UNIVERSIDADE FEEVALE

SANDRO MADRUGA SILVEIRA

# CRIAÇÃO DE UM FRAMEWORK MODULAR E EXPANSÍVEL PARA GERAÇÃO DE CÓDIGO FONTE

(Título Provisório)

Anteprojeto de Trabalho de Conclusão

Novo Hamburgo
2011

SANDRO MADRUGA SILVEIRA

# CRIAÇÃO DE UM FRAMEWORK MODULAR E EXPANSÍVEL PARA GERAÇÃO DE CÓDIGO FONTE

(Título Provisório)

Anteprojeto de Trabalho de Conclusão de Curso, apresentado como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação pela Universidade Feevale

Orientador: Edvar Bergmann Araújo

Novo Hamburgo
2011

# RESUMO

A utilização de ferramentas produtivas para o desenvolvimento de software garante o incremento de qualidade com redução de custos. Uma das formas de reduzir tarefas repetitivas durante a codificação é através da reutilização do conhecimento passado em novos projetos. Isto pode ser aplicado a partir de técnicas de geração automatizada de código fonte ao invés da reescrita de artefatos muito parecidos. Embora existam muitas ferramentas que se propõem a gerar pelo menos alguns trechos de código fonte, é bastante complexo atender situações específicas de cada projeto. Sendo assim, este trabalho tem como objetivo propor uma ferramenta de geração de código fonte flexível e que seja de aplicação simples em projetos distintos, onde a partir de algumas especificações básicas de entradas seja possível gerar vários tipos de artefatos de software. Também prevê que novas especificações possam ser implementadas ampliando as possibilidades de uso da ferramenta.

Palavras-chave: Geração de Código; Metadados; Padrões de Projeto; MDA (*Model Driven Architecture*); DSL (*Domain-Specific Language*).

# SUMÁRIO

# MOTIVAÇÃO

A crescente demanda por soluções de software nas mais variadas áreas do conhecimento humano ampliou sobremaneira os requisitos, metodologias, técnicas e conhecimentos que os profissionais de engenharia de software precisam dominar. Toda essa exigência objetiva enfrentar de maneira eficaz e com alta qualidade aos vários desafios que se apresentam em todo o ciclo de vida de desenvolvimento. Há muito tempo entende-se que é necessário usar um amplo conjunto de ferramentas para o processo de desenvolvimento. Segundo Pressman (1995, p. 945): "Os engenheiros de software agora reconhecem que precisam de um número maior e mais variado de ferramentas (apenas ferramentas manuais não atendem às exigências dos modernos sistemas baseados em computador)".

Muitas abordagens são utilizadas no processo de desenvolvimento de software visando obter maior produtividade e qualidade. Tais abordagens são aplicadas desde a obtenção preliminar de requisitos até a manutenção corretiva e evolutiva de um software. No entanto, por mais maturidade que o processo de produção de software possa atingir, sempre existem projetos onde alguma parte será construída de forma artesanal. Comparado aos modelos industriais onde rígidos padrões e altíssimos níveis de automatização são usados, muitos problemas no desenvolvimento de software poderiam ser evitados, reduzindo custo e tempo necessários para a obtenção dos resultados.

Analistas de sistemas e programadores, de forma recorrente, criam soluções similares, muitas vezes aplicando conceitos baseados em algum padrão pré-existente e em outras ocasiões inventam novas abordagens para solucionar algo que já foi feito de outra forma.

A reutilização de conhecimento prévio de projetos bem sucedidos, criando-se modelos aplicáveis a novos problemas, é a base para alguns paradigmas que visam reaproveitar esforço passado em novas soluções. Esta abordagem abrange o uso integrado de metodologias, linguagens, programas, bibliotecas e componentes muitas vezes bastante heterogêneos para a obtenção de um resultado final que na maioria das vezes não poderia ser atingido não fosse desta forma. No entanto, a interconexão total de ferramentas disponíveis para o desenvolvimento de software ainda não é uma realidade, exigindo muitas vezes a utilização de diversas soluções independentes com um nível de acoplamento normalmente baixo.

