MICHEL BEGIO SINHORETTI

PROPOSTA DE UMA FASE DE AVALIAÇÃO NO PROCESSO DE DESENVOLVIMENTO DE UMA LINGUAGEM DE DOMÍNIO ESPECÍFICO

Monografia apresentada ao PECE – Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para conclusão do curso de MBA em Tecnologia de Software.

São Paulo
2013

MICHEL BEGIO SINHORETTI

PROPOSTA DE UMA FASE DE AVALIAÇÃO NO PROCESSO DE DESENVOLVIMENTO DE UMA LINGUAGEM DE DOMÍNIO ESPECÍFICO

Monografia apresentada ao PECE – Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para a conclusão do curso de MBA em Tecnologia de Software.

Área de Concentração: Tecnologia de Software

Orientador: Prof. Fábio Levy Siqueira

São Paulo
2013

DEDICATÓRIA

Dedico esta monografia às pessoas que sempre estiveram ao meu lado, me apoiando nas difíceis decisões da vida e principalmente acreditando no meu potencial.
Aos meus pais Antonio e Helena e minha esposa Andrea.

AGRADECIMENTOS

Agradeço a todos que contribuíram de maneira direta ou indireta para a elaboração dessa monografia, em especial:

Ao meu orientador Fábio, pelo seu apoio e dedicação, o que tornou possível a realização deste trabalho.

Ao meu colega Wagner, pela ajuda e auxilio para a conclusão desta monografia.

Ao meu pai Antonio e minha mãe Helena, pelo amor, dedicação e incentivo em todas as decisões da minha vida.

À minha esposa Andrea, pelo carinho, apoio e, principalmente, seu amor.

Ao PECE – Programa de Educação Continuada em Engenharia – pelos bons serviços oferecidos, que foram fundamentais para a conclusão deste curso.

RESUMO

Em busca de uma maior produtividade na construção de um software, tem-se dado uma grande importância no uso de linguagens de domínio específico (DSL, do inglês *Domain Specific Language*). Ao fornecer notações e construções adaptadas para um determinado domínio, as DSLs oferecem ganhos na facilidade de utilização. Isso faz com que o desenvolvedor tenha apenas que focar no problema em questão e não nos detalhes de como implementar a solução.

No entanto, foi detectada a ausência de um processo que auxilie a tomada de decisão de se criar uma DSL, ou que avalie a sua construção. A proposta deste trabalho é apresentar um processo responsável pela avaliação do desenvolvimento de uma DSL, também sendo apresentado um exemplo prático de um processo de avaliação, aplicando um experimento formal, a fim de ilustrar os seus benefícios.

ABSTRACT

Searching for a better productivity in software construction, it has been given a great importance in the use of domain specific languages. By providing notations and constructions adapted for a specific domain, DSLs provide benefits in the ease of use. By using a DSL, the developer can focus only in the problem itself and not in details of how to implement the solution.

However, it has been detected the absence of a process that assists the decision-making of creating a DSL, or that evaluates its construction. The proposal of this work is to present a process that will be responsible for evaluating the development of a DSL. It also presents a practical example of an evaluation process, applying a formal experiment, in order to illustrate its benefits.

LISTA DE ILUSTRAÇÕES

Pág.

LISTA DE TABELAS

Pág.

LISTA DE ABREVIATURAS E SIGLAS

DSL	*Domain-Specific Language*
SQL	*Structured Query Language*
GPL	*General Purpose Language*
XML	*Extensible Markup Language*
IDE	*Integrated Development Environment*
GUI	*Graphical User Interface*
API	*Application Programming Interface*
AVOPT	*Doman-specific analysis, verification, optimization, parallelization, and transformation*

SUMÁRIO

Pág.

1. INTRODUÇÃO

A Engenharia de Software visa a produção de um software com alta qualidade e baixo custo. Essa necessidade faz com que alguns profissionais e estudiosos busquem maneiras de alcançar esse objetivo. Com isso, novos processos são desenvolvidos e discutidos, novos padrões definidos e discussões realizadas, o que permite a criação de produtos cada vez melhores.

Essa busca faz com que processos ou padrões definidos sejam alterados ou até estendidos para que suas contribuições possam ser readaptadas para um mercado em constante evolução. Este trabalho leva em consideração essa busca e apresenta maneiras de tornar o software um produto cada vez melhor.

1.1. Motivações

Uma linguagem de domínio específico (DSL, do inglês *Domain Specific Language*) é uma linguagem de programação que possui um alto nível de abstração e um vocabulário próximo ao utilizado pelos especialistas do domínio (DEURSEN, KLINT, VISSER, 2000).

De acordo com Deursen, Klint e Visser (2000), não é de hoje que se faz uso de uma DSL para abstrair a complexidade do domínio do problema e fazer com que a programação de um software se torne uma tarefa mais fácil.

Ao utilizar uma DSL não se pretende extinguir a profissão de programador, e sim auxiliar em um processo de entendimento do domínio do problema, fazendo com que o especialista do domínio – que muitas vezes não é um programador – possa alterar e modificar as DSLs (DEURSEN, KLINT, VISSER, 2000).

Este trabalho foi iniciado a partir da visão de se ter um produto "DSL" com a finalidade de substituir algumas partes de um processo de desenvolvimento e fazer com que o uso da DSL torne a tarefa mais produtiva. Porém surgiu a dúvida de como avaliar se a abordagem será mais vantajosa ou se trará os benefícios

esperados. Houve também a questão de como apresentar à gerência dados plausíveis e consistentes dos benefícios no uso de tal DSL.

1.2. Objetivo

O objetivo deste trabalho é apresentar uma proposta para estender as fases de desenvolvimento de uma DSL descritas por Mernik et al. (2005), fazendo uso de um protótipo e de um experimento formal a fim de avaliar a criação de uma DSL.

1.3. Justificativa

A decisão de se utilizar uma DSL não é uma tarefa fácil. Para auxiliar essa decisão Mernik et al. (2005) definiu alguns padrões e fases de desenvolvimento a serem levados em consideração. Porém, esses padrões auxiliam apenas na decisão de se criar ou não uma DSL, mas eles não abordam como avaliar se a DSL trará os benefícios esperados, como, por exemplo, um ganho maior de produtividade. Essa avaliação de ganho de produtividade pode ser analisada de diversas formas, por exemplo, a medição no tempo de desenvolvimento de um artefato ou pelas contagens de linhas de códigos.

Apesar de ser possível encontrar relatos de experiência utlizando uma DSL, não foi encontrado um trabalho que propusesse uma maneira de avaliar a utilização de uma DSL de forma metódica, dentro do processo de desenvolvimento da DSL.

1.4. Estrutura do Trabalho

Este trabalho está estruturado em cinco capítulos:

- *Capítulo 1 – Introdução* - Apresenta as motivações, o objetivo, a justificativa e a estrutura do trabalho.
- *Capítulo 2 – Linguagem de Domínio Específico* - Apresenta a definição de DSL, classificando os tipos de DSLs e por final destaca as vantagens e desvantagens em sua utilização.
- *Capítulo 3 – Avaliação de uma DSL* - Apresenta a fase de avaliação, descrevendo de que maneira será feita a avaliação.

- *Capítulo 4 – Exemplo da Avaliação* - É apresentado um exemplo de avaliação através de um experimento formal.
- *Capítulo 5 – Considerações Finais* - Descreve as considerações finais do trabalho, destacando os resultados obtidos.
- *Apêndice A* – Apresenta o plano tabular proposto para o experimento.
- *Apêndice B* – Apresenta o plano tabular de exemplo.

2. LINGUAGEM DE DOMÍNIO ESPECÍFICO

Segundo Deursen, Klint e Visser (2000) em todos os ramos da ciência e engenharia podem-se ter duas abordagens: as genéricas e as que são específicas. A abordagem genérica fornece uma solução geral para diversos problemas em uma determinada área, mas tal solução pode ser de qualidade inferior. Uma abordagem específica fornece uma solução melhor, porém para um pequeno conjunto de problemas.

Linguagens mais antigas como Cobol, Fortran e Lisp, surgiram como linguagens dedicadas a resolver problemas de uma determinada área, mas com o tempo foram evoluindo como linguagens de propósito geral (DEURSEN, KLINT, VISSER, 2000). Para tentar especializar novamente a linguagem, foram criadas algumas soluções, por exemplo, bibliotecas de sub-rotinas que executam tarefas relacionadas em domínios bem definidos, como equações diferenciais, gráficos, interfaces de usuário e banco de dados (DEURSEN, KLINT, VISSER, 2000).

Este capítulo apresenta os conceitos relacionados à DSL. Com isso são discutidos métodos de desenvolvimento, tipos existentes e classificações a fim de apresentar as vantagens e desvantagens no uso de DSLs.

