UNIVERSIDADE FEDERAL DE OURO PRETO

DEPARTAMENTO DE COMPUTAÇÃO

Isadora Fonseca Alves

UM ENSAIO SOBRE O ENSINO DE ENGENHARIA DE SOFTWARE NO BRASIL E NO MUNDO.
LIÇÕES PARA A UFOP

Ouro Preto, MG
2020

Isadora Fonseca Alves

UNIVERSIDADE FEDERAL DE OURO PRETO

DEPARTAMENTO DE COMPUTAÇÃO

Monografia II apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Dr. Tiago Garcia de Senna Carneiro

Ouro Preto, MG
2020

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

A474e Alves, Isadora Fonseca .
Um ensaio sobre o ensino de engenharia de software no Brasil e no mundo [manuscrito]: lições para UFOP. / Isadora Fonseca Alves. - 2020.
78 f.: il.: color..

Orientador: Prof. Dr. Tiago Carneiro.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Biológicas. Graduação em Ciência da Computação .

1. Ensino - Metodologia.. 2. Engenharia de software. 3. Industria de software. I. Carneiro, Tiago. II. Universidade Federal de Ouro Preto. III. Título.

CDU 004.41

Bibliotecário(a) Responsável: Celina Brasil Luiz - CRB6-1589

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
DEPARTAMENTO DE COMPUTACAO

FOLHA DE APROVAÇÃO

Isadora Fonseca Alves

Um ensaio sobre o ensino de Engenharia de Software no Brasil e no mundo - Lições para a UFOP

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Ciência da Computação

Aprovada em 15 de Dezembro de 2020

Membros da banca

Tiago Garcia de Senna Carneiro (orientador) - Doutor - Universidade Federal de Ouro Preto (UFOP)
Amanda Sávio Nascimento e Silva - Doutora - Universidade Federal de Ouro Preto (UFOP)
Igor Muzetti Pereira - Mestre - Universidade Federal de Ouro Preto (UFOP)

Tiago Garcia de Senna Carneiro, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 15/12/2020

Documento assinado eletronicamente por **Tiago Garcia de Senna Carneiro**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 15/12/2020, às 18:25, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0115882** e o código CRC **178126D1**.

Referência: Caso responda este documento, indicar expressamente o Processo nº 23109.009794/2020-16 SEI nº 0115882

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35400-000
Telefone: 3135591692 - www.ufop.br

Resumo

O ensino da disciplina Engenharia de Software nas graduações em Ciência da Computação, nas instituições de ensino, ocorre majoritariamente por meio de aulas expositivas e teóricas de acordo com a metodologia tradicional de ensino. No entanto, empresas e estudantes necessitam de uma abordagem mais abrangente que resulte em profissionais que dominem as habilidades requeridas pelo mercado de desenvolvimento de software, capazes de analisar problemas, além de propor e comunicar soluções. Diante deste cenário, este ensaio analisa criticamente como instituições de ensino do Brasil e do mundo vêm modificando sua forma de ensinar Engenharia de Software ao incorporar aspectos das metodologias ativas de ensino: PBL e PjBL. O objetivo desse trabalho é fundamentar as mudanças realizadas até o momento na disciplina Engenharia de Software ministrada no DECOM/UFOP, analisá-las e, se necessário, reconstruí-las. Para isso, foram criados novos meios para engajar os alunos dentro das salas de aula, responsabilizando-os pelo seu próprio aprendizado. Foi desenvolvido um novo conjunto de material didático a partir de diferentes Objetos de Aprendizado e estudos de caso especialmente projetados para esse propósito. Então, esses recursos foram utilizados em sala de aula e a opinião de docentes e discentes foram colhidas, junto aos indicadores acadêmicos (de aprovação, retenção, evasão e trancamento), para avaliar os impactos das mudanças aplicadas. Desta maneira, espera-se que a melhoria contínua da disciplina reflita na formação de melhores profissionais, superando a distância entre as habilidades dos graduandos e as demandas do mercado.

Palavras-chave: Metodologias de Ensino 1. Engenharia de Software 2. Industria de Software 3.

Abstract

The teaching of the Software Engineering academic discipline in Computer Science graduations, in educational institutions, occurs mainly through expository and theoretical classes according to the traditional methodology of teaching. However, companies and students need a more embracing approach that results in professionals who master the skills required by the software development market, able to analyze problems, and propose and communicate solutions. In this scenario, this essay critically analyzes how teaching institutions in Brazil and the world have been modifying their way of teaching Software Engineering by incorporating aspects of active teaching methodologies: PBL and PjBL. The purpose of this work is to base the changes made so far in the academic discipline of Software Engineering taught in DECOM / UFOP, to analyze them and, if necessary, to reconstruct them. To this end, new means were created to engage students in classrooms, making them responsible for their own learning. A new set of didactic materials were developed from different Learning Objects and case studies specially designed for this purpose. So, these resources were used in classrooms and teachers and students opinion's were collected along with the academic indicators (of approval, retention, escape and locking) to evaluate the impacts of the applied changes. In this way, it is expected that the continuous improvement of the academic discipline reflects on the formation of better professionals, overcoming the distance between the students' skills and the demands of the market.

Keywords:Teaching Methodologies 1. Software Engineering 2. Software Industry 3.

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

ABNT	Associação Brasileira de Normas Técnicas
DECOM	Departamento de Computação
UFOP	Universidade Federal de Ouro Preto

Sumário

1 Introdução

A inserção da disciplina Engenharia de Software (ES) no curso de Ciência da Computação (Bacharelado) pretende apresentar aos estudantes como o mercado lida com o processo de Desenvolvimento de Software. Nas universidades, ES é uma disciplina que estuda todos os aspectos da produção de um software, desde a especificação do produto até a sua manutenção depois de construído (Sommerville, 2007). As aulas apresentam as melhores práticas adotadas pelos profissionais envolvidos no processo de desenvolvimento e na gestão de projetos de software. Ela capacita os estudantes a adentrar na indústria de software e no mercado de trabalho. O ensino de ES na maioria das instituições ocorre por meio de aulas teóricas e expositivas, nas quais o professor é o principal ator dentro das salas de aula, ou seja, ocorre segundo o modelo mais tradicional modelo de ensino. No entanto, atualmente, os estudantes e as empresas apresentam uma necessidade de maior visão de mundo do mercado. Para os estudantes é de grande valia ter acesso a aquilo que o mercado realmente vivencia, aos reais desafios enfrentados pelas empresas e aos processos e padrões de qualidade adotados para o desenvolvimento de software. O mercado também necessita receber profissionais melhores capacitados para analisar e resolver problemas recorrentes na indústria.

A estrutura atual da disciplina ES busca ensinar desde a concepção até a liberação dos produtos de software, passando pelas etapas de concepção, projeto, construção, teste e implantação. Nota-se que é desafiador elucidar e formalizar as necessidades de um cliente para um determinado produto, uma vez que é comum o cliente não entender da área tecnológica de criação de software, como também não conseguir transmitir precisamente, por meio de diálogos, os requisitos dos produtos. Isto posto, a disciplina aborda como deve ser realizado o levantamento e a análise de requisitos dos produtos e como deve ser gerido um projeto para o desenvolvimento desses produtos. Ao ensinar como conceber, projetar, construir, testar e implantar/operar um produto de software, a disciplina busca salientar as melhores práticas desempenhadas pelos diversos tipos de atores que atuam nas equipes de desenvolvimento, suas atribuições, que artefatos produzem e consomem, que processos e tecnologias utilizam e que modelos de qualidade seguem. Trabalhar em equipe para construir sistemas complexos é pensar no projeto desde a interface gráfica, persistência de dados, passando pela lógica de negócios atendendo a indispensabilidade do que o cliente deseja, independentemente das estruturas de dados e algoritmos usados utilizados na construção. É uma mudança de perspectiva para os estudantes, que precisam ver o software de fora para dentro, como faz o cliente, ao invés de utilizar a perspectiva oposta que um programador está acostumado.

Tendo em vista que no cenário acadêmico, a Engenharia de Software não consegue abarcar em sua totalidade os projetos voltados para a indústria e para o mercado de trabalho, e que seu principal objetivo é justamente capacitar os estudantes para esses cenários, passa a ser uma

solução desafiadora estimulá-los para que conheçam a disciplina através de práticas que envolvam a simulação da diversidade de problemas reais. Por isso, além de uma abordagem teórica, forte atenção é dada ao ensino de práticas que os capacitem na utilização de Application Programming Interfaces (APIs), padrões de projetos de softwares, técnicas avançadas de programação, processos e tecnologias largamente utilizadas na indústria de software. Além disso, as instituições de ensino devem reforçar o conhecimento produzido na indústria e repassá-lo aos estudantes. Isto é, os métodos, processos e técnicas criados na indústria para sanar os problemas recorrentes devem ser utilizados como material de estudo dentro das universidades. Por isso, compreender as dificuldades enfrentadas nas empresas de software é essencial para a capacitação dos estudantes e para o crescimento da própria indústria.

A necessidade de cooperação entre a indústria de software e as instituições de ensino para a capacitação dos estudantes dá origem a alguns desafios no que se refere ao modo de ensinar a ES. Como captar os problemas das empresas de software? Como trazer as empresas para serem atores nas salas de aulas das universidades? Como abordar os problemas captados juntamente dos estudantes? Qual a melhor metodologia de ensino para engajar os estudantes no processo de aprendizado? Como avaliar o desempenho dos estudantes e a qualidade da aprendizagem em sala de aula? Como inseri-los na realidade do que o mercado realmente necessita? Quais práticas devem ser realizadas em sala de aula pelos estudantes? Qual o papel das empresas, estudantes e professores nas salas de aula? Como avaliar os sentimentos dos professores, empresas e estudantes sobre a condução das aulas?

Como um primeiro passo para responder essas questões, no período de quatro anos, de 2015 a 2019, esta pesquisa foi desenvolvida no escopo da disciplina Engenharia de Software I (BCC322) do Departamento de Computação da Universidade Federal de Ouro Preto.. Analisou-se as aulas juntamente com o corpo docente e discente. A cada semestre, em um total de 8, as aulas foram avaliadas pelos autores deste trabalho e pelos discentes, permitindo a remodelagem do programa da disciplina, buscando substituir a metodologia tradicional de ensino por uma metodologia ativa, na qual o próprio estudante é o principal responsável por seu aprendizado. Como consequência de diversas mudanças gradativas, os indicadores de evasão, retenção e rendimento dos estudantes vêm sendo impactados. Contudo, nos semestres iniciais deste estudo, as mudanças foram concebidas e aplicadas de forma intuitiva com pouca fundamentação nas teorias de ensino-aprendizagem.

Diante deste cenário, este trabalho objetiva fundamentar as mudanças realizadas até o momento, criticá-las e, quando necessário, reconstruí-las. Para isso, foi necessário selecionar, avaliar e aperfeiçoar o método de ensino utilizado em sala de aula. Foi preciso o desenvolvimento de novo material didático, composto por estudos de caso e objetos de aprendizado, que permitem que as metodologias de ensino baseadas em problemas (PBL) e baseadas em projetos (PjBL) fossem utilizadas como veículos para suprir as necessidades de capacitação dos estudantes e da indústria. Desta maneira, buscou-se melhorar o engajamento e aprendizado dos estudantes,

além de melhor capacitá-los para o mercado de trabalho, resultando na melhoria dos indicadores de aprendizado, na redução dos indicadores de reprovação e evasão e, finalmente, na evolução da qualidade do ensino de ES na UFOP. Espera-se constatar todas essas melhorias de forma qualitativa, mas também quantitativas, a partir de métricas coletadas ao longo deste projeto e nos anos seguintes à sua realização. Apesar dos diversos trabalhos correlatos a este, o processo de ensino de ES está em contínuo desenvolvimento para melhor atender estudantes, docentes e empresas.

1.1 Justificativa

O ensino de Engenharia de Software atual pretende apresentar aos estudantes como a indústria lida com projetos de desenvolvimento de software, desde sua concepção até o produto final, passando pelas etapas de concepção, projeto, construção, teste e implantação/operação. Apesar deste objetivo claro, observa-se que há uma grande distância entre a forma na qual os estudantes vivenciam o conteúdo da disciplina ES e a vivência demandada pela indústria de software. Se por uma lado, as universidades fornecem o rigor do conhecimento acadêmico exercitado em aulas expositivas e avaliações práticas e teóricas compostas por exercícios idealizados, a indústria requer vivência na resolução de problemas reais, conhecimento prático e habilidades no uso de tecnologias de ponta, além de experiência na condução de projetos de software. Diante desta dualidade, se faz necessário estreitar esta distância e contribuir diretamente no interesse comum a estes dois ambientes, universidade e indústria, que é a formação de profissionais melhores capacitados para o mercado de trabalho. Para isso, é preciso aperfeiçoar a forma de lecionar a disciplina ES, para a formação de profissionais críticos e ativos, capazes de lidar com as dificuldades enfrentadas pela indústria de software do século XXI.

1.2 Objetivos

Engajar estudantes no seu próprio processo de aprendizado, em quaisquer disciplinas, é um desafio recorrente para as instituições de ensino. Formar estudantes capacitados para o mercado de trabalho na área de desenvolvimento de software é também uma tarefa desafiadora. Para superar esses desafios, este trabalho tem como objetivo fundamentar, conceber, planejar, implantar e avaliar os impactos de mudanças no ensino de Engenharia de Software no âmbito do Departamento de Computação (DECOM) da Universidade Federal de Ouro Preto (UFOP). A partir da cooperação continuada com o corpo docente para remodelagem do programa da disciplina Engenharia de Software, este trabalho buscou melhorar o engajamento e aprendizado dos estudantes, bem como melhor capacitá-los para o mercado de trabalho, além de reduzir os índices de reprovação e evasão ao longo do tempo.

Como objetivos específicos deste trabalho enumera-se: (i) estudar e criticar o cenário atual de ensino da disciplina Engenharia de software nas universidades; (ii) customizar um novo

método para lecionar o conteúdo do programa da disciplina, afastando o método de ensino da forma tradicional e aproximando-o à uma forma ativa que coloca o estudante como ator ativo e principal responsável por seu próprio aprendizado; (iii) reforçar as necessidades do atual mercado de software e comportamento profissional no programa da disciplina para a formação de profissionais capacitados na área; (iv) desenvolver novo material didático para as aulas que ilustrem e exercitem os desafios mais frequentes enfrentados pela empresas de software em seu cotidiano; (v) aplicar e avaliar o uso desse material didático e método de ensino na disciplina BCC322 – Engenharia de Software I do DECOM/UFOP; (vi) oferecer aos alunos a vivência de muitos dos vários papéis encontrados na indústria de desenvolvimento de software; (vii) oferecer aos estudantes vivência no uso dos processos e das tecnologias utilizados pela indústria de software; (viii) avaliar de forma quantitativa e qualitativa os impactos e o novo cenário de aprendizado dessa disciplina, a fim de elucidar as lições aprendidas no estudo de caso e construir um guia contendo diretrizes de ações que promovem o aprendizado e assim, torná-lo o experimento replicável para outras disciplinas e universidades.

1.3 Organização do Trabalho

Os demais capítulos deste texto se organizam da seguinte maneira. O capítulo 2 apresenta a fundamentação teórica necessária ao entendimento deste trabalho, assim como os trabalhos encontrados na literatura que a ele são correlatos. O capítulo 3 apresenta o contexto organizacional anterior a esse estudo, isto é, o cenário passado do ensino de ES no curso de Ciência da Computação da Universidade Federal de Ouro Preto (UFOP), e em seguida, apresenta a metodologia adotada para a realização deste trabalho. O capítulo 4 apresenta os resultados alcançados ao longo deste estudo e, finalmente, o capítulo 5 discute em que medida os objetivos deste trabalho foram atingidos considerando-se os resultados alcançados e as lições aprendidas ao longo do seu desenvolvimento.

2 Revisão Bibliográfica

Este capítulo tem como objetivo discutir estudos realizados acerca dos desafios, melhores práticas e metodologias encontradas ao lecionar Engenharia de Software (ES) nas universidades brasileiras e estrangeiras, de modo a compreendê-las, identificar e fundamentar mudanças que possam ser úteis à melhoria do ensino de ES na UFOP.

Ao discutir o cenário de educação de ES nas universidades, buscando-se maneiras de melhorar o processo de ensino e aprendizagem, notou-se que que em cada instituição de ensino a disciplina é ministrada de forma distinta, assim, trataram de diferentes obstáculos no ensino e aprendizagem de ES.

De modo a contemplar as duas perspectivas, mundial e nacional, o capítulo foi dividido em duas sessões, retratando as duas realidades para o entendimento do estudo: Ensino de engenharia de software no mundo e Ensino de engenharia de software no Brasil.

2.1 Trabalhos Relacionados

Este capítulo irá apresentar os trabalhos correlatos a este trabalho em uma escala global e nacional.

2.1.1 Ensino de ES no mundo

As metodologias de ensino sofrem mudanças à medida que dificuldades são encontradas dentro dos ambientes de aprendizado. As metodologias tradicionais de ensino tem como personagem principal o professor, sendo ele o portador do conhecimento (realizando exposição oral das aulas) e os estudantes como coadjuvantes no cenário pedagógico. Consequência dessa prática de ensino é a redução da aprendizagem a níveis superficiais, não ressaltando a oportunidade do estudante de questionar e estar presente ativamente das aulas, causando comodismo e passividade dos estudantes (Lopes, 2000). Outro desafio presente no ensino de ES é a dificuldade de trazer problemas reais do mercado de trabalho para dentro das salas de aula. Nesse sentido, processos e práticas são necessários para que haja uma colaboração entre indústria e universidade.

2.1.1.1 Chile

No Chile, desde 2015, está em prática a Reforma Educacional nas escolas e universidades do país. A mudança no ensino superior está diretamente ligada à transformação da metodologia tradicional de ensino para a metodologia ativa, com o intuito de desenvolver as habilidades requeridas pela indústria tecnológica do séc. XXI. Constatou-se que a abordagem tradicional de ensino nem sempre garante uma resposta adequada e rápida aos desafios atuais encontrados

no universo educacional (Flores, Gomez, 2016). As aulas expositivas, ou seja, o professor transmitindo seu conhecimento aos estudantes não envolve descobertas por parte dos estudantes (Escobar,1988). Isto posto, as metodologias ativas de ensino-aprendizagem aparecem neste cenário pedagógico como solução e inovação de aprendizagem nas salas de aulas.

Metodologias ativas de ensino tem a função de colocar o estudante como um ator ativo durante as aulas e estimula processos construtivos de ação-reflexão-ação (Freire, 2006), incentivando os estudantes a exercerem o papel central das aulas como responsáveis por sua própria aprendizagem (Flores, Gomez, 2016). Uma aula expositiva dialogada é uma abordagem ativa da construção do conhecimento, tendo um estudante participativo e questionador do conteúdo apresentado pelo professor (Silva, 2016). Assim, os estudantes se tornam mais críticos e indagadores do conhecimento. Para Elizabeth Yu Me Yut (2012), a universidade deve ser um lugar para os estudantes adquirirem habilidades educacionais, profissionais, analíticas e de trabalho, sabendo assim usar o pensamento científico. Assim, o uso de metodologias ativas de ensino-aprendizagem correlaciona o teórico e o prático para gerar um cenário criativo e soluções para problemas reais.

No contexto do ensino de ES, como o mercado atual precisa de gente capacitada, com senso crítico, ativamente participativa na solução de novos desafios e na resolução de problemas emergentes, o Chile decidiu apostar em integração de estudantes, material didático e desenvolvimento de práticas em projetos trazidos do mundo real para sala de aula. Desta maneira, buscou-se engajar os estudantes apresentando-os a um contexto da realidade no qual as habilidades requeridas vão de encontro ao que as empresas do setor tecnológico necessitam, nos quais seus conhecimentos práticos e teóricos são exercitados e complementados. Considerou-se que as ementas da disciplina ES devem apresentar uma visão abrangente entre práticas e base teórica, deve existir um diálogo entre práticas dadas e o que é necessário e atual para o mercado. Deste modo, um número maior de estudantes irá sair do campo acadêmico aptos ao que é requerido como currículo mínimo exigido pelo mercado. Por essas razões, os programas de Engenharia da Computação da Universidade Católica do Norte (UCN) e da Universidade Católica do Maule (UCM), integraram diversas metodologias para o ensino de ES: Metodologia de aprendizagem baseada em problemas (PBL) e a metodologia ágil de desenvolvimento de software SCRUM.

O PBL é uma metodologia ativa de ensino-aprendizagem que utiliza situações como forma de aprendizado, isto é, estudo de casos especialmente projetados para esse propósito. Nessa metodologia, o professor fornece cenários que permitam que o estudante tenha acesso a problemas reais para a construção do conhecimento. Os estudantes fazem uma análise do problema inicial e verificam-se as possíveis soluções e objetivos de aprendizado. Assim, os estudantes têm acesso à realidade profissional, aumentam a capacidade de análise de problemas, obtêm uma melhor fixação da aprendizagem e exercem um compartilhamento de conhecimentos adquiridos durante todo o processo. Os estudantes também se tornam mais participativos, buscando novos conhecimentos, articulando teoria-prática e realizando reflexões críticas sobre os problemas, parte importante da sua formação profissional (Gemignani, 2012).

No estudo de caso da UCN (Flores, Gomez, 2017), visando o desenvolvimento de competências para trabalho em equipe, diminuir a taxa de abandono do curso e motivá-los a contribuir com os projetos, agrupou-se os estudantes em equipes e estabeleceu-se papéis para cada membro conforme suas habilidades, facilidades e potencialidades. A equipe era dividida em líderes de projetos, administradores de banco de dados, gerente de projetos, programadores e analistas de testes, todos esses são papéis que os estudantes podem futuramente desempenhar no mercado. O professor selecionava como o projeto seria realizado. Limitava-se o número de estudantes com diferentes habilidades em cada grupo, analisava-se o problema vindo do cliente estudando a viabilidade técnica dos projetos, estimava-se tempo de realização do projeto, definia-se formatos dos relatórios garantindo o feedback do projeto e definia-se e priorizava-se o backlog do produto. Os projetos foram realizados segundo três atividades chaves: levantamento e análise de requisitos pelo gerente de projetos, desenvolvimento pelos programadores e administradores de bancos de dados, e testes pelos analistas de testes. O professor atuou como treinador ao instruir os estudantes no decorrer do trabalho e mantendo-os comprometidos com o rigor do processo de desenvolvimento de softwares e resultados. As empresas colaboram com as especificações dos produtos e ao orientar tecnicamente os estudantes em como solucionar os problemas reais. A metodologia ágil SCRUM (Schwaber, Sutherlan, 2019) foi usada para a criação e gestão do backlog dos produtos. Para cada tarefa do backlog foram determinados responsáveis e tempo de duração estimado. Ao final dos projetos, o professor mensurou cada produto e atribuiu notas para os estudantes conforme a qualidade, o tempo de entrega, e número de tarefas realizadas. Após utilizar essa forma de ensino por dois anos seguidos, verificou-se que os estudantes obtiveram um desempenho melhor do que dos anos anteriores, e que o professor conseguiu feedback ativo em cada entregável realizado, e que as equipes mais comprometidas para o sucesso do projeto e mais participativas nas aulas foram aquelas que renderam melhores resultados. Estes fatos evidenciaram que as atividades realizadas foram motivadoras para os estudantes e que o uso de dinâmicas como essas fazem com que eles se sintam mais envolvidos e engajados até o fim do curso.

