Compilador Web: uma Experiência Interdisciplinar entre as Disciplinas de Engenharia de Software e Compiladores

Renata Correa Pimentel[1], Andréia Damásio de Leles[1], Luciana A. M. Zaina[2]

[1]Faculdade de Engenharia de Sorocaba (FACENS) – Rodovia Senador José Ermírio de Moraes, 1425, Castelinho km 1,5 - Alto da Boa Vista - Sorocaba - São Paulo - Brasil

[2]Universidade Federal do São Carlos (UFSCar) – campus Sorocaba
Rdv. João Leme dos Santos (SP-264), Km 110 - Sorocaba - São Paulo - Brasil

renata,andreia{@facens.br},lzaina@ufscar.br

Abstract. *The interdisciplinarity has been a topic widely discussed in the teaching of Computing. This article presents an interdisciplinary experience carried out in a Computer Engineering course concerning the subjects of Software Engineering and Compiler. The integrated project aimed to building a web compiler whose primary focus was to teach basic programming. The analysis of the tools was done by a group of students which studied basic programming.*

Resumo. *A interdisciplinaridade tem sido um tema amplamente discutido no ensino da Computação. Este artigo apresenta uma experiência interdisciplinar entre as disciplinas de Engenharia de Software e Compiladores realizada em um curso de Engenharia da Computação. O projeto integrado envolvia a construção de um compilador web cujo foco principal era ensinar programação básica. A análise dos ambientes desenvolvidos pelos alunos foi realizada por um grupo de alunos que estudavam a disciplina básica de programação.*

1. Introdução

A interdisciplinaridade no ensino de graduação é uma prática que contribui diretamente para o desenvolvimento de competências e habilidades a partir da visão integradora e do constante relacionamento entre a teoria e a prática. A adoção da interdisciplinaridade permite que o aluno trabalhe não somente com especificidades das disciplinas, mas também que abra seu espectro de visão sobre as múltiplas aplicações que um determinado conteúdo possui.

Além das questões relacionadas aos conteúdos estudados, a interdisciplinaridade também funciona como um motivador ao aluno, pois permite que ele trabalhe focado em um único tema e também gera contribuição para diversas disciplinas. Isto evita a redundância entre disciplinas e a realização de diversos projetos durante um período letivo, sobrecarregando o aluno com tarefas de entendimentos de cenários diferenciados ao mesmo tempo. As diretrizes curriculares de recomendação de projetos pedagógicos para a área de Computação [Diretrizes Curriculares SBC 2005] apontam para a necessidade de oportunizar ao aluno meios de vivenciar a o relacionamento ntre os

diversos conhecimentos que compõe sua formação. Observando esta necessidade, as docentes responsáveis pelas disciplinas de Engenharia de Software e Compiladores na Faculdade de Engenharia de Sorocaba (FACENS) se propuseram a realizar um projeto interdisciplinar envolvendo os dois componentes curriculares no período letivo de 2011. Foi proposto um projeto onde os alunos deveriam desenvolver uma ferramenta, utilizando as disciplinas de Engenharia de Software, que permitisse o desenvolvimento de códigos em uma dada linguagem e que implementasse as técnicas de compiladores.

Dentre as motivações para a realização do projeto pode-se destacar o fato das docentes desejarem oferecer aos alunos um cenário diferenciado dos comumente utilizados para a aplicação das diretrizes e metodologias da Engenharia de Software. Outra questão observada pelas docentes foram os comentários dos alunos em relação às dificuldades para construção de um compilador. Neste sentido, partiu-se da premissa que utilizar as práticas de Engenharia de Software, poderia minimizar a complexidade do desenvolvimento do compilador e obter melhores resultados com o projeto. Outro ponto importante é que os alunos teriam que gerenciar requisitos obrigatórios para a aplicação. A docente de Compiladores faria o papel da cliente determinando as características que eram mandatórias para a aplicação.

O objetivo deste artigo é apresentar a experiência do curso de Engenharia da Computação, da FACENS – Sorocaba, que integrou as disciplinas de Engenharia de Software e de Compiladores, reportando os resultados obtidos. O restante do artigo está organizado da seguinte forma: seção 2 apresenta o planejamento da experiência interdisciplinar, a execução da experiência incluindo os testes de aceitação dos ambientes desenvolvidos e os benefícios e dificuldades da experiência são apresentadas na seção 3, a seção 4 apresenta os resultados da experiência e a seção 5 realiza as conclusões do artigo.

