Sistema de Controle de Alteração e Configuração[1]

André Villas Boas
CPqD–TELEBRÁS
email: acvillas@cpqd.ansp.br

Edésio Costa e Silva
CPqD–TELEBRÁS
email: acedesio@cpqd.ansp.br

Júlio Cardoso Pereira
CPqD–TELEBRÁS
email: acjcp@cpqd.ansp.br

Resumo

Este trabalho tem como objetivo apresentar o Sistema de Controle de Alteração e Configuração desenvolvido como suporte às atividades de garantia de qualidade dos sistemas a cargo do Departamento de Sistemas de Operação do Centro de Pesquisa e Desenvolvimento da TELEBRÁS.

Será apresentada sua forma de funcionamento, eventos, transações e um pequeno exemplo de sua utilização.

Abstract

This article presents the Change and Configuration Control System designed to support the quality assurance activities helded by the Operation Systems Department of the Brazilian Telecom (Telebrás) R&D Center.

It will be shown its functionality, events, transactions and a short example of usage.

1 Introdução

No processo de produção de *software* um objeto em desenvolvimento não tem um comportamento estático, ele evolui com o tempo. Chama-se de versão ao estado particular de um objeto, e por configuração entende-se como a relação entre versões de um objeto composto, ou seja, configuração é uma instância do sistema composta da união de uma versão específica de cada objeto componente[19].

Gerência de Controle de Alteração e Configuração de *Software* pode ser definida como uma disciplina para gerenciamento efetivo da produção de *software*. Cabe a ela identificar a configuração de um sistema em relação ao tempo com a finalidade de, sistematicamente, controlar as mudanças desta configuração, manter sua integridade e, desta forma, possibilitar seu rastreamento através do ciclo de vida do sistema[3].

As funções de um gerenciamento de configuração são: identificação de configuração (que itens constituem uma configuração), controle de configuração (que passos no processo de alteração afetam uma configuração), auditoria de configuração (quais são as diferenças entre as versões) e *status* de configuração (que modificações foram feitas por determinado programador)[5][1].

Uma característica importante de um gerenciador é o modo de armazenamento dos objetos, já que isto implica em uso de espaço. Alguns modos propostos são: orientação por arquivos, orientação por páginas[13], orientação por registros[10] e orientação por deltas (utilizado pelo SCCS[4] e pelo RCS[18]). Além do modo

[1]Este trabalho contou com a coordenação técnica de Péricles Gama Negri, então coordenador do projeto ASSISTE.

temos também que definir um sistema de armazenamento. O SCCS e o RCS são gerenciadores implementados sobre o sistema operacional UNIX (neles uma versão é reconstruída a partir de seus arquivos delta), o DSEE[12] e o Gypsy[7] possuem o controle de versões integrado ao sistema operacional (e uma versão pode ser diretamente recuperada pelo sistema operacional), já o Adele[2], o CCC/Manager[6] e o MVC[20] são gerenciadores onde o núcleo do sistema é baseados em banco de dados.

A integridade dos dados é garantida pelo controle de acesso. Gerenciadores como o RCS utilizam, além da proteção mínima provida pelo sistema operacional, uma lista de acesso, onde cada objeto detém uma lista de usuários que tem direitos sobre ele. O CCC, por exemplo, divide os usuários em classes e define o acesso e operações sobre os objetos segundo estas classes.

O presente trabalho contém os resultados de uma atividade realizada no âmbito do Projeto ASSISTE[15], com vistas a apresentar uma solução de controle de alteração e configurações de *software* para os projetos sob responsabilidade do DSO (Departamento de Sistemas de Operação, CPqD–TELEBRÁS), que suportasse a metodologia definida na MUSA (Metodologia Unificada de Sistemas Aplicativos do Sistema TELEBRÁS[9]).

Ele apresenta a dinâmica de criação, manutenção e liberação de um item sob controle de configuração, que está associada a uma máquina de estados, mostrando os eventos relacionados com este processo, a estrutura de Organização e Métodos (O&M) do sistema (criando um estrutura funcional para as responsabilidades), as transações ligadas aos eventos possíveis e um exemplo que descreverá a funcionalidade do sistema. É importante ressaltar que o sistema que será apresentado tem como premissa que os projetos clientes possuem uma política de nomenclatura que garanta unicidade dos nomes de arquivos no repositório.