Na Universidade Católica do Maule (UCM), as aulas incluíram o desenvolvimento de um projeto de software juntamente de um cliente real. Os objetivos requeridos diante do trabalho foram: Desenvolver soluções de informática conforme os padrões de software e qualidade que respondessem às necessidades das empresas; além de diminuir a taxa de abandono do curso através de projetos de software reais. O trabalho consistia na criação de grupos diante de papéis específicos conforme a metodologia MOPROSOF (Oktaba, 2006). Eles deveriam especificar as necessidades do cliente e transformar em um projeto de software. O professor tinha o papel de instrutor e gerente do projeto, que obtinha um banco de empresas e suas necessidades. A empresa era contactada no início do projeto para a criação do documento backlog, no meio para a apresentação do desenvolvimento do projeto e para sanar dúvidas encontradas, e no fim do projeto para a realização dos testes e apresentação final do trabalho. Ao finalizar cada fase do projeto (análise do projeto, construção e integração de testes), um relatório era gerado para

analisar o progresso de cada equipe e assim, dar as notas de cada grupo juntamente da aprovação dos clientes. Ao concluir o trabalho, pode-se notar uma melhoria no desempenho dos estudantes e aumento da participação em sala de aula. Ressaltando que os estudantes puderam lidar com o cliente desde o começo do projeto e assim, muitos deles tiveram a oportunidade de estágios dentro das mesmas e realização de trabalhos finais de graduação sobre o trabalho dado.

Os trabalhos realizados nestas universidades mostraram um desenvolvimento mais engajado dos projetos, dando aos estudantes uma aproximação real de como o mercado de desenvolvimento de software atua juntamente das necessidades concretas dos clientes. O papel dos professores mudou de expositores de aulas tradicionais e passaram a ter um acompanhamento mais efetivo dos trabalhos e aprendizado dos estudantes. A metodologia PBL auxiliou na validação das atividades e aprendizado dos estudantes, tendo um resultado mais rápido na evolução dos estudantes.

2.1.1.2 Mundo

Ao analisar a literatura sobre a educação de Engenharia de Software nas universidades em uma escala global, nota-se que há cada vez mais pesquisas relatando experiências e novos conceitos para melhorias do ensino. Muitas destas pesquisas buscam promover o crescimento das colaborações entre indústria e universidades para que desafios comuns sejam vencidos. No entanto, são poucas as colaborações efetivas diante do volume de atividades dessas organizações (Garousi; Petersen; Özkan, 2016). Por isso, muitos pesquisadores e profissionais vêm publicando estratégias de sucesso e novos métodos de ensino nesta área.

A Association for Computing Machinery (ACM) possui o projeto de um Grupo de Interesse Especial em Engenharia de Software (SIGSOFT). O projeto consiste em um fórum para interligar profissionais da área que fazem parte da indústria, governo, pesquisadores e educadores das instituições de ensino. Desse modo a comunidade cria melhorias, discute novas ideias, promove desenvolvimento profissional de engenheiros de software e debate sobre novas direções da ES no mundo. Conforme discussões deste grupo, a maneira mais poderosa de ajudar na construção de softwares de excelência seria aumentar a habilidade e competência dos envolvidos na área (SIGSOFT, 2019). Ou seja, o foco para se obter ganhos tanto na indústria quanto no crescimento da ES é o ensino. Desse modo, ter uma receita bem-sucedida de como construir softwares a partir da integração entre quem necessita, estuda e pesquisa e quem constrói softwares, é uma medida de satisfação e de crescimento da área. No ensino de Engenharia de Software, a qualidade dos profissionais está diretamente relacionada à qualidade da educação, embora também existam outros fatores que contribuem para isto (Beckman et al., 1997). A qualidade do ensino está diretamente ligada ao aperfeiçoamento do estado da arte do desenvolvimento de software e ao auxílio da solução de alguns problemas tradicionais e crises relacionadas às práticas da indústria de software (Gibbs, 1994).

Garousi e Özkan (2016) mostram a importância do fomento a colaboração entre indústria

e instituições de ensino (Industry Academic Collaborations – IAC) para o aprendizado da ES nas universidades. Ele fornece uma síntese dos desafios e melhores práticas a serem usadas por ambas as partes. Os autores realizam uma revisão sistemática da literatura no qual selecionam os mais relevantes estudos sobre IAC que têm como foco o aprendizado de ES. Ao final da revisão, obteve-se respostas para três perguntas: "Modelos de colaboração: Que tipos de modelos de IAC foram propostos?"; "Desafios / impedimentos: Quais desafios ou impedimentos para as IACs foram levantados pelas pesquisas?"; "Padrões (melhores práticas): Quais padrões foram propostos para resolver os desafios das IACs?"

Para a primeira pergunta, que tange os modelos de IAC, destacam-se cinco artigos. Gorschek (2016) cria um modelo de transferência de tecnologia; Sandberg, Pareto e Arts (2011) apresentaram um modelo para gerenciamento dos projetos de colaboração; Connor, Buchan, Petrova e Bridging (2009) trouxeram a ideia de sistemas reflexivos para a comunicação efetiva e o desenvolvimento de uma visão de mundo compartilhada; Rombach, Achatz (2007) apresentaram um modelo em espiral para o desenvolvimento industrial baseado em inovação desenvolvido e usado pelo Instituto Fraunhofer para Engenharia de Software Experimental (IESE) na Alemanha; e Runeson e Min (2014) apresentaram um modelo arquitetural de colaboração.

No modelo de transferência apresentado por Gorschek (2016), analisou-se a transferência de tecnologia entre pesquisadores da instituição de ensino (Blekinge Institute of Technology) e profissionais de duas empresas do setor de desenvolvimento de software (Danaher Motion Särö AB- DHR e ABB). De acordo com o estudo, pesquisadores e profissionais se beneficiam dessa colaboração. Os pesquisadores têm a oportunidade de observar os desafios atuais da indústria e realizar suas pesquisas não apenas de assuntos relevantes para a academia, mas também relevantes para a indústria. Os benefícios para a indústria serão a possibilidade de usufruir de novas tecnologias que são desenvolvidas com base nos problemas reais identificados nas empresas colaboradoras. A metodologia de transferência é constituída em sete etapas, que são: Etapa para basear a pesquisa nas reais necessidades do setor envolvendo todas as partes afetadas e importantes do estudo na empresa e na academia; Etapa de Formulação do Problema, ou seja, o que deve ser feito e melhorado. Cada um oferece o que detém de conhecimento para uma colaboração de sucesso. Nessa etapa, o vocabulário necessário para o entendimento mútuo é explicado e aprendido por ambas as partes. Os requisitos são apresentados e estudados; Formulação de uma solução candidata realista. Os pesquisadores nessa etapa, oferecem o que tem de conhecimento técnico, processos e ferramentas já estudadas para auxiliar na formulação da solução e melhoria dos requisitos apresentados. Há também a validação ao testar a solução apresentada. Testes de usabilidade, escalabilidade e se aquela solução atende às necessidades da empresa; Etapa de Validação de Laboratório, em que antes da solução encontrada ser aplicada na indústria ela é testada nos laboratórios das universidades, apresentando um feedback valioso para encontrar falhas e corrigi-las em tempo de execução; Etapa de Validação Estática, no qual ocorre a apresentação para todos os profissionais da empresa da solução encontrada e testada nos laboratórios. Assim, consegue-se receber feedback de todos os envolvidos; Etapa de Validação

Dinâmica é o processo em que o mercado utiliza a solução estudada em pequena escala em forma de protótipos; Como etapa final, obteve-se a Etapa de Solução de Liberação é a oficialização da utilização da solução estudada dentro da empresa. Como resultado, obteve-se a importância dada no comprometimento para o sucesso das duas partes, sem esse não seria possível o sucesso da parceria. Esse modelo é sugerido como inspiração para outros modelos que podem ser criados sob a demanda de cada empresa e cada universidade.

Para as últimas duas perguntas, sobre desafios e padrões das IACs, associou-se os desafios encontrados e as suas soluções. Para cada desafio, um padrão foi criado a partir da análise das respostas encontradas pelos diversos estudos ao sanarem problemas recorrentes nas indústrias, alinhado ao conceito padrão de projeto de software (Gamma et al., 1998). Usualmente, para a indústria há a necessidade de que a universidade compreenda totalmente seu problema, para que assim possa encontrar uma solução viável que satisfaça suas necessidades, soluções parciais apenas elevarão os custos dos projetos. Até o momento, notou-se que o assunto IAC em SE é desafiador e possui muitos obstáculos a serem vencidos. Há a necessidade de mais estudos sobre o tema na área para criação de mais processos, padrões, técnicas e estudos de melhoramento, destacando a necessidade de aprendizado na indústria e na universidade. Damm (2006) apresentou um exemplo de IAC que consegue transferir um resultado satisfatório para a indústria.

Um outro estudo foi realizado sobre a educação nos cursos de Engenharia das universidades numa escala global para determinar as habilidades técnicas e pessoais exigidas dos engenheiros pela indústria. Esse estudo indicou preocupações com relação ao ensino para a preparação dos estudantes a fim suprir as necessidades das empresas. Capacidade de se comunicar e trabalhar em equipe, compreender os problemas e ter agilidade para solucioná-los, são importantes habilidades requeridas no mercado de trabalho. Porém, o cenário atual de ensino está formando profissionais com um bom conhecimento teórico, mas com um baixo nível de conhecimento prático. Assim, existe uma necessidade de mudança na forma de ensino e aprendizagem das instituições, para que os estudantes possam ter um contato satisfatório com o conhecimento teórico e prático relacionando-os às técnicas industriais e assim se profissionalizaram de acordo com o que o mercado demanda. A mudança relaciona-se às metodologias de ensino usadas nas salas de aula, na maioria das vezes utiliza-se as metodologias tradicionais de ensino. Algumas universidades optaram por modificar a maneira de se ensinar, seguindo as técnicas usadas nas metodologias ativas de aprendizado baseadas em problemas ou projetos (PBL e PjBL).

A Aprendizagem Baseada em Problemas (PBL), tem sido implementada em alguns programas de engenharia. Woods (1997), implementou no programa de Engenharia Química as práticas do PBL na McMaster University nos de workshops realizados. Relatou-se que essas oficinas auxiliam os estudantes a desenvolver a habilidade de resolução de problemas, habilidades interpessoais e habilidades de equipe. Uma pesquisa qualitativa mostrou a satisfação dos estudantes em relação ao modelo de ensino utilizado. Notou-se ao se aplicar PBL em outros cursos de Engenharia que ainda há obstáculos a serem enfrentados. Para Feletti (1993), há dificuldades ao

utilizar PBL na engenharia diante da divergência de gênero de profissões entre medicina, área em que o PBL foi criado, e a engenharia. Outras questões foram levantadas em relação ao uso da metodologia PBL: engenheiros necessitam, na maioria das vezes, de um pré-aprendizado teórico para solucionar problemas, ou seja, eles precisam aplicar seu conhecimento a partir de problemas em contrapartida ao que o PBL utiliza como técnica de aquisição do conhecimento a partir de problemas; Os cenários de problemas enfrentados nas empresas são diferentes dos aprendidos nas universidades, ou seja, há uma série de diferentes problemas a serem resolvidos; A ordem de ensino é parcialmente definida pelos estudantes no PBL, assim tópicos importantes no aprendizado podem ser negligenciados. Portanto, nota-se que a utilização da PBL pode ser uma ajuda parcial na aprendizagem dos estudantes. No entanto, outros métodos de aprendizagem ativas são utilizados para sanar as questões apresentadas anteriormente, como é o caso da Aprendizagem Baseada em Projetos (PjBL).

A Aprendizagem Baseada em Projetos (PjBL) tem como objetivo aplicar o conhecimento teórico em tarefas práticas que se baseiam em projetos. Nessa metodologia há a combinação de ensino tradicional, que oferece a base teórica para aplicação para a realização dos projetos, como também o ensino ativo que conduzirá o estudante a solução dos projetos. Os projetos podem ser realizados individualmente ou em pequenos grupos de estudantes, e o professor atua como consultor do projeto. Na Universidade de Aalborg o ensino é baseado em projetos e fortemente orientado para resolução de problemas da indústria. Argumenta-se que as técnicas do PBL e do PjBL enfatizam diferentes aspectos da aprendizagem, mas que as duas têm as mesmas ideias ao enfatizar a aprendizagem em vez de ensinar (Mills et al., 2004). No trabalho realizado em sala de aula os estudantes analisam o problema, identificam uma solução a partir do conhecimento estudado em sala de aula, gera um produto final e um relatório final para discussão entre estudantes e professor. Cada membro do projeto é atribuído para uma função específica de acordo com suas habilidades dentro do projeto. Analisou-se que ao final dos projetos os estudantes estavam melhor treinados em suas habilidades na equipe, melhor capacitados para realização de um projeto total e também havia uma melhora na comunicação dos mesmos. A educação em Engenharia de Software é o principal fator para a promoção da indústria de software de vários países (Garg, 2008). Nota-se que o modo mais eficaz de se obter melhores resultados para a comunidade é estar cientes dos desafios, melhores práticas e o que não fazer ao se realizar uma colaboração entre indústria e academia.

2.1.1.3 Utilização de projetos FLOSS no ensino de ES

Um estudo realizado por M. Brito et al. (2018), analisou a utilização de FLOSS (Free/Libre/Open Source Software) na educação de engenharia de software com o intuito de reduzir a lacuna entre o curso de engenharia de software (ES) e a indústria. O estudo recuperou, analisou e selecionou 33 artigos publicados no período de 2013 a 2017 para comparar com um estudo realizado anteriormente para confirmar e descobrir novas tendências na área de ensino de ES. A partir disso evidenciou que a utilização de FLOSS na ES expõem os estudantes a grandes projetos, melhores

práticas e ferramentas profissionais para controles de versões, compilação e gestão de projetos. Da mesma forma, os projetos de FLOSS também permitem aos estudantes experimentar como é o processo de colaboração nas dinâmicas sociais em equipe. As antigas e novas tendências de pesquisas confirmadas por M. Brito et al. (2018, p.251), estudo foram: (1) o uso de FLOSS para aprender de forma abrangente sobre ES; (2) escolha do currículo focado principalmente nos cursos regulares; (3) avaliação baseada principalmente em relatórios e artefatos de software; (4) maior uso de relatórios de experiência e redução de propostas de soluções; (5) crescente interesse em usar FLOSS na manutenção e evolução de software;(6) aumento do uso de abordagens onde os instrutores não têm controle, e os alunos interagem com a comunidade FLOSS em tarefas reais; (7) menos uso de projetos FLOSS predefinidos e mais escolha de projetos livre pelos alunos; (8) equilíbrio entre os objetivos de aprender FLOSS e aprender conceitos de SE; e (9) o uso de suítes de testes automatizados como um tipo de avaliação relevante.

Os resultados desse projeto foram utilizados para fornecer uma visão abrangente da utilização dos projetos de FLOSS nos últimos 20 anos no estudo de ES. A partir desse estudo, verificou-se que apesar do aumento do número de relatos da utilização de FLOSS na ES, as evidências mostraram que a pesquisa na área ainda não está tão evoluída.

Para Silva, Fernanda Gomes, et al.(2019), não é fácil abordar os softwares desenvolvidos nas indústrias dentro de sala de aula. Em seu estudo, notou-se que a utilização de projetos FLOSS pode ser uma solução viável para diminuir essa dificuldade ao introduzir as práticas de mercado de desenvolvimento de software no processo de ensino aprendizagem de ES nas universidades.

Como complemento, os autores relatam como foi a experiência do ensino de diagrama de classes utilizando UML, na disciplina de ES em um curso de Ciência da computação. Nesse estudo, o instrutor recebeu auxílio em atividades relacionadas ao planejamento de software a partir de um FLOSS pré selecionado. Ao final da atividade, o instrutor aplicou uma atividade avaliativa e um questionário sobre o desenvolvimento daquela atividade. Os autores do estudo também realizaram uma entrevista com o instrutor para que ele relatasse como foi a sua experiência diante do experimento. Como resultado, os estudantes relataram que sua experiência com o projeto FLOSS aumentou a sua capacidade para lidar com projetos reais e código de terceiros, desenvolveram suas habilidades pessoais como por exemplo, proatividade e comunicação,e, consequentemente, tiveram contato com a visão do mercado de trabalho da área. Para o tutor, a experiência mostrou que os grupos de alunos se mostraram mais entusiasmados e dinâmicos ao interagir durante a atividade aplicada. Dessa forma, pode-se notar que a utilização de FLOSS no ensino de ES pode ser considerada uma boa alternativa para aprender sobre os conceitos da disciplina de forma dinâmica e prática.

2.1.2 Ensino de ES no Brasil

No ano de 2017 o Brasil ocupou a 9a posição no ranking mundial de crescimento na área de software e serviços de acordo com a Associação Brasileira de Empresas de Software

(ABES, 2018).Ao todo, são 17.000 empresas atuando no setor. Baseado nesses dados, fica evidente a necessidade de apoiar a pesquisa na área de Engenharia de Software no Brasil. Formar profissionais capacitados e qualificados para o mercado de trabalho se faz necessário diante da importância dada à indústria de software e a exigência por qualidade dos produtos por ela produzidos. Da mesma maneira que no cenário mundial, as necessidades requeridas na área de ES não se diferem no cenário nacional. Os resultados alcançados muitas vezes permanecem isolados e localizados, tornando- se necessário diagnosticar e reunir quais são os principais problemas, soluções e desafios de educação em ES, além das peculiaridades do cenário nacional (Santos et al., 2008).

O ensino de ES, no panorama nacional, aborda os conceitos em aulas teóricas e expositivas, conforme a abordagem tradicional de ensino. Desse modo, os graduandos não têm um acesso satisfatório aos problemas reais da indústria. Raramente, os ensinamentos das aulas são voltados para os problemas do mercado (Santos et al., 2008). Diante do grande volume de documentos e processos para a execução de um bom projeto de software, estudantes se desinteressam pela disciplina de ES, pois esta é tida como demasiadamente teórica e burocrática. Os estudantes preferem escrever programas e vê-los funcionando à documentar formalmente os seus projetos (Lucena, 2006). Por outro lado, os profissionais da indústria estão insatisfeitos diante do pouco preparo dos recém graduados quando estes ingressam no mercado de trabalho, forçando a indústria a complementar a educação dos estudantes por meio de treinamentos (Conn, 2002). Assim, este cenário resulta em uma indústria deficitária de profissionais qualificados e estudantes insatisfeitos no ensino de ES.

Diante de todas estas questões, o paradigma experimental da ES é apontado como uma possível estratégia para vencer os desafios do ensino de ES nas universidades (Santos et al., 2008). Este paradigma envolve a coleção e análise de dados e evidências, e os utiliza para caracterizar, avaliar e revelar relacionamentos entre tecnologias, práticas e resultados do processo de ensino e aprendizagem de ES. Além disso, é preciso que a indústria reconheça que também é de sua responsabilidade preparar os estudantes para os reais desafios de seu interesse (Meyer, 2001).

Os trabalhos relacionados ao processo de ensino e aprendizagem de ES, geralmente não se aprofundam em problemas e desafios específicos da realidade brasileira, e seus resultados não convergem para o estabelecimento de um corpo de conhecimento e estabelecimento de uma comunidade de educadores no Brasil (Santos et al. 2008). Por isso, o projeto “EduESBrasil 1.0” (Santos et al., 2009) elaborou um survey que analisa problemas, dificuldades, soluções e singularidades do cenário nacional em ES focada na experimentação, auxiliando indústria e academia simultaneamente. O portal EduES oferece o diálogo entre educadores e pesquisadores para desenvolver um grupo de conhecimento em educação em ES no Brasil. Realizou-se uma revisão sistemática e uma pesquisa de opinião de educadores da área, analisando as estratégias de ensino utilizando objetos de aprendizado (OA).

Incluir Novas Tecnologias da Informação e Comunicação (NTICs) nos ambientes de

ensino com o intuito de auxiliar o aprimoramento profissional de estudantes cresce com a facilidade de acesso à informação a partir da internet. As instituições de ensino estão se adequando a esse novo cenário ao utilizar novas tecnologias como recursos ao acesso e a informação para complementar o processo de ensino e aprendizagem nas salas de aula. Esses novos recursos são chamados de Objetos de Aprendizagem. Os OAs são definidos como recursos digitais que são utilizados e reutilizados como material de suporte ao ensino e aprendizagem e, ao mesmo tempo, estimulam a criatividade e imaginação dos estudantes, tal como slides, vídeos, jogos digitais entre outros. Santos et al (2009) abordaram questões de interesse na ES, são elas: Em que situação se encontra a educação em ES no Brasil comparada ao cenário internacional; Quais são os problemas e desafios da educação em ES no Brasil; Quais são as questões de pesquisa prioritárias e estratégicas que impactam a indústria de software brasileira; Quais são as estratégias para minimizar as dificuldades de ensino-aprendizagem; Quais as dificuldades e discrepâncias em ensino considerando as diferentes regiões do país; Quais as adaptações no ensino para atender as demandas da indústria de software; e quais características peculiares são encontradas nos cursos de graduação em ES recém-criados, comparadas às de outros cursos de graduação na área de Computação. O projeto "EduESBrasil 2.0" provê novas características para que o portal se torne um repositório que concentra os esforços de diversos colaboradores, permitindo-os publicar, classificar, testar, avaliar e reutilizar conteúdos educacionais em ES (Borges et al., 2011). Este portal dará a essa área de ensino maior visibilidade e poder de interação entre seus usuários, sejam eles estudantes, pesquisadores ou profissionais do mercado de trabalho. Os autores do presente trabalho acreditam que dar acesso a esse tipo de informação auxilia no crescimento e melhoramento da ES no Brasil.

Outros estudos também apoiam a utilização de OA para o ensino de ES. Eles identificam os tipos de OAs utilizados e em quais áreas eles são mais aplicados na computação. A partir de estudos em ES, notou-se que os jogos estão à frente do ensino, ou seja, é uma nova maneira (OA) de se aprender em sala de aula. Isso é validado por Souza et al (2013) que mostram que o ensino e aprendizagem podem ser potencialmente facilitados por OA baseados em jogos. Neste contexto, a área de algoritmos é a que mais se beneficia do uso de OAs (Cardoso et al., 2015). A engenharia de software é apontada como a segunda maior beneficiária. Por isso, os autores do presente trabalho acreditam que as maneiras e os utensílios usados para a aprendizagem devem evoluir conjuntamente à área de ES no mercado. Acredita-se que o desenvolvimento de mais OAs trará um resultado benéfico aos estudantes e, como consequência, à indústria. Esta abordagem tem grande potencial para estreitar a distância entre teoria e prática.

Diante desse cenário, acredita-se que desenvolver OAs atrativos e que vão de encontro às necessidades da indústria de software potencialmente traz benefícios para a academia e para a indústria. Ir além dos métodos de ensino tradicionais, desafiando os estudantes a lidarem com problemas reais enfrentados pela indústria, por meio de OAs especialmente projetados para esse propósito, favorece os estudantes e as empresas de softwares. Logo, quanto maior for a demanda de pesquisa, colaborações entre indústria e ensino e plataformas de discussões, melhor será o

desenvolvimento de ES no cenário nacional a nível mundial.

2.2 Fundamentação Teórica

Este capítulo discute as metodologias de ensino existentes e utilizadas para ensino de Engenharia de Software, dando ênfase às metodologias ativas. Ele também descreve métodos de gestão de projetos de software, processos de desenvolvimentos de softwares, dando ênfase ao processo BOPE (Pereira, 2014). Finalmente, outros conceitos relevantes ao entendimento, execução e avaliação deste trabalho são apresentados: Testes de software, dando ênfase aos níveis de testes e testes caixa branca e caixa preta.

