Considerações sobre Educação em Engenharia de Software

A.v. Staa[1]
arndt@inf.puc-rio.br

Departamento de Informática
Pontifícia Universidade Católica
22453-900 Rio de Janeiro,
Brasil

Junho 1995

PUC-RioInf.MCC 19/95

Resumo

Neste artigo apresentamos e justificamos alguns dos requisitos a serem satisfeitos por qualquer programa de ensino em engenharia de software. Não procuramos apresentar mais um "currículo ideal". Um bom currículo é fruto da consolidação de opiniões de diversos profissionais.

Caracterizamos dois tipos de desenvolvedores: o amador e o engenheiro de software. O amador tem pouco conhecimento formal em engenharia de software. Para estes recomendamos a criação de material didático para dotá-lo de um mínimo de conhecimento formal, visando especialmente métodos e processos.

Os requisitos do currículo levam em conta ser a engenharia de software uma área inerentemente experimental. Assim não é suficiente dar meramente um bom embasamento formal. É fundamental que se dê também um bom embasamento experimental através de trabalho controlado realizado em laboratórios.

Abstract

In this article we identify and justify a set of requirements which should be met by all software engineering curricula. We are not presenting yet another "ideal curriculum", since it is our belief that such a curriculum should reflect the opinion of a large number of professionals.

We identify two classes of software developers: the amateurs and the software engineers. Amateurs have little formal knowledge of software engineering. For them we recommend creating training material providing a minimal formal knowledge, specifically with regard to methods and processes.

It is our belief that software engineering is inherently experimental. Hence it is not sufficient to provide students with theory and models, also a strong experimental foundation is needed. This knowledge requires a reasonable amount of well equipped laboratories.

[1] Apoiado pelo CNPq, Bolsa de Pesquisador 300029/92-6

Software Engineering ... is the part of computer science that is too difficult for the computer scientist.

— FL.Bauer [Bauer71]

What were the lessons learned ..., is the fact that SOFTWARE IS HARD... From now on I shall have significantly greater respect for every successful software tool that I encounter.

— D.E. Knuth [Knuth74]

Judging by the quantity and quality of journals, conferences, academic programs, and industrial training courses, the intellectual discipline of software engineering has made great strides in recent years. Despite these advances, software engineering is far from a mature discipline.

R.E. Fairley Keynote Address, Conference on Software Engineering Education

Education — The world's best investment

Anônimo

Introdução

Computadores estão se tornando cada vez mais onipresentes. São produzidos mais de um bilhão de *chips* de CPU por ano. Uma parcela considerável deles custando em torno de um dólar ou menos. Computadores são utilizados como instrumentos de lazer, de apoio ao ensino, controladores nas mais variadas e insuspeitas máquinas e equipamentos. São utilizados por crianças, donas de casa, artistas, educadores, jornalistas, médicos, operários. Ah sim, são utilizados também por profissionais nas tradicionais atividades de processamento de dados, de controle de processos e de comando e controle.

Confortavelmente instalados em nossas casas, utilizando uma linha telefônica, podemos estudar arte consultando um dos *www-sites* de algum museu. Podemos nos afogar em literatura, buscando pela rede as referências mais atuais sobre um assunto qualquer. Podemos entabular discussões sobre os mais variados assuntos com pessoas que jamais vimos e que moram em lugares que jamais conseguiremos visitar.

Para quem há mais de trinta anos começou programando em um dos dois únicos computadores no pais, ambos a válvula (por falar nisso: o que é um duplo triodo?), que fez programas fiando painéis de tabuladoras eletro-mecânicas, a evolução foi espantosa, requerendo um imenso esforço de contínua adaptação e atualização. Mesmo para aqueles que começaram a sua vida profissional em Informática há uns dez anos, as mudanças com a chegada e disseminação dos microcomputadores foram espantosamente rápidas. Tendo em vista a rapidez do processo, pode-se falar numa explosão, muito mais do que numa revolução.