Esta política, os itens que perfazem uma configuração, a constituição do grupo de gerência de configuração e o relacionamento da atividade de controle de alteração e configuração, são assuntos que devem ser definidos na documentação de planejamento do projeto, ou em um plano específico de controle de alteração e configuração. Um padrão internacionalmente aceito para este plano é o ANSI/IEEE[1].

Deve-se lembrar que os itens sob controle de configuração tanto podem ser módulos do sistema em desenvolvimento quanto programas, dados de teste e documentação (especificação de requisitos, documentos de projetos, etc). Como sugestão fica a idéia de que a geração de módulos do sistema e de programas de teste deveria ser encarada como dois projetos distintos, desta forma, a partir dos documentos de especificação, a equipe responsável pelo desenvolvimento gera o *software*, enquanto um outra equipe gera os casos de teste.

A solução *software* do SCAC (Sistema de Controle de Alteração e Configuração) está baseada nas ferramentas RCS e GNU make[17] do UNIX, já que a premissa básica para o desenvolvimento deste sistema era a não incidência de custos de aquisição. Ela apresenta ao usuário um conjunto de comandos que mapeia as transações do sistema, garante sua dinâmica e a segurança durante o acesso aos itens sob controle.

Os requisitos definidos para o SCAC são os seguintes:

- Criação/controle de versões
- Criação/controle de configuração
- Segurança de acesso
- Controle de alterações
- Garantia de integridade dos itens
- Avaliação de alterações

- Auditoria
- *Status* do projeto
- Histórico do projeto

2 Responsabilidades

As responsabilidades aqui descritas são puramente funcionais e visam apenas tornar possível a estrutura de O&M do SCAC. A forma de relacionamento entre as entidades, bem como seu grau de formalismo não faz parte do escopo do sistema, nem impacta seu funcionamento, e devem estar definidos em documentação apropriada. No presente sistema a forma de comunicação entre as gerências é feita por *e-mail*. As siglas aqui definidas serão utilizadas ao longo deste trabalho.

O organograma da figura 1 define a estrutura funcional atendida pelo SCAC.

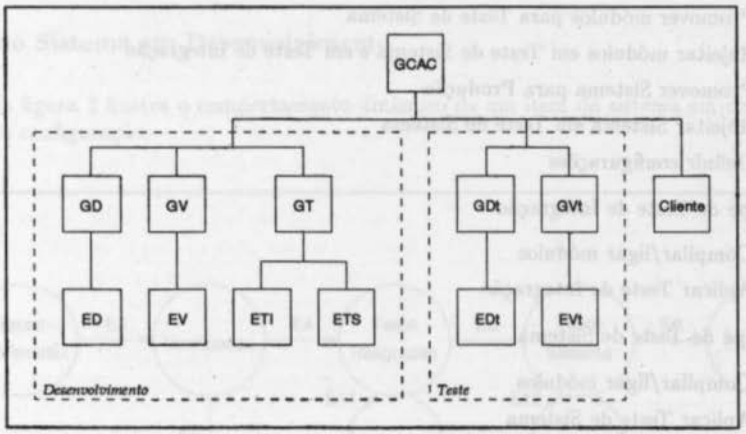

Figura 1: Estrutura funcional atendida

GCAC: Gerência de Controle de Alteração e Configuração

- Cadastrar as outra gerências (GD, GV, GT, GDt, GVt)
- Colocar itens (do sistema e de teste) em Manutenção
- Criar as versões iniciais de cada item (do sistema e de teste)

GD: Gerência de Desenvolvimento dos módulos do sistema

- Definir/alterar a lista de acesso aos módulos
- Promover/submeter módulos para Verificação
- Criar/alterar a equipe de desenvolvimento