2. Planejamento da Experiência

Para iniciar o planejamento da disciplina as docentes envolvidas pesquisaram alguns trabalhos correlatos. Dentre eles pode-se destacar o apresentado por Brereton, Turner e Kaur (2009) que apresenta uma revisão sistemática de trabalhos que utilizaram programação em par como apoio ao ensino de linguagem de programação. Schocken, Nisan, Michal e Armoni (2009), apresentaram uma proposta de uma disciplina que abordasse os conceitos de hardware, software e compiladores em conjunto. O objetivo principal era possibilitar ao aluno uma visão integrada dos conceitos que envolvem essas três áreas através do desenvolvimento de projetos. Yamamoto et al (2005) apresentaram experiências com projetos interdisciplinares em um curso de Ciência da Computação com diferentes projetos pedagógicos. More, Kumar e Renumol (2011) apresentam uma ferramenta para ensino de programação em disciplinas introdutórias.

Inicialmente, as docentes fizeram reuniões buscando definir os objetivos e metas do projeto integrado. Definiu-se que os alunos deveriam desenvolver uma ferramenta que permitisse aos usuários codificar um programa em uma dada linguagem, de maneira que a ferramenta executasse todas as etapas referentes à compilação de código (análise léxica, sintática e geração de código). Ao elaborar de forma conjunta os planos de ensino das disciplinas envolvidas, analisou-se que a ferramenta poderia ter características de sistemas Web, abrangendo assim conteúdos abordados na disciplina de Engenharia de

Software, como conceitos de arquitetura de software e padrões de projeto. A partir da visão interdisciplinar, foi construída uma proposta criativa e desafiadora de desenvolver um compilador didático que rodasse em ambiente *desktop* e Web.

Além da interdisciplinaridade realizada pelas disciplinas que estão num mesmo período letivo, as disciplinas de Arquitetura e Organização de Computadores e Algoritmos também foram envolvidas no projeto do compilador Web, demonstrando aos alunos a interligação entre conceitos e trabalhos já desenvolvidos anteriormente no curso. Um computador didático foi desenvolvido através de um projeto de iniciação científica, orientado pelo docente da disciplina de Arquitetura de Computadores [Garcia e Legaspe, 2011]. Este computador tem por finalidade facilitar o ensino-aprendizagem da disciplina de Arquitetura, que é ministrada no semestre anterior as disciplinas envolvidas no projeto interdisciplinar. A grande contribuição para o projeto interdisciplinar foi a possibilidade dos alunos executarem e testarem as ferramentas desenvolvidas no computador didático .A característica didática do compilador, fez com os conhecimentos adquiridos na disciplina de Algoritmos fossem também resgatados e utilizados durante o projeto.

A justificativa para o desenvolvimento do compilador que pudesse ser executado em duas plataformas (*Desktop* e Web), não se baseou apenas no fato do projeto utilizar conceitos importantes da área de Engenharia de Software, mas também que ele se sustentasse como um produto. Sendo assim, vislumbrou-se dois cenários de utilização para o compilador Web: (1) apoio a aprendizagem eletrônica; (2) apoio ao desenvolvimento de software com programadores distribuídos em diferentes máquinas e/ou locais geográficos distintos. Em ambos os cenários, possibilitar a programação aos pares e armazenar as versões de código em base dados, foram definidas como características interessantes a serem desenvolvidas no projeto.

Durante o planejamento do projeto as docentes definiram requisitos funcionais e não funcionais que deveriam ser implementados por todas as equipes. Ou seja, havia um conjunto de requisitos que eram comuns a todos. Porém, cada equipe também deveria definir alguns requisitos que seriam o diferencial do seu produto final. Do ponto de vista da linguagem a ser reconhecida pelo compilador, cada equipe foi direcionada a criar uma linguagem original e nativa, de fácil entendimento, mais próxima possível da linguagem natural e pseudocódigo. A docente da disciplina de Compiladores faria o papel do cliente determinando características e necessidades do produto final.

