sistema Polvo. O POLVO é uma plataforma que foi concebida como ferramenta de apoio a aprendizagem com o objetivo de auxiliar as aulas e ajudar na interação entre aluno e professor. A versão corrente estabilizou-se após algumas tentativas de implementação.

Atualmente apresenta-se em uma arquitetura MVC (*Model View Controller*) implementada em PHP (*PHP: Hypertext Preprocessor*), com um sistema central "(*core*)" responsável pelo controle e ligação entre os módulos. Baseada em um modelo de desenvolvimento orientado a módulos, no qual cada problema identificado, é planejado e modelado para ser solucionado no seu próprio contexto com alta coesão e pouca dependência.

Cada um dos módulos poderia caracterizar um sistema completo e é concebido com alguma demanda feita por alunos ou professores com o intuito de ajudar algum ponto na relação entre alunos e professores e a realização dos trabalhos acadêmicos.

O módulo cujo acompanhamento será feito a fim de analisar o processo de desenvolvimento é o módulo Magi[1]. Este, apresenta como objetivo principal auxiliar na estruturação

[1] A especificação completa se encontra em apêndice

e elaboração de um TCC (Trabalho de Conclusão de Curso) seguindo as normas da ABNT (Associação Brasileira de Normas e Técnicas).

Concebido inicialmente para facilitar a normatização de um documento segundo a ABNT, acabou tornando-se uma ferramenta para desenvolvimento de um modelo estrutural de TCC, acompanhamento do processo e desenvolvimento do projeto, finalizando com a geração do documento em formato LaTeX (.tex) segundo normas descritas pela ABNT. Sendo ainda pensado para facilmente acoplar outras estruturas de modelo de documento, como por exemplo, um artigo da IEEE (*Institute of Electrical and Electronics Engineers*).

O módulo Magi do Polvo será usado como exemplo e projeto piloto do modelo aqui desenvolvido.

4.1.2 Necessidades do laboratório

O laboratório apresenta uma mudança constante de parte dos integrantes, os quais são responsáveis pelas mais variadas áreas do desenvolvimento dos projetos, e sendo necessário que, nessa mudança, não haja perda de informações durante a troca dos responsáveis por cada tarefa.

As tarefas podem ser as mais variadas, desde a parte de coleta de requisitos e modelagem do fluxo das regras de negócio, feito em grande parte pelos estudantes de administração, até a parte de desenvolvimento, manutenção e planejamento de bases de dados, cuidados com infra estrutura de rede e servidores e serviços.

Se algum dos projetos necessitar de algum recurso especial por parte do servidor, é necessário documentar para que ao instalar o Polvo em outros lugares que o responsável por essa instalação tenha ciência dos recursos necessários.

Por filosofia do laboratório, procura-se desenvolver sempre pensando no caso genérico e voltado a RAD e com isso a documentação acaba ficando um pouco de lado, tanto pelo trabalho extra necessário para fazê-lo quanto por conta do tempo muitas vezes reduzido. Outro ponto é a constante mudança nos escopos dos projetos, pois muitas vezes, além de caracterizarem-se por projetos de P&D os clientes não são da área de computação e não sabem direito o que pedir, dificultando a manutenção das documentações.

4.2 Proposta de modelo

Nesta proposta de modelo, foram definidos 3 (três) passos principais e 1 (um) passo de pré projeto.

Para identificar os itens a serem desenvolvidos, quando estiver sendo tratado do lado do cliente será denominado *funcionalidade*, para que seja dada a ideia de um fluxo de processo completo. O mesmo item do lado do desenvolvedor como será denominado

objetivo, para que de a ideia de um ponto a ser desenvolvido que deva funcionar sem que o restante dos objetivos tenham sido implementados.

O cliente pode ser o próprio desenvolvedor, neste caso, o emprego dos termos ajuda a definir qual é a ideia inicial e o que de fato será desenvolvido. Pois a *funcionalidade* imaginada pelo cliente, depois de modelada, pode apresentar algumas correções e este item corrigido, passa a ser o *objetivo* a ser desenvolvido.

Essas *funcionalidades* são melhor visualizadas e identificadas em forma de casos de uso e modeladas como diagramas de atividade da notação UML (*Unified Modeling Language*), contudo é necessário um certo nível de conhecimento da linguagem para diagramá-lo e lê-lo.

Uma alternativa um pouco mais simples é o BPMN (*Business Process Model and Notation*), onde apesar de existir a mesma necessidade de conhecimento para estruturá-la a sua leitura é mais intuitiva, pois mostra o processo de modo menos abstrato.

Conquanto o UML seja um pouco mais complexo que o BPMN, por ser mais detalhado e voltado à programação, pode facilitar na hora do desenvolvimento uma vez que apresenta menos margem à dupla interpretação. Será usado o BPMN como base de exemplo, por sua característica de menor abstração, mas o diagrama de atividade UML pode ser utilizado da mesma maneira.

Etapa 0 - (Identificação e definição das tecnologias que serão usadas)

Definido o escopo geral do projeto, deve-se buscar inferir quais os possíveis pontos críticos a serem enfrentados durante o desenvolvimento. Uma vez identificados estes pontos o desenvolvedor deve estudar quais são as tecnologias disponíveis e quais são mais adequadas. Nesse ponto entra a escolha de linguagem, banco de dados e estruturas auxiliares como bibliotecas.

Escolhido as tecnologias faz-se um pequeno relatório descrevendo as tecnologias, a razão de sua escolha e se necessário suas limitações.

Esta etapa não é necessária se o projeto apresentar o mesmo ambiente de projetos anteriores, já modelados.

Etapa 1 - (Lista de objetivos)

Enquanto é realizado o estudo das tecnologias o cliente tem um tempo pra amadurecer a ideia e mesmo que não tenha o escopo inteiramente decidido, a essa altura já terá alguma noção de *funcionalidades* que o seu projeto deverá apresentar.

A partir destas ideias, descreve-se uma lista com essas *funcionalidades*. Com a lista pronta gradua-se a importância de cada *funcionalidade* no projeto.

Etapa 2 - (Desenvolvimento)

A partir da lista de objetivos gerada, escolhe-se um para desenvolver. Segue-se o descrito no BPMN e os teste unitários podem ser aplicados nesta fase.

Se o projeto estiver sendo desenvolvido em equipe, o gerente de projeto pode dividir o objetivo em partes menores para paralelizar o processo. E caso o problema ainda

se apresente muito complicado, é possível aplicar outros modelos de desenvolvimento somente no objetivo corrente fazendo com que os problemas mais complicados apresentem maior nível de documentação.

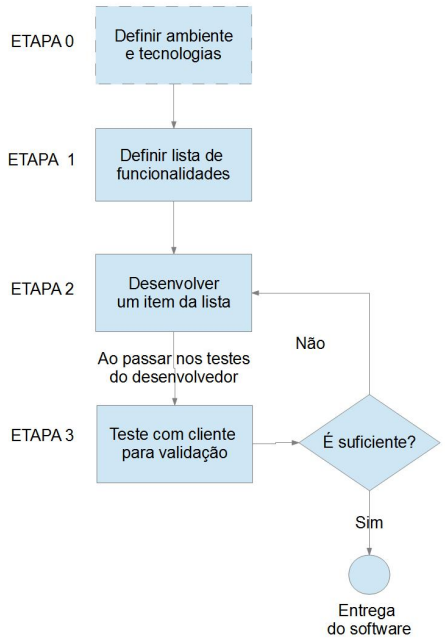
Etapa 3 - (Verificação de finalização)

Com o *objetivo* escolhido já desenvolvido e testado pelo desenvolvedor, cria-se um versão de "*release*". Esta versão deve ser então testada pelo cliente e, caso esteja de acordo com o solicitado, deverá validá-lo. Caso contrário o BPMN deve ser revisto e reinicializado o fluxo até o cliente validar a *funcionalidade*.

Cabe dizer que, é nesta fase que uma possível equipe de teste deve atuar, embora esta equipe seja desejável, muitas vezes os pequenos grupos de pesquisa não apresentam este tipo de recurso.

Se este protótipo satisfizer as necessidades do cliente então o projeto pode ser finalizado. Se ainda não for o suficiente escolhe-se outro item da lista e retorna-se à etapa de desenvolvimento (fig. 1).

Figura 1 – Fluxograma da proposta de modelo

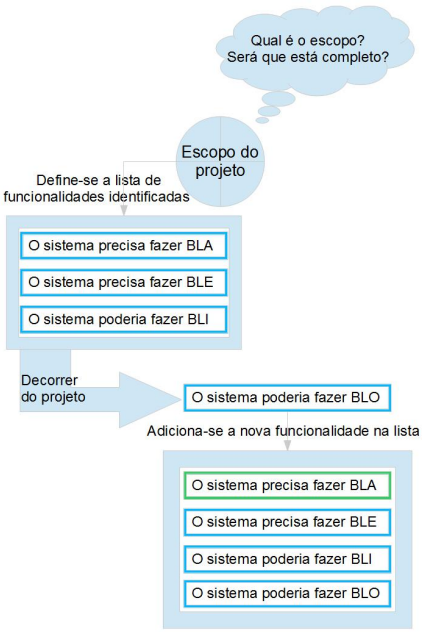


Ao final de cada etapa de desenvolvimento os testes que o desenvolvedor faz, quando aplicável, deve ser feito também nos itens anteriormente desenvolvidos para garantir que as alterações no código não comprometam o desenvolvido anteriormente.

A lista de *funcionalidades* pode ser incrementada o tempo todo, mas a mudança de um *objetivo* já descrito deverá ser evitada e, quando houver a necessidade de fazê-lo justificar para que possa ficar documentado o motivo da mudança.

A possibilidade de adicionar as *funcionalidades* o tempo todo permite que o *software* seja desenvolvido mesmo com escopo incompleto, e ao planejar a lista de *funcionalidades* esta deve ajudar o cliente a clarear o seu próprio objetivo principal (fig. 2).

Figura 2 – Durante a elaboração da lista de funcionalidades pode não se identificar todas e no decorrer do projeto identificar esse novo requisito e adiciona-lo na lista.



Os *objetivos* devem ser *funcionalidades* que possam existir sozinhas e serem usadas no objetivo principal. Isso permitirá que os *objetivos* sejam ao mesmo tempo pequenos, permitindo micro-ciclos de desenvolvimento, e concisos, facilitando o teste e validação.

5 Aplicação do modelo

Como projeto para avaliação do modelo, será usado o módulo Magi do sistema Polvo. Este módulo é uma ferramenta para auxiliar na formulação de documentos seguindo um modelo previamente definido e como resultado da interação apresenta um documento estruturado tal qual o modelo definido segundo as normas escolhidas.

Inicialmente o objetivo principal deste módulo divide-se em 2 partes, onde a primeira é a possibilidade do professor gerar um modelo de fluxo para desenvolvimento do texto para os seus orientandos e a segunda é, ao final do processo, o aluno ter o texto de acordo com as normas ABNT.

Contudo não se sabia como este documento seria gerado, nem qual o nível de alteração necessária depois de sua finalização, e por haver a possibilidade de querer um documento alterável gerar um documento pdf, por exemplo, foi descartado. Como, no final do processo, desejava-se que o documento obedecesse as normas da ABNT para a confecção de monografias, decidiu-se que os documentos gerados pelo sistema seriam no formato LaTeX já pronto para ser compilado e permitindo que o usuário possa fazer as alterações desejadas no documento.

Para possibilitar gerar este documento em LaTeX, a partir de uma interface web, foi empregado a tecnologia XSLT (*Extensible Stylesheet Language Transformations*) transformando um documento em notação HTML para a notação em LaTeX.

Assim definiu-se os elementos da Etapa 0, onde o LaTeX e o XSLT são as tecnologias adotadas que foram agregadas ao ambiente do POLVO, consistido em sua base atual com MySQL, PHP e Javascript (JQuery).

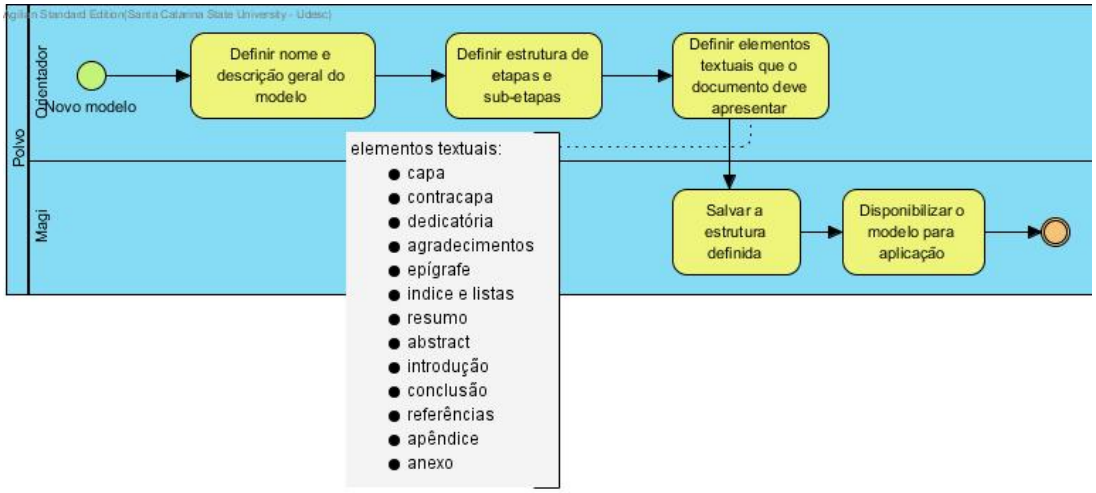
Para a etapa 1 são descritos alguns dos casos identificados e os BPMs correspondentes à título de exemplificação, o restante dos BPMs que compõem a lista de *funcionalidades* apresentam-se no apêndice.

Uma *funcionalidade* identificada foi, "Criar fluxo de desenvolvimento de modelo"(fig. 3). Para esta *funcionalidade*, é descrito, o seguinte caso de uso, um professor define um modelo de fluxo de projeto onde o ele descreve quais as etapas o modelo apresenta e o que cada uma deve apresentar ou como deve ser conduzida.

À titulo de exemplo, em UML, para o mesmo caso de uso descrito anteriormente, é definido o diagrama de atividade apresentado na fig. 4.

Outra *funcionalidade* identificada foi "Orientar projeto". Neste caso, o professor, já com o modelo definido, deve, ao selecionar um aluno, definir qual o modelo de fluxo de projeto o aluno deve seguir e ao atribuir o modelo para este, o professor torna-se orientador do mesmo (fig. 5).

Figura 3 – Descrição através da notação BPMN do caso de uso de desenvolvimento do modelo de fluxo de projetos onde o professor define quais etapas e itens textuais o projeto final apresentará.



Foram identificados mais os seguintes casos de uso:

- Adicionar referências e citações;
- Definir textos de elementos textuais da monografia;
- Compor o texto (conteúdo);
- Acompanhar a composição do texto;
- Finalizar projeto.

Observa-se que nenhuma das *funcionalidades* levantadas levam, isoladamente, ao objetivo (estruturar uma monografia, TCC, segundo as normas da ABNT), mas são todas necessárias para que o texto seja construído.

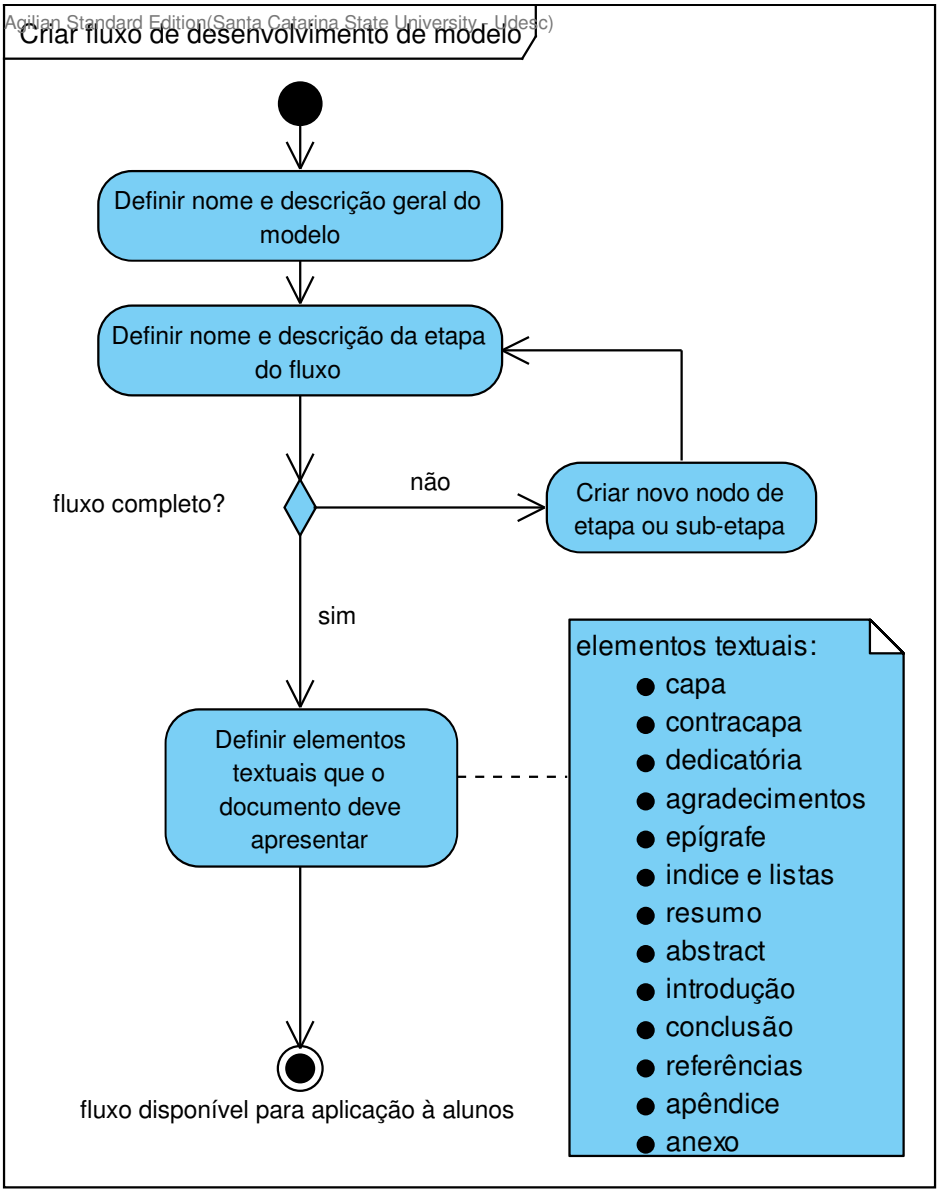
Depois de levantado a lista de *funcionalidades*, é possível graduar cada um dos itens para identificar quais são os mais importantes, e com isso dar prioridades na hora do desenvolvimento.

Assim segue-se acrescentando itens à lista até que "esgotem-se"as possibilidades de funcionalidades, com isso, caracteriza-se a etapa 1.

Com qualquer *objetivo* definido e modelado é possível iniciar a etapa 2, contudo, quanto maior for a lista de funcionalidades melhor será para o desenvolvedor, pois facilitará a identificação de pré-requisitos e dependências entre as funcionalidades.

Na etapa 2, a fase de desenvolvimento, escolhe-se um dos itens da lista definida para implementar. Ao analisar o BPM do *objetivo* podemos verificar os atores envolvidos, como estes interagem com o sistema e qual o resultado esperado nessa interação. Neste caso, o *objetivo* escolhido é "Criar fluxo de desenvolvimento de modelo"(fig. 3).