Para quem não é do ramo, a velocidade da evolução é mais espantosa ainda e, muitas vezes, traumática. Alguém pode imaginar um mundo sem banco eletrônico? Alguém pode imaginar escrever um livro com lápis e papel, desenhos feitos a nanquim em papel vegetal, os manus-

critos laboriosamente transcritos usando máquina de escrever, para depois serem compostos utilizando uma foto-compositora, revistos, e finalmente diagramados com tesoura e cola. Pois bem, há menos de quinze anos era assim. Quantos gerentes em carreiras promissoras foram demitidos em virtude de alguma "reestruturação" das respectivas empresas, reorganização esta tornada possível pelos modernos computadores? Quantos operários são necessários hoje para produzir e montar um carro? E quantos robôs? Qual o nível de escolaridade que operários precisam ter para poderem receber o treinamento necessário para poderem exercer suas tarefas? Qual o incremento da qualidade média paradoxalmente acompanhado de significativa redução de custos dos produtos? Tudo isto se tornou possível somente por serem utilizados processos fabris altamente automatizados.

Mas grassa também na Informática a mesma arrogância dos que se julgam capazes de resolver qualquer problema, de construir qualquer artefato, sem cuidar devidamente das conseqüências. A mesma arrogância que levou à construção de navios que jamais poderiam afundam: Titanic e Andrea Doria, cujos acidentes ensinaram conceitos básicos de segurança em alto mar. A falta de humildade que levou à construção de aviões tipo President e Comet, cujos acidentes ensinaram conceitos básicos de resistência dos materiais, viabilizando a construção de aviões estruturalmente seguros.

Para que permaneçamos humildes, recomendo a leitura da seção de acidentes tecnológicos publicada no Software Engineering Notes da ACM SIGSOFT. Para ilustrar a importância desta humildade, o anel de grau dos engenheiros do Canadá é um aro de aço, em memória a uma ponte pênsil literalmente levada pelo vento. A foto correspondente se encontra na maioria dos livros de resistência dos materiais.

Embora não devemos justificar os nossos erros com base nos erros dos outros, os parágrafos acima têm por objetivo mostrar que também nas engenharias tradicionais cometem-se erros e aprende-se muito com os erros cometidos. Infelizmente, alguns desses erros têm conseqüências trágicas.

Mas por quê esta longa introdução aparentemente fora do contexto? Computadores não funcionam se não existir software. É este software que controla os robôs, que calcula a estrutura de aeronaves, que controla o vôo destas mesmas aeronaves, que mantém usinas nucleares sob controle, que diz onde se deve cavar um poço para encontrar petróleo, que nos informa do nosso saldo no banco, que nos dá algumas horas de entretenimento com algum jogo eletrônico, que nos permite escrever, compor e diagramar livros com uma qualidade visual de dar inveja a qualquer editora de meados do século XX.

É também software que faz cair aviões, que causa colapsos em sistemas de comunicação, que põe em risco a vida que circunda usinas nucleares, que nos incomoda quando recebemos mensagens do gênero *general protection fault*, que registra e divulga informação pessoal nossa sem que saibamos disso, que nos frustra quando resiste a todas as tentativas de convencimento na busca de um resultado desejado.

Com este pano de fundo chegamos ao tema central deste artigo: o que são e o que ensinar aos futuros desenvolvedores de software. Como fazer com que sejam capazes de evoluir profissionalmente, mantendo-se competitivos pelos próximos vinte anos. Como fazer com que o fruto de seu trabalho não represente um risco para a sociedade.

Nas seções a seguir procurarei identificar o perfil de desenvolvedores de software, em particular de engenheiros de software. A seguir apresentarei propostas de áreas de conhecimento que julgo relevantes para a formação de engenheiros de software tendo o perfil identificado,

bem como as características a serem levadas em conta ao estabelecer programas de ensino em engenharia de software.

Perfil dos desenvolvedores de software

Inúmeras pessoas desenvolvem software útil e amplamente utilizado, desde adolescentes a profissionais altamente qualificados em computação, passando por toda a sorte de amadores e curiosos. Visicalc, um software que revolucionou a forma de realizar cálculos em tabelas — planilhas eletrônicas —, foi desenvolvido por alunos de administração com conhecimento básico de computação. Uma grande parte do software de engenharia é desenvolvido por engenheiros com conhecimentos básicos de Fortran e cálculo numérico. Um dos anseios da comunidade de processamento de dados é tornar possível que usuários desenvolvam uma parte significativa das aplicações de que necessitam.