Outro ponto definido pelas docentes foi que as equipes deveriam adotar o ciclo de desenvolvimento de software SCRUM e a disciplina proposta pelo XP - Extreme Programming para o desenvolvimento do projeto [Pressman, 2011 e Beck, 2004]. Cada fase das etapas de análise e síntese deveriam ser desenvolvidas através das iterações ou *Sprints*, conforme sugere o SCRUM. As fases, entregas e pacotes de trabalho, foram definidas, considerando o SCRUM: *Pré-game*, *Game* e *Pós-game*. Na fase de *Pré-game*, determinou-se as entregas de gerência de projetos, o protótipo funcional e a lista de requisitos ou *Product Backlog*. No plano de projeto, as equipes deveriam especificar marcos por *Sprint* e por Fase. Cada marco seria comum para avaliações e validações das duas disciplinas. No final de cada mês, um *Sprint* ou iteração deveria ser finalizado, tendo como marco, a entrega funcional de uma parte do compilador. Na disciplina de Compiladores, a qualidade do compilador era o foco da avaliação. Já na disciplina de

Engenharia de Software, o alvo era o cumprimento das entregas do projeto e a conformidade com os requisitos. No final de cada fase, as equipes também deveriam realizar entregas cuja avaliação destas comporia a nota bimestral das duas disciplinas.

Por se tratar de um projeto com requisitos dinâmicos e evolutivos, adotou-se a abordagem iterativa. Embora o SCRUM não recomende a utilização dos fundamentos da engenharia de requisitos, modelos da *Unified Modeling Language*[1] (UML®) e as diretrizes do *Project Management Institute*[2] (PMI®) em suas fases, estes foram utilizados durante o ciclo de vida de desenvolvimento. A linguagem de programação que os alunos deveriam usar era o Java[3], por ser *open source*, mas principalmente, devido as suas características de portabilidade, fundamentais a um projeto de compilador.

As disciplinas planejaram critérios de avaliação independentes na teoria e totalmente vinculados na prática. O vínculo entre as disciplinas que influenciariam as notas dos alunos correspondiam 30% do total da nota de laboratório. Na disciplina de Engenharia de Software as entregas de cada *Sprint* envolvia documentação, modelos e a liberação de uma parte do compilador, como por exemplo: análise léxica, que correspondia exatamente o que havia sido solicitado na disciplina de Compiladores. Neste sentido, se a equipe não realizasse a entrega, perderia nota nas duas disciplinas.

3. A Experiência Prática

3.1 Execução da Interdisciplinaridade

No início do ano letivo de 2011, em fevereiro, as atividades interdisciplinares foram incorporadas às atividades realizadas pelas docentes. Porém, a execução do projeto teve início em 04 de abril e término em 18 de novembro, com duração total de 7 meses. Durante os dois meses iniciais, fevereiro e março, as duas disciplinas envolvidas na atividade interdisciplinar, introduziram em suas aulas fundamentos que seriam essenciais para o início da execução do projeto. Na disciplina de Compiladores, a teoria sobre análise léxica foi explanada aos alunos e na disciplina de Engenharia de Software, os processos de desenvolvimento de software, tradicionais e ágeis, e a Engenharia de Requisitos foram apresentados.

Durante as aulas de laboratório, os alunos foram divididos em 14 grupos de 3 integrantes. Para a disciplina de Engenharia de Software cada grupo era uma equipe com papéis e atividades bem definidas. Cada equipe tinha um gerente de projetos, um engenheiro de software e um programador. A primeira tarefa de laboratório da disciplina de Engenharia de Software foi escolher o processo de desenvolvimento de software mais adequado. Todas as equipes optaram pelo SCRUM, pois acreditaram que para o cenário em questão esta seria mais efetiva. As características iterativa e incremental foram as principais justificativas para desenvolver projetos de software em que os requisitos são desconhecidos e evolutivos como foi o caso do Compilador. Nas aulas de Compiladores, os alunos evoluíram durante o período letivo conhecendo as técnicas relativas à análise léxica, sintática e semântica, além dos procedimentos de geração de código [Aho et al.,

[1] http://www.uml.org/
[2] http://www.pmi.org/
[3] http://www.java.com/pt_BR/

2008]. Tais conhecimentos foram subsidiando as equipes para que tivessem condições de desenvolver o produto.

Após a definição do processo de desenvolvimento, as atividades de gerência de projetos com foco no PMI tiveram início e os documentos Termo de Abertura do Projeto, Declaração de Escopo e a *Work Breakdown Structure* (WBS) foram desenvolvidos [PMI, 2008]. Na WBS foram esquematizadas as entregas, de forma padrão para todos os *Sprints*. A partir dos requisitos, casos de uso foram descritos e os modelos da UML foram gerados, testes unitários e de integração foram efetuados, gerando o produto compilador de forma iterativa e incremental.