O isolamento da complexidade e a reusabilidade permitem ocultar detalhes da implementação, reduzir a propensão a erros e facilitar os testes e a manutenção de um software. A criação de produtos com alto padrão de qualidade à um custo menor, demanda a produção em massa, ou pelo menos a produção múltipla de artefatos. Conclui-se que desta forma bons projetos podem ser facilmente modificados e reutilizados (BRAUDE, 2005).

Percebe-se hoje no mercado de desenvolvimento de software que ocorrem mudanças nos requisitos dos sistemas durante praticamente todas as fases do processo. Inclusive as metodologias ágeis de desenvolvimento de software apontam que arquitetos e programadores de sistemas precisam estar com a mentalidade aberta a acolher constantemente as mudanças de requisitos apresentadas pelos clientes. Segundo Beck (2004), XP (*EXtreme Programming*) convida o cliente para ser uma parte integrante do time, com isto a especificação do sistema é continuamente refinada durante o desenvolvimento. O resultado é que os desenvolvedores precisam estar aptos à rapidamente responder as mudanças solicitadas.

Independente dos projetos seguirem metodologias ágeis ou metodologias mais tradicionais considera-se produtivo o uso de ferramentas de prototipação rápida e de base para o desenvolvimento de novos artefatos. Sobre a utilização de geradores de código para o uso em prototipação, McConnel (2005, p. 737) diz que

> Os geradores de código também são úteis para fazer protótipos de código de produção. Usando um gerador de código você pode montar em poucas horas um protótipo que demonstre os principais aspectos de uma interface com o usuário ou pode experimentar várias estratégias de projeto. Talvez demore semanas para codificar a mesma funcionalidade manualmente.

Para cada projeto de software, pode-se usar um conjunto de métodos e técnicas de acordo com a tecnologia utilizada e o problema a ser resolvido. Em alguns projetos parte do trabalho manual, maçante e sujeito à erros, pode ser implementado por softwares inteligentes. Em sua tese de doutorado, Czarnecki (1998) explica que a programação generativa centra-se na concepção e implementação de software reutilizável para geração de sistemas ao invés de escrevê-los do zero. Portanto o âmbito desta metodologia é a construção de famílias de sistemas.

Estudando diversas ferramentas de desenvolvimento de software encontram-se muitos recursos que visam reaproveitar um trabalho prévio para a construção de uma próxima etapa. Este reaproveitamento permite o aumento de produtividade facilitando tarefas repetitivas, seja fazendo parte do trabalho ou integrando-se com outras aplicações. Algumas

destas ferramentas permitem algum tipo de personalização para que os desenvolvedores possam aplicar de acordo com a necessidade de seus próprios projetos. Por outro lado, muitas outras não permitem tal nível de personalização. Disponibilizam apenas formas rudimentares de auxílio que acabam exigindo uma intervenção posterior com ajustes no código fonte que podem interferir na qualidade final do que foi produzido.

Também ocorre de determinados recursos mais completos de auxílio na escrita de código fonte estejam disponíveis apenas em um IDE (*Integrated Development Environment*) de uma linguagem específica. Isto facilita determinadas tarefas em algumas linguagens, mas não torna-se aplicável em outras, mesmo que para tarefas similares.

Em trabalhos acadêmicos relacionados percebe-se que muitas propostas de ferramentas de geração de código fonte focam apenas para a obtenção de objetivos muito específicos. Em relação a ferramentas gratuitas ou comerciais também muitas delas tem uma única origem e um único destino, limitando a transformação de artefatos de software.

Sendo assim, a proposta deste trabalho é a construção de um *framework* ou ferramenta com padrão aberto que possa ser altamente acoplável a novas especificações de entrada. Este conceito parte de um núcleo bem definido que use conceitos de geração de código fonte, de forma que possa haver uma dinâmica entre definições de entradas e saídas. Busca-se a resolução de problemas genéricos de geração de código, através do reaproveitamento de algumas definições de entrada comuns pré-existentes, tais como uma especificação padrão de XML. Pretende-se também que a ferramenta possa ser utilizada em vários projetos, independente de linguagem de programação, sistema operacional, metodologia de desenvolvimento ou ferramentas utilizadas, reduzindo a incompatibilidade entre ferramentas que acabam prejudicando a produtividade das equipes de desenvolvimento de software.

