Por que Falham os Projetos de Implantação de Processos de Software?

Cássio Adriano Nunes Teixeira, Henrique Luiz Cukierman

Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, Bloco H, CT, Cidade Universitária, Ilha do Fundão, Rio de Janeiro, RJ, CEP. 21.995-970

{cant, hcukier}@cos.ufrj.br

Abstract. *Software Engineering (SE) is generally understood has having as its main practice the diffusion of "universal" standards and models but, in a great extent, its practitioners don't perceive that their performance is tutored to take this assumption as "logic" and "natural". In this paper, we approach SE as a modernist construction, with its military links, as the origin of such belief, briefly exploring: (i) what can be called a "diffusionist" SE; and (ii) an alternative approach to SE as a set of translation practices.*

Resumo. *É de uso corrente, na Engenharia de Software (ES), considerar a difusão de modelos e padrões "universais" como sua principal prática, de sorte que, na maior parte das vezes, nós, seus praticantes, sequer percebemos que nossa atuação pauta-se pela idéia de que se trata de uma prática "lógica" e "natural". Neste artigo, abordamos a construção modernista da ES, com seus vínculos militares, como a origem dessa crença, de forma a explorar brevemente: (i) o que se poderia chamar de uma ES difusionista; e (ii) uma abordagem que lhe seja alternativa, a da ES como uma prática de tradução.*

1. Introdução

Este artigo começa com uma provocação já em seu título. Passa longe de nossas possibilidades trazer, nestas poucas páginas, respostas convincentes e completas sobre porque falham tantos projetos de implantação de processos de software. Contudo, podemos discutir sobre a própria existência de perguntas desse tipo e porque a engenharia de software (ES) não lida bem com elas. Em nosso ponto de vista, a resposta permeia os vínculos limitantes da ES decorrente de ser uma construção moderna, herdeira da racionalidade científica ocidental. Com um enquadramento tecnocêntrico, a ES limita, *a priori*, quais tipos de problemas, e quais tipos de soluções, podem ser considerados válidos em seu campo de atuação (TEIXEIRA, 2006).

É preciso reconhecer a ES como uma construção, bem como reconhecer o *estilo de pensamento*[1] que fundamenta essa construção, assim "desnaturalizando" a idéia de que a ES é como é em decorrência da "ordem natural das coisas". A ES se estrutura em torno da busca de padrões e modelos "universais" que, acredita-se, podem ser

[1] Ludwik Fleck define o *estilo de pensamento* não como um tom particular dos conceitos ou uma forma peculiar de reuní-los. "*Trata-se de uma coerção determinada de pensamento e mais ainda: a totalidade da preparação e disponibilidade intelectual orientada a ver e atuar de uma forma e não de outra. A dependência de qualquer fato científico*[e tecnológico] *ao estilo de pensamento é evidente*." (FLECK, 1986, p.111).

difundidos para replicar as "melhores práticas", das quais, supostamente, são os porta-vozes. Problemas com a difusão dos modelos têm, muitas vezes, respostas aparentemente pré-elaboradas, e aqui voltamos à provocação do título: a falha, na maioria das vezes, é culpa dos "fatores não-técnicos", algo fora dos modelos (no caso, de processos de software). Para vislumbrarmos a existência de tais perguntas e seu padrão comum de resposta, é preciso discutir a idéia de difusão de modelos "universais", pela qual cabe ao "não-técnico" a culpa pelos fracassos (seção 3). Antes, porém, para entendermos porque a ES é difusionista, é preciso traçar sua contextualização histórica, o que ensaiamos na seção 2, apontando vínculos da ES com os projetos militares americanos. Nas seções 4 e 5, apresentamos um ponto de vista alternativo ao difusionismo que nos ajuda a criticar esse tipo de pergunta e suas respostas mais comuns, além de motivar a elaboração de sínteses alternativas, locais, brasileiras para os desafiantes projetos de software. Na seção 6, traçamos nossas considerações finais.

2. A Engenharia de Software, uma construção moderna

Dificilmente depreenderemos o estado atual da ES se não conhecermos seu passado. A ES como um produto da tecnociência do século XX, faz parte de uma "máquina de guerra" (LATOUR, 2000, p.282). Seu desenvolvimento, tal qual o da própria computação, está associado aos bilionários projetos de defesa norte-americanos durante a Guerra Fria — os "BIG-L". Uma resposta dos EUA à ameaçadora existência de mísseis balísticos intercontinentais soviéticos. Precursor e padrão para vários outros, o projeto 416L da Força Aérea criou o sistema SAGE — *Semi-Automatic Ground Environment* — nos anos 1950. O SAGE capitaneou o desenvolvimento de tecnologias extremamente importantes na computação, tais como memória de núcleo magnético, terminais de vídeo, canetas óticas, a primeira linguagem algébrica de computador, técnicas de computação gráfica, técnicas de simulação, lógica de sincronização paralela (transmissão paralela de bits, ao invés de serial), técnicas para conversão analógico-digital, e vice-versa, transmissão de dados digitais por linhas telefônicas, multiprocessamento, interligação de computadores em redes, além de idéias básicas para o desenvolvimento dos conceitos de compiladores e interpretadores. (EDWARDS, 1997, p.99-102).