ED: Equipe de Desenvolvimento dos módulos do sistema

- Recuperar módulos para alterações

- Recolocar os módulos alterados
- Editar/compilar módulos
- Testar módulo

GV: Gerência de Verificação dos módulos do sistema

- Promover/submeter módulos para Teste de Integração
- Rejeitar módulos em Verificação
- Criar/alterar a equipe de desenvolvimento

EV: Equipe de Verificação dos módulos do sistema

- Recuperar módulos disponíveis para Verificação
- Gerar listagens dos módulos
- Executar procedimentos de Verificação

GT: Gerência de Teste

- Promover módulos para Teste de Sistema
- Rejeitar módulos em Teste de Sistema e em Teste de Integração
- Promover Sistema para Produção
- Rejeitar Sistema em Teste de Sistema
- Definir configurações

ETI: Equipe de Teste de Integração

- Compilar/ligar módulos
- Aplicar Teste de Integração

ETS: Equipe de Teste de Sistema

- Compilar/ligar módulos
- Aplicar Teste de Sistema

Cliente: Usuário final

- Copiar itens em Produção

GDt: Gerência de Desenvolvimento de Testes

- Definir/alterar a lista de acesso aos programas e dados de teste
- Promover/submeter programas e dados de teste para Verificação

EDt: Equipe de Desenvolvimento de Testes

- Recuperar programas e dados de teste para alterações
- Recolocar os itens alterados
- Editar/compilar os programas de teste
- Testar programas e dados de teste

GVt: Gerência de Verificação de Teste

- Promover/submeter programas e dados de teste para Produção

- Rejeitar programas e dados em Verificação

EVt: Equipe de Verificação de Teste

- Recuperar programas e dados de teste disponíveis para Verificação
- Gerar listagens dos programas de teste
- Executar procedimentos de Verificação

3 Dinâmica do Sistema

A idéia de definir um ciclo de vida para os objetos e organizar as versões segundo este ciclo está implementada em vários gerenciadores, como descrito por Leblang[12] ou Katz[11].

Conforme a sua evolução uma versão vai mudando de estado, sendo que cada estado define as operações possíveis sobre os itens e os responsáveis por estas operações.

3.1 Itens do Sistema em Desenvolvimento

O diagrama da figura 2 ilustra o comportamento dinâmico de um item do sistema em desenvolvimento sob controle de configuração.

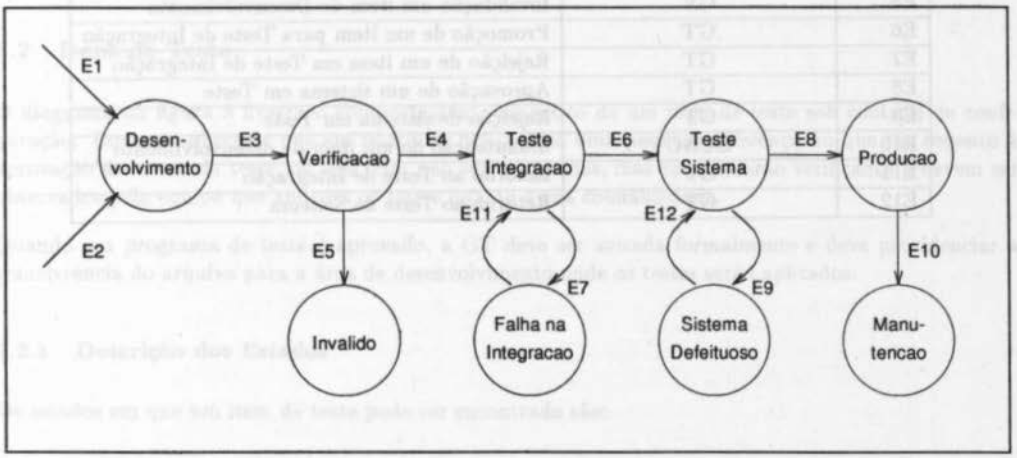

Figura 2: Estados de um item sob controle de configuração

3.1.1 Descrição dos Estados

Os estados em que um item de desenvolvimento pode ser encontrado são:

