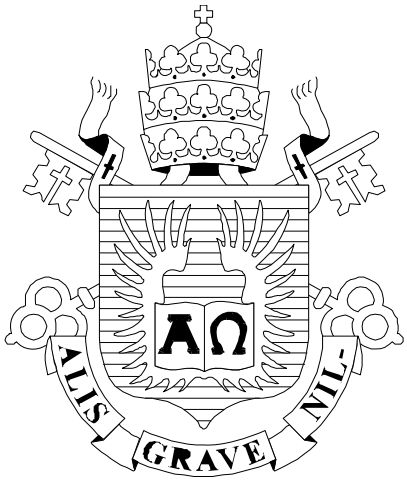

PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 14/08

Uma Agenda de Pesquisa para o Jovem Profissional em Engenharia de Software

Carlos J. P. de Lucena

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 14/08 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Março, 2008

Uma Agenda de Pesquisa para o jovem Profissional em Engenharia de Software

Carlos J. P. de Lucena

lucena@inf.puc-rio.br

Parnas, David L. (1998). "Software Engineering Programmes are not Computer Science Programmes".

Annals of Software Engineering 6: 19–37, p. 19

Abstract. The development and operation of high-quality widely-available software is essential for modern society to function, and the knowledge to create such software, called software engineering, can lead to a stimulating and fulfilling career. Thus, young people with very different backgrounds should consider this exciting field as it contains many problems related to both technology and economics, whose clear solution promises substantial rewards. For those with an interest in a research career, software engineering also offers many open problems in need of solution.

Keywords: Computer Science, Software Engineering, Research Agenda, young Software Engineering Professional.

Resumo. Saber como fazer software em maior quantidade e de melhor qualidade é um requisito essencial para o desenvolvimento da sociedade. Acreditamos que jovens com as mais variadas formações básicas deveriam considerar uma carreira em engenharia de software. A área é relevante dos pontos de vista tecnológico e econômico e promete grandes recompensas àqueles que começarem a produzir problemas claramente resolvidos. Para os jovens com interesse em uma carreira de pesquisador o que não falta na área são problemas em aberto a espera de soluções

Palavras-chave: Ciência da Computação, Engenharia de Software, Agenda de Pesquisa, jovem profissional de Engenharia de Software.

Responsável por publicações :

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br

Preâmbulo

Este capítulo foi escrito numa época em que a área de Ciência da Computação convive, nos âmbitos nacional e internacional, com uma crise de grandes proporções. A crise se manifesta através da diminuição da procura pela área por alunos candidatos à graduação e à pós-graduação e apoio reduzido de agências de fomento e da indústria à pesquisa básica na área (especialmente nos EEUU e UE). Paradoxalmente, o setor produtivo e o governo procuram, sem sucesso, grandes quantidades de especialistas em áreas que o mercado de trabalho sequer reconhece como parte da Ciência da Computação.

A solução para a crise foi muito bem discutida em [1]. Os autores advogam uma reorientação da Ciência da Computação de forma a lidar mais diretamente com problemas sociais em equipes interdisciplinares. Argumentam também que líderes acadêmicos devem tratar a necessidade de uma maior diversidade no corpo docente e no perfil de alunos de seus programas acadêmicos para reverter o declínio da procura pela área. Jovens com um uma amplitude maior de talentos, interesses e formação básica seriam, nestas condições, atraídos por novos temas da Ciência da Computação como, e este último argumento é nosso, o tema Engenharia de Software.

Concordando com o nosso ponto de vista, a University of Southern Califórnia, inverteu a estrutura acadêmica tradicional e criou uma School of Software Engineering à qual está subordinado um Computer Science Department. Do nosso ponto de vista, a Engenharia de Software é a subárea (?) da Ciência da Computação com maior potencial para promover a interdisciplinaridade de que falam Klawe e Schneiderman [1].

Saber como fazer software em maior quantidade e de melhor qualidade é um requisito essencial para o desenvolvimento da sociedade. Acreditamos que jovens com as mais variadas formações básicas deveriam considerar uma carreira em engenharia de software. E, se estes jovens tiverem inclinação para o trabalho de pesquisa, há inúmeras oportunidades neste campo como discutiremos, em termos gerais, nas seções seguintes.

