# Políticas de Controle de Concorrência no Desenvolvimento Distribuído de Software

**João Gustavo Prudêncio, Leonardo Murta, Cláudia Werner**

PESC/COPPE – Universidade Federal do Rio de Janeiro
Caixa Postal 68.511 – 21.945-970 – Rio de Janeiro – RJ – Brasil

{gustavo,murta,werner}@cos.ufrj.br

***Abstract.*** *Software development has become a complex activity, due to intricate requirements implemented by distributed development teams. Software Configuration Management represents an important discipline for controlling the software evolution and for allowing the cooperation among developers, supporting concurrent teamwork. This paper systematizes the concurrency policies choice, aiming at productivity improvements in this scenario.*

***Resumo.*** *O desenvolvimento de software tem se tornado uma atividade cada vez mais complexa, devido aos requisitos cada vez mais difíceis de serem alcançados, aliado ao aumento e à distribuição da equipe de desenvolvimento. A Gerência de Configuração de Software representa uma disciplina importante para o controle da evolução de software e possibilita uma maior cooperação entre as pessoas envolvidas, apoiando o trabalho concorrente. Esse trabalho tem como objetivo sistematizar o processo de escolha de políticas de concorrência para cada artefato de um projeto de software, visando o aumento da produtividade.*

## 1. Introdução

Na engenharia de software, a redução do ciclo de vida do produto, aliado com o aumento da complexidade das soluções e da equipe de desenvolvimento, resultou em uma maior pressão por trabalho concorrente [ESTUBLIER 2000]. Essa pressão se torna ainda mais forte devido à distribuição geográfica dos participantes de um projeto. Dessa forma, há a necessidade de definir estratégias de cooperação para o desenvolvimento de sistemas, possibilitando que vários desenvolvedores distribuídos geograficamente modifiquem e gerenciem de forma eficiente o mesmo conjunto de artefatos.

A Gerência de Configuração de Software (GCS) é uma disciplina que possibilita a evolução de sistemas de software de forma controlada [DART 1991] e, portanto, contribui no atendimento às restrições de qualidade e de tempo [ESTUBLIER 2000]. Os sistemas de GCS são fundamentais para prover controle sobre os artefatos produzidos e modificados por diferentes desenvolvedores, dando suporte ao trabalho concorrente [MURTA et al. 2007].

As soluções existentes de GCS provêem uma infra-estrutura para a definição de quais artefatos serão passíveis de GCS, os chamados itens de configuração (ICs), e permitem que estes artefatos sejam obtidos por meio de um processo conhecido como *check-out*. Com isso, o engenheiro de software faz uma cópia desses artefatos para o seu

espaço de trabalho, local destinado à criação, alteração e teste dos ICs. Depois de efetuar mudanças, o engenheiro de software retorna os artefatos para o repositório, mediante um processo conhecido como *check-in*. Quando um engenheiro de software tenta fazer um *check-in* e o IC em questão sofreu modificações de outros engenheiros desde o seu último *check-out*, pode ocorrer uma situação de conflito.

Diferentes políticas de controle de concorrência podem ser aplicadas a cada IC de um projeto, permitindo ou não o paralelismo no desenvolvimento. O objetivo deste trabalho é identificar qual política de controle de concorrência é a mais recomendada para cada artefato de um projeto de software, tendo em vista uma maior produtividade no processo de desenvolvimento.

Este trabalho está organizado em outras cinco seções. Na Seção 2, são discutidos alguns conceitos importantes para o trabalho. A Seção 3 apresenta detalhadamente a abordagem proposta. Na Seção 4, é discutido um exemplo de aplicação da abordagem. Na Seção 5, são apresentados alguns trabalhos relacionados. Por fim, na Seção 6, são feitas algumas considerações finais e são destacados alguns trabalhos futuros.