Como já insinuado, nenhum desses amadores seria capaz, no entanto, de desenvolver sistemas integrados e abrangentes de processamento de dados, sistemas de controle de processo confiável, ou sistemas operacionais.

Identificação do perfil do desenvolvedor de software amador

Uma parcela significativa do software é e continuará sendo desenvolvida por pessoas que não possuem uma educação formal profunda em engenharia de software. Todos estes esforços de desenvolvimento estarão limitados quanto à complexidade e, com raras e honrosas exceções, produzem resultados de qualidade duvidosa. Para vencer estes limites é necessário conhecimento formal de engenharia de software, tal como é necessário conhecimento de matemática para o engenheiro.

Se por um lado desenvolver software complexo e de boa qualidade é difícil ("... *software is hard* ..."), por outro lado, enquanto não ultrapassar certos limites de complexidade, em geral o desenvolvimento de software não requer conhecimento técnico muito aprofundado. Vale aqui a analogia da construção de uma casa. Para construir uma casa não é necessário muito conhecimento de engenharia civil. No Rio é só olhar para os morros e contemplar os prédios de até três andares que se encontram nas favelas. Nenhum desses prédios de favela foi construído por engenheiro. Percorra o interior do país e conte o número de casas, barragens e pontes construídas artesanalmente. Porém, estes artesãos conseguiriam construir prédios de trinta andares? Verifique se Santos Dumont possuía um diploma de engenheiro aeronauta. No entanto, utilizando somente os conhecimentos adquiridos empiricamente, Santos Dumont seria capaz de construir um 747?

No entanto, mesmo os artesãos devem possuir uma formação básica de engenharia de software. Voltemos à analogia da casa. O "pedreiro" que não sabe utilizar corretamente um fio de prumo, ou que economiza demasiadamente no cimento, não conseguirá construir nem uma casa de um único andar.

Temos então o primeiro perfil: o *desenvolvedor de software amador*, este é um especialista (ou entendido) em uma área de aplicação e que desenvolve programas para esta área, não tendo formação específica em engenharia de software. São exemplos de amadores: engenheiros que desenvolvem aplicações para apoiar tarefas de cálculo de sua especialidade; artistas que desenvolvem aplicações para a geração ou o estudo de obras de arte; e — perdoem-me os meus co-

legas — os cientistas da computação que desenvolvem protótipos ou mesmo sistemas de uso mais amplo, porém sem utilizar processos definidos e sem qualquer preocupação com custos e garantia da qualidade.

A sociedade e estes desenvolvedores de software, muito se beneficiariam se possuíssem um conhecimento básico em engenharia de software que os habilitasse a trabalhar melhor. Infelizmente, estas pessoas não querem gastar tempo nem esforço para adquirir o conhecimento básico necessário.

O conhecimento básico de engenharia de software necessário a um amador não se limita a conhecimentos rudimentares de uma linguagem de programação, a conhecimentos de alguns conceitos elementares de estruturas de dados e de bancos de dados, e a conhecimentos de uso de um conjunto de API's. O conhecimento básico requerido para um amador deve enfatizar método de trabalho e prevenção de defeitos, ou seja deve ser suficiente para habilitar o amador a desenvolver direito e a saber o que se entende por direito. Tornando disponível cursos curtos e material auto-didático bem focalizado visando este conhecimento básico, em muito já se estará contribuindo para diminuir os problemas de produtividade e de qualidade do software disponível.

Identificação do perfil do engenheiro de software

Consideremos agora o engenheiro de software. Primeiro o que é engenharia de software? A seguir vão algumas definições examinadas a fundo em [Berry92]:

1. *Software Engineering is that form of engineering that applies the principles of computer science and mathematics to achieving cost-effective solutions to software problems* [Ford90]
2. *Software Engineering: (1) The application of systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1).* [IEEE Standard Glossary of Software Engineering Terminology]
3. *Software Engineering is that form of engineering that applies (i) a systematic, disciplined and quantifiable approach; (ii) the principles of computer science, design, engineering, management, mathematics, psychology, sociology, and other disciplines as necessary; (iii) sometimes pure invention; to creating, developing, operating and maintaining cost-effective, reliably correct, high-quality solutions to software problems.*