Figura 4 – Descrição através da notação UML de um diagrama de atividades do caso de uso de desenvolvimento do modelo de fluxo de projetos onde o professor define quais etapas e itens textuais o projeto final apresentará.



Para o desenvolvimento deste *objetivo*, verifica-se a necessidade guardar do fluxo, informações sobre a estrutura, nome e descrição das etapas que o fluxo apresenta, qual a ordem das etapas e subetapas e quais os elementos textuais que o modelo pode apresentar.

Durante o desenvolvimento, deste *objetivo*, foi levantado uma outra *funcionalidade* para o sistema, "Adicionar exemplos de etapa", que descreve como adicionar arquivos previamente elaborados ao sistema com exemplos e explicações sobre o que a etapa representa ou deve apresentar (fig. 6). Identificado a nova *funcionalidade*, esta foi adicionada a lista.

Ao terminar o desenvolvimento do *objetivo* e os testes básicos sobre ele finaliza-se a etapa de 2. Então, para seguir à etapa 3, é "empacotado" uma versão protótipo onde o

Figura 5 – Descrição através da notação BPMN do caso de uso de atribuição de projeto aos alunos iniciando uma orientação seguindo um dos modelos de fluxo desenvolvido.

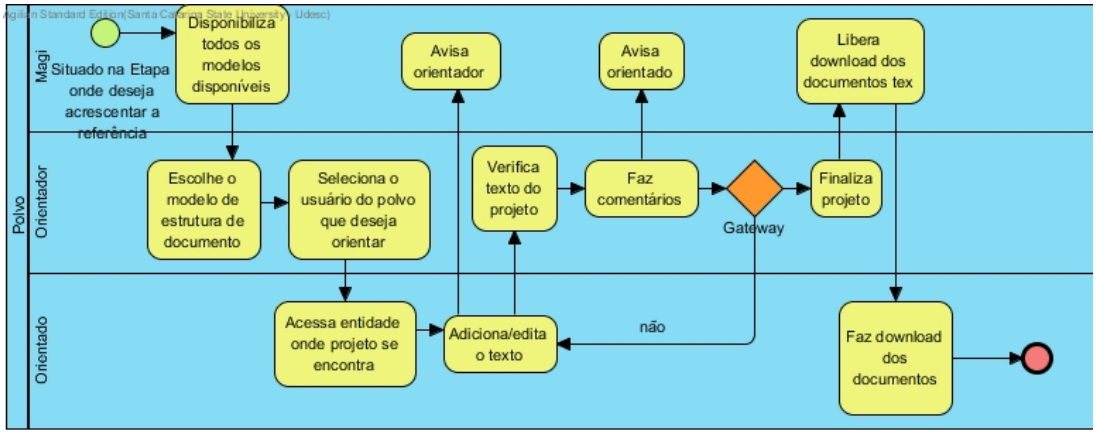
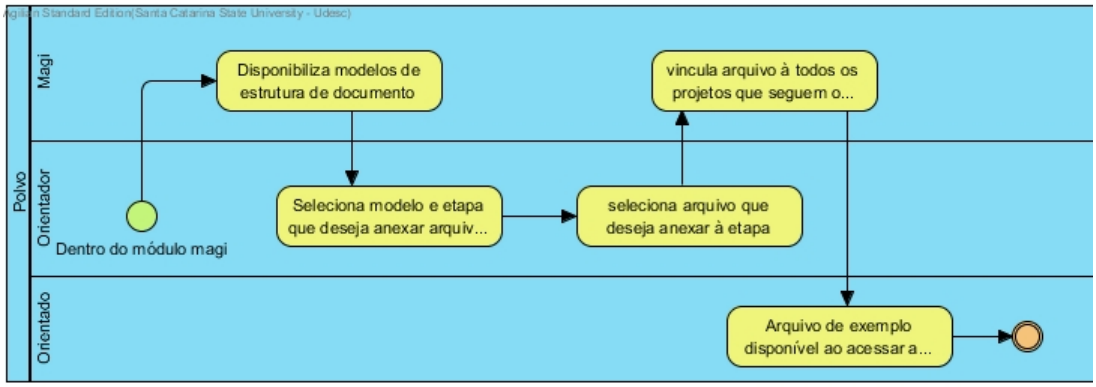


Figura 6 – Descrição através da notação BPMN do caso de uso de adição no sistema de arquivos de exemplo sobre a etapa corrente.



cliente deve testar e ver se é isso mesmo que ele esperava e se há necessidade de alguma alteração. Se a necessidade existir verificar no BPM correspondente, o que foi modelado e ver se é condizente com o pedido feito, em caso afirmativo fazer as devidas alterações no programa, cabe colocar uma nota no BPM para elucidar qualquer dúvida, e em caso negativo pedir para o cliente a razão para realizar a alteração no programa, caso esta seja viável.

Se não houver necessidade de alterações, conclui-se com isso, a etapa 3. Após esta etapa verifica-se se este protótipo é suficiente para cumprir o objetivo principal do projeto, (um documento LaTeX com um projeto de TCC). O objetivo desenvolvido foi o de criação de fluxo e, com, somente isso, ainda não há como gerar o documento em formato tex, logo, escolhe-se outro *objetivo* da lista e retorna-se à fase de desenvolvimento.

Quando o protótipo gerado for suficiente para realizar o objetivo geral então esta passa a ser uma versão de "entrega" de onde, a partir de então, hão 3 alternativas:

1. Fechar o desenvolvimento e entregar ao cliente o *software* que está funcional mas, possivelmente, não otimizado;

2. Fechar o desenvolvimento e fazer um "*refactoring*"do código a fim de otimizá-lo para entregar um código limpo ao cliente;

3. continuar o desenvolvimento dos itens da lista até chegar a um ponto desejado;

Em qualquer das alternativas a partir deste ponto o *software* poderá ser tratado como completo e/ou apresentável. Sendo que, as próximas iterações dependerão do prazo estabelecido, do controle de qualidade, necessidade de mais requisitos entre outras causas.

6 Considerações Finais

Esta proposta de modelo visa propiciar o desenvolvimento de um *software* concomitantemente ao desenvolvimento dos conceitos empregados nele, ou seja, o escopo do *software* não necessita estar bem definido para que o desenvolvimento possa começar, além disso, a participação do cliente no desenvolvimento, teste das funcionalidades e modelagem dos BPMs devem ajudá-lo a amadurecer as ideias, pois os pedidos de desenvolvimento são interações que devem ser possíveis serem realizadas no sistema, e ao mapear estas interações ele terá noção de como deve ser o fluxo das informações, podendo auxiliar em seu desenvolvimento.

Através da graduação de importância da funcionalidade em relação ao objetivo geral do projeto, é possível identificar quais são os objetivos que devem ser priorizados na hora do desenvolvimento focando o esforço de trabalho para os de maior grau.

O modelo foi idealizado para que um único desenvolvedor possa realizar todo o processo mas também para que possa ser adotado por uma equipe. O esforço de trabalho pode ser quantificado em homem hora e facilmente calculado para cada objetivo desenvolvido. Com o passar do tempo e a adequação do modelo as características do desenvolvedor ou equipe a previsão de tempo deve tornar-se mais natural e precisa.

Além disso, esta lista de objetivos pode facilmente ser cadastrada em um gerenciador de desenvolvimento de projetos (trac, Redmine etc) para ser feito o controle e distribuição das tarefas. Apesar de não ser muito fácil paralelizar o desenvolvimento, dado que um item da lista de objetivo pode ter algum outro item como pré-requisito, as próprias funcionalidades podem ter seu desenvolvimento feito de forma concorrente pelos membros da equipe. E, como a equipe estará toda voltada ao desenvolvimento de um *objetivo*, diminui a necessidade de reuniões para interação da equipe sobre o que cada um está fazendo.

Caso hajam *objetivos* em que o desenvolvimento necessite de maior cuidado, documentação mais detalhada, ou quaisquer outras necessidades especiais, é possível empregar outros modelos de desenvolvimento de *software* ou *frameworks* dentro da etapa de desenvolvimento e detalhar o processo do *funcionalidade* cuja necessidade foi identificada ou pedida.

Os BPMs gerados durante o processo, descrevem tudo o que o *software* deve fazer e se descritos em linguagem comum servem como fonte para o desenvolvimento de manuais do *software*, conforme necessidade, e se foram atualizados conforme as mudanças ocorreram então apresentam informações suficientes para replicar o *software*.

O modelo estrutural do presente trabalho foi gerado a partir do módulo Magi cujo desenvolvimento seguiu o modelo de desenvolvimento de *software* proposto neste mesmo

trabalho. Após algumas interações com o sistema, optou-se por trabalhar o texto em uma interface LaTeX padrão (texmaker) utilizando o documento gerado pelo sistema. A opção por trabalhar no texmaker se deu por motivos de conveniência pois, por se tratar de um sistema em desenvolvimento, foi constantemente necessário limpar a base para recomeçar testes.

Uma primeira proposta de trabalho futuro seria a de utilizar este modelo para desenvolver um sistema para importação de arquivos tex para um modelo de projeto dentro do sistema.

Há ainda várias funcionalidades a serem desenvolvidas para o Magi mas, este já está em situação de "entrega", ou seja, já é possível gerar um documento da notação tex, e as funcionalidades ainda não implementadas visam apenas melhorar a qualidade do documento produzido assim como a usabilidade do sistema.

Outras propostas de trabalhos futuros seriam a de utilizar o modelo em mais projetos dentro do LabTIC para amadurecê-lo e posteriormente, aplicação do modelo em outros grupos para averiguar a sua aderência neles.

Referências

ABRAHAMSSON, P. et al. *Agile software development methods - Review and analysis*. Finland, 2002. ISBN 9513860094.

AHMED, F.; CAPRETZ, L. Best practices of rup ® in software product line development. In: *Computer and Communication Engineering, 2008. ICCCE 2008. International Conference on*. [S.l.: s.n.], 2008. p. 1363 –1366.

BARBARA, A. K. Controlling software projects. *Electronics and Power*, v. 33, n. 5, p. 312 –315, may 1987. ISSN 0013-5127.

BARR, A.; TESSLER, S. *The Globalization of Software R&D: The Search for Talent*. dez. 1996.

BENINGTON, H. D. Production of large computer programs. In: *Proceedings of the 9th international conference on Software Engineering*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1987. (ICSE '87), p. 299–310. ISBN 0-89791-216-0. Disponível em: <http://dl.acm.org/citation.cfm?id=41765.41799>.

BOEHM, B. A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes*, ACM, New York, NY, USA, v. 11, n. 4, p. 14–24, ago. 1986. ISSN 0163-5948. Disponível em: <http://doi.acm.org/10.1145/12944.12948>.

BORGES, P.; MONTEIRO, P.; MACHADO, R. Tailoring rup to small software development teams. In: *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*. [S.l.: s.n.], 2011. p. 306 –309.

CHANG, C.; TRUBOW, G. Joint software research between industry and academia. *Software, IEEE*, v. 7, n. 6, p. 71 –77, nov. 1990. ISSN 0740-7459.

FORSBERG, K.; MOOZ, H. The relationship of system engineering to the project cycle. In: • (Ed.). *Proceedings of National Council For Systems Engineering First Annual Conference*. [S.l.: s.n.], 1991.

GAO, Y. Research on the rule of evolution of software development process model. In: *Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on*. [S.l.: s.n.], 2010. p. 466 –470.

GILB, T. Evolutionary delivery versus the "waterfall model". *SIGSOFT Softw. Eng. Notes*, ACM, New York, NY, USA, v. 10, n. 3, p. 49–61, jul. 1985. ISSN 0163-5948. Disponível em: <http://doi.acm.org/10.1145/1012483.1012490>.

IIVARI, J. A hierarchical spiral model for the software process. *SIGSOFT Softw. Eng. Notes*, ACM, New York, NY, USA, v. 12, n. 1, p. 35–37, jan. 1987. ISSN 0163-5948. Disponível em: <http://doi.acm.org/10.1145/24574.24576>.

LIU, D.; XU, S.; BROCKMEYER, M. Investigation on academic research software development. In: *Computer Science and Software Engineering, 2008 International Conference on*. [S.l.: s.n.], 2008. v. 2, p. 626 –630.

PAIVA, D.; TURINE, M.; FORTES, R. de M. A comparative review of processes for research development on applied computing. In: *Software Engineering Research, Management and Applications, 2008. SERA '08. Sixth International Conference on*. [S.l.: s.n.], 2008. p. 117 –124.

RAYMOND, E. S. *The Cathedral and the Bazaar*. 1st. ed. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 1999. ISBN 1565927249.

RISING, L.; JANOFF, N. The scrum software development process for small teams. *Software, IEEE*, v. 17, n. 4, p. 26 –32, jul/aug 2000. ISSN 0740-7459.

ROYCE, W. W. Managing the development of large software systems: concepts and techniques. In: *Proceedings of the 9th international conference on Software Engineering*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1987. (ICSE '87), p. 328–338. ISBN 0-89791-216-0. Disponível em: <http://dl.acm.org/citation.cfm?id=41765.41801>.

RUPARELIA, N. B. Software development lifecycle models. *SIGSOFT Softw. Eng. Notes*, ACM, New York, NY, USA, v. 35, n. 3, p. 8–13, maio 2010. ISSN 0163-5948. Disponível em: <http://doi.acm.org/10.1145/1764810.1764814>.

SUTHERLAND, J.; SCHWABER, K. nov. 2012. Disponível em: <http://assets-.scrumfoundation.com/downloads/2/scrumpapers.pdf?1285932052>.

SUTHERLAND, J.; SCHWABER, K. *Scrum Guide*. nov. 2012. Disponível em: <http://www.scrum.org/Scrum-Guides>.

APÊNDICE A – Artigo - Modelagem de Projetos

Modelagem de projetos

Daniel K. Nakano[1]

[1]Departamento de Informática e Estatística (INE) – Universidade Federal de Santa Catarina (UFSC)
88.040-100 – Florianópolis – SC – Brasil

dankoudi@gmail.br

Abstract. *This paper is related to the term paper called "Modelo de desenvolvimento de software em pequenos grupos de pesquisa" (Software development model for small research groups). The survey that was made for the term paper to understand the software development models that can be applied in the context of R&D groups is presented in this paper.*

Resumo. *Artigo referente ao projeto "Modelo de desenvolvimento de software em pequenos grupos de pesquisa", desenvolvido para fins de trabalho de conclusão de curso. Neste trabalho apresenta-se a pesquisa feita durante o trabalho a fim de colher informações sobre os modelos de desenvolvimento de softwares existentes no mercado que apresentam relevância no contexto de aplicação de grupos de pesquisa.*

1. Introdução

Este trabalho apresenta o resultado de um *survey* realizado nas bases de artigos da IEEE e da ACM acerca de modelos de desenvolvimento de software, para o desenvolvimento do modelo proposto no trabalho de conclusão de curso (TCC) sob o título "Modelo de desenvolvimento de software em pequenos grupos de pesquisa" e visa mostrar, em linhas gerais, os modelos de desenvolvimento de software que foram adotados, parcialmente ou apenas como referência, no modelo proposto no trabalho.

1.1. Objetivos

Apresentar os modelos de desenvolvimento de software que foram pesquisados e usados como base para o desenvolvimento do modelo proposto para o trabalho supra citado.

2. Modelos

Modelos de desenvolvimento de software, têm como objetivo descrever o que é necessário fazer para que um software seja produzido, não lhes cabendo dizer como fazer. Como implementar um modelo é trabalho de um framework de desenvolvimento, muitas vezes estes frameworks levam o mesmo nome do modelo em si mas podem haver diferentes frameworks para um mesmo modelo. Os modelos podem ser categorizados em lineares, iterativos e mistos onde, nos lineares as etapas são sequenciais e o final de uma etapa leva ao inicio da próxima e ao final de todas as etapas o software está pronto, enquanto nos iterativos é possível que ao final do último passo volte-se ao inicial revisitando cada etapa ou, alguma em especial, para eventual modificação. Nos mistos segue-se de modo geral a linha dos modelos iterativos e em

certo momento é definido que se siga o linear para que seja possível ter uma versão do sistema (RUPARELIA, 2010).

2.1.MODELO EM CASCATA (WATERFALL MODEL)

Considerado o primeiro modelo de desenvolvimento de software, foi inicialmente documentado por Benington (BENINGTON, 1987), e posteriormente adaptado por Royce, na década de 70, (ROYCE, 1987; RUPARELIA, 2010).

Apresenta basicamente 7 passos: análise operacional, especificação operacional, especificações de projeto e modificação, desenvolvimento, teste, entrega e avaliação. Na adaptação de Royce, ele sugere que se faça um *feedback*, para evitar os problemas de projeto que não foram previstos. Vale lembrar que na época de Benington, a programação era feita em ata e os computadores eram máquinas que precisavam ser previamente agendadas para utilizá-las. Desenvolver e testar um software nessas condições exigia um esforço bem maior do que atualmente e problemas de compilação e lógica exigiam muito mais esforço para identificação. Por essa razão a documentação se faz essencial para identificação dos objetivos, características, entre outras informações. Royce sugere que ao menos 6 tipos de documentos devem ser produzidos:

1. Documento de especificação de requisitos;
2. Projeto preliminar;
3. Especificações de design de interface;
4. Projeto definitivo;
5. Plano de testes;
6. Manual de operação ou instruções.

A produção destes documentos ajudam o cliente a controlar a produção, os desenvolvedores a saber o que deve realmente ser produzido, projetar e realizar os testes. Facilitam a manutenção e utilização do programa pelo usuário. E ao realizar aprimoramentos, no futuro, ajuda a identificar onde devem ou podem ser realizados as mudanças e ou reprojeto (ROYCE,1987).

Este modelo foi desenvolvido pensando em projetos de grande porte e foi a base para grande parte dos modelos que vieram a surgir posteriormente.

2.2.INCREMENTAL

Modelo desenvolvido como uma forma iterativa do modelo *waterfall*. O projeto tende a ser divido tomando como base suas funcionalidades, tornando cada projeto resultante um projeto menor e mais fácil de implementar. Apresenta vantagens em relação ao modelo waterfall tradicional pois feedbacks de iterações anteriores podem ser aproveitadas nas iterações posteriores, aproximando-se do modelo espiral. Os envolvidos podem interagir durante cada iteração e ajudar a identificar os riscos mais cedo. Entrega-se rapidamente uma primeira versão pois esta versão apresenta um pequeno escopo, este escopo é incrementado a cada iteração ampliando as características do programa `a cada rodada. E, por ser incremental, permite que as mudanças sejam monitoradas e os problemas isolados e resolvidos com menor esforço (RUPARELIA, 2010).

A entrega de uma primeira versão é adiantada uma vez que o escopo reduz ampliando a cada iteração. E por ser incremental, permite que as mudanças sejam monitoradas e os problemas isolados e resolvidos com menor esforço (RUPARELIA, 2010).

2.3. MODELO ESPIRAL (*SPIRAL MODEL*)

Modelo baseado no waterfall, apresentado por Boehn este modelo tem como principal objetivo aumentar a produtividade na produção do software com a restrição de manter os custos em um patamar aceitável. é um modelo voltado as pessoas e exige alto grau de comprometimento, os papéis devem ser bem definidos, apresenta uma estrutura pouco aderente a mudanças e os planos para as futuras iterações devem ser feitas com certa antecedência ao final da etapa atual (BOEHM, 1986).