1 Motivação

Pouco depois da redação deste capítulo (maio de 2008), será realizado nos dias 29 e 30 de maio, em São Petesburgo, na Rússia, o segundo *Spring Young Researchers'Colloquium on Software Engineering*.O colóquio se destina a jovens pesquisadores com menos de 30 anos de idade, interessados em se familiarizar com o trabalho dos seus colegas e trocar experiências. Foi prevista a submissão de trabalhos não apenas de pesquisa mas também de trabalhos de interesse industrial. A aproximação de pesquisadores com profissionais da indústria visa reduzir a permanente defasagem (*gap*) entre teoria e prática nesta área. Esta notícia sobre o segundo colóquio visou mostrar ao leitor uma tendência atual na direção do que foi dito acima. Aproveitamos a menção do "ever-existing gap" mencionado na chamada para participação no Colóquio como o tema da próxima seção.

Neste ponto, coloca-se a questão de porque estudar Engenharia de Software ? A informação contida em computadores e os sistemas de controle estão se tornando cada vez mais embarcados (*embedded*) e integrados ao tecido da sociedade humana. Ao invés de aguardar para prover informação e assistência apenas quando requisitados, estes sistemas estão atualmente intimamente envolvidos no complexo processo da vida diária.

Eles controlam nossos relógios, telefones celulares, máquinas de lavar, veículos a motor, a eletricidade das casas e os processos essenciais de produção da economia. Como conseqüência, sistemas computacionais não operam independentemente mas são, em geral, componentes de sistemas complexos muito maiores envolvendo hardware, software, pessoas e todos os eventos imprevisíveis da natureza.

Nossas próprias vidas dependem do funcionamento confiável e intermitente desses sistemas interdependentes. A maioria das pessoas, mesmo aquelas formadas em Ciências da Computação, não estão muito bem informadas tanto sobre as dificuldades envolvidas na construção desses sistemas complexos quanto sobre a necessidade das pessoas que os desenvolvem estarem familiarizadas com técnicas avançadas que não são ensinadas em cursos típicos de programação. A Engenharia de Software é a disciplina dedicada aos princípios e técnicas necessários para a construção competente dos sistemas de computação de hoje e do futuro.

Um engenheiro de software deve ser bem formado em técnicas para, (a) modelar e compreender complexos sistemas interativos, (b) especificar como construir sistemas de informação computacionais para aperfeiçoar estes sistemas., (c) gerenciar a construção de componentes para sistemas de informação, (d) assegurar que procedimentos foram definidos para o teste continuado e manutenção de sistemas em fase de operação.

Nossa prosperidade econômica depende da existência de profissionais capazes de supervisionar a construção de sistemas de informação complexos que viabilizarão a nova sociedade do conhecimento. Estes sistemas devem interoperar com outros situados em locais distantes. Ao mesmo tempo, eles devem ser eficientes, confiáveis e seguros na presença de uma ambiente global imprevisível e muitas vezes hostil. Os profissionais que serão líderes neste contexto são os engenheiros de software educados sobre as tecnologias atuais.

Finalmente, a essência do método científico é construir modelos do mundo e depois testar estes modelos para determinar se eles são válidos. Engenheiros de software estão fundamentalmente engajados neste tipo de atividade porque sistemas de software são essencialmente modelos parciais do mundo e estes modelos devem ser imediatamente testados sob condições operacionais para verificar se funcionam. Em resumo, engenheiros de software são construtores de modelos ambiciosos (e testadores destes modelos) e, dentre eles, os realmente competentes, estão engajados em modelagem científica todo o tempo. O estudo da engenharia de software envolve a aquisição dos tipos de competências que são valiosas em praticamente todas as profissões e dão suporte aos fundamentos da nossa cultura moderna.