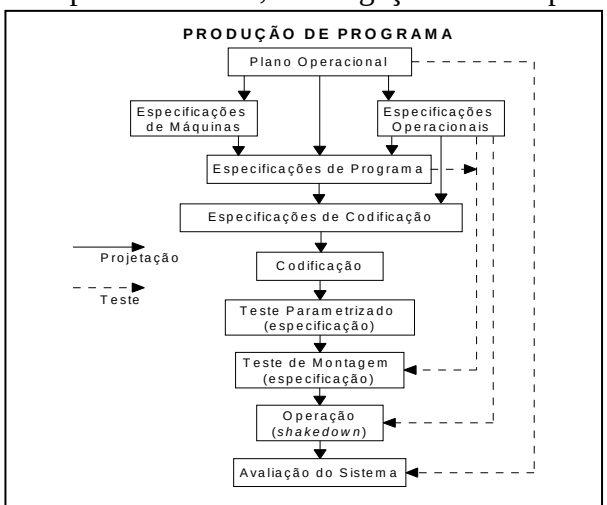

Figura 1 Gerenciamento de Projetos SAGE
(JACOBS, 1956 apud CAMPBELL-KELLY, 2004, p.68)

Mais ainda, o SAGE impulsionou o desenvolvimento da disciplina de gerenciamento de projetos de software. Técnicas de gerenciamento já maduras na engenharia foram adaptadas em uma seqüência de estágios que partia do estabelecimento inicial dos conceitos e seguia até o artefato final de programação (Figura 1). Esse esquema hierárquico, guiado por especificações, foi largamente difundido na indústria de software no final dos anos 1950, por milhares de programadores que o vivenciaram no projeto SAGE, que, de fato, formou toda uma geração de profissionais, altamente

qualificados para o desenvolvimento de software militar de missão crítica. (CAMPBELL-KELLY, 2004, p.67-69).

Em meados dos anos 1960, o poder de processamento e armazenagem de dados dos computadores havia crescido vertiginosamente e seu custo relativo havia diminuído muito. Em decorrência, passaram a existir um expressivo número de instalações civis de computadores, nos EUA, e uma explosiva demanda por software. Na década anterior, como praxe de mercado, os fabricantes de computadores disponibilizavam "sem custo" para as organizações usuárias, além do software básico para a utilização dos computadores, vários aplicativos que suportavam atividades dos maiores ramos de negócio, como o setor bancário, de seguro, industrial e de varejo. O custo do desenvolvimento desses aplicativos era considerado uma despesa de *marketing* pelos fabricantes, uma condição necessária para a venda de hardware, este sim seu verdadeiro negócio na época. No entanto, a partir de meados dos anos 1960, o custo relativo do software começou a se tornar mais expressivo do que o custo do hardware e o fator preponderante nos custos, à medida que também crescia explosivamente a demanda por programadores, passou a ser a mão-de-obra[2] (CAMPBELL-KELLY, 2004, p.89-98).

> A competição por programadores elevou os salários tão rapidamente que a programação tem provavelmente se tornado a mais alta remuneração de um posto na área tecnológica do país [EUA]. ... Mesmo assim, algumas companhias não conseguem encontrar programadores experientes a preço algum. (BYLINSKY, 1967 apud ENSMENGER, 2002, p.142).

Começou a ser questionado, como inspiração para o desenvolvimento de software, o exemplo do processo de produção de hardware, com ênfase em revisão e verificação exaustivas das especificações antes do processo de produção[3]. Os artefatos de software, aparentemente muito mais fáceis de alterar, permitiam uma abordagem de "tentativa e erro" — *code and fix* —, além do que produzir especificações precisas de software era (e ainda é) muitíssimo mais complicado do que para hardware. Além disso, a postura libertária dos anos 1960, questionadora do regime centralizado de autoridade, usualmente fazia com que os programadores seguissem métodos próprios em detrimento dos métodos que suas empresas tentavam utilizar. Surgia a "cultura *hacker*", a figura do programador herói que com expedientes inexplicáveis, na última hora, conseguia cumprir os prazos. "Código espaguete" em profusão começou a ser gerado, trocando o aparente sucesso dos projetos da década anterior pelos altos custos com retrabalho e casos de fracasso (BOEHM, 2006). A expressão *crise do software*[4], cunhada na famosa conferência da OTAN de 1968 (NAUR, 1969), sintetiza bem o que ocorria.

---

[2] Edwards (1997, pp.247-249) relaciona a falta de programadores com o desenvolvimento das linguagens de programação de alto nível. Os primeiros programadores eram basicamente os matemáticos e engenheiros que projetavam e construíam os computadores. Para eles, prevalecia a estética matemática de concisão como norma de elegância, ainda mais sob a injunção dos escassos recursos das máquinas. Linguagens de programação de alto nível, facilmente apreendidas por não especialistas, eram necessárias para viabilizar os projetos militares, como o SAGE, e a própria indústria da computação.

[3] A influência da engenharia do hardware é compreensível. Além dos primeiros programadores serem os próprios engenheiros que construíam os computadores, a maioria dos cursos de computação surgiram nos departamentos de engenharia, e, por isso, tendiam a ser mais direcionados para a máquina do que para a produção de software (ENSMENGER, 2002, p.143). Essa influência está manifestamente presente até nos nomes nas principais sociedades de profissionais de software, como, por exemplo, *Association for Computing **Machinery*** e *IEEE **Computer** Society* (BOEHM, 2006, grifos do autor).

[4] O termo (*gap* ou) *crise do software* refletia a crença na dificuldade de se escrever, com correção, softwares inteligíveis e passíveis de validação precisa, em decorrência do crescimento de sua

Surgia a percepção de que as organizações precisavam reduzir sua dependência das habilidades dos profissionais, sendo necessárias metodologias efetivas para o gerenciamento dos projetos e controle do processo de desenvolvimento de software.