Um ponto fundamental da arquitetura da solução proposta é que o núcleo de geração de código seja desvinculado das especificações das entradas. Pretende-se resolver esta questão através da definição de uma interface padrão que permitirá que novas especificações sejam escritas sem que seja necessário lidar com detalhes sobre o tratamento de *templates* (modelos de código), interação do usuário ou formatação de saída. O modelo proposto diferencia-se de outras ferramentas básicas que não fazem o *merge* (combinação) de entradas com *templates*, bem como de soluções mais avançadas que requerem a implementação do relacionamento entre os componentes em um único módulo, dificultando ou impedindo a expansão da ferramenta.

A solução proposta será estruturada na forma de módulos, que são: especificações, interface e núcleo. A ferramenta permitirá acesso a suas funcionalidades por linha de comando, interface gráfica e *plugins*, tendo como resultado final trechos de código fonte gerado de acordo com o especificado nos *templates*. A figura 1 apresenta esta estrutura.

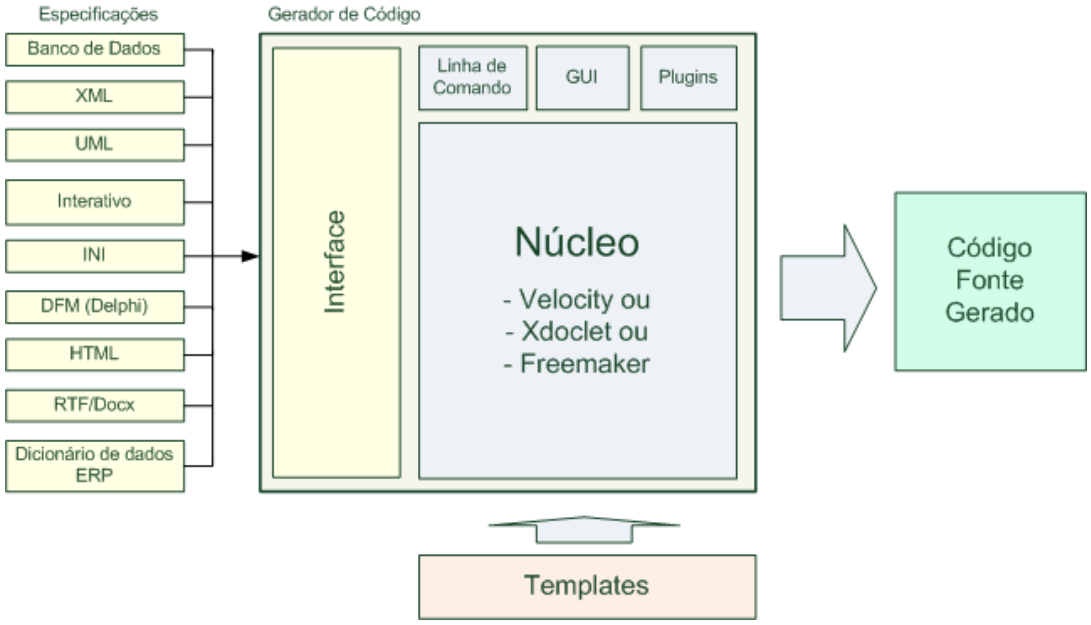


**Figura 1 - Arquitetura da ferramenta de geração de código**
Fonte: Do próprio autor

A fim de fundamentar o projeto e implementação da ferramenta proposta, serão explorados conceitos de MDA (*Model Driven Architecture*), MDD (*Model Driven Development*), Metadados, JDBC (*Java DataBase Connectivity*), *Reflection*, DSL (*Domain-Specific Language*), *Design Patterns*, UML (*Unified Modeling Language*), XML (*Extensible Markup Language*), *Plugins*, DAO (*Data Access Objects*), DTO (*Data Transfer Objects*), CASE (Computer-Aided Software Engineering), Ontologias, SQL (*Structured Query Language*), HTML (*HyperText Markup Language*), *Generative Programing*, *Software Factories*, *Anotations* e OOP (*Object-Oriented Programming*), entre outros. Muitos destes conceitos serão usados diretamente e outros como comparativos e de referência para modelar e implementar as soluções que serão propostas.