2 Os Problemas de Adoção da Inovação Tecnológica em Engenharia de Software

Há muitos anos foi constatado que a adoção de uma nova tecnologia de software é uma questão muito complexa. Fala-se do "*ever-existing gap*" e pergunta-se quanto tempo levará para os seres humanos conseguirem utilizar todo o hardware disponível. Redwine e Riddle [2] já em 1985 analisaram um grande número de tecnologias de software para determinar como elas se desenvolviam e propagavam. Eles descobriram que decorrem de 15 a 20 anos para uma tecnologia evoluir do nível de formulação conceitual até o ponto em que ela está pronta para popularização.

Eles identificaram seis fases típicas:

- **Pesquisa básica**. Investigação das idéias básicas, estabelecimento de uma estrutura básica para o problema, tratamento de questões críticas da pesquisa.
- **Formulação do conceito**. Circulação informal das idéias, desenvolvimento de uma comunidade, convergência para um conjunto compatível de idéias, publicação da solução para subproblemas específicos.
- **Desenvolvimento e extensão.** Uso preliminar da tecnologia, formulação clara das idéias subjacentes, generalização do enfoque adotado.
- **Aperfeiçoamento interno e exploração**. Extensão do enfoque para outro domínio de aplicação, uso da tecnologia em problemas reais, estabilização da tecnologia, desenvolvimento de material para treinamento, demonstração do valor dos resultados
- **Aperfeiçoamento externo e exploração**. Similar ao interno, mas envolvendo uma comunidade mais ampla de pessoas que não participaram do desenvolvimento, evidências substanciais do valor e aplicabilidade.
- **Popularização**. Aperfeiçoamento da qualidade do produto, versões da tecnologia com serviços de suporte disponíveis, comercialização e *marketing* da tecnologia, expansão da comunidade de usuários.

Redwine e Riddle [2] apresentaram linhas do tempo para várias tecnologias de software a medida que elas progrediram ao longo dessas fases, até meados dos anos 80. Outros estudos foram conduzidos nos anos 90. Por exemplo, CVS (Concurrent Version System) foi lançado no mercado em 1986. Passaram-se 15 anos, exatamente como previsto em [2], para CVS passar a ser usado em larga escala. O modelo proposto em [2] continua, aparentemente, válido nos dias de hoje. No entanto, a adoção das tecnologias de software está se tornando mais rápida nos últimos 10 anos graças ao aumento exponencial nas comunicações: televisão, satélites, telefones celulares e , naturalmente, a Internet e a proliferação recente do software livre.

3 Uma Agenda de Pesquisa em Engenharia de Software

A Engenharia de Software é uma disciplina muito recente que ainda vai requerer muita pesquisa e experimentação para atingir o mesmo patamar de conhecimento das engenharias dos sistemas físicos. É arriscado fazer prognósticos sobre quais são as principais tendências de pesquisa para a área. No entanto, o objetivo deste capítulo é estimular a curiosidade do candidato a engenheiro de software que é, em geral, um estudante de um curso de Ciência da Computação ou áreas correlatas. Com esta motivação vamos procurar explorar as tendências de pesquisa atuais da área.

A conferência "Future of Software Engineering"(FOSE) realizada durante ICSE (*International Conference on Software Engineering*) 2000 documentou o estado da arte em engenharia de software em 2000 e sugeriu uma lista com vários problemas para serem resolvidos durante a década seguinte. As seções especiais FOSE durante ICSE 2000 e ICSE 2007 também contribuíram para delinear o estado da arte em engenharia de software.

Nestes eventos houve a identificação de temas cujas relevâncias são hoje bastante consensuais na comunidade da área. São eles: aspectos, agilidade, experimentação, desenvolvimento baseado em modelos (*model-driven*) e linhas de produto. Esta é uma boa lista e contém preocupações muito atuais e pertinentes para a área. Nós consideramos a lista incompleta e acrescentamos: sistemas multiagentes, e desenvolvimen-

to tecnológico de qualidade (sistemas disponíveis, corretos, seguros, escaláveis, persistentes e ubíquos). Passamos, a seguir, a comentar resumidamente cada um dos temas mencionados com ênfase especial naqueles que consideramos de maior importância.