> Em meados dos anos 1960, um perceptível deslocamento nos custos relativos de hardware e software ocorreu. Os custos decrescentes do hardware possibilitaram a utilização dos computadores em mais e maiores aplicações, que, por sua vez, requeriam softwares maiores e mais complexos. À medida que a escala dos projetos de software aumentava, eles se tornavam extremamente difíceis de supervisionar e controlar. Os problemas que agora afligiam os desenvolvedores de software eram mais gerenciais do que técnicos. Em uma apresentação na *Fall Eastern Joint Computer Conference* em 1965, J. Presper Eckert argumentou que a programação se tornaria algo gerenciável somente quando pudesse ser referida como "*engenharia de software*". (ENSMENGER, 2002, p.154, grifo nosso).

A aplicação da metáfora da engenharia ao desenvolvimento de software tinha um apelo muito forte. Prevalecia a imagem de que projetos complexos como o SAGE, que "*alcançou suas especificações com aproximadamente apenas um ano de atraso*" (BOEHM, 2006), eram bem sucedidos em função do uso de princípios de engenharia, como relata Herbert Benington, um dos envolvidos com o SAGE:

> É fácil para mim apontar um único fator que garantiu nosso relativo sucesso: todos nós éramos engenheiros e tínhamos sido treinados para organizar nossos esforços junto a linhas [de produção] de engenharia. (BENINGTON, 1956 apud BOEHM, 2006).

O uso mais importante da frase de Eckert ocorreu na citada conferência da OTAN. Seus organizadores escolheram como título "'*engenharia de software*' ***para serem provocativos***" (NAUR, 1969, p.15, grifo nosso) e, desde então, essa metáfora vem alinhando esforços e interesses em infundir a disciplina da engenharia no processo de desenvolvimento de software. Inúmeros atores passaram a adaptar idéias de outras engenharias ao desenvolvimento de software. Houve um grande entendimento do papel da abstração e separação de conceitos, foram introduzidas as noções de modularidade e ciclo de vida de software, processos, medições, especificações abstratas e notações para registrá-las (LEVESON, 1997). A construção da disciplina de ES, uma *construção sociotécnica* (TEIXEIRA; CUKIERMAN, 2005), segue até hoje, transformando progressivamente a metáfora de Eckert de proposta abstrata em realidade fatual.

Todas as engenharias, e não seria diferente com a ES, tomam a matemática como sua linguagem natural (TOMAYKO, 2002, p.66). De fato, o conceito de software como um produto de engenharia, como um objeto matemático, encontrou livre curso nos primeiros 50 anos da história do desenvolvimento de software (LEVESON, 1997). Junto com as ciências naturais, a matemática passou a balizar o que conta ou não como conhecimento no ocidente, através da racionalidade científica ocidental. Elementos da visão de mundo moderna são facilmente resgatados na literatura da ES, com destaque para a busca de modelos universais, representativos de sistemas ordenados. A ES devota grande importância à busca por padrões e modelos influenciada pela própria busca da ciência por padrões, modelos, fatos e leis universais, válidos em qualquer local desde que respeitado o método subjacente (HANSETH; MONTEIRO, 1998, p.133). Para uma ES moderna, modelos e métodos universais — com a presumida competência de

---

complexidade ser muito mais acelerado que o progresso da "engenharia de software". Malgrado os grandes avanços, as demandas estariam além da capacidade, teoria, técnicas e métodos da época, o que supostamente traria um sombrio cenário futuro para a indústria de software e para a própria sociedade, cada vez mais dependente de artefatos de software (NAUR, 1969, p.17, p.121).

encarnarem as "melhores práticas"— garantiriam, *per si*, o sucesso dos projetos, requerendo, para isso, serem difundidos nos locais onde ela é praticada. Esses modelos e padrões facultariam aos praticantes uma ação pautada na racionalidade da ES.

3. Difundindo modelos/padrões "universais"

Hanseth e Monteiro (1998, p. 3-6) mostram que a influência recíproca entre as tecnologias de informação/computação e de comunicação é antiga. Hoje em dia, sua convergência é tal que o termo TI — Tecnologia de Informação — tem cedido lugar para TIC — Tecnologia de Informação e Comunicação. Nas telecomunicações e na computação, a cultura de busca por padrões "universais" é muito forte. No entanto, além de apenas umas poucas companhias, muitas vezes monopólios, estarem envolvidas com as telecomunicações, a funcionalidade básica dos sistemas de telecomunicações estão estabilizadas há mais de 100 anos (discar, falar, desligar), o que favorece o estabelecimento de padrões "universais". Todavia é muito diferente a situação no desenvolvimento dos sistemas de informações, virtualmente capazes de suportar qualquer atividade de negócio, com uma infinidade de funcionalidades, em ambientes de atuação extremamente dinâmicos. Se, dada a estabilidade da funcionalidade básica do telefone, o usuário tornou-se irrelevante para o projetista, para os sistemas de informações o quadro é radicalmente distinto, a ponto das questões tecnológicas poderem ser menos complicadas do que, por exemplo, as culturais e organizacionais. Esse fato não é levando em conta pela ES tecnocêntrica que se vem consolidando. Uma ES que tenta estabelecer o desenvolvimento de sistemas de informações (a maioria extremamente baseada em software) como uma disciplina ordinária de engenharia.