O modelo pode ser dividido em 4 fases por iteração, onde, o término dessas fases seria um marco para entrega ou avaliação do projeto. A primeira fase tem como objetivo identificar os objetivos, a segunda analisar alternativas para a solução do problema, identificar riscos e mitiga-los a medida do possível, a terceira desenvolver e testar e a quarta e última planejar a próxima iteração.

A cada ciclo completo, tem-se um protótipo do projeto com certas funcionalidades desenvolvidas e validadas. Caso os riscos diagnosticados na segunda fase sejam de ordem de desenvolvimento então o caminho a seguir é o do modelo *waterfall* para que estes possam ser tratados. Caso os riscos sejam, apenas de ordem de performance, segue-se para o próximo passo do modelo espiral, garantindo que o protótipo produzido anteriormente apresentem o menor risco possível associado.

Posteriormente foram sugeridas alterações no modelo que possibilitam um melhor controle do ciclo de desenvolvimento. A alteração proposta por Iivari no que chamou de modelo espiral hierárquico (IIVARI, 1987), consistia em subdividir cada uma das fases em duas para que haja um planejamento para cada etapa e possam ser estabelecidos metas e prazos para cada etapa.

Uma vantagem do modelo espiral é o seu trabalho em volta do risco mas sua adoção requer que os projetos tenham um controle de projeto que consiga se adaptar a ele, os contratos precisam ter certo nível de flexibilidade e depende do analista conseguir identificar os riscos corretamente para não afetar o desenvolvimento (RUPARELIA, 2010).

2.4.RAPID APPLICATION DEVELOPMENT - RAD (DESENVOLVIMENTO RÁPIDO DE APLICAÇÕES)

Também conhecido como desenvolvimento ágil de aplicações é outro modelo que utiliza-se de protótipos para identificação de requisitos e validação, desenvolvida, principalmente, por Martin, é semelhante ao modelo de desenvolvimento de software de código aberto (*open source software development model*), também conhecido como *the cathedral and bazaar* model (RAYMOND, 1999), que segue a filosofia do “Libere cedo; Libere frequentemente; Ouça o seu cliente” (*Release early; Release often; Listen to your customers*).

O modelo utiliza-se do protótipo para unir todos os atores envolvidos no processo em um desenvolvimento iterativo. Onde os clientes são incentivados a participar da

prototipação e dos testes unitários (*Unit Tests*) para que possam ser identificados o maior número de requisitos nas fases iniciais do desenvolvimento, e os programadores possam se familiarizar com o fluxo das regras de negócio (RUPARELIA, 2010).

Focados no desenvolvimento do produto, com alta produtividade e alta adaptabilidade, seu uso torna-se inadequado quando o escopo é impreciso, é necessário um grande domínio de ao menos um integrante da equipe, normalmente o gerente do projeto, para que a implementação seja bem sucedida.

A integração da equipe também é importante, quanto maior o entrosamento da equipe mais fluído é o processo de desenvolvimento, por singularizar os problemas a troca de membros pode ser feita sem muitos problemas, mas a velocidade do processo quando um novo membro é incluído pode acarretar em um processo mais longo, devido a necessidade de adaptação do novo membro dentro da equipe.

2.4.1. XP

Também segue a filosofia RAD. O desenvolvimento na metodologia XP (*eXtreme Programming*) é o que apresenta a menor necessidade de documentação pois um código em XP bem feito deve, ser auto explicativo.

A programação XP é bem aderente ao Modelo Orientado a Objetos, mas como a abstração é pessoal o que pode ser considerado um código autoexplicativo para um desenvolvedor, para outro pode não ser tão claro.

Ela é bem empregada em equipes bem pequenas onde o grupo está habituado com o código dos companheiros ou em projetos individuais. Como desenvolve somente o necessário, quando preciso, é extremamente eficiente na resolução de casos específicos e que apresentam tempo bem limitado, para estruturas de sistemas complexos e com equipes grande acaba necessitando de bastante trabalho para que fique adequado as necessidades.

2.4.2.SCRUM

Metodologia com alta aderência a mudanças (RISING; JANOFF, 2000; ABRAHAMSSON et al.,2002), o scrum[2] apresenta "rodadas" de desenvolvimento chamados "*Sprints*". Dentro de um *sprint* há uma reunião chamada *Planning Meeting* (encontro de planejamento), nesta reunião toda a equipe deve participar para decidir o que será desenvolvido e quais os parâmetros de métrica para decidir se o objetivo foi alcançado, sugere-se que a reunião seja feita a cada duas semanas com isso a reunião deve durar no máximo quatro horas (SUTHERLAND; SCHWABER, 2012a).

Dentro destas duas semanas é realizado diariamente uma reunião que deve durar no máximo 15 minutos, onde somente os membros da equipe de produção devem participar e outros integrantes da equipe que desejem participar da reunião podem somente escutar sem direito a opiniões (RISING; JANOFF, 2000).

O Scrum, um framework de desenvolvimento RAD, consiste em equipes com papéis e

regras bem definidas. Propõe-se a ser uma metodologia leve, simples de entender mas extremamente difícil de dominar. Valoriza a transparência, inspecionabilidade e adaptabilidade (SUTHERLAND; SCHWABER, 2012b).

2.4.3.V-MODEL

Modelo desenvolvido pela NASA (*National Aeronautics and Space Administration*) agência do Governo Americano responsável pela pesquisa espacial. Este modelo também é uma variação do modelo waterfall. Apresentado na primeira conferência da NCOSE (*National Council on Systems Engineering*), que mais tarde veio se tornar a INCOSE (*International Council on Systems Engineering*), o nome é caracterizado por seu fluxo apresentar-se em forma de "V" onde a primeira metade (metade esquerda) foca na definição do requisitos, dividindo-os em componentes cada vez menores, enquanto na segunda parte (metade direita do V) o objetivo é integrar e verificar as etapas de implementação dos componentes detalhados na primeira parte (RUPARELIA, 2010).

Ao colocar este modelo em um eixo cartesiano onde o eixo das abcissas representa o tempo e o das ordenadas o nível de detalhamento dos requisitos podemos notar que quanto mais detalhado mais tempo será necessário para que o software fique pronto. O modelo em "V" é simétrico, o que significa que ambos os lados devem apresentar o mesmo número de passos, onde a verificação do lado direito deve corresponder a uma especificação apresentada no lado esquerdo.

O modelo pode ser dividido em cinco grandes passos dentro das duas etapas:

- Primeira etapa (decomposição e definição)
    - Requisitos de sistema
    - Requisitos da aplicação
- Desenvolvimento
- Segunda etapa (integração e verificação)
    - Testes
    - Validação

O Modelo em "V" tem como objetivo o desenvolvimento do software, visando minimizar os riscos de projeto através de uma estrutura em que formula-se os métodos de avaliação de cada requisito a medida em que estes são identificados. Identificar como avaliar os requisitos ajuda a prever possíveis problemas que ele acarreta.

Ao minimizar os problemas a verificação de cada etapa torna-se mais simples possibilitando a todos os envolvidos (de desenvolvedores a clientes) interagir e validarem cada ponto. Com a redução do problema a documentação também pode ser simplificada ajudando na comunicação e consecutivamente no acordo e validação entre os envolvidos do que deve ser entregue (RUPARELIA, 2010).

A identificação de problemas em etapas iniciais do desenvolvimento do projeto evita o desperdício de tempo e esforço no projeto reduzindo seus custos. Além disso a identificação em estágios iniciais dos resultados esperados de modo pontual ajuda na estimativa tanto de custos quanto de esforço.

Contudo, o modelo é limitado ao desenvolvimento do software, não engloba como devem ser tratados a questão de contratos, operação, manutenção ou reparos (FORSBERG; MOOZ,1991).

Adaptações em cima do modelo em "V" foram propostas e por Mooz e Forsberg em (BOEHM, 1986) chegando em um modelo denominado "V+". Nesta versão são adicionadas etapas de decomposição de análise e resolução do processo do lado esquerdo e verificação de análise e decomposição do processo do lado direito, para assim ter maior visão do processo em si possibilitando enxergar o processo do ponto de vista do usuário. Como o *waterfall*, este modelo apresenta mais aderência ao desenvolvimento de software que sustentam a base dos programas (*back-end*) e as adaptações do "V+" possibilitam sua expansão para software com interface para o usuário final.

O modelo em "V", contudo, é aconselhável para programas de grande porte onde envolvam uma equipe bem diversificada a fim de que todos devem verificar os possíveis erros e validar antes de continuar (RUPARELIA, 2010).

3.Conclusão

Dentre os modelos de desenvolvimento de *software* pesquisados os que seguem a corrente de desenvolvimento ágil, foram as que melhores se adequaram às necessidades encontradas dentro do ambiente em questão. Contudo, os ambientes em si, descritos nos modelos tradicionais adequam-se melhor aos ambientes de pesquisa. Por terem sido desenvolvidos em uma situação onde ainda não haviam sido identificados os pontos em comum de grandes projetos e mesmo os grandes projeto envolviam grande parte de pesquisa e desenvolvimento (P&D).

Atualmente, têm-se mais noção das necessidades de cada tipo de projeto e as diferenças que projetos de variados escopos apresentam, além de uma maior diversidade de modelos de desenvolvimento de softwares o que facilita no estudo, implementação e uso dos modelos.

Dado os diferentes modelos e as diferentes necessidades de cada ambiente é possível escolher aquele que melhor se adequa a situação ou a mais conveniente seguindo as adaptações necessárias.

4.Referências

ABRAHAMSSON, P. et al. Agile software development methods - Review and analysis.Finland, 2002. ISBN 9513860094.

AHMED, F.; CAPRETZ, L. Best practices of rup R in software product line development. In: Computer and Communication Engineering, 2008. ICCCE 2008. International Conference on.[S.l.: s.n.], 2008. p. 1363–1366.

BARBARA, A. K. Controlling software projects. Electronics and Power, v. 33, n. 5, p. 312–315, may 1987. ISSN 0013-5127.

BARR, A.; TESSLER, S. The Globalization of Software R&D: The Search for Talent. dez.1996.

BENINGTON, H. D. Production of large computer programs. In: Proceedings of the 9th international conference on Software Engineering. Los Alamitos, CA, USA: IEEE Computer Society Press, 1987. (ICSE '87), p. 299–310. ISBN 0-89791-216-0. Disponível em:<http://dl.acm.org/citation.cfm?id=41765.41799>.

BOEHM, B. A spiral model of software development and enhancement. SIGSOFT Softw. Eng. Notes, ACM, New York, NY, USA, v. 11, n. 4, p. 14–24, ago. 1986. ISSN 0163-5948. Disponível em: <http://doi.acm.org/10.1145/12944.12948>.

BORGES, P.; MONTEIRO, P.; MACHADO, R. Tailoring rup to small software development teams. In: Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on. [S.l.: s.n.], 2011. p. 306 –309.

CHANG, C.; TRUBOW, G. Joint software research between industry and academia. Software, IEEE, v. 7, n. 6, p. 71 –77, nov. 1990. ISSN 0740-7459.

FORSBERG, K.; MOOZ, H. The relationship of system engineering to the project cycle. In: • (Ed.). Proceedings of National Council For Systems Engineering First Annual Conference. [S.l.: s.n.], 1991.

GAO, Y. Research on the rule of evolution of software development process model. In: Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on. [S.l.: s.n.], 2010. p. 466 –470.

GILB, T. Evolutionary delivery versus the ”waterfall model”. SIGSOFT Softw. Eng. Notes, ACM, New York, NY, USA, v. 10, n. 3, p. 49–61, jul. 1985. ISSN 0163-5948. Disponível em: <http://doi.acm.org/10.1145/1012483.1012490>.

IIVARI, J. A hierarchical spiral model for the software process. SIGSOFT Softw. Eng. Notes, ACM, New York, NY, USA, v. 12, n. 1, p. 35–37, jan. 1987. ISSN 0163-5948. Disponível em: <http://doi.acm.org/10.1145/24574.24576>

LIU, D.; XU, S.; BROCKMEYER, M. Investigation on academic research software development. In: Computer Science and Software Engineering, 2008 International Conference on. [S.l.: s.n.], 2008. v. 2, p. 626 –630.

PAIVA, D.; TURINE, M.; FORTES, R. de M. A comparative review of processes for research development on applied computing. In: Software Engineering Research, Management and Applications, 2008. SERA ’08. Sixth International Conference on. [S.l.: s.n.], 2008. p. 117–124.

RAYMOND, E. S. The Cathedral and the Bazaar. 1st. ed. Sebastopol, CA, USA: O’Reilly & Associates, Inc., 1999. ISBN 1565927249.

RISING, L.; JANOFF, N. The scrum software development process for small teams. Software, IEEE, v. 17, n. 4, p. 26 –32, jul/aug 2000. ISSN 0740-7459.

ROYCE, W. W. Managing the development of large software systems: concepts and techniques. In: Proceedings of the 9th international conference on Software Engineering. Los Alamitos, CA, USA: IEEE Computer Society Press, 1987. (ICSE ’87), p.328–338. ISBN 0-89791-216-0. Disponível em: <http://dl.acm.org/citation.cfm?id=41765.41801>.

RUPARELIA, N. B. Software development lifecycle models. SIGSOFT Softw. Eng. Notes, ACM, New York, NY, USA, v. 35, n. 3, p. 8–13, maio 2010. ISSN 01635948. Disponível em: <http://doi.acm.org/10.1145/1764810.1764814>.

SUTHERLAND, J.; SCHWABER, K. Scrum Guide. nov. 2012. Disponível em: <http://www.scrum.org/Scrum-Guides>.

APÊNDICE B – Manual Magi - Casper

B.1 Apresentação

Dentro do Magi o usuário pode gerar um modelo de documento, definindo a estrutura do mesmo, quais as seções e qual deve ser o conteúdo de cada seção. Todo modelo apresenta uma etapa de fim onde é possível habilitar ou desabilitar os elementos textuais e pre textuais que deseja-se no modelo, como por exemplo capa, índice, referências bibliográfica etc.

Depois de definir a estrutura de um documento define-se o autor. O autor usará o modelo definido e poderá ou não alterá-lo de acordo com a vontade do orientador (criador do modelo).

Durante a confecção do documento, o orientador pode entrar dentro do documento e adicionar comentários sobre o texto em cada uma das seções, ou alterar o documento.

Se o orientador tem vários orientandos em vários modelos ele pode ver através de um "*status*"na frente do título de cada projeto, para saber quando houve alterações do documento. O orientador entra em cada modelo e pode ler cada um das partes e fazer comentários para cada uma destas partes e se ao ler todas as etapas achar que está pronto pode sinalizar o projeto como "Pronto"e o autor poderá gerar o seu documento LaTeX.

B.1.1 Objetivo do Magi-Casper

O objetivo geral é gerar um documento, nos moldes de um TCC, permitindo uma interação entre o orientador e orientado (autor) para a elaboração do projeto.

O Casper gerará o documento LaTeX e é necessário que o autor tenha toda a estrutura LaTeX com o abnTeX para poder compilar o documento.

B.1.2 Funcionamento

Para o correto funcionamento do modulo Magi-Casper é preciso definir os papéis no polvo.

- Orientador
- Autor
- Co-orientador

- Membros banca

Para cada um destes papéis definir as ações

- Magi-criarModelo
- Magi-verModelo
- Magi-criarFluxo
- Magi-finalizarModelo
- Magi-atribuirProjeto
- Magi-visualizarUsuarios
- Magi-verProjeto
- Magi-editarProjeto
- Magi-avaliarProjeto

Onde a orientação é:

- Orientador
  - Magi-criarModelo
  - Magi-finalizarModelo
  - Magi-atribuirProjeto
  - Magi-visualizarUsuarios
  - Magi-verProjeto
  - Magi-editarProjeto
  - Magi-avaliarProjeto
- Autor
  - Magi-visualizarUsuarios
  - Magi-verProjeto
  - Magi-editarProjeto
- Co-orientador
  - Magi-verModelo
  - Magi-visualizarUsuarios

–Magi-verProjeto

–Magi-editarProjeto

–Magi-avaliarProjeto

•Membros banca

–Magi-verProjeto

–Magi-avaliarProjeto

Se os papéis não estiverem definidos apenas os nome do orientador e autor aparecerão no documento gerado.

A partir do momento em que os papéis estiverem definidos ao incluir um novo integrante na entidade do projeto pode-se editar suas informações para que apareça corretamente no documento.

Embora a parte de avaliação e gerenciamento dos documentos (Magi-Melchior) não esteja implementado o controle da edição de um projeto também usa o papel avaliação então é preciso que seja atribuído aos papéis.

B.1.2.1 Criação de modelos

Para criar um novo fluxo de desenvolvimento de documento é necessário que o módulo Magi esteja disponível na entidade.

Como dito anteriormente recomenda-se que se crie uma entidade separada para os projetos como pode ser visto a seguir (fig. 7):

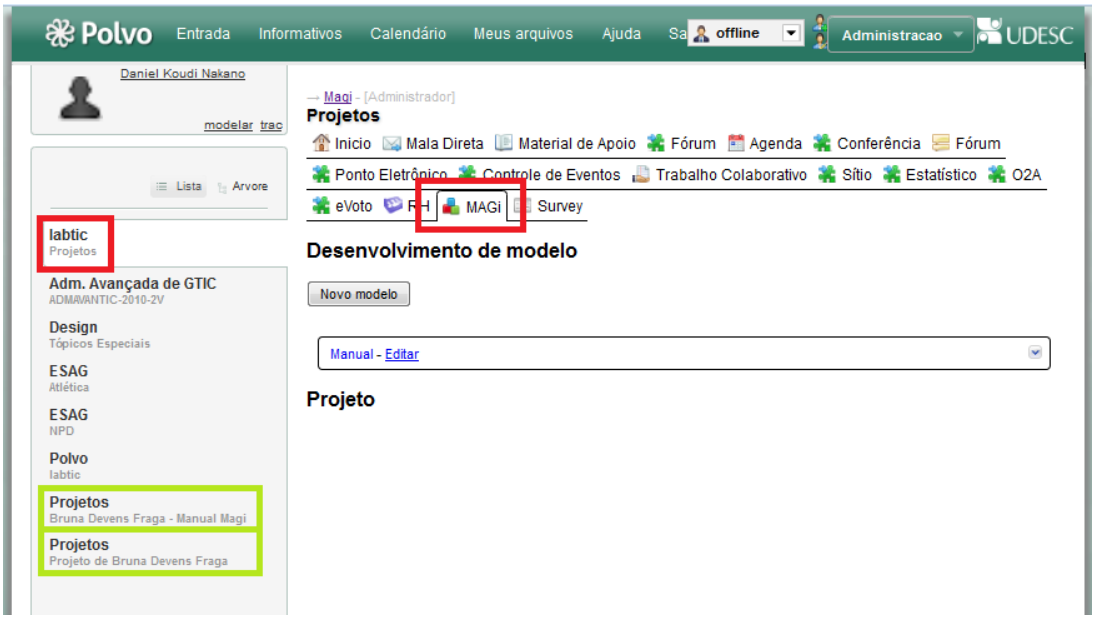


Figura 7 – Uma entidade separada para a criação do modelo

A partir do momento em que se vincula projetos a um modelo os projetos ficam dentro de entidades filhas da entidade onde foi feito o modelo.

