PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 13/03

Acompanhamento de Projetos

Arndt von Staa

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

PUC RIO - DEPARTAMENTO DE INFORMÁTICA ISSN 0103-9741

Monografias em Ciência da Computação, Nº 13/03
Editor: Carlos J. P. Lucena Maio, 2003

Acompanhamento de Projetos

Arndt von Staa

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
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br

Acompanhamento de Projetos

Arndt von Staa
arndt@inf.puc-rio.br

Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)
Rua Marquês de São Vicente 225, Gávea
22453-900 Rio de Janeiro, RJ, Brasil

PUC-Rio.Inf.MCCxx/aa Mês/Ano

Abstract: In this article we examine some of the more relevant features of project tracking tools. It is not our intention to design the best possible tool, if such a thing exists. Rather we aim at providing a set of guidelines that may help a software development manager to choose among available tools. Of course the sketches may also be used as the starting point of a tool development project.

Following three tools will be outlined: time sheet; problem recording and tracking; and configuration item registering and tracking. The three tools can be integrated. If integrated, they help to reduce the effort to acquire tracking data. In this way we expect that more reliable project tracking data could be gained. An interesting side effect is the possibility to gather data that helps finding the true causes of quality and productivity problems.

Keywords: Artifact registering, Configuration management, Problem tracking, Software development plan, Software process, Time sheet.

Resumo: Neste artigo procuraremos identificar as propriedades mais relevantes das ferramentas de acompanhamento de projetos de desenvolvimento de software. Não pretendemos especificar o melhor conjunto de ferramentas possível, se é que isto existe. Queremos somente caracterizar propriedades que ferramentas adequadas deveriam ter. Ou seja, os esboços poderão ser utilizados como critérios de avaliação de ferramentas existentes. Embora não seja objetivo principal, é claro que, além de servir como critérios para a escolha de ferramentas, os esboços poderão ser utilizados como ponto de partida para o desenvolvimento de ferramentas de apoio ao acompanhamento de projetos de software.

Serão esboçadas três classes de ferramentas: a folha de tempo, o acompanhamento de problemas e o registro e acompanhamento de itens de configuração. Estas três ferramentas podem ser integradas. Através da integração pode-se reduzir o esforço de aquisição de dados, aumentando, assim, a chance de se ter um acompanhamento fiel do desenvolvimento em progresso. Em muitas ocasiões a coleta de dados também permitirá identificar as causas sistêmicas dos problemas de qualidade e produtividade.

Palavras chave: Acompanhamento de problemas, Folha de tempo, Gerência da configuração, Plano de desenvolvimento de software, Processo de desenvolvimento de software , Registro de artefatos.

1. Introdução

É lugar comum iniciar artigos que tratam de engenharia de software dizendo que o desenvolvimento de software está em crise, uma vez que um número considerável de projetos, ou não concluíram no prazo ou não produziram sistemas com qualidade tal que possam ser utilizados a contento. Para substanciar esta afirmação, cita-se um percentual de projetos mal sucedidos extraído de algum artigo, sem questionamento algum quanto à época em que a estatística foi coletada, a forma de sua obtenção e a natureza das organizações provocadoras da catástrofe. Seguem-se propostas para a solução do problema[1]. Infelizmente estas propostas são exatamente isto: propostas, uma vez que é raro terem sido avaliadas com o cuidado que merecem.

Existem autores que contestam essa visão calamitosa. Hoje é difícil encontrar alguma atividade humana que não dependa de alguma forma de software. A rede de distribuição de energia, as redes de televisão e telecomunicações, os bancos, os aparelhos de monitorização de pacientes, os tomógrafos, os aviões, os trens, os carros, os aparelhos auditivos digitais, os controles remotos e uma miríade de artefatos ou sistemas, todos são computarizados. Não somente isto, software tornou-se uma indústria que movimenta centenas de bilhões de dólares ao ano. Ora se software fosse tão ruim assim, isso tudo não seria possível. Por outro lado, não há dúvida que existem problemas, alguns de longa data, carecendo de solução adequada até hoje.

Boehm e Basili [BB 2001] mencionam que fazem mais de 15 anos que se recomenda identificar faltas o mais cedo possível, uma vez que o crescimento do custo de sua remoção cresce com expoente maior do que 1 com relação ao tempo decorrido entre o instante em que ocorre o erro que introduz a falta e o instante em que ela estará completamente removida. Estes mesmos autores dizem que a remoção de faltas detectadas após a entrada em serviço podem custar até 100 vezes mais do que o que custaria a remoção durante o desenvolvimento. Na realidade pode-se dizer que fazem mais de 30 anos que se recomenda a prevenção de faltas, pois já por volta de 1969 era proposto o uso de prova da corretude de programas como um instrumento de prevenção de faltas [Floyd 1967, Hoare 1969].

Para muitos o problema de produzir sistemas de qualidade satisfatória[2] reside na forma de gerenciar projetos. Tem autores que afirmam que, utilizando processos apropriados, é perfeitamente possível conseguir-se realizar um número expressivo de projetos dentro dos orçamentos estipulados e produzindo resultados com qualidade satisfatória [BABCCHMRS 2000, Humphrey 1989, Humphrey 1995, Humphrey 2000, PWCC 1995]. Esmiuçando-se as explicações de todos esses autores, temos que, na realidade, são 3 os principais fatores: gestão, disciplina e proficiência da equipe.

O emprego de pessoal sem adequada formação, sem oferecer treinamento local nos padrões e processos da organização, sem dar oportunidade de aperfeiçoamento contínuo, e a própria inexistência de padrões e processos devidamente documentados, inegavelmente é um fator que contribui para a "crise" do software [Cockburn 2001]. Na realidade não é crise do setor de software, é, sim, crise na organização que utiliza pessoas sem adequada proficiência por

---

[1] Há muito [Brooks 1986] afirmava não acreditar que alguém fosse capaz de produzir a bala de prata (leia-se tecnologia ou metodologia) capaz de matar o lobisomem (leia-se a "crise" de software).

[2] Um artefato que possui ***qualidade satisfatória*** é adequado e não possui faltas nem deficiências do ponto de vista dos usuários, clientes e desenvolvedores. Tampouco possui excesso de qualidade encarecendo inutilmente o artefato.

uma razão de falsa economia. É crise de organizações que não investem em padrões, na explicitação dos processos que usam e nem na memória quanto à realização de projetos passados. É o tipo de economia que acaba saindo cara.

O programador, ou analista, que abomina padrões, que adora trabalhar em horários não convencionais, que altera artefatos sem avisar os outros, que considera especificações serem de pouca utilidade, que não procura entender o porquê das coisas que está fazendo ou usando, é outro gerador de crise. Indisciplina há muito vem sendo citada como uma das grandes causas para a "crise" de software. Artigos publicados na década de 70 já apontavam isto. Existem até propostas tipo *Cleanroom* [MDL 1987] em que a imposição de disciplina é levada a um extremo. Mesmo a atualmente tão badalada *Extreme Programming* [Beck 1999, Wake 2001], apesar de parecer uma volta aos áureos tempos da programação heróica, depende fortemente de disciplina de trabalho e obediência a padrões e a processos estabelecidos.