Com a valorização dos padrões e modelos "universais", os projetos de software, sejam eles desenvolvimento / implantação de sistemas ou melhoria/implantação de *processos de software*[5], são guiados sob uma visão que divide, nitidamente, o conteúdo técnico (leia-se: o modelo "universal" a ser implantado) do contexto social / organizacional, de implantação. "Fatores não-técnicos", exteriores ao conteúdo (ou seja, fora dos enquadramentos propiciados pelo modelo "universal"), são reconhecidos como determinantes no sucesso dos projetos e, por isso, deve estar garantido que preexistam em uma configuração apropriada no contexto (Figura 2).

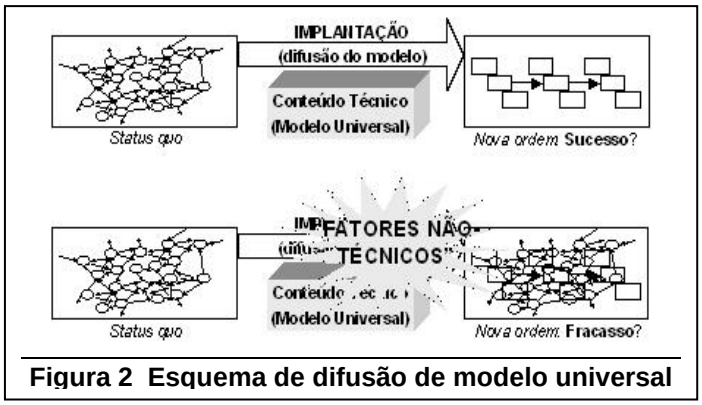

Figura 2 Esquema de difusão de modelo universal

As narrativas usualmente encontradas na literatura da ES baseiam-se em uma linearidade explicativa que não contempla, na maioria das vezes, a complexa dinâmica que se pode observar nos projetos de implantação de processos de software. Pressupondo que os projetos se submetem a um controle hierárquico, tais narrativas, em linhas gerais, apresentam uma fase de planejamento, onde é definida a estratégia a ser

[5] Um *processo de software* é um conjunto de atividades, e seus resultados associados, que produzem um produto de *software* (SOMMERVILLE, 2004, p.8).

seguida; uma fase de preparação do contexto, onde os vários "fatores não-técnicos" — como patrocínio, motivação, envolvimento —, por premissa, devem ser estabelecidos; e, por último, uma fase de definição/implantação propriamente dita do processo de software[6]. No final do projeto, os resultados subsidiam o julgamento acerca do sucesso ou fracasso obtido.

Malgrado o esforço para estabelecer nitidamente o fracasso ou o sucesso dos projetos, essas narrativas transparecem a grande assimetria que carregam, qual seja, a de partir de bases analíticas diferentes para explicar sucesso e fracasso. Nos casos de sucesso, os méritos são creditados à excelência técnica do modelo "universal" implantado e/ou de seu "plano de implantação". Contudo, para os casos de fracasso, os ditos "fatores não-técnicos" — cultura, política, lutas de poder —, supostamente não pertencentes ao escopo dos modelos "universais", prontamente são resgatados como raiz explicativa do malogro, eximindo por completo o modelo e seu "plano de implantação" de quaisquer responsabilidades pelo fracasso[7]. A seguir, utilizaremos a explicação de Bruno Latour (2000, p.218ss.), quando descreve o que chama de *modelo de difusão* de fatos e artefatos, para discutirmos os mecanismos relacionados à dinâmica de implantação dos modelos "universais". Tomaremos como exemplo de modelo "universal" um modelo de melhoria de processo de software, tal como o CMMI[8].

Estabelecendo o conceito de *determinismo técnico*, o modelo de difusão quer que acreditemos que fatos e artefatos, quando constituem *caixas-pretas*[9], podem se mover, progredir, difundir, até mesmo, existir sem as pessoas. É como se o CMMI pudesse existir por si só, "algo extraído da natureza", sem uma *Carnegie Mellon*, um SEI — *Software Engineering Institute* — e um DoD — *Department of Defense*. O determinismo técnico considera que a transformação de proposições e protótipos em fatos e artefatos dá-se por conta de suas qualidades intrínsecas, embutidas no momento de sua concepção por cientistas e engenheiros de gênio. Dito de outra forma, para o determinismo técnico a trajetória de fatos e artefatos independe do comportamento das pessoas e entidades alistadas pelas redes sociotécnicas que os consubstanciam e ratificam. Por este ângulo, a ES tornou-se fato por conta da afirmação original de Eckert, e não por conta de todos que vieram em seguida e escolheram obedecer à inspiração que dela advinha.

Segundo o *estilo de pensamento* dominante na ES, o engenheiro de software (pelo menos a esmagadora maioria, incluindo "todos" — do Brasil inclusive — dos países periféricos em termos de pesquisa e desenvolvimento em ciência da computação)

[6] Um exemplo pode ser visto em: VIVEIROS, S.M; et al., 2005, "Estratégia para Melhoria de Processos em Conformidade com o CMMI e o MR mps Br no BNDES. In IV Simpósio Brasileiro de Qualidade de Software, Porto Alegre/RS: PUCRS, 2005, pp.61-68.

[7] Sobre a assimetria na análise do sucesso e do fracasso do empreendimento científico e tecnológico, veja: BLOOR, D., 1991, *Knowledge and Social Imagery*. Chicago, University of Chicago Press.

[8] CMMI — *Capability Maturity Model Integration* — é um agregado de modelos para o desenvolvimento e melhoria de processos, desenvolvido pelo *Software Engineering Institute*, Univ. Carnegie Mellon, sob o patrocínio do Departamento de Defesa norte-americano (DoD) (CHRISSIS, M., et al., 2003, *CMMI: Guidelines for Process Integration and Product Improvement*. Addison-Wesley.).