## 2. Contextualização

Leon [2000] apresenta uma evolução das técnicas de GCS, onde, inicialmente, os artefatos de software eram compartilhados por diversos desenvolvedores, porém modificações efetuadas por um desenvolvedor geravam efeitos colaterais nos artefatos de outros desenvolvedores. Uma solução inicial foi a criação de várias cópias do mesmo artefato compartilhado, entretanto, faltava controle sobre as cópias existentes e era necessário o retrabalho nas diferentes cópias para implementar os mesmos requisitos e corrigir os mesmos defeitos. Adotou-se, então, a utilização de repositórios centralizados que armazenavam versões comuns dos artefatos, no entanto, era necessária a criação de mecanismos de controle sobre o repositório centralizado. Esses mecanismos são implementados por meio da utilização de políticas de controle de concorrência pelos sistemas de controle de versões. Dentre elas, podemos citar as políticas pessimista e otimista.

A política pessimista enfatiza o uso de *check-out* reservado, fazendo um bloqueio (*lock*) e inibindo o paralelismo do desenvolvimento sobre um mesmo artefato. Utilizando essa política, as situações de conflito são evitadas. Por outro lado, a política otimista assume que a quantidade de conflitos é naturalmente baixa e que é mais fácil tratar cada conflito individualmente, caso eles venham a ocorrer. O mecanismo de tratamento de conflitos utilizado pela política otimista é a junção (*merge*) das duas versões do IC. Existem situações onde uma determinada política é mais indicada do que a outra. Nos casos onde a junção dos trabalhos tende a ser complexa, quando, por exemplo, os ICs não são textuais e a ferramenta que conhece a representação binária dos ICs não dá apoio automatizado para junções, é mais indicado trabalhar usando políticas pessimistas. Por outro lado, nos casos onde os ICs são textuais e a junção não é complexa, a utilização de políticas otimista é mais indicada [Estublier 2001].

## 3. Abordagem Proposta

Além de sugerir a política de controle de concorrência aparentemente mais adequada para cada IC, este trabalho busca também identificar os ICs do sistema que merecem uma maior atenção em relação ao controle de concorrência, visando uma maior agilida-

de no processo de desenvolvimento de software. Esses ICs são chamados de *elementos críticos* do sistema. Cada IC é tratado como uma unidade de versionamento (UV), definida como um elemento atômico associado ao versionamento. Cada UV, por sua vez, é composto por sub-ICs, chamados de unidades de comparação (UC), que representam elementos atômicos usados para computação de conflitos [MURTA et al. 2007]. Dessa forma, o engenheiro de software define o nível de granularidade desses elementos e analisa o sistema em diferentes níveis de abstração. Por exemplo, considerando documentos texto como UVs, parágrafos ou frases podem ser considerados UCs. Por outro lado, adotando classes Java como UVs, atributos e métodos podem ser definidos como UCs.

Com a identificação dos elementos críticos, de certa forma, estão sendo identificados também alguns riscos no desenvolvimento do sistema. Alguns conhecimentos existentes na área de gerência de riscos são úteis para a análise desses riscos. A IEEE Std 1540 [IEEE 2001] define risco como a probabilidade de um evento, perigo, ameaça ou situação ocorrer associado às conseqüências indesejáveis, ou seja, um problema potencial. Nesse contexto, o risco tratado por esse trabalho é a perda de produtividade. De acordo com a gerência de riscos, os riscos que devem receber maior atenção em um projeto de software são os que têm maior probabilidade de acontecer e que têm grande potencial de causar um maior impacto no projeto. Assim, é calculado o valor referente à probabilidade de um dado risco ocorrer e o valor do impacto desse risco. Esses valores são multiplicados e um terceiro valor, chamado de exposição, é obtido. A exposição indica o quão preocupante é o risco [SOFTEX 2006].