Existem vários tipos de gerentes de projeto. Caracterizarei somente 3. O mais danoso é o *ditador*. Este tipo de gerente determina quem fará o que, quanto tempo poderá utilizar para fazê-lo, quantos recursos poderá utilizar, etc. Quem faz as estimativas é ele. Tipicamente utiliza sistemas de planejamento e controle de projetos, gera belíssimos diagramas de Gantt que ninguém leva a sério, quase invariavelmente não termina dentro de prazos ou custos, pois assumiu compromissos para "agradar o chefe" ou o setor de marketing. Outra categoria de gerente é o *coordenador*. Este ouve os que realizarão as tarefas e dá a eles a autoridade para o dimensionamento de prazos e necessidades de recursos. Acompanha a execução e promove revisões das estimativas sempre que observar desvios sistemáticos ou significativos. Também utiliza diagramas de Gantt mas que passam a ser levados a sério pois representam compromissos viáveis assumidos pelos desenvolvedores. Este tipo de gestão é preconizada por CMM e outros [PWCC 1995, FSM 1998]. Os projetos costumam ser realizados dentro dos prazos e custos estabelecidos na última revisão dos planos. Finalmente, o terceiro é o *facilitador*. Este se preocupa em facilitar e agilizar o trabalho. O estabelecimento de prazos e necessidades de recursos está estritamente ao cargo dos desenvolvedores. Para reduzir os riscos utiliza-se o desenvolvimento incremental, em que cada incremento requer de 3 a 6 meses para ser completado. Esta forma de gestão é proposta pelas metodologias ágeis [Cockburn 2001, Highsmith 2002], em particular por *Extreme Programming* [Beck 1999, BF 2000] e têm a virtude de serem altamente adaptáveis às mudanças que vão surgindo no decorrer do projeto. Também aqui prazos e recursos são respeitados, uma vez que o escopo de cada incremento será diminuído sempre que se observar que não será possível atingir a meta estabelecida.

Evidentemente o segredo para o acerto das estimativas está nas revisões dos planos e/ou do escopo do projeto e que ocorrem com alguma freqüência. Mas para isto é necessário um sistemático acompanhamento do desenrolar do projeto, confrontando continuamente o que está sendo realizado com o que se esperava que fosse realizado conforme registrado no plano.

No restante deste artigo examinaremos em mais detalhe aspectos relacionados com o acompanhamento de projetos de desenvolvimento de software. Na seção 2 são resumidamente abordados o que se entende por plano, processo definido do projeto e meta-processos. Na seção 3 são abordados os conceitos básicos de acompanhamento de projetos. Na seção 4 são esboçadas as ferramentas folha de tempo, registro e acompanhamento de problemas, e registro e acompanhamento de itens de configuração. A seção 5 conclui o artigo.

2. Planos, processos definidos e meta-processos

Ao desenvolver software lidamos com duas grandes categorias de conceitos: processos e artefatos. Um ***artefato*** é qualquer resultado tangível (existe sob a forma de papel ou arquivo de computador), composto ou não, fruto da realização de uma ou mais atividades. São exemplos: módulos de programas, manuais, arquivos contendo auxílio on-line, especificações, planos, *logs* de teste, cadernos de anotação de problemas, folhetos de propaganda, bases de dados históricos, etc.

Artefato é um conceito recursivo. Assim o sistema como um todo é um artefato. Os programas, manuais, bases de dados etc. que o constituem são também artefatos, e assim por diante. No nível elementar um artefato é um arquivo (por exemplo: `xxx.java`) ou documento (por exemplo: o caderno de anotação de problemas). O objetivo de um projeto de desenvolvimento é, portanto, criar um conjunto harmonioso[3] de artefatos que juntos implementam satisfatoriamente a aplicação almejada.

Um plano de desenvolvimento pode ser visto como um programa que será executado por diversas pessoas ao invés de sê-lo por um autômato. Nas disciplinas de sistemas de computação aprende-se que é bastante complexa e sujeita a panes a programação de sistemas distribuídos (vários processadores cooperando para realizar um determinado cálculo). É exatamente isto que é um plano – um sistema distribuído –, com pelo menos mais três agravantes:

i. não poderá ser testado antes de ser posto em "uso". Diferente de um programa, um plano será executado exatamente uma vez e, nesta única vez, tem a obrigação de dar certo. Isto é, deve produzir o sistema desejado com qualidade satisfatória e dentro dos prazos e custos estimados. Quem programa sabe como é difícil escrever programas complexos que estejam corretos antes do primeiro teste.

ii. o executor não é um autômato e, portanto, é difícil, se não impossível, estabelecer a priori a confiabilidade dos resultados, tampouco é possível determinar com exatidão a necessidade de tempos e de recursos;

iii. o plano é alterado durante a sua execução em função do conhecimento adquirido ao executá-lo (mudanças de requisitos, mudanças de arquitetura, etc.), e de problemas[4] observados durante a sua execução (erros sistemáticos de estimativas, rotação de pessoal, etc.). Além disso projetos são sujeitos a muitos imprevistos (falta de energia, quebra de equipamento, doenças, etc.). Note que uma das primeiras recomendações em disciplinas de engenharia de software é que os programas não devem modificar a si próprios, pois tornam virtualmente impossível saber, através de inspeções, se poderão produzir o resultado desejado. Mas é exatamente isso o que acontece com planos.

É claro que isso mostra que planejar o desenvolvimento de um determinado software é quase uma missão impossível, se não forem tomadas algumas precauções.

Uma das precauções é a de desenvolver um meta-plano a ser instanciado, produzindo um plano de desenvolvimento de uma aplicação. A instanciação se dá com base nas características do ambiente de desenvolvimento e da aplicação a desenvolver. Utilizando a terminologia de CMM [PWCC 1995, FSB 1998] um tal meta-plano é o *Processo de Software*

---

[3] Um ***conjunto harmonioso*** de artefatos é formado por artefatos que não guardam contradições, conflitos ou inconsistências entre si.

[4] Entendemos por ***problema*** coletivamente qualquer relato de falha, relato de defeito ou solicitação de adaptação, melhoria ou evolução.

Definido do Projeto. Este é um processo de software estabelecido e bem caracterizado visando determinados domínios de aplicação[5] e/ou de solução[6], descrito em termos de padrões, procedimentos, ferramentas e métodos de desenvolvimento de classes específicas de artefatos.

As definições de processos podem variar desde um simples esquema (estrutura de tarefas genéricas) até uma complexa máquina de estados indicando todas as tarefas e as respectivas pré e pós-condições, todos os tipos de artefatos a serem produzidos e respectivos critérios de aceitação, todas as ferramentas a serem utilizadas, todos os padrões a serem respeitados e todos os papéis a serem desempenhados pelas pessoas que realizarão as tarefas.

Processos são instanciados caso a caso, estabelecendo o plano de desenvolvimento de um projeto específico. Pelo fato de estarem documentados e amparados em dados históricos, os processos reduzem significativamente os riscos de produzir planos incompletos ou não realizáveis. Cabe ressaltar que poderão existir vários processos de software definidos em uma mesma organização. Isto decorre do fato desses processos visarem explicitamente determinados domínios.

Infelizmente, existe o risco de se estabelecer processos definidos, que sistematicamente produzirão maus planos. Para reduzir este risco pode-se utilizar meta-processos. Estes são processos genéricos que determinam as características que qualquer processo específico deve satisfazer. A instanciação de um meta-processo visando um determinado domínio resulta no processo de software definido do projeto. São exemplos de meta-processos o CMM [PWCC 1995, FSB 1998] e o SPICE [SPICE 1998]. Faz parte do meta-processo dizer como o processo instanciado será avaliado quanto à sua eficácia e eficiência. Também faz parte dizer como será evoluído caso apresente problemas ou se torne tecnologicamente defasado.

Caso o meta-processo, os processos dele derivados e os planos instanciados a partir desses processos tiverem sido gerados com cuidado e caso eles sejam sistematicamente revisados, aumenta-se a capacitação da organização que os utiliza a produzir software de qualidade satisfatória dentro dos limites de tempo e recursos estipulados. É essencialmente esta a idéia que está por trás de normas tipo CMM e SPICE.