2.1. Definição

A ideia de uma linguagem de domínio específico não é uma novidade. Há décadas já são utilizadas, porém não se tem um consenso sobre os termos usados e o seus significados (DEURSEN, KLINT, VISSER, 2000). Várias definições foram citadas no artigo de Mernik et al. (2005): linguagem orientada a aplicação, linguagem de propósito especial, linguagem especializada, linguagem específica de uma tarefa, linguagem específica de uma aplicação e linguagens pequenas.

DSL é uma linguagem de programação focada em um problema de um domínio específico, diferentemente das Linguagens de Propósito Geral (GPL, do inglês *General Purpose Languages*) como, por exemplo, C, C#, Java e Visual Basic, em que se propõe resolver os problemas gerais na construção de um software ou

para qualquer domínio (THIBAULT, MARLET, CONSEL, 1999). Ao fornecer notações e construções adaptadas para um determinado domínio, as DSLs em comparação com as GPLs, oferecem ganhos expressivos na facilidade de utilização, ganhos de produtividade e redução de custo (MERNIK et al., 2005).

Outro motivo que ajuda a utilização das DSLs é a abstração do domínio em questão, o que permite que o usuário, no caso de um programador, execute tarefas de programação mais simples (SERRA, 2004).

2.2. Classificação das DSLs

Segundo Deursen, Klint e Visser (2000) existem centenas de DSLs criadas, porém dentre todas, apenas um subconjunto é descrito na engenharia de software, sendo que os seus domínios podem ser agrupados nas seguintes áreas:

- **Software de engenharia:** produtos financeiros, controle, coordenação, software de arquitetura e banco de dados.
- **Software de sistemas:** descrição e análise de árvore de sintaxe abstrata, driver de vídeo, cache de protocolos, estruturas de dados em C e especialização do sistema operacional.
- **Software multimídia:** manipulação de imagens, animação 3D e desenho.
- **Telecomunicações:** protocolos de comunicação.
- **Outros:** simulação, controle de robôs e projeto de hardware digital.

Das áreas descritas acima, Domingues (2009) e Fowler (2010) classificam as DSLs a partir da forma que são construídas. Com isso, existem DSLs textuais, que podem ser internas e externas e DSLs gráficas.

DSLs gráficas necessitam de uma ferramenta que represente sua notação e também um gerador de artefatos (código, dados, arquivo de configuração). Uma DSL gráfica pode fazer uso de um ambiente integrado de desenvolvimento (IDE, do inglês *Integrated Development Environment*) especializado para definição e execução de DSLs. Esse ambiente é usado não apenas para determinar uma

estrutura de uma DSL, como também para customização e edição de um ambiente para que outras pessoas possam criar diagramas (DOMINGUES, 2009).

Existem diversas ferramentas no mercado com o foco na criação, execução e manutenção de DSLs gráficas. Por exemplo, existe o plug-in *DSL Tools* para o Visual Studio da *Microsoft* (MICROSOFT, 2013).

DSL textual externa é uma linguagem separada da linguagem principal e que pode ser compilada, interpretada ou executada, e tem uma sintaxe própria ou usa alguma representação pré-definida como, por exemplo, XML. Um script em DSL externa geralmente é analisado por um código na linguagem principal usando técnicas de análise de texto. Exemplos de DSL externa são: expressões regulares, SQL, AWK e arquivos de configuração XML para sistemas como Struts e Hibernate (FOWLER, 2010).

DSL textual interna é uma linguagem que é expressa dentro de uma linguagem GPL. Nesse caso, a DSL requer uma linguagem principal e sua sintaxe é válida dentro da linguagem de propósito geral. Com isso, a linguagem base pode personalizar símbolos ou mecanismos de controle de uma maneira especial, de acordo com a necessidade do domínio (FOWLER, 2010). Exemplos de DSL interna são: parte do framework Ruby on Rails, por exemplo, (*Active Records* para mapeamento objeto-relacional) entre outras (DOMINGUES, 2009).

A utilização de uma DSL interna também se faz pelo uso de *Fluent Interfaces*, que é uma técnica usada para fazer com que uma GPL que possa conter códigos complexos se torne mais fluente e de fácil entendimento, fazendo uso de determinadas API (do inglês *Application Programming Interface)* (FOWLER, 2010).

Conforme citado por Fowler (2010), atualmente, esses dois estilos de DSLs desenvolveram suas próprias comunidades para troca de ideias. Podem-se encontrar pessoas que são muito experientes em DSL internas, mas não sabem como construir uma DSL externa. Isso pode ser problemático, porque as pessoas podem não estar escolhendo a melhor abordagem para um projeto.

2.3. Vantagens e desvantagens na utilização de uma DSL

A utilização de uma DSL envolve muitas vantagens e desvantagens. Algumas vantagens são (MERNIK et al., 2005; DEURSEN, KLINT, VISSER, 2000; SERRA, 2004; FOWLER, 2010):

- Devido ao fato de uma DSL abstrair o domínio do problema, o especialista do domínio pode entender, validar, modificar e muitas vezes até desenvolver programas em DSL;
- DSLs bem construídas podem aumentar a produtividade, confiabilidade e facilitar a manutenção;
- Como as DSLs incorporam o conhecimento do domínio, elas permitem a conservação desse domínio tornando possível a reutilização desse conhecimento;
- DSL permite a validação e otimização de um domínio;
- DSL permite melhorar as abordagens de teste, utilizando DSL específicas para teste, tanto para GPL como para DSL.

Por outro lado, as desvantagens são (MERNIK et al., 2005; DEURSEN, KLINT, VISSER, 2000; SERRA, 2004; FOWLER, 2010):

- Os custos de concepção, implantação e manutenção de uma DSL são altos se comparado com um software construído com uma linguagem GPL;
- Os custos de treinamento para usuário da DSL;
- A disponibilidade limitada de DSLs;
- O código gerado em GPL é escrito sob medida para uma determinada situação, porém o código gerado a partir de uma DSL pode ser menos eficiente;
- Desenvolvimento de uma DSL é difícil, exigindo conhecimento do domínio e também conhecimento no desenvolvimento de uma nova linguagem.

Essas vantagens e desvantagens devem ser avaliadas atentamente pelo gerente de um projeto, buscando o equilíbrio entre elas (DEURSEN, KLINT, VISSER, 2000).

Em relação ao custo de construção de uma DSL, ele pode ser apenas uma parte do custo de um projeto. Porém ainda sim é um custo, e um custo a ser mantido tanto no desenvolvimento, como na manutenção da DSL (FOWLER, 2010).

2.4. Fases no Desenvolvimento de uma DSL

Segundo Mernik et al. (2005), para se desenvolver uma DSL deve-se passar por algumas fases que são: decisão, análise, projeto e implementação. Tais fases não são necessariamente sequenciais, porém estão todas relacionadas entre si.

Deursen, Klint e Visser (2000) descrevem as fases de outra maneira, não incluindo a decisão de se criar uma DSL como uma fase de desenvolvimento. Nesse caso, as fases são: análise, implementação e uso. Para este trabalho será seguida a organização de Mernik et al. (2005) que é ilustrada na Figura 1. A seguir serão discutidas cada uma das fases.

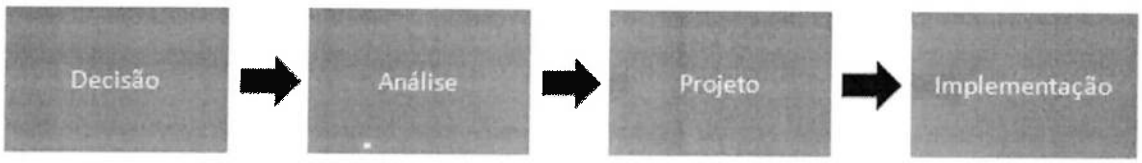

Figura 1: Fases no desenvolvimento de uma DSL adaptado de Mernik et al. (2005).

2.4.1. Fase de decisão

Conforme comentado anteriormente, Mernik et al. (2005) descrevem que na fase de decisão é quando se deve fazer uma análise para levantar as vantagens e desvantagens da criação de uma DSL. Altos custos no desenvolvimento de uma DSL não devem superar o desenvolvimento principal do software.

A DSL deve proporcionar benefícios após sua implantação e não impedimentos. Segundo Fowler (2010), não se deve utilizar uma DSL se em algum caso específico não se conseguir visualizar os benefícios do seu uso, ou se os custos de construção forem maiores que os benefícios.