Utilizando os conhecimentos de gerência de riscos, para cada IC do sistema é calculado um valor que representa o quão crítico é esse item. Esse valor leva em consideração duas métricas: nível de **concorrência** de um IC (semelhante à probabilidade do risco, pois quanto maior for a concorrência, maior será a probabilidade da necessidade de junção) e nível de **dificuldade de junção** do IC (semelhante ao impacto do risco, pois quanto mais difícil for a junção, maior será o impacto na produtividade da equipe).

O nível de concorrência é calculado comparando o tempo em que mais de um desenvolvedor teve posse do item para modificação e o tempo em que ao menos um desenvolvedor teve posse do item para modificação. A Figura 1 apresenta uma situação que ilustra o cálculo dessa métrica. No tempo de vida do IC, três desenvolvedores ficaram em posse desse item para modificação. O tempo de posse para modificação é determinado pelo intervalo entre *check-out* e *check-in*.

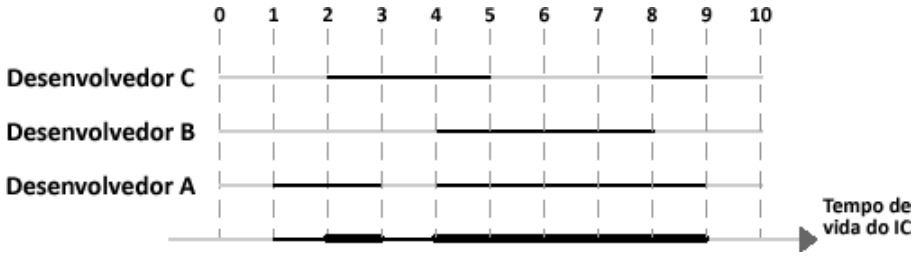


**Figura 1. Situação para análise do nível de concorrência.**

Na Figura 1, podem ser identificados períodos em que o IC não ficou em posse de nenhum desenvolvedor; períodos em que ficou em posse de apenas um desenvolvedor; e períodos em que o item ficou em posse de mais de um desenvolvedor, ou seja, períodos em que houve concorrência. Com isso, é calculado um número

normalizado (i.e., entre 0 e 1, inclusive) que indica o nível de concorrência de cada IC analisado. Quanto mais próximo de 1, maior o nível de concorrência desse item.

Para um dado IC, essa métrica é calculada por meio da seguinte fórmula:

$$concorrência(IC) = TC(IC) / TP(IC); \quad \textbf{(Fórmula 1)}$$

onde TC(IC) significa o tempo que houve concorrência e TP(IC) representa o tempo em que o IC ficou em posse de ao menos um desenvolvedor. Um exemplo da aplicação dessa fórmula é apresentado na seção 4.

A segunda métrica quantifica a dificuldade de se fazer a junção de duas versões de um dado IC. Ou seja, considerando que houve uma situação de conflito, é quantificada, entre 0 e 1, a dificuldade de resolução desse conflito. Essa métrica leva em consideração a quantidade de UCs, que são sub-ICs do IC em questão, que sofreram conflitos. Quanto mais próximo de 1, maior a dificuldade de se fazer a junção.

Após o desenvolvedor fazer o *check-in*, podem ser destacadas quatro versões de um determinado IC: versão original, que o desenvolvedor fez *check-out*; versão do usuário, que foi modificada na sua área de trabalho; versão atual, que está no repositório e que pode ser diferente da original, visto que pode ter havido c*heck-ins* de outros desenvolvedores; e, finalmente, a versão final, que representa a junção das versões do usuário e atual (Figura 2).

