UNIVERSIDADE FEDERAL DE MINAS GERAIS

FACULDADE DE EDUCAÇÃO

Lucas Tadeu Pereira Soares Gomes

CRIAÇÃO DE UM DOCUMENTO DE PADRONIZAÇÃO DE CODIFICAÇÃO JAVA NO LCC

Belo Horizonte

2013

LUCAS TADEU PEREIRA SOARES GOMES

CRIAÇÃO DE UM DOCUMENTO DE PADRONIZAÇÃO DE CODIFICAÇÃO JAVA NO LCC

Projeto de Intervenção apresentado a Faculdade de Educação da (UFMG), como requisito parcial para obtenção do certificado no Curso de Especialização Gestão de Instituições Federais de Educação Superior.

Linha de Pesquisa: Gestão & Tecnologias

Orientador: Professor Antônio Mendes

BELO HORIZONTE

JUNHO DE 2013

CRIAÇÃO DE UM DOCUMENTO DE PADRONIZAÇÃO DE CODIFICAÇÃO JAVA NO LCC

Projeto de Intervenção apresentado a Faculdade de Educação da (UFMG), como requisito parcial para obtenção do certificado no Curso de Especialização Gestão de Instituições Federais de Educação Superior.

Linha de Pesquisa: Gestão & Tecnologias

Orientador: Professor Antônio Mendes

Aprovado em onze de julho de 2013

BANCA EXAMINADORA

_____

Antônio Mendes Ribeiro – UFMG

_____

Antônio Otávio Fernandes – UFMG

_____

Eucídio Pimenta Arruda – UFMG

Agradecimentos

Agradeço a Deus, ao Universo, à meus pais Luiz e Joana, minha irmã Carol, à Luana e aos meus colegas do LCC, pelo apoio e entendimento nos momentos de dificuldade. À UFMG e a toda a equipe do GIFES pela oportunidade e ao meu orientador Antônio pela luz ao longo do caminho.

Dedico este trabalho aos meus pais.

“A única possibilidade de descobrir os limites do possível é aventurar-se um pouco além deles, para o impossível”

Arthur C. Clarke

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas

CENAPAD-MGCO – CEntro NAcional de Processamento de Alto Desempenho de Minas Gerais e Centro-Oeste

BD – Banco de Dados

BDTD – Biblioteca Digital de Teses e Dissertações

LCC – Laboratório de Computação Científica

LDAP – *Lightweight Directory Access Protocol*

MVC – *Model-View-Controller* e Modelo-Visão-Controlador

NIP – Número de Identificação de Pessoa

UFMG – Universidade Federal de Minas Gerais

XP – *eXtreme Programming*

Resumo

Uma das etapas mais importantes no processo de desenvolvimento de software é a da codificação, onde a ideia de elaboração de um sistema realmente se tornará um novo sistema. Esta etapa conta, em geral, com a participação de vários desenvolvedores, muitas vezes distribuídos em equipes distintas para a execução da codificação em paralelo. Além disto, novos desenvolvedores podem ser incluídos nos projetos, demandando tempo para a compreensão do que já existe para poderem finalmente participar ativamente do desenvolvimento dos sistemas. Para garantir uma boa comunicação entre os desenvolvedores envolvidos, a existência de um padrão na codificação é necessária. Além da etapa de codificação, grande parte da vida útil do software é gasta na sua manutenção. Quando a codificação do sistema não segue um padrão bem estabelecido e o código presente no sistema é confuso, grande é a dificuldade na manutenção do código e na própria adaptação do sistema a novos requisitos. A utilização deste padrão garante uma melhora significativa na comunicação e na agilidade de produção de software, tendo uma equipe mais entrosada e com maior agilidade no entendimento de códigos produzidos por colegas, além de proporcionar um processo de manutenção de software mais simples e proveitoso. No Laboratório de Computação Científica da UFMG (LCC) vários sistemas Java são mantidos, a todo dia demandando novas funcionalidades e manutenção de código. Este trabalho consiste na elaboração de um documento de padronização de codificação Java, baseado nos sistemas já existentes no LCC, visando a melhora na comunicação das equipes já existentes e na adaptação de novos servidores, profissionais terceirizados e estagiários às equipes do laboratório. O documento abordará diversos aspectos, estruturais e estéticos, de codificação Java, buscando agregar informações atuais e significativas para a produção de um código de maior qualidade no LCC.

Sumário

1. Apresentação

O desenvolvimento de sistemas é uma atividade de vital importância na sociedade contemporânea. Grandes sistemas diariamente são utilizados pelas pessoas, propiciando maior segurança, maior comodidade e muitos outros ganhos. Esta utilização cresce a cada dia, mostrando o quão necessário é o processo de desenvolvimento de software. Este processo é dividido em diversas fases, e uma fase muito importante, muitas vezes longa, é a fase de codificação.

A codificação é a transformação das ideias e requisitos em um sistema funcional. As definições de como o sistema deverá ser, incluindo seu comportamento, sua aparência, sua interface, são transformadas em código de máquina funcional, executando efetivamente suas designações.

Na Universidade Federal de Minas Gerais, um dos órgãos responsáveis pelo desenvolvimento de sistemas é o LCC – Laboratório de Computação Científica. Nele são desenvolvidos e mantidos diversos sistemas utilizados na universidade. Como exemplo temos o sistema Perfil e a Biblioteca Digital de Teses e Dissertações, usando DSpace.

A codificação realizada no LCC utiliza diversas tecnologias, próprias para seus sistemas especificamente. Diferentes tecnologias permitem a codificação de sistemas, assim como a construção de um mesmo objeto pode ser realizada utilizando ferramentas distintas. Uma das tecnologias utilizadas no LCC é a linguagem de programação Java[1], da empresa Oracle.

O Java é uma linguagem de programação muito utilizada, muito atual e com muitos recursos, permitindo uma codificação rica e a criação de sistemas complexos de forma segura e completa. A linguagem Java permite a criação de arquivos texto contendo as instruções de

[1] http://www.oracle.com/technetwork/java/javase/downloads/index.html

execução de um programa, que são traduzidos para a linguagem de máquina através de um compilador. Essa programação é feita utilizando palavras chave, operadores e variáveis neste arquivo texto.

Atualmente não existe na equipe do LCC de desenvolvimento um documento definindo um padrão a ser utilizado na codificação das aplicações Java mantidas pelo setor, embora exista um documento com tais definições para a criação e manipulação de bancos de dados, que é de grande auxílio para o setor. A criação deste documento para a linguagem Java então é interessante para auxiliar na interação entre diferentes membros da equipe do LCC, na integração de novos servidores, profissionais terceirizados e bolsistas.

1.1. LCC – Laboratório de Computação Científica

Fundado em 1981, o LCC foi criado pela iniciativa de um grupo de professores dos departamentos de Física e de Ciência da Computação da UFMG, agrupando recursos computacionais já existentes oferecendo apoio computacional às atividades de ensino e pesquisa. Em 1997 no LCC foi inaugurado o CENAPAD-MGCO, Centro Nacional de Processamento de Alto Desempenho em Minas Gerais e Centro-Oeste, que disponibiliza acesso a um cluster de alto desempenho para a comunidade acadêmica. O laboratório também desenvolve e oferece serviços administrativos a setores da UFMG, a unidades acadêmicas e à própria comunidade acadêmica.

O LCC possui 5 subdivisões: Computação Científica, Desenvolvimento, Sistemas (Análise/Operação), Administração, Atendimento e Help Desk. A figura 1 mostra o organograma do órgão.

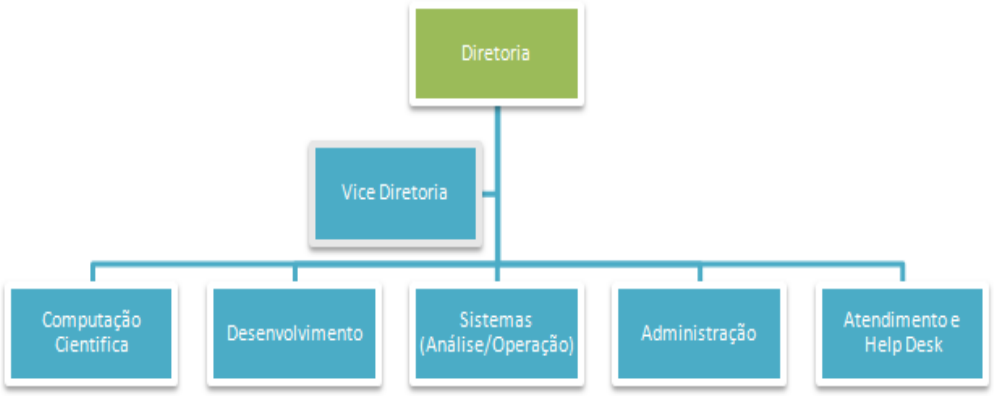


Figura 1 Organograma do LCC

Alguns dos serviços e sistemas disponibilizados pelo LCC são: Suporte LCC – CENAPAD, minhaUFMG, Computação Científica, UFMG Virtual (Moodle), Questionários Online (QPL), Biblioteca Digital, Softwares Licenciados, Acesso ao Portal CAPES, Repositório de Objetos UFMG, Sistema Perfil, Quem Sabe, OPUS (LABORATÓRIO DE COMPUTAÇÃO CIENTÍFICA, 2011).