# OBJETIVOS

## Objetivo geral

O objetivo do trabalho é modelar e desenvolver uma solução de geração de código fonte baseada em um padrão que permita que com algumas especificações básicas possa ser possível automatizar a criação de diversos tipos de artefatos de software. Objetiva-se propor uma estrutura flexível e padronizada que permita a implementação de novas especificações de entrada. Cada nova especificação de entrada permitirá o aumento das possibilidades de uso da ferramenta.

## Objetivos específicos

- Explorar conceitos importantes para a construção da ferramenta;
- Definir padrões de projeto que podem ser usados;
- Analisar *templates engines* existentes;
- Avaliar ferramentas similares, suas vantagens e limitações;
- Especificar a arquitetura da ferramenta proposta;
- Projetar e implementar um núcleo de geração de código;
- Desenvolver utilitário para linha de comando e interface gráfica;
- Avaliar a possibilidade de integrar com as IDEs Netbeans ou Eclipse através de *Plugins*;
- Validar a ferramenta proposta em um estudo de caso;

# METODOLOGIA

A construção da solução proposta começará pela coleta de dados, que será efetuada através de pesquisas bibliográficas e análise de ferramentas similares. A base será a pesquisa bibliográfica para conhecimento de conceitos relacionados e embasamento teórico. Serão pesquisados livros de engenharia de software, padrões de projeto e linguagens de programação, tais como Sebesta (2000), McConnel (2005), Gamma et al (2000) e Braude (2005). Mais especificamente sobre o assunto de geração de código existem algumas obras de autores que são referenciados em muitos trabalhos científicos, tais como Czarnecki e Eisenecker (2000), Fowler (2010) e Herrington (2003). Também são encontrados alguns trabalhos de mestrado e doutorado na área que podem servir como fonte de informação, por exemplo a tese de doutorado de Czarnecki (1998).

Padrões de projetos serão estudados com a finalidade de verificar quais as características importantes para a construção de uma ferramenta desta categoria devem ser aplicados. O trabalho de avaliação de componentes será efetuado a partir do estudo e experimentação de uso de componentes conhecidos como *template engines* (motores de *template*), que são componentes altamente conhecidos e utilizados: Apache Velocity, Xdoclet e Freemaker, entre outros. Tais componentes serão pesquisados, explorados e comparados. Também serão analisados softwares relacionados que usam os conceitos apresentados no trabalho, procurando descobrir quais que podem ser referenciados como projetos bem sucedidos no que diz respeito ao assunto de geração de código fonte. Todas as avaliações serão feitas com o intuiro de apontar pontos fortes e fracos a fim de fundamentar a ferramenta a ser desenvolvida.

Após a coleta de dados partir-se-á para a proposição e implementação da ferramenta. O desenvolvimento será com a linguagem de programação Java, devido à grande quantidade de bibliografia e recursos encontrados, além de alguns dos *template engines* estarem disponíveis para esta plataforma. O autor do trabalho também tem interesse pessoal e profissional no estudo e aprimoramento desta tecnologia. Mesmo com a opção pelo desenvolvimento em Java não pretende-se ficar restrito ao âmbito desta linguagem, visto que haverão exemplos contemplando a integração com outras linguagens distintas.

A implementação será baseada no núcleo e na interface da ferramenta. Também serão projetadas e elaboradas algumas especificações de entrada com as quais seja possível utilizar o gerador de código, por exemplo, a partir de arquivos XML e de metadados de banco

de dados. Também será estudada a possibilidade da ferramenta ser integrada em IDEs através de *plugins* ou outras alternativas que permitam a utilização de uma forma ágil e produtiva.