Se o orientador permitir que o orientado modifique o fluxo que ele criou, as alterações feitas no projeto do orientado não serão transmitidas ao fluxo definido pelo autor, assim como alterações feitas no fluxo do orientador após atribuição do fluxo a um orientado, não alterarão o projeto do orientado.

Para visualizá-los, pode-se entrar na entidade do modelo e ver todas as entidades que são filhas dela (fig. 9), ou então na pagina inicial do Magi clicar no botão do lado direito da barra para ver todos os projetos vinculados àquele modelo (fig. 8).

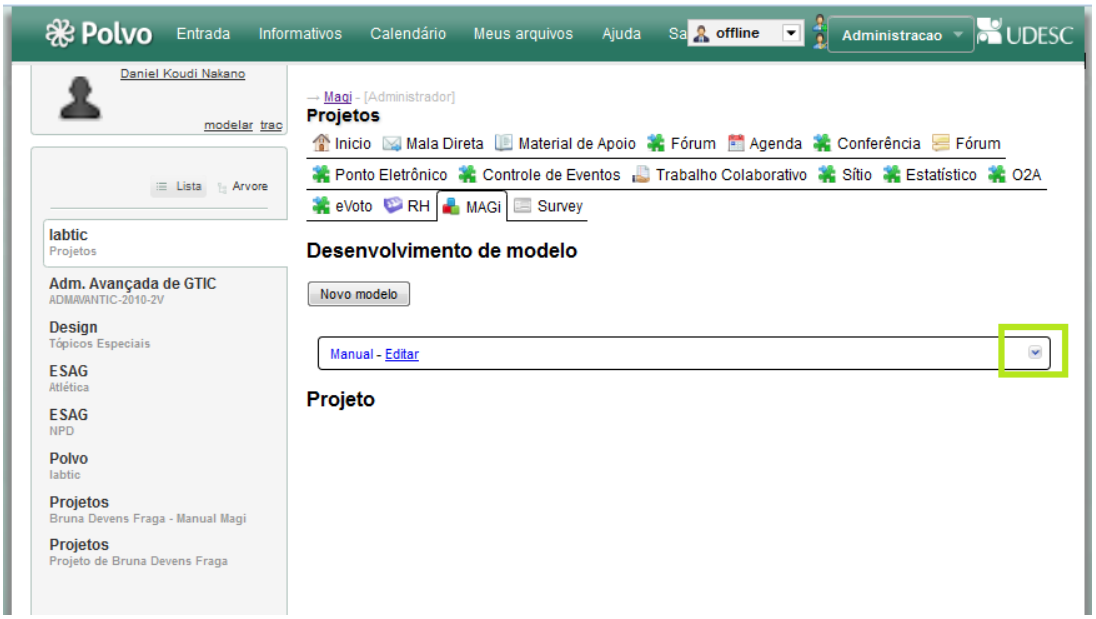


Figura 8 – Visualização dos projetos de um modelo

A criação de modelos dá-se clicando no botão "Novo modelo"no início da página inicial (fig. 10).

Uma janela aparecerá para dar nome e uma breve descrição de para que o modelo deverá servir (fig. 11).

Descreve-se então o título do modelo e a descrição sobre o que vai se tratar o modelo clicando em "Salvar"para confirmar a criação do modelo.

O novo modelo estará então disponível para edição e/ou atribuição para os possíveis orientandos (fig. 12).

Para editar o modelo clique no nome do modelo e este lhe apresentará o fluxo do modelo, caso seja um modelo novo, será apresentado um fluxo com 2(duas) caixas e 2 botões com "setas" onde as caixas representam as seções (subseções caso esteja dentro de uma seção e assim consecutivamente) e os botões servem para mostrar onde serão incluídos as novas seções (fig. 13).

Para editar os nomes das seções basta clicar no nome da seção na caixa e uma janela aparecerá para que sejam alterados o nome ou descrição da etapa (fig. 14).

Ao salvar (fig. 15)a janela se fechará e a etapa editada passará a apresentar as novas informações (fig. 16).

Para adicionar novas etapas basta clicar no botão com a seta e uma nova etapa será

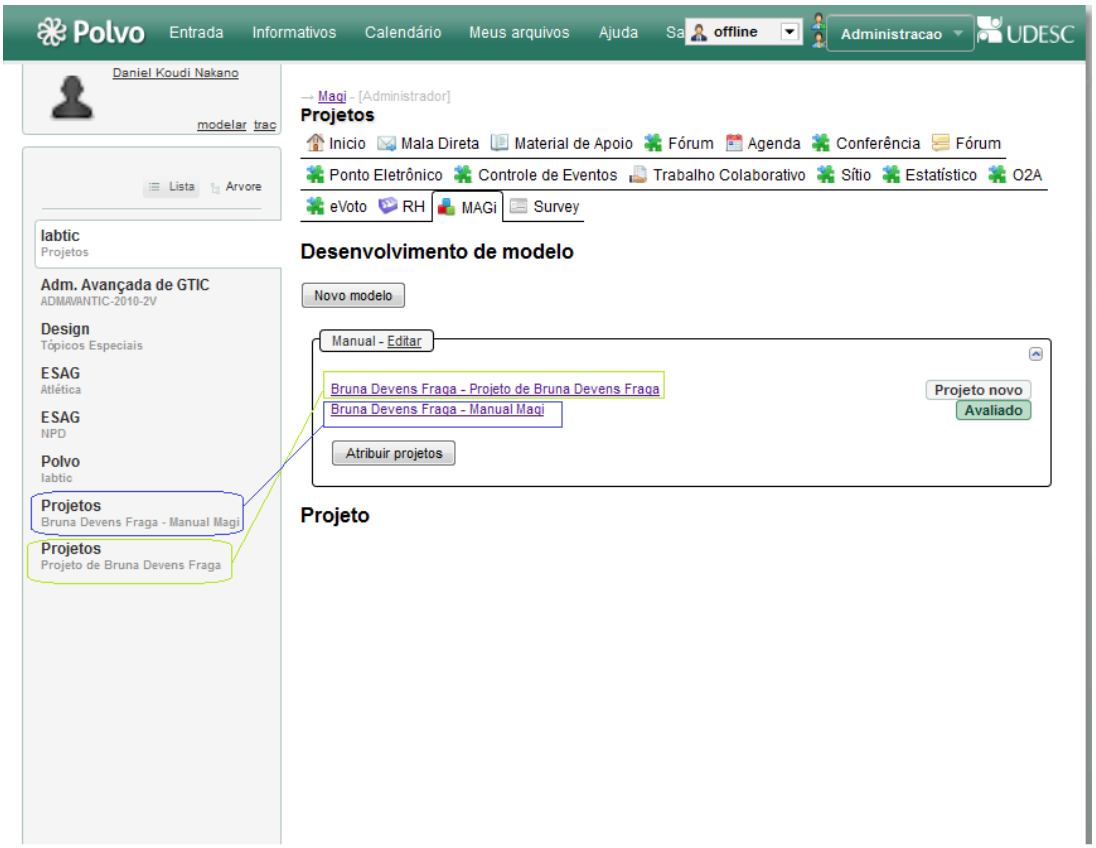


Figura 9 – Visualização dos projetos

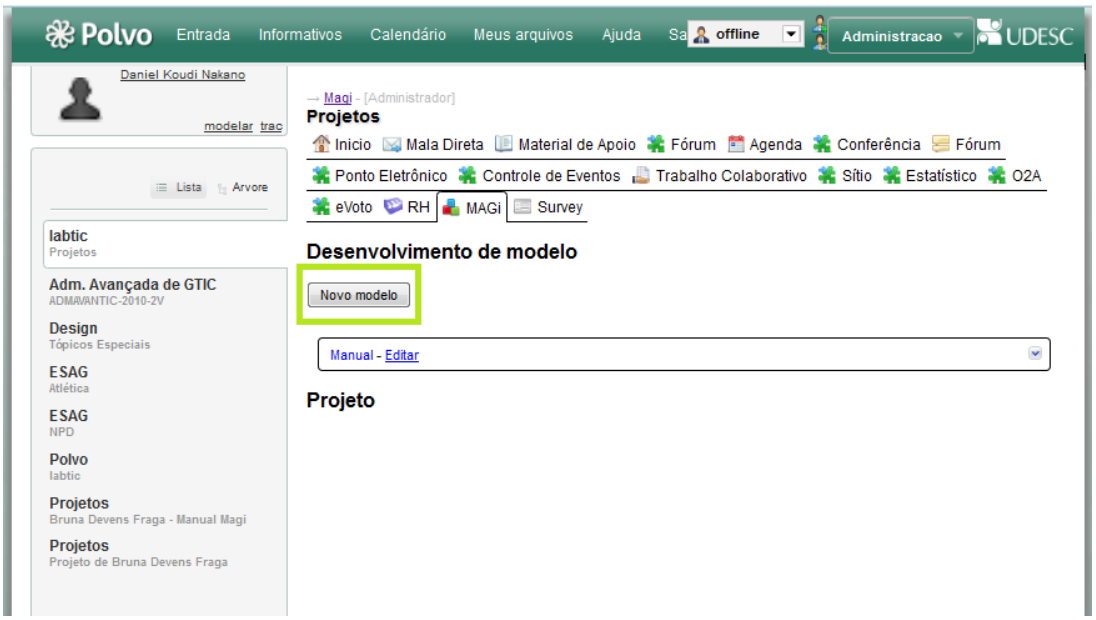


Figura 10 – Localização do botão para dar início ao processo de criação de um novo modelo

adicionada no exato local em que se localizava o botão (fig. 17).

Uma janela, como a janela da edição, aparecerá onde pode ser colocado as informações sobre a nova etapa, ao salvar esta etapa estará disponível no modelo onde o botão de adição estava (fig. 18).

É possível em cada etapa adicionar subetapas formando um fluxo interno, para isto,

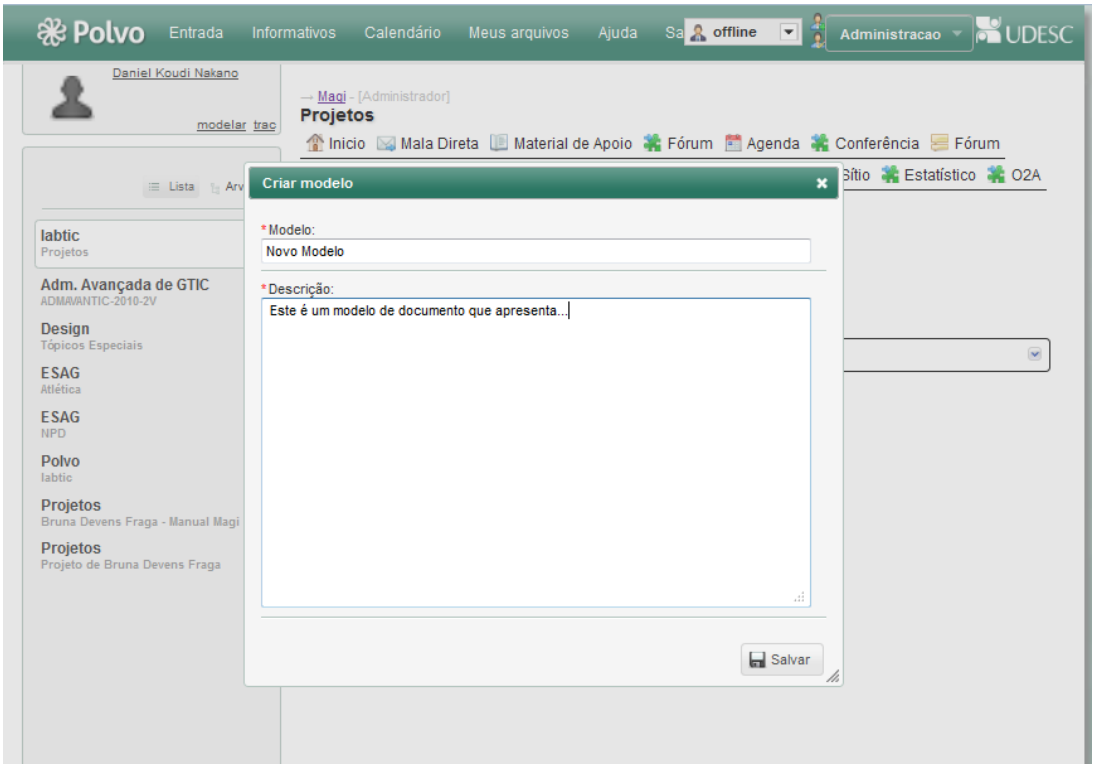


Figura 11 – Tela para inserção do nome e descrição do modelo

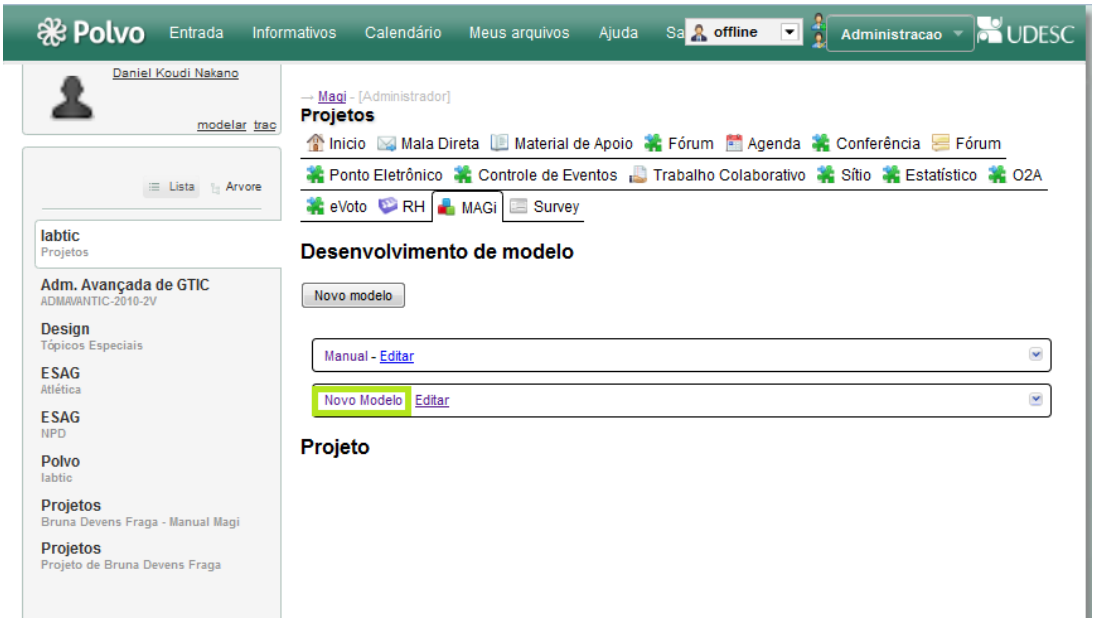


Figura 12 – Criação de um novo modelo

clica-se no nome da etapa, assim como se fosse editar suas informações, e dentro da janela há o botão "Adicionar subetapa" (fig. 19), então um novo fluxo será disponibilizado dentro da etapa (fig. 20) que pode ser editada assim como no fluxo principal, formando uma cadeia interna de subetapas(fig. 21).

Quando a etapa apresentar subetapas ela, estará marcada com um asterisco (*) para

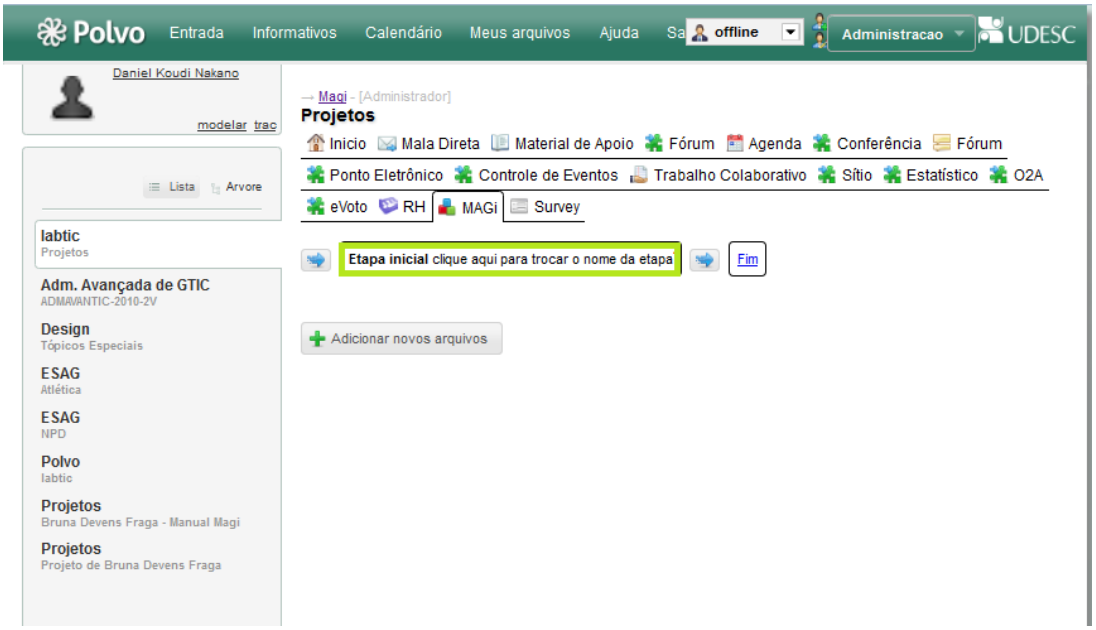


Figura 13 – Edição do fluxo do modelo

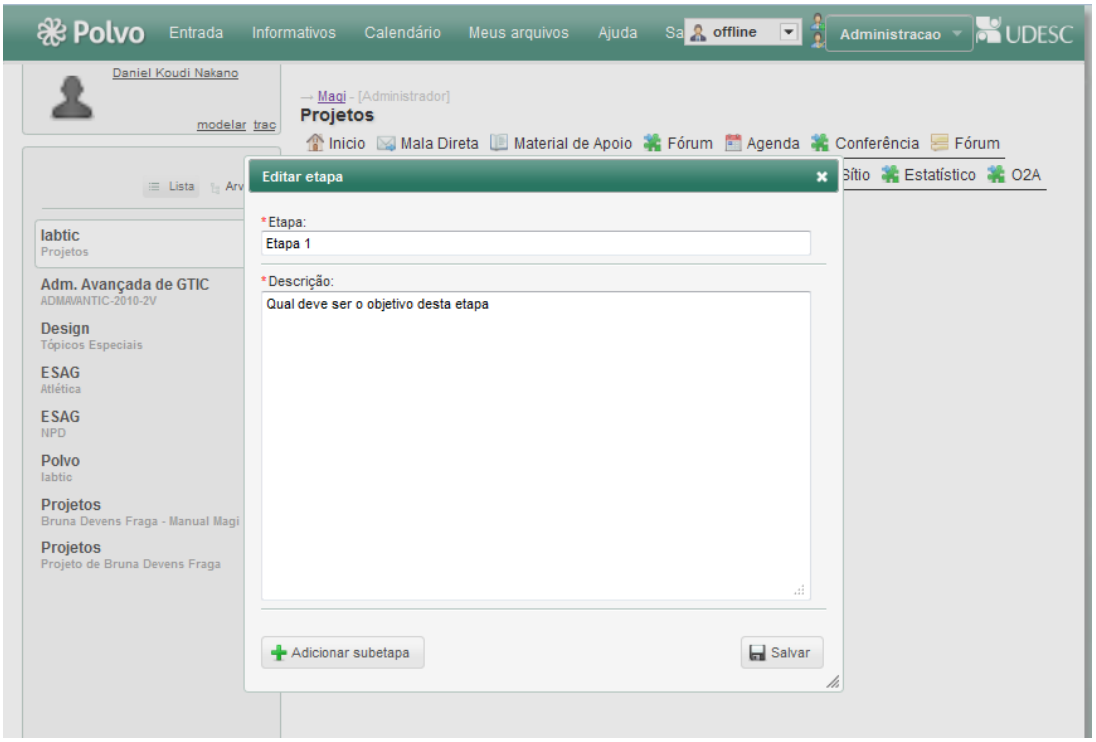


Figura 14 – Alterar nome de uma etapa

sinalizar que apresenta etapas filhas (fig. 22).