Com base nos processos de Engenharia de Requisitos, os alunos construíram a lista *Product Backlog* descrevendo e priorizando os requisitos. Durante o processo de descoberta de requisitos, as aulas de Compiladores foram primordiais. Os alunos assimilavam os conceitos e depois, em equipe, durante as aulas de Engenharia de Software, através do processo e das ferramentas de Elicitação, geravam os requisitos do software. No processo de análise e negociação de requisitos, as equipes classificavam os requisitos em funcionais e não-funcionais e os categorizaram como sendo essenciais, importantes e desejáveis. Como exemplos de requisitos essenciais especificados pode-se destacar "Gerar Análise Léxica" (requisito funcional) e "Rodar em ambiente Web e *desktop*" (requisito não-funcional de portabilidade). Na categoria importante e desejável, tem-se respectivamente: "Identificar Erros de Sintaxe do Usuário" e "Endentar Código".

Para a etapa de validação dos requisitos, cada equipe teve que desenvolver o protótipo funcional do compilador para ambiente Web e *desktop*, o qual foi apresentado para as professoras das disciplinas para avaliação. Ainda na fase de *Pré-Game*, o cronograma do projeto, orçamentação e definição dos riscos foram realizados finalizando o plano do projeto. Na fase de *Pré-Game*, os alunos apresentaram os documentos de projeto e o protótipo funcional.

A fase de *Game* utilizou o planejamento e a execução dos *Sprint*, considerando a duração de 1 mês para cada um, trabalhando todos da equipe, em média, 10 horas por semana, totalizando 7 *Sprints* no projeto todo. Na reunião de início de *Sprint*, a equipe definiu quais requisitos seriam desenvolvidos, tendo como premissa as etapas e os resultados intermediários de um compilador. Assim, cada etapa de compilação foi especificada como requisito funcional essencial do *Sprint* e os respectivos requisitos importantes e desejáveis relacionados. Portanto, no primeiro *Sprint* se realizou a análise léxica onde definiu-se uma expressão regular para cada *token* da linguagem desenvolvida. Após essa etapa, foi desenvolvida uma documentação para formalização da gramática da linguagem estabelecendo-se todas as regras de produção para realizar o reconhecimento sintático. Nessa documentação, preocupou-se em definir regras que não produzissem linguagens que possuíssem ambiguidades. Em seguida, orientado pela análise sintática, realizou-se a análise semântica onde preocupou-se, por exemplo, com atribuições de tipos distintos. Para finalizar com as três análises realizadas, gerou-se o código de máquina para o computador didático. Os três últimos *Sprint* foram dedicados ao desenvolvimento das características Web, programação aos pares e geração de versionamento do código com armazenamento em base de dados. A fase de *Game* teve dois grandes marcos: (1) entrega do compilador *desktop*; (2) entrega do compilador Web.

A fase de *Pós-Game* teve como principais entregas o teste de aceitação com o usuário, testes com o compilador didático e as documentações de instalação e o manual do usuário. Para finalizar e encerrar o projeto, foi solicitado o documento de lições aprendidas para relatar experiências, destacando, pontos fracos e fortes do projeto.

3.2 Teste de Aceitação do Ambiente

Para realização do teste de aceitação, alunos do 1º ano de Engenharia da Computação foram convidados a utilizar e testar as ferramentas dos compiladores desenvolvidos, tanto em ambiente Web como *desktop*. Cada aluno-usuário, tinha apoio do manual do usuário desenvolvido pelas equipes. Para que fosse possível testar as características Web e programação aos pares, foram necessários 2 laboratórios de informática com rede *wireless*: em um deles ficavam os usuários testando os ambientes e no outro, ficavam as equipes observando os testes. No momento de testar a programação aos pares, o aluno-usuário foi orientado a solicitar ajuda de um integrante da equipe para realizar a codificação pareada. O código gerado pela ferramenta deveria ser executado no computador-didático.

Cada aluno-usuário recebeu um questionário de avaliação e validação, tendo que responder as seguintes questões, assinalando (1) muito fraco (2) fraco (3) regular (4) bom (5) ótimo. Também havia uma seção para reportar problemas encontrados na utilização da ferramenta.