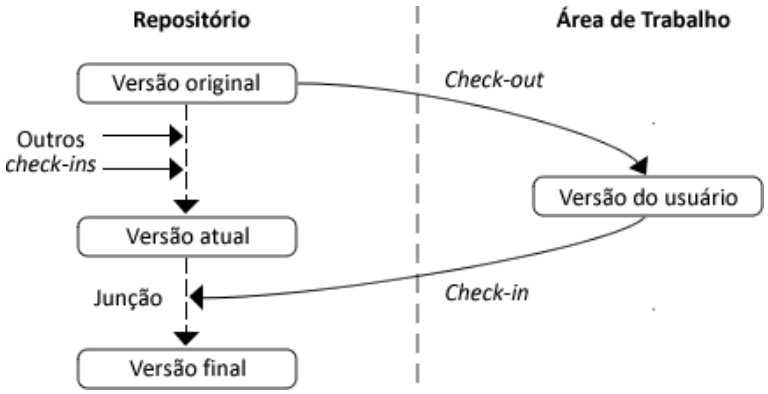


**Figura 2. Versões de um IC no momento do *check-in*.**

Para calcular a métrica de dificuldade de junção, duas informações são utilizadas: o conjunto de operações sofridas pelas UCs desde a sua versão original – O(IC) – e o conjunto de operações que realmente foram efetivadas na versão final – OE(IC). Para o cálculo dessa primeira informação, são identificadas as operações feitas na versão do usuário (VU) e na versão atual (VA), tendo como base a versão original (VO), utilizando o algoritmo $diff_3$ [CONRADI e WESTFECHTEL 1998]:

$$O(IC) = diff_3(VU, VA, VO); \quad \textbf{(Fórmula 2)}$$

A segunda informação, o conjunto de operações efetivadas, compara a versão final (VF) com a versão original das UCs, utilizando o algoritmo diff [CONRADI e WESTFECHTEL 1998]:

$$OE(IC) = diff(VO, VF). \quad \textbf{(Fórmula 3)}$$

O e OE representam conjuntos de operações (inclusão, exclusão ou alteração de UCs) e são representados na Figura 3. A intersecção de O e OE representa as operações efetuadas nas versões atual e do usuário que foram efetivadas na versão final, ou seja, operações que não trouxeram dificuldade no momento de junção e que seriam corretamente combinadas por um algoritmo automático de junção. As operações que pertencem ao conjunto O, porém que não pertencem à intersecção, são operações que foram feitas, mas não foram efetivadas na versão final, ou seja, esforço desperdiçado. Já as operações que pertencem a OE, porém não pertencem à intersecção, são operações que ocorreram para resolução de conflitos, ou seja, esforço adicional. Este último subconjunto, representado pela diferença entre OE e O, simboliza a efetiva dificuldade de junção de um IC. Dessa forma, o nível de dificuldade de junção é calculado por meio da Fórmula 4.

$$dificuldade\_junção(IC) = |(OE - O)| \,/\, |OE|; \qquad \textbf{(Fórmula 4)}$$

onde |X| representa o número de elementos de um conjunto X.

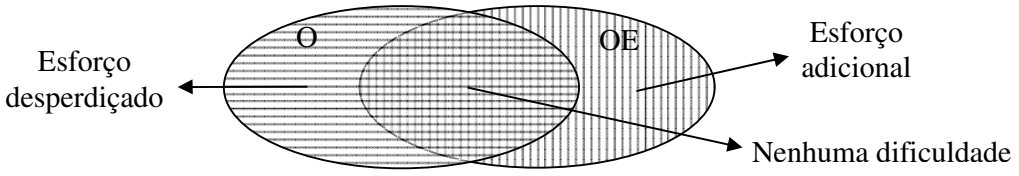


**Figura 3. Representação dos conjuntos O e OE.**

Quando não houver conflitos ou a resolução destes não demandar nenhum esforço adicional (OE – O = Ø), a métrica de dificuldade de junção terá como resultado 0. Por outro lado, quando todas as alterações que resultaram na versão final do IC representarem esforços adicionais (OE – O = OE), a métrica de dificuldade de junção terá como resultado 1, o valor máximo.