2.2.1 Metodologias de Ensino, uma análise geral da história

Seguir o fluxo da sociedade globalizada significa fazer mudanças ao longo dos anos para podermos nos enquadrar no contexto moderno. Na educação atualmente, pode existir a necessidade de acompanhar o fluxo da sociedade globalizada para nos enquadrar as mudanças que ocorrem a todo momento. A educação deve então se atualizar em seus métodos e paradigmas. Para Mitre (2008), no contexto social atual, com as crescentes transformações nas relações dinâmicas, a discussão da necessidade de mudanças nas instituições de ensino superior se torna presente para que haja a reconstrução de seu papel social. Dessa forma, tornou-se preciso desenvolver novas metodologias que promovam o diálogo entre o homem e o trabalho e entre o homem e a natureza, ou seja, a discussão de como é o mundo em que vivemos e como esses diálogos são essenciais para o cenário educacional. Assim, o homem se adapta a seu tempo, compreendendo nas contradições e dificuldades a urgência de se adentrar nesses diálogos para benefícios futuros (Feitosa,Couto,1999). Na universidade, a educação está diretamente ligada à formação profissional dos estudantes e na captação de novos conhecimentos. Nesse contexto, tem-se uma variedade de conceitos e práticas que interferem diretamente no modelo de transmissão de conhecimentos, entre eles a transmissão vertical do conhecimento, o voluntarismo, a ação sócio comunitária institucional, extensão universitária e o acadêmico institucional. As metodologias de ensino são etapas didáticas que possuem métodos e técnicas de ensinos próprios para alcançar o objetivo de aprendizagem eficaz para os estudantes.

A transmissão vertical do conhecimento constitui uma prática pedagógica autoritária vinda das instituições de ensino para a sociedade. Essa metodologia de ensino, se consolidou na Grécia antiga (Serrano, 2008). Os estudantes eram parte minoritária de acesso ao conhecimento. Via-se as universidades como instituições portadoras de toda sabedoria. O termo vertical, está ligado à hierarquia de quem detinha o conhecimento, ou seja, o que as instituições diziam ser verdade era entendido como verdade absoluta. Os estudantes são componentes passivos do processo, em que apenas obtêm o que é dado. Logo, não são seres atuantes, críticos e pensantes. Para Freire (2006): “todos estes termos envolvem ações que, transformando o homem em quase

"coisa" o negam como um ser de transformação do mundo". Por outro lado, por meio da chegada dos jesuítas na idade média, nasce a metodologia voluntarista. A educação estava voltada para mudanças sociais por causa de seu caráter ideológico vindos dos ideais políticos de professores e estudantes da época (Serrano, 2008). Séculos depois, no Brasil surgiram as Universidades Livres, nasceram os movimentos estudantis, a junção entre cultura local e cultura universitária. São os primeiros passos para a visualização da capacidade de diálogos das universidades juntamente ao mundo (Serrano, 2008). Dois olhares de mundo diferentes se interligam para achar a veracidade dos pontos de vista (Gadotti, 2003). Para Freire (2006), o diálogo é a base da pedagogia na luta para transformações sociais.

Devido a reforma universitária no Brasil, na década de 30, criou-se o modelo de ação sócio comunitária institucional. Trata-se de um modelo manipulador, onde apenas a universidade detém o conhecimento perante a sociedade. Durante a consolidação desse modelo, ocorreu a ditadura no Brasil. Então, contra a ditadura e esse modelo conservador, surgem os movimentos estudantis que traziam a visão de mundo da comunidade para as universidades. Os movimentos estudantis eram ameaças ao novo governo. Tinha-se dois extremos, a visão conservadora de modelo de ensino ditada pelo governo e os ideais libertários dos movimentos estudantis.

Já na década de 60, ainda no regime ditatorial, as universidades são um modelo mais independente. Nasce então, o movimento de Extensão Cultural, como emancipação dos ideais autoritários vigentes naquela época. Ao longo dos anos seguintes, vigora a ideia de conhecimento como uma via de mão dupla, onde você pode ensinar como também aprender. Surge a ideia de extensão como sendo a produção de ideias e conhecimentos como um processo educacional. Na extensão universitária, tem-se fixada a ideia de não dissociação de ensino e pesquisa, transformando o cenário de Universidade e Sociedade. Nesse processo, a sociedade nos dá a prática do conhecimento adquirido nas instituições de ensino. O processo do conhecimento terá, não somente a teoria, como também a prática do mundo. O Brasil se beneficia desse modelo pois detém e produz o conhecimento como uma troca de saberes entre a academia e a sociedade. Desde então, a sociedade vem sendo estudada e colocada em prática nas instituições.

A partir disso, na busca pelo conhecimento as instituições de ensino têm a função acadêmica (repassar o conhecimento), uma função social (construir cidadania e entender a sociedade que vivemos) é uma função articuladora (participação conjunta de universidade e sociedade) (Ser- rano, 2008). Notou-se a grande variedade de modelos de transmitir o conhecimento nos últimos tempos. As mudanças são fruto do que o meio necessita naquele momento. Nesse sentido, novas metodologias de ensino ainda estão sendo criadas, estudadas e testadas nas universidades brasileiras. Novos conceitos de aprendizagem que se diferem do modelo tradicional e vertical e que avançam juntos das mudanças da sociedade ampliam o acesso ao saber. Como por exemplo a inserção da tecnologia no cenário de estudos (Nerice, 1978).

Contudo, a forma didática de ensino atual ainda reflete os métodos antigos de ensino, como é o caso dos métodos usados pelos jesuítas no modelo de ação sócio comunitária institucional

(Anastasiou, 2001). O modelo de aulas expositivas onde ocorrem a memorização do conteúdo pelos estudantes e as avaliações realizadas pelos professores são os reflexos vistos hoje na educação de algumas instituições brasileiras, apesar de haver constante debate e mudanças. As instituições de Ensino Superior no Brasil são relativamente novas comparadas a criação das mesmas em um cenário mundial. A primeira instituição de nível superior foi construída em 1912, a Universidade Federal do Paraná (UFPR), em Curitiba. O surgimento de novas universidades teve seu ápice nos anos 70 apenas. A finalidade da educação superior foi declarada na Lei de Diretrizes e Bases da Educação Nacional (LDBEN 9.394/96) que visava estimular cultura, desenvolvimento científico e reflexão, como também, formar estudantes em diferentes áreas, incentivar pesquisa científica e aperfeiçoar o profissional (Teixeira, 2015). Nota-se a evolução das instituições a partir do questionamento para aprimoramento das tendências pedagógicas e didáticas utilizadas.

As tendências pedagógicas são interpretações dos papéis da educação na sociedade que são divididas em dois grupos: Pedagogia Liberal (a crítica) e Pedagogia Progressista (crítica) (Luckesi,1994). Na pedagogia liberal, tem uma identidade conservadora (o que se difere do seu nome) usada no Brasil desde os princípios da educação. As instituições de ensino visam preparar o indivíduo para papéis sociais diante das suas aptidões, se adaptando às normas da sociedade de classes (Libâneo, 1990). Um exemplo é a pedagogia tradicional, em que o professor emite e tem controle do saber, como dominador do cenário de aprendizagem e têm o estudante como o ator que apenas aprende o que lhe é passado (transmissão vertical do saber). As aulas nessa abordagem são expositivas, não iterativas e dependem da memorização dos estudantes. Para avaliar os estudantes, realizam-se provas escritas, em que não se avalia o processo de aprendizado, mas serve apenas como instrumento de autoridade. As aulas nas universidades brasileiras ainda são voltadas para esse tipo de ensino, visto que os docentes foram submetidos a essa prática em sua formação acadêmica (Freire, 2007; Gil, 2008). Para Veiga (2006), na perspectiva atual, o professor não pode assumir esse modelo tradicional e sim, assumir um papel de mentor e facilitador, dando acesso para o estudante ao saber.

A Pedagogia progressista tem as instituições de ensino como agente transformador da sociedade, criticando e questionando a realidade social vigente contra o caráter autoritário dos modelos anteriores. Essa nova tendência se torna uma pedagogia problematizadora que cria diálogos entre o professor e o estudante para, em conjunto, compartilharem seus conhecimentos. Ou seja, ocorre uma transmissão horizontal de saber. “Ensinar não é transferir conhecimento, mas criar a possibilidade para a sua própria produção ou construção” (Freire,1996). Para Teixeira (2015), não é o bastante ser apenas especializado em certa área, o professor necessita ser articulador, questionador, motivador de produção de conhecimento e formador de opiniões para os estudantes.

2.2.1.1 Metodologia Ativa

Nas metodologias de ensino, os docentes apresentam seus conteúdos de aulas, que são uma junção de técnicas e métodos que auxiliam na aprendizagem dos estudantes. Observou-se a partir das análise das mudanças realizadas em sala de aula que os conteúdos dados em sala de aula, muitas vezes são extensos para o tempo de aula pré-estipulado nas ementas. A partir disso, os docentes têm a preferência em ministrar aulas expositivas apenas, ou seja, utilizando as metodologias tradicionais de ensino-aprendizagem. Por esse motivo, essa metodologia ainda é muito presente no cenário educacional nacional nas universidades e escolas. Alguns autores acreditam na transformação do tradicional como forma de melhoramento da aprendizagem e estimulação do pensamento dos estudantes. Para Gemignani (2012), o método tradicional de ensino ao longo dos anos tem se mostrado ineficaz e ineficiente em função das exigências da realidade social. Assim, surgiram as metodologia ativas de ensino.

A solução de melhoramento encontrada foi a criação do diálogo entre estudante e professor para a estimulação dos processos construtivos de ação-reflexão-ação (Freire, 2006). Nota-se que o diálogo traz consigo uma técnica, a problematização ao conduzir o aluno a observar e discutir a sua realidade (Gemignani, 2012). Assim, problematizar o teórico, pode auxiliar o estudante na forma de ampliar seus olhares para a realidade dos problemas. As metodologias ativas permitem ter um estudante mais ativo diante do processo de ensino, discutindo sobre problemas reais permitindo a formação de indivíduos críticos (Silva, 2017). Nesse cenário, surgem duas metodologias que abordam problemas reais para auxílio do ensino: a aprendizagem baseada em problemas (Problem Based Learning – PBL) e a aprendizagem baseada em projetos (Problem based Projects – PjBL). Enquanto o PBL é utilizado para a aquisição, fazendo com o estudante o adquira ao buscar uma solução para o problema apresentado pelo professor, o PjBL é direcionado para a aplicação do conhecimento que lhe é apresentado a priori pelo professor. Ao final do processo do processo de aprendizado, o PBL geralmente produz um resultado abstrato, enquanto o PjBL resulta em algum tipo de produto concreto.

Nesses cenários de aprendizagem, o ensino se inicia a partir de problemas e situações encontradas no dia-a-dia e se transforma em prática ao necessitar de soluções para esses problemas. Nesse sentido, o estudante consegue ser mais ativo dentro do processo de aprender, é o principal responsável por seu próprio aprendizado e, ainda, vivencia realidades do mundo mais perto de si. As metodologias ativas estão alicerçadas em um princípio teórico significativo que é a autonomia dos estudantes (Mitre, 2008). As duas metodologias são baseadas na auto-orientação disciplinar e colaboração conjunta para o aprendizado. Para Mitre (2008), essas novas técnicas de ensino não garantem por si só a ruptura das técnicas tradicionais e velhos paradigmas, sendo necessário que haja a transformação das concepções próprias do processo para ressignificá-las em uma educação emancipadora.

A metodologia ativa PBL começou a ser usada na década de 1970 na Universidade McMaster, Canadá, no curso de medicina e hoje é amplamente usada nesse cenário. A metodologia

tem como foco principal os problemas reais que são apresentados aos estudantes, nos quais todo o processo de aprendizagem é direcionado para a solução dos mesmos. Ao invés de exigir que os alunos estudem o conteúdo para depois praticar o que foi aprendido, o PBL incorpora os alunos no processo de aprendizagem em problemas da vida real (Wood, 2003). A partir da apresentação do problema, os estudantes o analisam formulando quais serão os objetivos de aprendizagem, estudos e relatórios finais. Esse processo é realizado em tempos curtos e não produzem nenhum artefato, apenas são apresentadas as soluções encontradas como material de debate entre os estudantes. Atualmente, universidades em todo o mundo adotam PBL no processo de ensino e aprendizagem em diversos cursos, não somente na medicina (Noordin et. al., 2011). O processo da dinâmica utilizando a metodologia PBL pode ser analisada na Figura 2.1 a seguir.

A metodologia ativa PjBL está diretamente ligada ao aprender fazendo. O processo inicia com uma ou mais tarefas a serem realizadas que resultam na produção de um produto final que pode ser um design, um modelo, um dispositivo, um software, entre outros. O foco principal dessa metodologia é a execução de um projeto que encerra um problema real de um cliente. O PjBL se aproxima mais do contexto real das empresas por abranger todos artefatos de um projeto, ou seja, planejamento de tempo, custo e recursos. É necessário um tempo maior para a realização dos projetos. A aprendizagem está direcionada à aplicação do conhecimento, ou seja,

Um conhecimento anteriormente adquirido será utilizado para sanar os problemas apresentados. Os projetos são realizados por grupos de estudantes, para que possam colaborar uns com os outros na aquisição do conhecimento. Para Lowenthal (2006), a aprendizagem baseada em projeto é a junção de métodos do PBL, aprendizagem cooperativa, aprendizagem ativa e a teoria de gestão de projetos.

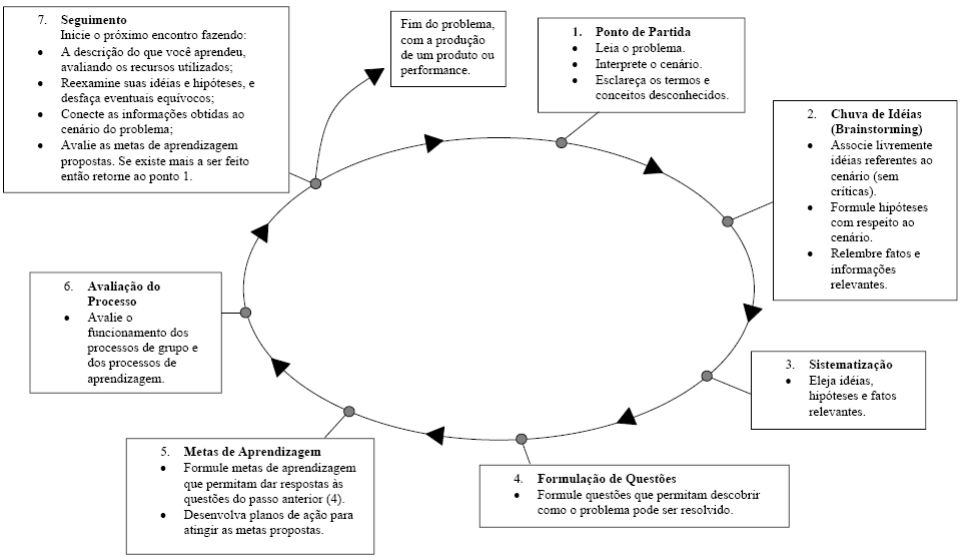


Figura 2.1 – Passos da discussão de uma sessão tutorial

As duas metodologias possuem propósitos em comum para conectar os estudantes a partir de tarefas do mundo real, a fim de melhorar o aprendizado através do estudo de projetos e

problemas representativos e recorrentes no mundo real. Para Anafiza (2017), conforme definido na literatura, as duas metodologias compartilham várias características, uma vez que ambas são estratégias de ensino com o objetivo de envolver os alunos em tarefas reais para aprimoramento do aprendizado. Os alunos recebem projetos abertos ou problemas com mais de uma abordagem ou resposta, destinados a simular situações profissionais. As principais diferenças são o foco de cada metodologia, no PBL um problema é analisado e resulta em uma pesquisa, investigação e uma solução abstrata. Enquanto que, a abordagem principal da PjBL é a produção de um produto final focado no processo de produção. Muitas outras metodologias foram criadas e remodeladas para diferentes contextos de ensino e hoje são testadas nas salas de aulas, mas fogem ao escopo deste trabalho.

2.2.2 Gestão de Processos de Desenvolvimento de Software

As constantes mudanças na indústria de desenvolvimento de software requerem novos processos e metodologias para atender os novos desafios encontrados. O mercado cresce e junto dele há um crescimento de novas demandas de como gerir o desenvolvimento de um novo projeto de software, garantindo qualidade. Os processos de desenvolvimento de software são necessários para sanar problemas que aparecem ao longo dos projetos de uma forma organizada e padronizada, garantindo a aplicação daquilo que foi estabelecido como modelo de qualidade (Silva et al., 2009).

Sabe-se que não existe um processo ideal de desenvolvimento de software (Sommerville, 2011). Para cada tipo de problema a ser resolvido por um software, as empresas adotam um determinado processo. Pode-se definir um processo de software como sendo um conjunto de atividades e resultados que ajudam na produção de um software (Soares, 2004). A literatura classifica esses processos em dois grupos: Processos tradicionais ou dirigido a planos; ou Processos ágeis.

Os processos tradicionais são frequentemente usados para situações onde os requisitos do sistema não sofrem tantas alterações (estáveis) do começo ao fim do projeto. Para os tipos de software que são sistemas críticos, como por exemplo um sistema de controle de segurança, em que uma análise completa do sistema é essencial, a abordagem tradicional e dirigida a planos será a melhor opção (Sommerville, 2018). Dessa forma, podemos notar que esse modelo é usado para sistemas críticos, onde o processo de desenvolvimento tem que ser bem estruturado e mais rígido. O plano de atividades é pré-estabelecido desde o começo do projeto e cada atividade é desenvolvida a partir de outra já finalizada.

A respeito do cenário de mudanças nas empresas, Sommerville (2018, p.38) destaca:

"Essas empresas operam em um ambiente de mudanças rápidas, e por isso, muitas vezes, é praticamente impossível obter um conjunto completo de requisitos de software estável. Os requisitos iniciais inevitavelmente serão alterados, pois os clientes acham impossível prever como um sistema afetará as práticas de trabalho, como irá interagir com outros sistemas e quais

operações do usuário devem ser automatizadas. Pode ser que os requisitos se tornem claros apenas após a entrega do sistema e à medida que os usuários ganhem experiência. Mesmo assim, devido a fatores externos, os requisitos são suscetíveis a mudanças rápidas e imprevisíveis."

As metodologias ágeis surgem como uma resposta a esse cenário e abraçam as mudanças para rapidamente atender as necessidades de cada usuário, baseando-se no planejamento gradativo dos projetos. O cliente que encomenda um produto é parte importante do projeto e, por isso, um representante seu é inserido na equipe de execução. Os requisitos não são estáticos, podem ser alterados e criados no desenrolar do projeto. Mudanças ao invés de serem tratadas como exceções e evitadas a todo custo, podem ser vistas como possibilidades cotidianas. As equipes de execução preparam-se para responder rapidamente às mudanças ao invés de seguir planos rígidos. Relatos da indústria evidenciam que essas metodologias são eficazes diante da necessidade de diminuir riscos e custos, em especial, quando os requisitos de um produto não são bem determinados pelos clientes ou são instáveis. Pequenas equipes de desenvolvimento podem interagir entre si durante o projeto, para responder a mudanças e sanar os desafios, resultando em um planejamento gradativo e colaborativo dos projetos. Essas características são mais adequadas ao desenvolvimento de produtos cujos requisitos são flexíveis.

2.2.2.1 Gestão de processos de software BOPE

Os processos de desenvolvimento de software na maioria das vezes são destinados aos ambientes empresariais nos quais os recursos humanos são integralmente dedicados e comprometidos com o sucesso dos projetos. Desenvolveu-se na Universidade Federal de Ouro Preto um processo que atendesse às peculiaridades desse ambiente no qual os estudantes têm como prioridade suas atividades acadêmicas e, por isso, são apenas parcialmente dedicados à realização de projetos de pesquisa, desenvolvimento ou extensão. Os estudantes de graduação não estão totalmente comprometidos com o sucesso dos projetos, uma vez que a falha não traz a eles prejuízos diretos. Neste cenário, a rotatividade (entrada e saída) nas equipes dos laboratórios de pesquisa, desenvolvimento e inovação tende a ser maior que aquela observada nas equipes encontradas nas empresas. O processo ágil BOPE (Pereira ,2014) mescla práticas de vários outros processos de desenvolvimento de software e processos de gestão de projetos para resolver osdesafios desse cenário.

De acordo com Pereira (2014), o BOPE adota a estrutura geral do processo ágil SCRUM, apostando no planejamento incremental, ciclos de desenvolvimentos curtos direcionados por feedbacks rápidos do cliente; Os ciclos de desenvolvimento são dirigidos por testes automatizados (Test Driven Development – TDD) e capitaneados por um treinador que mantém a equipe seguindo fielmente ao processo, conforme preconiza o Extreme Programming (XP); Os critérios de aceitação do cliente são capturados, documentados e materializados por meio de cenários especificados de acordo com o Behavior Driven Development (BDD); A gestão de projeto (custo, prazo, risco, comunicação, qualidade) acontece em conformidade com as diretrizes

fornecidas pelo Project Management Institute (PMI). Como qualquer processo ágil, o cliente é voz ativa dentro da equipe do projeto. A gestão de resultado e impactos acontece por meio da do monitoramento e análise de métricas quantitativas sobre as equipes, processo e produtos. A discussão dessas métricas com toda a equipe auxilia no engajamento e melhoria de desempenho dos estudantes. Ao contrário das equipes horizontais adotadas pelos processos ágeis, o BOPE estabelece uma clara hierarquia entre os papéis desempenhados pelos membros das equipes de desenvolvimento, pois os estudantes ainda encontram-se em diferentes estágios de sua formação e, por isso, possuem habilidades diferentes. As atividades de cada um desses papéis são claramente definidas, assim como os artefatos por eles produzidos são simples, padronizados e mensuráveis, garantindo a qualidade do produto. Assim, o BOPE fornece aos laboratórios os insumos que eles necessitam para realizar projetos de longo prazo e de qualidade.

Como no SCRUM, o BOPE é dividido em sprints que resultam em um novo incremento testável de software, agregando valor ao negócio do cliente. Uma sprint é um ciclo de desenvolvimento de duração fixa (mês ou meses) e pré-determinado pelo gestor do projeto em acordo com o cliente. Por sua vez, cada sprint é subdividida em ciclos menores de desenvolvimento (dias ou semanas) chamados iterações. Nas iterações realiza-se às atividades de concepção, projeto, construção, teste e integração. A Figura 2.2, ilustra o fluxograma de atividades do BOPE.

Detalhadamente, na primeira sprint ocorre a definição do escopo do projeto, que descreve o contexto e o objetivo do projeto, determina a missão e as limitações do produto, e esclarece os benefícios pretendidos pelo cliente. Além disso, identifica-se o backlog do produto, detalhando todos os entregáveis do projeto. O backlog fica aberto durante todo o projeto para atender as necessidades de mudanças do cliente, podendo ser modificado no início de cada sprint. A partir do backlog inicial, define-se o custo e tempo de duração do projeto.

As iterações de qualquer sprint seguem a seguinte estrutura. Primeiramente, ocorre a identificação e criação das histórias de usuários que capturam os critérios de aceitação do cliente, indicando de forma geral como o produto deve se comportar e como deve ser testado. Depois, os cenários de testes são especificados de forma objetiva e detalhada para materializar os critérios de aceitação do cliente na forma de testes automatizados. Então, as funcionalidades do produto são implementadas e corrigidas até que sejam homologadas nos testes automatizados. Finalmente, as métricas de produto, equipe e processo são coletadas e discutidas em equipe.