[9] Caixa-preta é um fato plenamente aceito ou um objeto [artefato] não problemático. É um todo organizado que o construtor de fatos [e artefatos] quer propagar no tempo e no espaço e que dá a impressão de manter, por si só, o controle do comportamento de todos os seus componentes, assumindo ares de verdade e eficiência indiscutíveis. (LATOUR, 2000, p.216).

passa a ser mero difusor de modelos "universais" estabelecidos longe dos locais de seu exercício profissional. Toda vez que o projeto é bem sucedido, o difusionismo garante os méritos para o próprio modelo (como o CMMI), diminuindo, por conseguinte, o rico valor da prática local de projetação do engenheiro de software e reforçando o valor intrínseco "abstrato", "teórico", do processo/modelo "universal" implantado. Nos casos em que a implantação do modelo fracassa (*por que falham os projetos de implantação de processos de software?*), ou quando, uma vez implantado, não materializa os benefícios esperados, ou mesmo quando, na prática, o modelo é utilizado de uma maneira não totalmente alinhada ao que se previa, os difusionistas socorrem-se nos "fatores não-técnicos" como explicação. Para o difusionismo, o social é algo criado pela necessidade, e com o objetivo, de explicar o fracasso e a inconstância da difusão de modelos "universais". A sociedade, oferecendo diferentes níveis de resistência, é apenas um meio onde os modelos trilham seu caminho de difusão. O fracasso sempre será uma questão de resistência, passividade ou mesmo de cultura local, posto que ao modelo "universal" é sempre garantido o posto de verdade incontestável e comprovada eficácia.

O difusionismo impõe uma incoerência à ES cuja retórica, ao mesmo tempo que os exclui de seu enquadramento, valoriza os "fatores não-técnicos" reputando-os determinantes no sucesso dos projetos. Para o modelo de difusão necessariamente temos "*ciência e técnica, de um lado, e sociedade, do outro*" (LATOUR, 2000, p.233), os grupos que potencialmente podem resistir e que, portanto, precisam participar, serem motivados e envolvidos[10]. Se por definição estão em lados opostos, como um pode envolver o outro?

4. Projetando modelos/padrões localmente — traduções

Elaborar as relações entre o modelo "universal" e sua aplicação local é uma matéria típica de projetação (HANSETH, 1998, p.135). Na implantação de CMMI, por exemplo, o desafio não é apenas implantar um modelo "universal", mas sim de projetar relações, papéis, habilidades e comportamentos que se esperam dos desenvolvedores. Dominar as prescrições do CMMI talvez seja a parte simples do esforço, porquanto a grande tarefa é construir localmente uma instância desse modelo. Alternativamente ao modelo de difusão, o campo interdisciplinar dos *Estudos de Ciência e Tecnologia* (ECT) — reconhecido internacionalmente como *Science and Technology Studies* (STS) — traz o *modelo de tradução*[11]. Neste modelo, não são admitidas as diversas dicotomias

[10] Motivação, envolvimento e participação são usuais "fatores não-técnicos" citados na literatura da ES. Por exemplo: "*é necessário* [na implantação de processo de software] *envolver o time ao longo de todo o processo de mudança, entendendo suas dúvidas e envolvendo-o no planejamento do novo processo. [...]*." (SOMMERVILLE, 2004, p.680); "*Gerentes de projetos têm que resolver problemas* ***técnicos*** *e* ***não técnicos*** *através das pessoas alocadas em suas equipes da maneira mais efetiva possível. Eles têm que* ***motivar*** *as pessoas [...]. Um gerenciamento pobre das pessoas é um dos mais significantes contribuintes para o fracasso do projeto.*" (*ibid*., p.592, grifo nosso).

[11] Tradução é um conceito chave para os ECT (que chegam a ser denominados de *sociologia das traduções*) no qual os construtores de fatos e artefatos, em movimentos de ajustes mútuos e negociações de interesses com os elementos que querem alistar, buscam equivalências e coincidências, nem sempre existentes *ab initio*, de objetivos que viabilizem a associação e socialização de entidades heterogêneas para a construção e disseminação do fato ou artefato. Não se faz distinção analítica entre elementos humanos e não humanos alistados, nem delimitações nítidas entre fatores "técnicos" e fatores "sociais". O fato ou artefato produzido ("conteúdo") é indissociável de seu "contexto" de produção. Uma *rede sociotécnica* resulta do processo de tradução. Existe uma forte característica de incerteza no processo de

inerentemente convencionais como técnico x social, contexto x conteúdo, sujeito x objeto, evidenciando desta forma a impossibilidade de dissociar o "natural" ou "técnico" do "social" ou "cultural". Assim, por princípio, o modelo de tradução nega a existência em separado de “fatores sociais”, ou “fatores não-técnicos”, capazes de conformar, influenciar, dirigir ou retardar a trajetória da ciência e da técnica, supostamente puras, representadas nos modelos “universais”.

O social nunca pode ser usado como explicação para o fracasso, e, simetricamente, para o sucesso. Nunca se está diante de uma realidade dotada de uma essência, de uma existência *a priori*, cujas supostas características enumeráveis pudessem ser causas explicativas. A conformação do social e do modelo para a realidade local ocorrem simultaneamente, à medida que traduções vão urdindo uma rede sociotécnica. O social antes, durante e depois da implantação de um modelo vai se transformando, concomitantemente à transformação do próprio modelo “universal” à medida que é recriado localmente.