Uma vez calculadas as duas métricas para cada IC, o valor que representa a criticidade do item (que também varia entre 0 e 1) é calculado por meio da multiplicação do nível de concorrência com o nível de dificuldade de junção do IC em um determinado período de tempo. Além disso, a abordagem sugere a política para o controle de concorrência mais indicada para cada item em questão. Essa sugestão é feita com base em pontos de cortes das métricas, que definem se o valor obtido é alto ou baixo e que podem ser definidos pelo engenheiro de software (Tabela 1). Quando certo IC possui um alto nível de concorrência e um baixo nível de dificuldade de junção, a probabilidade de ocorrer um conflito é alta, porém não é difícil de resolvê-lo, logo a política otimista é indicada. Em casos onde o nível de concorrência é baixo e o nível de dificuldade de junção é alto, a política indicada é a pessimista, pois um bloqueio no item não traz muitos problemas, já que o nível de concorrência é baixo. Quando um IC apresenta valores baixos nas duas métricas, qualquer uma das duas políticas pode ser utilizada. Nos casos onde o IC apresenta ambos os níveis elevados, o elemento é identificado como crítico.

Uma vez aplicada a abordagem, o engenheiro de software pode fazer alterações na política de controle de concorrência ou pode alocar melhor seus recursos, tentando evitar que muitos desenvolvedores trabalhem em um elemento crítico ao mesmo tempo, minimizando possíveis conflitos. Ou ainda, o engenheiro de software pode decidir fazer uma reestruturação nos elementos críticos do sistema para diminuir os níveis de concorrência e de dificuldade de junção.

**Tabela 1. Indicação das políticas de concorrência.**

| | | Nível de Concorrência | |
|---|---|---|---|
| | | **Baixo** | **Alto** |
| Dificuldade de Junção | **Baixo** | Indiferente | Política otimista |
| | **Alto** | Política pessimista | Reestruturação |

A abordagem apresentada necessita de dados históricos do projeto de desenvolvimento, dessa forma, ela só pode ser aplicada após o início do mesmo em momentos escolhidos pelo engenheiro de software. Além disso, a abordagem pode ser aplicada mais de uma vez ao longo da fase de desenvolvimento e é flexível quanto ao período de tempo analisado.

## 4. Exemplo

Nesta seção, é apresentado um exemplo do cálculo do nível de criticidade e indicação de uma política de controle de concorrência para um determinado IC. Na aplicação real da abordagem, esse cálculo deve ser feito para todos os ICs do sistema.

A Figura 1, apresentada anteriormente, ilustra a situação que será utilizada. Nela são exibidos os períodos de tempo que o IC ficou em posse dos desenvolvedores para modificação. Utilizando a Fórmula 1, o resultado obtido é o seguinte:

$$concorrência(IC) = TC(IC) / TP(IC) = 6/8 = 0{,}75 = 75\%.$$

Durante o período analisado no exemplo, aconteceram 5 *check-ins*. Para definir o nível de dificuldade de junção desse item no mesmo período, é necessário calcular a dificuldade de junção em cada um dos 5 *check-ins* e fazer a média aritmética desses valores. Para simplificar esse exemplo, será calculado o valor referente a um *check-in* e será considerado que a dificuldade de junção nos outros 4 casos é a mesma. Assim, será calculado o nível de dificuldade de junção durante o período apresentado na Figura 1.

A Figura 4 apresenta a situação para exemplificar o cálculo da dificuldade de junção. São analisadas as quatro versões do IC apresentadas pela Figura 2. Os números circulados representam diferentes versões de uma certa UC, que é representada por um número. Na versão original do IC existiam 7 UCs (1 a 7). Na versão do usuário foram removidos as UCs 4 e 6, adicionada a 9 e alteradas as 2, 3 e 7. No momento do *check-in* (versão atual), haviam sido removidas as UCs 6 e 7, adicionada a 8 e modificadas as 3, 4 e 5. Após *check-in* do usuário, o IC (versão final) se apresentava da seguinte forma: a UC 2 ficou com a versão do usuário; a 3, devido a um possível conflito, teve uma nova versão gerada, baseada nas versões final e atual; a UC 4 foi removida; a 5 teve as modificações dos outros desenvolvedores descartadas e permaneceu com a versão original; a UC 6 foi removida; a 7, apesar de ter sido excluída na versão atual, foi mantida com as alterações do usuário; e, finalmente, as UCs adicionadas 8 e 9 permaneceram. Para encontrar os conjuntos O e OE, neste exemplo, são aplicadas as Fórmulas 2 e 3:

$$O = diff_3(VU, VA, VO) = \{2', 3', 3'', \cancel{4}, 4'', 5'', \cancel{6}, \cancel{7}, 7', 8, 9\},$$

$$OE = diff(VO, VF) = \{2', 3''', \cancel{4}, \cancel{6}, 7', 8, 9\},\ \text{e, portanto,}$$

$$OE - O = \{3'''\};$$

onde os números cortados representam a remoção de uma UC e as aspas indicam em qual versão o IC foi modificado.

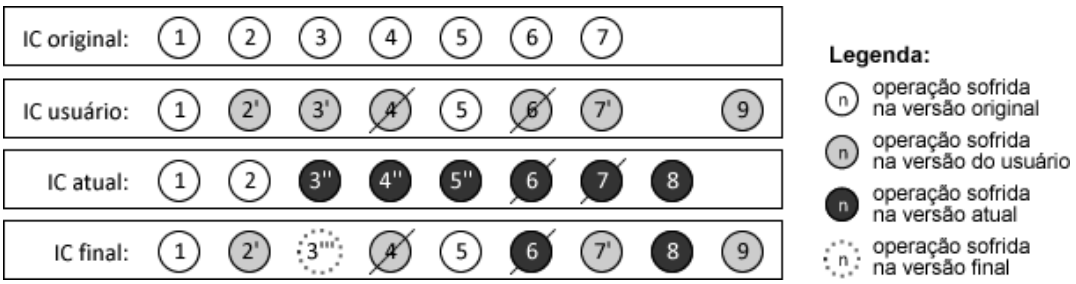


**Figura 4. UCs para exemplificar o cálculo da métrica de dificuldade de junção.**

Finalmente, a métrica pode ser calculada pela Fórmula 4:

$$dificuldade\_junção(IC) = |(OE - O)| / |OE| = 1/7 = 14,3\%.$$

Analisando os resultados das duas métricas e com base na Tabela 1, a política otimista seria a indicada para o IC. Além disso, para quantificar a criticidade do IC durante o período de tempo analisado, faz-se a multiplicação dos resultados das duas métricas e chega-se ao seguinte valor: 0,75 x 0,14 = 0,105, aproximadamente 10%, o que indica (dependendo de valores definidos pelo engenheiro de software) que o IC não é crítico e não demanda necessidade de reestruturação.

## 5. Trabalhos Relacionados

O trabalho concorrente desempenha um papel importante no processo de desenvolvimento de software, principalmente no desenvolvimento distribuído. Porém, poucos trabalhos tratam da definição e escolha de políticas para o controle de concorrência.

Estublier [2001] afirma que as infra-estruturas de GCS não atentem às restrições de escalabilidade e eficiência necessárias ao controle de concorrência e, por isso, propõe uma nova arquitetura para os sistemas de GCS. Essa nova arquitetura vai de encontro às que existiam na época em que o trabalho foi desenvolvido. Entretanto, algumas das deficiências apontadas pelo trabalho já são atendidas pelos atuais sistemas de GCS, são elas: arquitetura distribuída e possibilidade de definir diferentes políticas para diferentes artefatos de um projeto.

Estublier *et al* [2003] apresentam um novo modelo de trabalho baseado em grupos e propõem operações com as quais políticas para o controle de concorrência são construídas. É proposta uma linguagem para a definição das políticas, permitindo que o engenheiro de software crie políticas que podem variar desde políticas mais restritivas (uso de bloqueios) até outras mais flexíveis (sem bloqueios). Porém, esse trabalho não apóia a escolha da política mais indicada para cada situação.