Os diferentes sistemas existentes no LCC são desenvolvidos utilizando tecnologias mais apropriadas para seu funcionamento, exigindo analistas com domínio em diferentes tecnologias, incluindo php, Java, MySQL, Linux, Lotusscript, Oracle. O setor de desenvolvimento do LCC conta atualmente com:

- Oito analistas/desenvolvedores: responsáveis pelo levantamento, desenvolvimento e manutenção dos sistemas e serviços.
- Um administrador de banco de dados: responsável por gerenciar e monitorar as atividades de banco de dados.
- Uma gerente de desenvolvimento: responsável pelo planejamento e coordenação da execução dos trabalhos de toda a equipe e as atividades relativas ao setor.

1.2. Modelos de Desenvolvimento e a Etapa de Codificação

O desenvolvimento de um sistema segue um processo lógico que tem origem na concepção da ideia até a implantação e utilização do software. São vários modelos de desenvolvimento de software: em cascata, incremental, em espiral, modelos ágeis.

Talvez o mais conhecido seja o modelo em cascata, que tem este nome devido ao sequenciamento de cada etapa, se assemelhando a uma cascata. Este encadeamento se dá devido a todo o produto da etapa anterior ser utilizado para dar início à próxima etapa do processo, até que o sistema esteja completo e operacional.

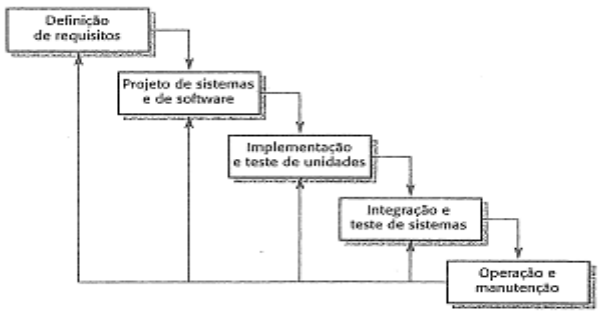


Figura 2 Modelo em cascata (SOMMERVILLE, 2004)

O maior problema do modelo em cascata é sua inflexibilidade e dificuldade em lidar com as constantes mudanças nos requisitos de software, levando à soluções paleativas e muitas vezes com prazos curtos. Novos modelos surgiram, buscando solucionar problemas existentes no modelo em cascata, como o incremental. Este modelo cuida dos requisitos gradativamente, atribuindo relevância e desenvolvendo as funcionalidades de acordo com as necessidades dos clientes. Este ciclo de desenvolvimento é repetido enquanto o software não está completo, sendo possível a inserção de novos requisitos até que o cliente fique satisfeito.

O modelo em espiral é atualmente muito conhecido, e consiste em ciclos bem definidos, contando com quatro etapas: definição de objetivos, avaliação e redução de riscos, desenvolvimento e validação, e planejamento da próxima fase. Um fator marcante neste modelo é a explicita avaliação de riscos, tarefa importante na gerência de projetos. Cada etapa

permite que outros modelos sejam utilizados, permitindo um desenvolvimento mais apropriado às necessidades dos requisitos.

O desenvolvimento de código fonte propriamente dito é realizado na etapa de implementação. O tempo de duração desta etapa pode variar muito, dependendo principalmente do tamanho do sistema que está sendo construído. Quando esta etapa é realizada por equipes de desenvolvedores, passos distintos da construção podem ser feitos em paralelo, independentes, para depois serem unidos no sistema em sua completude. Tal recurso é muito utilizado na programação Java graças à suas muitas características, neste caso, a de modularização. Tais características não serão aprofundadas no presente trabalho, podendo ser vistas no trabalho do autor Deitel, "Java, how to program, 7th edition" (DEITEL, 2006).

Em 2001, uma nova abordagem nas metodologias de desenvolvimento tornou-se popular. As metodologias ágeis, que em detrimento aos modelos clássicos, tinha como princípio uma importância maior nas interações entre os indivíduos, os clientes e no software executável, e dava uma importância secundária aos processos e ferramentas, documentação e demais elementos antes considerados primordiais. Exemplos destas metodologias são o SCRUM e o XP – *Extreme Programming*[2] (SOARES, 2010).

A divisão de tarefas na codificação pode permitir um ganho considerável na etapa de codificação, porém ela acarreta um tempo de compreensão e de arranjos para que desenvolvedores diferentes possam entender e trabalhar na conexão entre estas diferentes tarefas. Quando não há um rigor maior nas definições de codificação, não há um padrão mínimo, há o perigo de se gastar mais tempo no entendimento de um código alheio do que na codificação e na execução das tarefas.

Outro acontecimento comum é a rotatividade das equipes de desenvolvimento, onde novos desenvolvedores são integrados aos projetos, principalmente em projetos de sistemas maiores, que podem ficar meses sendo desenvolvidos. Ao ingressarem em uma equipe, os desenvolvedores precisam de tempo para compreender códigos, algumas vezes incompletos,

[2] Programação Extrema

algumas vezes obscuros. A utilização de um padrão no processo de codificação permite que o entendimento de códigos de outros desenvolvedores seja mais dinâmico pelos novos desenvolvedores, permitindo que eles possam iniciar a execução de suas tarefas mais rapidamente.

1.3.O Problema da Manutenção de Software

A manutenção de software consiste na fase final do ciclo de vida de um software. Esta fase existe para garantir que um sistema continue operacional e que continue atendendo às expectativas do cliente. Swanson, *apud* LIENTZ *et. al.*(1978), resume as atividades de manutenção de software em:

- Manutenção corretiva: ocasionada por erros no sistema, corrigindo falhas para manter o sistema funcional;
- Manutenção adaptativa: realizada prevendo futuras alterações no ambiente de execução e de processamento de dados; e
- Manutenção perfectiva: buscando eliminar ineficiências, otimização de execução ou melhora da manutenibilidade.

A manutenção de software envolve intrinsecamente a alteração de código fonte já existente, seja incrementando-o ou editando-o. Para que a manutenção seja bem feita, o responsável por ela precisa compreender o código a ser trabalhado e seu contexto, o que pode ser problemático caso o código não seja claro e bem formatado. Além disso, devido às práticas de orientação a objetos, este código fonte pode estar distribuído ao longo de diferentes classes, gerando assim uma necessidade de maior cuidado na análise e compreensão do mesmo.

O responsável pela manutenção no código não possui o mesmo contexto do funcionamento do sistema que o desenvolvedor original do código em manutenção, por isso é necessário que seja bem claro o código para que seu entendimento seja fácil por qualquer outro profissional da equipe. Mesmo quando o responsável pela manutenção é o

desenvolvedor quem realizou originalmente a codificação a atividade pode se revelar muito complexa devido a fatores humanos como esquecimento ou mudança na maneira de pensar na tarefa de codificação.

2. Objetivos

A criação de um documento que padroniza a utilização da linguagem Java no desenvolvimento e manutenção dos sistemas no LCC é o objetivo principal deste trabalho. Este documento padronizará os diversos elementos da linguagem Java: nomenclatura de variáveis, de classes, recuo de texto, espaçamento e definição de blocos de código, utilização de letras minúsculas e maiúsculas, e outros que possam surgir ao longo do desenvolvimento do trabalho.

A existência deste documento permitirá uma maior facilidade na comunicação entre a equipe no desenvolvimento de seus sistemas, facilitará no ingresso de novos servidores, profissionais terceirizados e bolsistas. Em especial os bolsistas poderão aproveitar mais de seu tempo limitado no setor para o aprendizado enquanto exercem tarefas de desenvolvimento, já que tal padronização permitirá uma integração mais rápida à equipe.

A integração de novos servidores e de profissionais terceirizados ao LCC traz profissionais já experientes na área de tecnologia da informação, que podem dominar tecnologias diferentes das tecnologias livres utilizadas no LCC. Porém, as tecnologias possuem aplicações semelhantes, sendo utilizadas com o fim de desenvolvimento, e assim apresentam uma base parecida. Isso garante uma maior facilidade no aprendizado e na adaptação de um novo profissional ao setor.