Algumas características destas definições merecem destaque:

- o software deve poder operar economicamente em ambientes reais. Ou seja, deve-se produzir software para que usuários possam realizar a contento as suas tarefas. Esta é uma das razões do sucesso do software desenvolvido por amadores.
- o software deve ser produzido economicamente. Ou seja, deve-se poder utilizar engenheiros de software com um nível de formação normal. Certamente não será econômico utilizar PhD's para desenvolver software. Tampouco é de se esperar que a distribuição de competência no meio dos desenvolvedores de software seja diferente da encontrada em outras profissões. Por outro lado, a utilização de pessoas com formação inadequada é comprovadamente um fator de prejuízos

[Boehm81]. Caem nesta categoria as inúmeras pessoas ludibriadas por cursos que deformam, ou formam mal, supostos profissionais que acabam sendo empregados por salários miseráveis e, fatalmente, produzirão software de baixa qualidade.

- software deve ser mantido. É conhecido que software, quando posto em ação, interage com o seu ambiente de tal modo que os requisitos inicialmente identificados mudam [Lehman91]. É conhecido, também, que o próprio processo de desenvolvimento de software é um processo de aprendizado e, conseqüentemente, sujeito a repetidas alterações. Note que manutenção estrutural existe em outras áreas de engenharia também. Em muitos prédios os apartamentos do primeiro andar se transformaram em lojas ou em garagens. Mas talvez o exemplo mais gritante poderá ser visto ao visitar a PUC do Rio. Perto do campus tem um prédio pelo qual hoje passa uma auto-estrada. Esta foi construída muitos anos depois do prédio ter sido concluído e certamente não constava dos planos originais.
- deve-se conhecer os princípios básicos e a matemática necessária para poder desenvolver software. Com exceção de algumas poucas áreas de aplicação, estes princípios e matemática pouco tem a ver com os tradicionais princípios de física utilizados nas outras áreas de engenharia. Como mencionado em [NRC94] engenharia de software é uma disciplina sintética, no sentido que estuda fenômenos que são estritamente fruto da criação humana.
- nem sempre se sabe o que deve ser desenvolvido. Ou seja, produzir uma especificação de requisitos completa e correta a priori é, muitas vezes, inviável. Segue-se que desenvolver software é um processo de aproximações sucessivas, definindo, desenvolvendo, avaliando, evoluindo a definição e assim sucessivamente, até que se tenha adquirido suficiente conhecimento para construir a versão final [Boehm88]. Note que em engenharias tradicionais é comum desenvolver vários protótipos, avaliá-los cuidadosamente, modificá-los até que se tenha adquirido suficiente conhecimento e certeza passar para uma produção em série. Quem já encomendou uma obra com finalidade específica, por exemplo um laboratório, deve ter observado que um período considerável é consumido negociando a arquitetura externa e interna, onde por quais equipamentos, etc. A medida que obra vai sendo realizada, sempre surgem alterações. Para reduzir os custos destas alterações, os modernos prédios de escritórios evitam embutir nas paredes e lajes a fiação, os canos, as luminárias, e os dutos de ar condicionado. Desta forma tornam-se estes prédios manuteníveis.
- é necessário saber planejar e gerenciar o desenvolvimento de software. É conhecido que o processo utilizado para desenvolver qualquer artefato é um dos itens de maior influência na qualidade média esperada.
- é necessário ter conhecimentos em áreas que são não convencionais em contextos eminentemente técnicos, sendo as mais marcantes: redação técnica, psicologia e sociologia. Usando um exemplo como justificativa, muitas das dificuldades de interfaces de usuário derivam diretamente da inadequação ao usuário real do modelo do usuário embutido no software.

Nas definições acima falta evidenciar que software não é fim em si mesmo. Software sempre visa resolver problemas em alguma área de aplicação. Esta pequena constatação tem enormes efeitos sobre a engenharia de software, uma vez que implica dizer que o corpo de conheci-

mento de engenharia de software é complementar ao conhecimento das áreas de aplicação. Em outras palavras, ou a aplicação a ser desenvolvida é especificada pelos correspondentes especialistas e o engenheiro de software constrói o sistema que atenda a esta especificação, ou o engenheiro de software deverá ter conhecimentos na área da aplicação e, neste caso, produz tanto a especificação como o sistema. Corolário destas constatações é ser fundamental a existência de especificações de qualidade satisfatória.