Estado	Ações	Responsável
Desenvolvimento	Disponível para correção/alteração	ED
Verificação	Disponível para execução de procedimentos de Verificação	EV
Inválido	Disponível (novamente) para correção/alteração	ED
Teste Integração	Disponível para execução dos Testes de Integração	ETI
Falha na Integração	Disponível para avaliação/correção/alteração	ED
Teste Sistema	Disponível para execução dos Teste de Sistema	ETS
Sistema Defeituoso	Disponível para avaliação/correção/alteração	ED
Produção	Disponível para o Cliente	NOTA*
Manutenção	Disponível para avaliação/correção/alteração	ED

*A responsabilidade é da Divisão de Transferência de Tecnologia.

3.1.2 Eventos

Evento	Executor	Descrição
E1	GCAC	Criação de um item de desenvolvimento
E2	usuário na lista de acesso	Criação de uma nova versão de um item de desenvolvimento
E3	GD	Promoção de um item para Verificação
E4	GV	Promoção de um item para Teste de Integração
E5	GV	Invalidação um item de Desenvolvimento
E6	GT	Promoção de um item para Teste de Integração
E7	GT	Rejeição de um item em Teste de Integração
E8	GT	Aprovação de um sistema em Teste
E9	GT	Rejeição de sistema em Teste
E10	GCAC	Manutenção de um item de desenvolvimento
E11	GT	Retorno ao Teste de Integração
E12	GT	Retorno ao Teste de Sistema

3.1.3 Transações

Transações	Executor	Evento
Criação de um item de desenvolvimento	GCAC	E1
Alteração da lista de acesso dos itens em desenvolvimento	GD	–
Promoção de um item para Verificação	GD	E3
Recuperação de um item para consulta	ED, EV, ETI, ETS, Cliente*	–
Recuperação de um item para edição	ED	–
Criação de uma nova versão de um item de desenvolvimento	usuário na lista de acesso	E2
Promoção de um item para Teste de Integração	GV	E4
Invalidação de um item de desenvolvimento	GV	E5
Promoção de um item para Teste de Sistema	GT	E6
Rejeição de um item em Teste de Integração	GT	E7
Aprovação de um sistema em Teste	GT	E8
Rejeição de sistema em Teste	GT	E9
Manutenção de um item de desenvolvimento	GCAC	E10
Criação de configuração	GCAC	–
Cópia da configuração de teste	GT	–

*Esta transação pode ser efetuada nas seguintes situações: ED, sempre; EV, quando *status* = Verificação; ETI, quando *status* = Teste Integração; ETS, quando *status* = Teste Sistema; Cliente, quando *status* = Produção.

3.2 Itens de Teste

O diagrama da figura 3 ilustra o comportamento dinâmico de um item de teste sob controle de configuração. Devemos observar que em relação à figura 2 há uma pequena diferença no que diz respeito à aprovação dos itens de teste, ou seja, eles não sofrerão testes, mas apenas serão verificados e devem ser observados pela equipe que aplicará os testes quanto à sua confiabilidade.

Quando um programa de teste é aprovado, a GT deve ser avisada formalmente e deve providenciar a transferência do arquivo para a área de desenvolvimento onde os testes serão aplicados.

3.2.1 Descrição dos Estados

Os estados em que um item de teste pode ser encontrado são:

Estado	Ações	Responsável
Desenvolvimento	Disponível para correção/alteração	EDt
Verificação	Disponível para execução dos procedimentos de Verificação	EVt
Inválido	Disponível para a para correção/alteração	EDt
Produção	Disponível para a GT	GT
Manutenção	disponível novamente para correção/alteração	EDt

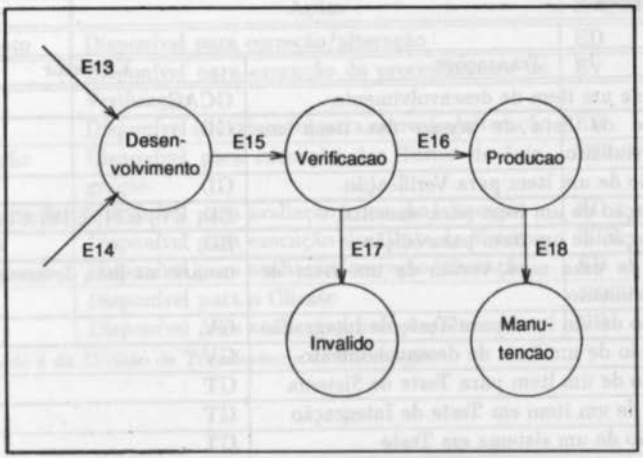

Figura 3: Estados de um item sob controle de configuração (programa de teste)