O mesmo vale para a integração de bolsistas ao setor, aumentando o aproveitamento destes e o aprendizado das tecnologias empregadas no setor, contribuindo para um grande desenvolvimento técnico.

Visando também a comunicação do LCC com outros setores de criação e de desenvolvimento e tecnologia da universidade, uma vez que os sistemas elaborados no LCC sejam apresentados sob um padrão bem definido, a colaboração entre os setores pode ser realizada de maneira ágil e completa.

3. Fundamentação Teórica

3.1. Linguagem Java

A linguagem Java é uma linguagem de desenvolvimento de software orientada à objetos. Baseada na linguagem C++, o projeto de criação da linguagem Java foi iniciado em 1991 pela empresa Sun *Microsystems* e teve seu nome sugerido em uma visita a um café local. Seu uso se tornou popular graças à grande explosão da *World Wide Web* (WWW), a Internet, e o uso do Java para adicionar conteúdo dinâmico às páginas da WWW. O Java foi oficialmente anunciado em 1995.

O Java se baseia em classes para a construção de aplicações. Cada classe deve ser escrita em um arquivo Java, que então é compilado para um código intermediário que é verificado e interpretado pelo *framework Java runtime environment*, gerando o código de máquina para que os comandos sejam executáveis e as operações sejam exercidas pelo computador. A Figura 3 demonstra o processo.

A estrutura de um programa Java utiliza diversos recursos de programação, definidos a seguir (DEITEL, 2006):

- Instruções: São as operações únicas realizadas pelo processador dentro de um programa. São encerradas pelo uso do sinal ";" (ponto-e-vírgula).
- Variáveis: São entidades que, após definidas, referenciam endereços de memória específicos para armazenar valores.
- Tipos: Determinam os tipos de valores que serão manipulados, definindo a quantidade de bytes em memória reservados para eles e como eles serão tratados.
- Constantes: São entidades que armazenam valores, porém não permitem alteração em seu valor após serem definidas.
- Funções: São trechos de código que realizam um determinado processamento sobre valores e retornam o resultado destas operações.

- Métodos: São trechos de código que realizam determinado processamento, alterando os valores de entrada, mas sem retornar o resultado de alguma operação.
- Palavras reservadas: São palavras próprias da linguagem Java que determinam a execução de um programa e que não podem ser utilizadas em outros contextos da aplicação.
- Estruturas de repetição: Determinam que certo trecho de código será repetido até que uma determinada operação lógica seja atendida.
- Condicionais: Alteram o fluxo de execução de um programa com base em decisões sobre operações lógicas.
- Operadores lógicos e aritméticos: Realizam operações lógicas e/ou matemáticas, permitindo testar e comparar entidades e definindo valores em funções e atribuições.
- Bibliotecas: São conjuntos de operações recorrentes que podem ser usadas nas aplicações que estão sendo desenvolvidas, garantindo diversos recursos necessários à construção de aplicações.
- Referências: São elementos que guardam endereços de acesso à memória.
- Classes: São a definição dos objetos que serão criados. Elas possuem todos os recursos disponíveis na linguagem para criar uma representação de dados e operações que serão executadas.
- Objetos: São a representação das classes, possuindo dados manipuláveis para a realização de operações e a execução efetiva de um aplicativo.
- Mensagens: São as informações trocadas entre classes.
- Comentários: São informações adicionais no arquivo Java que não são traduzidas para código de máquina. Usados para dar informações adicionais sobre o processamento que está sendo realizado e como o está sendo realizado.

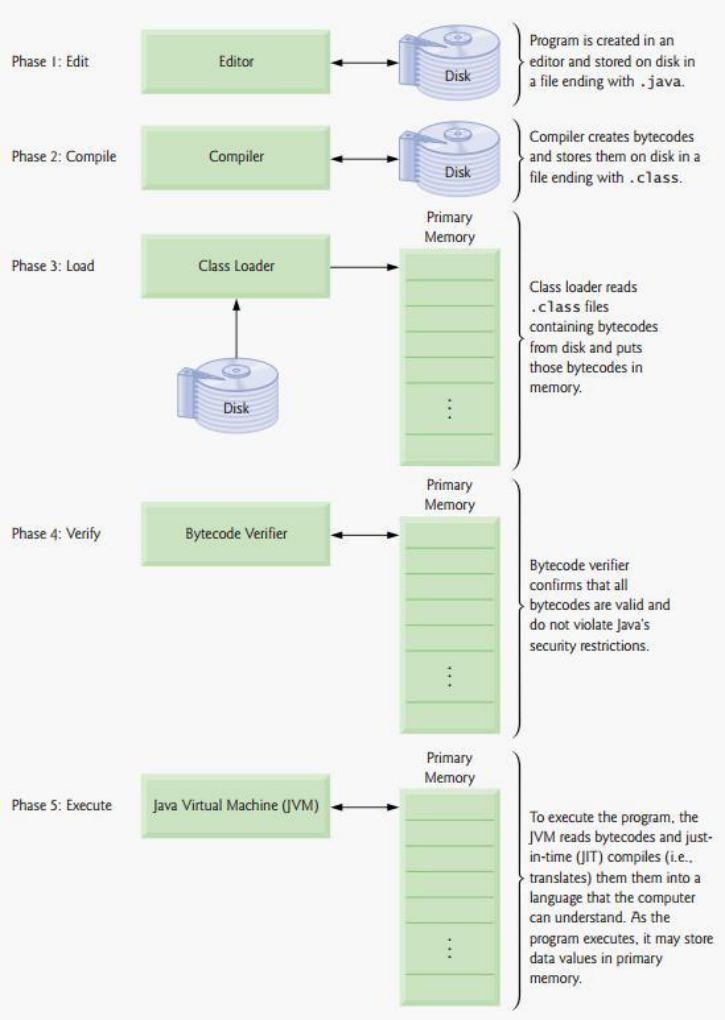


Figura 3 Fases de execução de uma aplicação Java (DEITEL, p. 12, 2006)

Por ser uma linguagem orientada a objetos, o Java apresenta suas características principais, que são o encapsulamento, a herança e o polimorfismo. Encapsulamento é a capacidade de objetos Java apresentarem diferentes tipos de acesso aos seus atributos e de particionar sua composição, proporcionando uma maior flexibilidade no software. Herança é a capacidade de uma classe reutilizar e partilhar atributos, métodos e funções com outras classes específicas. Polimorfismo é a possibilidade de uma classe mudar o seu comportamento de acordo com a utilização de um tipo mais abstrato de dados em momento de execução.

Uma aplicação Java consiste em diversos objetos realizando operações distintas e interagindo entre si através de mensagens para que no final um processamento maior seja realizado. Por exemplo, uma classe é encarregada de criar uma janela no sistema operacional e outra classe informa que na parte mais baixa desta janela será exibida uma mensagem ou um menu.

Trabalhar com poucas classes pode ser uma tarefa simples, mas quando estamos lidando com sistemas de grande porte, como os sistemas da UFMG, muitas classes são usadas para realizar o processamento de tarefas específicas, o que pode tornar o processo de desenvolvimento muito complexo, caso não se tenha o devido cuidado para ter um código legível.

3.2.Java Code Conventions

A própria *Sun Microsystems*, empresa criadora da linguagem Java, recomenda que a codificação de software apresente um padrão de fácil entendimento no trabalho em equipe. Para tal, está disponível o documento Java *Code Conventions* (SUN MICROSYSTEMS, 1997), que apresenta o padrão de codificação utilizado pela própria empresa no desenvolvimento dos códigos-fontes Java. Este documento apresenta padrões para:

- Nomes de arquivos;
- Organização de arquivos e pastas;
- Indentação;
- Comentários;
- Declarações de tipos;
- Expressões;
- Estruturas de controle;
- Estruturas de repetição;
- Espaços em branco;
- Nomeação de tipos e variáveis;
- Práticas de programação.

Já os principais argumentos para a padronização de código apresentadas são:

- 80% do tempo de vida de um software é direcionado à manutenção.
- Dificilmente um software é mantido toda a vida pelo mesmo autor.

- Padrões aumentam a legibilidade do software, permitindo aos desenvolvedores compreenderem mais rapidamente e com mais facilidade um novo código.
- Se o código fonte for vendido como um produto, é necessário garantir que ele está bem embalado e claro como qualquer outro produto criado.

3.3. Legibilidade do Código Fonte

O desenvolvimento de software é um processo de codificação, onde um desenvolvedor, ou uma equipe de desenvolvedores, atua construindo o software desejado. Em geral, equipes trabalham em conjunto na codificação, atribuindo diferentes classes para diferentes desenvolvedores realizarem a codificação.