Questões:

1. A linguagem de programação definida é didática, ou seja, fácil de compreender e utilizar?
2. Quando você gera um erro a ferramenta o ajuda a compreender onde o erro foi cometido?
3. A IDE de desenvolvimento é agradável e fácil de utilizar?
4. A documentação (manual do usuário) está bem escrita e compreensível?
5. As versões são salvas corretamente? É fácil o controle de versões?
6. A programação aos pares funcionou corretamente? O ajudou na utilização da ferramenta?

A execução do teste de aceitação foi importante para que as equipes tivessem uma retroalimentação sobre o produto. A Figura 1 apresenta um gráfico contendo a média de pontos obtidos por cada equipe, considerando as questões de 1 a 6. Observa-se que apenas as Equipes 4, 6, 8, 12 e 13 obtiveram uma média maior que 2,5. Para estas equipes grande parte dos problemas foram relacionados à “programação em pares” e a “retroalimentação dos erros”. As equipes com menores médias (Equipes 5 e 7) apresentaram diversos problema, sendo alguns deles problemas básicos como a execução de cálculos aritméticos.

Através do gráfico de médias feitas a partir das questões aplicadas (Figura 2), observa-se que as maiores dificuldades encontradas referem-se ao “controle de versões” e a “programação em pares” (questão 5 e 6). Acredita-se que os grupos ficaram mais focados no desenvolvimento de requisitos básicos e deixaram a implementação destes requisitos para o final do projeto.

Ao final do teste de aceitação os resultados do questionário foram enviados aos respectivos grupos para que estes pudessem avaliar os problemas encontrados pelos alunos.

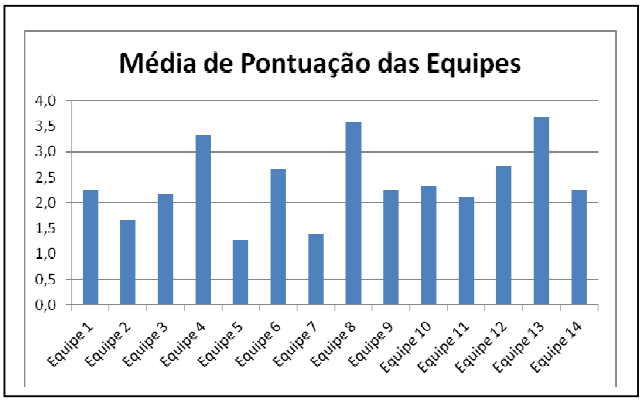

Figura 1. Média de pontuação realizada através do resultado do questionário

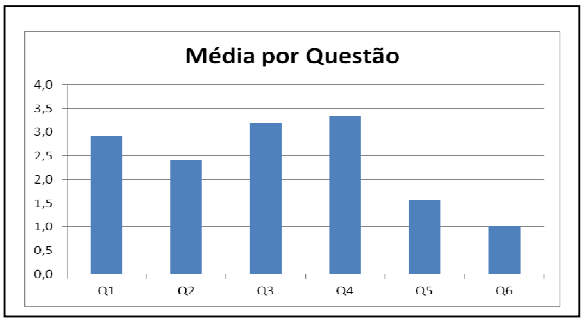

Figura 2. Média dos valores atribuídos a cada questão

3.3 Benefícios e Dificuldades na Execução

O grande desafio na execução do projeto interdisciplinar foi compatibilizar o ensino-aprendizagem das duas disciplinas com a realização do projeto. Este fato fez com que muitos alunos apresentassem resistência e descrença durante a apresentação da proposta. Um exemplo de dificuldade foi no momento de definição dos requisitos, onde os alunos argumentavam que não conheciam o processo de construção de um compilador, sendo, portanto, difícil elicitar e especificar os requisitos. Com o andamento do projeto, os alunos foram direcionados e perceberam que os conteúdos apresentados nas aulas teóricas das duas disciplinas eram fundamentais para a execução prática do projeto e, consequentemente, conseguiriam um bom desempenho nas duas disciplinas.

Com o tempo, os benefícios da interdisciplinaridade ficaram notórios. Os alunos observaram que a dedicação ao projeto garantiriam conhecimentos e boas notas nas duas disciplinas. Este fato fez com que houvesse maior motivação e dedicação às aulas de laboratório. Nas aulas de teoria não foi diferente: os alunos praticamente não faltavam, ocasionando em grande porcentagem de assiduidade. A participação foi significativa, tornando as aulas mais dinâmicas, principalmente pelos questionamentos e conclusões, que relacionavam a teoria e a prática e o evidente entendimento da interdisciplinaridade.