Infelizmente, o fato de não existir uma teoria adequada para validar a eficácia destes três artefatos e do processo de derivação, faz com que nem sempre esta forma de atuar seja coroada de êxito. Os artefatos bem como os respectivos processos de instanciação são criados empiricamente a partir da experiência profissional de várias pessoas. Isto não quer dizer que não possam levar a bons resultados. Quer dizer somente que não se consegue ter *certeza* de atingir bons resultados em cada projeto. Quer dizer também que determinadas pessoas ou organizações terão mais sucesso do que outras, uma vez que o resultado depende da proficiência e de características individuais dos membros da equipe.

3. Acompanhamento

Infelizmente, a obediência a um plano de boa qualidade não assegura que o resultados, isto é os artefatos produzidos, sejam de qualidade satisfatória. Tampouco assegura um desenrolar eficiente do processo. Isto decorre do fato dos executores serem pessoas, dos planos

---

[5] ***Domínio de aplicação*** identifica a área em que se situa a aplicação, por exemplo: sistemas de controle de estoque, sistemas de comando e controle, sistema de controle de processos químicos, sistemas de registro acadêmico.

[6] ***Domínio de solução*** identifica a tecnologia utilizada para implementar os software, por exemplo: cliente / servidor; orientado a objetos; baseado em múltiplos agentes distribuídos.

evoluírem durante a execução, da distribuição de tarefas e de uma grande gama de imponderáveis. Como já foi insinuado, as causas do "mau funcionamento" podem residir tanto na execução (erros de pessoas ou instrumentos), como também no próprio plano, ou pior, nos processos, meta-processos e processos de instanciação. Isto nos obriga a conduzir freqüentes avaliações quanto à corretude dos artefatos e da execução do plano, bem como do acerto do próprio plano.

Acompanhamento é o processo de monitorar a execução de um plano. Segundo CMM acompanhar projetos tem os seguintes objetivos:

- Acompanhar os resultados e desempenhos reais confrontando-os com o plano de desenvolvimento de software.
- Tomar ações corretivas e gerenciá-las até sua conclusão, sempre que resultados ou desempenhos reais desviarem significativamente do que foi estabelecido (estimado) no plano de desenvolvimento de software.
- Assegurar que as alterações nos compromissos de software se dêem através de acordo entre as pessoas e os grupos envolvidos.

Na realidade devemos estender estes objetivos de modo que não somente os planos sejam controlados, mas também os processos e meta-processos. Temos então mais um objetivo:

- Acompanhar processos e meta-processos obtendo indicadores quanto à sua eficácia em instanciar planos e processos[7].

Ao redigir programas complexos é fortemente recomendado que sejam inseridas assertivas executáveis no código. Estas verificam, durante a execução do programa, se o estado do programa no ponto onde se encontra a assertiva corresponde a um estado válido. Caso o estado encontrado durante a execução não seja válido, é sinalizada a ocorrência de um problema. Da mesma forma o acompanhamento de um plano de projeto somente auxiliará no controle da sua execução, caso contenha indicações claras do que é esperado em pontos de controle e marcos. Estas indicações estão substanciadas no plano, devendo escalonar o desenvolvimento dos artefatos, estabelecer os prazos e recursos disponibilizados para cada tarefa, explicitar os critérios de aceitação e os métodos de verificação da qualidade dos artefatos (pré e pós condições das tarefas).

De tudo o que foi dito é fácil concluir que o acompanhamento poderá ocorrer em variados graus de granularidade, bem como pode focar uma variedade de propriedades de um plano. Isto explica a grande variedade de propostas do que seria um bom plano e de ferramentas de acompanhamento e medição existente no mercado. Explica também a superficialidade com que freqüentemente o assunto é tratado na literatura.

4. Formas de acompanhamento

Nesta seção discutiremos algumas formas de acompanhamento. Não pretendemos nem exaurir o assunto, nem definir a melhor de cada uma das formas. Pretendemos meramente raciocinar sobre a necessidade e o porquê de cada forma de acompanhamento. O objetivo é fornecer critérios para gerentes e equipes técnicas capazes de auxiliá-los ao selecionar as ferramentas e técnicas mais adequadas às suas necessidades.

---

7 Para simplificar a redação estamos utilizando o termo *processo* para denotar o Processo de Software Definido do Projeto e *meta-processo* para denotar o processo padronizado pelo CMM ou SPICE.

4.1. Folha de tempo

A folha de tempos visa a medição do esforço dos executores de um plano. Tipicamente relaciona as atividades realizadas por um executor em determinado dia. Uma ***atividade*** é qualquer ação executada por determinada pessoa. As atividades incluem todo o trabalho que o corpo técnico e gerencial realiza para desempenhar tarefas do projeto e da organização. Atividades podem ser recorrentes, bem como podem ter nada a ver diretamente com o projeto, como, por exemplo atividades pessoais, participar em treinamento, ou registrar medições em planilhas de acompanhamento. São exemplos de atividades: redigir a especificação de um módulo, redigir o código de um módulo, inspecionar um módulo, refatorar[8] um módulo, testar um módulo, redigir um capítulo de um documento, registrar tempo na folha de tempo, participar de uma reunião, explicar o sistema para usuários, resolver problemas particulares.

O registro do esforço é fundamental para poder-se estimar prazos e recursos necessários em projetos futuros. É necessário também para se verificar se existem erros de estima sistemáticos no plano. Como já foi mencionado, um plano é um artefato vivo, no sentido que é evoluído no decorrer de sua execução. No entanto, não basta registrar-se a duração das atividades, o registro deve estar relacionado com o plano de desenvolvimento. Além disso deve servir ainda de subsídio para controlar o desempenho do ambiente[9] e da equipe de desenvolvimento.

Do ponto de vista de planejamento, atividades nem sempre são de interesse, uma vez que não necessariamente conduzem a um resultado de qualidade controlada. Usualmente um plano é estabelecido em termos de tarefas[10]. Uma ***tarefa*** é uma unidade de trabalho bem definida no processo de software. O início e a conclusão de uma tarefa fornecem à gerência um ponto de controle visível do progresso do projeto. Tal como projetos, tarefas têm início e fim e não são recorrentes. Tarefas estão associadas a artefatos e possuem uma estrutura recursiva semelhante à destes. Tarefas podem ser realizadas através de uma ou mais atividades cada qual envolvendo uma ou mais pessoas. São exemplos de tarefas: produzir um módulo a ser aceito segundo algum critério explicitamente definido no plano; produzir um documento (manual) a ser aceito segundo algum critério explicitamente definido no plano.

As tarefas têm critérios de prontidão (pré-condições) e critérios de finalização (pós-condições, critérios de qualidade, padrões) [IEEE 1990]. As condições de prontidão precisam estar satisfeitas para que a tarefa possa ser iniciada. As condições de conclusão devem estar asseguradas ao terminar a tarefa. Tal como em programação, pode-se verificar se a seqüência de execução de um plano é viável comparando as condições de conclusão com as condições de prontidão de tarefas consecutivas. Os critérios de aceitação devem ter uma parcela estabelecida através de padrões publicados. A outra parcela estará vinculada às características específicas do produto sendo desenvolvido. Exemplo de critério de prontidão: especificação do módulo ABC aceita segundo o padrão X verificado através de inspeção.