Via de regra, a narrativa dos “fatores não-técnicos” explicativos do fracasso estabelece-os como entidades singulares, dotadas de essência e existência prévia aos projetos de software. Tomemos o tão referenciado *patrocínio*. É de se perguntar se “o patrocínio” existe como uma instância essencial e imutável em alguma realidade organizacional, como um recurso claramente definido e delimitado. É possível, sem ambigüidades, resolvermos se um projeto de software dispôs ou não de patrocínio? Do modo como foi descrito em artigo debatido na segunda edição do WOSES (TEIXEIRA, 2006), um projeto de implantação de CMMI em uma grande empresa pública brasileira parece ter sido executado sem a existência de patrocínio, em decorrência da constante troca de gerência sênior, que por vezes sequer se inteirava do projeto, e do pouco prestígio da área de informática daquela empresa. Por outro lado, nessa mesma descrição, é possível resgatar elementos que atestam, contraditoriamente, que houve, sim, patrocínio, pois afinal o SEPG[12] — *Software Engineering Process Group* — não só não foi extinto, ainda que decorrido um extenso período com pouca materialização de resultados, como foi provido com recursos orçamentários suficientes para o projeto.

Destacamos, assim, a inconveniência de considerarmos o patrocínio (e qualquer outro “fator não-técnico”) como uma entidade essencial cuja existência ou presença no projeto contribui para o sucesso, e cuja inexistência ou ausência explica o fracasso na implantação de um processo de software. Ao contrário, devemos considerar o patrocínio como qualquer outro ator, humano ou não-humano, que precisa ser alistado na rede sociotécnica do processo de software na organização, através de traduções. A atuação do patrocínio, como dos demais atores envolvidos — inclusive os padrões ou modelos utilizados —, deve ser tal que contribua para a estabilização da rede, cujo efeito seria, este sim, reconhecido como, por exemplo, o “CMMI implantado” na empresa.

Para os ECT, um processo de software implantado numa organização seria o efeito observável da estabilização de sua rede sociotécnica, construção simultânea local de contexto e conteúdo. Desafio típico de projetação, no qual o projetista compõe com

---

tradução, pois, *a priori*, não se pode garantir que os elementos alistados agirão como esperado ou mesmo se permanecerão enredados. Maiores detalhes em Latour (2000).

[12] SEPG é o grupo encarregado de guiar a implantação e utilização de um processo de *software* (CARTER, L. et al., 2002, *The Road to CMMI: results of the first technology transition workshop*. Technical Report, CMU/SEI-2002-TR-007, Software Engineering Institute, Pittsburgh – PA).

entidades de variadas naturezas o ordenamento geral desejado, a partir da atuação individual de cada uma dessas entidades — humanos e não-humanos — na rede. A seguir, muito sumariamente, trazemos alguns elementos que ilustram a idéia de um processo de software ser o efeito de uma rede sociotécnica estabilizada em uma organização, não um modelo "universal" meramente implantado.[13]

5. **O desenvolvimento de software no BNDES dos anos 1970**[14]

A informática do BNDES — Banco Nacional de Desenvolvimento Econômico e Social — nos anos 1970 atuou com um alto nível de organização, padronização e produtividade. Os bons resultados alcançados naquela prática de desenvolvimento de software não podem, fria e simplificadamente, serem atribuídos à aplicação da ES e seus modelos "universais", mesmo porque, àquela altura, a ES apenas ensaiava seus primeiros passos como disciplina autônoma.

Com um *mainframe* IBM centralizando todo processamento, a instalação de computação do BNDES era algo de operação complexa, que envolvia diversas categorias de profissionais, tais como digitadores, preparadores de lotes, operadores, programadores e analistas de sistemas (à época com atuações bem distintas). Esses profissionais traziam uma formação muito aderente ao negócio de informática, pois virtualmente todos tinham uma formação prática nas empresas — tanto nas fornecedoras, como IBM, Burroughs, quanto nos birôs de prestação de serviço[15].

Naquela época, a instalação sequer dispunha de terminais de vídeo, de sorte que toda interação com os sistemas era feita através de papel: a entrada de dados ocorria através de cartões perfurados e a saída era sempre através de relatórios impressos. Somava-se a isso três questões adicionais. A primeira é que inexistiam sistemas de informática na empresa (uma situação de "campo verde"[16]). A segunda é que as atividades de negócio, também registradas em papel, praticamente não eram transformadas com a introdução dos sistemas de informática. A parte visível desses sistemas para os usuários (os fornecedores e/ou beneficiários das informações) também era em papel, embora os dados passassem a ser armazenados e processados no computador. As atividades preexistiam e continuavam a existir praticamente da mesma forma imediatamente após a implantação dos sistemas — bem diferente de hoje em dia —. A terceira e última questão é que tudo isso localizava-se em um cenário estável de

[13] Outro exemplo ilustrativo da idéia do processo de software ser o efeito de uma rede sociotécnica, ao contrário de ser simplesmente fruto de implantação de um modelo "universal", pode ser obtido em: TEIXEIRA, C.A.N., CUKIERMAN, H., 2005, A COBRA teve uma partitura. In I Workshop um Olhar Sociotécnico sobre a Engenharia de Software, Rio de Janeiro/RJ. Disponível em <http://www.cos.ufrj.br/%7Ehandrade/woses/woses2005/include/anais-woses2005.pdf>. Acesso em: 16 abr. 2007.

[14] Esta seção está baseada em entrevistas semi-estruturadas, com registro em áudio e duração aproximada de uma hora e meia, realizada entre junho e agosto de 2006, com 16 profissionais, de várias gerações, do BNDES. Também são utilizadas informações de diversos documentos internos, tais como PDI's, relatórios de auditorias, relatórios de atividades e manuais.