3.1 Concerns e Aspectos

O princípio da separação de concerns [3] defende que, para superar a complexidade, deve-se resolver uma questão (ou concern) importante por vez. Na engenharia de software, esse princípio está relacionado à *modularização* e à *decomposição* de sistemas. Os sistemas de software complexos devem ser decompostos em unidades modulares menores e claramente separadas, cada uma lidando com um único *concern*. Se bem realizada, a separação de *concerns* pode oferecer muitos benefícios . Ela oferece suporte a uma melhor manutenibilidade com alterações aditivas, em vez de invasivas, melhor compreensão e redução da complexidade, melhor reutilização e uma integração de componentes simplificada. Os *concerns* podem ser gerais, dependentes do domínio e dependentes da aplicação. Os *concerns gerais* estão presentes em quase todos os sistemas de software, independente do domínio e da funcionalidade da aplicação. Alguns exemplos típicos de concerns gerais são tratamento de erro, distribuição, persistência, auditoria etc. Há várias abordagens recentes para a separação avançada de *concerns*, incluindo a programação orientada a aspectos [4], A *Programação orientada a aspectos* (POA) e a *Separação multidimensional de Concerns* (SMDC) são as técnicas mais conhecidas. Elas introduzem novas abstrações de modularização e mecanismos de composição para melhorar a separação de *crosscutting concerns* no nível da implementação.

3.2 Agilidade

O desenvolvimento de software ágil guia projetos de desenvolvimento de software que evoluem rapidamente com mudanças de expectativas em mercados competitivos. Os proponentes do método acreditam que métodos "pesados" orientados para a produção de muita documentação (como TickIT, CMM e ISO 9000) são considerados hoje de menor importância [5]. Muitos acreditam que companhias exportam as posições de trabalho de profissionais que trabalham usando métodos baseados em documentação intensiva. O método que, por excelência, caracteriza o enfoque ágil é chamado de *Extreme Programming* [6].

As metodologias ágeis surgiram com a proposta de aumentar a ênfase nas pessoas e reduzir a importância dos processos de desenvolvimento. Além disso, existe a preocupação de gastar menos tempo com documentação e mais com resolução de problemas de forma iterativa.

3.3 Engenharia de Software Experimental

Esta é a área da engenharia de Software [7] interessada na experimentação com software, na aquisição de dados decorrentes de experimentos e no estabelecimento de leis e teorias a partir destes dados. Os proponentes deste método argumentam que a natureza do software é de tal ordem que só será possível evoluir o conhecimento sobre software através de experimentação. Somos partidários da idéia que experimentação não é apenas uma área da Engenharia de Software, mas deverá ser, junto com o desenvolvimento baseado em software livre, uma metodologia essencial para todas as áreas dessa disciplina. Isto será mais detalhado na seção 4.1.

3.4 Desenvolvimento baseado em Modelos (*Model Driven*)

O desenvolvimento baseado em modelos utiliza modelos gráficos e textuais como os artefatos básicos para o desenvolvimento. A partir da transformação de modelos e geração de código uma aplicação parcial ou completa pode ser gerada.

A MDA (padrão OMG) é um *framework* para o desenvolvimento de sistemas baseados em fundamentos estabelecidos da engenharia de *software*, que separam a especificação da funcionalidade de um sistema da especificação desta funcionalidade em uma plataforma tecnológica específica. A MDA utiliza modelos de alta abstração para especificar a lógica da aplicação, cujos conceitos sobre linguagens ou plataforma são irrelevantes; trata-se de modelo independente de plataforma (PIM - *Platform Independent Model*). Este modelo de alto nível é posteriormente utilizado para criar novos modelos que expressem os requisitos do sistema em uma plataforma específica, o modelo específico (PSM -*Platform Specific Model*). Esses modelos não são usados apenas como documentação de sistemas, mas como uma ferramenta para a implementação. Cada atividade do processo de desenvolvimento requer um número de modelos de entrada que produzam outros modelos como saída. Sendo assim, o processo de construção de uma aplicação pode ser visto como um conjunto de transformações que levam ao sistema final [8].