---

[8] ***Refatorar*** (*refactoring*) é a atividade de reorganizar um ou mais módulos de modo que as interfaces e a estrutura interna dos módulos tornada mais manutenível. Proceder ao refatoramento aumenta a evolutibilidade do módulo [Fowler 2000]. Uma idéia similar pode ser aplicada a outros artefatos, tais como auxílio, documentação, arquiteturas e modelos.

[9] O ***ambiente de desenvolvimento*** é formado por processos, procedimentos, métodos, ferramentas, plataformas de desenvolvimento e uma variedade de bases de dados utilizadas para registrar as medições e os artefatos.

[10] Muitos sistemas de planejamento e controle de projetos utilizam o termo *atividade* para o que entendemos aqui por *tarefa*.

Exemplo de critério de conclusão: o módulo ABC, o correspondente módulo de teste específico e os arquivos de *script* de teste estão disponíveis e foram verificados segundo o padrão Y.

A distinção entre tarefas e atividades é necessária. Considere o desenvolvimento de um módulo. O código estar redigido não indica que possa ser utilizado. Tampouco ter sido testado é suficiente para indicar que possa ser utilizado. Enquanto não estiver completamente redigido, inspecionado, testado, integrado e aceito segundo os critérios estabelecidos, o módulo não existe do ponto de vista do restante do projeto. Evidentemente, a produção do módulo pode envolver várias pessoas e levar vários dias. A tarefa é então *produzir módulo W aceito segundo padrão X, a especificação Y e o teste Z.* Esta tarefa é formada por várias atividades que serão registradas nas folhas de tempo de quem tiver realizado a atividade. Note que uma atividade pode ser realizada por várias pessoas (ex. atender a uma reunião de inspeção), mesmo assim ela figurará na folha de tempo de cada uma das pessoas que participaram na atividade. Por exemplo, no processo *Extreme Programming* é exigido que programas sejam codificados por duas pessoas trabalhando juntas em um mesmo computador [Beck 1999]. É exigido, ainda, que os testes sejam redigidos antes de se iniciar o desenvolvimento dos módulos.

Ao registrar as atividades de um dia pode-se registrar ainda:

- uma descrição resumida do que foi feito e das dificuldades encontradas ao realizar a atividade. A análise deste texto livre permitirá extrair informações relevantes e recorrentes que, no futuro, deveriam ser solicitadas explicitamente. Desta forma pode-se evoluir gradualmente o sistema de folha de tempo mantendo a sua adequação às características específicas da organização. Ao aplicar GQM – *Goal Question Metric* – [BR 1988, GHW 1995] é fortemente recomendado que sistemas de medição sejam estabelecidos somente quando se sabe para que medir. Caso contrário provavelmente o esforço gasto na medição será desperdiçado uma vez que as medidas não serão utilizadas para aprimorar o ambiente.
- natureza da atividade, como por exemplo: estudo, especificação de módulo, codificação, redação dos casos de teste, realização dos testes, depuração, alteração do código, refatoração, revisão final do módulo. As possíveis naturezas devem ser estabelecidas pela gerência de projetos. Esta forma de definir atividades permite utilizar como nome da atividade o nome da tarefa, especializando esta para uma atividade através da sua natureza. Permite também realizar estudos sobre as naturezas de atividades que mais consomem tempo. Se, por exemplo, a equipe de desenvolvimento gasta muito tempo em depuração, é conveniente providenciar treinamento e, possivelmente, acrescentar ferramentas de controle da qualidade do código fonte ao acervo do ambiente. Depois, pode-se verificar se as mudanças no ambiente e no processo surtiram o desejado efeito de redução do tempo gasto em depuração.
- artefatos criados ou alterados pela atividade. De maneira geral espera-se que uma atividade somente altere os artefatos resultantes da tarefa à qual pertence a atividade. No entanto, ao desenvolver um artefato a sua interface pode ser modificada. Isto obriga um ajuste em todos os artefatos interrelacionados com o artefato em produção. Por sua vez, isto pode acarretar uma considerável propagação de alterações. Os artefatos criados ou alterados deveriam aparecer nas pós condições da tarefa. Note que isto realça a dificuldade de se estabelecer corretamente as condições de prontidão e as de conclusão.

- artefatos utilizados. Artefatos utilizados são artefatos necessários para realizar a atividade e que podem ou não ser afetados pela atividade. Por exemplo, ao redigir o código de um módulo, em princípio a sua especificação não deve ser alterada. O registro de artefatos utilizados e de artefatos alterados, evidencia uma relação de dependência entre artefatos. Manter esta relação nos bancos de dados do projeto é bastante interessante para a tomada de decisão de passos de manutenção futuros. O conjunto de todos os artefatos utilizados ou alterados deveria aparecer nas pré condições da tarefa, enquanto que o de artefatos alterados ou gerados deveria aparecer nas pós-condições da tarefa.
- artefatos podem, na realidade devem, ser reutilizados. Conseqüentemente uma tarefa poderá estar desenvolvendo artefatos pertinentes a mais de um projeto.
- conforme será discutido na seção a seguir, uma atividade também deve referenciar as FAP – Ficha de Acompanhamento de Problema – a que diz respeito [FS 1998].

A figura a seguir esboça o diagrama de classes para uma folha de tempo.

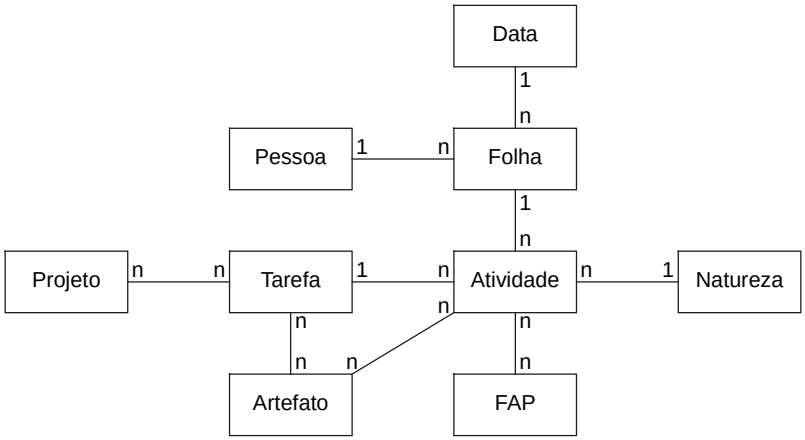

Figura 1. Esboço da estrutura de classes do subsistema folha de tempos

4.2. Acompanhamento de problemas

Uma das dificuldades que assedia o desenvolvimento de software é assegurar a integridade dos sistemas em face às contínuas alterações. Por mais que se tome cuidado ao especificar sistemas, é inevitável que estas venham a ser alteradas [Beck 1999, Lehman 1998, LB 1985], ainda mesmo durante o desenvolvimento. Isto indica que, na realidade, o desenvolvimento de software é um *processo de amadurecimento* no qual, através de várias iterações, se converge para o sistema desejado. Ou seja, problemas sempre surgirão, tornando necessário o seu registro e o acompanhamento até que tenham sido completamente resolvidos. Ao acompanhar deve-se procurar identificar as causas. Se estas forem sistemáticas deve-se alterar o processo, ou o meta-processo, de modo que em projetos futuros não mais ocorra esta classe de problemas.

Em [BB 2001] Boehm e Basili afirmam que cerca de 40 a 50% do esforço é de um projeto é consumido em retrabalho desnecessário. Infelizmente esta estatística não menciona a fração que se deve a erros de desenvolvimento, nem a fração devida às mudanças de especificação que poderiam ter sido evitadas se a especificação tivesse sido mais cuidadosa.