3.2.2 Eventos

Evento	Executor	Descrição
E13	GCAC	Criação de um item de teste
E14	usuário na lista de acesso	Criação de uma nova versão de um item de teste
E15	GDt	Promoção de um item de teste para Verificação
E16	GVt	Aprovação de um programa de teste
E17	GVt	Invalidação de um item de teste
E18	GCAC	Manutenção de um item de teste

3.2.3 Transações

Transações	Executor	Evento
Criação de um item de teste	GCAC	E13
Alteração da lista de acesso dos itens de teste	GDt	–
Promoção de um item de teste para Verificação	GDt	E15
Recuperação de um item de teste para consulta	EDt, EVt, Cliente*	–
Recuperação de um item de teste para edição	EDt	–
Criação de uma nova versão de um item de teste	usuário na lista de acesso	E14
Aprovação de um programa de teste	GVt	E16
Invalidação de um item de teste	GVt	E17
Manutenção de um item de teste	GCAC	E18
Criação de configuração de teste	GVt	–

*Esta transação pode ser efetuada nas seguintes situações: EDt, sempre; EVt, quando *status* = Verificação; Cliente, quando *status* = Produção.

4 Funções do Sistema

A idéia principal do SCAC é que o trabalho de desenvolvimento de *software* seja feito em um ambiente limpo (*clean-room*). Para isto cada projetista deverá trabalhar em sua conta (associada ao seu *username*) em diretório particular, sem fazer referências a módulos de outros projetistas (excessões a esta regra são arquivos de descrição de interfaces ".h"), isto é, sempre que o projetista fizer referência a elementos que perteçam a outros módulos, ele deve criar simulacros (*stubs*) daqueles módulos. Desta forma ele terá completo controle do ambiente durante os testes de unidade. O SCAC não se interessa pelo que se passa na área do projetista. Apenas quando o projetista torna um item público, este item passa ao controle do SCAC. O SCAC utiliza o *username* como indicador de função e do nível de privilégio de cada pessoa, portanto cada pessoa em cada função deve ter um *username* próprio.

O SCAC prevê a utilização de um repositório único para os itens de configuração. Ele apresenta um conjunto de funções que garante a dinâmica do sistema (promoções dos itens aos vários estados) bem como sua estruturação (criação de gerências, construção das listas de acesso, etc) e evita que o usuário tenha necessidade de conhecer os comandos do `RCS`. Também proporciona uma visão mnemônica do sistema, pois os nomes das funções tentam de algum modo retratar seu funcionamento.

As premissas de responsabilidade sobre os comandos fornecidos pelo sistema devem ser seguidas para seu bom funcionamento, embora nada impeça que alguém com bom conhecimento dos comandos do `RCS` venha a burlar o funcionamento aqui descrito.

Vamos ver agora alguns dos comandos do sistema, mostrando sua função e sua sintaxe (escritas em `Bourne Shell`).