Ao longo do projeto, cada estudante tem oportunidade de desempenhar diversos papéis e irá adquirir as habilidades exigidas pelo papel que escolher desempenhar de acordo com sua vocação, são eles: Analista de Negócios que é responsável pelo relacionamento com o cliente e por modelar o negócio do cliente definindo o escopo, backlog, cronograma e custo do projeto; Gerente de Projeto responsável pela execução e administração do projeto no que tange a prazos, custos, recursos humanos e qualidade; Líder de Equipe que é o líder técnico de cada equipe dentro do processo; Analista de Sistemas que faz a análise dos requisitos do software (estórias de usuários); Engenheiro de Teste que planeja, implementa e executa os casos de testes; e o

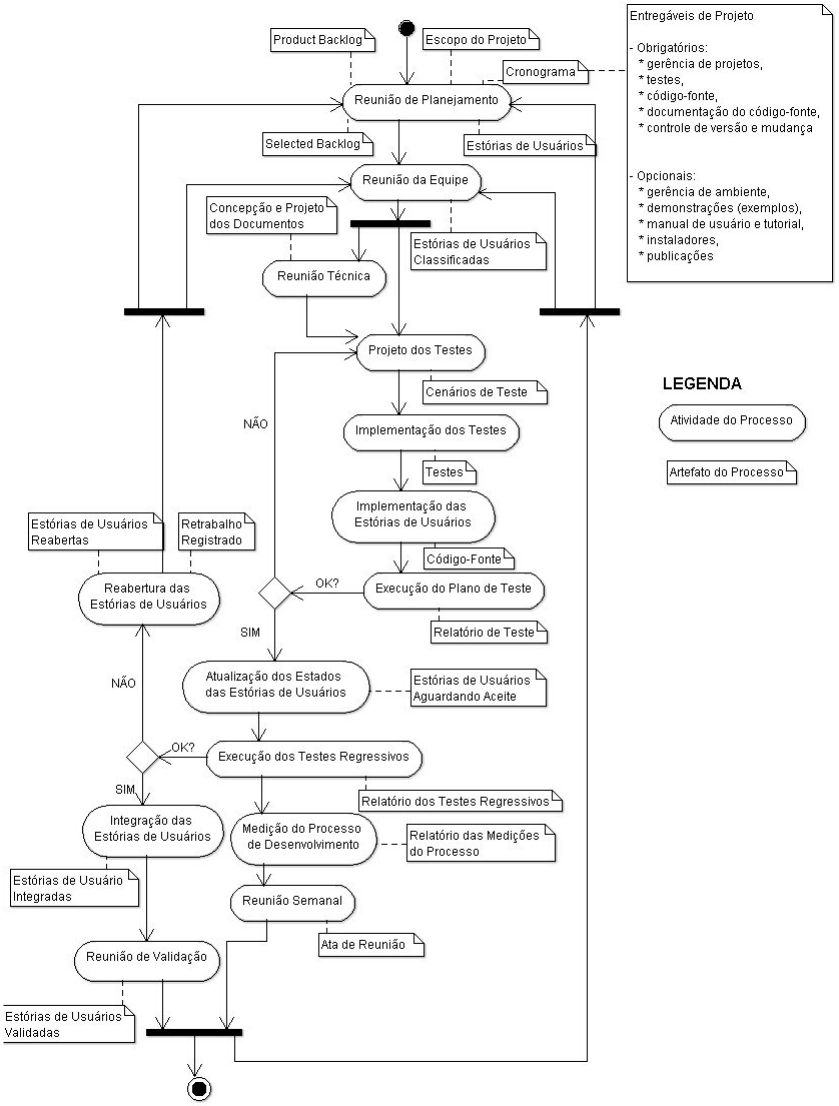


Figura 2.2 – Processo de desenvolvimento BOPE. Obtido de (Pereira, 2014)

Engenheiro de Software que implementa as funcionalidades do produto e testes unitários. Assim, o BOPE busca assegurar qualidade, dividindo claramente as tarefas e responsabilizando entre os membros da equipe de desenvolvimento.

2.2.3 Testes de Software

No processo de desenvolvimento de um software é necessário analisar se o mesmo realiza todas as funcionalidades a que é previamente proposto a fazer. O teste de software tem como objetivos apresentar um software que atenda aos requisitos e descobrir situações no qual o software não se comporta da maneira correta (Sommerville, 2011). Diante disso, temos os testes de validação, no qual analisa se um software executa corretamente a partir dos casos de testes e os testes de defeitos, que são utilizados para expor defeitos do software criado. Para Dijkstra et

al. (1972): “Os testes podem mostrar apenas a presença de erros, e não sua ausência.” Os testes podem ser divididos em dois grupos, testes de caixa preta e testes de caixa branca que serão discutidos a seguir.

2.2.3.1 Testes Caixa Branca e Testes Caixa Preta

Os testes de caixa branca, são assim denominados por ter conhecimento da estrutura interna (código) dos softwares, também podem ser chamados de testes estruturais. O código fonte do programa é utilizado para identificar testes de defeitos. O objetivo desses testes é realizar a cobertura do programa, em que a lógica do código é executada corretamente pelo menos uma vez.

Já os testes caixa preta, são assim chamados por não ter conhecimento prévio do código fonte do software ou seus componentes para a sua criação. Também podem ser denominados como testes funcionais. Os testes estão diretamente ligados na especificação do sistema. Seu objetivo principal é demonstrar se o sistema atende ou não os requisitos capturados. É criado um conjunto de entradas e um conjunto de saídas esperadas. Os testes podem ser para um método, função, programa, componente, conjunto de programas e componentes e funcionalidades.

Os testes são usualmente divididos em 4 camadas, são elas: Teste de Unidade que realiza os testes nos módulos do sistemas (menores unidades do software) para identificar erros de lógica e de implementação; Teste de Integração que tem o objetivo de identificar erros na interação dos módulos de um sistema, é uma técnica usada para integrar os módulos estruturais do software; Testes de Sistema que é realizado após a integração do sistema com o objetivo de captar erros de funções e de desempenho que não estão de acordo com a especificação; e o Teste de Validação utilizado para verificar a qualidade do conjunto de casos de testes.

3 Desenvolvimento

Esta seção apresenta os materiais, métodos e técnicas utilizados no período de 2015 a 2019 no ensino de Engenharia de Software na Universidade Federal de Ouro Preto (UFOP) e tiveram como prioridade alcançar os objetivos propostos por esse estudo. Nesta seção de texto são discutidos: O contexto organizacional anterior a este estudo ou seja, as principais mudanças que produziram o cenário atual das aulas da disciplina de Engenharia de Software do curso de Ciência da Computação da Universidade Federal de Ouro Preto (UFOP) – código BCC322; Os desafios observados ao longo do acompanhamento da disciplina e o planejamento das mudanças implantadas no seu modelo de ensino; Quais são os meios utilizados para melhorar o engajamento dos estudantes e diminuir índices de evasão e reprovação dos mesmos; Finalmente, o planejamento de novos materiais didáticos para auxiliar estudantes e professores nos próximos semestres.

3.1 Contexto organizacional anterior a este estudo

No início deste estudo, no ano de 2015, assim como em diversas outras universidades, as aulas eram ministradas usando as técnicas do modelo de ensino tradicional, o professor detém todo o conhecimento e transmite aos estudantes, com raros diálogos e muito monólogo. A transmissão de conhecimento era realizada por aulas expositivas e todo conteúdo teórico era apresentado antes de realizar as práticas e avaliações da disciplina. Então, o professor aplicava exercícios e provas sobre os conteúdos dados com o intuito de fixar a matéria apresentada. Neste cenário, notou-se que os estudantes, no momento em que eram apresentados aos exercícios teóricos e práticos, não obtinham sucesso na sua resolução dos problemas, carecendo do auxílio do professor. Os estudantes possuíam muitas dúvidas sobre o conhecimento teórico já apresentado. Consequentemente, o professor se via obrigado a repetir o conteúdo. O professor tinha a clara sensação de estar sobrecarregado com retrabalho, ao apresentar repetidamente o conteúdo, e frustrado por não conseguir que os alunos aprendessem e ganhassem novas habilidades.

A grande necessidade do professor, que se prolonga desde 2015 até os dias de hoje, é a notável necessidade de criação de um guia que informe ao próprio professor “como” cada aula deve ser ministrada, “como” e “em que tempo” os conteúdo da ementa e programa serão exercitados nos estudos de caso e práticas. Desse modo, além do professor ter conhecimento daquilo que está disposto no programa da disciplina, isto é, o conteúdo a ser lecionado em cada aula e o tempo reservado para o aprendizado de cada tópico da ementa da disciplina, o professor teria um conjunto de diretrizes que o informaria como abordar cada assunto, como conduzir o diálogo com os estudantes, quais dificuldades são notadas pelos estudantes de forma recorrente, quais resultados são esperados, quais objetos de aprendizado utilizar e como utilizá-los. Idealmente, este guia garantiria qualidade à disciplina no que diz respeito à transmissão do

conteúdo básico, independentemente da pessoa que desempenha o papel de professor. Neste cenário, o professor consumiria pouco tempo na produção de material didático ou decidindo "como" ministrar um conteúdo, ele então poderia esforçar-se em transmitir aquilo que o diferencia de outros profissionais, trazendo para a sala aula toda sua experiência profissional junto à indústria e à academia. Para que essas mudanças sejam possíveis, seria necessário reformular o programa e o material didático da disciplina, fundamentando, replanejando e documentando os casos de estudos e objetos de aprendizado, focando a forma na qual devem ser utilizados por professores e estudantes. Assim, em quase toda totalidade, a disciplina foi reformulada, desde a forma de ensinar até os materiais utilizados.

3.2 Ciclos PDCA semestrais

Ao longo de 4 anos, a cada semestre letivo, um ciclo PDCA foi realizado para avaliar o atual estado da disciplina, seu impacto sobre o rendimento e engajamento dos estudantes e, finalmente, para o planejamento e implementação de mudanças. De acordo com Deming (2000), o ciclo PDCA (Plan-Do-Check-Act), criado por Shewhart na década de 20, é um método de desenvolvimento iterativo com o intuito de realizar a melhoria contínua dos processos produtivos de uma empresa. O ciclo PDCA contém quatro processos que são: Plan (Planejar), Do(Executar), Check(Verificar), Act(Agir). De acordo com Muzzeti (2014), o conhecimento adquirido em cada ciclo será utilizado para aproximar o objetivo pretendido e assim, resultará em uma melhora no processo de ensino. Segundo Muzzeti (2014, p.15), os passos do ciclo PDCA são:

"Plan (Planejar): Estabelecer as metas desejadas e identificar a estratégia e os métodos que serão utilizados para atingi-las.

Do (Executar): Seguir o plano, executar os experimentos e gerar os relatórios para análise posterior.

Check (Verificar): Analisar e comparar os resultados obtidos com os resultados esperados. Buscar por variações no seguimento do plano, identificar erros ou falhas do percurso inicialmente traçado.

Act (Agir): Tomar ações corretivas para combater as diferenças significativas entre os resultados alcançados e esperados. Estudar as diferenças a fim de entender suas causas. Agir de acordo com o avaliado e planejado."

Neste contexto, todos os processos e dinâmicas utilizados durante as aulas foram desenvolvidos e, semestralmente, avaliados e melhorados ao longo deste trabalho. Para a avaliação do aprendizado por parte dos estudantes, os trabalhos práticos e exames escritos foram divididos e realizados de diversas maneiras, constituindo-se como experimentos, com vistas a aferir melhorias qualitativas e quantitativas de aprendizado. Os alunos foram convidados a realizar revisões bibliográficas e listas de exercícios a fim de se prepararem para estas avaliações. Análises

dos dados de desempenho acadêmico das turmas de estudantes foram utilizadas para avaliar os impactos das mudanças realizadas semestralmente em cada versão da disciplina, entre eles: Índice de reprovação, Índice de Evasão, e Desempenho Médio. Entrevistas e enquetes foram aplicadas com estudantes e professores para verificar aspectos qualitativos, de cunho emocional e psicológico, do ambiente educacional.

É parte relevante da metodologia utilizada neste trabalho ponderar três dimensões distintas do ensino de Engenharia de Software: O conteúdo científico que obrigatoriamente precisa ser ensinado e por isso faz parte da ementa das disciplinas BCC322; As técnica e tecnologias que compõem o estado-da-arte na indústria de desenvolvimento de software; e Finalmente, as estratégias e métricas utilizadas para avaliação do aprendizado por parte dos estudantes.

Além disso, uma vez que a ementa da disciplina é obrigatória e definida pelos colegiados de curso, respeitando-se as diretivas de conteúdo definidas pelas associações de classe, que no caso da Ciência da Computação, possui ementas modelos propostas pela organização internacional ACM - Association Computer Machinery e pela organização nacional SBC - Sociedade Brasileira de Computação, é portanto o programa da disciplina de Engenharia de Software que, a cada semestre, foi alterado ao longo de 4 anos, de forma a dedicar mais esforço a um determinado assunto em um dado semestre ou que, por diversas vezes, teve a ordem na qual os assuntos foram abordados alterada em diferentes semestres. Enfim, diversas estratégias de ensino foram experimentadas ao longo de anos, a fim de melhorar as estatísticas de desempenho e o engajamento dos estudantes. Para se ajustar a cada versão melhorada do programa de disciplina, tanto os conteúdos das aulas, quanto os materiais didáticos, quanto as atividades avaliativas sofreram modificações especialmente desenvolvidas para esse propósito.

3.3 Mudanças na metodologia de ensino

O mercado atual necessita de profissionais ativos nos processos de soluções de problemas. Assim, a participação dos estudantes em discussões é essencial para a formação de profissionais mais críticos, que saibam se comunicar e opinar diante das dificuldades apresentadas pelas empresas de desenvolvimento de software. Uma vez que a metodologia tradicional de ensino ainda é predominante nas universidades, observou-se que no primeiro semestre de 2015 os estudantes não desenvolviam as habilidades que o mercado necessita. Por essa razão, gradativamente a metodologia de ensino tradicional foi combinada ou cedeu lugar para metodologias ativas de ensino, PBL ou PjBL, estabelecendo o estudante como componente ativo e principal responsável por seu aprendizado.

Na sala de aula, as práticas da metodologia ativa foram aplicadas sobre os mesmos problemas que anteriormente eram abordados pela metodologia tradicional. O professor traz, em forma de discussão, problemas realistas (da indústria) que são representativos do conteúdo que exige a ementa da disciplina, para que os estudantes durante as aulas possam analisar, discutir e

relatar possíveis soluções para essas questões. Assim, notou-se que os estudantes se tornam mais participativos das aulas, desenvolvem senso crítico nas discussões sobre os problemas relatados, participam ativamente ao relatar seu ponto de vista diante do tema e aprendem a se comunicar e a agir em equipe para achar soluções viáveis.

Ao longo da disciplina, notou-se a necessidade de mesclar as duas metodologias (ativa e tradicional) para que os estudantes possam adquirir o conhecimento teórico em forma de palestras expositivas, mas que também utilize diferentes "objetos de aprendizagem" para fixação do conhecimento durante e após as aulas. Contudo, ao invés do professor ministrar aulas expositivas onde ocorrem a inserção de pequenas práticas, optou-se por aulas majoritariamente práticas, nas quais problemas são solucionados conjuntamente, com inserções pontuais de conteúdo expositivo após a busca de soluções prévias conforme propõe a metodologia PBL. Assim, o estudante passou a ser o ator principal das aulas, se transformando de ator passivo, ao apenas receber o conhecimento, para ator ativo que aplica e desenvolve seu conhecimento diante de discussões propostas pelo professor.

Neste cenário, primeiramente o professor tem o papel passivo de apenas apresentar situações em que problemas precisam de solução e o aluno se torna ativo ao aprender diante de pesquisas sobre como sanar tais problemas, apresentá-los e discuti-los com os outros estudantes (o estudante guia seu conhecimento). No segundo momento, o professor tem o papel de único detentor e transmissor do conhecimento e o estudante passivamente absorve o ensinamento que lhe é oferecido ao assistir o professor solucionar o problema em estudo. Ao longos dos últimos anos, as experiências nas aulas de BCC322 mostram que, os estudantes após a busca prévia por soluções, tornaram-se mai participativos, mais apto a entender as soluções expostas pelo professor e motivaram-se ao diálogo, reconhecendo o professor como detentor do saber e da técnica, nutrindo por ele um certo tipo de admiração que facilita o aprendizado.

Para fomentar este ambiente, os autores deste trabalho desenvolveram roteiros de aulas (para o professor) e novos "objetos sem aprendizado"para fixação de conteúdo com o intuito de auxiliar o ensino e o aprendizado de conhecimentos práticos e teóricos seguindo as diretrizes das metodologias ativas de ensino PBL e PjBL. O roteiro das aulas é apresentado na seção "4. Resultados".

3.4 Mudanças no material didático

No início deste esforço, o material didático consistia em slides para apresentação do conteúdo da ementa, listas de exercícios para treinamento e aprimoramento do conteúdo e um trabalho prático individual. Atualmente, a disciplina conta dois trabalhos práticos muito extensos, sendo um individual e um em grupo. O trabalho em grupo visa abranger o trabalho colaborativo onde membros das equipes vivenciam diferentes papéis existentes na indústria de software. Durante os últimos semestres, verificou-se que para aumentar o engajamento dos estudantes seria

preciso aprimorar o enunciado desses trabalhos, esclarecendo a maneira na qual os resultados são avaliados e, por vezes, os reformulando completamente, como é o caso das listas de exercícios.

Por isso, algumas das listas utilizadas foram remodeladas e novos enunciados foram planejados e seus conteúdos digitalizados. As listas que contém códigos de programação foram remodeladas para que o aluno aprenda tentando achar erros e melhorias nas mesmas. É importante que o aluno aprenda a lidar com código legado a partir da resolução destas listas de exercícios. Os slides utilizados nas porções expositivas das aulas foram avaliados e reconstruídos para harmonizá-los com os estudos de caso e facilitar o aprendizado.

De acordo com o que sugere o PBL, os autores buscaram oferecer aos estudantes uma base de casos de estudos de problemas recorrentes do mercado atual de desenvolvimento de software. No entanto, ao contrário do que vinha acontecendo, após análise de formulário (ver Anexo A) respondido pelos alunos, os autores sugeriram que os trabalhos práticos fossem realizados separadamente, um após o término do outro, evitando-se a todo custo que fossem intercalados como vinha acontecendo. Desse modo, os estudantes ficam menos confusos e sobrecarregados, resultando em compreensão e engajamento diante dos trabalhos práticos. As respostas dos estudantes, também revelaram que havia a necessidade de reformulação dos enunciados dos dois trabalhos práticos para que eles entendessem aquilo que é solicitado, e assim realizassem um trabalho coerente com o que é esperado pelo professor. As listas e trabalhos práticos passaram a considerar uma combinação de técnicas das metodologias ativas PBL e PjBL, de acordo com suas naturezas, as listas seguiram o PBL e os trabalhos ao PjBL.

Seguindo a metodologia PjBL, os trabalhos práticos resultam em um produtos finais a partir de projetos de software. Foi no desenvolvimento destes produtos que as necessidades do mercado de software passaram a ser exercitadas com os estudantes que, para além do conteúdo acadêmico exigido pela ementa da disciplina, passaram a ter contato com processos e ferramentas para teste, automação e controle de versão de software que o mercado necessita. No trabalho em grupo, os estudantes além vivenciarem as diferentes funções existentes no ecossistema de desenvolvimento de software e de treinarem suas habilidades de comunicação e de colaboração, também agiram avaliando trabalhos de outras equipes no contexto de uma relação cliente-fornecedor, onde cada grupo especificou e encomendou um produto a outro grupo, ao mesmo tempo que forneceram a solução encomendada por um terceiro grupo. Desta maneira, eles passaram a ter maior clareza das falhas existentes nas especificações que idealizaram e puderam corrigir e alinhar as especificações dos produtos que estiveram desenvolvendo. Assim, buscou-se melhor capacitá-los para o mercado de trabalho.

Para promover a fixação do conhecimento por parte dos estudantes, os autores deste trabalho desenvolveram um acervo contendo videoaulas e slides disponíveis online apresentando o conteúdo contido na ementa da disciplina Engenharia de Software. Desse modo, os alunos podem acessar as aulas em qualquer momento e estudar de forma dirigida os conhecimentos lecionados nos seguintes módulos: Revisão C++, Processo de Desenvolvimento de Software,

Qualidade e Métricas de Software, Especificação de Requisitos, Gestão de Projeto de Software, Planejamento e Estimativas de Software, Verificação e Validação de Software e Gestão de Configuração de Software.

3.5 Mudança no programa da disciplina

O programa da disciplina fornece um cronograma que informa em que semana e por quanto tempo o conteúdo da ementa da disciplina será ministrado na forma de aulas. Ao longo dos últimos anos, diversos programas foram utilizados a fim de avaliar a ordem na qual o conteúdo deve ser abordado para facilitar seu aprendizado.

Este programa foi remodelado de modo a satisfazer todas as necessidades levantadas durante este estudo. A experiência dos últimos três semestres confirmou que os trabalhos realizados em separado beneficiam o aprendizado. Surpreendentemente, também observou-se que para maior engajamento e melhor absorção do conteúdo por parte dos estudantes, os trabalhos devem ser executados em ordem inversa em que os conteúdos por eles exercitados são apresentados nos livros didáticos que, por sua vez, correspondia à ordem que vinham sendo executados pelos estudante nos semestres iniciais deste estudo. Isto é, o trabalho "2. Construção e teste de software" deve ser realizado após o trabalho "1. Concepção e projeto de software". Noutras palavras, mesmo que o ciclo de vida natural de um projeto de software seja Concepção, Projeto, Construção e Teste, e que apesar desta ser a ordem em que estes conteúdos são abordados nos livros didáticos, observou-se que foi benéfico ao aprendizado o ensino das atividades de concepção e projeto de software somente após os estudantes entenderem quais informações são necessárias para guiar estas atividades de construção e teste, pois os estudantes passaram a adquirir, documentar e prover tais informações com mais coerência a maturidade. É importante, registrar que a concepção e o projeto de software são majoritariamente documentados em linguagem natural e que, sem maturidade e discernimento, os textos produzidos são quase sempre ruidosos, prolixos, incompletos e, portanto, ineficazes à especificação de produtos de software.

Concluiu-se que após os estudantes entenderem na prática os desafios enfrentados por programadores e testers, papéis com os quais estão mais acostumados devido à experiências vivenciadas em disciplinas prévias, tornou muito mais fácil o aprendizado de conhecimento mais abstratos como a concepção, o planejamento e a gestão de projetos de software, pois eles já entendem que informações os programadores e testers demandam durante o desenvolvimento dos projetos.

3.6 Mudanças nas atividades avaliativas

No semestre de 2015/1 as atividades avaliativas seguiam o modelo tradicional de avaliação. A disciplina continha em seu programa duas provas teóricas que tinham como objetivo avaliar

o conhecimento dos alunos a partir da matéria dada em sala de aula. As provas eram aplicadas logo após o professor apresentar as matérias referentes a cada uma delas, isto é, a ementa da disciplina era dividida em duas partes para que a partir de cada uma delas fosse avaliado o conhecimento apresentado. Elas eram realizadas individualmente e sem consulta a outros materiais. A partir desse cenário, o que se esperava dos alunos era que eles se preparassem para cada uma delas, estudando apenas pelo material apresentado em sala de aula com apoio da bibliografia recomendada. Assim, eles tinham contato apenas com o que o professor lhes passava, ou seja, o professor era o único detentor do conhecimento e os alunos, em sua maioria, não participavam ativamente de seu aprendizado. Utilizando essa forma de avaliar o conhecimento dos alunos, notou-se que o aproveitamento nas provas não era satisfatório. Dessa forma, algumas mudanças foram realizadas ao longo dos semestres para melhoria do aprendizado.