Nem todo o retrabalho é desnecessário. Quando for seguido um processo de desenvolvimento incremental, uma das regras básicas é manter a complexidade instantânea

a menor possível. Isto é fortemente sugerido nos processos ágeis, tais como *Extreme Programming* [Beck 1999, Cockburn 2001] e no ciclo de vida espiral [Boehm 1988]. Desta forma os artefatos são desenvolvidos visando estritamente a iteração corrente. Evidentemente que, à medida que se vai evoluindo de incremento para incremento, pode-se ter a necessidade de alterar artefatos já concluídos no passado. Neste caso é sempre recomendado proceder a uma revisão rigorosa da organização interna dos artefatos através de uma refatoração [Beck 1999, Fowler 2000]. Isto assegurará a evolutibilidade do artefato no futuro. Cabe salientar que diversas estatísticas evidenciam que manutenção sucessiva de artefatos leva à sua deterioração estrutural, a menos que se realize um esforço consciente de reorganizar a sua estrutura.

É necessário, então, controlar e coordenar as alterações e, ainda, identificar instâncias de retrabalho evitável e as respectivas causas. O simples uso de um sistema de controle versões não é suficiente, uma vez que pouco ou nada informa sobre as causas das alterações. O objetivo destes sistemas é possibilitar, sempre que necessário, a reconstrução automática das várias versões de um artefato. Mas o que queremos é registrar os problemas, identificar as suas causas e acompanhar a sua resolução até que estejam completamente resolvidos. Queremos, ainda, viabilizar o acompanhamento de problemas pelos diferentes interessados, tais como usuários, clientes, gerentes e desenvolvedores. Finalmente, desejamos que os desenvolvedores afetados pela alteração de um artefato sejam informados desta alteração, evitando o retrabalho despendido ao ajustar os artefatos conseqüentes às alterações de um ou mais dos seus artefatos antecedentes.

Problemas são registrados por algum observador. Este pode ser uma pessoa pertencente à equipe de desenvolvimento, bem como qualquer pessoa externa a ela como, por exemplo, o usuário. Problemas podem ser registrados em *logs* de teste automatizado. Neste caso o observador é formado pelos casos de teste que evidenciaram o problema.

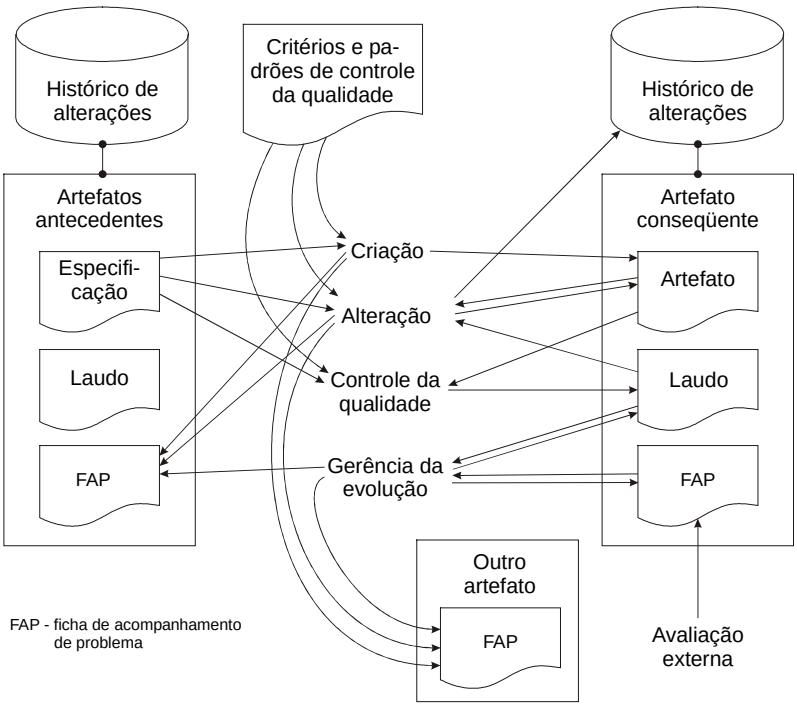

Figura 2. Atividades ao realizar uma tarefa com controle da qualidade.

A Figura 2 mostra interação das atividades de desenvolvimento e manutenção com as de controle da qualidade e de acompanhamento de problemas. Um artefato conseqüente pode

ser criado a partir de um ou mais artefatos antecedentes. Em geral estes são especificações. Ao utilizar testes automatizados redigidos antes de se desenvolver [Beck 1999] existem pelo menos dois artefatos antecedentes, a especificação (conjunto de historietas, casos de uso) e o *script* de teste. Ao criar um artefato devem ser obedecidas as especificações bem como diversos padrões ou normas[11] técnicas.

Uma vez criado o artefato, ele deverá ter a sua qualidade controlada. Este controle gera um laudo. É fortemente recomendado que o laudo seja um documento. São exemplos: logs de execução de testes automatizados, artefatos contendo anotações de inspeção, cadernos de anotação de problemas. No PSP (*Personal Software Process*) [Humphrey 1995] é exigido que sejam registrados inclusive os erros de sintaxe de programas fonte, tais como falta de ponto-e-vírgula. A justificativa é que pessoas precisam saber que têm problemas para que tomem alguma iniciativa para resolver ou evitá-los. Problemas recorrentes evidentemente merecem atenção específica, já problemas esporádicos são fruto do simples fato de humanos serem falíveis[12].

Uma vez gerado um laudo deve ser decidido o que fazer. Entre as diversas possibilidades temos:

- resolver completamente o problema através da alteração do artefato, procedendo, a seguir a um novo controle da qualidade. Evidentemente isto configura retrabalho que, de maneira geral, seria evitável caso o desenvolvimento fosse correto por construção.
- protelar a resolução do problema para uma oportunidade mais tarde. Neste caso o problema registrado no laudo é transformando numa solicitação de alteração. Todas as solicitações de alteração, relatórios de falha ou defeito, solicitações de adaptação, melhoria ou evolução são registradas em uma ficha de acompanhamento de problema – FAP – [FS 1998].
- relacionar o problema com algum dos artefatos antecedentes. Neste caso cria-se uma solicitação de alteração a ser vinculada a este artefato antecedente.
- registrar uma FAP a ser relacionada com um artefato específico. Durante o desenvolvimento, teste, avaliação, ou uso podem ser identificados problemas relacionados com artefatos de elevado nível de abstração. Por exemplo, pode-se observar um problema no produto X e não no artefato que efetivamente contém a falta causadora. Como artefato é um conceito recursivo, torna-se necessário procurar o artefato específico com o qual o problema deve ser relacionado para, somente então, iniciar a sua resolução.

Utilizando esta abordagem pode-se restringir a aceitação de um artefato ao fato do seu laudo estar vazio após o controle da qualidade. Isto não quer dizer que não existam pendências, quer dizer somente que as possíveis pendências estão registradas como solicitações de alteração e poderão ser resolvidas no futuro sem comprometer o uso do artefato neste momento.

Uma vez um artefato tendo sido aceito, cria-se uma versão imutável dele. Conseqüentemente qualquer alteração a ser feita no futuro, mesmo as decorrentes de FAPs já conhecidas mas não concluídas, gerará uma nova versão do artefato. Todas as versões de

---

[11] Padrões e normas embora similares, gozam de uma diferença relevante. Padrões são desenvolvidos e mantidos por uma ou mais organizações visando as suas necessidades específicas. Já as normas são desenvolvidas e mantidas por órgãos de normatização (p.ex. ABNT, ANSI, DIN, ISO) e potencialmente afetam todas as organizações que atuam no ramo de negócio visado pela norma.