Se não há um "comum acordo" entre a forma de codificação dos desenvolvedores, em um futuro onde seja necessária uma manutenção em uma classe que seja realizada por um desenvolvedor diferente daquele que a escreveu originalmente, o tempo dessa manutenção aumentará consideravelmente. Além do problema a ser resolvido na lógica de programação, ainda será necessário o tempo para o entendimento do código alheio.

LIENTZ e SWANSON (1980) e ARFA et al (1988-1989) *apud* RAMOS (2009) estimam que 48,8% e 44,63% do orçamento de um software é gasto em manutenção, contra 43,3% e 46,59% respectivamente é gasto no desenvolvimento; além de custos não monetários como adiamento de oportunidades de desenvolvimento e insatisfação de clientes.

As tarefas necessárias para a manutenção de código fonte abrangem entender o funcionamento do software; interpretar os elementos da linguagem de programação, elementos de interface e desempenho; analisar, avaliar, projetar, codificar e testar as modificações (RAMOS, 2009).

Destas tarefas, um código legível e bem construído pode diminuir o tempo dos dois primeiros elementos da manutenção, permitindo um ganho na efetiva elaboração e correção de funcionalidades.

VON MAYRHAUSER et. at. *apud* KOSKINEN *et. al.* (2004) conduziram estudos empíricos buscando levantar quais os elementos mais comuns com carência de informação relevante ao entendimento do funcionamento do sistema. KOSKINEN resume todos os pontos em carência levantados em uma tabela (ver tabela 1), identificando onde e com que frequência aparecem.

Rank	Information need	Frequency	Source			
			S	E	D	H
1	Domain concept descriptions	68	x		x	
2	Location and uses of identifiers	54	x			
3	Connected domain–program–situation model knowledge	41	x		x	x
4	List of browsed locations	33				x
5	List of routines that call a specific routine	29	x			
6	A general classification of routines and functions such that if one is understood, the rest in the group are understood	19	x		x	
7	Format of data structure plus description of what field is used for in program and application domain, expected field values and definitions	18	x	x	x	
8	History of past modifications	18	x		x	x
9	Bug behavior isolated	18	x	x		
10	List of executed statements and procedure calls, variable values	17		x		
11	Call graph display	16	x			
12	Variable definitions, including why necessary and how used, default and expected values	14	x	x	x	
13	Location of desired code segment	12	x	x		
14	Directory layout/organization: include files, main file, support files, library files; file structures	12	x			
15	Highlighted begin/ends of control blocks	11	x			
16	Location of where to put changes	10	x		x	x
17	Conditions under which a branch is taken or not, including variable values	9	x	x		
18	Exact location to set breakpoint	7	x	x		x
19	Location and description of library routines and system calls	7	x		x	
20	Ripple effect	7	x	x		
21	Naming conventions separated by systems or library objects that use them; rules for naming new procedures	7	x		x	
22	Expected program state, e.g. expected variable values when procedure is called	7	x	x	x	
23	Good direction to follow given what is already known, possible program segments to examine	7	x	x	x	
24	Nesting level of a particular procedure	7	x			

The 'Frequency' column states how often the subjects needed the information in the four studies altogether. The 'Source' columns refer to the sources from which the information could be found, the categories being S (source code), E (code execution), D (documentation and other written material), and H (session history).

Tabela 1 Carência de informação x Frequência nos projetos (KOSKINEN et. al., 2004)

Os pontos apresentados na Tabela 1 foram identificados em sessões reais de manutenção de software, ordenados por maior frequência e com indicação de quais momentos eles são noticiados. Estes momentos são: S – momento de análise do código fonte (source code); E – momento de execução de código (code execution); D – momento de análise de

documentação e demais documentos escritos (documentation and other written material); e H – momento da própria sessão de manutenção (session history).

Fica claro perceber pelos cinco primeiros itens que a carência de informação é mais frequente no momento de análise do código, devido à pouca preocupação em criar um código legível, de fácil compreensão e com comentários de baixa qualidade.

A carência de descrição acerca de conceitos de domínio do problema gera grande dificuldade para desenvolvedores que não fizeram parte do desenvolvimento original, devido à falta de contexto da aplicação para o entendimento do funcionamento do sistema como um todo. Estas informações muitas vezes não estão documentadas ou apresentam informações insuficientes para a realização da tarefa de manutenção.

A falta de preocupação na utilização de identificadores e da localização da definição dos mesmos causa problemas de identificação de atributos e da finalidade dos mesmos, exigindo de quem realiza a manutenção a pesquisa pela definição e pelos usos destes atributos para a realização de qualquer alteração que se relacione com eles.

O conhecimento do modelo de desenvolvimento é necessário tendo em vista a existência de diferentes padrões de desenvolvimento (como exemplo temos o MVC[3]) contendo partes importantes da solução diluídas em diferentes partes do código, até mesmo em classes totalmente distintas. Como no primeiro item, muitas vezes a arquitetura do sistema não está devidamente documentada, com poucas informações dos padrões e *frameworks* utilizados no desenvolvimento da aplicação.

A lista de locais navegados é construída ao longo da sessão de manutenção, agrupando locais que possuem informação necessária à compreensão do código ao longo do próprio código. Muitas vezes quem realiza a manutenção não tem acesso à tal elemento ou não o produz, tornando necessária a busca pelos locais mais de uma vez.

[3] Ver capítulo 3.4

O quinto elemento é uma lista de métodos que utilizam determinado método. Quando esta lista não existe, não há conhecimento de quais métodos utilizam o método que está sendo modificado, podendo causar efeitos colaterais na sua modificação que não serão identificados até que um novo erro se apresente, seja nos testes ou na execução do sistema. A maioria das ferramentas atuais já permitem acesso à esta lista, colaborando para uma melhor realização de codificação e navegabilidade ao longo do código.

Este cinco pontos de carência de informação apresentam frequência acima de 25%, no estudo realizado por KOSKINEN et. al.; sendo que quatro deles são encontrados no código fonte, principalmente devido à falta de padrões estabelecidos na construção dos sistemas e a não documentação de informações importantes sobre a organização deste código.

Apresentando um código legível, bem organizado e documentado, a tarefa de manutenção e de entendimento do código não apresentará tantos problemas como os apresentados na tabela 1. Eles serão fáceis de entender, e consequentemente, fáceis de serem submetidos à manutenção.

BOSWELL e FOUCHER (2012) partem da ideia de que “Códigos devem ser fáceis de entender” (2012, p. 2) para elaborar a Teoria Fundamental da Legibilidade: “Códigos devem ser escritos de modo a minimizar o tempo necessário para sua compreensão” (2012, p. 3). O que torna um código melhor é um código que respeite espaçamento, alinhamento de estruturas, necessidade de menos linhas escritas, dentre outros aspectos. Um código mais enxuto e simples é preferível. Porém nem sempre é a melhor opção, pois alguns elementos que tornam um código enxuto podem parecer confusos, dificultando o seu entendimento.

Por isso a Teoria Fundamental da Legibilidade é tão importante. Seguindo este princípio, garantimos que mais da metade do esforço gasto no ciclo de vida de um software, que é a sua manutenção, seja realizado sem o dispêndio de força desnecessária, além da garantia de uma equipe com boa comunicação ao longo de todo o processo de desenvolvimento de software.

3.4. Padrões de Projetos

Os Padrões de Projeto são soluções propostas por Gamma *et. al.* (2006) em seu livro "Padrões de Projeto". Baseados na obra de Christopher Alexander, "Uma linguagem de padrões", em que são apresentados padrões de projeto com soluções para problemas recorrentes na arquitetura, o mesmo princípio foi aplicado para solucionar os problemas recorrentes na arquitetura de um software.

Segundo Gamma *et. al.*, os padrões de projeto são "descrições de objetos e classes se comunicam, customizados para resolver um problema de projeto genérico em um contexto particular"[4] (GAMMA et. al., p. 13, 2006).

O MVC – Modelo-Visão-Controlador – é um exemplo de padrão de projeto muito utilizado. Este padrão consiste na divisão do sistema em três componentes principais: modelo, responsável pelo controle da aplicação; visão, apresentando ao usuário a tela da aplicação; e controlador, recebendo entrada de dados do usuário e interpretando ações a serem tomadas pelo modelo.

Os padrões de projeto são divididos em três categorias: padrões criacionais, padrões estruturais e padrões de comportamento. Cada categoria possui diversos padrões de projeto que podem ser utilizados para resolver problemas comumente encontrados no momento de codificação de software. Alguns exemplos de padrões de projeto são:

- Padrão *observer*[5]: padrão de comportamento, onde um objeto é responsável por observar diversos outros objetos e quando um objeto muda, o *observer* cuida para que todas as dependências do objeto alterado tomem conhecimento desta mudança.