A primeira mudança realizada foi a aplicação de apenas uma prova ao final do semestre e a adição de um trabalho prático individual como atividade avaliativa. A prova teórica foi mantida para avaliar em que medida o conteúdo teórico da disciplina foi assimilado pelos estudantes e também para auxiliar o professor na análise dos impactos das mudanças metodológicas de ensino. Durante um único semestre, houve a tentativa de abolir as provas, contudo observou-se que diante da ausência de provas os estudantes não se comprometiam com a leitura da bibliografia obrigatória da disciplina. O trabalho prático de longa duração permitiu ao professor tratar de problemas mais complexos e realistas que aqueles passíveis de serem resolvidos em uma prova Além disso, o trabalho prático oferecia aos estudantes a oportunidades de vivenciar o uso de processos, ferramentas e técnicas praticadas pela indústria. Desta maneira, esse trabalho prático aproxima as habilidades exercitadas na disciplinas com aquelas exigidas pela indústria.

No entanto, este único trabalho apesar de permitir a avaliação individualizada de cada estudantes, não oferecia vivência no trabalho em equipes colaborativas, um habilidade de suma importância sob a ótica da indústria. Este trabalho também não permitia aos estudantes vivenciar diferentes papéis (funções/cargos) existentes no ecossistema de software e entender quais deles ele desempenha com maior facilidade. Por estas razões um segundo trabalho foi planejado e adicionado às atividades avaliativas da disciplina. Desta maneira, ao final de 2018, os estudantes passaram a ser avaliados por um trabalho individual, um trabalho em grupo e um prova teórica individual.

3.6.1 Mudanças para atender exigências da indústria de software

Nos primeiros semestres do estudo foi observada uma carência no perfil dos estudantes de conteúdo que atendesse a demanda da indústria de software ao finalizarem o curso de Ciência da Computação. Os estudantes finalizavam o curso, em sua maioria, sem ter contato com o estado da arte em processos, métodos, técnicas, e tecnologias demandadas e utilizadas pela indústria de desenvolvimento de software. Por esse motivo verificou-se uma necessidade de mudança tanto no programa da disciplina como também na forma de ensino aprendizagem do conteúdo da

ementa para apresentar aos estudantes esses tópicos citados anteriormente. Os objetivos dessas mudanças foi agregar aos estudantes o que o mercado requer como pré-requisito e diferencial em suas contratações ,como também diminuir a mão de obra gasta pelas indústrias ao capacitar os profissionais recém saídos da academia.

Os trabalhos práticos e as dinâmicas em sala de aula, tiveram papel essencial para alcançar esses objetivos. Do mesmo modo, as listas de exercícios foram remodeladas para preparar os estudantes para lidar com software legado, atividade que consome aproximadamente 70% do esforço das empresas (Fox and D. Patterson, 2012).

A partir das dinâmicas e trabalhos individuais os estudante vivenciaram na prática o projeto software dirigido por API, a componentização de software, o desenvolvimento de arquiteturas em camadas, o uso de padrões de projeto de software e o emprego testes unitários e funcionais automatizados, seguindo um processo cíclico e dirigido por teste (TDD). Todas estas são práticas amplamente utilizadas na indústria para promover o reuso de código e para garantia de qualidade dos produtos de software. Ao desenvolver o trabalho prático em grupo, os estudantes além de exercitar as habilidades interpessoais (soft skills) também tiveram contato com a concepção e especificação dos requisitos de software, e com os diferentes funções/papéis existentes em uma equipe de software: Dono do Produto ou Analista de Negócio, Analista de Sistema, Arquiteto de Software, Engenheiro de Software, Engenheiro e Gerente de Projetos. Os estudantes precisaram colocar em prática técnicas como o desenvolvimento dirigido por comportamento (BDD) e seguir processos de desenvolvimento de software ágeis, tendo a liberdade de escolher a adoção de um entre os seguintes processos: Kanban, SCRUM ou BOPE. As diretivas para gestão de projetos definidas pelo Project Management Institute (PMI) também foram vivenciadas de forma prática, assim como o desenvolvimento de testes de aceitação, a documentação de código, o projeto arquitetural e a componentização de software. Não menos importante, os estudantes precisaram utilizar tecnologias amplamente utilizadas no mercado como o framework QT [], como as implementações de APIs para a conexão de bancos de dados relacionais (SQL) e para o desenvolvimento de interfaces gráficas com o usuário (GUI), assim como as ferramentas para controle de versões de código: GitHub ou GitLAB.

4 Resultados

Esta seção de texto descreve os resultados alcançados pelos autores durante a realização deste trabalho. Estes resultados dividem-se em grandes blocos a saber:

- **Programa da disciplina**: Ao longo deste estudo foram desenvolvidos diversos programas para a disciplina Engenharia de Software. Esta seção de texto apresenta somente o programa vigente no final de 2019 e que permanece em uso até os dias de hoje.
- **Objetos de Aprendizado**: São formados por conjuntos de slides, vídeo aulas, listas de exercícios, quizzes e dinâmicas para serem conduzidas pelo professor em sala de aula.
- **Estudos de casos** Estudos de casos: São formados por dois trabalhos acadêmicos que devem ser realizados ao longo de 6 semanas cada um. Estes trabalhos estão em acordo com metodologia de ensino PjBL e, portanto, resultam na execução de projetos de software, compreendem na entrega de produtos que tratam problemas realistas vivenciados pela indústria e, finalmente, colocam os estudantes como principais responsáveis por seu aprendizado.
- **Indicadores de impacto** Indicadores de impacto: São formados por um conjunto de indicadores quantitativos que avaliam o rendimento médio, o índice de evasão e o índice de retenção dos estudantes ao longo do tempo, assim como indicadores qualitativos que avaliam a qualidade emocional e psicológica do ambiente de ensino.

O programa da disciplina, os objetos de aprendizado e os estudos de caso foram desenvolvidos de forma colaborativa pelos autores deste trabalho. Os indicadores de impacto foram coletados pela estudante autora deste trabalho durante o semestre letivo, por meio de entrevistas e formulários respondidos pelo(a)s aluno(a)s da disciplina. Os indicadores de rendimento, evasão e retenção foram obtidos diretamente com o professor, ao final de cada semestre e registrado de maneira anonimizada pela autora. A autora principal deste trabalho foi a principal responsável por propor soluções para os problemas de ensino-aprendizado percebidos no decorrer da disciplina. Então, no período de tempo entre os semestres letivos, as soluções foram por ela desenvolvidas e preparadas, sob a supervisão do professor orientador deste trabalho. Desta maneira, o professor a cada semestre dispunha de versões melhoradas dos objetos de aprendizado. Cada um destes resultados será descrito em detalhes a seguir para melhor entendimento dos resultados alcançados neste trabalho

4.1 Programa da disciplina

Ao longo deste estudo foram desenvolvidos diversos programas para a disciplina Engenharia de Software. Esta seção de texto apresenta somente o programa vigente no final de 2019 e que permanece em uso até os dias de hoje, como pode ser visto na Figura 4.1 e Figura 4.2 .

Esse programa foi criado para cumprir o conteúdo obrigatório preestabelecido na ementa da disciplina. Dessa forma, o programa foi modelado para apresentar aos estudantes temas como modelagem clássica, modelagem orientada a objetos, projeto orientado a objetos, desenvolvimento modular, desenvolvimento dirigido por API (Application Programming Interface), qualidade de software, reuso de software, ferramentas para desenvolvimento de software, evolução de Software, a pesquisa e o futuro da Engenharia de Software. Ao final do curso, o que se espera é que os estudantes tenham vivência suficiente em cada um desses temas de forma a se sentirem confiantes para atuarem com profissionalismo no mercado de trabalho. Assim, os estudantes se tornariam capazes de combinar a visão acadêmica com o domínio das práticas e tecnologias demandas pela indústria de software..

4.2 Objetos de Aprendizado

Os objetos de estudos são formados por um conjuntos de slides, acervo de vídeo aulas, listas de exercícios e dinâmicas para serem conduzidas pelo professor em sala de aula. A criação desse material teve como principais objetivos engajar os estudantes em seu aprendizado, dar suporte ao professor ao transmitir o seu conhecimento e também servir como recurso para complementar o material didático utilizado no processo de ensino-aprendizagem dentro e fora de aula.

Baseando-se nesses objetivos um conjunto de slides e um acervo de videoaulas foi desenvolvido para complementar o material de aulas expositivas do professor, pois via-se uma necessidade de complementação no material didático preexistente. O acervo de videoaulas conta com vinte e um vídeos, os quais abordam todo o conteúdo exposto acima no programa da disciplina, oferecendo a base teórica de que o estudante necessita. Cada um dos vídeos cobre todo o conteúdo de uma aula. Eles iniciam aprendendo o sumário do conteúdo abordado naquela aula e possuem marcações nas mudanças de tópicos abordados. No entanto, acredita-se que seria benéfico aos estudantes subdividir cada vídeo em vídeos menores com duração máxima de 20 minutos, facilitando uma abordagem modularizada do processo de estudo.

Durante as aulas, notou-se a necessidade de criação de dinâmicas que atraíssem os estudantes a participar ativamente do seu aprendizado. Assim, utilizou-se conceitos da metodologia PBL como base para a criação dessas dinâmicas. Nesse sentido, as dinâmicas criadas apresentam problemas vividos no mercado de trabalho, porém abordando versões simplificadas destes problemas e que pudessem ser resolvidos rapidamente em sala de aula. As dinâmicas criadas foram

Aulas	Descrição do Conteúdo	Aulas	Descrição do Conteúdo
Aula 1 a 4	• Apresentação da Disciplina, Objetivo, Conteúdo, Material Didático e Método de Avaliação. • Ferramentas: "Porque C++.?"; Introdução ao ambiente de Desenvolvimento C++ em MinGw. Ferramentas: "Porque Qt?"; Introdução à biblioteca Qt. • Projeto arquitetural de software: Arquiteturas monolíticas, em camadas, cliente-servidor e distribuídas. • Estudo de caso: arquiteturas dos sistemas Linux, Windows DNA, Pilha de Protocolos TCP/IP, Java RMI, WhatsApp, TerraLab e TerraME". • Lista de exercício 1: Introdução a Engenharia de Software; • Lista de Exercício 2: "a vocação de C"; • Lista de Exercício 3: "Introdução a C++"	Aula 25 a 28	• Ferramentas: Controle de versão de software e controle de mudanças no GitHub. • Enunciado do trabalho prático – Parte 2: artefatos de software - documento de visão de sistema, escopo do projeto, missão e benefício do produto, backlog, storyboard, estórias de usuário, cenários de teste de aceitação..
Aula 5 a 8	• Reuso de código; • Projeto de software dirigido por API, fraco acoplamento entre módulos, alta coesão dos módulos, encapsulamento de informação e eficiência. • Dinâmica: Projeto de API de uma Classe container. • Enunciado do Trabalho prático. Parte 1: Desenvolvimento dirigido por API e teste (TDD) de um framework C++	Aula 29 a 32	Trabalho prático – Avaliação da parte 1‡ Seminários e entrevistas.
Aula 9 a 12	• Qualidade de Software em C++: regras para nomenclatura, forma canônica das classes, herança, delegação e polimorfismo, teste e integração continua de software segundo o TDD • Projeto modular de software: classes concretas, classes abstratas e o conceito de interface; • Desenvolvimento modular em C++: Programação Genérica, templates e iteradores em C++. Lista de Exercício 4:" Programação genérica em C++".	Aula 33 a 36	• Conceitos e princípios fundamentais para o desenvolvimento de projetos de software de software. • Qualidade de software: confiabilidade, manutenibilidade, usabilidade e desempenho; • Gestão de projetos segundo o Project Management Institure (PMI e PMBook) e o processo SCRUM; • Projeto, coleta e análise de métricas (de produto, processo e equipe). • Processos de desenvolvimento de software tradicionais e ágeis: RUP, SCRUM e XP
Aula 13 a 16	• Estudo de caso: "o tratamento de exceções em C++". • Introdução aos padrões de projeto de software: o padrão Singleton	Aula 38 a 40	• Análise de requisitos orientada a objetos: elicitação de requisitos, identificação de classes, objetos, heranças, relacionamento e interações. Notação UML. Elicitação de critérios de aceitação segundo o BDD.
Aula 17 a 20	• Auxílio na resolução do trabalho prático- parte 1: Elicitação de testes segundo a técnica BDD e a implantação de testes segundo o processo TDD	Aula 41 a 44	• Desenvolvimento de aplicações MVC em Qt.
Aula 21 a 24	• Auxílio na resolução do trabalho prático- parte 1: padrões de projeto de software method factory e composite; e o idioma handle-body (ou bridge)	Aula 45 a 48	• Parte 2: Desenvolvimento em 3 camadas Lógica de Negocio, Interface com o Usuário, Acesso a Dados) de uma aplicação CRUD (Create, Read, Update, Delete)
Aula 25 a 28	• Ferramentas: Controle de versão de software e controle de mudanças no GitHub. • Enunciado do trabalho prático – Parte 2: artefatos de software - documento de visão de sistema, escopo do projeto, missão e benefício do produto, backlog, storyboard, estórias de usuário, cenários de teste de aceitação..	Aula 49 a 52	• Refatoração de código e os ciclos incrementais e evolutivos dos processos de desenvolvimento software (BDD+TDD+SCRUM+PMI).

Figura 4.1 – Programa da disciplina ao final do ano de 2019 Parte 1

Dinâmica (1) – "Criação de um Projeto de uma API para uma Classe Container ", Dinâmica (2) - "Construção de testes funcionais para a função Fatorial", Dinâmica (3) - "Construção de uma

Aula 29 a 32	• Trabalho prático - Avaliação da parte 1: Seminários e entrevistas.	**Aula 53 a 56**	• APIs amplamente utilizadas na indústria de software: GUI em Qt, SQL em Qt, Threads em Qt e Soquetes em Qt.
Aula 33 a 36	• Conceitos e princípios fundamentais para o desenvolvimento de projetos de software de software. • Qualidade de software: confiabilidade, manutenibilidade, usabilidade e desempenho; • Gestão de projetos segundo o Project Management Institure (PMI e PMBook) e o processo SCRUM; • Projeto, coleta e análise de métricas (de produto, processo e equipe). • Processos de desenvolvimento de software tradicionais e ágeis: RUP, SCRUM e XP	**Aula 57 a 62**	• Trabalho prático - Avaliação da parte 2: Seminários e entrevistas.
Aula 38 a 40	• Análise de requisitos orientada a objetos: elicitação de requisitos, identificação de classes, objetos, heranças, relacionamento e interações. Notação UML. Elicitação de critérios de aceitação segundo o BDD.	**Aula 63 a 64**	• Trabalho prático - Avaliação da parte 2: Seminários e entrevistas.
Aula 41 a 44	• Desenvolvimento de applicações MVC em Qt.	**Aula 65 e 66**	• Exame especial - A ser realizada na forma de uma entrevista online.

Figura 4.2 – Programa da disciplina ao final do ano de 2019 Parte 2

arquitetura de software em camadas usando bibliotecas de ligação dinâmica (DLLs)" e Dinâmica (4) – "Concepção de uma aplicação CRUD básica". Observou-se que as práticas que dinâmicas induziram os estudante a realizar, refletiu no maior engajamento dos estudantes e culminaram no melhor entendimento do conteúdo exposto. Estes fatos ficaram evidentes quando os estudantes se mostraram mais participativos, adotaram uma postura mais profissional ao durante discussões sobre os problemas e soluções abordados, compartilhando suas opiniões e seus conhecimentos.

As nove listas de exercícios desenvolvidas para complementar o material utilizado pelo professor tiveram como objetivo preparar os estudantes para realizar as atividades avaliativas da disciplina e para que os estudantes tivessem contato com software legado, gerando para eles a necessidade de realizarem correções e melhorias nos códigos fontes que acompanham estas listas de exercícios. As listas foram desenvolvidas em linguagem C++, muitas delas utilizando o framework Qt, e contêm falhas propositalmente projetadas. Assim, observou-se que os estudantes vão em busca dessas falhas para solucioná-las. As listas criadas foram Lista de exercício 1: Introdução a Engenharia de Software; Lista de Exercício 2: "a vocação de C"; Lista de Exercício 3: "Introdução a C++"; Lista de Exercício 4: "Programação genérica em C++"; Lista 05: "Instalação e Configuração do Qt"; Lista 06: "Desenvolvimento de GUI em Qt; Lista 07: "Uso da Ferramenta Qt Designer"; Lista 08: "Múltiplas Janelas no Qt Designer"; Lista 09: "Testes Unitários de GUIs em Qt".

A linguagem C++ foi adotada devido à experiência prévia dos estudantes com esta linguagem de programação em diversas outras disciplinas. Desta maneira, buscou-se evitar que o aprendizado de novas linguagens trouxessem desafios ao aprendizado do conteúdo obrigatório. O framework Qt foi utilizado por dominar o mercado de desenvolvimento de software em C++ com larga vantagem se comparado ao uso de outros frameworks. Ele provê APIs para

o desenvolvimento de Interfaces Gráficas com o Usuário (GUI), de aplicações multitarefas (multithreads), de aplicações em redes (sockets) e de aplicações baseadas em bancos de dados relacionais (SQL).

4.3 Estudos de casos

Esta seção apresenta os estudos de caso desenvolvidos neste trabalho e que figuram como principais atividades avaliativas desempenhadas pelos estudantes. São dois trabalhos acadêmicos que devem ser realizados ao longo de 6 semanas cada um. Estes trabalhos, exigem o uso prático de processos ágeis de desenvolvimento de software, apoiados por ferramentas para automação do processo de desenvolvimento. A cada sprints semanal, os estudantes devem entregar incrementos ou evoluções de um produto de software. O trabalho prático individual convida o estudante a projetar, construir e testar uma API (Application Programmig Interface) de uma biblioteca C++ para desenvolvimento de simulações contínuas. O trabalho prático em grupo convida os grupos de estudantes a colaborarem a fim de conceber e especificar o escopo e os requisitos de uma aplicação CRUD (cadastrate-read-update-delete) de pequeno porte. Este último exige que cada grupo encomende o desenvolvimento do produto por ele especificado a outro grupo, responsabilizando-se por validar os artefatos produzidos por esse grupos, atuando como cliente. Simultaneamente, todos os grupos também atuam fornecedores de soluções, se comprometendo a construir, testar e implantar a aplicação CRUD encomendada por um outro grupo, agindo como fornecedor da solução. Assim, os estudantes vivenciam a dualidade cliente-fornecedor, recebendo críticas ao seu trabalho, observando quais informações estão ausentes em suas especificações e que impediram por completo o entendimento do produto encomendado. Por outro lado, os estudantes são educados a criticar o trabalho alheio de forma assertiva e, finalmente, exercitam suas habilidades interpessoais nas colaborações dos membros de seu grupo (fornecedor) e com os membros de outros grupos (clientes).

4.3.1 Trabalho prático individual - Desenvolvimento de um Framework C++ para a Construção de Simulações Baseadas na Dinâmica de Sistemas

Neste trabalho, os estudantes têm como objetivo projetar, construir e testar uma API de um biblioteca C ++ destinada à simulação baseados na Teoria Geral de Sistemas formalizada pelo biólogo austríaco Ludwig von Bertalanffy (1968). Este problema foi proposto porque traz para o estudante a dificuldade de construir um código que será utilizado por outros programadores, uma realidade comum nas empresas de software formadas por diversos times de desenvolvimento. Além disso, é consenso que empresas se preocupem em produzir códigos que possam ser reutilizados em projetos futuros. Assim, as questões de reusabilidade e de estabilidade de API

ganham relevância, uma API mal projetada implicaria em muitas mudanças que refletiriam na necessidade de reconstrução das aplicações sobre ela desenvolvidas.

O domínio de problema deste trabalho, isto é, desenvolver um simulador, exige que os estudantes produzam um componente de software cujas regras de simulação somente serão conhecidas futuramente, pelos programadores que irão desenvolver uma nova aplicação sobre a biblioteca, isto é, um modelo de simulação. Mesmo sem poder antever a natureza destas regras (funcionalidades), os projetistas da biblioteca (estudantes) precisam ser capazes de empacotar-lá em componentes executáveis e garantir seu perfeito funcionamento em todas as condições de uso futuro. Neste contexto, os estudantes são conduzidos a abandonar uma arquitetura de biblioteca e construir o simulador de acordo com uma arquitetura de framework (arcabouço). Enquanto as bibliotecas de códigos são conjuntos de funcionalidades que não se organizam em uma arquitetura bem definida e são úteis para a construção de aplicações de diferentes arquiteturas, os frameworks possuem arquiteturas bem definidas que traduzem as decisões de projetos que especialistas conceberam para solucionar os principais problemas encontrados no domínio de problema a que se aplicam. Portanto, os usuários dos frameworks (programadores) podem desenvolver aplicações apenas pela customização de seus componentes, ganhando mais produtividade e qualidade. Ao contrário das bibliotecas, é o framework quem dita o fluxo de controle da aplicação, mecanismo conhecido como inversão de controle.

Para desenvolver um framework de simulação, os estudantes precisarão desenvolver uma API extensível e compreender que o mecanismo de herança presente na Orientação por Objetos é útil não somente para o reaproveitamento das representações dos objetos, podendo ser utilizados para a especialização de comportamentos no nível da camada de aplicação. Neste contexto, durante a implementação do framework, os estudantes terão de compreender conceitos como injeção de dependência (Fower, 2004) e compreender o funcionamento do mecanismo de callback functions (Sommerville, 2007). Ambos são conceitos essenciais da Engenharia de Software, muito úteis à implementação de arquiteturas onde as regras de negócio aparecem desacopladas dos requisitos tecnológicos, promovendo o reuso e a boa manutenabilidade do código.

Para auxiliar os estudantes, inicialmente, foi disponibilizado o enunciado do trabalho, que explica os preceitos da Teoria Geral de Sistemas, assim como foi dada aos estudantes informações completas sobre a estrutura de dados e o algoritmo que implementa a solução do problema, isto é, foi dada a implementação de um simulador discreto. Estes fatos deixam claro que o desafio deste trabalho não é a implementação dos aspectos funcionais ou estruturais da biblioteca. Certamente, o foco do trabalho é dado ao projeto de uma API que privilegia o reuso de código e à construção e teste de uma implementação cuja arquitetura é bem definida e tem sua qualidade garantida, por meio da componentização de serviços fracamente acoplados e por meio de teste automatizados.

A complexidade do domínio de aplicação, Teoria de Sistema, e a simplicidades da implementação dos simuladores discretos para esse domínio, contando com apenas três classes e um algoritmo formado dois laços de repetição não encadeados que percorrem um vetor, coloca os

estudante diante de uma realidade comum na indústria: A necessidade de gerar soluções simples para discursos elaborados dos clientes ou usuários de um produto.