A edição pode continuar até que o modelo apresente todas as etapas e subetapas desejadas, caso haja necessidade, é possível excluir as etapas clicando no (X) no canto superior direito de cada etapa (fig. 23).

Após a finalização da estruturação do modelo, deve-se escolher os elementos textuais que o documento deve apresentar, para tal, no fluxo principal do modelo, existe uma etapa

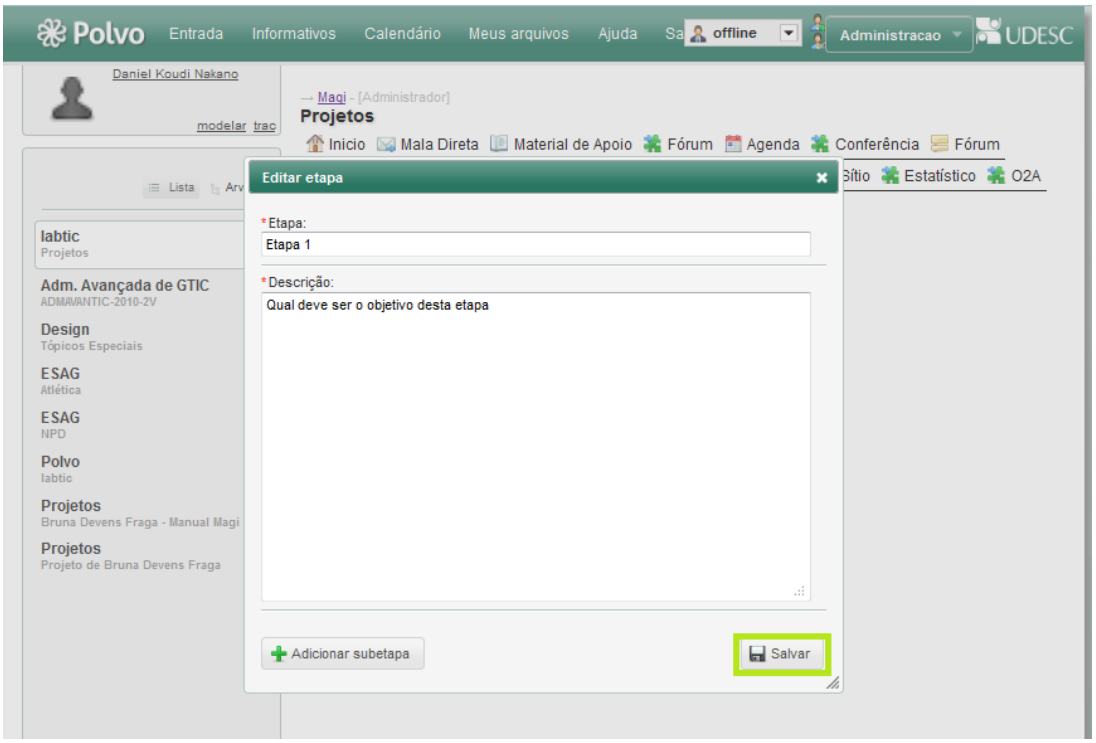


Figura 15 – Salvar alteração

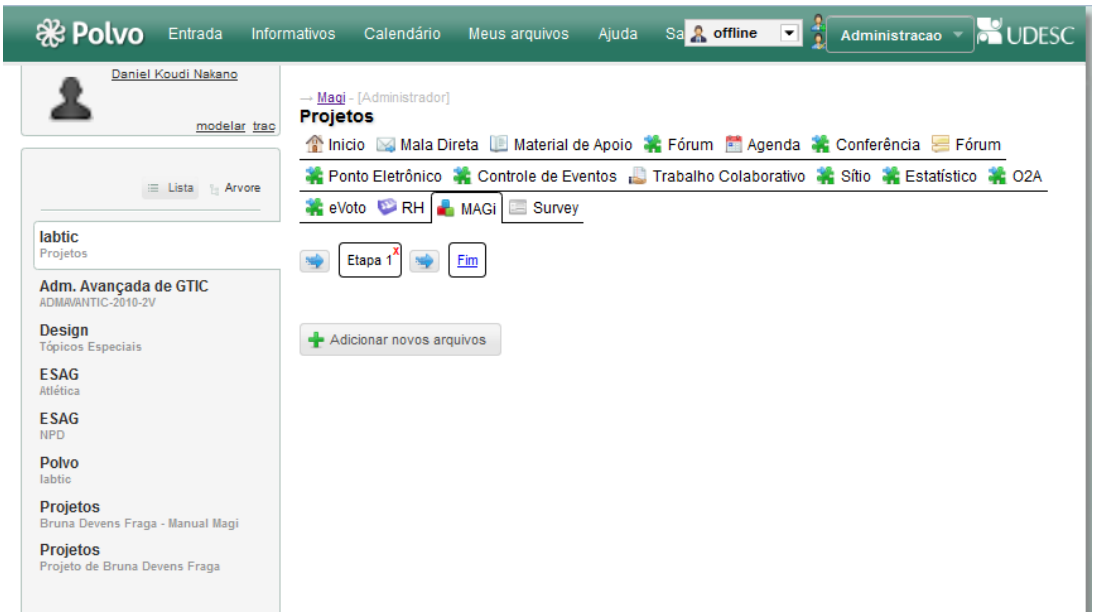


Figura 16 – Apresentação do fluxo alterado

indelével chamada “Fim”, ao clicar nele existem vários elementos textuais, que podem ser habilitados para aparecer no documento final, os elementos disponíveis são:

- Capa
- Folha de rosto
- Folha de aprovação
- Dedicatória

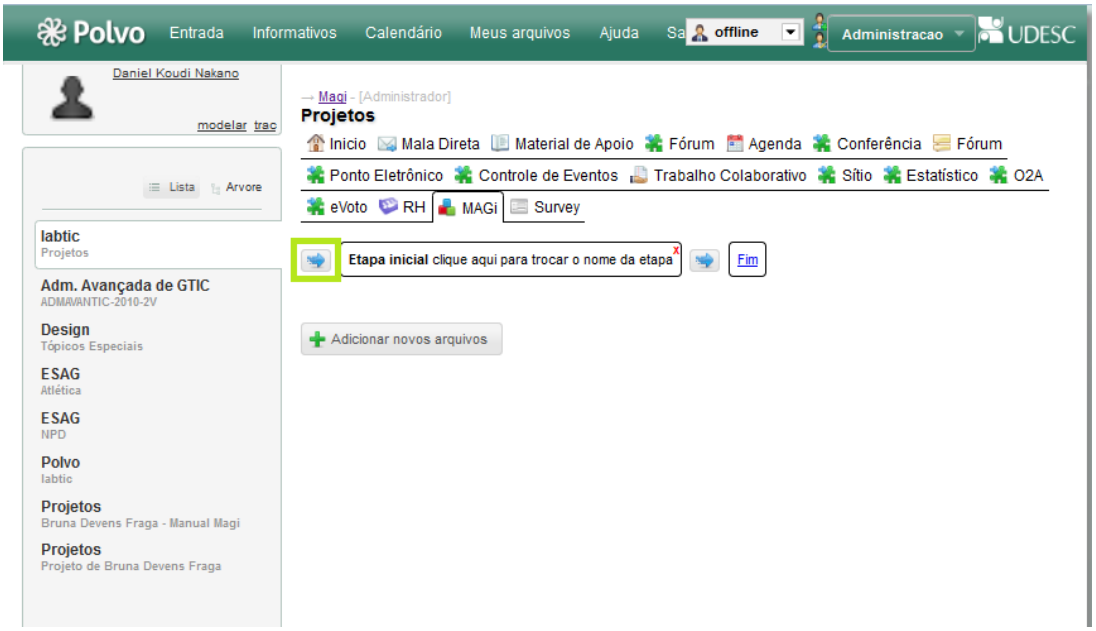


Figura 17 – Adição de uma nova etapa

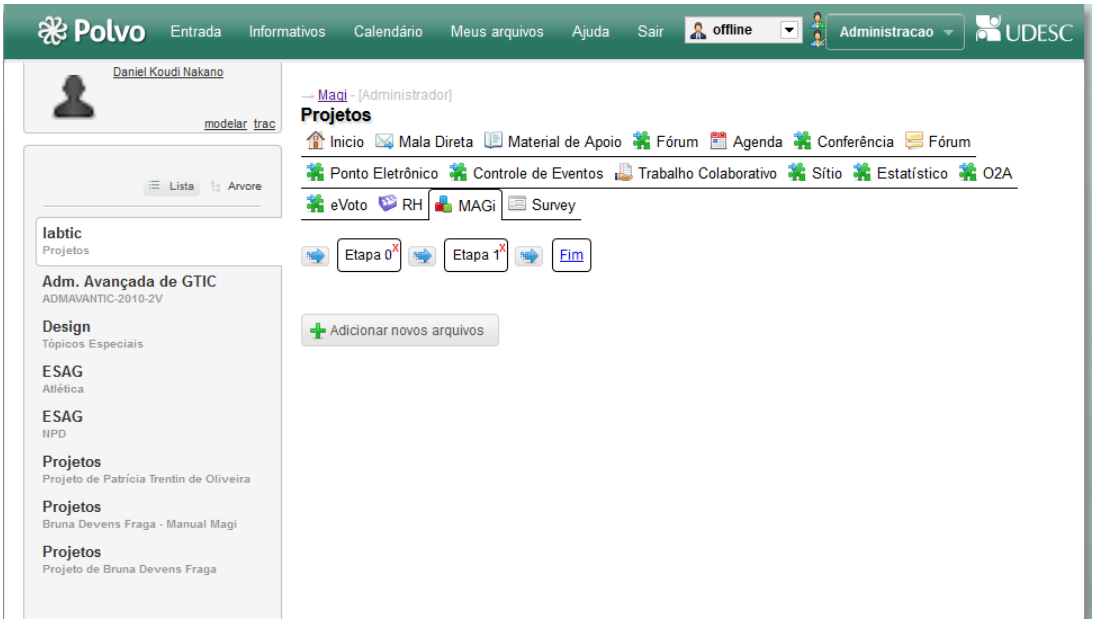


Figura 18 – Fluxo com o novo nodo

•Agradecimentos

•Epígrafe

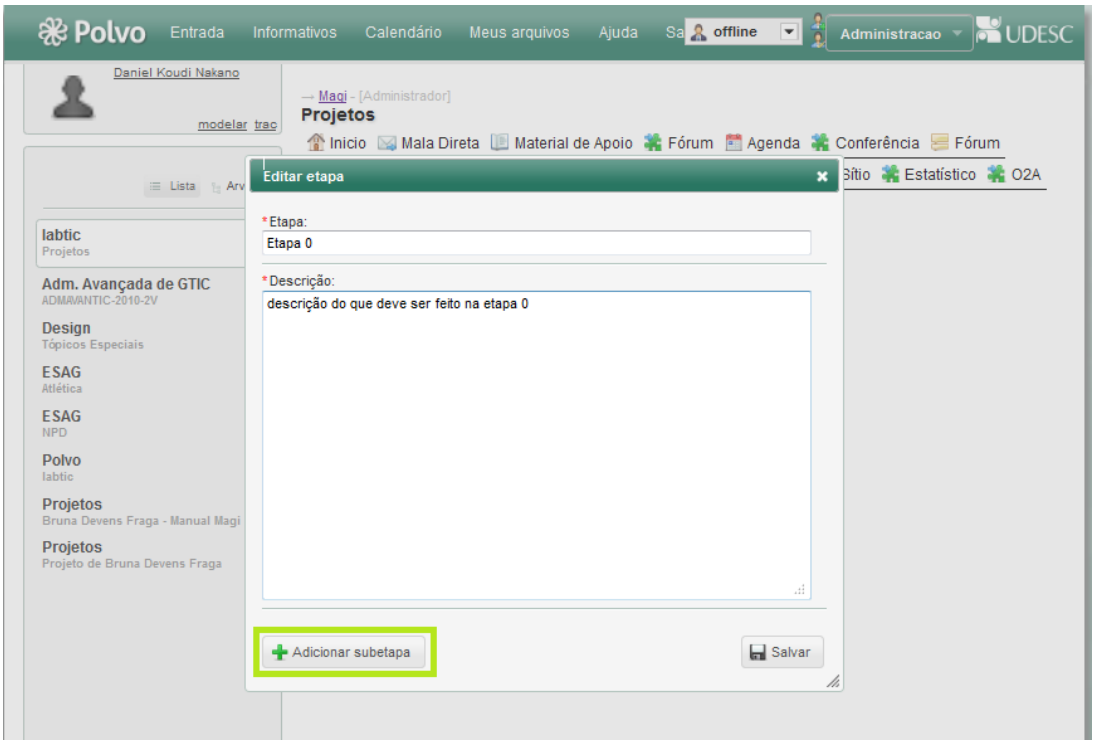


Figura 19 – Criação de subetapas

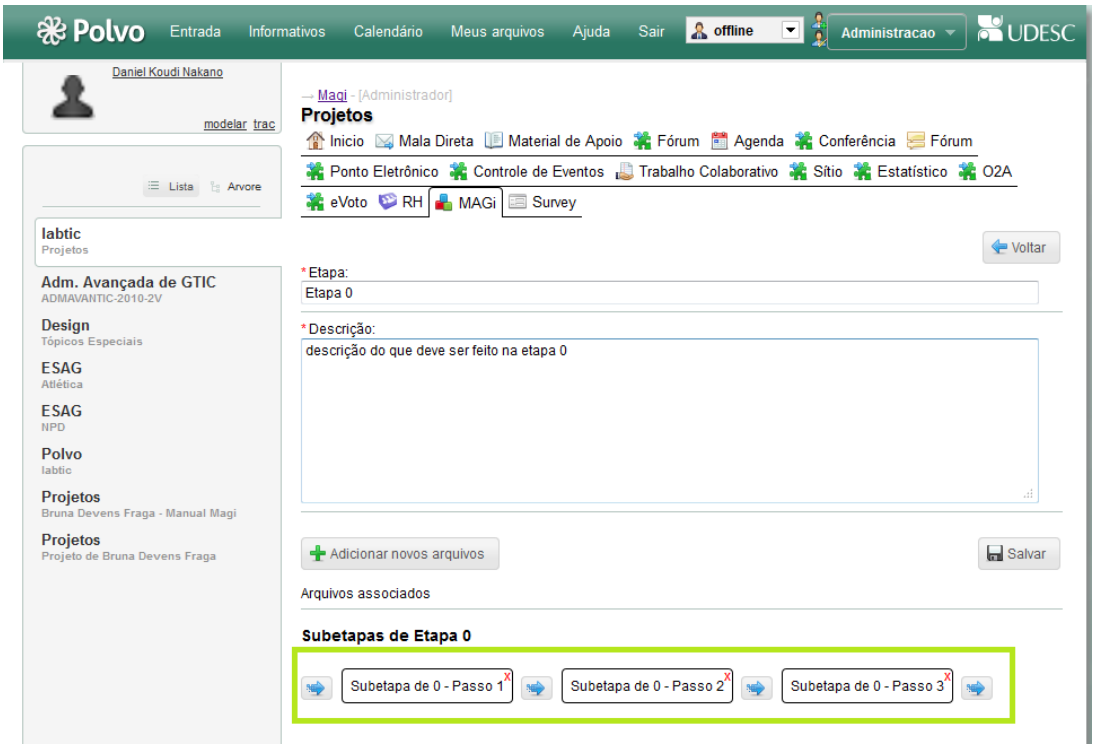


Figura 20 – Edição de etapa com a subetapa

•Resumo

•Abstract

•Sumário

•Lista de ilustrações

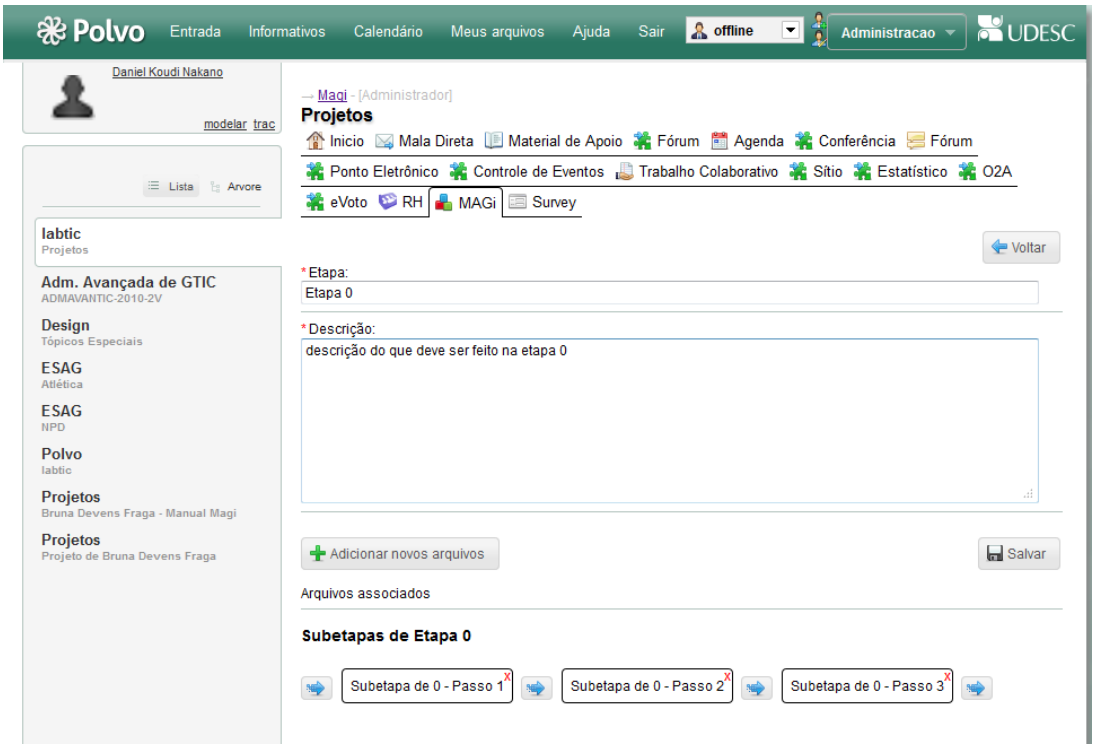


Figura 21 – Fluxo de subetapas

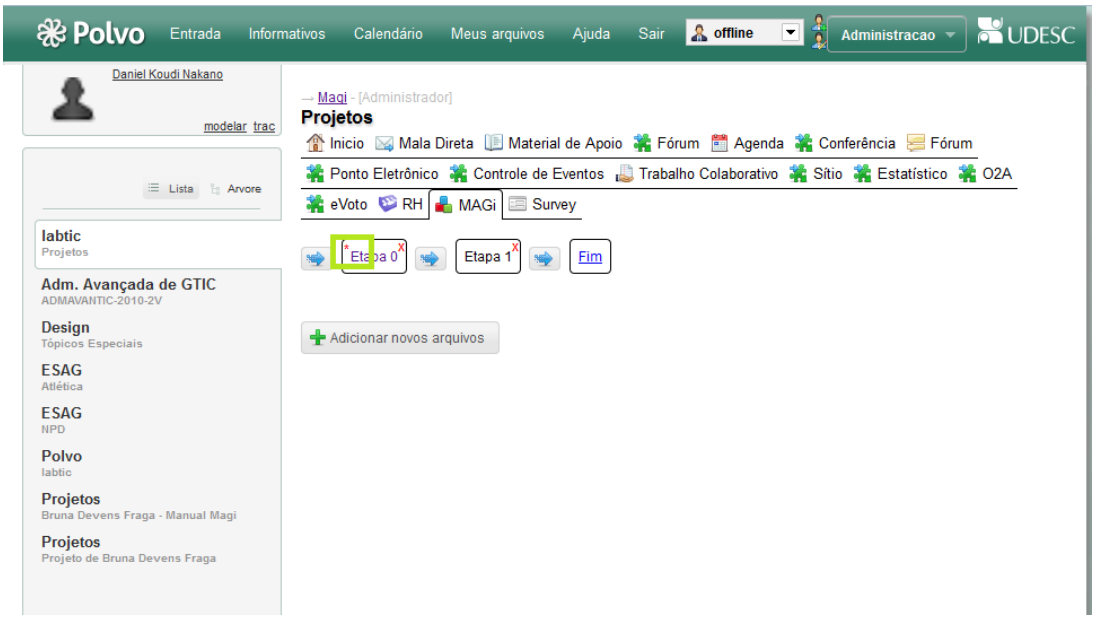


Figura 22 – Etapa com subetapas

•Lista de tabelas

•Introdução

•Conclusão

•Referências bibliográficas

•Apêndice

•* Nome do apêndice

•Anexo

•* Nome do anexo

Onde os itens "Nome do apêndice"e "Nome do anexo"só ficam disponíveis quando os itens "Apêndice"e "Anexo"estão incluídos no projeto. Para incluir um item no projeto é preciso selecionar o item desejado e clicar no botão "Enviar". O conteúdo textual destes itens só é possível ser trocado nos projetos dos orientandos (fig. 24).