[5] Observador, tradução nossa

- Padrão *Decorator*[6]: Padrão estrutural, insere uma responsabilidade adicional a um objeto de forma dinâmica (durante a execução do sistema), permitindo que um objeto seja manipulado sempre de uma mesma forma, mas realize tarefas distintas de acordo com o contexto de execução.

- Padrão *Singleton*[7]: Padrão de criação, garante que uma classe terá apenas uma instância, proporcionando um ponto global de controle. Por exemplo, garante que exista apenas uma classe de configuração e controle de um sistema.

- Padrão *Factory Method*[8]: Padrão de criação, disponibiliza uma interface para a criação de objetos, permitindo que uma subclasse decida qual o tipo de classe será criada, dinamicamente. Permite que diferentes classes sejam criadas de acordo com o contexto de execução do sistema.

A grande vantagem da utilização de padrões é o ganho de tempo no processo de desenvolvimento de software ao lidar com problemas comumente encontrados nesta etapa, tornando desnecessária a mobilização de um ou mais programadores para decidirem como lidar com tais problemas.

3.5. Sistemas LCC

O LCC possui diversos sistemas construídos com a tecnologia Java. Muitos destes sistemas foram feitos previamente à composição da atual equipe de desenvolvimento do órgão, de forma que os desenvolvedores originais não estão mais acessíveis.

[6] Decorador, tradução nossa
[7] Coisa única, tradução nossa
[8] Método fábrica, tradução nossa

Sistema Perfil:

> O Programa Sempre UFMG, objetivando estreitar cada vez mais o relacionamento entre a Universidade e seus ex-alunos, criou o PERFIL UFMG – Rede de Oportunidades, ferramenta que, pela captação e divulgação, via Internet das mais diversas modalidades de trabalho, oferece aos Ex-alunos UFMG um meio eficaz de colocação profissional. Ao mesmo tempo, permite às organizações em geral, acesso a um cadastro organizado de profissionais qualificados pela universidade. O Sistema Perfil foi desenvolvido pelo LCC e colocado à disposição da comunidade universitária em 2009 (LABORATÓRIO DE COMPUTAÇÃO CIENTÍFICA, [2011]).

Bibliotecas Digitais:

> O projeto da Biblioteca Digital de Teses e Dissertações da UFMG - BDTD UFMG - tem por objetivo disponibilizar, para as comunidades interna e externa, a produção científica, oriunda dos programas de pós-graduação stricto sensu da universidade. Inclui registros correspondentes a textos completos digitalizados em formado pdf, partes de textos, devidamente autorizados pelos autores, ou referências e resumos referentes a teses e dissertações, proporcionando rapidez e facilidade de busca e acesso (LABORATÓRIO DE COMPUTAÇÃO CIENTÍFICA, [2011]).

Repositório de Objetos de Aprendizagem UFMG:

> O Repositório de Objetos de Aprendizagem da UFMG foi desenvolvido numa parceria GIZ (Diretoria de Inovação e Metodologias de Ensino) e LCC [...] com objetivo de abarcar objetos de orientação pedagógica, criados em suporte digital, que sejam plausíveis de uso no âmbito da prática de ensino. Deste modo, toda comunidade acadêmica terá a sua disposição um ambiente adequado para guarda, disponibilização e recuperação de materiais digitais apropriados às práticas de ensino/aprendizagem (LABORATÓRIO DE COMPUTAÇÃO CIENTÍFICA, [2011]).

Geração Automática de NIPs e Impressão de NIPs pela Web:

> Cada NIP é um número randômico com dez dígitos decimais, e funciona como uma senha que só pode ser usada uma única vez. NIPs são exigidos em ocasiões especiais, como o cadastro inicial no minhaUFMG, ou para a obtenção de senhas (LABORATÓRIO DE COMPUTAÇÃO CIENTÍFICA, 2011). Estes sistemas cuidam da criação e acesso aos NIPs.

O LCC também possui os sistemas de Manutenção da base de dados do repositório LDAP[9] e o sistema Esfera Semântica, em desenvolvimento no presente momento.

Muitos dos sitemas desenvolvidos no órgão tiveram ínicio antes da atual formação da equipe. Em uma época onde não havia tanto conhecimento e preocupação com padrões de desenvolvimento, o que deixou sistemas codificados de formas distintas. A seguir está um relato de um dos analistas da equipe quanto a utilização de padrões no LCC:

> Nos sistemas mais recentes tem-se adotado um padrão de se agrupar as classes do projeto de acordo com suas funcionalidades. Para esse agrupamento utiliza-se como referência principalmente o modelo MVC - Model / View / Controller de forma a se dividir a responsabilidade de cada uma dessas camadas MVC entre as classes do projeto. Quanto aos sistemas herdados[10], o único que realmente foi desenvolvido com base em um padrão é o Portal minhaUFMG (muito em função de ter sido construído em cima de um framework). Os demais não possuem padronização em sua codificação. O que tenta-se fazer é, à medida que tais sistemas são evoluídos, tentar enquadrá-los aos padrões atualmente adotados. Isso, entretanto, nem sempre é possível pois em muitos casos implicaria na reescrita de grande parte do código o que, além de demandar tempo, pode introduzir erros nas aplicações.[11]

E um relato sobre a ocorrência de dificuldades devido à compreensão de código de sistemas herdados.

> Sim [já houve tal dificuldade] . O Portal minhaUFMG é o caso mais clássico. Essa dificuldade no entendimento do código, entretanto, não é decorrente de código mal escrito. Ela ocorre pela complexidade do framework uPortal e da própria aplicação. Além disso, a autenticação no Portal anteriormente era feita através da validação do usuário em bases de documentos Lotus Notes. Para se alcançar esse objetivo foi necessário utilizar uma API externa desenvolvida e disponibilizada pela IBM e também escrever código de comunicação em mais baixo nível entre os ambientes Java e Lotus Notes.[12]

---

[9] *Lightweight Directory Access Protocol*
[10] Sistemas prévios ao ingresso na equipe sob responsabilidade do analista
[11] Relato do analista de sistemas Renato, atuante no LCC
[12] Relato do analista de sistemas Renato, atuante no LCC

4. Estratégia de Ação

Para a elaboração do documento de padronização de codificação para o LCC, primeiramente serão realizados estudos nas aplicações Java desenvolvidas no LCC. Sistemas como o Perfil e o DSpace terão seu código analisado e padrões de escrita de código identificados serão utilizados para a criação de documentos com tais padrões sistematizados.

O artigo Java Code Conventions (SUN MICROSYSTEMS, 1997) será utilizado para guiar os principais elementos a serem definidos no documento de padronização de codificação. Com base nos sistemas já existentes, na experiência dos desenvolvedores envolvidos e no documento da Sun Microsystems será definido o padrão a ser utilizado nos elementos de codificação: Nomes de arquivos; Organização de arquivos e pastas; Indentação; Comentários; Declarações de tipos; Expressões; Estruturas de controle; Estruturas de repetição; Espaços em branco; Nomeação de tipos e variáveis; e Práticas de programação.

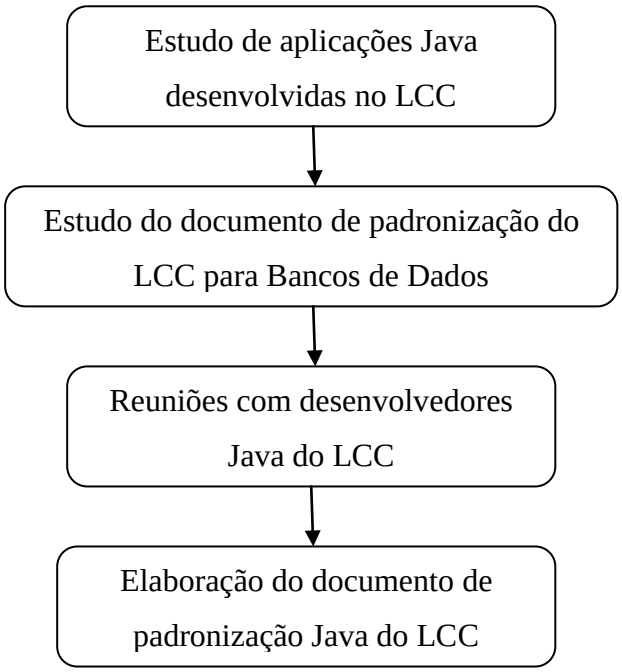
O documento de padronização de Bancos de Dados[13] (BD) utilizados pelas aplicações do LCC será então analisado, buscando correlacionar seu conteúdo com os padrões sistematizados pela análise dos códigos dos sistemas Java para que sejam levantadas relações entre os dois documentos.