3.5 Linhas de Produto de Software

Atualmente, um cenário comum, é a existência de organizações que desenvolvem produtos similares que são customizados para diferentes clientes. Usualmente elas praticam reutilização de software de uma forma empírica. Linhas de Produto para Software (*Software Product Lines*) são uma nova tendência em reutilização de software que possibilita a construção de aplicações de uma maneira sistemática. Clements & Northrop [9] definem as LPSs da seguinte forma: "um conjunto de sistemas intensivos em software que compartilham um conjunto comum de *features* que é gerenciável e que satisfazem necessidades específicas de um segmento de mercado ou missão particulares que são desenvolvidos a partir de um conjunto de recursos comuns segundo um processo definido". Um *feature* pode ser visto como uma propriedade de um sistema que é relevante para algum usuário e que é usada para capturar pontos em comum ou para discriminar entre produtos numa linha de produção. A idéia central da engenharia baseada em LPSs é analisar as *features* comuns e variáveis de aplicações em um domínio específico e desenvolver uma infra-estrutura reutilizável que forneça suporte ao desenvolvimento de software. Este conjunto de aplicações é chamado de família de produtos [10].

3.6 Sistemas Multiagentes

O desenvolvimento de sistemas baseados em agentes de software é uma das mais promissoras e crescentes áreas de pesquisa na academia e na indústria [11], [12], [13], [14], [15]. Avanços na tecnologia de redes, o aumento da penetração de dispositivos móveis no mercado (como celulares, PDAs, Smartphones, etc.) e uma demanda crescente por soluções computacionais para problemas complexos têm estimulado a investigação da tecnologia de agentes como um novo paradigma para a engenharia de software de sistemas complexos e distribuídos.

Atualmente, a tecnologia de agentes é adotada em uma grande variedade de domínios de aplicação, incluindo comércio eletrônico, educação à distância, computação móvel, otimização e interface homem–máquina, dentre outros. O progresso de diferentes áreas dentro da Ciência da Computação é evidente, existindo uma demanda crescente por soluções para a integração de métodos e processos de geração de soluções originárias das diferentes áreas. A tecnologia de agentes de software está evoluindo com o objetivo de reduzir problemas que possam surgir com aplicações destas áreas, facilitando o uso integrado de inovações tecnológicas em computação.

3.7 Desenvolvimento Tecnológico de Qualidade (sistemas disponíveis, corretos, seguros,escaláveis, persistentes e ubíquos)

Esta seção foi diretamente baseada no documento sobre Desafios de Pesquisa elaborado pela SBC no ano de 2006 (ref.: Portal da SBC). A Tecnologia da Informação está cada vez mais presente em nosso quotidiano. Não é preciso ir muito longe para buscar exemplos. Vários eletrodomésticos e elevadores, por exemplo, têm controle via software; carros, tratores, aviões, celulares, sistemas de controle de tráfego e salas de cirurgia também dependem desta tecnologia. Enquanto alguns desses exemplos correspondem a sistemas relativamente simples – um forno de microondas –,outros custam muito caro e envolvem milhões de linhas de código e hardware sofisticado.

Se esta ubiqüidade traz conforto, também acarreta problemas. Como dependemos desses sistemas, eles precisam estar sempre disponíveis e não apresentarem falhas; devem funcionar da forma prevista e ser escaláveis e seguros. Este desafio visa, desta forma, a pesquisa em ambientes, métodos, técnicas, modelos, dispositivos e padrões de arquitetura e de projeto capazes de auxiliar os projetistas e desenvolvedores de grandes sistemas de software e hardware a atingirem esses objetivos.

As várias propriedades abrangidas por este desafio foram agrupadas, durante o seminário, em um novo termo – computação *onivalente* – cunhado para designar a ubiqüidade associada à segurança, fidedignidade e evolução de sistemas computacionais, em especial software.

4 Novos Métodos para a Pesquisa em Engenharia de Software

O que decidimos chamar de novos métodos para pesquisa em Engenharia de Software inclui metodologias que se aplicam a todas as áreas da disciplina. São elas o uso de métodos empíricos e o uso de software livre para o desenvolvimento de software. Ambas as metodologias estão fortemente interligad**as.**

4.1 Métodos Empíricos

O nosso ponto de vista é o de que em todas as áreas da Engenharia de Software métodos empíricos de pesquisa devem alavancar o desenvolvimento de conhecimento científico sobre a medida da utilidade das diferentes tecnologias da Engenharia de Software para diferentes tipos de atores, que desempenham tipos diferentes de atividades aplicadas a diferentes tipos de sistemas. Isto é parte da visão de que este tipo de conhecimento científico vai guiar a criação de novas tecnologias em Engenharia de Software e é um insumo importante para decisões sobre a aplicação desta área na indústria.