Detalhadamente, o enunciado do trabalho dispõe de uma explicação sobre o desenvolvimento de uma linguagem para a descrição de sistemas dinâmicos onde sistemas são descritos como variáveis numéricas (acumuladores) que representam o estoque de certa quantidade de energia ou matéria, e as interações entre os sistemas são representadas por fluxos que carregam energia ou matéria de um sistema para outro. A linguagem DYNAMO, originalmente desenvolvida por J. W. Forrester em 1968, utiliza um retângulo para denotar sistemas e uma seta para denotar fluxos, onde a direção da seta indica o sentido do fluxo (da origem para o destino) pode ser representado pela Figura 4.3

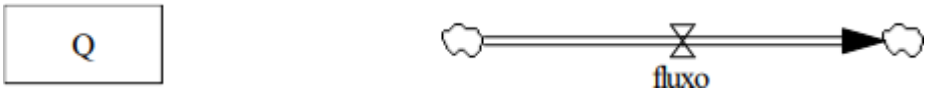


Figura 4.3 – Notação da linguagem DYNAMO: setas denotam fluxo e retângulos denota sistemas

O algoritmo geral para a simulação de modelos baseados na Teoria de Sistemas independem das hipóteses do modelo (aplicação da biblioteca). Estas hipóteses, assim como as regras dos modelos, serão definidas pelo modelador (usuário da biblioteca) com base em seu conhecimento sobre o domínio de problema de seu interesse. Durante a simulação, a cada instante de tempo, o algoritmo simula sequencialmente todos os fluxos definidos no modelo e os valores resultantes são armazenados temporariamente. Depois, o algoritmo percorre novamente o vetor de fluxos subtraindo os respectivos valores dos sistemas na origem dos fluxos e os adiciona aos sistemas no destino dos fluxos. Desta maneira, a quantidade de energia armazenada em cada sistema é atualizada. No final da simulação, o algoritmo reporta aos usuários as quantidades estocadas em cada sistema que forma o modelo.

O trabalho é dividido em ciclos de desenvolvimento semanais, denominados sprints, em que versões melhoradas da biblioteca devem ser entregues ao professor de acordo com o que foi demandado em cada sprint. A seguir relata-se a divisão de entregas por sprints que resultou no melhor desempenho dos estudantes, ao longo dos 4 anos em que este estudo foi realizado. A sprint 1 deve resultar na especificação da API da biblioteca. A sprint 2 deve resultar na implementação mais simplista da API projetada, contendo testes funcionais para verificar sua qualidade. Os estudantes são convidados a entender como um projeto de software deve ser organizado em uma estrutura de diretório. Ele também tem o primeiro contato prático com o conceito de testes regressivos. Na sprint 3, os estudantes compreendem de forma prática os benefícios da documentação de código e compreendem como a estrutura de dependência deve

ser analisada para garantir o fraco acoplamento entre os módulos do software. Na sprint 4, os estudantes compreendem como o desacoplamento entre os módulos deve ser implementado separando-se as interfaces das implementações dos módulos. Eles também devem entregar os testes unitários uma vez que, à esta altura do projeto, a API já pode ser considerada estável e provavelmente sofrerá pouquíssimas mudanças. Na sprint 5, os estudantes devem entregar uma versão da biblioteca dividida em componentes fracamente acoplados e organizados em um estrutura multicamadas. Para isso, eles precisam colocar em prática os conhecimentos acerca do padrão de projeto Factory Method e compilar os componentes na forma bibliotecas de ligação (dynamic link library). No sprint 6, os estudante precisam entregar uma versão da biblioteca onde as abstrações e implementações das podem ser evoluídas de forma independente, além de implementar um coletor de lixo (garbage collector) simplificado para evitar problemas como o vazamento (leak) de memória. Para isso, eles precisam utilizar o idioma Handle-Body (também conhecido como Bridge) para separar abstrações e implementações e, então, colocar as classes concretas em uma formas canônicas. Neste momento, fica claro para o estudante que o uso de padrões de projeto deve ser pensado considerando as necessidades do projeto de maneira a evitar o uso exagerado de engenharia (over engineering), todo aumento de complexidade do código deve ser justificado pelos benefícios gerados. Finalmente, no sprint 7, os estudantes são convidados a refatorar completamente o código de suas bibliotecas para atender um requisito do domínio de problema, no qual sistemas podem ser definidos recursivamente a partir de outros sistemas. Para isso, eles são convidados a implementar uma solução simples para o problema complexo, reconhecendo que estruturas recursivas são problemas recorrentes na computação, e que o padrão de projeto Composite pode ser utilizado para sua solução.

- Sprint 1 - Projeto da API do simulador de Sistemas Dinâmicos

O estudante irá apresentar ao professor um arquivo PDF, de nome "[nome-estudante]-sprint1.pdf", contendo o projeto da API do seu simulador que executa modelos de simulação descritos segundo paradigma de dinâmica de sistemas. Para isso, ele deve responder às questões:

1. Identifique os “casos de uso” aos quais sua API deverá satisfazer. Então, identifique os “critérios de aceitação” do cliente. Dica: Baixe o software “Vensim PLE”, abra o arquivo “ValidacaoMyVensim.mdl” e o execute. Assim, os “critérios de aceitação” poderão ser identificados.

2. Valide os “casos de uso” e “critérios de aceitação” com o cliente (papel interpretado pelo professor).

3. Para projetar sua API, estude diversas alternativas de uso dessa API na implementação dos modelos que configuram os “casos de uso” validados pelo cliente. Estes estudos, devem dar origem a “cenários de testes” que verificam os diversos “casos de uso” e “critérios de aceitação”. Finalmente, apresenta os pseudo-códigos materializam os “cenários de teste”

4. Utilize a notação UML (diagramas de classe) para especificar a API (Application Programming Interface) de uma biblioteca C++ destinada ao desenvolvimento de modelos ambientais baseados na Teoria de Sistemas.

- Sprint 2 - Classes concretas na Forma Canônica + Testes Funcionais

O código do simulador deve ser armazenado em uma estrutura que permita sua fácil manutenção e evolução. Para isso, todos os arquivos fonte gerados para o projeto devem ser armazenados em um único diretório, que recebe o nome do produto principal, os arquivos que formam o produto deve ser armazenados em um subdiretorio chamado "src", os arquivos de testes armazenados no diretório "test"e os arquivos binarios (exe, dll, lib, so) no diretorio "bin". Desta maneira a seguinte estrutura será gerada e haverá 3 arquivos "main.cpp- 1 para o produto, 1 para os testes funcionais e outro para os testes unitários.A forma geral da árvore de diretório do projeto pode ser vista na Figura 4.4 a seguir:

```
- \MyVensim
     +\bin
    + \src
         main.cpp
        mySim.cpp
       mySim.h
    - \test
          - \unit
              main.cpp
              unit_tests.h
              unit_tests.cpp
          - \funcional
             main.cpp
             funcional_tests.h
             funcional_tests.cpp
```

Figura 4.4 – A forma geral da árvore de diretório do projeto

É importante salientar que os 3 testes funcionais devem ser separados em 3 diferentes funções. Esta 3 funções devem ser colocadas no arquivo "funcional-tests,cpp"e suas assinaturas no arquivo "funcional-tests,h"que deve ser includa no arquivo "main.cpp"(testesRegressivos) do subdiretório "/MyVensin/test/funcional", conforme ilustra o código o arquivo "main"na Figura 4.5 abaixo:

- Sprint 3 - Documentação de código em Doxygen (com gráficos)

Os estudantes devem documentar o codigo do simulador por meio da ferramenta Deoxygen que pode ser obtida em: http://www.doxygen.nl/download.html. É essencial que os gráficos de

```
#ifndef  MAIN_FUNCIONAL_TESTS
#define MAIN_FUNCIONA_TESTS
#include "funcional_tests.h"
#include "..\..\src\model.h"
#include "..\..\src\system.h"
#include "..\..\src\flow.h"
int main(){
    exponentialFuncionalTest();
    logisticalFuncionalTest();
    complexFuncionalTest();
    return true;
}
#endif
```

Figura 4.5 – Exemplo de código arquivo main

dependência de arquivos e das estruturas de classes sejam gerados. Para isso, muitas vezes é necessário instalar a ferramenta Graphviz disponível em https://www.graphviz.org/. Veja explkicações detalhadas em em: http://www.doxygen.nl/manual/diagrams.html

- Sprint 4 - API definidas por interfaces separadas das Classes Concretas + Testes Unitários

O código do simulador deve ser armazenado em uma estrutura que permita sua fácil manutenção e evolução. Para isso, todos os arquivos fonte gerado para o projeto devem ser armazenados em um único diretório, que recebe o nome do produto principal, os arquivos que formam o produto deve ser armazenados em um subdiretório chamado "src", os arquivos binários dos componentes no diretório "bin"e os arquivos de testes armazenados no diretório "test". Desta maneira a seguinte estrutura será gerada e haverá 3 arquivos "main.cpp- 1 para o produto, 1 para os testes funcionais e outro para os testes unitários. A Forma Geral da Árvore de Diretório do Projeto pode ser analisada na Figura 4.6 a seguir:

Seguem algumas informações importantes para o desenvolvimento do trabalho:

- Para cada classe do produto, por exemplo "System"haverá um arquivo e "*.h"e um arquivo "*.cpp"contendo os testes unitários desta classe.
- O nome desses arquivos devem ser formados pelo nome da classe precedido pelo prefixo "unit-", por exemplo "unit-System.h"e "unit-System.cpp"como pode ser visto nas imagens representadas pelas Figura 4.9 e Figura 4.10.
- Cada um dos métodos das classes deve ter uma função independente de teste unitário para testá-lo cujo nome é formado pelo prefixo "unit-"concatenado ao nome da classe e ao nome do método que será testado, por exemplo "void unit-System-getValue(void)".

```
\MyVensim
  + \bin
  + \src
        main.cpp
  - \test
  - \unit
      main.cpp
      unit_tests.h
      unit_tests.cpp
     unit_system.h
      unit_system.cpp
      ...
 - \funcional
      main.cpp
    funcional_tests.h
    funcional_tests.cpp
```

Figura 4.6 – Exemplo de Forma Geral da Árvore de Diretório do Projeto

- Cada arquivo "*.cpp"deve conter uma função de que invoca todos os testes unitários e cujo nome é formado pelo prefixo "run-unit-test-"concatenado o nome da classe testada. Neste exemplo o nome seria "void run-unit-test-System(void)"
- Para funções de escopo global, o nome das funções que implementam os testes unitários devem anexar o prefixo "unit-"ao nome da função testada.
- Por exemplo, a função de escopo global "...minhaFuncao(...)"deve ser testada pela função "void unit-minhaFunca(void)"e os códigos dos testes unitários devem ser armazenados no arquivo "unit-tests,cpp"e as assinaturas no arquivo "unit-tests,h"como estão representadas nas Figuras 4.7 e Figura 4.8.
- Esse arquivo "*.cpp"deve conter uma função de que invoca todos os testes unitários e cujo nome é formado pelo prefixo "run-unit-test-"concatenado string "globals". Neste exemplo o nome seria "void run-unit-test-globals(void)".
- O arquivo "main.cpp"do subdiretório "/MyVensin/test/unit"deve incluir todos os arquivos .h e invocar o método "void run-unit-test-...()"de cada arquivo "*.cpp". , conforme ilustra a Figura 4.11 contendo os códigos.

- Sprint 5 - Arquitetura em camadas: Componentização + Fábricas

A implementação correta de uma ARQUITETURA em CAMADAS exigirá a componentização (DLLs, SOs) do seu produto.

```
#ifndef UNIT_TESTS
#define UNIT_TESTS
        void unit_minhaFuncao1( void );
        void unit_minhaFuncao2( void );
        void run_unit_tests_globals( void );
#endif
```

Figura 4.7 – Exemplo de Arquivo unit-tests.h

```
#include <assert.h>
#include "unit_test.h"
#include "..\..\src\minhasFuncoes.h"
  void unit_minhaFuncao1( void ){
        /*insira o codigo de depuração, que invoca a função
global ...minhaFuncao1(...) aqui"*/
        assert(/*coloque a condição a ser verificada
aqui*/);
  }
  void unit_minhaFuncao2( void ){
        /*insira o codigo de depuração, que invoca a função
global ...minhaFuncao2(...) aqui"*/
        assert(/*coloque a condição a ser verificada
aqui*/);
  }
  void run_unit_tests_globals( void ){
        unit_minhaFuncao();
        unit_minhaFuncao();
  }
```

Figura 4.8 – Exemplo de Arquivo unit-tests.cpp

```
#ifndef UNIT_SYSTEM
#define UNIT_SYSTEM
        void unit_System_constructor( void);
        void unit_System_destructor( void);
        void unit_System_setValue( void);
        void run_unit_tests_System( void );
#endif
```

Figura 4.9 – Exemplo de Arquivo unit-System.h

- Sprint 6 - Handle Body

Aplique o idioma Handle-Body, também conhecido como bridge, em todas suas classes concretas. Faz sentido aplicá-lo às interfaces? Para isso, utilize o template fornecido pelo pro-

```
#include "unit_System.h"
#include "..\..\src\system.h"
  void unit_System_constructor( void){
        System s1();
        assert( s1.getValue() == 0 );
        System s2(10);
        assert( s2.getValue() == 10 );
  }
void unit_System_destructor( void){ };
void unit_System_getValue( void){
        System s(10);
        assert( s.getValue() == 10 );
}
void unit_System_setValue( void ) {
        System s();
        s.setValue(100);
        assert( s.getValue() == 100 );
}
void run_unit_tests_System( void ){
        unit_System_constructor();
        unit_System_destructor();
        unit_System_getValue();
        unit_System_setValue();
}
```

Figura 4.10 – Exemplo de Arquivo unit-System.cpp

```
#include "unit_tests.h"
#include "unit_System.h"
int main(){
        run_unit_tests_globals();
        run_unit_tests_System();
        /* invoque outros metodos "void run_unit_tests_...
(void)" aqui */
        return true;
}
```

Figura 4.11 – Exemplo de Arquivo Arquivo main.cpp

fessor: <https://www.terralab.com.br/lib/exe/fetch.php?media=terralab:curso:tecprog:engsoft_v06handlebody.rar> Tomando como base, o arquivo main.cpp, produza um teste unitário que verifique o uso que você fez deste template.

- Sprint 7 - Composite

Aplique o padrão de projeto COMPOSITE e consiga que Sistemas possam ser definidos a partir

de Sistemas que, por sua vez, podem ser formados por outros Sistemas e assim por diante.

4.3.2 Trabalho prático em grupo - CRUD em 3 camadas

Neste trabalho, os estudantes têm como objetivo conceber, especificar, implementar, testar e implantar um sistema CRUD (castrate-read-update-delete) simples para domínio de sua escolha. Sistemas CRUD caracterizam como aplicações que, por meio de uma interface gráfica (GUI), permitem a seu usuário cadastrar, consultar, atualizar e apagar informações em um banco de dados (neste caso, relacional). Para auxiliar os estudantes no desenvolvimento deste trabalho, os autores desenvolveram e disponibilizaram como base um "Documento de Especificação de Requisitos" (DER) que deveria ser preenchido pelos grupos de estudantes. A estrutura e informações solicitadas neste documento, obrigam os estudantes a compreender a teoria sobre especificação de projetos de software e também vivenciar os problemas práticos cujas soluções são demandados pelo cotidiano da indústria de software. Para o desenvolvimento deste trabalho sugere-se as seguintes recomendações:

- Deixar livre a escolha do domínio de atuação do sistema CRUD a ser desenvolvido, tanto para promover o engajamento dos estudantes quanto para que os estudantes possam vivenciar o desenvolvimento de uma vasta diversidade de produtos (ao observarem as discussões acerca dos produtos de seus colegas);
- O trabalho deve ser realizado em grupo, contendo idealmente 3 pessoas atuando nos papéis de Engenheiro de Teste, Engenheiro de Software e Dono do Produto (Product Owner). Aconselha-se que os grupos sejam formados por no máximo seis pessoas, quando é possível introduzir papéis como Analista de Experiência do Usuário, Analista de Infraestrutura (Operações) e Gerente de Projetos. Desta maneira, os estudantes podem exercer diferentes papéis existentes no mercado de trabalho. Neste texto, entende-se que o Dono do Produto, o Analista de Negócios e o Analista de Sistemas executam tarefas muito similares e são, neste sentido, entendidos para simplicidade como sinônimos;
- O sistema CRUD a ser desenvolvido deve conter pelo menos uma classe que é container/gerente de instâncias de objetos de outra classe de objetos. Um exemplo seria uma escola que contém professores, disciplinas e estudantes. O sistema deve ser de baixa complexidade, envolvendo no máximo dez classes de objetos do domínio de aplicação;
- Os tópicos do DER devem ser preenchidos pelos estudantes utilizando uma linguagem mais próxima ao domínio de aplicação, evitando-se termos técnicos da computação, ou seja, utilizar uma linguagem baseada no jargão utilizado pelo cliente (mesmo que este seja imaginário) para facilitar a validação da especificação pelo próprio cliente;
- Cada grupo irá encomendar seu desenvolvimento a outro grupo, responsabilizando-se por validar os artefatos produzidos por esse grupos, atuando como cliente.

- Simultaneamente, cada grupo também deve se comprometer a construir, testar e implantar a aplicação CRUD encomendada por um outro grupo, agindo como fornecedor da solução.

O modelo de DER que é disponibilizado aos estudantes contém os seguintes tópicos que deverão ser preenchidos durante a execução do trabalho, como pode ser visto na Figura 4.12.

Índice

Figura 4.12 – Índice do Documento de Especificação de Requisitos

De acordo com o índice apresentado, primeiramente cada grupo deverá descrever a seção (1) Objetivo que deverá conter o propósito do DER. Logo após, deverá descrever a (2) Descrição do Produto que se dividirá em duas seções que são o (2.1)-Escopo do Produto e (2.2) Serviços Oferecidos pelo Produto.

Na subseção (2.1) Escopo do Produto , o grupo deve detalhar o produto a ser desenvolvido, descrevendo, o seu nome e o nome de seus principais componentes, além da missão, dos limites e dos serviços oferecidos pelo produto. É importante ressaltar que o cliente e o Dono do Produto

(estudantes designados para esse papel) atuam em conjunto na etapa em que se estabelecem os escopo do produto, buscando manter o projeto focado em solucionar o problema apresentado pelo cliente. Na negociação entre os grupos (cliente-fornecedor), a missão, os benefícios esperados, e os limites do produto devem ser analisados, documentados e justificados para que no futuro sirvam como comprovação do que foi acordado pelas duas partes. Assim, os grupos terão subsídios suficientes para gerir as expectativas dos clientes ao longo do projeto.

Ao receber a primeira versão do documento dos estudantes, observou-se por diversas vezes uma grande dificuldade dos estudantes em entender a diferença de cada uma das subseções da seção (2.1) Escopo do Produto. Desta forma, torna-se necessário que o professor as analise junto com os grupos, guiando-os. A partir de um determinado ponto do projeto, é necessário que seja firmado um contrato entre os grupos, passando a responsabilizar cada grupo cliente a prover feedbacks aos grupo fornecedores, conferindo a eles a autonomia necessária à construção da confiança que necessitam de uma boa atuação profissional. Observou-se que esse processo leva ao amadurecimento dos grupos no decorrer da execução dos projetos em que atuam tanto como cliente quanto como fornecedores de solução.

Na seção (2.2) Serviços Oferecidos pelo Produto, os grupos deverão especificar quais serão os serviços oferecidos pelo produto e quais serão os atores (tipos de usuário) que irão ter acesso a cada um dos serviços. Uma das principais críticas aos métodos ágeis é o fato de não evoluírem em fases e de não existir fase de projeto na qual a arquitetura dos produtos de software são pensadas como um todo, considerando todos os requisitos elucidados. Como novos requisitos e as demandas por mudanças chegam a todo momento e são, em geral, aceitas, os projetos tendem a se tornarem colchas de retalhos. Assim, ao definir quais serão os serviços oferecidos pelo produto, o estudante deverá ser orientado a seguir os conceitos de alta coesão e fraco acoplamento entre serviços, de forma a produzir uma arquitetura baseada em micro-serviços e a estruturar o backlog do produto em uma hierarquia de serviços desacoplados formados por uma coleção de funcionalidades coesas. Desta maneira, o grupo de desenvolvimento será impelida a implementar e observar as vantagens em se produzir componentes modularizados, cada componente responsável por oferecer um único serviço. Por serviços de alta coesão entende-se aqueles que possuem todas funcionalidades necessárias para desempenhar a sua missão, minimizando a interdependência entre os serviços. Contudo, nem sempre é possível evitar que um serviço dependa de outros. Neste cenário, por fraco acoplamento entende-se que sempre que necessária a comunicação entre serviços para obtenção de dados ou de determinadas funcionalidades, em que essa comunicação se dê por meio da API de cada serviço, evitando-se acesso direto a estes itens. Nesse momento, também observou-se a importância de ressaltar para os estudantes a diferença de granularidade entre serviços e funcionalidades: Serviços são compostos por funcionalidades e na hierarquia do backlog dos produtos, serviços e funcionalidades devem ser mantidos nos níveis específicos, sendo que as funcionalidades são sempre os nodos folha da hierarquia do backlog.

Ao definir quais serão os atores do software, espera-se que os grupos identifiquem e

informem quais serão as classes de usuários do produto. Como por exemplo, podemos citar um "sistema acadêmico" que irá disponibilizar as notas das disciplinas para os estudantes de uma escola, tendo como atores: "Professor" e "Estudante". Cada grupo também deve ser capaz de elucidar a que serviços cada classe de ator terá acesso de escrita ou leitura. Observou-se que há por parte dos estudantes muita confusão entre o conceito de ator (papel desempenhado por um usuário) e o conceito de usuários (pessoa física, sistema ou equipamento que utiliza a interface de um software). É sempre necessário ressaltar que um mesmo usuário pode desempenhar diversos papéis ao utilizar um software. Por exemplo, uma mesma pessoa física, por ter sido estudante de uma escola e atualmente atuar como professor desta. Assim, ao utilizar o "sistema acadêmico" desta escola, ele poderá atuar como "Professor" para lançar as notas dos alunos, como pode atuar como "Estudante" ao recuperar seu histórico escolar. Para a documentação dos atores e prerrogativas de acesso aos serviços de um sistema, os estudantes são convidados a especificar os dois diagramas em notação UML - Unified Modeling Language: Diagrama de Hierarquia de Atores e Diagrama de Contexto.

Na seção (4) Requisitos de Interface, os grupos de estudantes deverão documentar os storyboards dos componentes do produto. O storyboard apresenta as telas desse componentes, os campos nelas presentes e os comportamentos esperados para cada ação do usuário. Observou-se ao longo dos semestre, que os estudantes preocupam-se, na grande maioria das vezes, apenas com a boa aparência das telas e da completude das informações coletadas ou exibidas nos campos que as formam. Neste momento, deve-se orientar aos grupos de estudantes que é de suma importância apresentar o fluxo de transições entre as telas para que o cliente e a equipe de desenvolvimento tenham o mesmo entendimento sobre o comportamento esperado para o produto. Para o cliente, o storyboard é considerado o melhor artefato de software para validar sua especificação. No início dos projetos, quando os requisitos identificados ainda são instáveis e pouco compreendidos até mesmo pelo cliente e, portanto são passíveis de muitas mudanças, os grupos devem compreender que as telas presentes no storyboard devem ser simples, evitando-se esforço que encareceria o projeto e geraria muito retrabalho. A Figura 4.13 ilustra essa situação.