Com estas relações, será criado um documento base para reuniões envolvendo responsáveis pelos sistemas Java do LCC. Nestas reuniões será discutida a melhor forma de sintetizar estes padrões identificados nos sistemas Java com o documento já existente de padronização de BDs, além de sugestões de novos padrões baseados em boas práticas de programação. O resultado destas reuniões irá compor um documento que sintetiza todas as melhores práticas e os melhores padrões de codificação de software para os sistemas do LCC.

[13] Ver Anexo A

A grande vantagem de se usar os sistemas já existentes no LCC será a sistematização de padrões que já existem nos diversos sistemas Java. Mesmo que todo este sistema não tenha sido construído seguindo tais padrões, grande parte dele já estará de acordo com o documento, facilitando o seu entendimento para futuros envolvidos em manutenção e extensão do software.

Desta forma, temos que os seguintes passos serão seguidos:



4.1. Proposta Do Documento de Padronização de Codificação

Tendo em vista a existência de diversos sistemas no LCC, com a grande maioria criado antes da existência da preocupação de seguir um padrão de codificação, é importante que o documento que será criado contemple, mesmo que não em sua totalidade, estes sistemas. Dessa forma, padrões identificados ao longo deles serão colocados em pauta para que seja discutida a adoção dos mesmos ou um padrão que se aproxime dos já existentes.

Com o auxílio do documento proposto pela *Sun Microsystems* (SUN MICROSYSTEMS, 1997), o documento seguirá a seguinte organização, abordando os tópicos:

- Objetivo e Definições gerais

  Objetivo e resumo do documento, definição do idioma a ser utilizado em nomes e comentários, definição de sistemas de controle de versão e outros serviços utilizados durante o desenvolvimento.

  As definições feitas neste item possuem grande importância, com destaque às definições de controle de versão de demais serviços, como os de bancos de dados e de LDAP. Muitas aplicações do LCC utilizam bases de dados e outros sistemas que são mantidos em recursos distribuídos, sendo necessário o conhecimento do endereçamento para acesso aos mesmos.

- Organização de pastas e arquivos

  Definição de hierarquias de pastas, divisões de recursos do sistema, arquivos de configuração, bem como a organização e localização de arquivos de código fonte (seus nomes são definidos junto dos nomes de classes, no item 4).

  Não apenas uma boa organização do código fonte é necessária. Uma boa organização de pastas e de arquivos de código possibilita um acesso rápido e intuitivo quando é necessário buscar por informações sobre uma mesma solução, mas com contextos e focos diferentes.

- Organização do código fonte

  Organiza um arquivo de código fonte, definindo se e onde serão exibidos comentários sobre a classe, como será a endentação e a separação visual dos elementos. Define também a utilização de espaços em branco e a utilização de blocos de tratamento de erros

  Este item abordará uma questão estética, definindo visualmente como será o código. Assim, um desenvolvedor será capaz de identificar claramente como o

código está dividido e compreender mais facilmente o processamento que está sendo realizado.

Conforme BOSWELL e FOUCHER dizem, “é mais fácil trabalhar com código que é esteticamente agradável. [...] A maior parte do tempo programando é gasta olhando para o código"[14] (tradução nossa) (p. 35, 2012).

BOSWELL e FOUCHER (2012) recomendam que quebras de linha sejam consistentes e compactas, que não sejam expressas irregularidades na apresentação estética do código, que exista um alinhamento vertical no código, uma endentação bem definida, que declarações e operações semelhantes sejam agrupadas em blocos e divididas em parágrafos, como um texto normal.

Já o tratamento de exceções utilizando blocos de *try/catch*[15] deve ser realizado buscando destaque no código, de forma que fique claro dos possíveis erros que podem ocorrer naquele trecho do código.

Com relação a classes muito grandes, a SUN MICROSYSTEMS recomenda que classes com mais do que duas mil linhas de código sejam evitados(SUN MICROSYSTEMS, 1997). Recomenda-se que classes tão extensas sejam repensadas de maneira que possam ser divididas em duas ou mais classes menores, de forma que cada classe seja o mais específica possível buscando dividir problemas grandes em sub-problemas menores tratados por cada classe.

Seguindo princípios estéticos como estes, o desenvolvedor saberá facilmente onde encontrar comentários sobre o propósito e funcionamento dos elementos presentes em uma classe.

[14] “It’s easier to work with code that’s aesthetically pleasing. […] most of our time programming is spent looking at code”

[15] “tentar/capturar”, traução nossa

- Nomeclatura de classes, variáveis e métodos

  Define como devem ser os nomes de classes, variáveis e métodos. Padrões de nomes e atributos, nomes de variáveis auxiliares e de retorno.

  O mais importante ao se tratar de nomes na codificação é fazer com que eles apresentem informação suficiente para identificar o seu propósito. Classes, variáveis e métodos com nomes confusos e muito simplistas podem gerar o seu uso errôneo, gerando bugs e dificultando o entendimento do código por desenvolvedores distindos trabalhando em um mesmo sistema ou realizando manutenção no código.

  Existem diferentes padrões de nomeação de elementos em diversas linguagens. Um destes padrões é o *Camel Case*[16] onde as primeiras letras de cada palavra são escritas em maiúsculo (ex: ExemploDeClasse).

- Declarações e instruções

  Como devem ser feitas as declarações de variáveis, atribuições de valores e as instruções.

  É importante que o código seja simples de entender. Mesmo que declarações de variáveis e atribuições de valores sejam operações muito simples de se visualizar, é importante que não sejam confundidas em meio a operações mais complexas.

- Organização e estrutura de classes

  Onde serão definidos variáveis de classe (estáticas), atributos e métodos e padrões de acesso a estes elementos.

[16] "Caixa Camelo", tradução nossa

Uma boa estrutura da classe garante um bom entendimento do propósito da mesma. Uma classe sem propósito bem definido pode ser descartada e suas funcionalidades adequadas à uma outra classe mais bem estruturada. Este item regulamenta como serão utilizadas as variáveis de classe e como os atributos e métodos de um objeto serão declarados, de forma que não se tornem confusos e nem estabeleçam classes e objetos sem propósitos bem definidos.

Segundo MARTIN (2009) e SUN MICROSYSTEMS (1997), uma classe deve ser organizada da seguinte forma: Iniciar com a lista de variáveis – constantes públicas, variáveis estáticas privadas, variáveis de instância privadas. Prosseguir então com os métodos – funções públicas após a lista de variáveis. Quando a classe não apresentar um dos elementos, eles podem ser omitidos. Funções privadas que sejam utilizadas por funções públicas são definidas logo após elas.

- Expressões lógicas e aritméticas

  Define espaçamento e posicionamento de testes lógicos, expressões aritméticas e binárias.

  Testes lógicos e expressões devem ser bem entendidos, pois um erro simples de entendimento pode causar um teste que deveria apresentar valor verdadeiro apresentar um valor falso, causando um desvio do fluxo de processamento de difícil localização na manutenção do código.

  Quando em uma estrutura de repetição, uma expressão lógica errada pode causar um *loop*[17] infinito, problema de difícil manutenção em alguns casos, causando um funcionamento anormal no sistema. Um problema destes, quando não evidenciado na execução da aplicação, pode causar alocação desnecessária de recursos até o travamento do sistema.

[17] "laço", tradução nossa

Todas as expressões podem ser expressas de uma maneira mais simples, tornando consequentemente muito mais simples o entendimento de uma condição que alterne o fluxo ou o significado de uma expressão aritmética. Deve-se buscar apresentar as expressões da maneira mais natural possível.Um teste expresso como "20 <= idade" não é tão claro quanto "idade > 20", Por exemplo.

- Estruturas de controle e de repetição

  Define utilização de testes lógicos (*if* e *else*, *switch*) e estruturas de repetição (*for*, *while*, *do...while*).

  Testes lógicos devem mostrar como controlam o fluxo de processamento de forma clara, evitando problemas maiores no momento de manutenção de código e possibilitando alteração deste fluxo de forma simples e compreensiva.

  O uso de estruturas de repetição deve seguir sempre o mesmo padrão ao longo do código, não utilizando diferentes nomes auxiliares e formatação, já que possuem o mesmo propósito ao longo do código, respeitando as caracteristicas de cada um.

  BOSWELL e FOUCHER apresentam uma ideia chave relacionada à expressões e a estruturas de controle e repetição: "Faça todas as suas condições, laços, e outras mudanças no fluxo de execução tão 'naturais' quanto possível – escritas de uma maneira que não faça o leitor parar e reler seu código"[18] (2012, p. 70).