Engenharia de software é largamente experimental. Já foi mencionado que é difícil produzir uma especificação completa e correta a priori. Segundo [Lehman91] (já insinuado em [Boehm88]), isto é uma propriedade inerente do software. De uma certa forma isto implica dizer que o processo de desenvolvimento e manutenção é um processo evolutivo que converge para soluções momentaneamente satisfatórias, onde esta convergência é assegurada através de laços de *feed-back*. Ou seja, constrói-se algo em conformidade com uma especificação, usa-se e conduz-se experimentos controlados com o que foi construído e, a partir dos resultados observados, define-se a próxima mutação a ser realizada.

Segundo [NRC94]:

> *Experimental computer science and engineering (ECSE) is the fundamental underpinning of the computer hardware and software that drive the information age. Many widely known computer advancements of the 1980s trace their origins to ECSE research. Examples are reduced instruction set computers, window systems, relational databases — but many more are hidden inside systems, making them faster more efficient, or more functional, ... Experimental work is also an essential intellectual element of the computer science and engineering discipline that enriches research and teaching in the field.*

Os experimentos devem focalizar:

- *técnicas*: linguagens de representação, controle de qualidade, algoritmos.
- *métodos*: ações a serem realizadas para produzir uma determinada representação (documento).
- *processos*: a composição de tarefas e atividades que permitem conduzir o desenvolvimento, a manutenção e a gestão do desenvolvimento, desde a concepção até a descontinuação, passando pela criação, implantação e sucessivas evoluções do software.
- *ambientes*: o conjunto de plataformas, ferramentas, métodos, processos, que apoiam o desenvolvimento, a manutenção e a gestão.

Os trabalhos experimentais devem ser realizados de forma controlada. Ou seja, do ponto de vista da engenharia de software, não basta desenvolver uns tantos programas e documentos, é necessário que este desenvolvimento se dê segundo um ritual bem definido, e que os experimentos sejam devidamente documentados. Isto implica a participação em projetos com alguma envergadura, gerando resultados de qualidade assegurada. Idealmente, estes seriam projetos reais.

Podemos concluir então que o perfil do engenheiro de software abrange, entre outras coisas:

- habilidade de estabelecer ambientes.
- habilidade de definir processos e planos, de desenvolver e manter segundo estes planos, e de gerenciar o desenvolvimento e a manutenção.
- habilidade de definir e medir qualidade do software, tanto do ponto de vista de serviço, como do ponto de vista de engenharia.

- habilidade de definição e uso de métodos e técnicas aplicadas às diferentes atividades do processo de desenvolvimento, manutenção e gestão do desenvolvimento.
- habilidade de conduzir experimentos controlados, registrá-los e extrair conclusões consistentes com os resultados obtidos.
- habilidade de comunicação escrita e oral.
- habilidade de estudo independente de modo a manter-se continuamente atualizado.

Características e requisitos dos currículos

Em [Denning89] são identificados três paradigmas, ou estilos culturais, utilizados para efetuarmos nossas atividades, a saber:

1. teoria, baseada em matemática.
2. modelagem (abstração) baseada no método científico experimental.
3. *design* (projeto) baseada em engenharia.

Para produzir uma teoria coerente e válida, realizam-se repetidamente os seguintes passos até não serem mais encontrados erros ou inconsistências:

- definição: caracterizar os objetos a serem estudados
- teorema: formular hipóteses quanto a possíveis relacionamentos entre estes objetos.
- prova: determinar se estes relacionamentos valem.
- interpretar os resultados.

Para produzir um modelo aceitável, realizam-se repetidamente os seguintes passos até que os valores preditos pelo modelo coincidam com os valores observados:

- formular uma hipótese.
- construir um modelo e gerar uma predição.
- projetar um experimento e obter os resultados experimentais
- analisar os resultados, confrontando os resultados obtidos com os resultados esperados.