Adotar uma DSL já existente é muito menos custoso e exige muito menos experiência do que desenvolver uma nova. Porém adotar uma DSL que não é bem conhecida pode ser arriscado. Para se decidir em criar ou não uma DSL, Mernik et al. (2005) sugerem alguns padrões de decisão que podem ser seguidos:

- **Padrão *Notation:*** esse padrão é utilizado quando é necessário a disponibilidade de notações adequadas (novas ou existentes) de um domínio específico.
- Dois subpadrões importantes são:
    - Transformar uma notação visual para textual, ao fazer uma notação visual existente disponível em textual. Isso pode tornar fácil a composição de grandes programas ou especificações e possibilitar a utilização do padrão de decisão AVOPT (do inglês *Doman-specifc analysis, verification, optimization, parallelization, and transformation*).
    - Adicionar uma notação de melhor usabilidade para uma API já existente ou transformar uma API em uma DSL.
- **Padrão *AVOPT:*** domínio específico de análise, verificação, otimização, paralelização e transformação de programas de aplicação escritos em uma linguagem GPL que geralmente não são viáveis o seu uso, devido à complexidade do código de origem ou que os padrões usados não são bem definidos. Portanto o uso de uma DSL adequada faz com que essas operações se tornem possíveis.
- **Padrão *Task automation***: programadores muitas vezes gastam seu tempo em tarefas de programação GPL que são entediantes e que geralmente seguem o mesmo padrão. Nesse caso, o código poderia ser gerado automaticamente por uma DSL.
- **Padrão *Produtct line***: os desenvolvedores de um software de linha de produto utilizam uma arquitetura comum e são desenvolvidos a partir

de um conjunto comum de elementos básicos. Nesse caso, ao utilizar uma DSL, pode-se muitas vezes facilitar a sua especificação.

- **Padrão *Data structure representation:*** *esse p*adrão é utilizado quando o código depende de estruturas de dados inicializadas e cuja complexidade torna difícil sua programação e manutenção. Tais estruturas são muitas vezes mais facilmente expressas usando uma DSL. Por exemplo, uma lista de gráficos pode ser facilmente expressa como uma lista de conexões de caminhos.
- **Padrão *System front-end***: uma DSL baseada em front-end pode muitas vezes ser utilizada para o tratamento de um sistema de configuração e adaptação. Por exemplo, um sistema de software complexo oferece centenas de opções de configuração, com isso, tornar um sistema programável através de uma DSL, oferece aos usuários um mecanismo organizado e aberto para configurar e adaptar.
- **Padrão *Interaction***: esse padrão objetiva tornar a interação programável. Interações baseadas em menu ou texto frequentemente devem ser suportadas por uma DSL apropriada para especificar entradas complicadas ou repetitivas. A linguagem de macro do Excel é um exemplo que suporta sua manipulação interativa e que a torna programável.
- **Padrão *GUI construction***: construção de GUI é geralmente feito usando uma DSL.

2.4.2. Fase de análise

No caso da fase de análise, tanto Mernik et al. (2005) como Deursen, Klint e Visser (2000) descrevem a necessidade de identificar o domínio do problema, reunindo todas as informações relevantes. Nesta fase deve-se analisar as particularidades, os pontos relevantes que podem ser implementados em uma DSL e buscar os objetos-chave do domínio como operações básicas sobre esses objetos. Também é necessário ter o envolvimento do especialista do domínio, pois frequentemente ele será acionado para ajudar a esboçar os problemas que devem ser resolvidos no domínio em questão.

2.4.3. Fase de projeto

Segundo Mernik et al. (2005), a fase de projeto pode ser caracterizada por duas dimensões ortogonais: a relação entre a DSL e uma linguagem existente e o método de especificação do seu *design*.

A maneira mais fácil de projetar uma DSL é através de uma linguagem existente, o que torna mais fácil a implementação. Outro ponto a ser destacado ao usar uma linguagem já existente é a maior familiaridade com a linguagem, porém isso se aplica apenas aos usuários programadores na linguagem existente. O uso da linguagem existente pode ser feito de três formas (MERNIK et al., 2005):

- ***Piggyback***: em que são usadas partes de uma linguagem existente;
- ***Especialização***: em que se restringe a linguagem existente ao domínio do problema;
- **Extensão**: que consiste adicionar novos recursos para uma linguagem existente atribuindo novas características.

O método de especificação do *design* de uma DSL pode ser formal ou informal. No método informal, a especificação da linguagem que está sendo criada, como a sua sintaxe, é feita sem quaisquer regras ou padrões estabelecidos. Já no caso do método formal, segue-se um conjunto de regras sintáticas e a semântica disponível.

2.4.4. Fase de implementação

Nesta fase é construída a DSL utilizando os padrões definidos na fase de projeto apresentada anteriormente. Quando se cria uma DSL do início é necessário implementar todo o processo de tradução da sintaxe da DSL para código de máquina. Nesse caso Mernik et al. (2005) destaca dois padrões de implementação: *compilação* e *interpretação*.

No padrão compilação a linguagem definida na DSL é analisada e traduzida em código de máquina ou em código de uma GPL, que será compilada

posteriormente. Já no caso do padrão interpretação, a linguagem será interpretada, não sendo necessária a construção de um compilador, devido fazer uso de ferramentas especializadas na criação de compiladores e interpretadores de DSL.

No caso da DSL ser desenvolvida a partir de uma linguagem existente, a tradução pode ser feita através de mais dois padrões: *embedding compiler/interpreter ou extensible compiler/interpreter.* No primeiro caso, todo o trabalho de compilação será feito pelo compilador da linguagem existente. Uma grande vantagem disso é não ser necessário conhecimento de compiladores. Já no caso do *extensible compiler/interpreter*, o compilador da linguagem existente é estendido na maneira de englobar os aspectos da DSL.

2.5. Considerações do capítulo

Nesse capítulo foi apresentada a definição de DSL, quais domínios podem ser agrupados e em quais áreas. Também foram apresentadas algumas vantagens e desvantagens em sua utilização.

Foi apresentada a proposta de Mernik et al. (2005) que é a implementação de algumas fases de desenvolvimento, visando tornar o processo de desenvolvimento mais fácil. No próximo capítulo será apresentada uma proposta de estender tais fases descritas por Mernik et al. (2005), fazendo uso de um protótipo e um experimento formal a fim de avaliar a criação de uma DSL.

3. AVALIAÇÃO DE UMA DSL

Após a discussão no capítulo anterior sobre o que é uma DSL e quais os riscos envolvidos em sua criação, pode-se notar que todo projeto de software, utilizando uma DSL ou não, deve proporcionar um retorno esperado, tanto em investimento como em benefícios, como, por exemplo, maior produtividade ou uma menor taxa de defeitos ao utilizar uma determinada ferramenta, método ou técnica.

No caso de um projeto de uma DSL, mesmo fazendo uso das recomendações de Mernik et al. (2005) sobre as fases de desenvolvimento e os padrões descritos, a DSL desenvolvida pode não acrescentar o valor esperado ao projeto. Neste capítulo será apresentada uma proposta de estender as fases de desenvolvimento proposta por Mernik et al. (2005), fazendo uso de um protótipo e um experimento formal a fim de avaliar a criação de uma DSL.

3.1. Decisão

Decidir o desenvolvimento de uma DSL não é uma tarefa trivial. O investimento no desenvolvimento de uma DSL (incluindo sua implantação) tem que ser justificado por um produto final mais econômico. Na visão do custo do desenvolvimento, adotar uma DSL já existente é muito menos custoso e exige menos experiência do que desenvolver uma nova (MERNIK et al., 2005).

Os padrões de decisão propostos por Mernik et al. (2005) e discutidos na Seção 2.4 auxiliam a decidir como criar uma DSL e de que maneira criá-la. Porém não é discutido se a DSL deve ser criada. Portanto, mesmo fazendo uso de padrões pode-se criar uma DSL e não ter como saber seus reais benefícios até o momento de seu uso.

Este trabalho propõe a criação de um protótipo de uma DSL seguindo as fases e padrões sugeridos por Mernik et al. (2005) para analisar os benefícios da utilização de uma DSL. Como forma de avaliar esse protótipo, será realizado um experimento formal e analisar os reais benefícios desse protótipo de DSL.

3.2. Fase de avaliação

Para o desenvolvimento do protótipo de uma DSL deve-se passar pelas fases descritas por Mernik et al. (2005) (decisão, análise, projeto e implementação) e seguir por uma nova fase chamada de avaliação.

Com essa nova fase, o fluxo de criação de uma DSL que foi apresentado na seção 2.4 será alterado e ficará conforme a Figura 2, em que foi criada a nova fase responsável pela avaliação através de um experimento formal.

Nessa fase será feita a avaliação da DSL através de um experimento formal, analisando se o protótipo desenvolvido trará os benefícios esperados após sua criação.

A vantagem de se criar um protótipo de uma DSL é reduzir o tempo entre o desenvolvimento da DSL e sua possível aprovação. Assim, espera-se que a fase de avaliação mostre se a DSL é válida ou não de acordo com o que motivou a sua criação, como, por exemplo, a melhoria na produtividade, menor taxa de defeitos ou até facilidade na manutenção. Com isso, caso uma DSL não seja aprovada, o tempo e custo gasto no protótipo não terão um grande impacto no projeto final.