[15] Nos anos 1970, apenas começavam a existir os primeiros cursos na área de computação no Brasil. Atualmente, os profissionais formados nos cursos de graduação, na maioria dos casos portadores de grande carga teórica, trazem uma formação mais distanciada das necessidades imediatas do mercado.

[16] "Campo verde", lugar comum, não só na informática mas em diversos ramos de negócio, que serve para figurar uma área, ou situação, ainda não cultivada, tratada, construída — *greenfield land* —, em contraste a uma outra situação — *brownfield land* — onde algo já foi cultivado, tratado, construído, podendo até conter os escombros do que ali teria existido.

negócios, bem diferente do tempo presente, com sua dinâmica de mudanças muito célere e em constante aceleração. Dessas questões resultava a possibilidade de construção, *em papel*, de protótipos dos sistemas, capazes, efetivamente, de minimizar o espaço de interpretações equivocadas, por parte do analista, sobre o que o sistema *deveria* fazer, e, por parte do usuário, sobre o que o sistema *iria* fazer[17]. Logo, uma vez definidos os protótipos em papel, o analista podia atuar "isoladamente" em seu esforço de projetação do sistema.

De acordo com os *Estudos de Ciência e Tecnologia* (ECT), para o entendimento de uma ordem estabelecida, mais do que as supostas essencialidades das entidades envolvidas, o que importa é considerar a atuação que cada uma delas desempenha nas relações que estabelecem com as demais. Para que resulte um ordenamento estável da rede sociotécnica, a atuação das entidades deve obrigá-las a perfazerem entre si suas respectivas ações esperadas e manterem-se em seus lugares apropriados (LAW, 1992). Após termos listados algumas das entidades existentes no cenário considerado, vejamos exemplos de suas relações e atuações.

Naquela prática de desenvolvimento, o analista de sistemas atuava com grande autonomia. Essa autonomia era garantida por conta da atuação de outras entidades. Por exemplo, o *mainframe*, com suas restrições tecnológicas e seus altíssimos custos, justificava a otimização dos recursos computacionais, mesmo em detrimento das necessidades dos usuários, garantindo ao analista autonomia de decisões durante a projetação do sistema. Mas, para isso, era imprescindível que atuassem os protótipos, posto que as funcionalidades do sistema, passíveis de uma completa definição nos relatórios de entrada e saída de dados, materializavam de antemão exatamente a interação que o usuário teria com o sistema que seria construído.

A atuação esperada do usuário era a de: (i) participar da etapa de projetação conceitual geral do sistema; (ii) de aceitar a autonomia do analista para conceber a solução; e (iii) de utilizar os sistemas à medida que fossem disponibilizados. De fato isso ocorreu, tendo contribuído para a participação do usuário (item (i) acima) a existência de um "campo verde", caracterizado pelo total conhecimento por parte deste mesmo usuário das atividades que realiza, todas elas ainda não informatizadas. No momento em que já existe um sistema, ocorre uma expropriação de parte de seu conhecimento, *inscrito*[18] no próprio sistema. Com o passar do tempo, é comum que os usuários percam o conhecimento acerca de suas próprias tarefas automatizadas, não participando mais tão efetivamente do levantamento dos requisitos. O esoterismo do desenvolvimento de sistemas e a injunção das restrições tecnológicas e econômicas impostas pelo *mainframe* mantinham o usuário conformado com a autonomia do

[17] Hoje em dia é difícil conseguir protótipos com essas características porque a própria introdução do sistema / protótipo na organização enseja uma transformação dos processos de negócio, realimentando transformações no próprio sistema, que talvez nem esteja totalmente desenvolvido ainda. Essa questão é explicitada por James Herbsleb, quando reconhece que sistemas e organizações exercem restrições mútuas, ocasionando uma evolução conjunta, concomitante, das estruturas "técnicas" — os sistemas — e da própria organização (HERBSLEB, J., 2005, "Beyond Computer Science". In: Proceedings of 27th International Conference on Software Engineering (ICSE), St. Louis, Missouri, EUA, pp. 23-27.).

[18] Inscrições são transformações que materializam uma entidade num signo, num arquivo, num documento, num pedaço de papel. Usualmente, mas nem sempre, elas são bidimensionais, sujeitas a superposição e combinação. São sempre móveis, isto é, permitem novas traduções e articulações ao mesmo tempo que mantém intactas algumas formas de relação (LATOUR, 2001 p.350).

analista (item (ii) acima) e com a utilização do sistema "possível" de ser implementado (item (iii) acima), segundo o julgamento do analista. Uma situação socialmente aceita em decorrência da atuação dessas diversas entidades (analista, usuário, relatórios e protótipos eficazes, mainframe, atividades e negócios estáveis, etc.). Hoje em dia, que usuário aceitaria, por exemplo, uma interface deficiente porque o analista julgou que ela devia ser como ele a projetou?

E a atuação esperada do *mainframe*? Essa pergunta parece estranha por ser comum pensar apenas na atuação das entidades humanas. No entanto, os ECT tratam humanos e não-humanos com equivalência analítica. Logo, o *mainframe* tinha sim que perfazer uma atuação esperada — a de operar dentro da faixa de custo e performance aceitáveis. Para isso, eram necessárias a atuação de outras entidades, entre elas diversas especializações de funções e de profissionais. Por sua vez, os profissionais dependiam de intermediários para a consecução de quase todas as tarefas que demandavam, como, por exemplo, a simples compilação de um programa, que dependeria, pelo menos, do perfurador de cartões e do operador. Configuravam-se interesses recíprocos entre os profissionais, a organização e o *mainframe* que viabilizavam a existência/aceitação de hierarquias e pontos de controle explícitos, favorecendo, no conjunto, a atuação esperada de todos eles, para que o custo de operação da instalação fosse aceitável e as tarefas que todos demandavam chegassem a bom termo.