[18] "*Make all your conditionals, loops, and other changes to control flow as 'natural' as possible – written in a way that doesn't make the reader stop and reread your code*", tradução nossa.

- Comentários

  Definição de como os comentários devem ser utilizados. Locais apropriados para inseri-los, e o que é necessário ser apresentado em um comentário.

  "O propósito do comentário é ajudar o leitor [do código] a saber tanto quanto o escritor [do código] sabia"[19] (BOSWELL, p. 46, 2012). Esta definição é bem clara quanto à melhor maneira de utilizar comentários. Ao escrever um código, o desenvolvedor possui um conhecimento mais profundo acerca do sistema como um todo, além de ter um conhecimento abrangente da solução que está implementando. Quando este código é transferido para outro desenvolvedor, seja da equipe ou para a realização de alguma manutenção, ele não tem conhecimento amplo da solução proposta pelo desenvolvedor original, demandando um tempo maior para que o código seja entendido e modificado ou continuado. É possível até que um código mais complicado acabe por ser descartado e o desenvolvedor que realiza a manutenção opte por reescrever a solução, de acordo com o conhecimento e o contexto que ele possui no momento.

  É importante que os comentários possuam uma estrutura simples e bem definida, evitando comentários simplórios e óbvios, servindo como uma boa fonte de informação para o leitor do código. Os locais em que os comentários são exibidos devem seguir uma lógica natural para que sejam facilmente encontrados e não ocupem muito da tela, atrapalhando a leitura do código. Em outras palavras, não se recomenda o uso de comentários longos, espalhados ao longo do código sem um cuidado maior, e explicando coisas óbvias como "isto é um teste".

[19] "*The purpose of commenting is to help the reader know as much as the writer did*", tradução nossa

- Padrões de Projeto

  Como visto no item 3.4 deste trabalho, existem vários padrões de projeto, para diferentes soluções. Este item irá definir padrões de projeto mais comumente utilizados no LCC, além de dar sugestões de como padrões de projeto que possam ser utilizados para uma melhor codificação.

  A utilização de padrões garante que muito tempo seja poupado ao se deparar com problemas recorrentes, fazendo com que o desenvolvedor se preocupe apenas com o âmago da solução, auxiliando na criação de uma solução bem construída e bem estruturada. O conhecimento prévio dos padrões mais adotados no LCC garante que o desenvolvedor que realiza uma manutenção ou continuidade no desenvolvimento da aplicação não precise gastar seu tempo entendendo grandes blocos de código pois já conhece previamente o funcionamento do padrão e qual a finalidade de sua utilização. Acompanhado de um comentário bem elaborado, centenas de linhas de código podem ser compreendidas em poucos minutos.

É importante reafirmar que cada item será colocado em pauta em reuniões dos envolvidos no desenvolvimento da versão do documento, definindo então a melhor abordagem de padrões apresentados às características já existentes no código dos sistemas do LCC.

5. Orçamento físico-financeiro

	Agosto	Setembro	Outubro	Novembro	Dezembro
Lucas Gomes (horas/semana)	3	4	3	4	5
Renato Veneroso (horas/semana)	2	2	2	2	2
Bruno Oliveira (horas/semana)	2	2	2	2	2
Leonardo Freitas (horas/semana)	2	2	2	2	2
Soraya Jardim (horas/mês)	2	2	2	2	2

Tabela 2 Orçamento físico-financeiro

As horas previstas de trabalho pela equipe não irão dimensionar custos adicionais à UFMG.

Sugere-se que reuniões entre os analistas sejam realizadas semanalmente, com duração máxima de trinta minutos. Ao final de cada mês serão realizadas reuniões de acompanhamento com a presença da gerente de desenvolvimento Soraya Jardim.

6. Equipe

A equipe envolvida na elaboração do documento de padronização de codificação Java no LCC será composta pelos analistas:

- Soraya Jardim (Gerente de desenvolvimento)
- Bruno Oliveira de Carvalho
- Leonardo Freitas
- Lucas Soares Gomes
- Renato Veneroso

A alocação de horas por semana para conforme a seção 5 (Cronograma) será administrada pelo próprio analista. A gerente de desenvolvimento irá acompanhar ao final de cada mês o progresso do trabalho em reuniões com esta finalidade.

Cronograma

Para a elaboração do documento de padronização de codificação Java do LCC, o seguinte cronograma será seguido:

	Agosto	Setembro	Outubro	Novembro	Dezembro
Estudo da codificação dos sistemas LCC	X	X			
Levantamento de sugestões		X	X		
Elaboração do documento – Prévia				X	
Elaboração do documento final					X
Reuniões de Acompanhamento	X	X	X	X	X

Tabela 3 Cronograma

Ao final de cada mês serão realizadas reuniões de checkpoint, podendo ser realizadas em conjunto com as reuniões voltadas à elaboração do documento. Estas reuniões irão tratar do acompanhamento do projeto elaboração do documento de padronização de codificação.

7. Resultados Esperados

Com a criação do documento de padronização de codificação Java para o LCC, espera-se que o processo de adaptação de novos servidores, profissionais terceirizados e estagiários se dê mais rapidamente e com maior segurança no envolvimento com as atividades de desenvolvimento exercidas no LCC. Também espera-se uma melhora na comunicação entre equipes quando é necessário o desenvolvimento de interfaces entre sistemas diferentes, tanto internamente no LCC quanto externamente, com sistemas que não pertencem ao LCC.

Os sistemas que serão desenvolvidos a partir da existência deste documento contará com padrões bem definidos e de fácil verificação, garantindo que qualquer manutenção realizada no sistema no futuro seja realizada de maneira mais ágil e mais simples. A vida útil dos sistemas do LCC será também extendida, já que o entendimento de seu funcionamento será mais fácil, permitindo que os sistemas apresentem uma maior adaptabilidade à novos requisitos informados pelos parceiros do LCC que os utilizam.

A definição de padrões de projeto também ajudará no processo de desenvolvimento adotado no LCC, permitindo que se ganhe tempo ao se deparar com problemas recorrentes. Uma vez que a utilização de padrões de projeto se torne mais habitual, os desenvolvedores conhecerão melhor os detalhes de utilização e de implementação destes padrões, tornando cada vez mais natural a utilização dos mesmos.

Estas práticas também serão muito aproveitadas quando o LCC contar com bolsistas para auxiliar nas atividades do setor. Visando a missão da ufmg de “gerar e difundir conhecimentos científicos, tecnológicos e culturais, destacando-se como Instituição de referência na formação de indivíduos críticos e éticos, dotados de sólida base científica e humanística e comprometidos com intervenções transformadoras na sociedade, visando o desenvolvimento econômico, a diminuição de desigualdades sociais e a redução das assimetrias regionais, bem como o desenvolvimento sustentável” (UNIVERSIDADE FEDERAL DE MINAS GERAIS, p. 6, 2013), o LCC proporcionará um ambiente ainda mais rico de informação e formação de qualidade ao bolsista, garantindo o uso de tecnologia atual, desenvolvendo habilidades de comunicação entre equipes, desenvoltura na compreensão de

código alheio e na busca da criação de sistemas que sejam ainda mantidos por vários anos por outros desenvolvedores.

Referências Bibliográficas

CAMARGO, V. P. D. C., 2011. Portugol, Elementos básicos de um programa Java. Disponível em http://www.scribd.com/doc/56646133/12/Elementos-basicos-de-um-programa-Java-Elementos-basicos-de-um-programa-Java, acessado em 30/04/2013,

LUDWIG, I. P., ANZANELLO, M. J., 2011. Minimização dos tempos de atraso na programação de tarefas em uma empresa de desenvolvimento de softwares. UFRS. Disponível em http://www.lume.ufrgs.br/handle/10183/33164, acessado em 30/04/2013,

LIENTZ, B. P., SWANSON E. B., TOMPKINS, G. E., 1978. Characteristics of application software maintenance. University of California. Los Angeles, USA. Communication of ACM, vol. 21, no. 6, pp. 466-471. Disponível em http://dl.acm.org/citation.cfm?id=359522.

LABORATÓRIO DE COMPUTAÇÃO CIENTÍFICA. Website oficial. Minas Gerais, Brasil. Disponível em http://www.lcc.ufmg.br, acessado em 30/04/2013.

RAMOS, R. A., 2009. Manutenção de Software. UNIVASF. Disponível em http://www.univasf.edu.br/~ricardo.aramos/disciplinas/ESI2009_2/Aula021_Manutencao.pdf, acessado em 30/04/2013,

DEITEL, H. M., 2006. Java™ How to Program. 7th edition, Prentice Hall. New Jersey, USA.