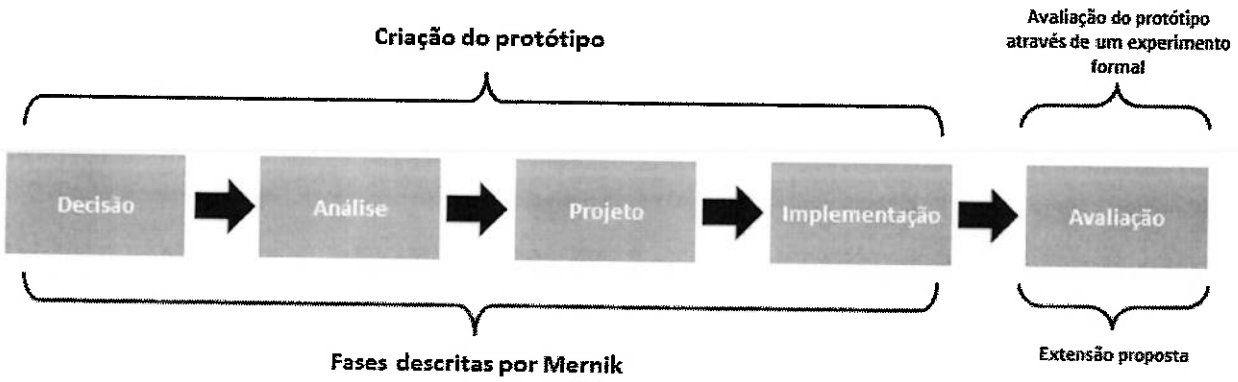

Figura 2: Extensão proposta para criação de um protótipo de DSL, adaptado de Mernik et al. (2005).

3.3. Técnicas de avaliação

Para se avaliar um método, técnica ou ferramenta de uma maneira científica, podem-se utilizar três tipos de estudos: estudo de caso, experimentos formais ou pesquisas (*survey*) (PFLEEGER, 1995).

Segundo Pfleeger (1995), várias orientações podem auxiliar na escolha entre um experimento formal ou um estudo de caso, porém, o fator principal é o nível de controle necessário que se tem para executar um experimento formal. Pfleeger (1995) descreve alguns fatores que auxiliam nessa escolha que são descritos abaixo e também apresentado na Tabela 1:

- **Nível de controle**: se for possível um elevado nível de controle das variáveis que afetam a verdade da hipótese, então pode-se considerar um experimento formal. Caso contrário, o experimento não é possível e a utilização de um estudo de caso é a técnica mais indicada.
- **Nível de replicação**: de acordo com o nível de controle que se tem em um experimento formal, os resultados podem ser aproveitados em uma comunidade maior. Já no caso de um estudo de caso, a utilização do resultado se dá apenas na organização que está sendo analisada ou no caso de outras organizações semelhantes.
- **Dificuldade de controle**: o nível de controle satisfaz as preocupações técnicas, mas também deve tratar das preocupações práticas da pesquisa. Em um estudo de caso pode ser possível, mas muito difícil, controlar as variáveis, devido ao alto custo de se fazer isso.
- **Custo de replicação**: um aspecto importante que se deve considerar ao replicar uma situação básica de experimentação é o custo envolvido. Como um experimento formal dispõe de um nível maior de replicação devido ao controle das variáveis, tal replicação demanda um investimento maior.

Tabela 1: Valores a serem considerados ao adotar uma técnica de avaliação adaptado de Pfleeger (1995).

Fator	Experimento formal	Estudo de caso
Nível de controle	Alto	Baixo
Dificuldade de controle	Baixo	Alto
Nível de replicação	Alto	Baixo
Custo de replicação	Alto	Baixo

A fase de avaliação proposta tem como objetivo avaliar se o protótipo de DSL é válido com base nas especificações de um determinado projeto. Com isso, essa fase irá fazer uso de um experimento formal para avaliar se tal DSL é válida.

A decisão em utilizar um experimento formal ao invés de um estudo de caso não se deu por acaso. Pfleeger (1995) comenta que o primeiro passo para um experimento é decidir o que deseja ser investigado. Uma técnica que ajuda nessa decisão é expressar o que deseja ser investigado como uma hipótese a ser testada, ou seja, deve-se especificar o que se deseja saber ou ser investigado. No caso da proposta desse trabalho, a fase de avaliação precisa avaliar as vantagens da DSL em comparação a outro método, portanto o experimento formal provê recursos necessários para que tal comparação possa ser feita.

Os detalhes na aplicação de um experimento formal serão apresentados de maneira mais ampla na Seção 4, na qual será mostrado um exemplo prático de um experimento formal.

3.4. Considerações do capítulo

Nesse capítulo foi feita uma discussão de como avaliar uma DSL e de que forma fazer essa avaliação antes de se desenvolver uma DSL por completa. Foi discutida uma maneira de como estender a fases descritas por Mernik et al. (2005) através da criação de uma nova fase chamada de avaliação. Essa fase faz uso de um protótipo que será avaliado através de um experimento formal.

No capítulo a seguir será apresentado um experimento formal desenvolvido especificamente para esse estudo como forma de apresentar os seus benefícios.

4. EXEMPLO DA AVALIAÇÃO

Seguindo a proposta apresentada no capítulo anterior, neste capítulo é apresentado um exemplo de avaliação através de um experimento formal que foi desenvolvido especificamente para esse estudo.

4.1. Contexto[1]

O desenvolvimento da DSL foi feito em uma empresa multinacional no ramo de Pesquisa e Processamento de Dados que está situada em São Paulo (SP) e que atualmente possui clientes em mais de 100 países. A empresa é dividida em seis áreas de atuação: Marketing, Publicidade, Mídia, Lealdade, Previsão e Opinião Pública.

A área de Lealdade realiza pesquisas para verificar qual o nível de satisfação e fidelidade que um determinado cliente possui. Grande parte dessas pesquisas é muito parecida, sendo assim o processamento e a entrega de resultados tendem a ser similares e contam com processos repetitivos. Isso viabiliza uma iniciativa de automação de alguns artefatos empregados por esse processo.

A cada novo processamento é desenvolvido um software que tem como saída os cálculos solicitados pelo cliente através de um documento conhecido como Plano Tabular. Esse documento contém todos os requisitos do cliente, que são os tipos de cálculos e regras de negócio que o software desenvolvido deverá respeitar (como exemplo, o Plano Tabular utilizado para o experimento no Apêndice A). Tal software utiliza um *framework* proprietário que facilita a construção, manutenção, além de manter um padrão de código exigido pela empresa.

Devido ao grande volume de dados processados diariamente, todos os cálculos que deverão ser feitos pelo software ficam armazenados no banco de

[1] Por motivos de sigilo comercial, informações como o nome da organização ou o nome do cliente serão omitidas.

dados em *stored procedures*. Esses cálculos são chamados a partir de uma classe no software que deverá ser criada em cada software de processamento construído.

4.2. Protótipo da DSL

A iniciativa de criar uma DSL para esse contexto se deve principalmente pela necessidade de eliminar algumas tarefas repetitivas que são feitas no software de processamento de dados. Por exemplo, a cada novo processamento solicitado, os scritps em *transact-sql* que contêm a regra de negócio muitas vezes devem ser alterados ou reescritos. Isso torna uma tarefa entediante e, em alguns casos, isso pode levar a erros de digitação.

Com isso, ao se decidir criar uma DSL, pode-se apoiar no padrão de decisão *Task automation* mencionado por Mernik et al. (2005), o qual trata exatamente da necessidade de eliminar tarefas repetitivas através de uma DSL.

A DSL utilizada no exemplo de avaliação foi construída a partir de um ambiente integrado de desenvolvimento (IDE, do inglês *Integrated Development Environment*) especializado para definição e execução de DSLs gráficas, conhecido como Visual Studio 2008 DSL Tools (MICROSOFT, 2013). Na Figura 3 é mostrada uma tela do ambiente criado com um diagrama de conceitos relacionado ao domínio do experimento.

A DSL utilizada para este exemplo gera como saída scripts em *transact-sql,* contendo a regra de negócio em forma de *stored procedures*, e classes em *Visual Basic.Net*, contendo as chamadas para as *stored procedures*. A geração desses scripts e classes de maneira automatizada deverá ser feita alinhada com os propósitos do domínio que, no caso desse exemplo, são o processamento de pesquisas que envolvem questões como quais cálculos devem ser gerados, que tipos de períodos devem ser processados e em quais níveis.

Essa análise do domínio foi feita por um especialista do domínio que coletou informações relevantes e operações básicas que o protótipo a ser criado deverá

conter. Tais dados foram mapeados na DSL e definidos como conceitos que se relacionam entre si.