Para projetar um artefato (sistema, programa, módulo) aceitável, realizam-se repetidamente os seguintes passos até que o artefato satisfaça os requisitos estabelecidos:

- estabelecer requisitos.
- estabelecer uma especificação que permita alcançar os requisitos.
- projetar e implementar o artefato.
- testar o artefato.

Espera-se que um currículo de engenharia de computação exponha o estudante a estes três paradigmas nas diferentes sub-áreas. Elaborando um pouco sobre o que se encontra em [Denning89] temos a matriz a seguir. Um programa bem organizado deve cobrir todos os elementos desta matriz.

	Teoria	Modelo	Projeto
Algoritmos e estruturas de dados			
Computação numérica e simbólica			

Linguagens de programação			
Arquitetura de computadores			
Sistemas operacionais e redes			
Bancos de dados			
Inteligência artificial			
Interface de usuários			
Métodos e técnicas de engenharia de software			
Processos de software			
Comunicação e redação técnica			

A título de ilustração, examinamos a sub-área métodos e técnicas de engenharia de software. Na parte teórica devem ser abordados tópicos tais como:

- especificação formal
- prova de programas
- lógica temporal
- teoria de confiabilidade

Na parte de modelos devem ser abordados tópicos tais como:

- métodos de especificação, tais como tipos abstratos de dados, notação axiomática
- metodologias, tais como projeto estruturado, projeto orientado a objetos, compilação em separado, níveis de abstração
- ferramentas de software, ambientes de desenvolvimento
- medição e avaliação de programas e sistemas

Na parte de projeto devem ser abordados tópicos tais como:

- linguagens de especificação tais como VDM, Z
- editores dirigidos por sintaxe e editores de tela
- técnicas de instrumentação
- técnicas de teste
- projeto de interfaces de usuário
- projeto visando plataformas utilizando janelas
- métodos de projeto de sistemas grandes que sejam confiáveis, tolerantes a falhas e seguros

Foge ao escopo deste artigo detalhar as demais sub-áreas. Um minucioso detalhamento encontra-se em [Denning89].

Com relação à proposta contida em [Denning89] foram acrescentados os itens processos de software e comunicação. O primeiro deve-se à evidência do processo e da gerência do desenvolvimento como fatores primordiais para o desenvolvimento e manutenção de software de qualidade. Hoje problemas de gerência não se limitam a examinar alguns modelos de ciclo de vida. Discutem-se a fundo todas as etapas, inclusive as de controle tais como gerência de configuração e controle de versão. Dá-se especial ênfase à normalização, à especificação da qualidade e ao controle de qualidade.

Outra habilidade a merecer atenção é a de comunicação escrita e oral. Software hoje tende a ser muito mais complexo do que era há uns poucos anos atrás. Em adição, destina-se a um público muito maior, muitas vezes pouco familiarizado com conceitos de computação. Além

disso, os usuários tendem a estar dispersos e muitas vezes não têm acesso à documentação. Em vista disso, a documentação em papel está sendo gradativamente reduzida, sendo substituída por sistemas de auxílio cada vez mais sofisticados. Conseqüentemente, o problema de comunicação está deixando de ser um problema somente de sintaxe, de estilo básico e de diagramação, está passando a ser um problema de autoria de hiper-documentos.

Além do auxílio em si, muitos sistemas estão começando a conter tutoriais capazes de ilustrar dinâmica e interativamente o comportamento de partes do sistema. Os textos precisam ser bem focalizados no público alvo, precisam ser fáceis de ler e, sobretudo, precisam ser consistentes e completos. Os modernos processos de desenvolvimento obrigam a geração da documentação e do auxílio ser realizada concomitante com a produção dos programas. Ou seja, uma versão inicial da documentação deve ser produzida pelos desenvolvedores e deve estar disponível antes mesmo do sistema entrar em teste. Numa etapa a seguir, redatores técnicos com bons conhecimentos de desenvolvimento realizarão a revisão e o ajuste da documentação. Ao final de tudo isto, o controle de qualidade verifica minuciosamente a consistência entre o que está implementado e o conteúdo da documentação. O problema não termina aí, uma vez que os produtos evoluem no tempo.