Outra dificuldade encontrada na disciplina de Engenharia de Software durante a execução da experiência interdisciplinar, foi praticar o desenvolvimento iterativo e incremental exatamente como o proposto na WBS. Em cada iteração ou *Sprint*, as seguintes entregas tinham que ser desenvolvidas: revisão de requisitos, descritivos de casos de uso, modelo de dados, modelo de classe, modelo de projeto, implementação, testes unitários, testes de integração e liberação.

Quando se considera um projeto de mercado, espera-se que a equipe tenha conhecimentos necessários para a realização das entregas. No caso de um projeto acadêmico, os alunos vão adquirindo os conhecimentos de acordo com o andamento da disciplina. Com isso, a padronização das entregas dos *Sprints* não funcionou conforme o planejado. Para contornar esta dificuldade, as entregas dos *Sprints* eram solicitadas em conformidade com o aprendizado. Esta experiência foi interessante, pois os alunos começaram a adquirir senso crítico sobre a real necessidade das entregas impostas na realização do *Sprint*. No primeiro *Sprint*, as entregas limitou-se em revisão de requisitos, descrição de casos de uso e modelagem de dados. Modelos da UML e testes foram realizados nos demais *Sprints* de forma gradativa. Nas conclusões, as equipes perceberam a diminuição dos defeitos e falhas, quando modelos e testes foram realizados, gerando uma liberação do compilador com mais qualidade.

4. Análise da Experiência

De forma geral a experiência com o projeto interdisciplinar foi positiva. A fase de planejamento exigiu grande interação das docentes para compatibilizar conteúdos, marcos e solucionar conflitos. A interação docente e discente foi intensa, com grande envolvimento participativo em todas as fases do projeto, pois todos se comprometeram com os resultados.

O fato do desenvolvimento de um compilador não ser um cenário conhecido pelos alunos foi de fundamental importância para o ensino-aprendizado de Engenharia de Software, principalmente no referente a elicitação de requisitos, arquitetura de software, controle de projeto e realização de testes. Nos processos de elicitação e validação de requisitos, os alunos realmente viram na prática a importância das técnicas de JAD e de prototipação. Após as validações dos protótipos, as equipes mostraram-se mais seguras para dar continuidade ao projeto.

Analisou-se também um envolvimento da equipe para garantir os resultados, baseando em técnicas de controle de projetos, como Análise de Valor Agregado para entender de forma clara e objetiva, através de métricas, os atrasos e os desvios em relação ao planejado. A cada semana, as métricas de Análise de Valor Agregado eram listadas, sendo a base de controle para a definição de plano de ação para alinhamentos.

Praticar a arquitetura de software foi algo fundamental para viabilizar a entrega do compilador para rodar nos ambientes Web e *Desktop* e também facilitar a reutilização de código e manutenção. Os alunos perceberam que trabalhar com o padrão arquitetural *Model View e Controller* (MVC) foi fundamental para cumprir os prazos dos *Sprints* [Pressman, 2011].

A realização de testes unitários trouxe uma experiência bastante importante. Devido a complexidade de algumas classes e falta do emprego de boas práticas de

orientação a objetos na codificação, algumas equipes tiveram dificuldade em realizar os casos de testes e sua execução. Em algumas equipes, a refatoração do código foi necessária, ocasionando em atrasos no projeto, evidenciando que não executar melhores práticas, impactam a qualidade e tempo de um projeto de software.

As características didáticas e usabilidade do compilador foram avaliadas como "regular" pelos participantes dos testes de aceitação e validação do ambiente. Isto trouxe uma retroalimentação interessante à disciplina de Engenharia de Software para dar maior ênfase nos conceitos e fundamentos sobre Interface Humano-Computador. Os alunos se sentiram motivados com a realização dos testes de aceitação e viram na prática como reagem usuários ao receber um produto de software.

Algumas equipes antes do teste com o computador didático tinham a certeza que o projeto estava concluído, no entanto, ao efetuar testes, o compilador apresentou falhas, sendo necessário correções e revisões para sanar os problemas. O código de máquina gerado por algumas equipes, após as análises, algumas vezes não previam laços encadeados, operações aritméticas sucessivas, entre outras funções, que dificilmente conseguiam visualizar sem o teste no equipamento físico. Ao realizar o teste no equipamento físico foi possível visualizar de forma direta se o compilador produzia a saída esperada. O desenvolvimento em camadas do MVC, foi destacado no documento de lições aprendidas como benéfico no referente a agilidade para realizações de manutenções e refatoração.