B.1.2.2 Atribuição de projetos

Com o modelo pronto podemos atribui-lo a um aluno, para isso, na tela inicial do Magi, onde são listados todos os modelos clicamos no botão do lado direito(fig. 8), essa ação

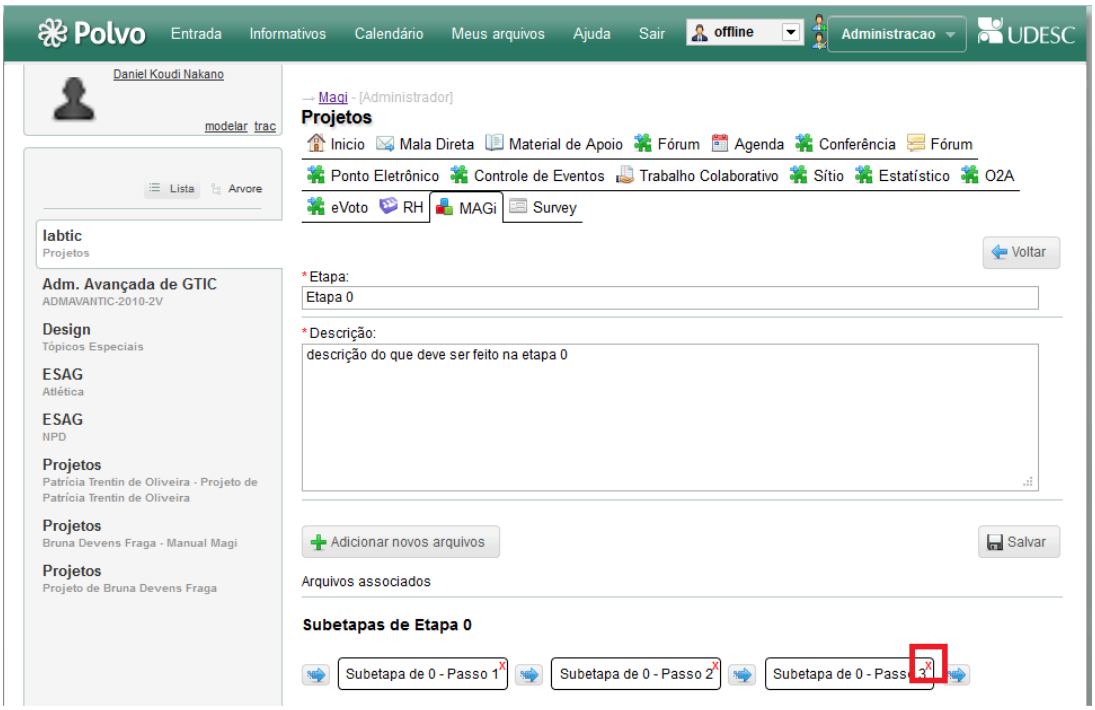


Figura 23 – Exclusão de uma etapa desnecessária

deve abrir a caixa mostrando todos os alunos que desenvolvem um projeto usando este modelo.

No final da lista há um botão "Atribuir projetos", ao clicar nele abre-se uma janela que permite pesquisar entre todos os usuários do sistema, através de uma busca por nome ou matricula (fig. 25).

Dentro da lista retornada pela pesquisa, seleciona-se o usuário do sistema que deseja orientar e na caixa no topo da janela e possível dizer se o orientando em questão poderá ou não fazer alterações na estrutura do modelo que foi definida. Ao salvar será criado uma nova entidade para o projeto, tendo por padrão o dono do modelo como orientador e o usuário escolhido como orientado (autor) do projeto (fig. 26).

O projeto agora fica disponível para o orientando desenvolver o seu documento.

A interação com o projeto é semelhante à interação com o modelo. Com algumas

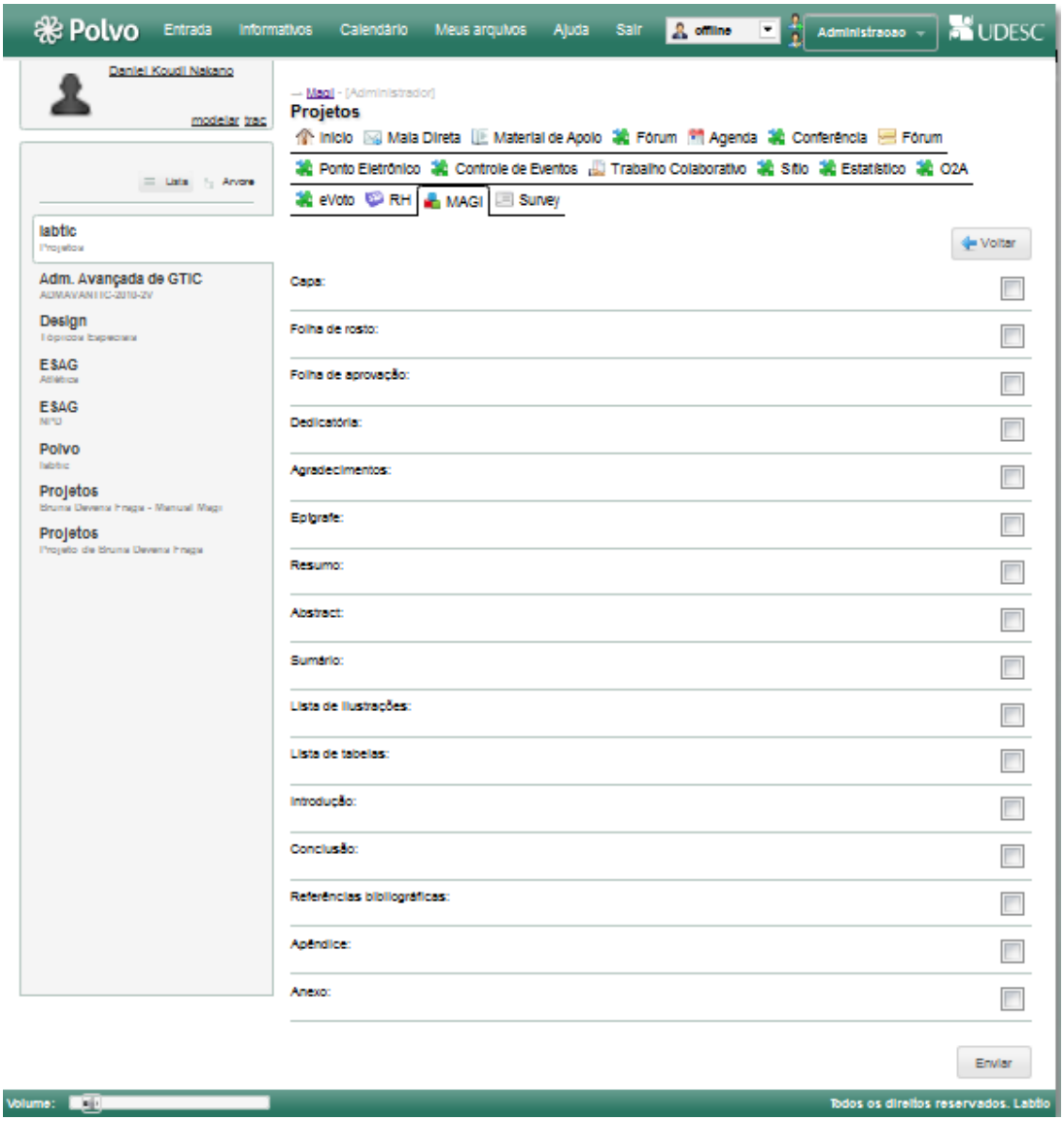


Figura 24 – Itens disponíveis para inclusão nos projetos

pequenas diferenças.

B.1.2.3 Desenvolvimento do projeto e interação com orientador

Ao entrar no sistema com o seu usuário o aluno pode acessar a entidade em que está situado o seu trabalho, o nome padrão da entidade é "Projeto de NOME DO AUTOR"e ao alterar o título do projeto a entidade passa a ter o título do trabalho como nome da entidade (fig. 27).

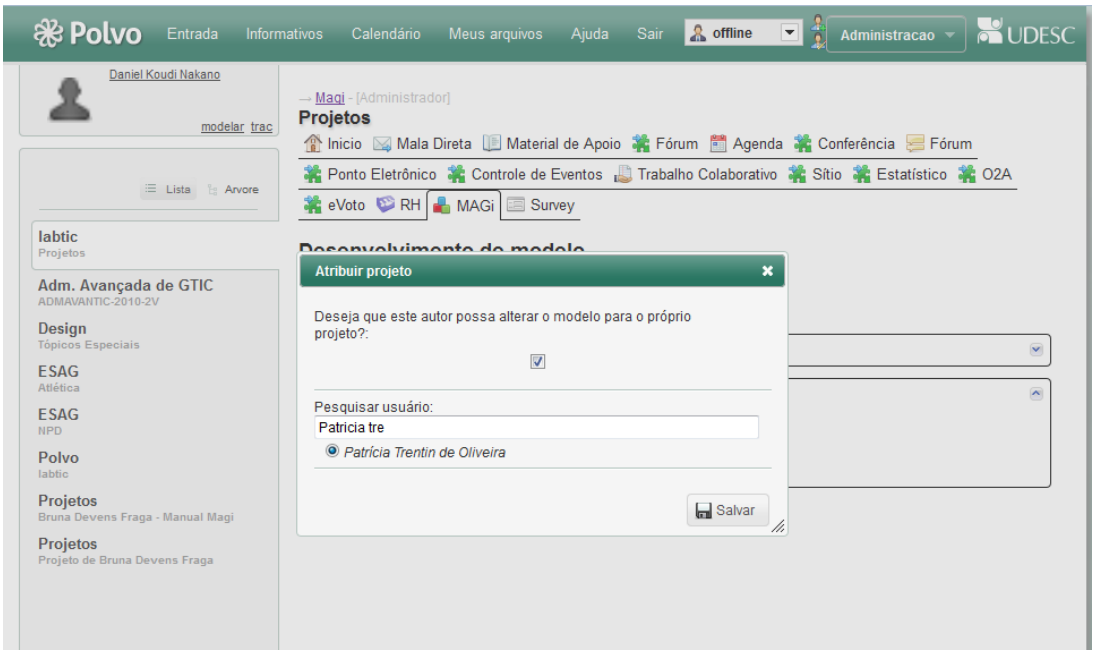


Figura 25 – Criação de um novo projeto a partir de um modelo

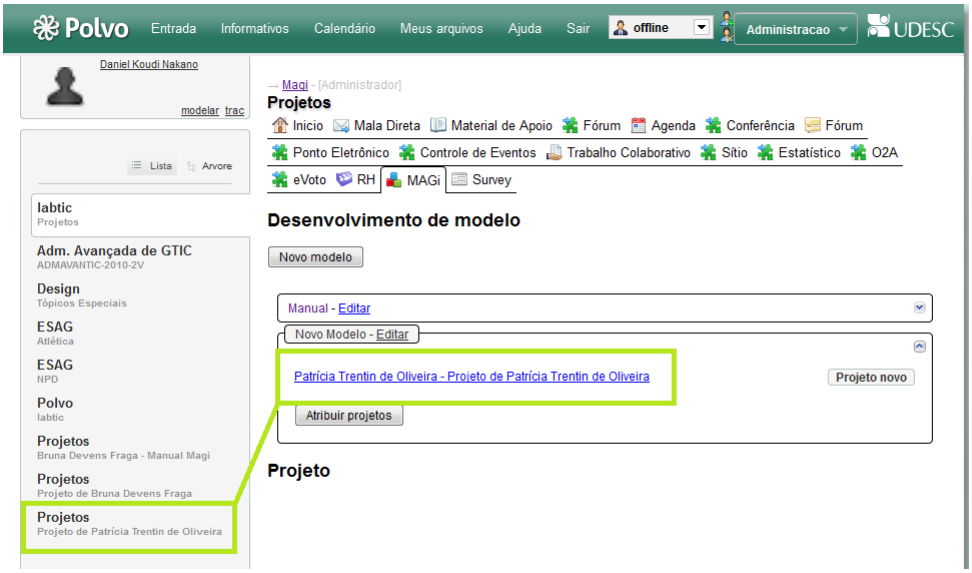


Figura 26 – Lista com os orientandos que usam este modelo e entidades dos projetos

Dentro da entidade o módulo Magi fica disponível e ao acessá-lo o orientado pode começar a editar as informações do seu projeto. Para alterar o nome do projeto, é preciso acessar o projeto (fig. 28, fig.29) clicando em seu título.

Após salvar o novo título do projeto o nome da entidade passa a ser composto por "NOME DO AUTOR - TÍTULO DO PROJETO"(fig. 30).

Junto com a alteração do título é possível adicionar outras informações relativas ao autor e ao documento. Todos os elementos pré-textuais que o orientador permitiu para

Figura 27 – Ferramenta disponível

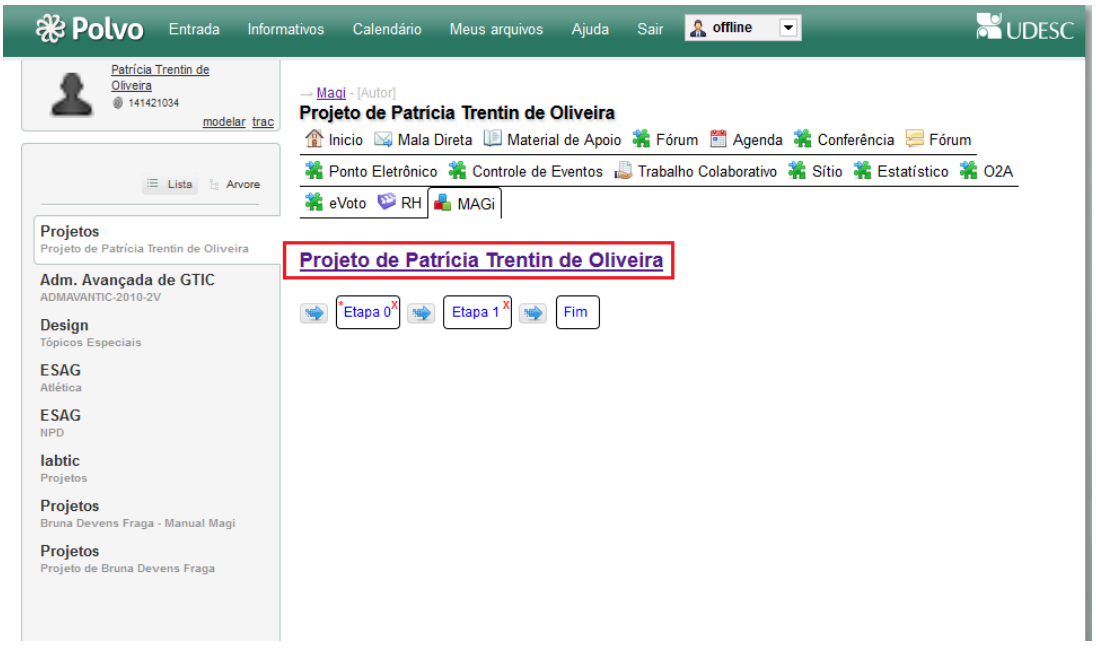


Figura 28 – Alteração do título: Ao clicar no nome abre-se a tela onde edita-se a informação

o modelo e que podem ter seu texto alterado também localizam-se nesta parte (fig. 31). Os elementos que podem ter seu texto alterado são:

- •Título do projeto
- •Curso
- •Departamento
- •Sigla do departamento
- •Instituição
- •Sigla da instituição
- •Local
- •Data de defesa
- •Folha de rosto
- •Folha de aprovação
- •Dedicatória
- •Agradecimentos
- •Epígrafe
- •Resumo

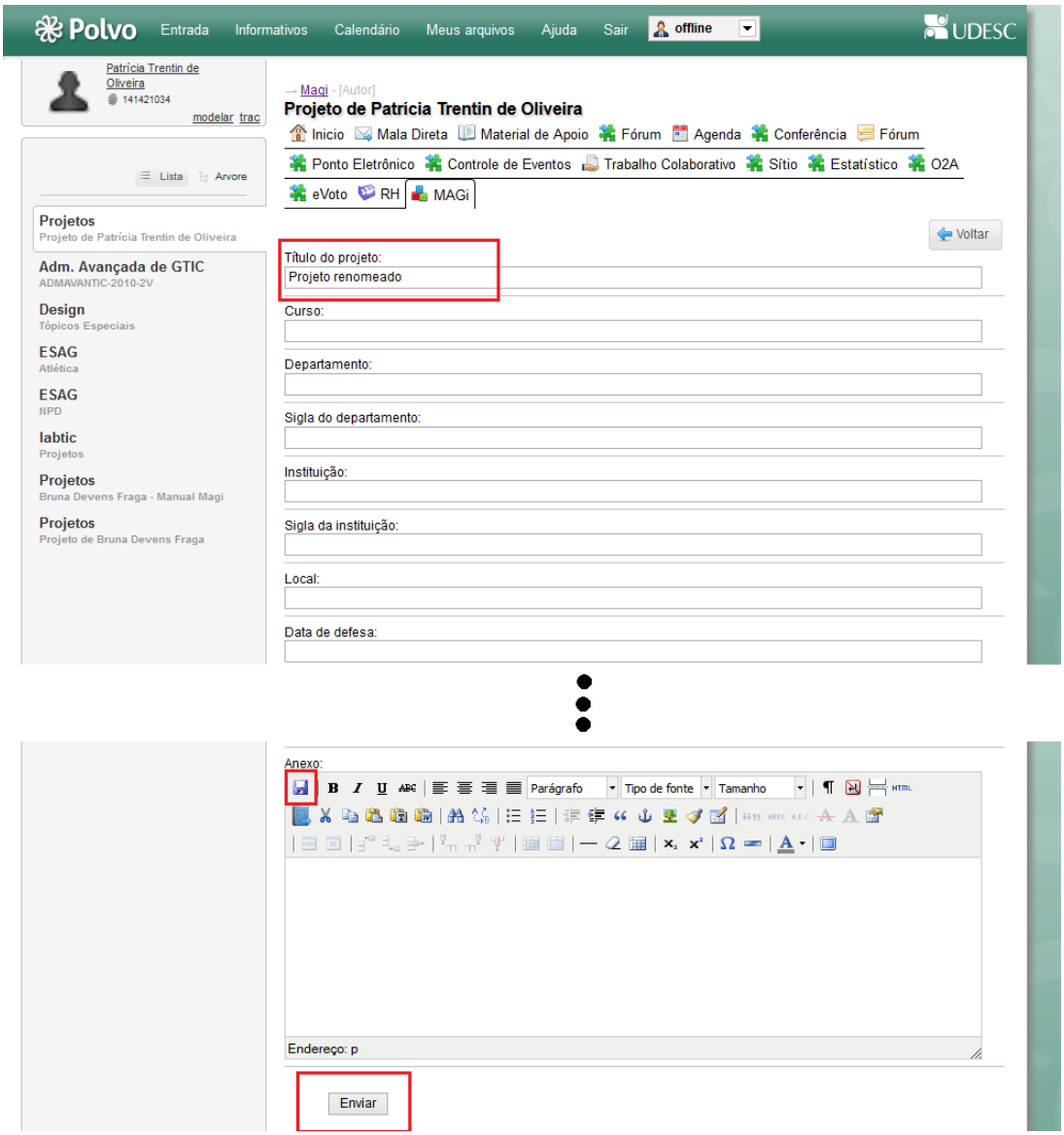


Figura 29 – Alteração do título: O novo nome do projeto

- Abstract
- Introdução
- Conclusão
- Apêndice
- Anexo

Caso hajam apêndices ou anexos cadastrados, estes também aparecerão aqui, identificados por seus títulos.