Função	Sintaxe
Cadastramento de gerências	`qcriager classe gerente`
Cadastramento das equipes	`qcriaequipe usuario ...`
Cadastramento de UI[2]	`qcriaui UI [+responsavel] [item] ...`
Alteração de UI	`qmudaui UI [+|-] item ...`
Cadastramento de elementos da UI	`qcriamod [-rversao] [-tdescricao] item`
Gerenciamento da lista de acesso	`qmudala [+|-|=]usuario [,usuario] item ...`
Transição de desenvolvimento para validação	`qpromove item[:versao] ...`
Transição de validação para inválido	`qinvalido item[:item] ...`
Transição de validação para teste de integração	`qparati item[:versao] ...`
Transição de teste de integração para falha de integração	`qfalhati item[:versao] ...`
Transição de teste de integração para teste de sistema	`qparats item[:versao]`
Transição de teste de sistema para sistema defeituoso	`qfalhats item[:versao]`
Transição de teste de sistema para produção	`qparaprod item[:versao] ...`
Transição de produção para manutenção	`qparamanut item[:versao] ...`
Definição do nome lógido da versão	`qcongela nome item[:versao] ...`
Recuperação de item para consulta	`qrecmod item[:versao] ...`
Recuperação de item para edição	`qeditamod item[:versao] ...`
Armazenagem de novo item	`qsalvamod item ...`
Criação de diretório de trabalho	`qcriadir dir ...`
Obtenção da lista de UIs de um sistema	`qlistasis`
Obtenção da lista de elementos de uma UI	`qlistaui ui`
Listagem da história de um item	`qlistalog item`
Cópia de uma configuração da área de produção de testes para a área de Testes de Integração e área de Teste de Sistema	`qpegateste item[:versao] ...`
Cadastramento de procedimento de teste (PT)	`tcriapt PT [+responsavel] item ...`
Alteração de PT	`tmudapt PT [+|-]item ...`
Cadastramento de casos-teste do PT	`tcriamod [-rversao] item`
Transição de validação para produção	`tparati item[:versao] ...`

5 Exemplo

Suponhamos que exista um grupo envolvido num projeto de desenvolvimento de Sistemas de Assentos Ejetáveis (SAE).

A primeira atividade gerencial, efetuada pela GCAC, será o cadastramento das outras gerências (GD, GV e GT), da seguinte forma (lembre-se que cada pesssoa trabalha na sua conta!):

```
qcriager desenv gabriel
qcriager desenv pedro
qcriager valida madalena
qcriager teste barrabas
```

[2]Unidade de Implementação, isto é, uma "parte" de um produto de *software* que pode ser implentada e testada separadamente, como descrito na MUSA[9].

Neste momento as equipes podem então ser cadastradas pelos seus respectivos gerentes. Por exemplo, o gerente de desenvolvimento **gabriel**, cadastra sua equipe da seguinte forma:

```
qcriaequipe baltazar belchior gaspar
```

A GCAC tem também que cadastrar as UIs (e seus módulos) pertencentes ao sistema e respectivos responsáveis. O responsável por uma UI é simplesmente aquele que presta contas ao gerente sobre esta UI. Ele não desenvolve necessáriamente os elementos que a compõem.

```
qcriaui assento +belchior encosto.c encosto.h suporte.c suporte.h
qcriaui ejetor +gaspar disparador.c disparador.h paraquedas.c paraquedas.h
```

A GCAC pode também alterar a composição de uma determinada UI:

```
qmudaui ejetor -paraquedas.c -paraquedas.h +sensor.c +sensor.h
```

A GCAC deve então proceder à criação das versões iniciais dos itens sob controle de configuração:

```
qcriamod -r0.0 disparador.c
enter description, terminated with single '.' or end of file:
>>Dispositivo 'disparador' do sistema ejetor.
>>.
```

Note que este módulo já deve ter sido criado através do comando qcriaui e por esta razão o nome da UI ao qual o módulo pertence **não** é necessário para a execução deste comando.

Para que os projetistas encarregados do desenvolvimento dos módulos possam ter direito de alteração sobre eles, é necessário que uma GD, por exemplo **gabriel**, proceda à alteração da lista de acesso, da seguinte forma:

```
qmudala +baltazar suporte.c suporte.h
qmudala +belchior,baltazar encosto.c encosto.h
qmudala +gaspar,baltazar disparador.c disparador.c
qmudala +gaspar sensor.c sensor.h
```

A tabela abaixo deixa mais clara a atribuição de responsabilidades sobre os módulos:

	baltazar	belchior	gaspar
disparador.c	√		√
disparador.h	√		√
encosto.c	√	√	
encosto.h	√	√	
sensor.c			√
sensor.h			√
suporte.c	√		
suporte.h	√		

Neste ponto os três projetistas estão aptos a darem início ao trabalho de implementação de seus módulos.