Laboratórios

Como havia sido mencionado, engenharia de software é intrinsecamente experimental [NRC94]. Sendo assim, depende fortemente de trabalhos realizados pelos estudantes em laboratórios. O trabalho experimental realizado em laboratórios serve aos seguintes propósitos [Denning89]:

1. Laboratórios devem demonstrar como os princípios discutidos em sala de aula se aplicam ao projeto, implementação e teste de sistemas práticos. Para tal, os trabalhos experimentais devem estar bem coordenados com as aulas expositivas. Devem promover experiências concretas que permitirão os estudantes a compreender conceitos abstratos.
2. Laboratórios devem enfatizar os processos, conduzindo à internalização de *know-how* eficaz. Laboratórios devem enfatizar programação e não programas; uso correto de ferramentas de software; utilização correta da documentação; geração de documentação de experimentos e projetos. Para atingir esta meta devem estar disponíveis diversas ferramentas de software.
3. Laboratórios devem introduzir métodos experimentais, incluindo o projeto de experimentos controlados, a realização sistemática de experimentos, uso de instrumentação, análise estatística dos resultados, e apresentação apropriada dos resultados dos experimentos. Os estudantes devem aprender a distinguir entre experimentos cuidadosos e observações superficiais.

Laboratórios com estas características não são os equipamentos que alunos têm em casa. Embora seja desejável que alunos realizem trabalhos em casa ou em laboratórios informais, os laboratórios que procuram atingir as metas acima enunciadas são formais.

O trabalho em laboratório deve ser cuidadosamente planejado e supervisionado. Ou seja, um laboratório não é um conjunto de máquinas que o estudante utiliza quando e se quiser. Uma parcela significativa de trabalho laboratorial deve ter horário e tarefas bem definidas. As tarefas devem estar descritas em detalhe, expondo o objetivo e o método de trabalho de cada experimento.

Devem ser realizados trabalhos nos quais o estudante é solicitado a ler, entender e alterar partes substantivas de programas fornecidos. Este tipo de trabalho destina-se a evidenciar a necessidade de programas bem organizados e documentados quando se deseja ser capaz de mantê-los.

Laboratórios devem estar equipados com máquinas e software atualizado. As máquinas devem possuir configurações que permitam desenvolver os trabalhos sem dificuldade. De nada adianta ter uma ou duas máquinas bem configuradas e todas as demais sub-equipadas. Equipamentos defeituosos ou obsoletos somente frustram o estudante sem adicionar conhecimento algum.

As ferramentas de software utilizadas devem induzir bons hábitos. Os laboratórios devem ter documentação do hardware e do software de modo que os estudantes se acostumem a utilizar a documentação e não intuição ou "achologia" ao tentar resolver um problema. Cuidado especial deve ser dado a compiladores e a ferramentas de suporte ao desenvolvimento, tais como editores, depuradores, editores de ligação, *make*, etc. Os equipamentos devem ser poderosos suficiente para permitir o uso confortável destas ferramentas.

Deve-se ensinar linguagens utilizadas no mundo real, mesmo que, do ponto de vista acadêmico, tenham propriedades pouco razoáveis. De nada adianta esconder o estudante da realidade que ele encontrará nas empresas, somente aumentará a dificuldade de se entrosar, contribuindo para o chavão de que universidades não ensinam coisas que interessam. Deve-se ensinar um leque de diferentes linguagens: procedurais, orientadas a objetos, LISP, Prolog e outras.

Devem estar disponíveis laboratórios temáticos nos quais o estudante possa participar, ou pelo menos ver sendo executados, projetos de desenvolvimento reais. Estágios realizados em empresas junto com as aulas têm-se mostrado prejudiciais, por induzirem o estudante a faltar às aulas e por não terem vínculos específicos com a matéria sendo lecionada. Estágios em empresas são aconselháveis, porém deveriam ser realizados em tempo integral e não concomitante com as aulas.

Estudo continuado

Foi mencionado que a evolução em computação é extremamente dinâmica. Em virtude disso é quase impossível formar um estudante que, sem estudo adicional, ainda esteja atualizado cinco anos após a conclusão de seu curso. Algumas vezes tem-se até a impressão que ocorre obsolescência já durante o período de estudos. Tentativas de atenuar estes problemas através de uma formação muito intensiva em teoria e modelos não resolve, dada a natureza intrinsecamente experimental da engenharia de software.