SUN MICROSYSTEMS, 1997. Java Code Conventions. USA. Disponível em http://www.oracle.com/technetwork/java/codeconventions-150003.pdf.

SOMMERVILLE, I., 2004. Engenharia de Software, 6ª edição. Pearson Education do Brasil. São Paulo, Brasil.

KOSKINEN, J., SALMINEN, A., PAAKKI, J., 2004. Hypertext support for the information needs of software maintainers. Department of Computer Science and Information Systems. University of Jyväskylä, Finlândia. Disponível em http://dl.acm.org/citation.cfm?id=1046822.

GAMMA, E., HELM, R., JOHNSON, R., VLISSIDES, J., 2006. Padrões de projetos: Soluções reutilizáveis de software orientado a objetos. Editora Bookman. São Paulo. Brasil.

SOARES, M. S., 2010. Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software. Unipac. Conselheiro Lafaiete, Minas Gerais. Disponível em http://xps-project.googlecode.com/svn/!svn/bc/6/trunk/outros/Met._Ageis.pdf.

BOSWELL, D., FOUCHER, T., 2012. The Art of Readable Code. O'Reilly Media. California, USA.

MARTIN, C. M., 2009. Clean Code – A Handbook of Agile Software Craftmanship. Pearson Education, Inc. Massachusetts, USA.

UNIVERSIDADE FEDERAL DE MINAS GERAIS, 2013. Plano de Desenvolvimento Institucional, website oficial, acessado em 20/06/2013, disponível em https://www.ufmg.br/conheca/dai-pdi.shtml.

Anexos

Anexo A – Documento de normalização de bancos de dados no LCC

Universidade Federal de Minas Gerais
LCC-CENAPAD

Padrão de Atributos de Banco de Dados

24/07/2008

Dados Gerais

Identificação do documento:	Padrão de Atributos – V1.0	
Histórico de Revisão:	**Atividade:**	**Responsável:**
	1.0 – Criação do Documento (08/11/2007)	Soraya Ferreira
	1.1 – Comentário sobre letras maiúsculas nos nomes de tabelas e atributos (09/05/2008)	Soraya Ferreira
	2.0 – Acréscimo de padrão para nome de índices (24/07/2008)	Soraya Ferreira

Sumário

Objetivo

Esse documento apresenta um padrão simples e prático para a formação de nomes de atributos e tabelas de banco de dados baseado na forma de trabalho já seguida pelo CECOM. O objetivo é criar no LCC uma uniformidade no processo de criação desses nomes, além de facilitar, pelas próprias características do padrão apresentado, o acesso e entendimento da estrutura das bases de dados.

Descrição

1. Tabelas

Usar o padrão: PREFIXO_NOME_TABELA, em letra maiúscula, onde:

- PREFIXO: Sigla de 3 letras que identifique a tabela. Deve ser único em uma mesma base de dados.
  - Ex: ALU, TPU, MAT, TUR
- NOME_TABELA: utilizar o nome completo, evitando abreviações. Para melhor leitura, acrescentar o caracter "_" (*underline*) entre palavras.
  - Ex: ALUNO, TIPO_USUARIO, MATRICULA, TURMA

Exemplos de nomes completos de tabelas:
ALU_ALUNO
TPU_TIPO_USUARIO
MAT_MATRICULA
TUR_TURMA

2. Atributos

Usar o padrão: PREFIXO_CAMPO_SUFIXO, em letra maiúscula, onde:

- PREFIXO: Sigla de 3 letras que foi usada para identificar a tabela à qual pertence o atributo.
  - Ex: ALU, TPU, MAT, TUR
- CAMPO: Efetivamente esse é o nome do atributo, sempre evitando abreviações.
  - Ex: NOME, CODIGO, RUA
- SUFIXO: Tipo do atributo, onde:
  - CHR: atributos do tipo string
  - NUM: atributos do tipo numérico
  - DAT: atributos do tipo data

Dessa forma, somente a partir do nome do atributo, já se consegue a informação de a qual tabela e a qual tipo ele pertence.

Ex.:

ALU_CODIGO_NUM
PRO_NOME_CHR
ALU_CADASTRO_DAT
MAT_CODIGO_NUM
TUR_CODIGO_CHR

3. Chaves Estrangeiras

As chaves estrangeiras permanecem com o mesmo nome nas tabelas onde são referenciadas, herdando seu nome da tabela original.
Assim, se o código do Aluno (ALU_CODIGO_NUM) é uma chave estrangeira na tabela de MATRICULA, o nome do atributo não se altera quando criado nessa tabela. Dessa forma, ao listar os atributos de uma tabela torna-se fácil observar seus relacionamentos com outras tabelas.

Ex: Atributos da tabela MATRICULA

MAT_CODIGO_NUM
ALU_CODIGO_NUM (chave estrangeira da tabela ALUNO)
TUR_CODIGO_CHR (chave estrangeira da tabela TURMA)

4. Índices

Usar o padrão PREFIXOINDICE_PREFIXOTABELA_NOMEINDICE, letra maiúscula, onde:

- PREFIXOINDICE: Sigla de 2 letras que identifica que o nome refere-se a um índice: IX
- PREFIXOTABELA: Prefixo da tabela à qual o índice pertence:
  - Ex: ALU, MAT, TUR
- NOMEINDICE: Nome que identifica o índice. Devido ao framework utilizado para desenvolvimento Java, as chaves primárias das tabelas são sempre criadas como contadores. Assim, nos casos em que houver necessidade de se criar um índice único para realizar a função que seria da chave primária, usar a palavra PRINCIPAL como nome do índice para identificá-lo.

Exemplos de nomes de índice:
IX_ALU_PRINCIPAL
IX_ALU_PERIODO_ADMISSAO

Termos e Definições

TERMO	DEFINIÇÃO
ATI	Assessoria de Tecnologia da Informação da UFMG
CECOM	Centro de Computação, órgão ligado à ATI

LCC	Laboratório de Computação Científica, órgão ligado à ATI

Documentos de Referência

NOME DO DOCUMENTO	REFERÊNCIA

Anexo B – Questionamentos sobre a codificação Java realizada no LCC feitos ao analista de sistemas Renato Veneroso.

P: Quais sistemas vocês trabalham que utiliza Java?

- Portal minhaUFMG,

- Geração Automática de NIPs,

- Impressão de NIPs pela Web,

- Manutenção da base de dados do repositório LDAP,

- Esfera Semântica (em desenvolvimento)

P: Desde quando trabalham com este sistema? O sistema foi herdado?

- Portal minhaUFMG: desde 2008. Esse sistema foi desenvolvido por equipe da própria UFMG com base no framework de Portais Acadêmicos chamado uPortal. Sistema herdado,

- Geração Automática de NIPs: desde 2009. Esse sistema foi desenvolvido por equipe da própria UFMG,

- Impressão de NIPs pela Web: desde 2008. Esse sistema foi desenvolvido por equipe da própria UFMG. Sistema herdado,

- Manutenção da base de dados do repositório LDPA: desde 2008. Esse sistema foi desenvolvido por equipe da própria UFMG. Sistema herdado,

- Esfera Semântica: está em desenvolvimento desde maio/2013 por equipe da própria UFMG.

P: Há alguma convenção para codificação? Há algum padrão na codificação dos sistemas herdados?

Nos sistemas mais recentes tem-se adotado um padrão de se agrupar as classes do projeto de acordo com suas funcionalidades. Para esse agrupamento utiliza-se como referência principalmente o modelo MVC - Model / View / Controller

de forma a se dividir a responsabilidade de cada uma dessas camadas MVC entre as classes do projeto. Quanto aos sistemas herdados, o único que realmente foi desenvolvido com base em um padrão é o Portal minhaUFMG (muito em função de ter sido construído em cima de um framework). Os demais não possuem padronização em sua codificação. O que tenta-se fazer é, à medida que tais sistemas são evoluídos, tentar enquadrá-los aos padrões atualmente adotados. Isso, entretanto, nem sempre é possível pois em muitos casos implicaria na reescrita de grande parte do código o que, além de demandar tempo, pode introduzir erros nas aplicações.

P: Já houve algum caso de problemas com código antigo de difícil entendimento?

Sim. O Portal minhaUFMG é o caso mais clássico. Essa dificuldade no entendimento do código, entretanto, não é decorrente de código mal escrito. Ela ocorre pela complexidade do framework uPortal e da própria aplicação. Além disso, a autenticação no Portal anteriormente era feita através da validação do usuário em bases de documentos Lotus Notes. Para se alcançar esse objetivo foi necessário utilizar uma API externa desenvolvida e disponibilizada pela IBM e também escrever código de comunicação em mais baixo nível entre os ambientes Java e Lotus Notes.