## 6. Considerações Finais

O trabalho concorrente possibilita uma melhor definição de estratégias de cooperação entre os desenvolvedores no processo de desenvolvimento de software, além disso, se torna ainda mais importante devido à distribuição geográfica dos participantes. Com a realização deste trabalho, foi desenvolvida uma sistematização do processo de escolha de políticas de concorrência para os artefatos de um projeto de software, visando um aumento na produtividade.

Algumas limitações foram identificadas na abordagem. Há casos em que as métricas não conseguem diferenciar duas situações muito ruins, mesmo se uma é pior do que a outra. Além disso, os dados utilizados pelas fórmulas que calculam as duas métricas, dependendo do sistema de controle de versões utilizado, podem não ser fornecidos. Porém, por meio de técnicas auxiliares, como a utilização de ramos ou *hooks*, é possível a obtenção desses dados.

Como trabalho futuro, será realizada uma avaliação da abordagem utilizando, como estudo de caso, um projeto *open-source* disponibilizado livremente na internet, onde serão analisados os arquivos gerados pelos sistemas de GCS. A abordagem será aplicada em um dado momento do passado e será verificado se as sugestões obtidas nesse momento seriam válidas no decorrer do projeto, Por exemplo, se foi sugerida a política pessimista para um certo IC, e continua otimista, mas com muita dificuldade de junção, então a sugestão da abordagem era realmente pertinente. Além disso, será desenvolvida uma ferramenta para a automatização da abordagem. Essa ferramenta utilizará técnicas de visualização de software [CEMIN 2001], que incluem técnicas de desenho, coloração, animação e agrupamento de informações, para apresentar em diagramas UML o resultado da aplicação das métricas de concorrência e dificuldade de junção. Essa representação tem o objetivo de facilitar o entendimento das informações [CEMIN 2001] e atuar como uma forma de comunicação entre as pessoas que estão explorando ou manipulando a mesma informação.

## Referências


CEMIN, C., 2001, *Visualização de Informações Aplicada à Gerência de Software*, Dissertação de M.Sc., Instituto de Informática, UFRGS, Porto Alegre, Rio Grande do Sul, Brasil.

CONRADI, R., WESTFECHTEL, B. (1998), "Version Models for Software Configuration Management", ACM Computing Surveys, v. 30, n. (2), pp. 232-282.

DART, S. (1991) "Concepts in Configuration Management Systems". In: *International Workshop on Software Configuration Management (SCM)*, pp. 1-18, Trondheim, Norway, June.

ESTUBLIER, J. (2000) "Software Configuration Management: a Roadmap". In: *International Conference on Software Engineering (ICSE)*, pp. 279-289, Limerick, Ireland, June.

ESTUBLIER, J. (2001) "Objects Control for Software Configuration Management". In: *International Conference on Advanced Information Systems Engineering*, pp. 359-373, London, UK, June.

ESTUBLIER, J., GARCIA, S., VEGA, G. (2001) "Defining and Supporting Concurrent Engineering policies in SCM". International Workshop on Software Configuration Management, pp. 1-15, Portland, Oregon, USA, May.

IEEE (2001) *Std 1540 - IEEE Standard for Software Life Cycle Processes - Risk Management*, Institute of Electrical and Electronics Engineers.

LEON, A. (2000) *A Guide to Software Configuration Management*, Artech House Publishers.

MURTA, L., OLIVEIRA, H., DANTAS, C., et al. (2007) "Odyssey-SCM: An Integrated Software Configuration Management Infrastructure for UML models", *Science of Computer Programming*, v. 65, n. (3), pp. 249-274.

SOFTEX (2006) "MPS.BR – Guia de Implementação, Parte 5: Nível C 1.0". In: http://www.softex.br, acessado em 01/03/07.