A validação da ferramenta proposta será realizada através da geração de código fonte para o ERP SIGER, da empresa Rech Informática LTDA, sendo utilizado como exemplo de aplicação prática. Para tanto deverá ser implementada uma especificação de entrada com base no dicionário de dados deste sistema. A meta é obter código fonte a partir das estruturas relacionadas, comparando seus resultados com as metodologias utilizadas atualmente.

# CRONOGRAMA

## Trabalho de Conclusão I

| Etapa | Meses | | | |
|---|---|---|---|---|
| | Ago | Set | Out | Nov |
| Levantamento bibliográfico | | | | |
| Estudo de padrões de projeto aplicáveis | | | | |
| Avaliação de componentes | | | | |
| Análise de soluções similares | | | | |
| Projetar a arquitetura da ferramenta de geração de código. | | | | |
| Redação do TC I | | | | |

## Trabalho de Conclusão II

| Etapa | Meses | | | |
|---|---|---|---|---|
| | Mar | Abr | Mai | Jun |
| Desenvolvimento do núcleo de geração de código | | | | |
| Desenvolvimento dos módulos de especificação básicos | | | | |
| Estudar possibilidade de criação de *plugins* | | | | |
| Validação da ferramenta de geração de código | | | | |
| Redação TC II | | | | |

# BIBLIOGRAFIA


BECK, Kent. **Programação extrema Explicada: acolha as mudanças.** Porto Alegre: Bookman, 2004. 182 p.

BRAUDE, Eric. **Projeto de software:** da programação à arquitetura: uma abordagem baseada em Java**.** Porto Alegre: Bookman, 2005. 619 p.

CZARNECKI, Krysztof; EISENECKER, Ulrich. **Generative Programming:** Methods, Tools, and Applications. Addison-Wesley, 2000. 864 p.

DATE, C.J. **Introdução a sistemas de banco de dados.** Rio de Janeiro: Elsevier, 2003.

FOWLER, Martin; SCOTT Kendall. **UML essencial:** Um breve guia para a linguagem-padrão de modelagem de objetos. Porto Alegre: Bookman, 2005. 160 p.

FOWLER, Martin. **Domain specific languages.** Addison-Wesley Professional: 2010. 240 p.

GAMMA, Erich et al. **Padrões de projeto:** soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000. 364 p.

GANE, Chris. **CASE:** o relatório Gane. Rio de Janeiro: LTC, 1990.

GREENFIELD, Jack. **Software Factories:** Assembling Applications with Patterns, Models, Frameworks, and Tools. Microsoft MSDN. 2004. Disponível em: <http://msdn.microsoft.com/en-us/library/ms954811.aspx>. Acesso em: 09 set. 2011.

HERRINGTON, Jack. **Code generation in action.** Greenwich: Manning, 2003. 372 p.

ISO/IEC 11179 (formally known as the ISO/IEC 11179 Metadata Registry (MDR) standard.

MARTIN, James. **Técnicas estruturas e CASE.** São Paulo: McGraw Hill, 1991.

MCCONNELL, Steve. **Code complete:** um guia prático para a construção de software. Porto Alegre: Bookman, 2005. 928 p.

MENDES, Sueli; AGUIAR, Teresa Cristina de. **Métodos para especificação de sistemas.** São Paulo: Edgard Blücher, 1989. 183 p.

PAULA FILHO, Wilson de Pádua. **Engenharia de Software:** Fundamentos, Métodos e Padrões. 3. ed. Rio de Janeiro: LTC, 2009.

PRESSMAN, Roger S. **Engenharia de Software**. 6. ed. Porto Alegre: AMGH, 2010. 720 p.

SEBESTA, Robert W. **Conceitos de linguagens de programação.** 4. ed. Porto Alegre, RS: Bookman, 2000. 624 p.

SILVA, Nelson Peres da. **Projeto e desenvolvimento de sistemas.** São Paulo, SP: Érica, 1994. 162 p.