As etapas do fluxo principal definidos no modelo, serão os capítulos do documento final, as subetapas deles serão as sessões e assim consecutivamente (fig. **??**). Tanto os textos do "Título"quando o "Texto"serão passados ao documento final gerado. Tanto para as capítulos quanto seções.

O texto desenvolvido em cada etapa ou subetapa pode ser acompanhado ou alterado pelos membros da entidade que tiverem a ação que permita, na parte inferior de cada tela de edição de conteúdo há um parte para textos que não serão incorporados no documento final e que podem ser usados para o orientador dar sugestões ou fazer observações, o orientando só pode observar o texto neste campo (figs. 34, 35).

B.1.2.4 Finalização de projeto

Para terminar o projeto, quando o orientador decidir que o projeto esta completo, ele pode sinalizar isso ao orientando clicando no botão "Pronto"disponível no fluxo principal do projeto (fig. 36).

Finalizando o projeto o Orientador pode colocar mais alguma observação para o aluno para fazer a alteração antes de baixar o documento final (fig. (fig. 37)).

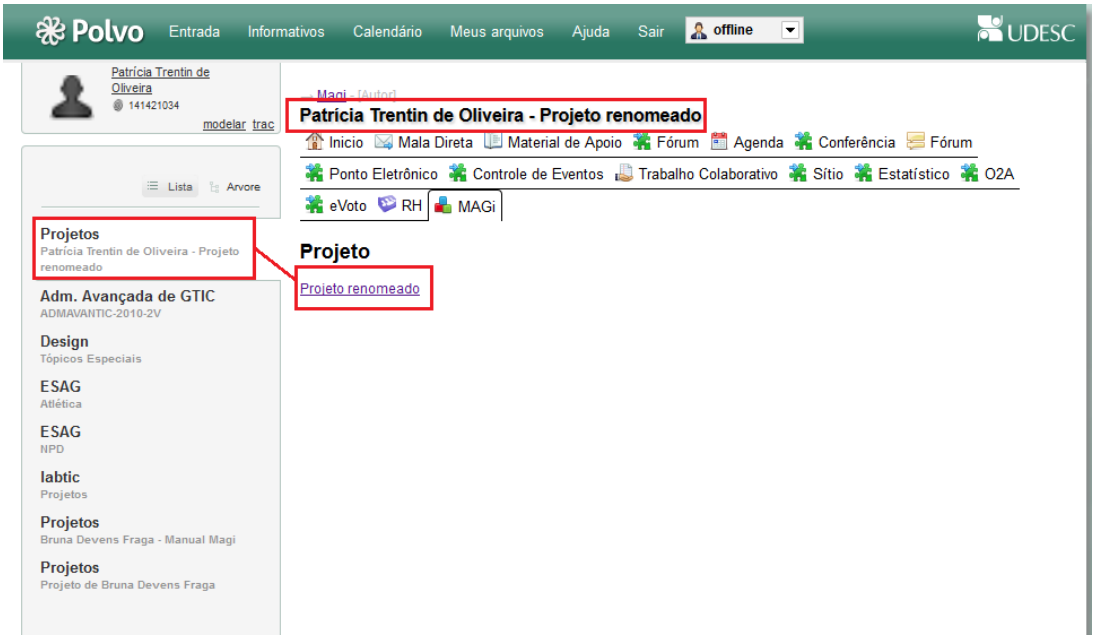


Figura 30 – Entidade e projeto com os nomes alterados

Uma vez o orientador sinalizando o projeto como pronto o orientado é avisado por meio de um aviso do mesmo modo que o aviso de cometários como o texto "Aprovado pelo orientador"e ficará disponível para ele 2 (duas) opções "Download bibTEX"e "Download LaTeX"(fig. 38).

- Download bibTEX: arquivo formato em bib com as referências das citações utilizadas no texto.
- Download LaTeX: arquivo em formato tex com o projeto na estrutura em que foi desenvolvida.

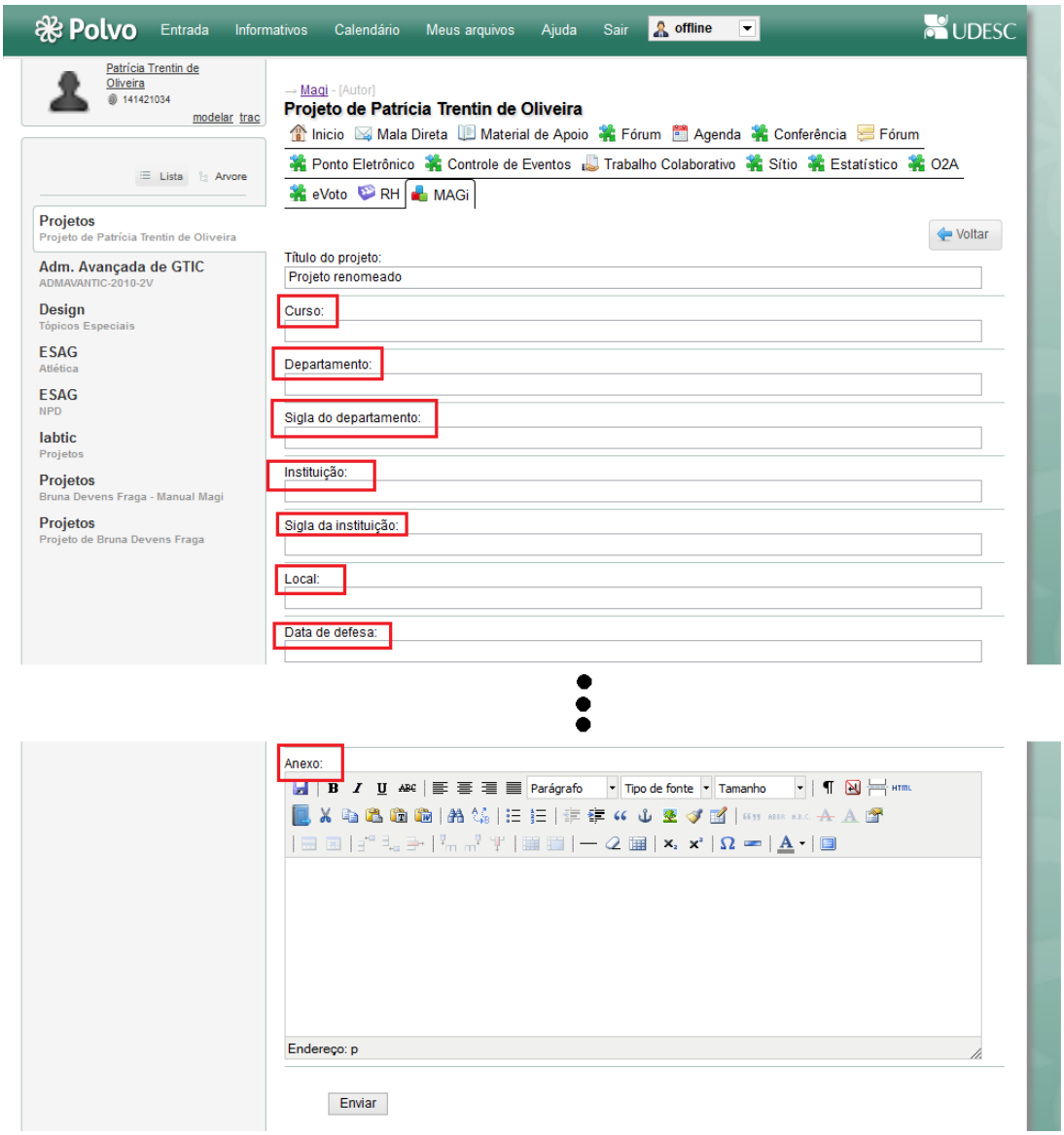


Figura 31 – Edição de texto de outros elementos do documento e informações sobre o projeto

B.1.3 Funções

Através destes BPM podemos ver as funcionalidades do Magi-Casper, e os quais foram a base para fazer este manual.

- Orientar projeto (fig. 39);
- Criar fluxo de desenvolvimento de modelo(fig. 40);

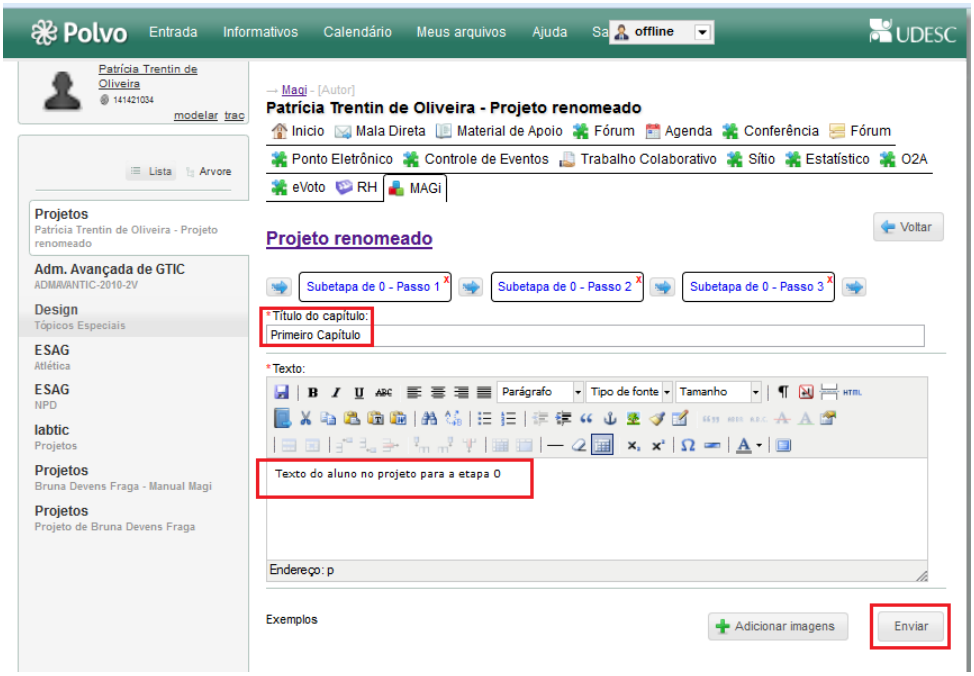


Figura 32 – Etapas

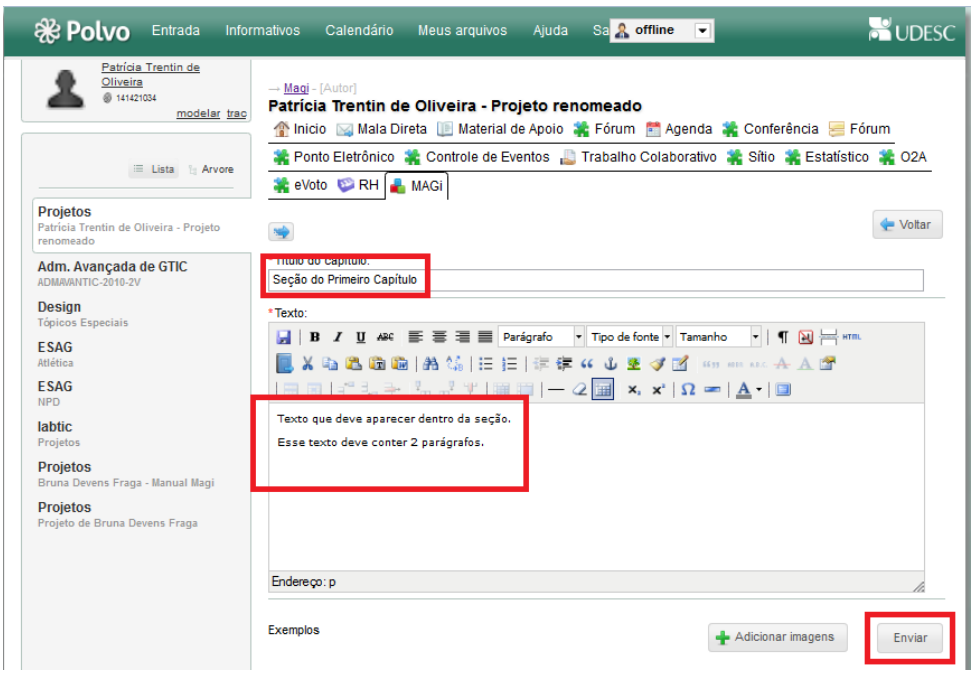


Figura 33 – Subetapas

•Adicionar referências e citações (fig. 41);

•Definir textos de elementos textuais da monografia (fig. 42);

•Compor o texto (conteúdo)(fig. 43);

•Acompanhar a composição do texto (fig. 44);

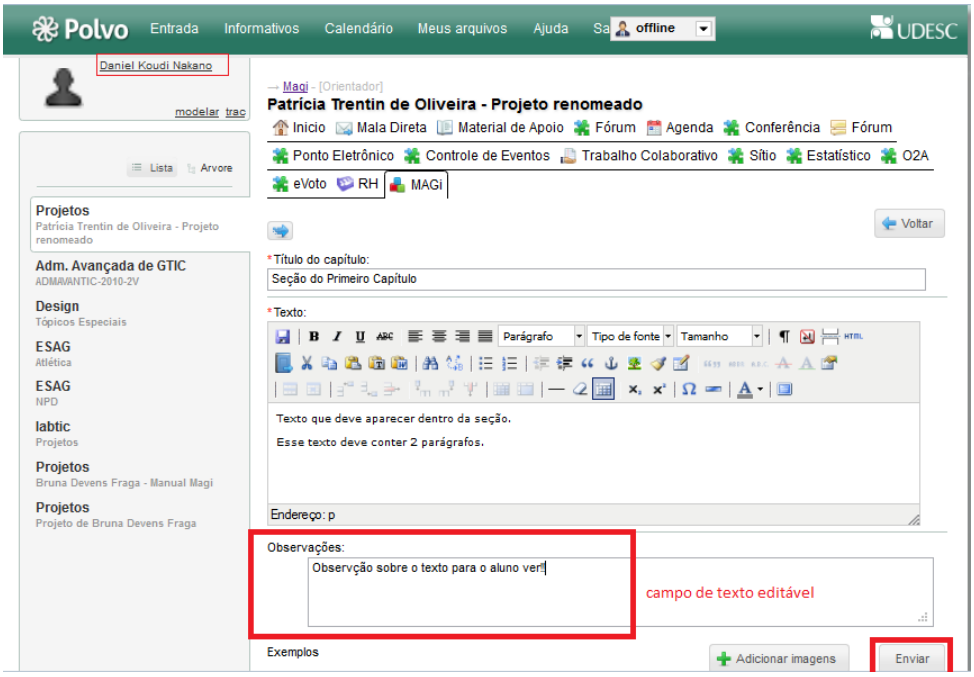


Figura 34 – Observações do orientador (Visão orientador)

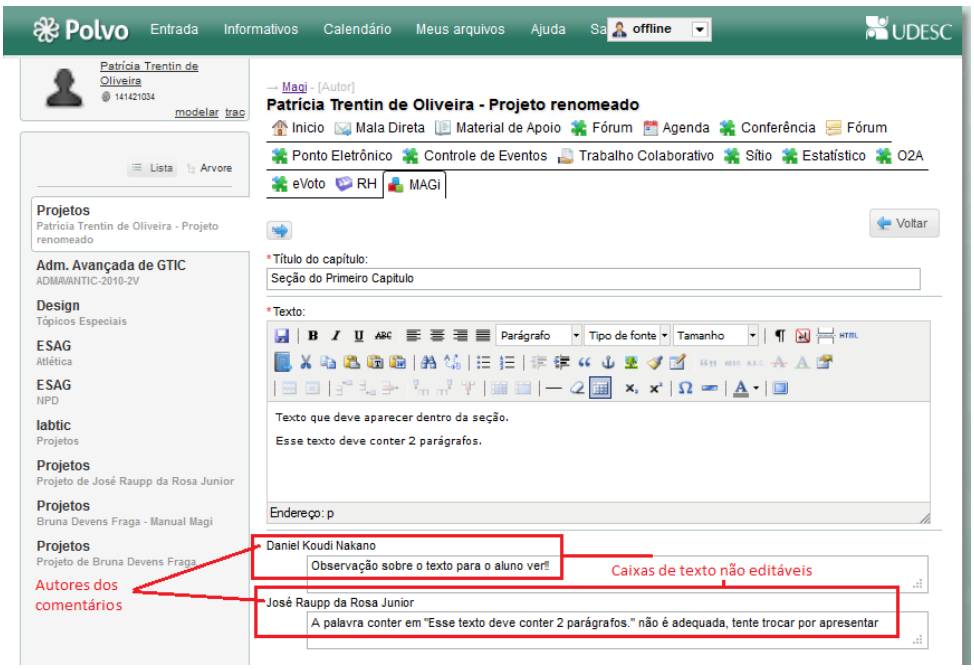


Figura 35 – Observações do orientador (Visão orientado)

•Finalizar projeto (fig. 45);

•Adicionar arquivos como exemplos ou para explicação de qualquer natureza (fig. 6);

B.1.4 Resultado esperado

Ao final do processo, o objetivo principal do módulo Magi-Casper é o documento gerado em formato tex que segue ao ser compilado resulta em um documento em formato ps ou pdf dentro das regras da ABNT e com as citações e referências também dentro das normas.