Eis abaixo alguns comandos que podem ser executados:

1. `baltazar` deseja criar um diretório de trabalho em sua área de trabalho (isto criará também um *template* de `Makefile`):

```
qcriadir mirra
```

2. `baltazar` deseja consultar o módulo `sensor.c` sobre o qual ele não possue direito de alteração:

```
qrecmod sensor.c
```

   A partir deste momento este módulo é colocado na sua área.

3. `belchior` deseja alterar o módulo `encosto.c` que foi iniciado no dia anterior por `baltazar`. Note que para que isto seja possível este módulo não poderá estar mais reservado (*locked*) para alterações por outro usuário:

```
qeditamod encosto.c
```

4. `gaspar` deseja salvar uma versão intermediária de desenvolvimento do módulo `sensor.c`:

```
qsalvamod sensor.c
```

Suponha que `belchior` não se lembra mais da estrutura da UI `assento`. O comando a seguir permite que ela seja, então, visualizada:

```
qlistaui assento
```

A qualquer momento pode também ser obtida uma lista das UIs que compõem o sistema, através do comando:

```
qlistasis
```

Imaginemos agora que `baltazar` está contente com o estágio atual dos módulos:

- `disparador.c` e
- `disparador.h`

Após tê-los salvo, ele notificará a GD (no caso, `gabriel`). A GD deverá promover estes módulos para o estágio de validação após avaliação dos mesmos, através do seguinte comando:

```
qpromove disparador.c disparador.h
```

A partir deste momento estes módulos passam a estar à disposição da EV. Para validá-los, deverá ser feita uma cópia para consulta dos mesmos, da seguinte forma:

```
qrecmod disparador.c disparador.h
```

Supondo que o módulo `disparador.h` tenha sido declarado inválido pela EV, destarte a GV deverá recolocar este módulo à disposição da ED, através do comando:

```
qinvalido disparador.h
```

Por outro lado, o módulo `disparador.c` que obteve sucesso na sua validação, deverá ser colocado à disposição da ETI pela GV através do seguinte comando:

```
qparati disparador.c
```

O mesmo ocorrerá em relação aos Testes de Integração e de Sistema, ou seja, um determinado módulo poderá obter sucesso (quando então será **promovido**), ou fracasso (quando então será declarado **inválido**), devendo a ED tomar as providências cabíveis.

Quando um conjunto de itens passa no Teste de Sistema, a GT deverá criar uma **configuração**, que nada mais é que atribuir um nome lógico a este conjunto de módulos, através do seguinte comando:

```
qcongela UI_assento_v1.0_alpha encosto.c encosto.h suporte.c suporte.h
```

Caso durante a fase de implantação o cliente encontre algum problema ou deseje alguma melhoria no sistema, ou mesmo a ED descubra uma maneira mais eficiente de implementar determinado módulo que já está em produção, deve ser feita uma solicitação formal de manutenção à GCAC que após análise de impacto de recursos e custos decidirá pela sua aprovação ou não. Se, por exemplo, a decisão do pedido de manutenção ao módulo `encosto.h` for positiva, a GCAC colocará o módulo em questão à disposição da ED para as devidas correções e/ou melhorias utilizando o seguinte comando:

```
qparamanut encosto.h
```

Se em algum momento do desenvolvimento **belchior** decidisse averiguar o andamento do módulo `encosto.c`, ele utilizaria o seguinte comando:

```
qlistalog encosto.c
```

Este comando lista as informações contidas no *log* associado ao módulo. Note que estas informações foram inseridas a cada alteração sofrida por este módulo.

6 Conclusões

Foi apresentado um sistema de controle de alteração e configuração em uso no CPqD-TELEBRÁS. Este sistema foi desenvolvido utilizando a filosofia UNIX de, ao invés de escrever novos programs, combinar os já existentes.

A implementação na forma de *shell scripts* possibilitou boa produtividade dada a inexistência da etapa de compilação dos módulos. Espera-se também um baixo nível de manutenção do sistema pela equipe de desenvolvimento. Como os programas são texto, o próprio usuário (GCAC) pode modificá-los.