Os principais desafios para a realização desta visão são os seguintes: (1) mais pesquisa em Engenharia de Software deveria ser baseada no uso de métodos empíricos, (2) a qualidade e a relevância, desses estudos usando esses métodos deveriam ser ampliadas e (3) deveria haver melhor síntese de evidências empíricas e (4) mais teorias precisam ser formuladas e testadas.

Os meios para alcançar estes objetivos incluem (1) aumento da competência sobre como aplicar e combinar métodos empíricos alternativos, (2) uma associação mais íntima entre a academia e a indústria, (3) o desenvolvimento de agendas de pesquisa comuns com foco em métodos empíricos, (4) mais recursos para pesquisa em métodos empíricos.

O desenvolvimento de software além fronteiras está se tornando uma vantagem competitiva importante para as indústrias de software na atualidade. No entanto, a globalização cada vez maior do desenvolvimento de software cria novos desafios para a área devido ao impacto dos diferentes fusos horários, diversidade de culturas e formas de comunicação e distância. Tudo isto impõe técnicas efetivas e inovadoras para atingir as metas de produtividade e qualidade pretendidas.

4.2 O Uso de Software Livre no Desenvolvimento de Software

O problema de se obter uma visão geral sobre o papel do software livre no desenvolvimento de software é ainda um problema em aberto [17]. Ainda são raros casos envolvendo o estudo detalhado de grandes coleções de projetos (centenas) cruzando as informações provenientes de vários tipos diferentes de repositórios ou a análise de padrões históricos em diferentes projetos no transcorrer de longos períodos de tempo.

Até o presente, os estudos tendem a utilizar pequenos números de projetos ou certos aspectos da informação sobre eles. No mundo do software livre, os mais estudados são Mozilla, Linux e Apache. Alguns destes estudos já começaram a considerar dados ao longo do tempo, realizando análises evolutivas simples. Infelizmente, ter uma visão global apresenta problemas devidos à escala, à gestão de enormes quantidades de dados com inconsistências, à informação incompleta, às imprecisões e representações diferentes.

Tais quantidades de informação não podem ser analisadas com base em técnicas estatísticas básicas como a regressão linear. Muitas correlações são claramente não lineares e, em muitos casos, a distribuição de eventos dos dados compilados ainda não é conhecida. Portanto, outras técnicas de processamento da categoria *soft computing* (tais como, redes neurais e lógica fuzzy) ou mineração de dados tornam-se necessários.

Muitos grupos têm priorizado a área de mineração de dados, projetando e implementando conjuntos de ferramentas para esta finalidade. Outros bons candidatos para análise holística no futuro são estudos amplos que investiguem a reutilização de códi-

go ou a existência de dependências técnicas entre milhares de projetos e como eles evoluem no tempo.

Com relação a ferramentas para *soft computing* têm sido desenvolvidos modelos de predição que podem descrever o comportamento de software livres de grande porte. Estas técnicas vem sendo utilizadas para verificar se a estatística linear clássica pode ajudar na extração de conclusões a partir destes dados. O objetivo de tais experimentos é produzir uma caracterização de sistemas de software baseada em vários critérios que vão além (mas incluem) a evolução do software.

5 Conclusões

Segundo [16] uma disciplina madura deve ser capaz de identificar e enumerar muitos "problemas resolvidos" (*dead problems*). Estes são os problemas que não requerem mais nenhum esforço por já terem deixado de existir. A Ciência da Computação está lentamente criando uma lista deste tipo de problema enquanto a Engenharia de Software está fazendo um esforço muito grande para encontrar os seus. Qualquer lista que seja montada para os problemas resolvidos da Engenharia de Software causará, necessariamente, muita controvérsia na comunidade da área. Por extensão a agenda de pesquisa que propusemos neste artigo também está sujeita a controvérsias. Isto não nos preocupa muito porque o objetivo da nossa agenda é motivar jovens pesquisadores e profissionais da área de Engenharia de Software. A área é relevante dos pontos de vista tecnológico e econômico e promete grandes recompensas àqueles que começarem a produzir problemas claramente resolvidos. Para os jovens com interesse em uma carreira de pesquisador o que não falta na área são problemas em aberto a espera de soluções.