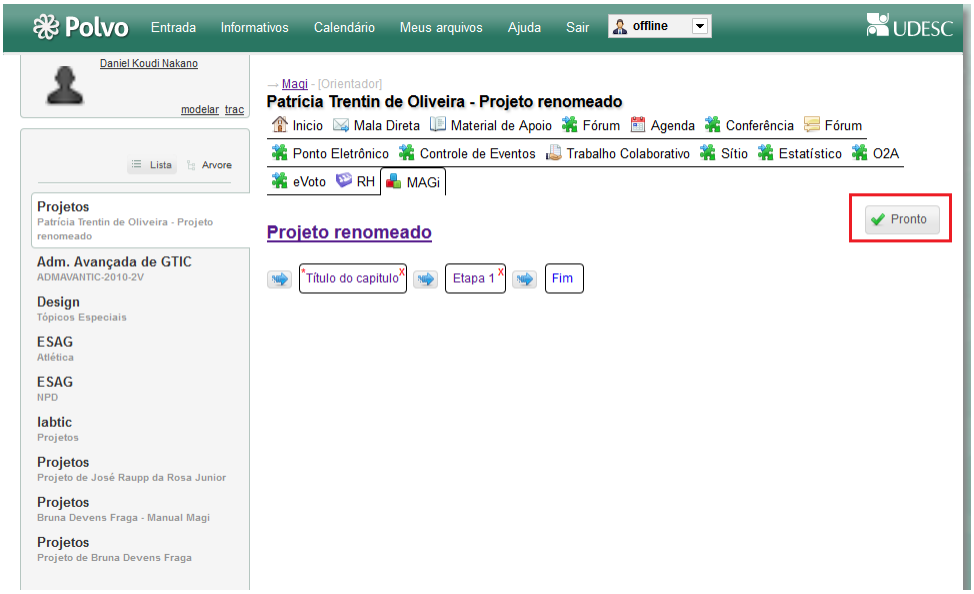


Figura 36 – Finalização do projeto

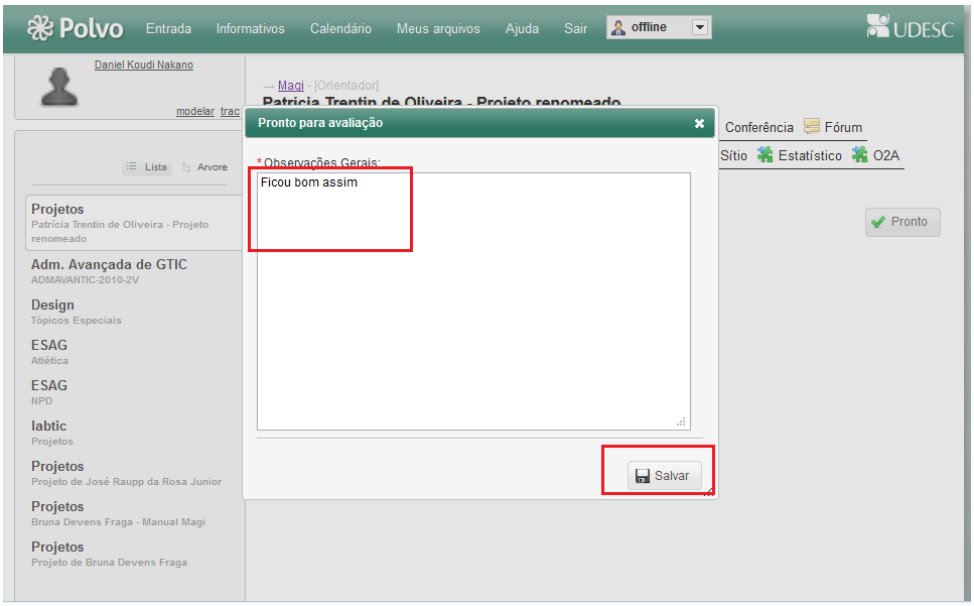


Figura 37 – Observação de conclusão

Como objetivo secundário, destaca-se o controle dos orientandos e a interação entre os eles e orientadores ou outros membros do desenvolvimento do projeto sem a necessidade de encontro físico.

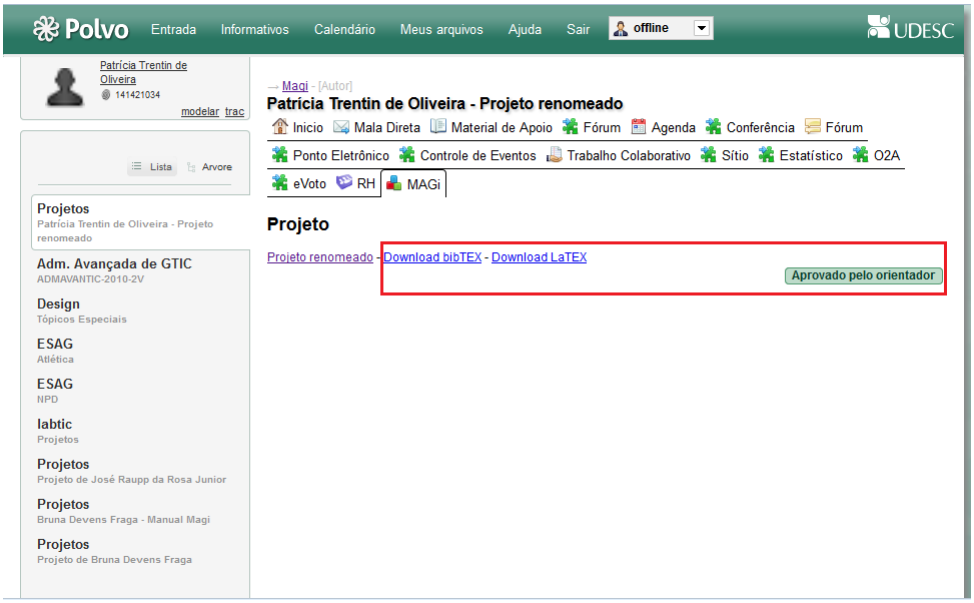


Figura 38 – Finalização do projeto links para baixar o documento final

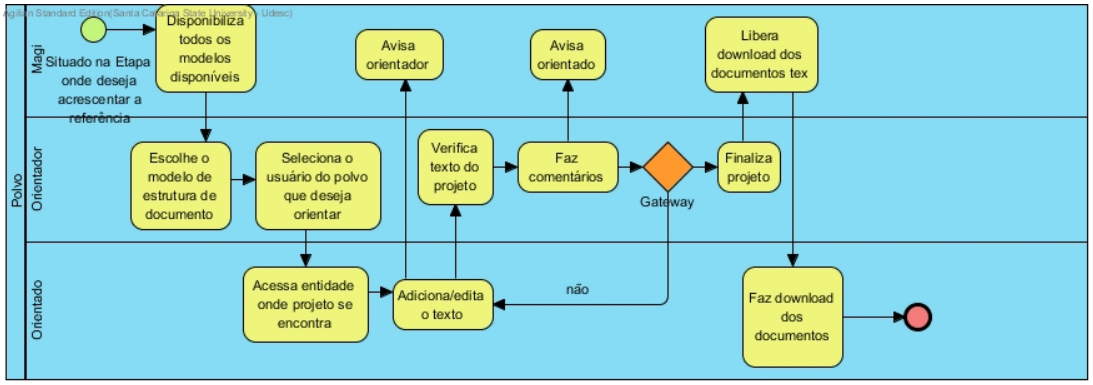


Figura 39 – Orientar projeto

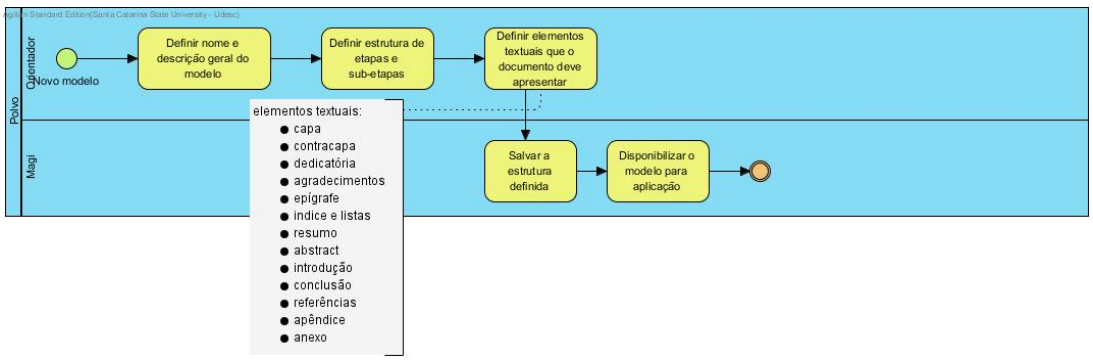


Figura 40 – Criar fluxo de desenvolvimento de modelo

B.2 Requisitos para funcionamento

B.2.1 Cliente

Para trabalhar com documentos LaTeX são necessários alguns requisitos no sistema operacional no qual trabalham. Todos os recursos que serão listados servem tantos para

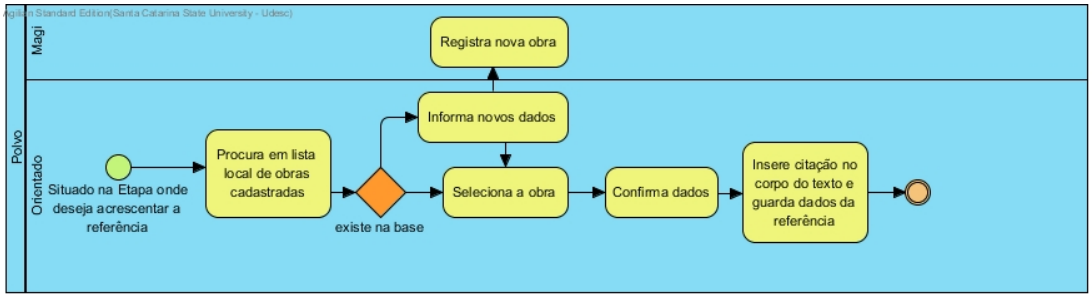


Figura 41 – Adicionar referências e citações

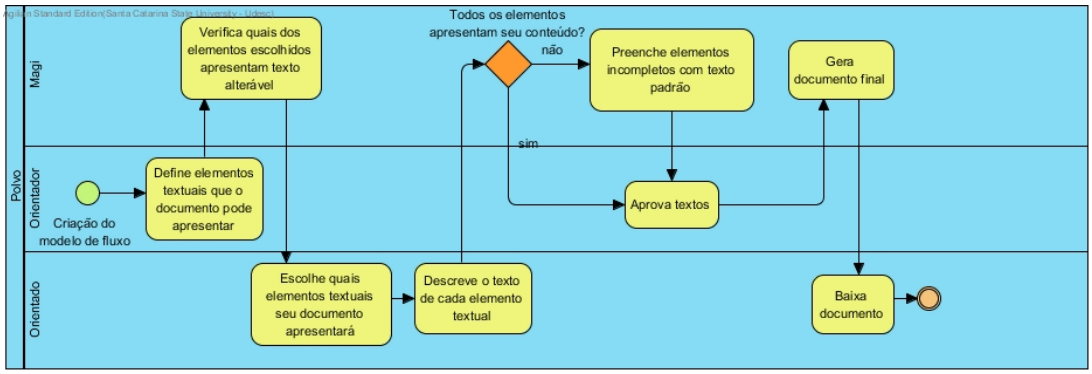


Figura 42 – Definir textos de elementos textuais da monografia

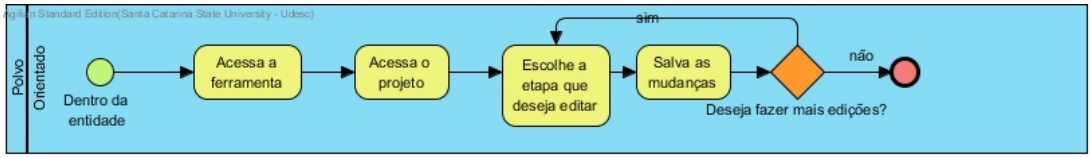


Figura 43 – Edição do projeto

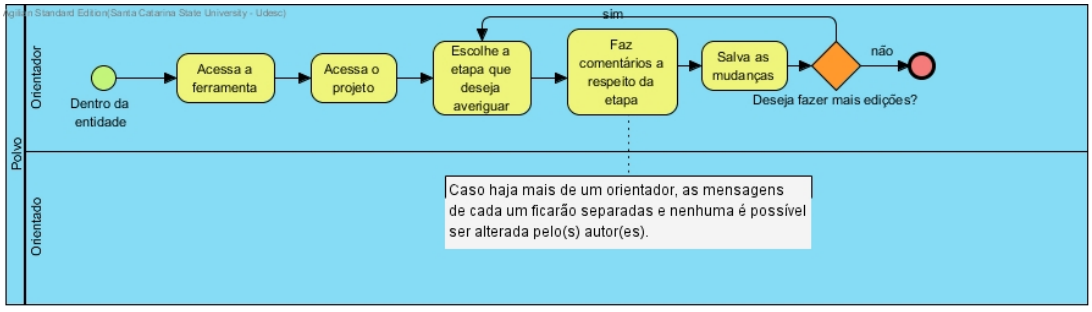


Figura 44 – Acompanhar a composição do texto

plataforma Windows quanto Linux.

Para que o documento final possa ser gerado em formato pdf/ps é necessário ter instalado na maquina do orientando Um conjunto de programas:

- MiKTeX: Disponível em <http://www.miktex.org/ >; Responsável por preparar o sistema para o tex e gerenciar pacotes;
- Ghostscript: Disponível em <http://pages.cs.wisc.edu/ ghost/>; Responsável por renderizar e rasterizar imagens em Post Script (alternativa a PDF);
- GSView: Disponível em <http://www.ghostscript.com/>; Necessário para visualizar os documentos;
- abnTeX: Disponível em <http://abntex.codigolivre.org.br/ >; Para adequar às normas da ABNT;

Opcionalmente, como ferramentas de auxilio, os seguintes componentes:

- Texmaker: Disponível em <http://www.xm1math.net/texmaker/ >; Interface para trabalho com o documento tex ;
- R: Disponível em <http://www.r-project.org/ >; Para trabalhar com gráficos e funções estatísticas ;
- Asymptote: Disponível em <http://asymptote.sourceforge.net/ >; Para trabalhar com funções matemáticas ;

B.2.2 Servidor

O módulo Magi-Casper precisa que o servidor apresente os pacotes para tratamento de XML e XSLT, pois a transformação do documento é feita com um arquivo de transformação XSLT.

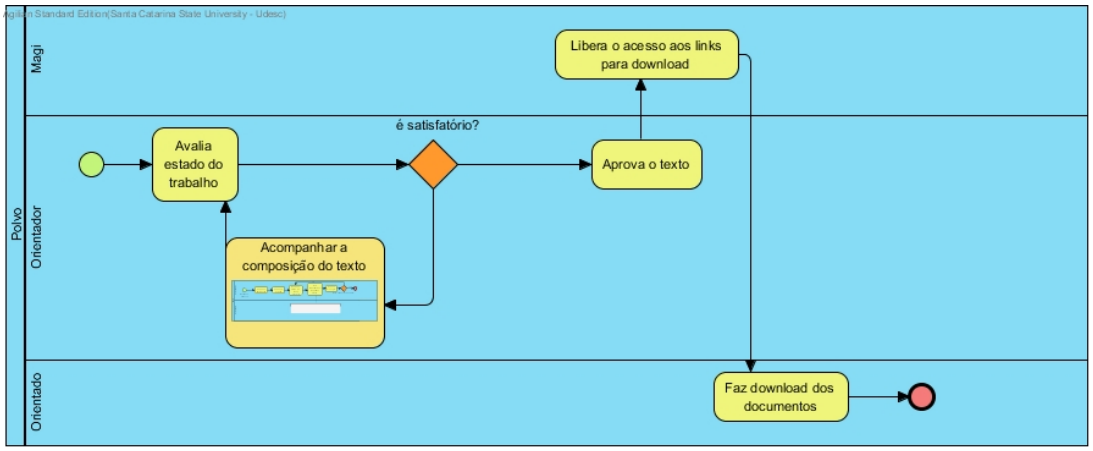


Figura 45 – Finalizar projeto

O Magi-Casper apresenta um modelo XSLT para a transformação dos dados fornecidos ao sistema em documento no formato de um TCC, é possível fazer transformadores para outros modelos de documento como, artigo ou carta por exemplo, para isso é necessário um conhecimento em XML/XSLT.

B.2.2.1 Modelo de documento

Para gerar outros modelos de documento basta gerar um arquivo de transformação XSLT. Os elementos existentes no XML fonte do projeto são:

- **<projeto>**
  - <config>
    - <capa label="Capa"></capa>
    - <folhaderosto label="Folha de rosto">Texto da Folha de rosto</folhaderosto>
    - <aprovacao label="Folha de aprovação"></aprovacao>
    - <dedicatoria label="Dedicatória"></dedicatoria>
    - <agradecimentos label="Agradecimentos"></agradecimentos>
    - <epigrafe label="Epígrafe"></epigrafe>
    - <resumo label="Resumo"></resumo>
    - <abstract label="Abstract"></abstract>
    - <sumario label="Sumário"></sumario>
    - <ilustracoes label="Lista de ilustrações"></ilustracoes>
    - <tabelas label="Lista de tabelas"></tabelas>
    - <introducao label="Introdução"> ds </introducao>
    - <conclusao label="Conclusão"></conclusao>
    - <refs label="Referências bibliográficas"></refs>
    - <apendice label="Apêndice"></apendice>
    - <anexo label="Anexo"></anexo>
    - <apendice></apendice>
    - <anexo></anexo>
  - </config>
  - <membros>
    - <orientador nome="Nome orientador" prof="profissão orientador" titulo="Titulação orientador" grad="Graduação orientador" instituicao="Instituição orientador"></orientador>
    - <autor nome="Nome autor" prof="profissão autor" titul="Titulação autor" grad="Graduação autor" instituicao="Instituição autor"></autor>

−</membros>

−<titulo><h1>Título da obra</h1></titulo>

−<datadefesa valor=></datadefesa>

−<curso valor=></curso>

−<instituicao valor=></instituicao>

−<siglainstituicao valor=></siglainstituicao>

−<departamento valor=></departamento>

−<sigladepartamento valor=></sigladepartamento>

−<local valor=></local>

−<**DESENVOLVIMENTO**>

•</projeto>

DESENVOLVIMENTO: Corpo do texto

•<desenvolvimento>

−<capitulo label="Nome do Capítulo">

∗<h2>Nome do Capítulo</h2>

∗ Texto do capítulo.

∗<**SECAO**>

−</capitulo>

•</desenvolvimento>

SECAO: corpo dos tópicos de cada capítulo

•<secao label="Título da Seção do Capítulo">

−<h3>Título da Seção do Capítulo</h3>

− texto da seção.

−<subsecao label="Título da Sub-Seção do Capítulo">

∗<h3>Título da Sub-Seção do Capítulo</h3>

∗ texto da sub-seção.

−</subsecao>

•</secao>