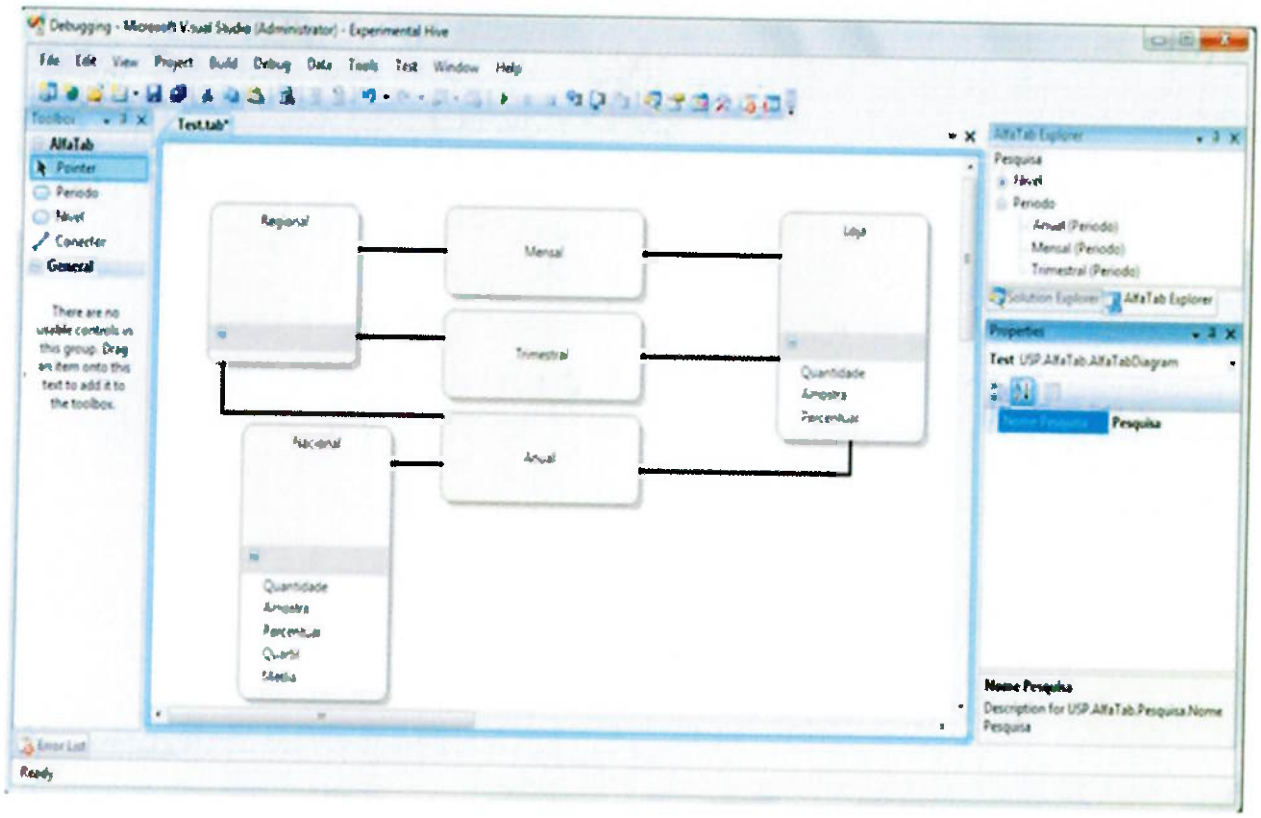

Figura 3: Tela do ambiente criado usando o Visual Studio 2008 DSL Tools.

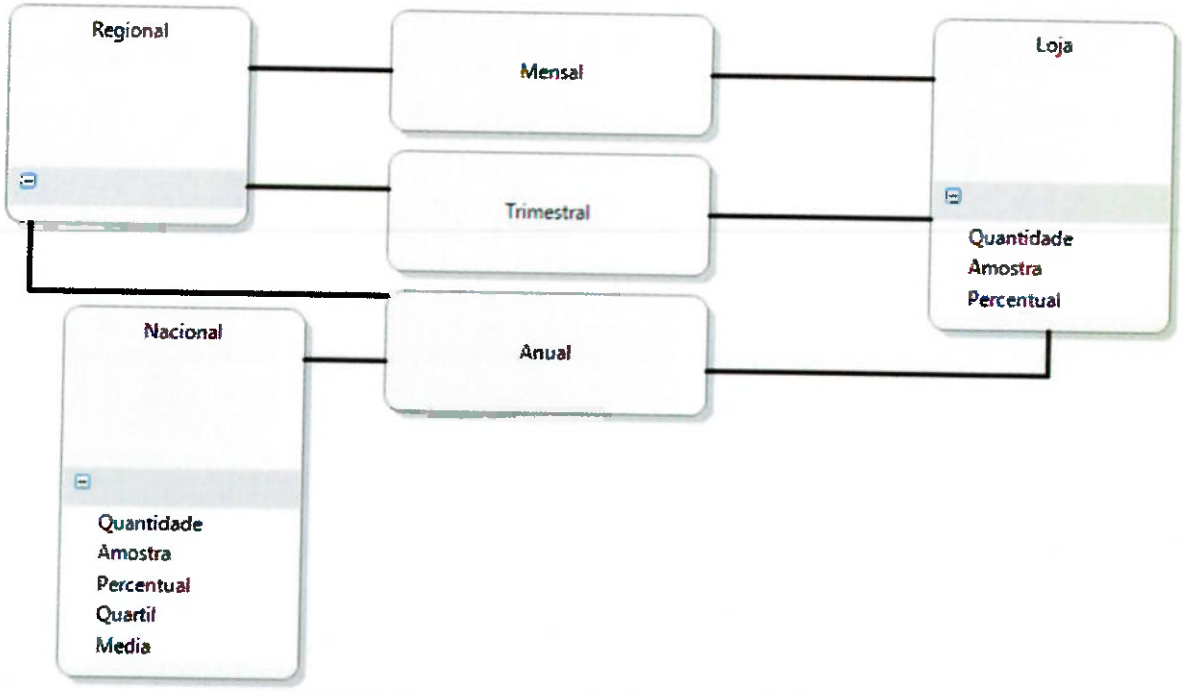

Figura 4: Conceitos mapeados no protótipo da DSL

Foram instanciados três conceitos que são ilustrados na Figura 4 e descritos mais detalhadamente abaixo:

- **Período**: em cada processamento é obrigatório à definição de qual período um cálculo deverá ser feito. Por exemplo: podem-se ter cálculos processados mensalmente, trimestralmente ou acumulados anualmente.
- **Nível**: todos os períodos definidos anteriormente nos conceitos de períodos devem ser calculados em determinados níveis de distribuição. Por exemplo: o universo de um processamento pode ser distribuído de forma regional ou em âmbito nacional.
- **Cálculos**: já com os conceitos de períodos e níveis definidos, devem-se definir quais cálculos serão atribuídos a esse conceito. Por exemplo: quantidade, amostra, percentual, etc.

Outro item importante que foi mapeado na DSL é a definição de relacionamento entre conceitos. Através do uso de um relacionamento, é feita a dependência entre conceitos, por exemplo, o conceito Período só pode existir se um conceito Nível for atribuído a ele, com isso, através da utilização do relacionamento entre esses conceitos que será definido essa dependência.

Esse item foi detectado pelo especialista do domínio e é fundamental para a criação consistente dos scripts, através do relacionamento entre um conceito e outro.

Para minimizar erros na geração dos scripts através da DSL, foram definidos alguns validadores semânticos a fim de inibir ações erradas e instruir de forma prática a criação correta dos scripts. Por exemplo: se um conceito for criado e nenhum cálculo for atribuído a esse conceito, a DSL irá informar ao usuário a inconsistência através de uma lista de erros que é ilustrada na Figura 5. A DSL permitirá gerar os scripts somente se todos os dados necessários forem criados e validados.

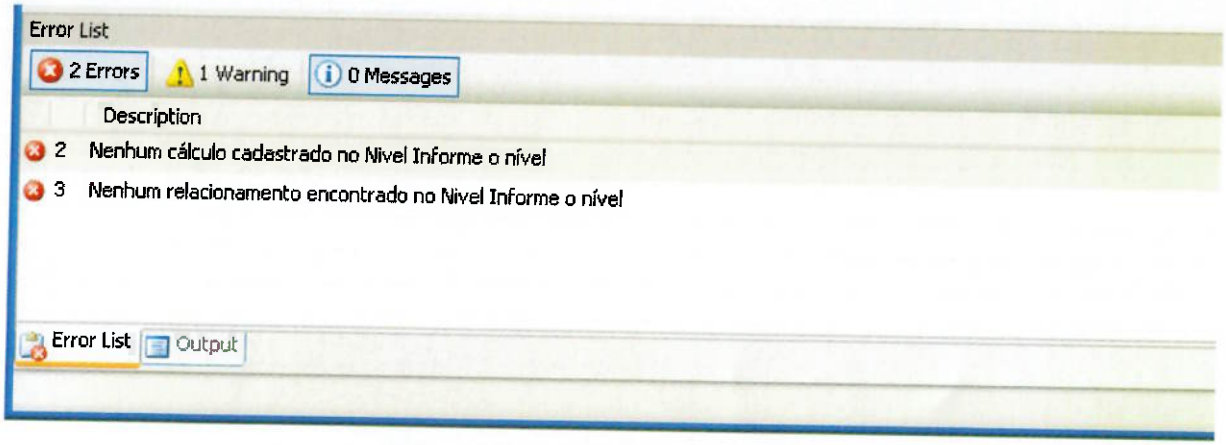

Figura 5: Validador Semântico.