Na seção seguinte, (5) Requisitos Funcionais, os grupos deverão apresentar o backlog do produto e a sprint backlog, conforme estabelece a metodologia ágil SCRUM. Para a especificação do backlog do produto, deverá ser apresentado todos os serviços e suas funcionalidades. Assim, ao estimar o tempo para desenvolvimento de cada funcionalidade, os grupos poderão estimar o tempo de implementação dos serviços, compreendendo e em quantos ciclos de desenvolvimento o projeto finaliza, ou seja, quantas sprints serão necessárias para que todos os serviços sejam implementados. Neste momento, é importante ressaltar aos estudantes que a missão não é produzir o melhor software possível dentre de suas habilidades. O importante, é se comprometer a desenvolver um software que resolva o problema a que se destina dentro do prazo estipulado pelo cliente (neste caso, prazo determinado pelo professor). Eles precisam ser conscientizados de que a busca pelo software perfeito é a causa de falha de muitos projetos e que um produto bem sucedido e entregue no prazo têm mais chances de conseguirem um oportunidades de

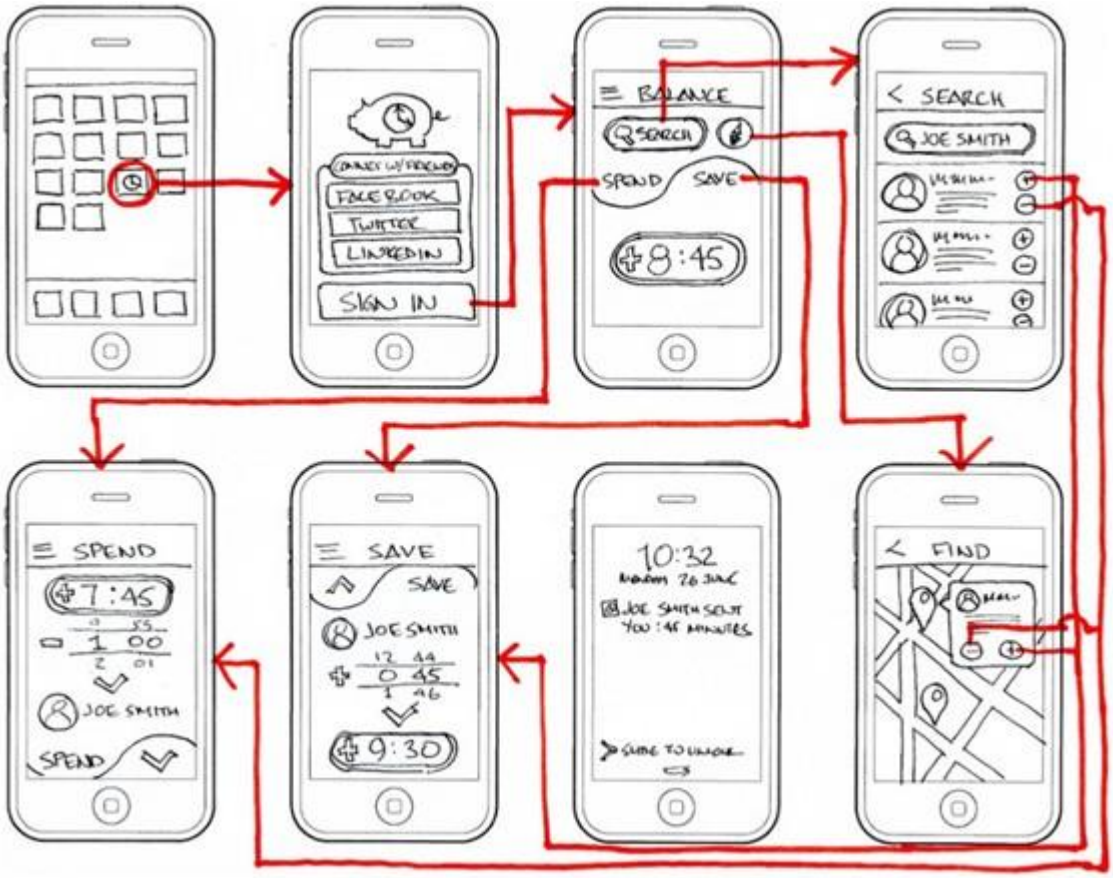


Figura 4.13 – Exemplo de Storyboard de um produto de software

serem melhorado em novos projetos onde serão desenvolvidas versões melhoradas do produto. Conforme discutido anteriormente, deve-se apresentar as vantagens de se criar uma arquitetura de microsserviços baseada nos projetos de uma APIs bem definidas para cada um dos serviços. Além da arquitetura de micro-serviço, o PMBOK é também base para estruturar o backlog de forma hierárquica conforme a decomposição de trabalho (work breakdown structure) utilizada nas melhores práticas de gestão de projetos.

Para a especificação do sprint backlog, os grupos definirão que funcionalidades serão entregues na próxima sprint do projeto. Observou-se que os estudantes tendem a definir sprints backlogs por componentes ou serviços, se comprometendo a entregar um serviço a cada sprint. Eles precisam ser conscientizados que esta não é uma boa estratégia. Espera-se que, a cada sprint, os grupos disponibilizem um incremento de software que agregam valor ao cliente, ou seja, dar ao cliente uma experiência no uso do software – mesmo que limitada - na qual ele enxergue retorno de seu investimento. Então, deverá ser analisado quais serão as funcionalidades que cumprirão esse objetivo, decidindo o que irá compor as épicas do projeto, ou seja, quais serão as funcionalidades dos diferentes serviços planejados anteriormente irão compor cada uma das entregas feitas ao cliente. A Figura 4.14, mostra que as épicas tendem a ser formadas por funcionalidades oriundas de diferentes serviços, formando cortes transversais na hierarquia do backlog. Nesta figura, os

termos ServX designam serviços e os termos FX denotam funcionalidades, enquanto que EpicX identificam 3 épicas distintas que, por sua vez, são formadas pelas funcionalidades circundadas pelos polígonos de mesmas cores.

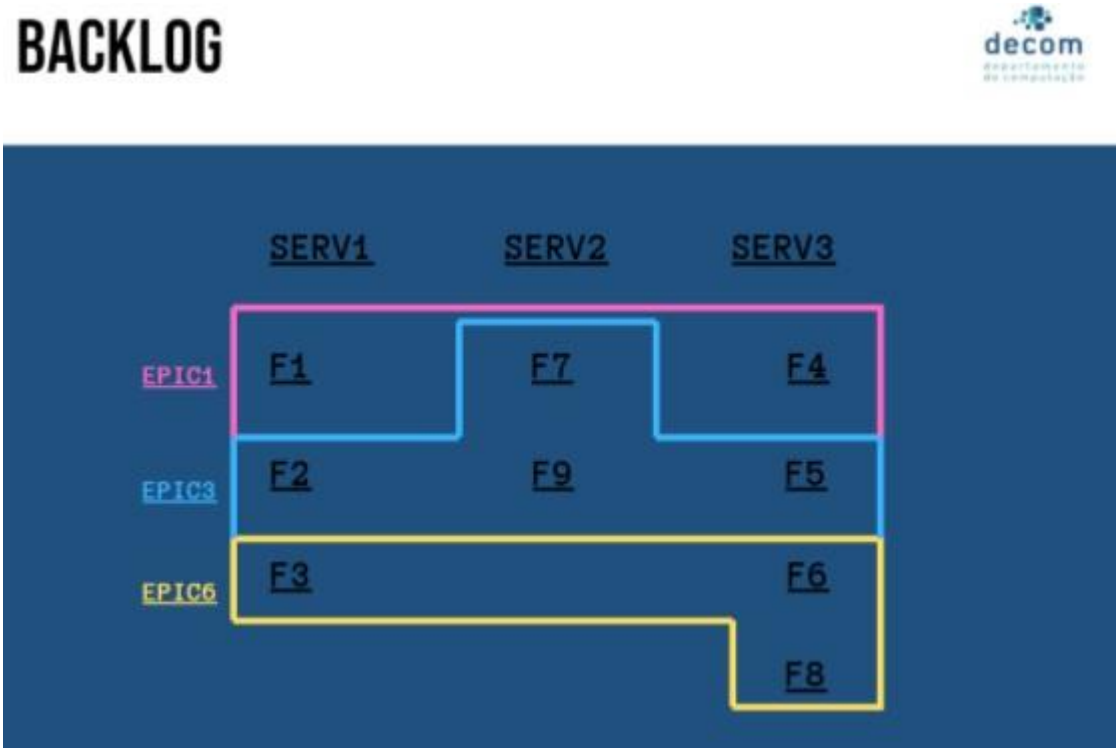


Figura 4.14 – Backlog

Ao final do documento, ao estabelecer os requisitos não funcionais do projeto, espera-se que os estudantes analisem diferentes casos em que os produtos devem atender critérios de qualidade, segurança, confiabilidade, manutenção, portabilidade ou legalidade.

O que se espera nesse trabalho é que os estudantes entendam a importância de uma especificação robusta e de qualidade e que vivenciem todas as etapas e dificuldades envolvidos no desenvolvimento de projetos de software, ganhando experiência no uso de diferentes ferramentas e no desempenho de diferentes funções existentes em um equipe profissional.

4.4 Indicadores de impacto

Os indicadores de impacto são formados por um conjunto de indicadores quantitativos que avaliam o rendimento médio, o índice de evasão e o índice de retenção dos estudantes ao longo do tempo, assim como indicadores qualitativos que avaliam a qualidade emocional e psicológica do ambiente de ensino. Nessa seção os indicadores de rendimento e evasão escolar são apresentados para analisar de forma quantitativa e quantitativa os impactos das mudanças que ocorreram ao longo dos experimentos realizados e descritos neste trabalho.

4.4.1 Impacto sobre os indicadores de rendimento e evasão escolar

No primeiro semestre de 2015, conforme ilustra a Figura 4.15, observou-se um alto nível de reprovação e baixo nível de aprovação.Notas obtidas pelos estudantes estavam em maioria abaixo damédia, conforme ilustra a Figura 4.16. Diante desse cenário, era evidente a necessidade de estudo sobre as causas dos problemas apresentados e o planejamento para uma remodelagem do programa a disciplina como um todo. No segundo semestre de 2016, a metodologia tradicional não é mais a única forma de ensinar o conteúdo da disciplina. O professor associou, em pequena escala, técnicas da metodologia ativa PBL e técnicas já utilizadas da metodologia tradicional para ministrar suas aulas. Nota-se também que a partir dessa data o desempenho dos estudantes tem uma alta significativa e permanece até os dias atuais com algumas oscilações. As primeiras versões dos trabalhos práticos (individual e em grupo) são criadas, de forma intuitiva, em conformidade com aquilo que preconiza a metodologia PjBL e, então, aplicadas aos estudantes. Os trabalhos práticos criados para complementar o aprendizado, não tinham um enunciado bem formatado, ou seja, não explicam com clareza as reais tarefas a serem realizadas, os conhecimentos que seriam exercitados, nem a forma de avaliação. Assim, os estudantes executavam os trabalhos do modo que achavam correto.

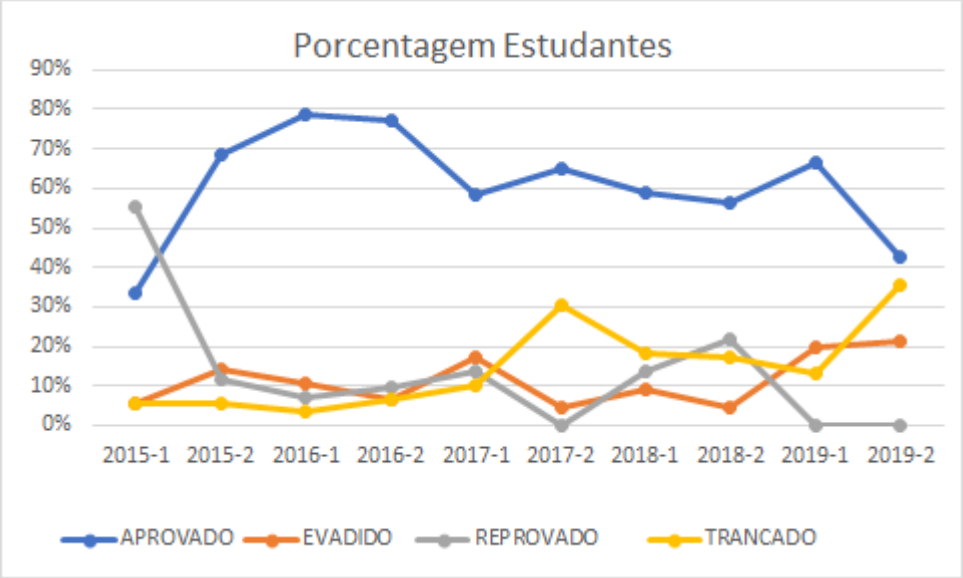


Figura 4.15 – Indicadores Academicos BCC322 de 2015 a 2019

Os estudos de caso adotados como meio de aprendizado são planejados e novos objetos de aprendizados são formulados para facilitar e promover o diálogo entre professor e estudante, buscando o engajamento deste último. Os estudos de caso são maturados de forma experimental e intuitiva, à medida que o professor percebe e analisa as respostas dos estudantes durante e ao final das práticas. Contudo, as mudanças aconteciam em um cenário onde há pouco embasamento sobre metodologias de ensino por parte do professor. No primeiro semestre de 2017, o docente teve a oportunidade de auxiliar uma startup formada por ex-estudantes da disciplina. Então,

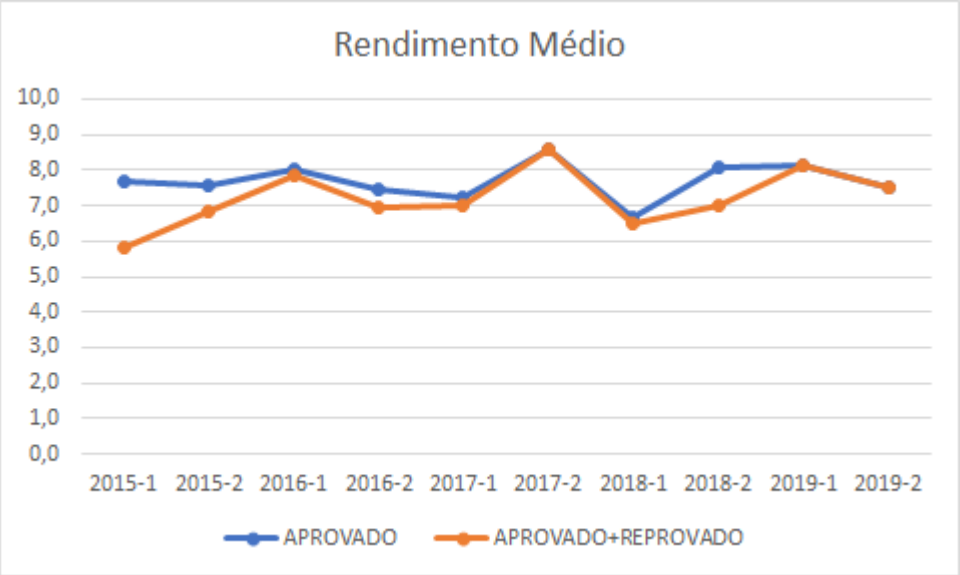


Figura 4.16 – Aproveitamento dos Alunos BCC322 de 2015 a 2019

pode absorver informações importantes para melhorar a disciplina. Ficou claro como deveria ser o processo de implantação de uma metodologia ágil e suas peculiaridades. Também ficaram claras quais são as dificuldades na transmissão do conhecimento dentro do grupo e como deveria acontecer a aplicação de todo conhecimento prático e teórico na condução de um projeto de desenvolvimento de software real. Assim, todos conhecimentos adquiridos pelo docente foram incorporados à disciplina. Desde então, pode-se notar que apesar de o conteúdo da disciplina ter se tornado mais denso e das avaliações serem mais desafiadoras, houve a manutenção dos índices de aprovação e da nota média obtida pelos estudantes em níveis satisfatórios.

De acordo com o gráfico da Figura 4.15, observou-se que no semestre de 2017/2 o alto índice de trancamento da disciplina, resultou em um alto índice de aprovação (aprovação de todos estudantes). Isso posto, pode-se concluir que somente estudantes engajados e comprometidos com sua própria aprendizagem estavam presentes em sala de aula. A maior nota média histórica pode ser entendida como mais uma evidência desse fato. Infelizmente, o professor da disciplina não sabe precisar que fato levou os estudantes a trancarem a disciplina ainda no início do semestre. O importante é notar que neste cenário, onde há poucos estudantes engajados, a transmissão de conhecimento tende a acontecer com maior sucesso. No semestre de 2018/2, houve uma alteração no programa da disciplina e os trabalhos práticos foram realizados em paralelo, uma aula da semana era destinada ao trabalho prático em grupo e na outra aula da semana era realizado o trabalho prático individual. Essa alteração foi realizada com o intuito ampliar o número de ciclos de desenvolvimento dos dois trabalhos. Neste caso, como ilustra o gráfico da Figura 4.15, que neste semestre houve o maior índice de reprovação dos estudantes se comparado aos outros semestres. Com base no questionário respondido pelos estudantes no fim do semestre 2018/2 (Anexo A), pode-se verificar que os estudantes relataram grande dificuldade ao realizar os dois

trabalhos simultaneamente, eles se sentiam confusos e sobrecarregados. Isso posto, pode-se concluir que a realização dos trabalhos em momentos distintos permite que os estudantes se tornem mais focados na realização dos trabalhos e consequentemente mais focados em seu aprendizado.

No semestre de 2019/2, seguindo o programa da disciplina, foi apresentado aos estudantes, já nas primeiras semanas, o trabalho prático individual. No final do primeiro mês e meio de aulas, três entregas do já haviam sido realizadas, referentes a três ciclos semanais de desenvolvimento. Isso posto, notou-se que muitos dos estudantes não se engajaram como era o esperado e dessa maneira não conseguiram acompanhar o ritmo das aulas. A partir desse cenário, um grande número de estudantes realizaram o trancamento da disciplina, como pode ser analisado na Figura 4.15. Este fato sugere que os estudantes podem necessitar de um tempo maior para se adaptarem ao modelo de ensino baseado no PjBL, visto que até esse primeiro momento estavam acostumados ao modelo tradicional de ensino. Por outro lado, a falta de engajamento e alto índice de trancamento podem revelar a falta de comprometimento de alguns alunos com seu aprendizado. Na ausência de estudantes pouco engajados, o professor pode dedicar atenção aos demais estudantes e o índice de reprovação naquele semestre foi nulo.

5 Considerações Finais

Este trabalho apresentou uma análise do contexto do ensino de ES em escala global e nacional com o intuito de utilizar o conhecimento adquirido como base para melhorias a serem realizadas na disciplina de ES na UFOP – nominalmente BCC322. O foco foi resumir as principais pesquisas realizadas sobre o tema, reutilizando as práticas que obtiveram bons resultados em outras universidades para remodelar o ensino de ES na UFOP, além de utilizar as opiniões dos estudante e os indicadores acadêmicos (índices de aprovação, retenção, trancamento e nota média) para avaliar o impacto das mudanças nas práticas e no programa da disciplina BCC322. Nota-se que o tema é discutido no mundo todo com objetivos análogos ao deste estudo. Assim, percebe-se que a disciplina de Engenharia de Software é enriquecida a cada pesquisa sobre o tema.

Ao longo deste trabalho, o planejamento e construção dos objetos de aprendizados para o ensino de ES exigiu da autora deste trabalho profundo conhecimento prático e teórico sobre Engenharia de Software e bom conhecimento sobre as metodologias ativas de aprendizado. Observa-se que, apesar dos indicadores de desempenho, evasão e retenção dos alunos não permitirem uma análise conclusiva sobre a melhoria do ensino, ao longo do tempo a disciplina tornou-se extremamente densa e que o rendimento médio dos estudante manteve-se oscilando levemente entorno da média história de 80%, mesmo sendo eles cada vez mais exigidos.

Como trabalhos futuros sugere-se a adaptação e avaliação da metodologia de ensino desenvolvida neste trabalho para situações em que o aprendizado deve acontecer de forma remota. Neste caso, o desafio é oferecer a vivência exigida pelos estudos de caso na ausência de encontros presenciais. Além disto, sugere-se que os indicadores quantitativos e qualitativos coletados neste trabalho continuem sendo coletados para monitoramento dos impactos deste trabalho sobre a qualidade de ensino de Engenharia de Software na UFOP. Para isso, novas entrevistas devem ser realizadas com alunos, ex-alunos e empresas que admitem alunos egressos nas funções de desenvolvimento de software.

Referências

A. M. Connor, J. Buchan, K. Petrova, Bridging the research-practice gap in requirements engineering through e_ective teaching and peer learning in: Sixth International Conference on Information Technology: New Generations, ITNG 2009, Las Vegas, Nevada, 27-29 April 2009, 2009.

A. Sandberg, L. Pareto, T. Arts, Agile collaborative research: Action principles for industryacademia collaboration, IEEE Software 28 (4)(2011).

Anastasiou, L. G. C. Metodologia de Ensino na Universidade Brasileira: elementos de uma trajetória. Campinas: Papirus, 2001

Anazifa

Beckman, K., Coulter, N., Khajenouri, S., Mead, N., Collaborations: Closing the industry– academia gap. IEEE Software 14 (6), pp. 49–57, 1997.

Borges, Hudson & Do Espírito Santo, Rafael & dos Santos, Rodrigo & Costa, Heitor & Werner, Claudia. (2011). Portal EduES 2.0: Uma Ferramenta para Apoiar a Gerência de Reutilização no Domínio de Educação em Engenharia de Software.

Conn, R. Developing Software Engineers at the C-130J Software Factory. IEEE Software, Los Alamitos, v. 19, n. 5, p. 25-29, Sep. 2002.

De Lucena, V.F.; Brito, A.; Gohner, P. A Germany-Brazil Experience Report on Teaching Software Engineering for Electrical Engineering Undergraduate Students. In: CONFERENCE ON SE EDUCATION & TRAINING – CSEET '06, 19., 2006, Turtle Bay. Proceedings … Washington: IEEE Computer Society, 2006. p. 69-76.

Dijkstra, E. W.; DAHL, O. J.; HOARE, C. A. R. Structured Programming. Londres: Academic Press, 1972.

Fagan, M. E. Advances in Software Inspections. IEEE Trans. on Software Eng, v. SE-12, n. 7, 1986, p. 744-751.

Feitosa, Sônia Couto Souza, 1999. Método Paulo Freire: princípios e práticas de uma concepção popular de educação. São Paulo: Feusp (Dissertação de Mestrado).

Feletti, G., Inquiry based and problem based learning: How similar are these approaches to nursing and medical education ? Higher Education Research and Development, 12, 2, 143-156, (

Freire, P. Pedagogia do Oprimido. São Paulo: Paz e Terra, 2006.

Freire, Paulo. Pedagogia da Autonomia: saberes necessários à pratica educativa. 34a edição. São Paulo: Paz e Terra, 2006.

Freire, Paulo. Pedagogia da autonomia: saberes necessários à prática educativa. 36ª ed. São Paulo: Paz e Terra, 2007.

Gadotti, M. Saber aprender: um olhar sobre Paulo Freire e as perspectivas atuais da educação. In: LINHARES, C. & TRINDADE, M. N. (Org.) Compartilhando o mundo com Paulo Freire. São Paulo: Cortez, 2003.

Garg, Kirti & Varma, Vasudeva. (2008). Software Engineering Education in India: Issues and Challenges. Software Engineering Education Conference, Proceedings. 110-117.10.1109/CSEET.2008.36.

Gemignani, Elizabeth Yu Me Yut. Formação de Professores e Metodologias Ativas de EnsinoAprendizagem: Ensinar Para a Compreensão. Revista Fronteira das Educação, Recife / PE, jan.2012.

Gibbs, W., Software's chronic crisis. Scientific American 271 3 (1994).

Gil, Antonio Carlos. Didática do ensino superior. São Paulo: Atlas, 2008 VEIGA, I. P. A. Técnicas de ensino: novos tempos, novas configurações. Papirus Editora, 2006. FREIRE, P. [1996] Pedagogia da autonomia: saberes necessários à prática educativa. 36. ed. São Paulo: Paz e Terra, 2007.