[12] Errare humanum est, perseverare in errore est diabolicum.

um artefato, junto com uma explicação da causa do problema, são armazenadas no histórico de alterações. Este é usualmente implementado através de alguma ferramenta de controle de versão.

A Figura 3, a seguir, mostra um possível processo de tratamento de FAPs. Nesta figura os hexágonos representam estados em que a FAP espera por algum serviço. Já as elipses representam estados em que a FAP está sendo processada por alguma atividade. Os rótulos das arestas representam as condições finais das atividades, indicando qual o estado a seguir ao concluir a atividade com aquela condição.

Todas as FAPs geradas, independentemente de sua origem, entram no estado *Aguardando análise inicial*. De lá progridem para o estado *em análise*. O objetivo deste é determinar se o problema deve ser resolvido imediatamente, se deve receber um tratamento emergencial, se se refere a um problema já resolvido ou idêntico ao de outra FAP, se existe suficiente informação para poder iniciar a resolução do problema, ou, finalmente, se se trata de um "não problema" a ser simplesmente ignorado.

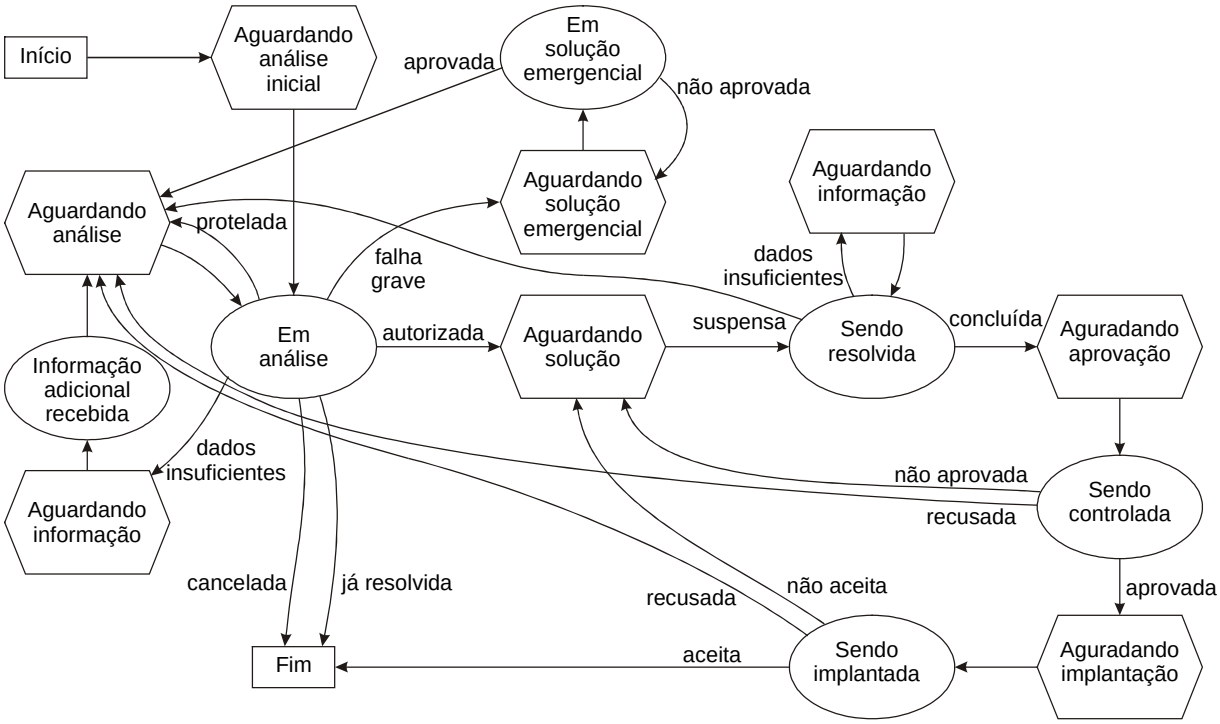

Figura 3. Processo de tratamento de uma FAP

Sempre que não possa ser dada uma solução, a FAP volta ao estado *Aguardando análise*. De tempos em tempos essas FAPs são reanalisadas. Esta nova análise leva em conta o estado corrente do projeto e corrige possíveis erros de análise no passado. Quando a FAP entra no estado *Aguardando solução* passa a ser responsabilidade da gerência selecionar os recursos a serem utilizados para resolvê-la. Desta forma estabelece-se um vínculo entre as atividades de planejamento e as de evolução dos artefatos. A solução dada a uma FAP deve sempre passar por uma aprovação. Uma vez aprovada, a nova solução pode ser implantada, ou seja, tornada disponível para os usuários e/ou os demais desenvolvedores. Especificações podem ser alteradas durante o desenvolvimento. As conseqüências se materializarão sob a forma de novos laudos evidenciando os testes que falharam com relação artefatos conseqüentes.

Em processos ágeis problemas não são documentados explicitamente. Problemas de funcionalidade e confiabilidade surgem sob a forma de laudos de teste, sendo que os testes

devem ser automatizados [Beck 1999, Cockburn 2001]. Pouco é dito quanto ao tratamento de solicitações de alteração depois da entrega do sistema. A norma ISO/IEC 15504 [SPICE 1998] requer explicitamente o registro e o acompanhamento de problemas identificados durante o desenvolvimento e após à entrega. O CMM [PWCC 1995, FSB 1998] exige o registro durante o desenvolvimento, nada sendo dito com relação aos problemas após a entrega.

Do ponto de vista dos dados manipulados, uma FAP precisa registrar:

- a pessoa que originou a FAP e a pessoa que aceitou a solução. Uma vez aceita a solução, a versão aceita do artefato é passada para a pessoa responsável por armazenar os artefatos no sistema de controle de versão. Restringir o número de pessoas autorizadas a fazer isto tem a vantagem de assegurar o cumprimento de padrões de desenvolvimento, teste e controle da qualidade.
- as versões dos artefatos resultantes da criação ou alteração realizada durante a resolução do problema identificado na FAP. É possível que várias FAPs sejam agregadas ao realizar um determinado passo de manutenção. Todas elas resultam em um ou mais artefatos criados, alterados, ou descontinuados.
- as versões dos artefatos inicialmente associadas à FAP. Esta relação facilita o trabalho de análise necessário para corretamente associar uma FAP aos artefatos que efetivamente resolverão o problema identificado.
- as formas de identificar os problemas, por exemplo: teste, inspeção, uso. Esta informação tem por objetivo auxiliar na aquisição de dados estatísticos relativos ao instante em que são identificados problemas. Evidentemente se uma taxa grande de problemas for identificada após a entrega, precisa-se tomar alguma ação de melhoria de processos para que tal não aconteça mais. Infelizmente isto é mais difícil do que se imagina. Em [BB 2001] Boehm e Basili dizem que 40 a 50% do software entra em produção contendo faltas não triviais.

Com vistas a uma integração entre o sistema de acompanhamento e as folhas de tempo é interessante que atividades relacionem as FAPs sendo resolvidas pela atividade. Desta forma a aquisição de dados de acompanhamento de esforço produzirá também as relações de interdependência entre FAPs e artefatos efetivamente alterados.

A figura a seguir esboça o diagrama de classes para o acompanhamento de problemas.