4.3. Experimento

Conforme comentado na Seção 2, a utilização de uma DSL fornece uma variedade de benefícios, porém não se tem como avaliar antes de seu uso se realmente os benefícios imaginados serão obtidos e, com isso, se é válido investir no esforço do desenvolvimento de uma DSL. Para tratar disso é proposta a fase de avaliação. Ela será aqui descrita de forma prática, através de um experimento formal que foi criado com o propósito de exemplificar a avaliação do protótipo da DSL descrita anteriormente.

Com isso, o objetivo do exemplo do experimento é analisar se a criação de scripts em *transact-sql* e classes em *Visual Basic.Net* através do uso de uma DSL é mais produtivo que manualmente.

4.3.1. Hipótese

Um experimento geralmente é formulado para testar uma hipótese. A hipótese principal se chama hipótese nula e declara que não há nenhum relacionamento estatisticamente significante entre a causa e o efeito. O objetivo do experimento é rejeitar a hipótese nula em favor das hipóteses alternativas (PFLEEFER, 1995).

No caso desse experimento formularam-se as seguintes hipóteses:

- H*(nula)* – É menos produtivo ou não existe diferença de produtividade usar a DSL para a criação de scripts *transact-sql* e classes em *Visual Basic.Net* do que criá-los manualmente.
- H*(alternativa)* – É mais produtivo usar a DSL para a criação de scripts *transact-sql* e classes em *Visual Basic.Net* do que criá-los manualmente.

4.3.2. Variáveis e fatores do experimento

Pfleeger (1994) comenta que após a identificação das hipóteses que serão analisadas no experimento deve-se decidir quais variáveis afetam essa verdade e qual o grau de controle que se terá em cada variável.

Em um experimento existem dois tipos de variáveis:

- **Variável dependente**: são as saídas, resultados a serem estudados, observados ou medidos.
- **Variável independente**: são os dados de entrada, os quais estão sob o controle do executor. Simplificando, as variáveis independentes são tudo que afeta o experimento (PFLEEFER, 1995).

Na tabela 2 são apresentadas as variáveis independentes consideradas para o experimento. Devido à impossibilidade de aplicar o experimento em um ambiente separado, foi escolhido como ambiente de aplicação o mesmo onde os participantes escolhidos trabalham (VI1). Com isso o horário para aplicação do experimento será logo após o fim do expediente de cada participante (VI4). Isso será feito em dias separados (VI9), pois não há espaço disponível para a aplicação do experimento para todos os participantes simultaneamente.

Todos os participantes foram escolhidos por terem conhecimento necessário para a realização do experimento. Porém dentre os selecionados o nível de conhecimento é variável, havendo diferenças no conhecimento de processamento de dados (VI3) e sobre a ferramenta utilizada para criar os artefatos (VI7). Essas variáveis serão randomizadas.

Tabela 2: Variáveis independentes utilizadas no experimento.

VI1	Ambiente de aplicação do experimento
VI2	Complexidade do escopo do projeto
VI3	Experiência em processamento de dados
VI4	Horário de aplicação do experimento
VI5	Ordem de execução do experimento
VI6	Formato do Plano Tabular
VI7	Conhecimento do Visual Studio 2008 DSL Tools
VI8	Similaridade do formato do exemplo da DSL e a descrição do experimento
VI9	Número de dias de aplicação do experimento
VI10	Método utilizado

A fim de fazer um experimento com um ambiente próximo ao que os participantes estão acostumados, foi desenvolvido um Plano Tabular cujo formato é similar ao utilizado pelos participantes profissionalmente (VI6), tanto no aspecto visual como em números de requisitos. Com isso, cada participante criará o produto duas vezes: uma manualmente e a outra utilizando a DSL (VI6). O produto a ser criado terá a mesma complexidade em ambas às aplicações no experimento (VI2), que pode destacar o número de cálculos, tipos de períodos e parâmetros que se pede no Plano Tabular. Com o objetivo de sanar possíveis dúvidas referentes ao manuseio da ferramenta utilizada no experimento, será apresentado ao início do experimento um Plano Tabular com o mesmo aspecto visual (VI8), porém nesse caso, a complexidade será baixa.

A execução do experimento seguiu a mesma ordem para todos os participantes envolvidos, iniciando pela criação com a DSL e em seguida com a criação manualmente (VI5). Como a execução do experimento é iniciada após a entrega do Plano Tabular, a primeira medição (DSL) tende a ser penalizada pelo tempo de entendimento desses requisitos pedidos no documento. Com isso, a escolha de se iniciar pela DSL assumiu-se que o processo automatizado pode ser mais rápido que o manual – mesmo com essa desvantagem. Ou seja, o projeto do experimento favorece o método manual.

Experimentos são feitos quando se deseja manipular o comportamento diretamente, avaliando de forma sistemática, disciplinada e controlada. Em um experimento é fundamental conhecer os fatores que afetam as variáveis. Isto é, os fatores são as variáveis independentes a serem estudadas em um experimento. Para estudá-los, são considerados valores diferentes deles, aplicado o experimento nessas situações (PFLEEFER, 1994).

A variável independente que será utilizada no experimento como fator é a variável (VI10) - Método utilizado. Os valores atribuídos para essa variável são: método DSL e o método manual.

4.3.3. Produtividade a ser medida no experimento

A produtividade de software tem sido muito estudada ao longo dos anos, e segundo Jones (1991) o termo "Produtividade de software" implica em reduzir o tempo necessário para se desenvolver novos sistemas e programas, e com isso, os custos de desenvolvimento tendem a melhorar.

Contudo, o intuito desse experimento é poder medir a produtividade no desenvolvimento de uma parte de um software. Com isso, a produtividade será medida apenas com tempo gasto para se criar um ou mais artefatos definidos no escopo do experimento.

4.3.4. Estratégia de análise dos dados

Uma das formas mais simples de se fazer um experimento é manipular uma variável e medir somente uma saída. Em geral, a manipulação da variável independente envolve ter uma condição experimental (FIELD, 2009).

Field (2009) comenta que o teste *t* foi projetado para analisar diversos cenários e de acordo com o experimento e como a variável independente foi manipulada se tem dois testes *t*:

- *Teste t para amostras independentes*: quando existem duas condições experimentais e diferentes participantes, ou seja, os participantes selecionados de amostras de forma distintas. Por exemplo, um grupo selecionado através de um universo de homens e outros selecionados através de um universo de mulheres.
- *Teste t para amostras dependentes*: quando existem duas condições experimentais e os mesmos participantes tomaram parte em ambas as condições, que é o caso do experimento deste trabalho.

Tanto o Teste *t* independente como o dependente são testes paramétricos baseados na distribuição normal. Segundo Guerra (1994), a distribuição normal ou Gaussiana é a mais importante distribuição considerando a questão prática e teórica. Essa distribuição apresenta-se em forma de sino, uni modal, simétrica em relação a sua média. Portanto é assumido que os dados que serão coletados no experimento serão de população normalmente distribuída.

Neste experimento o Teste t será feito utilizando o fator (VI10) - Método utilizado com valores DSL e manual. A análise será feita com a diferença dos valores desta variável, ou seja, é nessa variável que conterá os valores em tempo obtidos em cada método utilizado.

$$t = \frac{\overline{D} \cdot \sqrt{n}}{S_D}$$

Figura 6: Fórmula do teste *t* para amostras dependentes (BARBETTA, 2006).

Na figura 6 é apresentada a fórmula do teste t para amostras dependentes, onde: *n* é número de pares (Manual, DSL) observados, $\overline{D}$ é a média das diferenças observadas e S_D é desvio padrão das diferenças observadas. Assumindo que $\overline{D}$ tenha distribuição normal, sob H*(nula)*, t tem distribuição "t" com $n - 1$ grau de liberdade. O nível de significância do teste será de 5%. O nível de significância é o limite que se tem como base para afirmar que um determinado desvio é decorrente do acaso ou não. Na prática, considera-se satisfatório o limite de 5%, ou seja, em

uma população qualquer são retiradas 100 amostras e 95 delas poderão ter o seu resultado de teste dentro do mesmo intervalo (FIELD, 2009).