A rede sociotécnica de desenvolvimento de software estabilizada no BNDES dos anos 1970, poderia, simplificadamente, ser descrita como um processo com papéis segregados e estruturados hierarquicamente, controlado via especificações, bem ao estilo difusionista. No entanto, não se distingue um conteúdo (técnico) isolado de um contexto de aplicação, na medida em que ambos foram sintetizados concomitantemente no transcorrer da prática, produzindo um processo com forte lastro nas necessidades locais existentes, e, portanto, indissociável, dessa prática. O processo, ou a rede sociotécnica, que existiu deu conta das necessidades e desafios impostos, principalmente: desenvolver os sistemas e tornar a instalação operacional e produtiva.

6. Considerações Finais

Por que falham os projetos de implantação de processos de software? Influência dos "fatores não-técnicos"? Como vimos, uma resposta assim deriva de uma abordagem difusionista inerente ao estilo de pensamento que tutela a ES. Uma resposta alternativa poderia ser: porque não se trata de um esforço de "implantação", mas, sim, de projetação local do modelo envolvido. Esta perspectiva sugere um enfoque diferente para o problema, razão pela qual este artigo é também um convite ao estudo da complexa dinâmica local, situacional, existente nos projetos de software. Tal empreitada demanda reconhecer que um estilo de pensamento permite a formulação apenas de questões que consegue resolver, "naturalizando" a existência de um "dentro" e um "fora" da disciplina em questão — em nosso caso, a engenharia de software "com seus influentes fatores não-técnicos que lhe são externos". Com a alternativa das redes sociotécnicas, não se divide, *a priori*, o que está "dentro" e o que está "fora", pois tem-se um instrumental para contrapor o "desejo normativo" — simplificador, fragmentador, que busca regras e leis, similaridades e universalidades — a um "desejo descritivo"— que busca o detalhe, a particularização, a descrição densa, e, fundamentalmente, a desnaturalização de modelos através da explicitação de sua historicidade. Além de favorecer o surgimento de respostas à pergunta colocada, adequadas às nossas

necessidades, essa alternativa nos instrumentaliza para problematizar acerca da formação de nossos engenheiros de software: formamos *tradutores* ou *difusionistas*?

Referências

BOEHM, B., 2006, "A View of 20th and 21st Century Software Engineering". In: 28th International Conference on Software Engineering (ICSE), Shangai, China, p.12-29.

CAMPBELL-KELLY, M., 2004, *From Airline Reservations to Sonic the Hedghog: A History of the Software Industry*, MIT Press.

EDWARDS, P.N., 1997, The Closed World: computers and the politics of discourse in Cold War America. Massachusetts, MIT Press.

ENSMENGER, N.; ASPRAY, W., 2002, "Software as Labor Process". In: HASHAGEN, U.; KEIL-SLAWIK, R.; NORBERG, A. (eds), *History of Computing: software issues*, Berlin, Germany, Springer-Verlag, p.139-165.

HANSETH, O., MONTEIRO, E., 1998, *Understanding Information Infrastructure*. Manuscript. Disponível em <http://heim.ifi.uio.no/~oleha/Publications/book.pdf>. Acesso em: 01 abr. 2005.

LATOUR, B., 2000, *Ciência em Ação: como seguir cientistas e engenheiros sociedade afora*. São Paulo, UNESP.

LATOUR, B., 2001, *A Esperança de Pandora: ensaios sobre a realidade dos estudos científicos*. Bauru, SP, EDUSC.

LAW, J., 1992, "Notes on the Theory of the Actor-Network: Ordering, Strategy, and Heterogeneity", Systems Practice, v.5, n.4, p.379-393

LEVESON, N.G., 1997, "Software Engineering: Stretching the Limits of Complexity". In: Communications of the ACM, v. 40, n. 2, p.129-131.

FLECK, L., 1986, *La génesis y el desarrollo de un hecho científico*. Madrid, Alianza.

NAUR, P.; RANDELL, B. (eds.), 1969, *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee, Garmish, Germany, 7th to 11th October 1968*. Brussels, Scientific Affairs Division.

SOMMERVILLE, I., 2004, *Software engineering*. 7th ed., Addson-Wesley.

TEIXEIRA, C.A.N., CUKIERMAN, H., 2005, Apontamentos para Enriquecer o Perfil do Engenheiro de *Software*. In XXV Congresso da Sociedade Brasileira de Computação, São Leopoldo/RS. Disponível em <http://www.unisinos.br/_diversos/ congresso/ sbc2005/_dados/anais/pdf/arq0026.pdf >. Acesso em: 17 abr. 2007.

TEIXEIRA, C.A.N., 2006, Algumas observações sobre os vínculos entre a Engenharia de Software e o pensamento moderno. In II WOSES, Vila Velha/ES. Disponível em <http://www.cos.ufrj.br/~handrade/woses/woses2006/>. Acesso em: 17 abr. 2007.

TOMAYKO, J.E., 2002, "Software as Engineering". In: HASHAGEN, U., KEIL-SLAWIK, R., NORBERG, A. (eds), *History of Computing: software issues*, Berlin, Germany, Springer-Verlag, p.65-76.