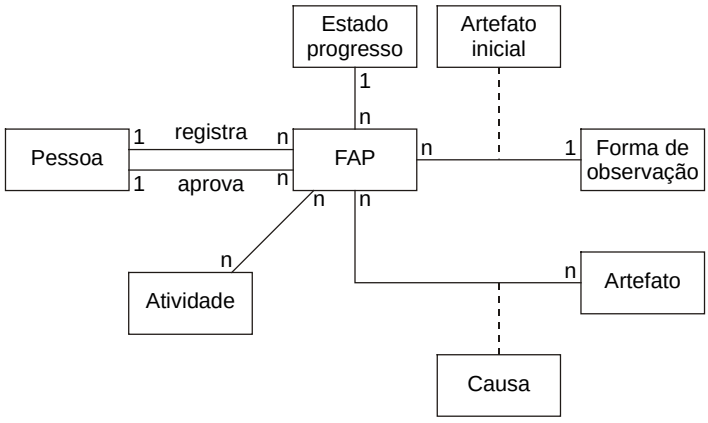

Figura 4. Esboço do diagrama de classes do subsistema de acompanhamento de problemas.

4.3. Registro de artefatos

É sempre conveniente registrar os artefatos e acompanhar a sua evolução. Ao desenvolver sistemas maiores isto se torna imprescindível. Estas ações são realizadas pela *gerência de configuração*[13]. A finalidade da GCS – *Gerência de Configuração de Software* é estabelecer e manter a integridade dos artefatos ao longo de todo o ciclo de vida do software. A GCS anda de mãos dadas com o registro e acompanhamento de problemas. A GCS envolve:

- identificar a cada momento o estado dos itens de configuração de software. Itens de configuração são artefatos de software selecionados, acompanhados de sua descrição.
- controlar sistematicamente as mudanças na configuração.
- manter, através da gerência, a integridade e rastreabilidade da configuração ao longo do ciclo de vida de software.
- controlar a integridade de artefatos compostos, levando em conta a versão de cada um dos componentes, ou seja, controlar a configuração do artefato composto.
- registrar e relatar o estado de progresso das FAPs.
- saber onde estão armazenadas, ou guardados os artefatos[14].

Os artefatos colocados sob gerência de configuração são denominados itens de configuração e incluem os artefatos que são entregues ao cliente (*produtos*), por exemplo, documentos de requisitos do software, o código executável e os manuais. Inclui ainda os artefatos gerados e utilizados no desenvolvimento mas que não são explicitamente disponibilizados para o usuário. São exemplos: projeto de arquitetura, casos de teste, módulos fonte, compiladores e ferramentas de software. Cabe salientar que são itens de configuração, entre outros, as versões de compiladores e das ferramentas CASE utilizadas. Isto se deve ao fato destes itens serem cruciais para o desenvolvimento. Em caso de acidente deve ser possível reconstruir corretamente o ambiente de modo que se evite a perda de continuidade no desenvolvimento. Finalmente, cabe salientar que nem todos os artefatos são itens de configuração.

À medida que se desenvolve o software, são identificados os itens de configuração e são estabelecidas *baselines*, para dar maior segurança ao desenvolvedor e permitir maior controle do desenvolvimento. *Baseline* é um item de configuração formalmente analisado, revisado e aprovado, que serve de base para o desenvolvimento posterior. É, portanto, um marco de referência no desenvolvimento de software, caracterizado pela entrega de um ou mais itens de configuração aprovados por meio de uma revisão técnica formalmente definida [Pressman 1995]. Os artefatos que constituem uma *baseline* podem ser alterados somente através de procedimento formal de controle de alterações [IEEE 1990]. Uma *baseline* é, então, uma versão estável de um artefato composto, contendo todos os componentes que constituem este artefato em um determinado momento.

A Figura 5 esboça o processo de gerência da configuração. Surgindo a necessidade de realizar uma alteração envolvendo um ou mais itens de configuração o grupo de pessoas responsáveis pela GCS realiza o *check-out* (retirada) dos itens requeridos, copiando-os do sistema de controle de versões onde se encontra o artefato para uma área de trabalho. É feita

---

[13] No contexto de Gerência da Configuração uma ***configuração*** é o conjunto de características físicas e funcionais de hardware e software conforme estabelecidas em algum documento técnico ou realizadas por um produto. [IEEE 1990]

[14] Nem todos os artefatos serão armazenados em algum meio computacional. Alguns serão documentos papel guardados em algum arquivo.

a alteração. Após, é realizado o controle da qualidade dos artefatos alterados. Caso estes sejam aprovados é realizado o *check-in* (incorporação) dos itens transferindo-os da área de trabalho para o sistema de controle de versões. Simultaneamente os itens são removidos da área de trabalho. Ao inserir um artefato na base de versões é criada uma nova versão deste. Assegura-se, assim, a possibilidade de recuperar qualquer uma das versões anteriores de um determinado item.

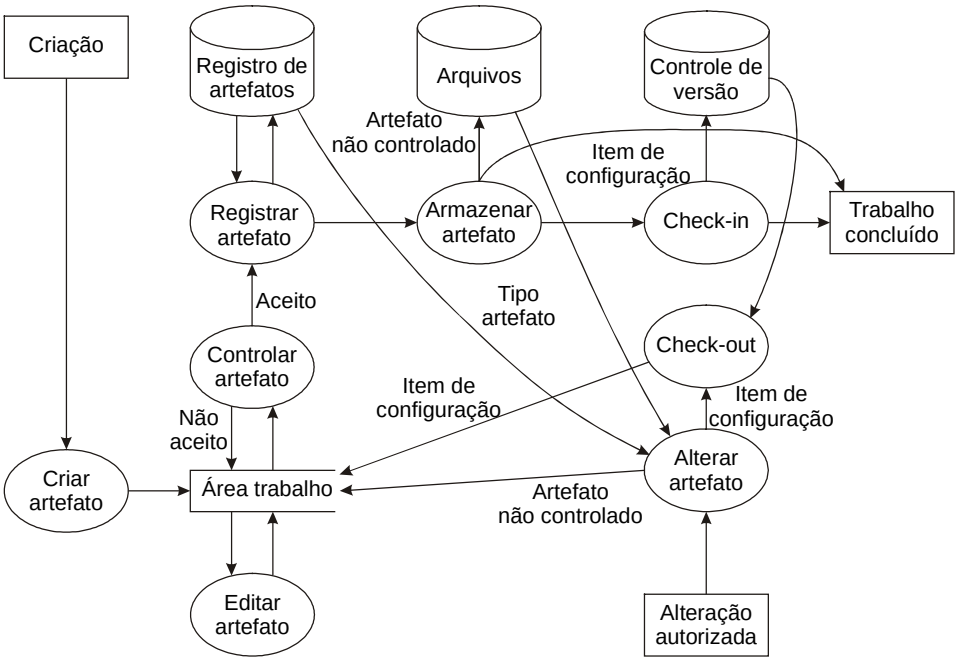

Figura 5. Esboço do processo de gerência da configuração

No caso de artefatos não controlados, o processo é simplificado desconsiderando-se as etapas de *check-in* e *check-out*. Para poder localizar os artefatos e diferenciar a sua natureza é necessário registrar os artefatos. Este registro pode conter informações complementares para facilitar a localização de um artefato caso existam muitos, ou caso seja desejado ver se é possível reutilizar algum existente.

Uma das atividades básicas da gerência de configuração é o registro de todos os artefatos, indicando aqueles que constituem itens de configuração. Ao registrar um artefato deve-se identificar todos os seus componentes, caso seja composto. Na realidade deve-se registrar, para cada versão de um artefato, as versões dos artefatos que o constituem, assim cada versão de um artefato poderá ser reconstruído a partir das versões de seus componentes.

Além de conhecer a composição de um artefato é necessário conhecer as relações de precedência. Ou seja é necessário saber quais são os artefatos conseqüentes de um determinado artefato. Isto permitirá verificar o impacto que uma alteração em um artefato pode ter sobre o sistema como um todo. Facilita, também, o rastreamento das propriedades identificadas em um dado artefato nos seus artefatos subseqüentes.