H. D. Rombach, R. Achatz, Research collaborations between academia and industry, in: International Conference on Software Engineering, ISCE 2007, Workshop on the Future of Software Engineering, FOSE 2007, May 23-25, 2007, Minneapolis, MN, USA, 2007.

L. Damm, L. Lundberg, C. Wohlin, Faults-slip-through - a concept for measuring the e_ciency of the test process, Software Process: Improvement and Practice 11 (1) (2006)

Libâneo, J. C. Democratização da escola pública: a pedagogia crítico-social dos conteúdos. São Paulo: Loyola, 1990.

Lopes, Antonia Osima. Aula expositiva: superando o tradicional. In: VEIGA, Ilma Passos Alencastro (Org.). Técnicas de ensino: por que não? 12. ed. Campinas: Papirus, 2001. p. 35-48. 158 p.

M. K. Noordin, A. N. M. Nasir, D. F. ALI and M. S. Nordin, "Problem-Based Learning (PBL) and Project-Based Learning (PjBL) in engineering education: a comparison," in Proceedings of the IETEC'11 Conference, Kuala Lumpur, Malaysia, 2011. Lowenthal, Jeffrey N. (2006, March). Project-Based Learning and New Venture Creation. Paper presented at the NCIIA 10th Annual Meeting, Oregon Museum of Science and Industry, Oregon.

M. Schots, R. Santos, A. Mendonca, C. Werner. Elaboração de um survey para a caracterização do cenário de educação em engenharia de software no Brasil. In Anais do FEES09 - Fórum de Educação em Engenharia de Software, Fortaleza, Outubro 9, 2009.

Meyer, B. Software Engineering in the Academy. Computer, Los Alamitos, v. 34, n. 5, pp. 28-35, May 2001.

Mitre, Sandra Minardi et al . Metodologias ativas de ensino-aprendizagem na formação profissional em saúde: debates atuais. Ciênc. saúde coletiva, Rio de Janeiro , v. 13, supl. 2, p. 2133-2144, Dec. 2008 .

Muzetti, Igor. Desenvolvendo software inovador em universidades públicas: Adaptando processos ágeis para a realidade de laboratórios de pesquisa e desenvolvimento. Mestrado, ICEB, Instituto de Ciências Exatas e Biológicas, UFOP, 2014..

Nérice, I. G. Didática geral dinâmica. 10 ed., São Paulo: Atlas, 1987.

Oktaba, H. Guia de Implementacao MoProSoft V 1.3.2. [S.l.], 2006. CARVALHO, M. de. Construindo o saber: técnicas de metodologia científica. [S.l.]: Papirus Editora, 1989. ISBN 9788530800710. 31

P. Runeson, S. Min•or, The 4+1 view model of industry-academia collaboration, in: Proceedings of the 2014 ACM International Workshop on Long-term Industrial Collaboration on Software Engineering, Vasteras, Sweden, September 16, 2014, 2014.

Pereira, Igor Muzetti. Desenvolvendo software inovador em universidades públicas: Adaptando processos ágeis para a realidade de laboratórios de pesquisa e desenvolvimento. 2014. 63 f. Mestrado (Ciência da Computação) – Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Ouro Preto, Ouro Preto. 2014

R. P. Santos, P. S. M. Santos, C. M. L. Werner e G. H. Travassos. 2008. Utilizando Experimentação para Apoiar a Pesquisa em Educação em Engenharia de Software no Brasil. In Anais do Fórum de Educação em Engenharia de Software (FEES), São Paulo, Brasil. Monografias em Ciência da Computação. Departamento de Informática PUC-Rio, pp. 55-64.

Rampazzo, L. Metodologia científica. [S.l.]: Edições Loyola, 2005. ISBN 9788515024988. GAMMA, E, et al. Design Patterns-Elements of Reusable ObjectOriented Software – 1a Edição, Addison Wesley, 1998.

Ronca, Antônio Carlos Caruso; ESCOBAR, Virgínia Ferreira. Aula expositiva. In: Técnicas Pedagógicas: domesticação ou desafio à participação. Petrópolis: Vozes, 1988. 5ª edição.

Santos, R.; Santos, P. S.; Werner, C.; Travassos, G. (2008) “Uma Estratégia para Apoiar a Pesquisa em Educação em Engenharia de Software no Brasil”, In: Proceedings of the 5th Experimental Software Engineering Latin American Workshop (ESELAW’08), Salvador, Brasil, 1- 10.

Schwaber, Ken. Sutherlan, Jeff. Guia do Scrum®: Um guia definitivo para o Scrum: As regras do jogo. Disponível em http://www.scrumguides.org/docs/scrumguide/v2016/2016-ScrumGuidePortuguese-Brazilian.pdf, acesso em 04/06/2019.

Serrano, R.M.S.M. (2008). Conceitos de extensão universitária: um diálogo com Paulo Freire. Extelar: Grupo de Pesquisa em Extensão Popular. Acesso em 23.jun.2011, Acessível em: http://www.prac.ufpb.br/copac/extelar/atividades/discussao/artigos/conceitos de extensao universitaria.pdf

Sigsoft, disponível em https://event.on24.com/eventRegistration/EventLobbyServlet?target=reg20.jsp&referrer=http%3A%2F%2Fwww.sigsoft.org%2Fresources%2Fwebinars.html&eventid=1797723&sessionid=1&key= B74CAD05FEF193B794A8E661185A79DF&regTag=&sourcepage=register, Acessado em: junho, 2019.

Silva, F. G., Hoentsch, C. P., Silva, L. 2009. Uma Análise das Metodologias Ágeis FDD e Scrum sob a Perspectiva do Modelo de Qualidade MPS.BR. Scientia Plena, v. 5, n. 12.

Silva, Mirian Jesus,. Abordagens Tradicional e Ativa: Uma Análise da Prática a Partir da Vivência no Estágio Supervisionado em Docência. Paraná: XIII Educere, 2017.

Sommerville, I. Engenharia de software. 8.ed. São Paulo: Pearson Addison-Wesley, 2007. PRESSMAM, R. Engenharia de software. 6. ed. São Paulo: McGraw-Hill, 2006 CHILE. Ministerio de Educación,2015.

Sommerville, Ian. Engenharia de Software. 9. ed. São Paulo: Pearson Prentice Hall, 2011. SOARES, MICHEL DOS SANTOS. Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software. 2004. Disponível em: http://www.dcc.ufla.br/infocomp/artigos/v3.2/art02.pdf. Acesso em: 17/06/2019.

Sommerville, Ian.Engenharia de Software / Ian Sommerville ; tradução Ivan Bosnic e Kalinka G. de O. Gonçalves ; revisão técnica Kechi Hirama. — 9. ed. — São Paulo : Pearson Prentice Hall, 2011.

Souza, C.R.; Silva, N.R. da; Queiroz, R.G. de; Nascimento, B.R.O. (2013). "ITILKnow: um objeto de aprendizagem baseado em jogos para o ensino de ITIL", In: VIII LACLO 2013, Valdivia, Chile. Cardoso, R., Santos, O., and Gatti, D. C. (2015). "Revisão Sistemática de Objetos de Aprendizagem para o Ensino de Computação". Conferencias LACLO, 6(1), 389.

T. Gorschek, P. Garre, S. Larsson, C. Wohlin, A model for technology transfer in practice, IEEE Software 23 (6) (2006) 88{95.

Teixeira, M.C. Metodologia do ensino superior. Guarapuava: Unicentro, 2015.

V. Garousi, K. Petersen, and B. Özkan. 2016. Challenges and Best Practices in Industry-academia Collaborations in Software Engineering. Information and Software Technology 79, C (Nov. 2016), 106–127. DOI:https://doi.org/10.1016/j. infsof.2016.07.006.

V.Flores, J. Gomez (2017). Applying Active Methodologies for Teaching Software Engineering in Computer Engineering. IEEE Revista Iberoamericana de Tecnologias del Aprendizaje. 12. 182-190. 10.1109/RITA.2017.2778358.

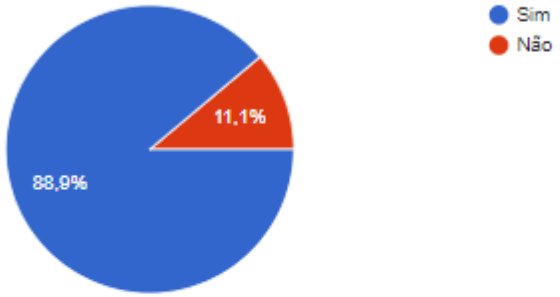
Woods, DR, Issues in implementation in an otherwise conventional programme. Em Boud, D.& Feletti, GI (eds.) The challenge of problem-based learning, 2nd ed, Kogan Page, London. 173-180, (1997).

Apêndices

APÊNDICE A – Questionário realizado no final do período do segundo semestre de 2018

Todos os tópicos propostos pela ementa da matéria foram dados no decorrer do semestre?

9 respostas



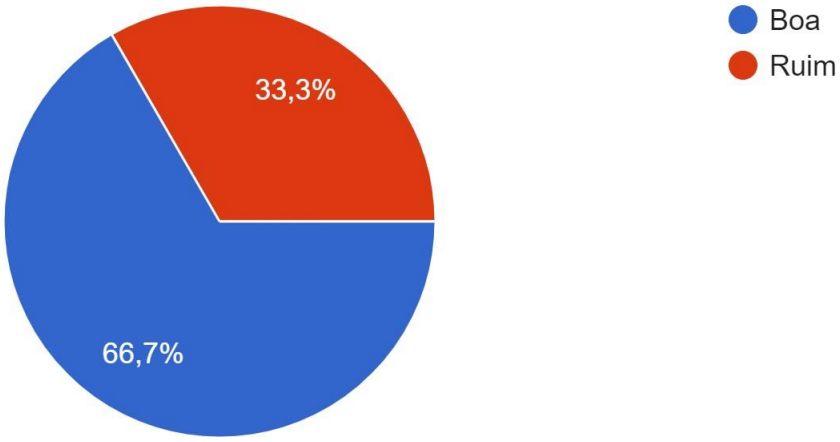
Se a resposta acima foi "não". O que faltou?

1 resposta

Gestão e configuração de software (que é isso?)

Como foi sua experiência com a metodologia de listas usadas ?

9 respostas



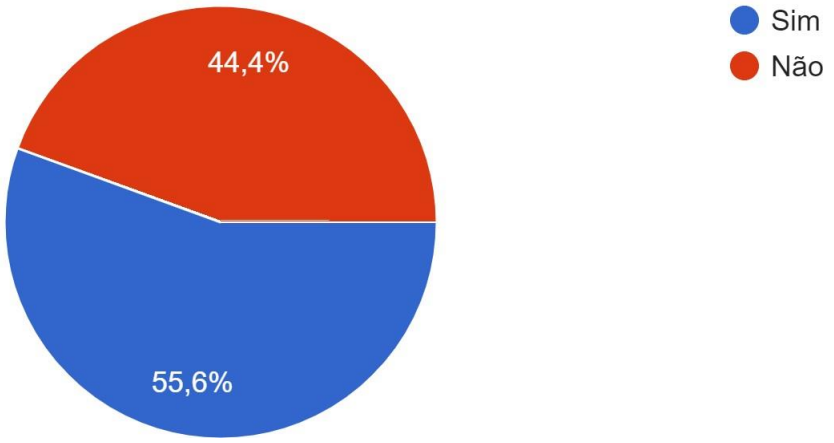
Explique em poucas palavras, como foi a sua experiência com as listas dadas e o que poderia ser feito para melhorá-las, caso você não tenha tido uma experiência boa.

9 respostas

Acho que algumas poderiam ser pra implementar alguns exemplos do livro de Qt do Sommerfield, isso faria as pessoas entenderem um pouco enquanto lêem e digitam os códigos do livro.
Não gostei da forma de correção das listas
Ser dadas ao final, quando o aluno ja tiver um entendimento do conteúdo uma vez que ele chega sem uma base boa pra esta disciplina como foi meu caso.
A experiencia foi boa pois o aluno é "forçado" a entrar no mundo do QT. Um ponto ruim foi apenas pelo fato de não conseguir realizar com êxito o que era proposto. Várias eram as soluçoes possíveis para um mesmo problema.
Eu tive uma boa experiência
As listas foram de grande importancia, deu pra entender conceitos bastante necessarios, os quais antes eu nao sabia tão been.
Passar não somente listas de correção de código, mas passar também listas de "construção" do código.
As listas deveriam ter questões de desenvolvimento próprio, ao invés de puramente debug de codigo
Não consegui realizar todas as listas, pois deixei por última hora e também, porque tenho dificuldades com o Qt creator.

Você conseguiu realizar todas as listas dadas?

9 respostas



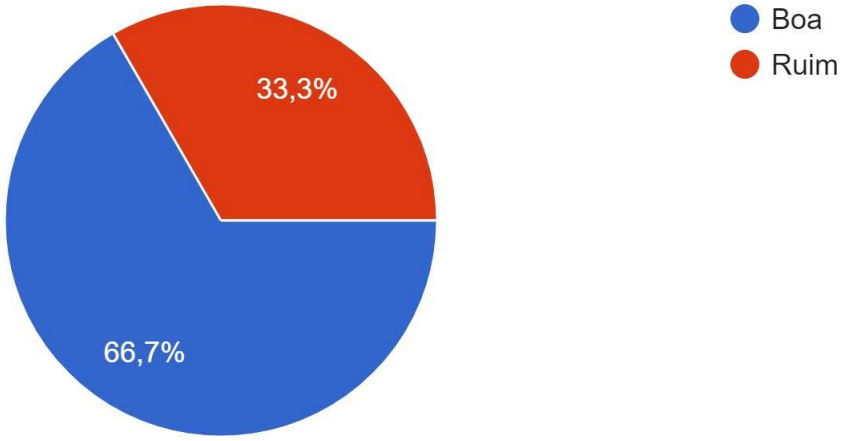
Quais foram as maiores dificuldades encontradas durante o semestre?

9 respostas

Ter computador pra levar pra aula (o professor me emprestou um) e me organizar pra conseguir entregar dois trabalhos por semana
Reta final muitos trabalhos ,poderia distribuir os trabalhos práticos ao longo do curso
Tempo para realização dos trabalhos dados em paralelo
A demanda da disciplina, a partir da segunda parte do curso (após a p1) tornou-se muito exaustiva, pois contava com resoluçoes de partes do tpIndividual e do tp em grupo na mesma semana. Para alunos que possuem atividades além das próprias disciplinas, conciliar horários para resolver os tps pode ser um desafio.
Conciliar o trabalho individual com o em grupo
Passar o teorico para o pratico, exemplo, enterder handle body é diferente de implementar o mesmo.
Acompanhar o conteúdo das aulas. Muitas coisas novas foram passadas em apenas uma aula.
O tempo para realizar as atividades era apertado, dado o contexto do semestre.
Conciliar os trabalhos desta disciplina (tanto a inidividual, quanto a em grupo) juntamente com outras atividades avaliativas envolvendo outras cadeiras/disciplinas.

Qual foi sua experiência ao usar o QTest?

9 respostas



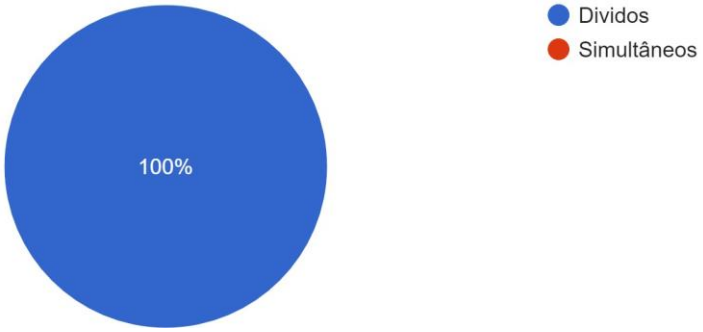
Os trabalhos dados, em grupo e indivual, foram satisfatórios? Vocês conseguiram aprender com esses trabalhos?

9 respostas

Sim, bastante.
Sim
Muito, genial a idea dos dois trabalhos aprendi em um semestre mais que no curso inteiro.
Sim.
Foram. Consegui.
Sim! Deu pra aprender muito
O trabalho em grupo foi o mais satisfatório, pois, pude aprender mais sobre o QT.
A experiência foi satisfatória e foi possível descobrir novos recursos.
De certa forma, aprendi sim. Mas não foram satisfatórios, pois estavam incompletos.

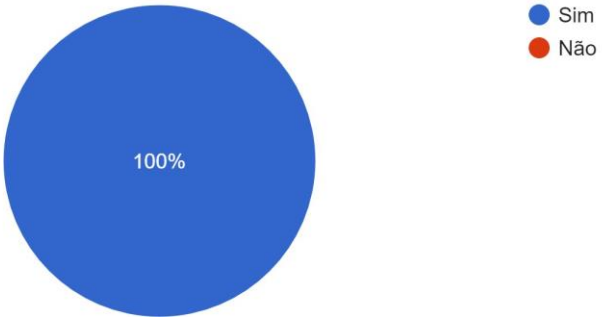
Os dois trabalhos deveriam ser realizados simultaneamente ou divididos no semetre?

9 respostas



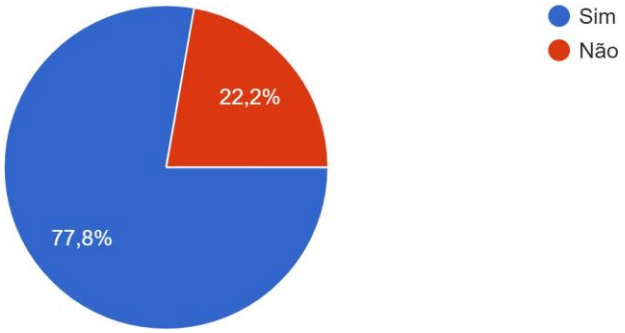
Os metódos usados para avaliação dos alunos foram satisfatórios?

9 respostas



O trabalho individual foi realizado com sucesso?

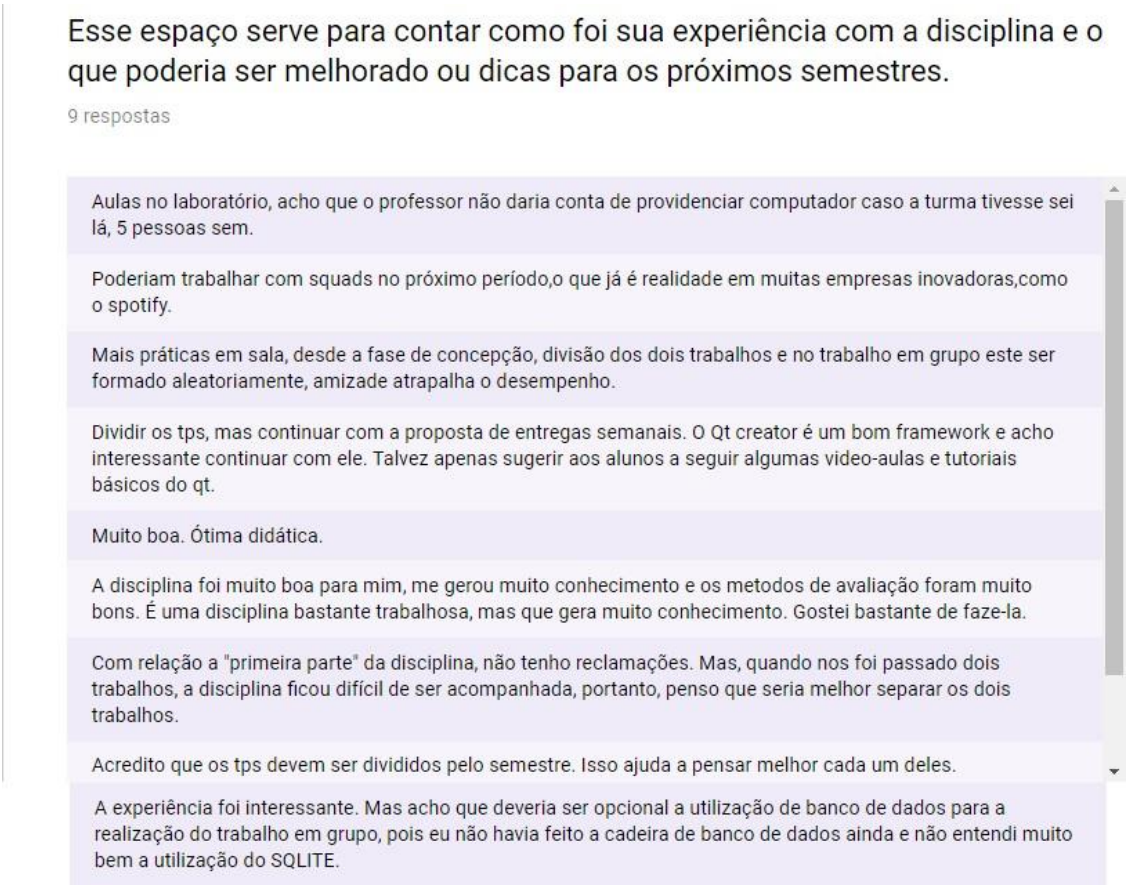
9 respostas



Quais foram as dificuldades encontradas ao realizar o trabalho individual?

9 respostas

Pensar em testes e em formas de fazer os patterns funcionar pra situações específicas como o fato de Flow ter que ser abstrato
Pouca documentação dos patterns na internet
Nenhuma, se houve dificuldade da parte de alguem é porque faltou dedicação e força de vontade
Dificuldades relacionadas com disciplinas anteriores (Programação Orientada a Objetos).
Composite
Passar o teorico para o pratico
Compreender o que deveria ser feito durante a semana.
Tempo para implementar cada alteração de forma que os testes continuem funcionando
Tive dificuldades na implementação, com relação ao design pattern: handle/body. Isso porque, não consegui implementar os destrutores e o construtor via cópia de Modelo. Além disso, não consegui modificar a sobrecarga do operador =.


Esse espaço serve para contar como foi sua experiência com a disciplina e o que poderia ser melhorado ou dicas para os próximos semestres.
9 respostas
Aulas no laboratório, acho que o professor não daria conta de providenciar computador caso a turma tivesse sei lá, 5 pessoas sem.
Poderiam trabalhar com squads no próximo período,o que já é realidade em muitas empresas inovadoras,como o spotify.
Mais práticas em sala, desde a fase de concepção, divisão dos dois trabalhos e no trabalho em grupo este ser formado aleatoriamente, amizade atrapalha o desempenho.
Dividir os tps, mas continuar com a proposta de entregas semanais. O Qt creator é um bom framework e acho interessante continuar com ele. Talvez apenas sugerir aos alunos a seguir algumas video-aulas e tutoriais básicos do qt.
Muito boa. Ótima didática.
A disciplina foi muito boa para mim, me gerou muito conhecimento e os metodos de avaliação foram muito bons. É uma disciplina bastante trabalhosa, mas que gera muito conhecimento. Gostei bastante de faze-la.
Com relação a "primeira parte" da disciplina, não tenho reclamações. Mas, quando nos foi passado dois trabalhos, a disciplina ficou difícil de ser acompanhada, portanto, penso que seria melhor separar os dois trabalhos.
Acredito que os tps devem ser divididos pelo semestre. Isso ajuda a pensar melhor cada um deles.
A experiência foi interessante. Mas acho que deveria ser opcional a utilização de banco de dados para a realização do trabalho em grupo, pois eu não havia feito a cadeira de banco de dados ainda e não entendi muito bem a utilização do SQLITE.