4.3.5. Ameaças à validade

Um dos fatores mais importantes que diferencia um experimento formal de um estudo de caso é o grau de controle que se tem. Quanto mais controle se tem em um experimento e fazendo uso de replicação e randomização a fim de assegurar um teste válido de importância, o controle local reduz significamente o erro experimental (PFLEEGER, 1995).

Durante qualquer experimento formal existem diversas ameaças à validade que precisam ser tratadas e identificadas antes do início do experimento para não invalidar os resultados (KOSAR, et al., 2010). Para restringir o impacto do ambiente na experiência dos resultados, os seguintes problemas foram identificados para o estudo:

- Como o experimento não será executado simultaneamente com todos os participantes, eles podem trocar informações entre si. Para minimizar o impacto, os participantes foram instruídos a não conversarem entre si sobre o experimento;
- O experimento foi realizado em um número reduzido de participantes. Conseguiu-se reunir apenas 4 profissionais, o que pode reduzir a comparação com o mundo real;
- Experimento utilizou um cenário fictício como escopo, o que pode resultar alguma diferença em comparação com cenários reais. Ainda assim foi elaborado um cenário o mais próximo do real visando minimizar o impacto dessa ameaça;
- O exemplo apresentado aos participantes no início do experimento pode ter ajudado os participantes a executar a tarefa mais rapidamente usando a DSL. Isso foi necessário dado o desconhecimento deles da DSL.

4.3.6. Cenário da experimentação

Todos os participantes deverão criar dois artefatos que fazem parte do processo de criação do software de processamento: um script em *transact-sql* contento todas as *procedures* responsáveis pelos cálculos e uma classe em *Visual Basic.Net* responsável por efetuar as chamadas das *procedures*.

Todas as etapas a seguir serão repetidas para cada participante.

- Ao início do experimento serão apresentadas informações essenciais para a execução do experimento, esclarecendo possíveis dúvidas.
- O participante deverá permanecer no local reservado para o experimento, não conversando com ninguém, apenas aguardando as instruções que serão passadas.
- Serão medidos dois tempos de criação de *scripts* em *transact-sql* e *Visual Basic.net:* utilizando a DSL e a de forma manual na criação dos mesmos scripts.
- No caso de dúvidas, apenas aquelas referentes ao uso das ferramentas utilizadas no experimento poderão ser esclarecidas.
- Após a breve apresentação, informar ao participante o local dos arquivos e as pastas a serem utilizadas no experimento. Informar também onde deverá ser criado os arquivos do experimento.
- Iniciar a apresentação da DSL, demonstrando com um exemplo prático as funcionalidades da ferramenta através de um Plano Tabular simplificado (apresentado no apêndice B).
- Em seguida, é entregue o Plano Tabular com as especificações desejadas (apresentado no apêndice A). No processo automatizado (utilizando a DSL), os participantes deverão primeiramente usar a DSL apresentada para criar a solução, seguindo o Plano Tabular. Após a conclusão, o operador do experimento deve anotar o tempo gasto pelo participante e dar uma pausa de dez minutos para o participante. Nesse período, os arquivos gerados pelo participante deverão ser enviados a um repositório a fim de ser analisados após o término do experimento.

- Após a pausa, o ambiente deverá ser reiniciado e todos os arquivos gerados deverão ser excluídos. Com o mesmo Plano Tabular entregue ao início do experimento, o participante deve ser instruído a criar os scripts manualmente de acordo com as especificações. Ao término, anotar o tempo gasto e informar o participante o fim do experimento. Novamente os arquivos gerados deverão ser enviados ao repositório.
- Com a conclusão do experimento, os participantes serão instruídos a fazer uma avaliação geral com ênfase na ferramenta utilizada para criação dos artefatos e no método manual.

4.3.7. Resultados do experimento

Como apresentado anteriormente, o experimento foi aplicado individualmente a um participante por dia e ao término de cada experimento foi anotado o tempo de criação dos artefatos para testar as hipóteses.

Após o término da aplicação do experimento para cada participante foi feita uma avaliação nos artefatos gerados para analisar se o que foi gerado poderia ser utilizado e aceito para o experimento.

No caso da utilização da DSL, todos os participantes que participaram do experimento criaram os artefatos de maneira correta sem muita dificuldade. Porém no caso do método manual, um participante entregou o script em *transact-sql* com alguns erros referentes a padrões de nomes. Esse erro não é muito grave, uma vez que o participante apenas escreveu o nome das chamadas de *procedures* fora do padrão comum – o que dificultaria apenas o processo posterior que não será avaliado neste experimento. Com isso, decidiu-se também considerar os resultados por se tratar apenas de erros de codificação. Na tabela 3 é apresentado o tempo de criação dos artefatos de todos os participantes avaliados no experimento

Tabela 3: Tempo de execução dos métodos por participante.

	Tempo (em minutos)	
Método utilizado	**DSL**	**Manual**
Participante 1	8	20
Participante 2	20	56
Participante 3	12	38
Participante 4	15	45

Após a conclusão do experimento, pediu-se para os participantes fazerem uma avaliação geral do experimento com ênfase na ferramenta utilizada para criação dos artefatos e no método manual. Com isso, foram feitos os seguintes comentários:

- **1º Participante**: o participante destacou como pontos positivos da ferramenta, a qualidade do código gerado (evitando erros de código), a alta produtividade e a familiaridade com ferramentas gráficas, por exemplo, ferramentas de geração de diagramas UML. Porém como ponto negativo, foi comentada a dificuldade na escalabilidade da ferramenta. Conforme o escopo de um projeto se altera, a DSL em questão deve ser construída visando essas mudanças ou possibilitando alterações, ou seja, a DSL deve ser capaz de suprir futuras alterações de escopo. No método manual, foi percebido pelo participante a baixa produtividade e complexidade do código necessário para a criação dos artefatos.
- **2º Participante**: novamente como ponto positivo foi destacado a produtividade obtida no método automatizado e, como a ferramenta apresenta uma interface gráfica (por ser uma DSL gráfica), foi destacada a visão geral que se tem do projeto em questão. No método manual foi comentado que em grandes projetos, que demandam um volume maior de linhas de código, essa abordagem pode induzir ao erro.
- **3º Participante**: o participante comentou sobre o gasto com treinamento para novos desenvolvedores. Como a ferramenta utilizada no experimento abstrai toda a complexidade do domínio, o gasto com treinamento tende a ser reduzido, porém seria necessário um estudo mais aprofundado para confirmar essa afirmação. Novamente foi destacada a alta produtividade na ferramenta utilizada. No método manual foi comentado sobre a necessidade do conhecimento do domínio.

- **4º Participante**: o último participante do experimento comentou a necessidade de uma pessoa responsável por futuras manutenções na DSL, o que não ocorre no método manual devido à necessidade de todos terem que conhecer o domínio. Foi comentado também sobre a alta produtividade na DSL.

De acordo com os valores de tempos anotados e mostrados na Tabela 3, os seguintes cálculos foram realizados:

n: número de pares (Manual, DSL) observados = 4
$\bar{D}$: média das diferenças observadas = 12, 36, 26, 30 = 26
S_D: desvio padrão das diferenças observadas = 10,20
t = 5,701

Analisando esses dados de acordo com a tabela *t-Student* (FIELD, 2009), é possível concluir que foi mais produtivo usar a DSL para a criação de scripts *transact-sql* e classes em *Visual Basic.Net* do que criá-los manualmente. O teste apresentou um resultado t = 5,701, mostrando diferença significativa considerando um nível de significância de 5% e grau de liberdade de 3 (*n-1*, onde *n* é o total de participantes) e o resultado obtido na tabela 4 é igual a 2,353. Com isso podemos rejeitar a hipótese H*(nula)* e aceitar a hipótese H*(alternativa)*.

De uma forma geral, através desse experimento pode-se observar o quanto pode ser útil à utilização de uma DSL para automatização de processos repetitivos, que no caso desse experimento foi à criação dos scripts mencionados anteriormente.

Embora esta experiência mostre a superioridade do uso desta DSL em comparação ao método manual, pode-se observar com base nos comentários coletados dos participantes que existem alguns pontos que devem ser levados em consideração ao se utilizar uma DSL. A utilização da DSL demanda menos tempo no processo de criação dos scripts, porém a manutenção da DSL demanda um profissional com o conhecimento da ferramenta e do domínio em questão para criá-la. A criação de uma DSL e também sua manutenção demanda tempo e

conhecimentos avançados na ferramenta e no domínio, que no caso desse experimento foi utilizado o Visual Studio 2008 DSL Tools. Por se tratar de uma ferramenta relativamente nova o Visual Studio DSL Tools não é muito comum no ambiente de desenvolvimento e deve-se reservar um tempo para treinamento.