Caso venha a existir uma segunda versão do sistema, ela deverá ser escrita em uma linguagem interpretada/compilada como `tcl`[16] ou `perl`[21]. Esta alternativa foi aventada, mas devido a exigüidade de tempo, teve que ser descartada.

A parte do sistema utilizada pela equipe de desenvolvimento já está em uso e começamos a ter sugestões para a evolução do sistema.

Dois produtos similares ao SCAC, embora muito maiores e mais sofisticados, são o `cvs`[8] e o `Aegis`[14]. O `cvs` já estava disponível quando do início do projeto, mas a sua interface foi considerada complicada. Já o `Aegis` só foi tornado público após o início do projeto. Os dois sistemas são muito interessantes e recomendamos a leitura da documentação existente.

Referências

[1] ANSI/IEEE, *IEEE Standard for Software Configuration Management Plans*, Std 828–1990.

[2] Belkhatir, N. e Estublier, J., *Experience with Data Base of Programs*, Proceedings of the ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development Environments, Palo Alto, CA, dezembro de 1986.

[3] Bersoff,E.H. et. al., *Software Configuration Management, An investment in product integrity*, Pretice-Hall, 1980.

[4] Bonanni, L.E. e Salemi, C.A., *Source Code Control Systems - User's Guide*, Bell Labs., 1981.

[5] Buckle, J.K., *Software Configuration Management*, The MacMillan Press Ltd, 1982.

[6] *CCC/Manager - Data Base Administrator's Guide*, ver 2.0, Softool Corp., Goleta, CA, 1991.

[7] Cohen, E.S, et. al., *Version Management in Gypsy*, ACM, 1988.

[8] Grune, D. et. al., *cvs – Concurrent Versions System*, shell script enviado para comp.sources.unix, volume 6, dezembro de 1986.

[9] Grupo Específico de Metodologias de Desenvolvimento de Sistemas, *MUSA - Metodologia Unificada de Sistemas Aplicativos, Versão 1.0*, TELEBRÁS, Departamento de Coodernação de Informática, julho de 1992.

[10] Katz, R.H. e Lehman, T.J., *Database Support for Versions and Alternatives of Large Design Files*, IEEE Trans. on Software Engineering, V10, N2, março de 1984.

[11] Katz, R.H., et. al., *Design Version Management*, IEEE Design & Test, fevereiro de 1987.

[12] Leblang, D.B. e Chase, R.P., *Computer-Aided Software Engineering in a Distributed Workstation Envirnment*, Proceedings of the ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development Environments, Pittsburgh, PA, abril de 1984.

[13] Lori, L.A., *Physical Integrity in a Large Segmented Database*, ACM Trans. on Database Systems, V2, N1, março de 1977.

[14] Miller, P., *Aegis – A Project Change Supervisor*, Free Software Foundation, 1993.

[15] Negri, P.G., *Ambientes de Suporte ao Desenvolvimento de Software – ASSISTE, Documento de abertura de projeto*, CPqD–TELEBRÁS, Departamento de Sistemas de Operação, Divisão de Informática, junho de 1992.

[16] Ousterhout, J., *Tcl: An Embeddable Command Language*, Proceedings of the 1990 Winter USENIX Conference.

[17] Stallman, R.M. e McGrath, R., *GNU Make*, Free Software Foundation, abril de 1993.

[18] Tichy, W.F., *RCS – A System for Version Control*, Software–Practice & Experience 15, julho 1985.

[19] Victorelli, E.Z., *Controle de Versões e Configurações em Ambientes de Desenvolvimento de Software*, Dissertação de Mestrado, DCC-IMECC-UNICAMP, 1990.

[20] Victorelli, E.Z. e Magalhães, G.C., *MVC: Um Modelo para Controle de Versões e Configurações*, Anais do V Simpósio Brasileiro de Engenharia de Software, Ouro Preto, MG, outubro de 1991.

[21] Wall, L., *Programming Perl*, O'Reilly, 1992.