Finalmente processos modernos utilizam freqüentemente ferramentas que geram artefatos completos ou esqueletos a partir de outros. É importante também saber quais são os artefatos gerados a partir de determinado artefato e quais as ferramentas utilizadas para tal.

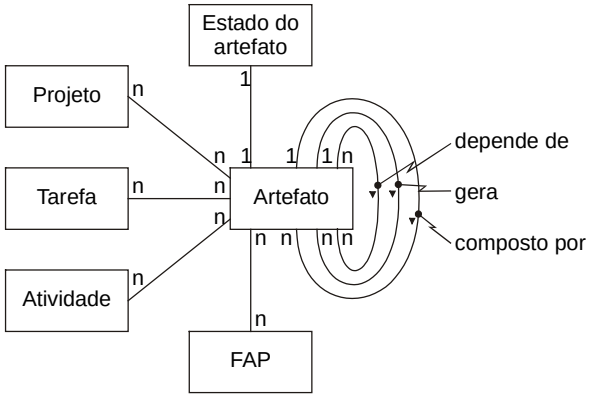

Figura 6. Esboço do diagrama de classes do subsistema de registro de artefatos.

5. Conclusão

Neste artigo procuramos identificar as propriedades mais relevantes das ferramentas de acompanhamento de projetos de desenvolvimento de software. Cabe observar, mais uma vez, que não pretendíamos especificar a melhor ferramenta possível, se é que isto existe. Queríamos tão somente caracterizar propriedades que ferramentas deveriam ter, de modo que se disponha de critério de escolha para ferramentas existentes.

Além de servir como critérios para a escolha de ferramentas, evidentemente os esboços podem ser utilizados como ponto de partida para o desenvolvimento de ferramentas de apoio ao acompanhamento de projetos de software. É claro que muitas lacunas deverão ser preenchidas de modo que se possa disponibilizar um conjunto de ferramentas adequado à organização.

Foram esboçadas três classes de ferramentas: a folha de tempo, o acompanhamento de problemas e o registro e acompanhamento de itens de configuração. Estas três ferramentas podem, e devem, ser integradas. Uma vez disponibilizadas, dever-se-ia ser capaz de evoluí-las de modo que se possa adicionar medições, bem como extrair as estatísticas relevantes para uma determinada organização. Esta forma incremental de desenvolvimento é particularmente interessante por facilitar a identificação dos problemas efetivamente vividos pela organização [BR 1988, GHW 1995, LSOHR 1998].

Utilizando um gerador de aplicações foi desenvolvido um protótipo de sistema de acompanhamento no Departamento de Informática da PUC-Rio, evidenciando a utilidade de uma ferramenta integrada, apesar da baixa qualidade de sua interface de usuário. Está em progresso o desenvolvimento de um novo conjunto de ferramentas que, esperamos, possa ser utilizado em escala maior. Esta ferramenta será eventualmente disponibilizada como *freeware*.

Referências bibliográficas

[BABCCHMRS 2000] Boehm, B.W.; Abts, C.; Brown, A.W.; Chulani, S.; Clark, B.K.; Horowitz, E.; Madachy, R.; Reifer, D.; Steece, B.; *Software Cost Estimation with COCOMO II*; Prentice Hall; 2000.

[BB 2001] Boehm, B.; Basili, V.; “Software Defect Reduction Top 10 List”; *IEEEComputer* 34(1); 2001; pags. 135-137.

[Beck 1999] Beck, K., *Extreme Programming Explained: Embrace Change*, Addison Wesley Longman, Inc., Massachusetts, 1999.

[BF 2000] Beck, K.; Fowler, M.; *Planning Extreme Programming*; Addison Wesley; 2000.

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

[BR 1988] Basili, V.R.; Rombach, H.D.; "The TAME project: Towards improvement-oriented software environments"; *IEEE Transactions on Software Engineering* 14(6); IEEE Computer Society; 1988; pags 758-773.

[Brooks 1986] Brooks, F.P.; "No Silver Bullet - Essence and Accidents of Software Engineering"; *IEEE Computer* 20(4); IEEE Computer Society; 1986; pags. 10-19.

[Cockburn 2001] Cockburn, A.; *Agile Software Development*; Addison-Wesley; 2001.

[Floyd 1967] Floyd, R.W.; "Assigning Meaning to Programs"; *Proceedings Symposium in Applied Mathematics* vol 19; American Mathematical Society; 1967; pas 19-32 (valor histórico).

[Fowler 2000] Fowler, M.; Refactoring: Improving the Design of Existing Code; Addison Wesley; 2000.

[FS 1998] Freire, F.; Staa, A.v.; "Concerto: Um Sistema de Acompanhamento da Qualidade Pós-Entrega"; *Workshop em Qualidade de Software*; Maringá, PR; SBC; 1998; pags. 19-26.

[FSM 1998] Fiorini, S.T.; Staa, A.v.; Martins, R.B.; *Engenharia de Software com CMM*; Brasport; Rio de Janeiro, RJ; 1998.

[GHW 1995] Gresse, C.; Hoisl, B.; Wüst, J.; "A Process Model fo GQM-Based Measurement"; *Software Technology Transfer Initiative Technical Report STTI-95-04-E*, University of Kaiserslautern, Department of Computer Science; 1995.

[Highsmith 2002] Highsmith, J; *Agile Software Development Ecosystems*; Addison Wesley; 2002

[Hoare 1969] Hoare, C.A.R.; "An Axiomatic Basis for Computer Programming"; Communications of the ACM 12(10); 1969; pags 576-583 (valor histórico).

[Humphrey 1989] Humphrey, W.S.; *Managing the software process*; Addison-Wesley; 1989.

[Humphrey 1995] Humphrey, W.S.; *A Discipline for Software Engineering*; Addison-Wesley; 1995.

[Humphrey 1995] Humphrey, W.S.; *Introduction to the Team Software Process*; Addison-Wesley; 2000.

[IEEE 1990] *IEEE Standard Glossary of Software Engineering Terminology*; ANSI / IEEE Std 610.12-1990.

[LB 1985] Lehman, M.M.; Belady, L.A.; Software Evolution: Processes of Software Change; Academic Press; 1985.

[Lehman 1998] Lehman, M.M.; "Software Future: Managing Evolution"; *IEEE Software* 15(1); IEEE Computer Society; 1998; pags. 40-44.

[LSOHR 1998] Latum, F.v.; Solingen,R.v.; Oivo, M.; Hoisl, B.; Ruhe, G.; “Adopting GQM-based measurement in an industrial environment”; *IEEE Software* 15(1); IEEE Computer Society; 1998; pags 78-86.

[MDL 1987] Mills, H.D.; Dyer, M.; Linger, R.; “Cleanroom Software Engineering”; *IEEE Software* 4(5); IEEE Computer Society; 1987; pags 19-25.

[Pressman 1995] Pressman, R. S.; *Engenharia de Software*, Makron Books do Brasil; 1995.

[PWCC 1995] Paulk, M. C.; Weber, C. V.; Curtis, W.; Chrissis, M. B.; *The Capability Maturity Model - Guidelines for Improving the Software Process*; Addison Wesley, SEI series; 1995.

[SPICE 1998] ISO/IEC 15504: *Information Technology - Software Process Assessment*; Technical Report Series TR 15504-1 through TR 15504-9; ISO; 1998.

[Wake 2001] Wake, W.C.; *Extreme Programming Explored*; Addison Wesley; 2001.