Tabela 4: tabela *t-Student* (FIELD,2009)

g.l.	0,200	0,150	0,100	0,050	0,025	0,010	0,005
1	1,376	1,963	3,078	6,314	12,71	31,82	63,66
2	1,061	1,386	1,886	2,920	4,303	6,965	9,925
3	0,978	1,250	1,638	2,353	3,182	4,541	5,841
4	0,941	1,190	1,533	2,132	2,776	3,747	4,604
5	0,920	1,156	1,476	2,015	2,571	3,365	4,032
6	0,906	1,134	1,440	1,943	2,447	3,143	3,707

4.4. Discussão

Após a conclusão do experimento, pôde-se notar que utilizar as fases descritas por Mernik et al. (2005) e estendê-la com uma nova fase de avaliação através de um experimento formal é um forte argumento em se apoiar ao se decidir em criar ou não uma DSL.

Como foi feito um protótipo, a versão final ainda precisará ser criada, mas após a conclusão do experimento, já se tem uma boa impressão de quais vantagens foram detectadas na experimentação e que as vantagens podem ser repassadas para a gerência como forma de obter a autorização para construção da versão final da DSL.

A utilização do experimento formal aponta um resultado consistente para um determinado ambiente. Esse ambiente deverá ser suficientemente parecido com o que a DSL será utilizada, caso contrário os resultados obtidos serão invalidados. Detalhes específicos de um projeto precisam ser considerados no experimento formal, a fim de antecipar possíves problemas no desenvolvimento de uma DSL.

Para coletar as informações de um experimento e posteriormente fazer uso desses dados, todo experimento formal deve ser iniciado e conduzido seguindo todos os requisitos necessários. Contudo, devido a necessidade da preparação que envolve uma experimentação, não são todas as empresas que se dispõe a implementar um processo de avaliação que envolva um experimento formal.

Como a proposta de se criar uma fase de avaliação depende diretamente do uso de um experimento formal, é de extrema importância que esse experimento seja feito seguindo todos os requisitos necessários.

Outro ponto importante, é que a empresa interessada em utilizar essa proposta, precisa dispor de tempo e pessoal capacitado para dar andamento ao processo, o que nem sempre pode ocorrer.

5. CONSIDERAÇÕES FINAIS

Neste trabalho foi proposto uma forma de avaliar uma DSL através da extensão das fases de desenvolvimento descritas por Mernik et al. (2005), fazendo uso de um protótipo e de um experimento formal. Com isso, foi elaborada uma proposta de fazer uma avaliação de uma DSL através de um experimento formal e, com o intuito de exemplificar a avaliação, foi criado especificamente para este trabalho um protótipo de uma DSL como exemplo e aplicado um experimento formal. Pode-se então concluir que o objetivo do trabalho foi alcançado.

Como a DSL é um protótipo, ela tem algumas limitações para o uso real. Mas ainda sim pôde ser utilizada para o experimento formal. Porém como trabalhos futuros, se pretende melhorar o protótipo com intuito de se fazer uso de suas funcionalidades.

5.1. Contribuições do Trabalho

As principais contribuições desse trabalho foram:

- Apresentação de uma extensão das fases de desenvolvimento de uma DSL descritas por Mernik et al. (2005);
- Apresentação de como avaliar uma DSL; e
- Apresentação prática do processo de avaliação através de um experimento formal.

5.2. Trabalhos Futuros

A partir dos resultados deste trabalho é proposto aplicar a abordagem discutida neste trabalho em outros domínios e analisar se a extensão proposta possui os mesmos resultados positivos obtidos.

Outro ponto interessante é desenvolver a versão final da DSL utilizada no experimento, implementando novas funcionalidades e monitorando se os resultados

obtidos no protótipo continuam sendo válidos após o desenvolvimento da versão final.

REFERÊNCIAS

BARBETTA, P. A.. **Estatística Aplicada às Ciências Sociais.** Editora da UFSC, 2006, 315p.

DEURSEN, A.; KLINT, P.; VISSER, J.. **Domain-Specific Languages: An Annotated Bibliography.** ACM SIGPLAN Notices, Junho, 2000, p. 26 - 36.

DOMINGUES, W.B.. **Linguagem Específica de Domínio para Governo Eletrônico em Mídia Cruzada. Dissertação (Mestrado) -** Instituto de Pesquisas e Tecnologias do Estado de São Paulo, Dezembro, 2009, p. 1 – 130.

FIELD, A.. **Descobrindo a Estatística usando o SPSS**. Artmed, 2009, 688p.

FOWLER, M.. **Domain-Specific Languages**. Addison-Wesley, 2010, 640p.

GUERRA, M. J.. **Estatística Indutiva: teoria e aplicações.** Livraria Ciência e Tecnologia Editora, 1982, 311p.

JONES, C.. **Produtividade no desenvolvimento de software.** Makron Books, McGraw-Hill, 1991, 370p.

KOSAR, T.; OLIVEIRA, N.; MERNIK, M.; PEREIRA, M. J. V.; ČREPINŠEK, M.; CRUZ, D.; HENRIQUES, P. R.. **Comparing General-Purpose and Domain-Specific Languages:** An Empirical Study Computer Science and Information Systems, Abril, 2010, p. 247 - 264.

MERNIK, M.; HEERING, J.; SLOANE, A. M.. **When and How to Develop Domain-Specific Languages.** ACM Computing Surveys, Dezembro, 2005, p. 316 - 344.

MICROSOFT. **Visual Studio 2008 SDK 1.1**. Disponível em: http://www.microsoft.com/en-us/download/details.aspx?id=21827. Acessado em 07/05/2013.

PFLEEGER, S. L.. **Design and Analysis in Software Engineering Part 1:** The Language of Case Studies and Formal Experiments. ACM SIGSOFT Software Engineering Notes, Outubro, 1994, p. 16 - 20.

PFLEEGER, S. L.. **Experimental Design and Analysis in Software Engineering Part 2:** How to Set Up and Experiment. ACM SIGSOFT Software Engineering Notes, Janeiro, 1995, p. 22 - 26.

SERRA, I.J. C.. **Uma Técnica para o Desenvolvimento de Linguagens Específicas de Domínio. Dissertação (Mestrado) -** Universidade Federal do Maranhão, 2004, p. 1 – 131.

THIBAULT, S.; MARLET, R.; CONSEL, C.. **Domain-Specific Languages:** from Design to Implementation Application to Video Device Drivers Generation. IEEE Transactions on Software Engineering, Maio 1999, p.363-377.

APÊNDICE A – Plano tabular proposto para o experimento

A pesquisa em questão tem como objetivo avaliar o nível de satisfação dos clientes atendidos na rede concessionárias da Empresa X, tanto com o atendimento em vendas como os serviços efetuados em garantia.

Os resultados apresentados ao cliente deverão ser representados por diversos agrupamentos, diferentes períodos e cálculos específicos. Todas as informações necessárias serão descritas logo abaixo:

Pesquisa:

Nome: Satisfação

Banco de dados: Experimento

Parâmetro: CodPeriodo

Níveis por Cálculos	Quantidade	Percentual	Amostra	Posição Quartil	Faixa Quartil	Correlação	Ranking
Dealer	x	x	x	x		x	x
Setor	x	x	x				
Grupo	x	x	x				
Região	x	x	x			x	
Nacional	x	x	x		x	x	

Níveis por Períodos	Mensal	Trimestral	Anual
Dealer	x	x	
Setor	x	x	
Grupo	x	x	
Região	x	x	
Nacional	x	x	x

Níveis por Parâmetros	1.	2.
Dealer	CodDealer	---
Setor	CodSetor	CodTipoQuest
Grupo	CodGrupo	---
Região	CodRegiao	---
Nacional	---	---

Arquivos gerados	Execução
clsTabulacao.vb	---
Procedures.sql	Executar diretamente no Banco de dados

APÊNDICE B – Plano tabular de exemplo

Cliente: X.

A pesquisa em questão tem como objetivo avaliar o nível de satisfação dos clientes atendidos na rede concessionárias da Empresa X, tanto com o atendimento em vendas como os serviços efetuados em garantia.

Os resultados apresentados ao cliente deverão ser representados por diversos agrupamentos, diferentes períodos e cálculos específicos. Todas as informações necessárias serão descritas logo abaixo:

Pesquisa:

Nome: Satisfação

Banco de dados: Experimento

Parâmetro: CodOnda

Níveis por Cálculos	Quantidade	Percentual	Amostra
Loja	x	x	x
Região	x	x	x
Nacional	x	x	x

Níveis por Períodos	Mensal	Trimestral
Loja	x	x
Região	x	x
Nacional	x	x

Níveis por Parâmetros	1.	2.
Loja	CodLoja	---
Região	CodRegiao	---
Nacional	---	---

Arquivos gerados	Execução
clsTabulacao.vb	---
Procedures.sql	Executar diretamente no Banco de dados