Referências Bibliográficas

[1] Maria Klawe, Ben Scneiderman, "Crisis and Opportunity in Computer Science", CACM, Nov. 2005/Vol. 48. No 11

[2] S Redwine, W Riddle "Software technology maturation", 8th IEEE - ACM International Conference on Software Engineering, London, 1985

[3] Alessando Fabricio Garcia. Objetos e Agentes: Uma Abordagem Orientada a Aspectos. 2004. 298 f. Tese (Doutorado em Informática) - Pontifícia Universidade Católica do Rio de Janeiro

[4] Andrew D. Eisenberg, Gregor Kiczales, "Aspect-oriented software development"; Vol. 208 Proceedings of the 6th international conference on Aspect-oriented software development, Vancouver, British Columbia, Canada

[5] Charette, R. "Fair Fight? Agile Versus Heavy Methodologies", Cutter Consortium E-project Management Advisory Service, 2, 13, 2001.

[6] Cockburn, A. e Highsmith, J."Agile Software Development: The Business of Innovation", IEEE Computer, Sept., pp. 120-122, 2001.

[7] H.Dieter Rombach (Editor), Victor R. Basili (Editor), Richard W. Selby (Editor)

"Experimental Software Engineering Issues:: Critical Assessment and Future Directions". International Workshop, Dagstuhl Castle, Germany, September 14-18, 1992. Proceedings

[8] GUELFI, Nicolas; RIES, Benoît; STERGES, Paul. "*MEDAL*: A Case Tool Extension for Model-Driven Software Engineering". in: IEEE International Conference on Software – Science, Technology & Engineering *(SwSTE'03),* 2003, Israel. Proceedings…November 2003 p.33-42.

[9] Paul Clements Linda Northrop Product Line Systems Program Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213

[10] David Lorge Parnas "Software Fundamentals: Collected Papers by David L. Parnas" , Addison-Wesley Professional , April 2001

[11] Ricardo Choren, Alessandro F. Garcia, Holger Giese, Ho-fung Leung, Carlos José Pereira de Lucena, Alexander B. Romanovsky: Software Engineering for Multi-Agent Systems V, Research Issues and Practical Applications [the book is a result of SELMAS 2006]. Springer 2007

[12] Alessandro F. Garcia, Ricardo Choren, Carlos José Pereira de Lucena, Paolo Giorgini, Tom Holvoet, Alexander B. Romanovsky: Software Engineering for Multi-Agent Systems IV, Research Issues and Practical Applications [the book is a result of SELMAS 2005]. Springer 2006

[13] Ricardo Choren, Alessandro F. Garcia, Carlos José Pereira de Lucena, Alexander B. Romanovsky: Software Engineering for Multi-Agent Systems III, Research Issues and Practical Applications [the book is a result of SELMAS 2004]. Springer 2005

[14] Carlos José Pereira de Lucena, Alessandro F. Garcia, Alexander B. Romanovsky, Jaelson Castro, Paulo S. C. Alencar: Software Engineering for Multi-Agent Systems II, Research Issues and Practical Applications [the book is a result of SELMAS 2003] Springer 2004

[15] Alessandro F. Garcia, Carlos José Pereira de Lucena, Franco Zambonelli, Andrea Omicini, Jaelson Castro: Software Engineering for Large-Scale Multi-Agent Systems, Research Issues and Practical Applications [the book is a result of SELMAS 2002] Springer 2003

[16] A Finkelstein, J Kramer, Software engineering: a roadmap - Proceedings of the Conference on The future of Software, 2000

[17] Dag I. K. Sjoberg, Dag I. K. Sjoberg, Magne Jorgensen, The Future of Empirical Methods in Software Engineering Research, Proceedings of the International Conference on Software Engineering 2007 (Future of Software Engineering)