Para dar ao estudante uma sobre-vida técnica prolongada, deve-se motivar e induzi-lo a estudar por conta própria. Deve-se induzir o estudante a entender o por quê das coisas e não meramente repetir o que foi dito em sala de aula. Isto tem por conseqüência uma forte ênfase em didática e interação de alunos com professores e assistentes. É necessário também dispor-se de uma biblioteca técnica bem fornida, não somente da literatura de apoio à pesquisa, mas também de literatura relatando técnicas, macetes e experiência profissional. Muito dessa literatura deve ser buscada através da Internet. Deve-se motivar os alunos a ler este material bibliográfico e deve-se capacitá-los a filtrarem a enxurrada de relatórios, artigos e livros que estão se tornando disponíveis.

Conclusão

Neste artigo procuramos caracterizar a área de engenharia de software, ressaltando as características mais marcantes. Estas características foram utilizadas para identificar alguns dos requisitos que programas de ensino em engenharia de software devem satisfazer.

Segundo a nossa experiência e o que é dito em [NRC94] e [Berry92], consideramos engenharia de software uma área inerentemente experimental. É evidente que é necessário um bom fundamento teórico, por exemplo: álgebra e teoria dos conjuntos, cálculo, álgebra linear, teoria da computação, lógica, linguagens formais, algoritmos, estruturas de dados. É evidente também que é necessário um bom fundamento em métodos e técnicas, tais como projeto estruturado, análise estruturada, análise orientada a objeto, máquinas de estado, sistemas operacionais e redes, projeto de máquinas virtuais. Todas estas teorias e métodos têm sido costumeiramente encontradas nos diversos currículos existentes.

Mas não é suficiente dar um bom embasamento formal. É fundamental que se dê também um bom embasamento experimental através de trabalhos controlados realizados em laboratórios. Para tal estes laboratórios devem estar bem equipados com hardware e software atualizado, e os trabalhos devem ser realizados com supervisão, principalmente nos semestres iniciais. Enfatizamos também a necessidade do estudante saber expor suas idéias com clareza, e saber redigir documentação, auxílios e tutoriais de forma clara, correta, completa e consistente.

Não procuramos aqui apresentar mais um "currículo ideal". Um bom currículo é fruto da conjugação de opiniões de muitos profissionais, possivelmente com opiniões divergentes. Alguns destes profissionais devem ser de universidades, outros devem ser de empresas. Ao invés de definir um currículo, procuramos identificar alguns dos requisitos básicos que, no nosso entender, qualquer bom currículo deveria satisfazer.

Referências bibliográficas

[Bauer71] Bauer, F.L; "Software Engineering"; *Information Processing IFIP'71*; Amsterdam; North Holland; 1971; pags 530-538

[Berry92] Berry, D.M.; *Academic Legitimacy of the Software Engineering Discipline*; CMU/SEI-92-TR-34; Software Engineering Institute, Carnegie Mellon University; Pittsburgh, PA; 1992

[Boehm81] Boehm, B.W.; *Software Engineering Economics*; Englewood Cliffs; Prentice-Hall; 1981

[Boehm88] Boehm, B.W.; "A Spiral Model of Software Development and Enhancement"; *IEEE Computer* 21(5); 1988; pags 61-72

[Denning89] Denning, P.J. (Chairman) et alii; "Computing as a Discipline"; Final report of the Task Force on the Core of Computer Science; *Communications of the ACM* 32(1); 1989; pags 9-23

[Ford90] Ford, G.; *1990 SEI Report on Undergraduate Software Engineering Education*; CMU/SEI-90-TR-3; Software Engineering Institute, Carnegie Mellon University; Pittsburgh, PA; 1990

[Knuth74] Knuth, D.E.; "Computer Programming as an Art"; *Communications of the ACM* 17(12); ACM Turing Award Lecture; December 1974; pags 667-675

[Lehman91] Lehman, M.M.; "Software Engineering, the Software Process and their Support"; *IEE Software Engineering Journal* 6(5); Londres; 1991

[NRC94] *Academic Careers for Experimental Computer Scientists and Engineers*; Computer Science and Telecommunications Board; National Research Council; Washington,D.C.; 1994