Embora as equipes tenham conseguido cumprir os objetivos gerais do projeto, os requisitos de programação aos pares e versionamento, foram concluídos com sucesso e com aprovação nos testes de aceitação apenas por duas equipes. Provavelmente, a maioria das equipes não tinham conhecimento em programação distribuída, conhecimento necessário para a realização destes recursos.

5. Considerações Finais

Este artigo apresentou a experiência interdisciplinar do curso de Engenharia da Computação da FACENS, Sorocaba. Em consonância com as diretrizes sobre interdisciplinaridade da Sociedade Brasileira de Computação, a proposta buscou integrar conceitos de duas disciplinas essenciais à formação de profissionais na área de desenvolvimento de software, trazendo aos alunos situações reais que são normalmente vivenciadas no mercado de trabalho. A primeira edição da experiência ocorreu no ano de 2011. Uma nova será executada no ano de 2012.

A experiência mostrou que a interdisciplinaridade permite aos docentes envolvidos não só a troca de vivências, mas também a geração de planejamentos mais criativos e desafiadores. Essa interação pode agregar valor à qualidade de ensino, como também para pesquisa e desenvolvimento motivacional dos discentes. O projeto também motivou alguns alunos seguirem na linha de pesquisa sobre Compilador Web, mostrando interesse em desenvolver o TCC neste tema.

Como lições aprendidas para as próximas experiências, deve-se destacar a necessidade de uma avaliação mais profunda das exigências de conhecimento para a realização de certos requisitos. Durante a fase de planejamento do projeto interdisciplinar é fundamental que os docentes envolvidos certifiquem-se que os

requisitos que serão necessários para implementação do produto não exigem conhecimentos que o aluno ainda não adquiriu durante o curso.

Considerando a aprendizagem do aluno pode-se observar que o cenário proposto possibilitou a experiência de desenvolvimento de um software de uma área que não fazia parte do senso comum dos alunos. Com isto o esforço para o entendimento do domínio em questão foi fundamental, mostrando aos alunos a importância do levantamento de requisitos e da validação destes. Outro ponto fundamental, é a aplicação de diretrizes de gerenciamento de projeto exercitando inclusive a hierarquia de papéis dentro dos grupos.

Um resultado fundamental da experiência foi que todas as equipes conseguiram desenvolver o compilador, destacando seus resultados intermediários tanto na modalidade *desktop* quanto Web. Este fato foi bem satisfatório, pois em anos anteriores, poucas equipes atingiam o objetivo de desenvolver um compilador apenas em ambiente *Desktop.*

6. Referências

Aho, A. V., et al. (2008), "Compilers: principles, techniques, and tools", Addison Wesley, 2th edition.

Beck, K. (2004), "Programação Extrema Aplicada - Acolha as mudanças", Bookman.

Brereton, P., Turner, M., Kaur, R. (2009) "Pair programming as a teaching tool: a student review of empirical Studies", 22nd Conference on Software Engineering Education and Training, pp. 240-247.

Diretrizes Curriculares da SBC (2005) "Currículo de Referência para Cursos de Graduação em Bacharelado em Ciência da Computação e Engenharia de Computação", Disponível em: http://portal.sbc.org.br/educacao/lib/exe/fetch.php?media=documentos:cr2005.pdf, Acessado em: 20/01/2011.

Garcia, F. D., Legaspe, E. (2011) "Kit didático CPU Board", Congresso Nacional de Iniciação Científica, Santos.

More, A., Kumar, J., Renumol V. G. (2011) "Web Based Programming Assistance Tool for Novices", IEEE International Conference on Technology for Education, pp. 270-273.

PMI (2008) "A Guide to the Project Management Body of Knowledge (PMBOK guide)", 4th edition.

Pressman, R. (2011), "Engenharia de Software", Mc Graw Hill, 7ª edição.

Schocken, S., Nisan, N., Armoni, M. (2009) "A Synthesis Course in Hardware Architecture, Compilers, and Software Engineering", SIGCSE'09, pp. 443-447.

Yamamoto, F. S., da Silva, A. F., Zanutto, J., Zampirolli, F. A. (2005) "Interdisciplinaridade no Ensino de Ciência da Computação", XXV Congresso da Sociedade Brasileira de Computação, WEI, pp. 2395-2406.